# Chapter 7

# A case study on decompounding in Indian language IR

## 7.1 Introduction

Decompounding (as introduced in Sec. 1.3.2) is an essential pre-processing step in text-processing tasks such as machine translation, speech recognition and IR. Here, the IR issues are explored from five viewpoints. A) Does word decompounding impact the Indian language IR? If yes, to what extent? B) Can corpus-based decompounding models be used in the Indian language IR? If yes, how? C) Can machine learning and deep learning-based decompounding models be applied in the Indian language IR? If yes, how? D) Among the different decompounding models (corpus-based, hybrid machine learning-based, and deep learning-based), which provides the best effectiveness in the IR domain? E) Among the different IR models, which provides the best effectiveness from the IR perspective? In this study, we propose different corpus-based, hybrid machine learning-based and deep learning-based decompounding models in Indian languages (Marathi, Hindi and Sanskrit). Moreover, we evaluate the effectiveness of each activity from an IR perspective only. It is observed that the different decompounding models improve IR effectiveness. The deep learning-based decompounding models outperform the corpus-based and hybrid machine learning-based models in Indian language IR. Among the different deep learning-based models, the Bi-LSTM-A model performs the best and improves MAP by 28.02% in Marathi. Similarly, the Bi-RNN-A model improves MAP by 18.18% and 6.1% in Hindi and Sanskrit, respectively. Among the retrieval models, the In_expC2 model outperforms others in Marathi and Hindi, and the BB2 model outperforms others in Sanskrit.

## 7.2 Problem Formulation

This section introduces the features of the Indian language compound words. A compound word is formed by combining two or more words that create a new word with a specific meaning. The combination generally occurs in two different ways. First, simple concatenation of two or more words following *Sandhi*[1] rules. In the case of simple concatenation, two or more words are combined to generate a new word. For example, in English, 'note' and 'book' are combined to form 'notebook'. In *sandhi*-ed compound word, the last character of the first word and the first character of the second-word change at the merging location. For example, '*vidya*' (meaning education) and '*alaya*' (meaning house or place) are combined to form '*vidyalaya*' (meaning school). An Indian language compound word derives the *sandhi* rules from the Sanskrit language. There are three types of *Sandhi*s as defined in azwADyAyI[2] as given below.

1. *Swara Sandhi*: Here, the constituents of the compound word are combined at the merging locations of vowels. i.e., vowels are the last character of the first word as well as the first character of the second word. For example, जन (jana) [= जन् + अ (jan + a)] and आदेश (adesha) are combined to form जनादेश (janadesha).

2. *Vyanjana Sandhi*: Here, constituents are combined at the merging location, which is a consonant. I.e., either the last character of the first word or the first character of the second word is a consonant, or both are consonants. For example, सत् (sat) and जन (jana) are combined to form सज्जन (sajana).

3. *Visarga Sandhi*: In this rule, the last character of the first constituent word is a *visarga*(':'). It means that a Devanagari script character placed at the end of a word generates an additional H sound. For example, अंतः (antaH) and गत (gata) are combined to form अंतर्गत (antargata).

In this study, we consider only the closed compound words, which means that two constituent words do not have a word boundary between them. Moreover, we do not consider the hyphens as word delimiters and do not split the hyphenated words. The word decompounding method requires correct syntactic splitting to obtain semantically meaningful constituent words. In word decompounding, there are two important tasks: 1. detecting the location of the split point and 2. splitting the compound word into two different constituents. The tasks get more complicated because of the different morphological features of Indian languages. An Indian

[1]The word Sandhi means placing together

[2]https://www.britannica.com/topic/Ashtadhyayi

language compound word can be split into multiple ways. Hence, detecting the splitting point location is a difficult task. A compound word '$w$' may be formed by concatenating two or more words, i.e., $w_1, w_2, \ldots, w_n$. However, in this work, we split the compound word into two parts only.

## 7.3 Contribution

Decompounding is used as an important pre-processing task in the morphologically rich European and Asian languages. However, the impact of decompounding has been less explored in the Indian languages. This study explores the impact of decompounding in low-resource Indian languages from an IR perspective. The contributions made in this chapter are centred around the following research questions and finding answers to them.

*[RQ 5.1] Can corpus-based decompounding models be used in the Indian language IR? If yes, how?* We discuss some existing corpus-based decompounding models in Section 7.3.1 and evaluate their effectiveness in the following sections.

*[RQ 5.2] Can machine learning and deep learning-based decompounding models be used in the Indian language IR? If yes, how?* We propose different hybrid machine learning-based and deep learning-based decompounding models and evaluate their effectiveness.

*[RQ 5.3] Among the different decompounding models (corpus-based, hybrid machine learning-based, and deep learning-based), which provides the best effectiveness in the IR domain?* Based on our series of experiments, we observe that the deep learning-based decompounding models perform better than their counterparts in Indian languages.

*[RQ 5.4] Among different IR models, which provides the best effectiveness from an IR perspective?* In our set of experiments, DFR-based retrieval models outperform other models.

### 7.3.1 Corpus-based decompounding approaches

Here we evaluate some of the existing decompounding techniques that have provenly yielded high effectiveness in different morphologically rich European and Asian languages [65], [32], [43]. The impact of decompounding has been investigated from an NLP perspective. However, we study it from the IR viewpoint. In the corpus-based decompounding approach, the dictionary comprises the simple words of the document collection.

### Frequency-based approach

In this approach, we split the compound words based on the frequency of probable constituents in the collection. The more frequently a probable constituent independently occurs in the collection, the more likely it is to be a part of a given compound word. This insight defines that a splitting point of a word depends upon the frequency of the constituent words in the collection. The splitting point of a compound $S$ is detected by the highest geometric mean obtained among considering word frequencies of all possible splits ($e_i$) given by equation 7.1 ($n$ being the number of splits, we consider here $n = 2$ only). This approach is in line with the earlier work of the compound splitting method applied in the German-English machine translation [65].

$$\text{argmax}_S(\prod_{e_i \in S} \text{frequency}(e_i))^{\frac{1}{n}} \tag{7.1}$$

### Maximum-branching factor approach

Here, we explore a language-independent decompounding technique that can be applied to any compounding language. The decompounding is implemented in two steps. The first step detects the split point location, and the second step validates the morpheme so identified. In the first step, we create a lexical character-based tree, where each node represents a character, and a word can be found by traversing the path of the tree to a leaf node. From the tree, a split point location is identified based on the frequency of the constituent terms (the prefix and the suffix part of a complete path). In the second step, we validate the morpheme boundary using a dictionary. This approach is in line with the earlier work of the compound splitting method applied in German LVCSR [3].

### Trie-based approach

Here, we use a trie-based dictionary to split the compound words. A potential compound word is detected if the currently processed word is part of a word already present in the trie or a word in the trie is a substring of the current word. At first, we store the prefix in a trie-based dictionary and look for potential candidate words to form a compound word. For example, in the compound word 'underworld', the algorithm traverses the trie-based dictionary, stores the prefix 'under', and looks for the candidate word 'world' by traversing the dictionary and then decompound the word 'underworld' into 'under' and 'world'. We also use the prefix list (Table 3) and apply sandhi[3] rules to deal with the morphological features of Indian languages. However,

[3]as described in *Ashtadhyayi* (azwADyAyI)

the previously evaluated decompounding in Hindi speech synthesis [32] does not consider the morphological characteristics of Indian languages.

### Co-occurrence based decompounding approach

In this approach, we first split the compound words using the frequency-based approach described above. We also apply relaxation and co-occurrence-based techniques to deal with the morphological features of Indian language compound words. The relaxed decompounding technique does not split the prefix-based compound words such as 'upanagar' into 'upa' and 'nagar'; the co-occurrence-based decompounding technique tries to avoid some of the wrong splitting of the frequency-based approach. For example, in the frequency-based approach, the word 'Loksabha' is split into 'lok' and 'sabha' due to the high frequency of compound constituents in the collection. However, in the co-occurrence-based technique, the compound word 'Loksabha' is not split into 'Lok' and 'sabha' because the terms 'Lok' and 'sabha' will not frequently co-occur in the document collection. The co-occurrence-based decompounding technique splits the compound word into constituents only if the compound constituents co-occur with the compound word higher than a particular threshold. Here, we consider an optimal threshold value $\tau = 0.2$ for different Indian languages as we find $\tau = 0.2$ provides the best retrieval effectiveness in Bengali IR [43]. The co-occurrence measure is the overlap coefficient between the set of documents $D(c)$ containing the constituent term $c$ and $D(w)$ containing the compound word. We measure the overlap coefficient by equation 7.2. This approach aligns with the earlier work of the compound splitting method evaluated in Bengali IR [43].

$$Overlap(w, c) = \frac{|D(w) \bigcap D(c)|}{min\{|D(w)|, |D(c)|\}} \tag{7.2}$$

## 7.3.2 Machine learning-based decompounding approaches

Basic machine learning models such as SVM, decision trees, CRF, and random forests are widely used in different computational areas like text classification, POS tagging, and other NLP tasks. However, these models are yet to be explored in the word decompounding task for Indian languages. Here, we evaluate the impact of different hybrid (dictionary and machine-learning-based) decompounding models in different Indian languages. We used machine learning models such as decision tree, random forest, $k$-nearest neighbours, SVM and CRF. At first, we split the simple concatenation of compound words using a dictionary in each language. To split *sandhi*-ed compound words, we use different machine learning models. We train the machine learning models using a set of training examples. The training examples comprise three types of *sandhi*

as described in Section 7.2. Post-training, the models are checked using compound words from the test data set and evaluation is done by looking at the change in the retrieval effectiveness of different IR techniques. Here, we first briefly describe different ML techniques used in this context, followed by the general steps adopted for these machine learning-based decompounding models.

### Decision tree approach

The decision tree [102] follows a tree-like structure, where each internal node represents the test, the branch represents the outcome of the test, and the leaf node represents the decision after computing all the attributes. In the decompounding method, the root node represents the compound word, and the children nodes represent the individual constituent words. This process is repeated recursively. The recursion is completed when each leaf node contains the constituent word. The decision tree construction does not require domain knowledge or parameter settings; hence, it is appropriate for exploratory knowledge discovery.

### Random forest approach

Random forest [52] takes a variety of decision trees and averages the effectiveness of all trees to improve the splitting accuracy of the system. Instead of just focusing on one decision tree, the random forest model uses multiple decision trees, and prediction is based on the majority votes of prediction of different trees. This approach is based on ensemble learning, combining multiple trees to improve the system's effectiveness. More trees in the forest lead to higher accuracy and prevent the over-fitting problem.

### $k$-nearest neighbors approach

The k-nearest neighbors algorithm [24] assumes that similar objects are close to each other. In this algorithm, the nearer neighbours contribute more to the average than the distant ones. The algorithm depends upon the distance function. If the features belong to different classes, normalizing the training data improves the system's efficiency. The neighbours are treated as the training set for the algorithm so that the algorithm does not require any explicit training data. The training example comprises a multidimensional feature vector with each class labelled.

### Conditional random field approach

Machine learning models are broadly categorised into two types: the generative models and the discriminative models. A conditional random field [74] is a discriminative model that predicts

the current task based on the contextual information or state of the neighbours. The discriminative model encodes many features from the input data and enhances the effectiveness of the prediction. CRF is mainly designed to label and segment sequence data. In recent years, CRF and SVM have been widely used in different NLP applications like POS tagging and named entity recognition, providing the best results in various computational tasks. So, we evaluate the model's effectiveness in decompounding tasks in Indian languages.

### Support vector machine approach

SVM [22] is a supervised learning algorithm primarily used for classification tasks. Suppose we have a set of training samples $D = (x_1, y_1)$, ..., $(x_m, y_m)$ where $x_i$ is the feature vector of $i$-th training sample, $y_i$ is the output class of $i$-th training sample, '$m$' being the number of training samples. The main idea behind SVM is to classify the samples into two or more classes by a hyper-plane. SVM finds a hyperplane which maximizes its margin. SVM chooses the extreme points near the boundaries that help create a hyper-plane. The extreme points are called support vectors; hence, the algorithm is called a Support Vector Machine.

### Algorithm for machine learning-based decompounding approaches

The basic machine learning-based decompounding model is implemented in the following steps. We first split the simple concatenation of compound words followed by the *sandhi*-ed compound word.

1. Read a set of input tokens from the user.
2. Remove the stopwords from the input tokens.
3. For each prefix in the *prefix list* (Table 3) ($L_{pre}$), check whether the prefix is there in the given token or not. If yes, consider the entire input word as a compound word only.
4. Check the length of the word in each token. If the token length is less than seven 7, then EXIT.
5. If the word length exceeds six 6, then traverse the dictionary and look for any compound constituent present; if yes, break that compound word into its constituent morphemes.
6. If any of the above conditions are not satisfied, check for the *sandhi*-ed compound word splitting.

**Sandhied compound word splitting**

7. Assign a numeric number (Unicode) to each letter in the Devanagari script.

8. Train different machine learning models by training data sets. The training data set comprises the compound words and their constituent morphemes. Store the number corresponding to the first morpheme's last and second morpheme's first character in the training data set.

9. During training, the machine learning model learns from different *sandhi* examples.

10. During testing, the machine learning model classifies the compound words into one of the trained sandhi examples and generates constituent morphemes.

11. After decompounding, different retrieval models are applied to evaluate the change in retrieval effectiveness using the MAP scores.

The primary difficulty in developing a machine learning-based decompounding model is that it requires enormous training data. We also observe that the machine-learning-based model requires a complex feature representation. Hence, we also evaluate the deep learning-based decompounding models.

### 7.3.3 Deep learning-based decompounding approaches

In recent years, neural network models such as RNN, LSTM, and GRU have shown to provide good effectiveness in the text segmentation task of Chinese [20], Japanese [64] and Sanskrit [49], [50], [108]. However, these neural network models have not yet been evaluated in the word decompounding task of Indian languages. This study explores different word decompounding models based on deep learning, such as RNN, LSTM, and GRU, its bidirectional versions such as Bi-RNN, Bi-LSTM, Bi-GRU, and attention versions such as Bi-RNN with attention (referred to as Bi-RNN-A), Bi-LSTM with attention (referred to as Bi-LSTM-A), and Bi-GRU with attention (referred to as Bi-GRU-A) model and evaluates their effectiveness in the Indian language IR. The decompounding task is seen here, in a sense, similar to the language translation problem - as the sequence of characters or compound words is taken as input and produces another sequence of characters or decompounded words as output. The sequence-to-sequence model (Sutskever et al. [137]) is widely used in the NLP domain. Here, we use the same model for word decompounding in Indian languages. The proposed model uses an RNN-based two-stage deep neural network for compound word splitting. The step for the decompounding algorithm

is given below. The code is implemented in Python 3.5 with Keras API running on the Tensor Flow back-end. The code for different decompounding models is publicly available on GitHub [4].

### Encoder-Decoder Model

The proposed encoder-decoder model is inspired by the character level sequence to sequence model [137]. The encoder comprises one recurrent layer, i.e., RNN, LSTM or GRU, and converts the input character sequence into a one-hot encoding vector and processes it to the subsequent layer. Similarly, the decoder comprises one recurrent layer (RNN, LSTM or GRU) followed by a dense soft-max layer. For training, we use the Marathi dataset developed by Singh et al. [129] and Sanskrit [5] dataset developed at the University of Hyderabad. We train the encoder-decoder model by providing a compound word at a time and corresponding decompounded words concatenated with a '+' in between. The input and output character sequences are replaced by their one-hot encoded vector sequences. Characters '&' and '$' are used as start and end markers in the output sequence. The output of the encoder-decoder model is the same one-hot encoded vector of decompounded words but with a left shift of characters by length one, i.e., offset by a one-time step in the future. We use a two-stage deep neural network for compound splitting. The reason behind using this two-stage deep neural network is that it detects the splitting point location efficiently [31], [11]. In the first stage, the model predicts a *sandhi*-window (a maximum of 5-letter window where at least two letters come from the first constituent word and at least another two letters from the second constituent word). In the second stage, the most probable splits that are likely to constitute the compound word are identified. Both the stages use the seq2seq model.

In the first stage, we provide a compound word as input to the sequential model that the model converts into a one-hot encoding vector and stores it in an array. All the array elements are assigned value zero except the *sandhi* window element, which is assigned value one, as shown in Figure 7.1 (the highlighted letters constitute *sandhi*-window). In the second stage, using the *sandhi* window, we split the compound word into two words. The input sequence is a *sandhi* window, and the output sequence is two words with a space delimiter between them. The model architecture of *sandhi* splitting is shown in Figure 7.2. The model is trained by the Adam optimizer [63] and the categorical cross-entropy loss function. The training vectors are divided into a batch size of 64. In the encoder-decoder model, we experiment with a different version of sequential models and parameters and the best MAP score achieved at a given parameter

[4]https://github.com/cse-iitbhu/Decompounding-code-and-Training-dataset

[5]http//sanskrit.uohyd.ac.in/Corpus/

setting is shown in Table 7.1. At first, we trained the basic RNN model, but the model was not effective in different Indian languages. Hence, we experiment with other sequential models such as Bi-RNN, Bi-RNN-A, LSTM, Bi-LSTM, Bi-LSTM-A, GRU, Bi-GRU, and Bi-GRU-A. A detailed explanation of different sequential models is given below.

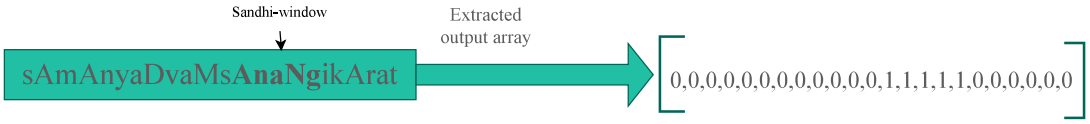


Figure 7.1: Sandhi-window (AnaNg) as the prediction target in a compound word

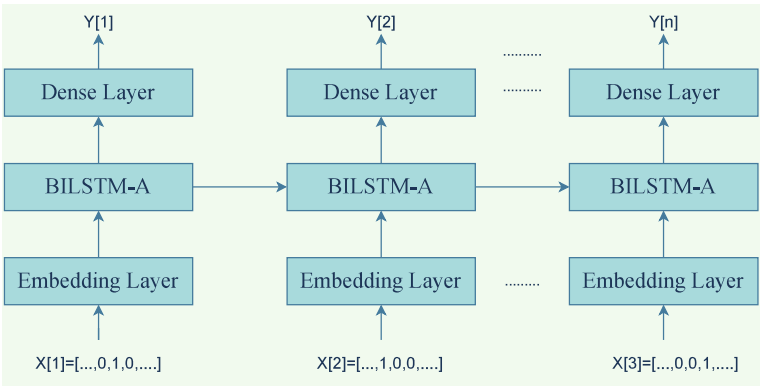


Figure 7.2: Model Architecture for Sandhi Split - Stage 1

### Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) [86] is a class of artificial neural networks which deals with the variable length sequential data $X = (x_1, x_2, \ldots, x_t, \ldots)$. RNN uses memory at each internal node to process variable-length sequential data. In an RNN, the output state $h(t)$ depends on the previous state information $h(t-1)$ and present input $x_t$. At any time-step $t$, the value of the current state is updated by the equation 7.3. A basic RNN architecture is shown in Figure 7.3.

$$h(t) = f(h(t-1), x_t) \tag{7.3}$$

Where $f$ represents the non-linear activation function. In a simple RNN, we use the soft-max activation function. We train the RNN by gradient-descent methods and back-propagation. In each training iteration, the gradient is evaluated and updated with the current weight. During the training of sequential data, the gradient will not be updated, which will prevent changes in the RNN model's weight. Hence, the model does not learn from the previous inputs and

causes the vanishing gradient problem. We use other sequential models like LSTM and GRU to overcome the vanishing gradient problem.

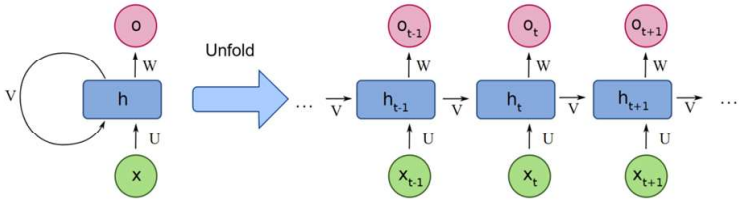


Figure 7.3: A basic RNN Architecture

### Long Short-Term Memory Network (LSTM)

The traditional RNN suffers from the problem of vanishing gradient. Hence, the RNN could not handle the long-term sequential data. To overcome the vanishing gradient problem, Hochreiter and Schmidhube [53] proposed an LSTM network. The LSTM network can resolve the issue by maintaining different memory blocks and gates. The memory block comprises memory cells with self-connections, and different gates are used to regulate the flow of information. Primarily, the LSTM network comprises four types of gates, i.e., input gate, output gate, forget gate and memory-cell activation gate, as described below.

1. **Input gate:** The input gate ($i_t$) determines which input should be kept at the cell state and which input is transmitted to the next long-term state, i.e. $c_t$. The gate determines that some part of the information is transmitted and reflected in long-term memory.

2. **Forget gate:** The forget gate ($f_t$) determines which part of the long-term data should be kept and passed to the next state ($c_t$) and which part of the long-term data is to be thrown away from the cell state. The decision is made by the sigmoid layer called the 'forget gate layer'.

3. **Memory-cell activation gate:** The memory-cell activation gate ($c_t$) updates the information in the memory state cell by taking the information from the input gate and forget gate.

4. **Output gate:** The output gate ($o_t$) determines which part of the long-term state should appear in the output network.

Mathematical expressions of different gates are as follows.

$$i_t = \sigma(W_{xi} \cdot x_t + W_{hi} \cdot h_{t-1} + W_{ci} \cdot c_{t-1} + b_i) \tag{7.4}$$

$$f_t = \sigma(W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1} + W_{cf} \cdot c_{t-1} + b_f) \tag{7.5}$$

$$o_t = \sigma(W_{xo} \cdot x_t + W_{ho} \cdot h_{t-1} + W_{co} \cdot c_{t-1} + b_o) \tag{7.6}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tanh(W_{xc} \cdot x_t + W_{hc} \cdot h_{t-1} + b_c) \tag{7.7}$$

There is a hidden state represented as follows.

$$h_t = o_t \cdot \tanh(c_t) \tag{7.8}$$

$W_{xi}, W_{xf}, W_{xo}, W_{xc}$ are the weight matrices of different layers connected to the input vector $X_t$.$W_{hi}, W_{hf}, W_{ho}, W_{hc}$ are the weight matrices connected to the state $h_{t-1}$ and $b_i, b_f, b_o, b_c$ are the biases at different layers. A single-cell LSTM architecture is shown in Figure 7.4.

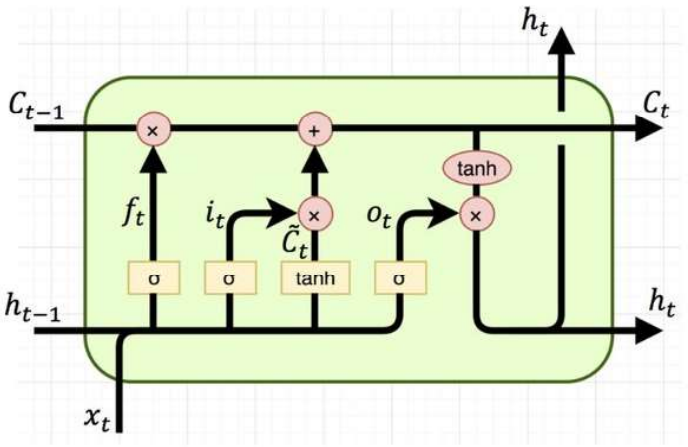


Figure 7.4: A single cell LSTM Architecture

### Gated Recurrent Unit (GRU)

A Gated recurrent unit is a variant of recurrent neural networks like RNN and LSTM introduced by Cho et al. [21]. GRU also solves the problem of vanishing gradient. GRU is like an LSTM network but comprises fewer parameters and gates.

1. **Update gate:** The update gate ($z_t$) determines the amount of information transferred from one hidden state to the next hidden state. If the output gate is close to 1, then all current hidden state information is transferred to the next hidden state.

2. **Reset gate:** The reset gate ($r_t$) determines the significance of previously hidden state information in updating current hidden state information. If the output of the reset gate is close to 0, the hidden state is forced to ignore the previously hidden state information and reset with the current input only. It signifies that the hidden state can drop any information that is found to be irrelevant, allowing a more compact representation.

3. **Hidden state:** The hidden state ($h_t$) is updated using the current input and the information obtained from the previous hidden state ($h_{t-1}$).

Mathematical expressions of different gates are as follows.

$$z_t = \sigma(W_z \cdot x_t + U_z \cdot h_{t-1} + b_z) \tag{7.9}$$

$$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1} + b_r) \tag{7.10}$$

$$h_t = (1 - z_t) \cdot \tanh(r_t \cdot U \cdot h_{t-1} + W \cdot x_t) + z_t \cdot h_{t-1} \tag{7.11}$$

$W_z, W_r$ and $W$ are the weight matrices of different layers connected to the input vector $X_t$, while $U_r$, $U_z$ and $U$ for the $U$ for $h_{t-1}$. $b_z$ and $b_r$ are the biases at different layers. A single-cell GRU architecture is shown in Figure 7.5.

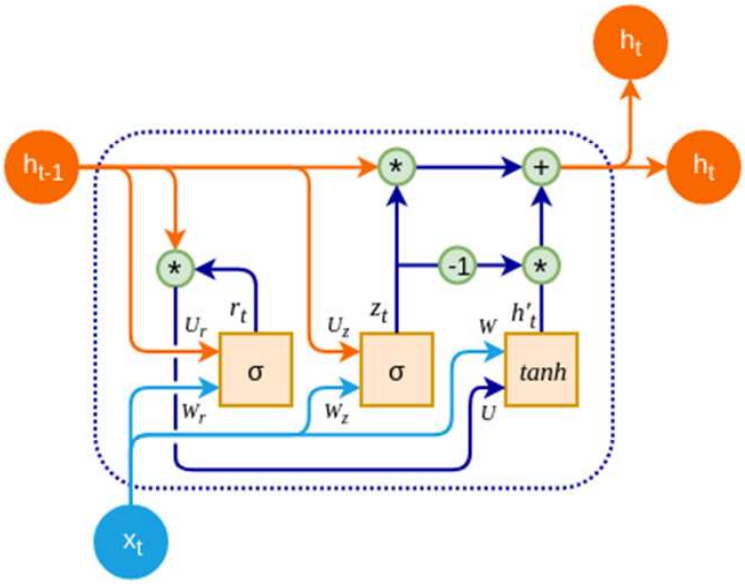


Figure 7.5: A single cell GRU Architecture

Table 7.1: Show the Parameter of different Encoder-Decoder models

| ↓ Models \| Parameter → | Latent Dimension | Learning Rate | Epochs |
|---|---|---|---|
| **RNN** | 256 | 0.0005 | 90 |
| **LSTM** | 256 | 0.001 | 40 |
| **GRU** | 256 | 0.001 | 40 |
| **Bi-RNN** | 256 | 0.0005 | 90 |
| **Bi-LSTM** | 256 | 0.001 | 40 |
| **Bi-GRU** | 256 | 0.001 | 40 |
| **Bi-RNN-A** | 256 | 0.0005 | 90 |
| **Bi-LSTM-A** | 512 | 0.001 | 40 |
| **Bi-GRU-A** | 512 | 0.001 | 40 |

## 7.4 Experimental Setup

Deep learning-based decompounding models for different Indian languages are evaluated in the following steps. The proposed decompounding models can split the simple concatenation and *sandhi*-ed compound words.

1. Neural network models are implemented using Keras library[6].

2. During compound splitting, the prefix-based compound word (Table 3) ($L_{pre}$) with word length less than seven and larger length compound words (length greater than 20) are ignored. The larger length of compound words reduces the effectiveness of decompounding models; hence, we ignore those compound words.

3. Different neural network models such as (RNN, LSTM, or GRU) are trained on both the simple concatenation and *sandhi*-ed compound words.

4. The neural network models are tested on FIRE[7] text collection for different Indian languages.

5. In the document collection, each compound word is replaced with its split words. The stopwords are removed from the document collection, and the effectiveness of decompounding models is evaluated in the IR settings.

[6]https://keras.io/

[7]http://fire.irsi.res.in

6. The MAP scores of the baseline approach (unmodified document collection) are compared with the split version of the compound word in the collection (modified document collection).

## 7.5 Evaluation

To address the research questions (RQs) described in the introduction section, we detail the results and our observations below.

### 7.5.1 Effect of corpus-based decompounding on retrieval

In the first set of experiments, we evaluate the impact of different corpus-based decompounding models in Indian language IR. The effect of different corpus-based decompounding models for Marathi, Hindi and Sanskrit languages is shown in Table 7.2, 7.3 and 7.4 respectively. In these tables, the best MAP scores among the retrieval models are shown in boldface. It is observed that the different corpus-based decompounding models improve retrieval effectiveness in different Indian languages IR. However, the effect of decompounding models varies from one language to another. The frequency-based decompounding model improves a MAP score of 12.94% in Marathi, 4.56% in Hindi and 0.37% in Sanskrit (In_expC2 model). Similarly, the maximum branching factor, tri-based, and co-occurrence-based decompounding models improve the MAP scores by 13.5%, 13.87%, and 14.98% in Marathi, 4.34%, 4.22%, and 4.86% in Hindi and 2.8%, 4.75% and 4.58% in Sanskrit languages respectively (In_expC2 model). Other models also show improvements in general, although at different levels, including some drops. These observations are in line with earlier findings as different corpus-based decompounding models provide comparable effectiveness in the European languages IR [89].

Table 7.2: MAP scores of different corpus-based decompounding evaluation in Marathi (39 T queries)

| | ↓ Parameter \| R.M. → | BM25 | TF-IDF | In_expC2 | BB2 | InL2 | Hiem_LM |
|---|---|---|---|---|---|---|---|
| Base line | MAP | 0.2833 | **0.284** | 0.2155 | 0.2533 | 0.2504 | 0.2692 |
| | Rel.Ret. | 458 | 458 | 458 | 462 | 458 | 462 |
| Frequency-based | MAP | 0.2801 | **0.2812** | 0.2434 | 0.2568 | 0.2494 | 0.2541 |
| | Rel.Ret. | 468 | 469 | 462 | 464 | 463 | 463 |
| Maximum branching factor | MAP | 0.2806 | **0.2822** | 0.2446 | 0.2551 | 0.2516 | 0.2543 |
| | Rel.Ret. | 468 | 470 | 462 | 463 | 462 | 463 |
| Trie-based | MAP | 0.281 | **0.282** | 0.2454 | 0.2545 | 0.2594 | 0.2549 |
| | Rel.Ret. | 469 | 470 | 462 | 463 | 463 | 463 |
| Co-occurrence based | MAP | 0.2866 | **0.2878** | 0.2478 | 0.2576 | 0.2605 | 0.2618 |
| | Rel.Ret. | 471 | 471 | 464 | 465 | 464 | 464 |

Table 7.3: MAP scores of different corpus-based decompounding evaluation in the Hindi (50 T queries)

| | ↓ Parameter \| R.M. → | BM25 | TF-IDF | In_expC2 | BB2 | InL2 | Hiem_LM |
|---|---|---|---|---|---|---|---|
| Base line | MAP | 0.4429 | **0.4439** | 0.3266 | 0.3606 | 0.3797 | 0.3951 |
| | Rel.Ret. | 1682 | 1679 | 1640 | 1657 | 1652 | 1668 |
| Frequency-based | MAP | 0.44 | **0.4426** | 0.3415 | 0.3742 | 0.3738 | 0.4064 |
| | Rel.Ret. | 1672 | 1671 | 1650 | 1663 | 1656 | 1660 |
| Maximum branching factor | MAP | 0.4445 | **0.4458** | 0.3408 | 0.3734 | 0.3725 | 0.4048 |
| | Rel.Ret. | 1672 | 1671 | 1649 | 1661 | 1660 | 1660 |
| Trie-based | MAP | 0.4451 | **0.4456** | 0.3404 | 0.3725 | 0.3801 | 0.4095 |
| | Rel.Ret. | 1671 | 1670 | 1649 | 1660 | 1662 | 1661 |
| Co-occurrence based | MAP | 0.4464 | **0.4472** | 0.3425 | 0.3755 | 0.3815 | 0.4105 |
| | Rel.Ret. | 1672 | 1672 | 1650 | 1662 | 1662 | 1662 |

Table 7.4: MAP scores of different corpus-based decompounding evaluation in the Sanskrit (50 T queries)

| | ↓ Parameter \| R.M. → | BM25 | TF-IDF | In_expC2 | BB2 | InL2 | Hiem_LM |
|---|---|---|---|---|---|---|---|
| Base line | MAP | 0.4208 | 0.4218 | 0.3995 | 0.4079 | 0.4025 | **0.4339** |
| | Rel.Ret. | 389 | 386 | 385 | 388 | 387 | 389 |
| Frequency-based | MAP | 0.4231 | 0.422 | 0.401 | 0.4057 | 0.4095 | **0.4164** |
| | Rel.Ret. | 389 | 387 | 386 | 387 | 388 | 386 |
| Maximum branching factor | MAP | 0.4272 | 0.4276 | 0.4108 | 0.4186 | 0.4139 | **0.4295** |
| | Rel.Ret. | 389 | 387 | 386 | 389 | 389 | 388 |
| Trie-based | MAP | 0.4282 | 0.43 | 0.4185 | 0.421 | 0.4147 | **0.4331** |
| | Rel.Ret. | 390 | 388 | 388 | 390 | 389 | 389 |
| Co-occurrence based | MAP | 0.4286 | 0.4294 | 0.4178 | 0.4218 | 0.4135 | **0.4341** |
| | Rel.Ret. | 390 | 388 | 387 | 391 | 389 | 389 |

### 7.5.2 Effect of machine learning-based decompounding on retrieval

In the second set of experiments, we propose hybrid machine learning-based decompounding models followed by their evaluation. The decompounding models comprise the dictionary and machine learning-based approach to split the compound words. Impact of different hybrid machine learning-based decompounding models on Marathi, Hindi and Sanskrit languages is shown in Table 7.5, 7.6 and 7.7 respectively. It is observed that the different decompounding models enhance the effectiveness of document retrieval. On closer observation, we find that the effect of decompounding models varies from one language to another. The SVM model improves MAP score by 22.08% in Marathi, 10.59% in Hindi and 2.95% in Sanskrit languages. Similarly, the CRF, KNN, Decision Tree and Random Forest-based models improve MAP scores by 22.27%, 15.73%, 16.84%, and 16.7% in Marathi, 10.83%, 16.41%, 16.65% and 17.14% in

Hindi and 2.82%, 4.75%, 4.58% and 4.43% in Sanskrit languages respectively (all in In_expC2 model). In other models, the scores also mostly improve with a few exceptions. Different hybrid machine learning-based models outperform the pure corpus-based models in Indian and European languages IR [89], [43]. The fundamental reason is that the machine learning-based decompounding models can efficiently split the simple concatenation and *sandhi*-ied compound words.

Table 7.5: MAP scores of different hybrid machine learning-based decompounding evaluation in the Marathi (39 T queries)

| | ↓ Parameter \| R.M. → | BM25 | TF-IDF | In_expC2 | BB2 | InL2 | Hiem_LM |
|---|---|---|---|---|---|---|---|
| Base line | MAP | 0.2833 | **0.284** | 0.2155 | 0.2533 | 0.2504 | 0.2692 |
| | Rel.Ret. | 458 | 458 | 458 | 462 | 458 | 462 |
| SVM | MAP | 0.2868 | **0.2873** | 0.2631 | 0.271 | 0.2681 | 0.2656 |
| | Rel.Ret. | 472 | 473 | 476 | 480 | 473 | 469 |
| CRF | MAP | 0.2864 | **0.2869** | 0.2635 | 0.2688 | 0.2685 | 0.2671 |
| | Rel.Ret. | 471 | 472 | 475 | 480 | 472 | 470 |
| KNN | MAP | 0.2828 | **0.2853** | 0.2494 | 0.2656 | 0.2646 | 0.2665 |
| | Rel.Ret. | 463 | 465 | 466 | 468 | 465 | 467 |
| Decision Tree | MAP | 0.2856 | **0.2865** | 0.2518 | 0.2646 | 0.2668 | 0.2676 |
| | Rel.Ret. | 465 | 466 | 466 | 468 | 467 | 467 |
| Random Forest | MAP | 0.285 | **0.2861** | 0.2515 | 0.2662 | 0.2673 | 0.2686 |
| | Rel.Ret. | 465 | 466 | 465 | 469 | 468 | 468 |

Table 7.6: MAP scores of different hybrid machine learning-based decompounding evaluation in the Hindi (50 T queries)

| | ↓ Parameter \| R.M. → | BM25 | TF-IDF | In_expC2 | BB2 | InL2 | Hiem_LM |
|---|---|---|---|---|---|---|---|
| Base line | MAP | 0.4429 | **0.4439** | 0.3266 | 0.3606 | 0.3797 | 0.3951 |
| | Rel.Ret. | 1682 | 1679 | 1640 | 1657 | 1652 | 1668 |
| SVM | MAP | 0.4471 | **0.4478** | 0.3612 | 0.3883 | 0.3856 | 0.3824 |
| | Rel.Ret. | 1684 | 1683 | 1670 | 1674 | 1673 | 1673 |
| CRF | MAP | 0.4465 | **0.4472** | 0.362 | 0.385 | 0.3845 | 0.3843 |
| | Rel.Ret. | 1684 | 1683 | 1669 | 1673 | 1673 | 1672 |
| KNN | MAP | 0.4649 | **0.4664** | 0.3802 | 0.4143 | 0.4249 | 0.403 |
| | Rel.Ret. | 1682 | 1678 | 1666 | 1669 | 1668 | 1662 |
| Decision Tree | MAP | 0.4545 | **0.4563** | 0.381 | 0.4186 | 0.4292 | 0.4062 |
| | Rel.Ret. | 1681 | 1677 | 1663 | 1671 | 1672 | 1664 |
| Random Forest | MAP | 0.4556 | **0.4573** | 0.3826 | 0.4154 | 0.4252 | 0.404 |
| | Rel.Ret. | 1682 | 1678 | 1663 | 1670 | 1669 | 1663 |

### 7.5.3 Effect of deep learning-based decompounding on retrieval

In the third set of experiments, we propose different deep learning-based decompounding models and evaluate their effectiveness from an IR perspective. The impact of different deep

Table 7.7: MAP scores of different hybrid machine learning-based decompounding evaluation in the Sanskrit (50 T queries)

| | ↓ Parameter \| R.M. → | BM25 | TF-IDF | In_expC2 | BB2 | InL2 | Hiem_LM |
|---|---|---|---|---|---|---|---|
| Base line | MAP | 0.4208 | 0.4218 | 0.3995 | 0.4079 | 0.4025 | **0.4339** |
| | Rel.Ret. | 389 | 386 | 385 | 388 | 387 | 389 |
| SVM | MAP | 0.4269 | 0.4278 | 0.4113 | 0.4183 | 0.4131 | **0.4287** |
| | Rel.Ret. | 389 | 387 | 387 | 389 | 388 | 388 |
| CRF | MAP | 0.4272 | 0.4276 | 0.4108 | 0.4186 | 0.4139 | **0.4295** |
| | Rel.Ret. | 389 | 387 | 386 | 389 | 389 | 388 |
| KNN | MAP | 0.4282 | 0.43 | 0.4185 | 0.421 | 0.4147 | **0.4331** |
| | Rel.Ret. | 390 | 388 | 388 | 390 | 389 | 389 |
| Decision Tree | MAP | 0.4286 | 0.4294 | 0.4178 | 0.4218 | 0.4135 | **0.4341** |
| | Rel.Ret. | 390 | 388 | 387 | 391 | 389 | 389 |
| Random Forest | MAP | 0.428 | 0.429 | 0.4172 | 0.4221 | 0.4208 | **0.4345** |
| | Rel.Ret. | 390 | 388 | 387 | 391 | 390 | 390 |

learning-based decompounding models for Marathi, Hindi and Sanskrit languages is shown in Table 7.8, 7.9, and 7.10. Deep learning-based decompounding models are seen to considerably improve the effectiveness of an IR system. However, the impact of deep learning-based decompounding models varies from one language to another. The RNN-based decompounding model improves MAP score of 23.66% in Marathi, 16.28% in Hindi and 2.38% in Sanskrit compared to the baseline approach (In_expC2 model). Similarly, the LSTM and GRU models improve MAP score by 23.94% and 23.66% in Marathi, 17.02% and 16.84% in Hindi, 4.94% and 1.15% in Sanskrit. We observe no significant difference in MAP score between the vanilla RNN, LSTM, or GRU and their bidirectional counterparts. However, the Bi-RNN-A-based decompounding model improves MAP score by 27.61% in Marathi, 18.18% in Hindi and 6.1% in Sanskrit compared to the baseline approach (In_expC2 model). Similarly, the Bi-LSTM-A and Bi-GRU-A decompounding models improve MAP score by 28.02% and 27.98% in Marathi, 17.11% and 17.45% in Hindi, 5.24% and 2.83% in Sanskrit. Among the different deep learning-based decompounding models, the attention-based models outperform the other models in the Indian language IR. We also observe that the deep learning-based models for word-decompounding outperform the hybrid machine learning-based and corpus-based models in Indian language IR.

### 7.5.4 Retrieval Effectiveness Analysis: some insights

We observe that the decompounding methods typically improve MAP scores in different retrieval models across the Indian languages. To get more insights into the effect of decompounding, we perform a query-by-query analysis. Here, we consider the In_expC2 model for Marathi and Hindi languages and the BB2 model for Sanskrit as the best-performing models for their

Table 7.8: MAP scores of different deep learning-based decompounding evaluation in the Marathi (39 T queries)

| | ↓ Parameter \| R.M. → | BM25 | TF-IDF | In_expC2 | BB2 | InL2 | Hiem_LM |
|---|---|---|---|---|---|---|---|
| Base line | MAP | 0.2833 | **0.284** | 0.2155 | 0.2533 | 0.2504 | 0.2692 |
| | Rel.Ret. | 458 | 458 | 458 | 462 | 458 | 462 |
| RNN | MAP | 0.2988 | **0.2992** | 0.2665 | 0.2765 | 0.2749 | 0.277 |
| | Rel.Ret. | 488 | 487 | 484 | 489 | 488 | 484 |
| LSTM | MAP | 0.2997 | **0.3001** | 0.2671 | 0.278 | 0.2783 | 0.2773 |
| | Rel.Ret. | 487 | 486 | 485 | 487 | 486 | 482 |
| GRU | MAP | **0.2998** | 0.298 | 0.2665 | 0.2776 | 0.2757 | 0.2804 |
| | Rel.Ret. | 486 | 485 | 485 | 488 | 486 | 480 |
| Bi-RNN | MAP | 0.2994 | **0.2999** | 0.2635 | 0.2761 | 0.2767 | 0.2818 |
| | Rel.Ret. | 490 | 489 | 485 | 489 | 490 | 487 |
| Bi-LSTM | MAP | 0.2986 | **0.2988** | 0.2731 | 0.2835 | 0.2809 | 0.2791 |
| | Rel.Ret. | 485 | 485 | 484 | 487 | 486 | 483 |
| Bi-GRU | MAP | 0.3039 | **0.3041** | 0.2733 | 0.2848 | 0.2826 | 0.2838 |
| | Rel.Ret. | 485 | 484 | 486 | 489 | 484 | 483 |
| Bi-RNN-A | MAP | **0.3029** | 0.3014 | 0.275 | 0.2827 | 0.2806 | 0.2834 |
| | Rel.Ret. | 488 | 486 | 482 | 488 | 487 | 485 |
| Bi-LSTM-A | MAP | 0.3035 | **0.3038** | 0.2759 | 0.2884 | 0.2855 | 0.2836 |
| | Rel.Ret. | 488 | 487 | 485 | 489 | 489 | 485 |
| Bi-GRU-A | MAP | 0.3075 | **0.3079** | 0.2758 | 0.2884 | 0.2863 | 0.2857 |
| | Rel.Ret. | 489 | 489 | 488 | 493 | 489 | 486 |

Table 7.9: MAP scores of different deep learning-based decompounding evaluation in the Hindi (50 T queries)

| | ↓ Parameter \| R.M. → | BM25 | TF-IDF | In_expC2 | BB2 | InL2 | Hiem_LM |
|---|---|---|---|---|---|---|---|
| Base line | MAP | 0.4429 | **0.4439** | 0.3266 | 0.3606 | 0.3797 | 0.3951 |
| | Rel.Ret. | 1682 | 1679 | 1640 | 1657 | 1652 | 1668 |
| RNN | MAP | 0.446 | **0.4463** | 0.3798 | 0.3973 | 0.4158 | 0.3944 |
| | Rel.Ret. | 1686 | 1686 | 1673 | 1683 | 1672 | 1667 |
| LSTM | MAP | 0.4514 | **0.454** | 0.3822 | 0.4009 | 0.4202 | 0.3981 |
| | Rel.Ret. | 1686 | 1686 | 1672 | 1682 | 1672 | 1667 |
| GRU | MAP | 0.4465 | **0.4466** | 0.3816 | 0.3982 | 0.4158 | 0.3944 |
| | Rel.Ret. | 1686 | 1686 | 1673 | 1682 | 1672 | 1667 |
| Bi-RNN | MAP | 0.4461 | **0.4481** | 0.3811 | 0.3986 | 0.4178 | 0.3958 |
| | Rel.Ret. | 1686 | 1686 | 1674 | 1682 | 1672 | 1669 |
| Bi-LSTM | MAP | 0.446 | **0.4482** | 0.3814 | 0.3985 | 0.4179 | 0.3964 |
| | Rel.Ret. | 1686 | 1686 | 1672 | 1682 | 1672 | 1667 |
| Bi-GRU | MAP | 0.4462 | **0.4482** | 0.3812 | 0.3985 | 0.4178 | 0.3967 |
| | Rel.Ret. | 1686 | 1686 | 1672 | 1682 | 1672 | 1667 |
| Bi-RNN-A | MAP | 0.4534 | **0.4552** | 0.386 | 0.4037 | 0.4223 | 0.4008 |
| | Rel.Ret. | 1686 | 1686 | 1674 | 1683 | 1673 | 1669 |
| Bi-LSTM-A | MAP | 0.4479 | **0.4499** | .3825 | 0.4003 | 0.4188 | 0.3985 |
| | Rel.Ret. | 1686 | 1686 | 1672 | 1683 | 1672 | 1667 |
| Bi-GRU-A | MAP | 0.4483 | **0.4503** | 0.3836 | 0.4003 | 0.42 | 0.3987 |
| | Rel.Ret. | 1686 | 1686 | 1673 | 1683 | 1672 | 1669 |

Table 7.10: MAP scores of different deep learning-based decompounding evaluation in the Sanskrit (50 T queries)

| | ↓ Parameter \| R.M. → | BM25 | TF-IDF | In_expC2 | BB2 | InL2 | Hiem_LM |
|---|---|---|---|---|---|---|---|
| Base line | MAP | 0.4208 | 0.4218 | 0.3995 | 0.4079 | 0.4025 | **0.4339** |
| | Rel.Ret. | 389 | 386 | 385 | 388 | 387 | 389 |
| RNN | MAP | 0.4253 | 0.426 | 0.4073 | 0.4157 | 0.4121 | **0.4369** |
| | Rel.Ret. | 388 | 384 | 388 | 391 | 387 | 388 |
| LSTM | MAP | 0.4286 | 0.4286 | 0.3867 | 0.413 | 0.4224 | **0.4361** |
| | Rel.Ret. | 389 | 389 | 386 | 390 | 391 | 388 |
| GRU | MAP | 0.4173 | 0.4168 | 0.3834 | 0.3896 | 0.3995 | **0.4389** |
| | Rel.Ret. | 386 | 385 | 382 | 387 | 387 | 389 |
| Bi-RNN | MAP | 0.4319 | 0.4313 | 0.411 | 0.4153 | 0.4105 | **0.4385** |
| | Rel.Ret. | 389 | 386 | 387 | 388 | 390 | 389 |
| Bi-LSTM | MAP | 0.4276 | 0.4289 | 0.4101 | 0.4191 | 0.4138 | **0.4343** |
| | Rel.Ret. | 388 | 388 | 386 | 391 | 388 | 387 |
| Bi-GRU | MAP | 0.4351 | 0.4366 | 0.413 | 0.4235 | 0.4258 | **0.4421** |
| | Rel.Ret. | 390 | 389 | 386 | 390 | 389 | 390 |
| Bi-RNN-A | MAP | 0.4352 | 0.435 | 0.4224 | 0.4328 | 0.4244 | **0.4447** |
| | Rel.Ret. | 390 | 387 | 388 | 389 | 388 | 390 |
| Bi-LSTM-A | MAP | 0.4376 | 0.4377 | 0.42 | 0.4229 | 0.4236 | **0.4408** |
| | Rel.Ret. | 388 | 388 | 387 | 390 | 388 | 388 |
| Bi-GRU-A | MAP | 0.4272 | 0.4273 | 0.3974 | 0.4017 | 0.4139 | **0.4366** |
| | Rel.Ret. | 389 | 388 | 386 | 388 | 387 | 389 |

respective languages. On closer observation, we find that the Bi-LSTM-A model improves the average precision scores in 28 topics and reduces in 5 topics in Marathi. The percentage change in the average precision score for each query is shown in Figure 7.6. Similarly, the Bi-RNN-A model improves the average precision scores in 42 and 29 topics in Hindi and Sanskrit languages and reduces in 8 and 15 topics, respectively. The percentage change in the average precision score for each query is shown in Fig 7.7 and 7.8. In Sanskrit, some queries provide phenomenal retrieval effectiveness. For example, in Topic 47,नवदेहल्यां भाजपादलस्य राष्ट्रियकार्यकारिण्युपवेशनम् (BJP party's national executive meeting held in New Delhi), the decompounding model improves the average precision score by 102.93%. A similar observation is found in Topic 34 पि–एन–बि धनापहरणम् (corruption in P.N.B. bank) (improvement of 86.59%).

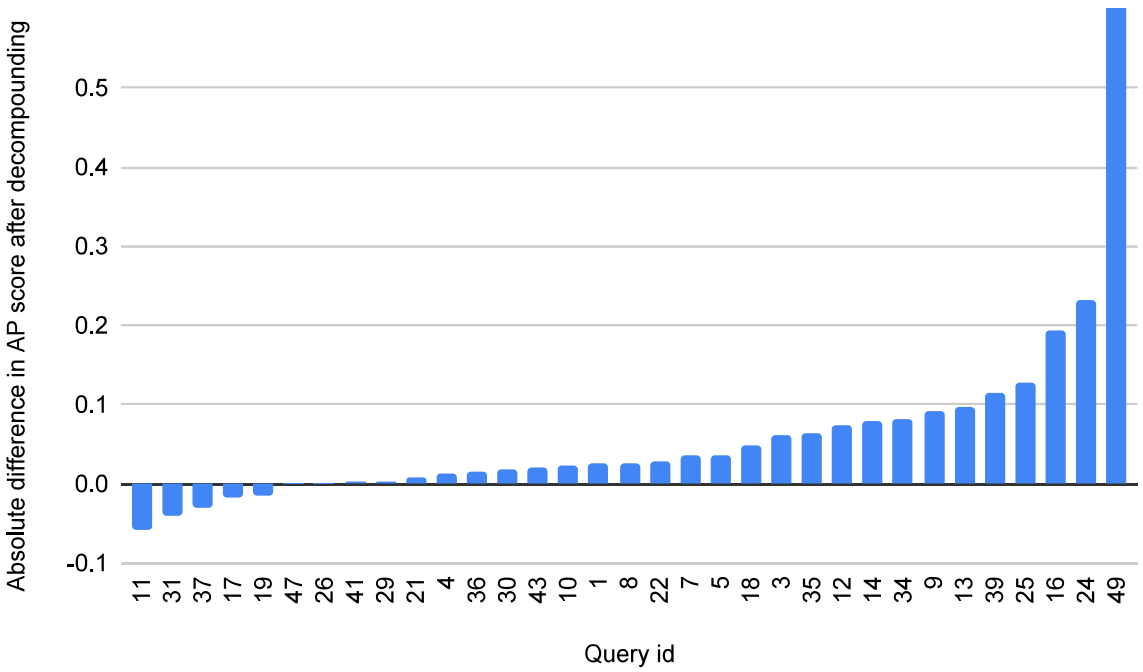


Figure 7.6: A query by query evaluation in the Marathi by In_expC2 model

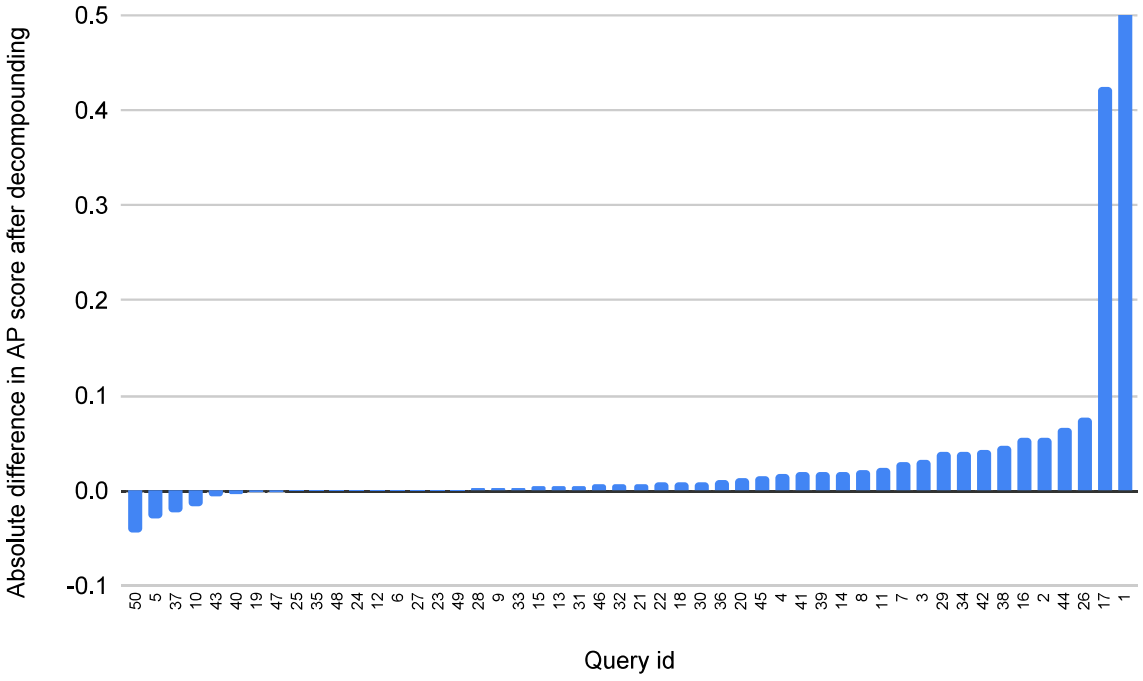


Figure 7.7: A query by query evaluation in the Hindi by In_expC2 model

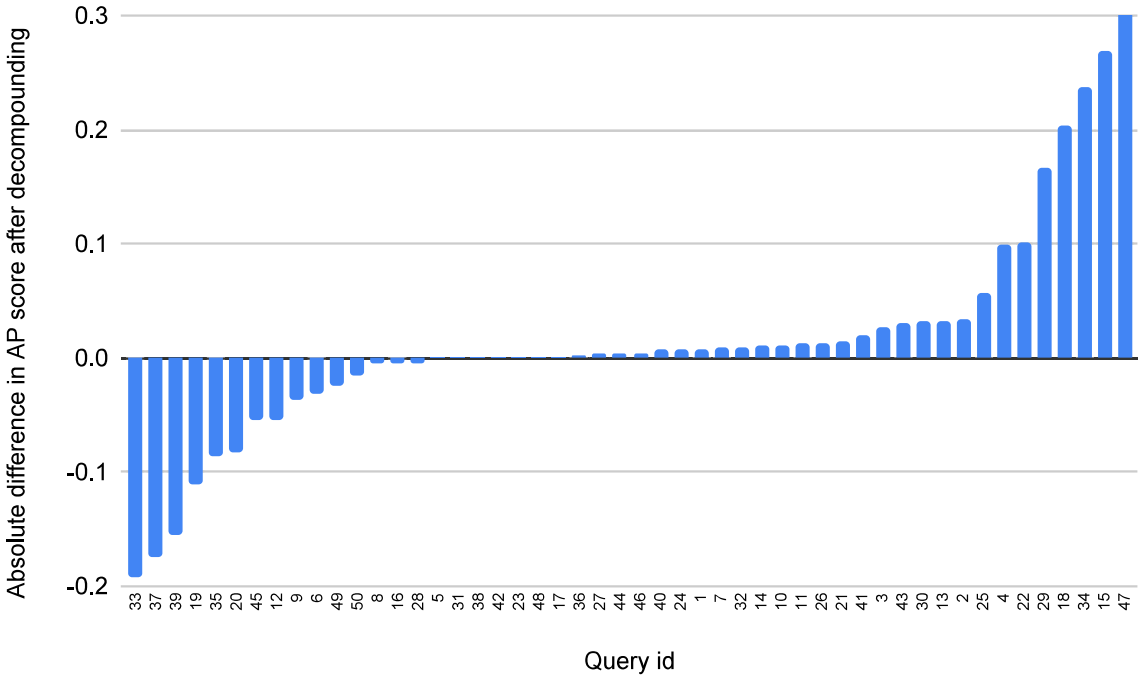


Figure 7.8: A query by query evaluation in the Sanskrit by BB2 model

## 7.6 Discussion

In the first set of experiments, we evaluate the effect of different corpus-based decompounding models (shown in tables 7.2, 7.3, and 7.4) in Indian language IR. Different corpus-based word-decompounding models are observed to improve the effectiveness of IR systems. Among them, the frequency-based approach provides the poorest effectiveness, while the co-occurrence-based model performs best in Indian language IR. The MAP score improvement in Marathi was 12.94% (the frequency-based approach) and 14.98% (co-occurrence-based approach) here, Ganguly et al. [43] and Monz and Rijke [89] reported that the corpus-based decompounding model improved a MAP score of 2.72% in Bengali, 6.1% in Dutch, and 9.6% in German. In the European language, the frequency-based decompounding approach performs best in German-English machine translation [65] but performs poorly in morphologically rich Indian language IR. The fundamental reason is that splitting a compound word based on the frequency of constituent words changes the word's original meaning in many cases in non-Indian languages. E.g., the compound word 'underworld' is split into 'under' and 'world' due to high-frequency constituents in the collection. Here, the compound splitting is syntactically correct but changes the semantic meaning of the word. Hence, compound splitting reduces the effectiveness of an IR system. The

characteristics of the Indian language compound words differ from the European ones, so the effectiveness is affected disruptively.

In the second set of experiments, we evaluate the effect of different hybrid machine learning-based decompounding models (shown in tables 7.5, 7.6, and 7.7) in Indian language IR. We notice that different hybrid machine learning-based models improve effectiveness in Indian language IR. The maximum MAP score gain in a hybrid machine learning-based decompounding model in Marathi is 22.27%. In contrast, the maximum MAP score gained by a corpus-based decompounding model in German is 9.6% [89]. No particular machine learning-based model performs best in Indian languages. We also observe that the hybrid machine learning-based models outperform the corpus-based models in general. The fundamental reason is that the corpus-based models primarily comprise the dictionary and could not split the *sandhi*-ed compound word efficiently. But, the hybrid machine learning-based models split both simple concatenation and *sandhi*ed compound words better. Hybrid machine learning-based models were also seen to outperform the corpus-based models in Dutch and German IR [89].

In the third set of experiments, we evaluate the effect of different deep learning-based decompounding models (shown in Tables 7.8, 7.9, and 7.10) in Indian language IR. We find that different deep learning-based word-decompounding models improve the effectiveness of an IR systems. In Marathi and Hindi languages, the Bi-LSTM-A and Bi-RNN-A models are seen to perform brilliantly, improving MAP scores by 28.02% and 18.18%, respectively. Similarly, in Sanskrit, the Bi-RNN-A model performs the best and improves a MAP score by 6.1%. The deep learning-based models provide better effectiveness in Marathi than Hindi and Sanskrit. The primary reason is that we used here a Marathi dataset particularly developed for a multi-word study by Singh et al. [129]. b) The number of relevant documents in Sanskrit is less than that of the other two languages, as the Sanskrit dataset is the smallest. No significant MAP score differences between the vanilla RNN, LSTM, and GRU and their bidirectional counterparts. Among the different deep learning-based models, the attention-based models outperform the other models in different Indian languages. The maximum MAP score gain in any deep learning-based decompounding models in the Marathi is 28.02%, whereas that by a corpus-based decompounding model in German is 9.6% [89]. Deep learning-based models outperform the corpus-based and hybrid machine learning-based models in Indian languages. The fundamental reasons are: a) deep learning models provide better location and splitting prediction accuracy, and b) Indian languages are morphologically rich and comprise various compound words.

Among the different retrieval models, we notice that the probabilistic models (BM25 and TF-IDF) offer the best MAP scores in Marathi and Hindi. The language model provides mod-

erate effectiveness. Among the DFR-based models (In_expC2, BB2 and InL2), the In_expC2 model provides a low MAP score. In Sanskrit, the language model provides the best MAP score, and the probabilistic models and DFR-based models provide similar MAP scores.

## 7.7 Summary

Decompounding is an important pre-processing step in the Indian language IR. In this study, we explore three decompounding techniques, i.e., corpus-based, hybrid machine learning-based and deep learning-based models in different Indian languages IR. The above experiments show that decompounding improves retrieval effectiveness in the IR domain. Among the corpus-based decompounding models, the co-occurrence-based model provides the best MAP score across the Indian languages. Other techniques (frequency-based, maximum branching factor and tri-based) are ineffective for the Indian languages studied here. We also observe that the corpus-based models provide poor effectiveness compared to other decompounding models. In hybrid machine learning-based models, the CRF model provides the best MAP score in Marathi IR. Similarly, the Random Forest and KNN models outperform the others in Hindi and Sanskrit IR. We observe that the hybrid machine learning-based models outperform the corpus-based models in the Indian language IR. In deep learning-based models, the Bi-LSTM-A model performs best in Marathi IR. Similarly, the Bi-RNN-A model outperforms other models in Hindi and Sanskrit IR. Among the different deep learning-based models, the attention-based models outperform the other deep learning-based models in different Indian languages. Moreover, the deep learning-based models outperform the corpus-based and hybrid machine learning-based models in Indian language IR. While evaluating different IR models, we notice that the In_expC2 model provides the best retrieval effectiveness gain in Marathi and Hindi IR. Similarly, the BB2 model is the most effective in Sanskrit IR.