Natural Language Processing (2024), 1–31
doi:10.1017/nlp.2024.16

CAMBRIDGE UNIVERSITY PRESS

ARTICLE

A case study on decompounding in Indian language IR

Siba Sankar Sahu[1] and Sukomal Pal[2]

[1]School of Computer Science Engineering and Technology, Bennett University, Greater Noida, Uttar Pradesh, India and [2]Indian Institute of Technology (BHU), Department of Computer Science and Engineering, Varanasi, Uttar Pradesh, India
Corresponding author: Siba Sankar Sahu; Email: sibasankarsahu.rs.cse17@itbhu.ac.in, siba.sahu@bennett.edu.in

(Received 29 November 2022; revised 7 August 2023; accepted 3 November 2023)

Special Issue on '**Natural Language Processing Applications for Low-Resource Languages**'

Abstract

Decompounding is an essential preprocessing step in text-processing tasks such as machine translation, speech recognition, and information retrieval (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? This study proposes 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 best and improves mean average precision (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.

Keywords: Information retrieval; Text segmentation

1. Introduction

In natural languages, compound words are formed by concatenating two or more words. Compound words create challenge for different natural language processing (NLP) applications like machine translation, text classification, speech recognition, and information retrieval (IR). In the IR domain, compound words generate a vocabulary mismatch between a query and the documents in the collection. For example, if a query comprises the term 'book', and the document includes 'notebook' it may be retrieved at a lower rank than the document consisting of 'note' and 'book'. Hence, decompounding is an essential preprocessing step for compounding languages like German, Finnish, Dutch, and Bengali, where compound splitting improves the effectiveness of an IR system Braschler and Ripplinger (2004), Ganguly et al. (2013). Although the effect of decompounding has been extensively studied in European languages, it is less explored in Indian languages like Marathi, Hindi, and Sanskrit. This study explores and evaluates the effect of decompounding in Indian languages IR.

The Indian language compound words are broadly categorised into two types. First is a typical compound word: a simple concatenation of two or more words. E.g., 'note' and 'book' are combined to form a 'notebook'. Second is a sandhied compound word: according to sandhi rules, the tail constituent of the first component changes at the merging location (end of the first word and start of the second word). For example, 'vidya' and 'alaya' are combined to form 'vidyalaya'. These modifications make splitting a sandhied compound word challenging in Indian languages. An additional feature of the Indian language compound word is that one of the compound constituents may not be used independently as a meaningful word. For example, in 'upapradhanmantri', 'upa' does not have independent use on its own but used as the prefix of another word. A predefined prefix list prevents the wrong splitting of compound words. We also noticed that splitting a compound word such as 'underworld' into 'under' and 'world' changes the original meaning of a compound word. Splitting of these compound words is syntactically correct but generates semantically unrelated meanings. Hence, we avoid semantically incorrect compound word splitting to improve the effectiveness of an IR system.

Different researchers proposed and evaluated different decompounding models such as corpus-based (Adda-Decker 2003; Deepa et al. 2004; Ganguly et al. 2013), machine learning based Ajees and Graham (2018), and deep learning-based (Hellwig and Nehrdich 2018; Aralikatte et al. 2018a; Hellwig 2015) in Asian and European languages. Among the different decompounding models, they observed that the corpus-based decompounding models perform poorly in morphological-rich Indian languages. The primary reason is the inherent characteristics of Indian languages. The performance and drawback of the corpus-based decompounding model in Sanskrit are described in Bhardwaj et al. (2018). Different machine learning-based models improve effectiveness in morphologically rich languages. In recent years, the deep learning-based models outperform the corpus-based decompounding models in Sanskrit (Dave et al. 2021; Hellwig and Nehrdich 2018; Aralikatte et al. 2018a; Hellwig 2015).

To the best of our knowledge, no previous studies exist on the effect of decompounding models in Indian languages IR Koehn and Knight (2003), Ajees and Graham (2018), Aralikatte et al. (2018a). This study explores the impact of different decompounding models, i.e. corpus-based, hybrid machine learning-based, and deep learning-based in Indian language IR. The corpus-based model comprises dictionary-based approaches. The hybrid machine learning-based model consists of a dictionary and machine learning-based approaches. We also explore different deep learning-based models. We used the Terrier[a] retrieval system to evaluate decompounding models' effectiveness in the IR domain.

Primarily, we investigated the following research questions (RQs).

RQ1: *Does word decompounding impact the Indian language IR? If yes, to what extent?*

RQ2: *Can corpus-based decompounding models be used in the Indian language IR? If yes, how?*

RQ3: *Can machine learning and deep learning-based decompounding models be applied in the Indian language IR? If yes, how?*

RQ4: *Among the different decompounding models (corpus-based, hybrid machine learning-based, and deep learning-based), which provides the best effectiveness in the IR domain?*

RQ5: *Among the different IR models, which provides the best effectiveness from the IR perspective?*

Therefore, we proposed different decompounding models in Indian languages and evaluated their effectiveness in the IR domain. We also summarise the benefits and drawbacks of decompounding models in Indian languages. The contributions of this article can be summarised as follows.

(1) We explore and evaluate three decompounding models: corpus-based, hybrid machine learning-based, and deep learning-based models in Indian languages.

[a]http://terrier.org/

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

(2) We propose two novel decompounding models (machine learning and deep learning-based).
(3) We evaluate the effectiveness of different decompounding models in different languages and compare them with the no decompounding approach in the IR domain.
(4) Analysis has been done for different decompounding models and suggests the best for a language.
(5) Analysis has been done for different IR models and suggests the best for a language.

The rest of the paper is organised as follows. Section 2 presents an overview of decompounding approaches in different languages. We elaborate on the problem statement in Section 3. Section 4 describes the decompounding approaches implemented in Indian languages. A brief explanation of different IR models is presented in Section 6. Section 7 describes the evaluation metrics used in the experimentation. The test collection statistic is presented in Section 8. The experimental setup is presented in Section 9 followed by the experimental results in Section 10. More insightful discussions have been summarised in Section 11. Finally, we conclude with future work in Section 12.

2. Background and related work

In a morphologically rich language, compound words are generated by concatenating two or more words, using sandhi[b] principles, or combining several morphemes using linking elements. Different compound words in a lexicon affect the performance of computational tasks like machine translation, information extraction, and creating OOV words in the dictionary. So, decompounding is an essential preprocessing step in the text analysis domain. Early literature suggests that the decompounding models primarily use basic 'mechanical' segmentation methods based on string matching. The most common practice in this approach is the dictionary-based method. Different machine learning-based decompounding models have been proposed in the last decade. These models use a feature engineering approach to perform the decompounding operation. In recent years, deep learning-based decompounding models have improved the effectiveness of different morphologically rich European and Asian languages. We formalised the state-of-the-art decompounding models in three subsections: corpus-based, machine-learning, and deep learning-based.

2.1 Corpus-based decompounding methods

Koehn and Knight (2003) proposed a corpus-based decompounding model in German-English statistical machine translation. They used the geometric mean of word frequency to detect the splitting point location. They pointed out that decompounding algorithms improve the efficiency of machine translation. Leveling et al. (2011) investigated a corpus-based decompounding model in German-English cross-language patent retrieval. They noticed that the decompounding model reduces the OOV and improves the effectiveness of the IR system. Hollink et al. (2004) evaluated the effect of stemming, lemmatisation, and decompounding algorithms in eight European languages. They observed that the corpus-based decompounding model improves retrieval effectiveness in Dutch, Finnish, German, and Swedish languages. The decompounding algorithm improves an mean average precision (MAP) score of 4–18.7% in different morphologically rich European languages. Airio (2006) examined the impact of stemming, lemmatisation, and decompounding methods on monolingual and bilingual retrieval. They noticed that the decompounding algorithm does not improve effectiveness in English, Finnish, German, and Swedish monolingual retrieval. However, the decompounding method enhances effectiveness in English to Dutch,

[b]The word Sandhi means placing together.

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

Finnish, German, and Swedish bilingual retrieval. Rigouts Terryn et al. (2016) investigated the impact of a decompounding module on the Dutch hypernym detection. They observed that the decompounding module improves precision and recall.

Braschler and Ripplinger (2004) evaluated the effects of stemming and decompounding techniques in German IR. They found that stemming and careful decompounding enhance the effectiveness of IR systems. Alfonseca et al. (2008) presented a language-independent decompounding method in German. They observed that the decompounding method improves precision, recall, and accuracy in morphologically complex European languages. Laureys et al. (2002) evaluated the impact of the decompounding module in large vocabulary continuous speech recognition (LVCSR) in Dutch. The decompound module combines a rule-based approach with statistical pruning. They noticed that the decompounding module decreased the 30% lexicon size and improved the 11% word error rate (WER) in the Dutch broadcast news recognition. Erbs et al. (2015) explored the effect of a decompounding model in keyphrase extraction. They found that the decompounding enhances the word frequency count and identifies more keyphrase candidates in German.

In the past decade, a few studies were also conducted on the corpus-based decompounding models in the Indian languages from NLP and IR perspectives. Deepa et al. (2004) used a trie-based dictionary for the splitting of a compound word presented in Hindi speech synthesis. They showed that the algorithm has a split rate of 92–96% of the input compound words, and the percentage of splitting accuracy is 83–87%. Devadath et al. (2014) proposed a hybrid sandhi splitting method in Malayalam. They applied statistical methods and predefined character-level sandhi rules. They observed that the hybrid sandhi splitting method outperformed the rule-based approach. Sahu and Mamgain (2019) evaluated a corpus-based decompounding model in Sanskrit and improved the splitting accuracy by different ranking methods. They found that the decompounding model enhances precision, recall, and accuracy. Ganguly et al. (2013) observed that relaxed and co-occurrence-based constituent selection techniques outperform the standard frequency-based decompounding approach in Bengali IR.

2.2 Machine learning-based decompounding methods

Pellegrini and Lamel (2009) investigated an unsupervised data-driven decompounding method in automatic speech recognition for Amharic text. They observed that the decompounding method reduces the out-of-vocabulary (OOV) words from 35% to 50% and slightly reduces the WER. Daiber et al. (2015) proposed an analogy-based greedy unsupervised decompounding algorithm for German-to-English machine translation. They noticed that the decompounding algorithm is highly effective for ambiguous compound words. They found that the semantic analogy-based algorithm outperformed the frequency-based approach in German-to-English machine translation. Riedl and Biemann (2016) presented an unsupervised decompounding method in Dutch and German. They observed that the unsupervised method outperformed existing decompounding methods in German; however, the rule-based approach provided better effectiveness in Dutch.

Haruechaiyasak et al. (2008) evaluated the effect of word segmentation in Thai. They looked at two different word segmentation methods. One is a dictionary-based, and the other is a machine-learning approach. In the dictionary approach, they investigated longest matching and maximal matching techniques. In the machine learning approach, they investigated naive bayes, decision tree, support vector machine, and conditional random field techniques. They observed that the conditional random field (CRF) outperformed all other word segmentation methods. Ganguly et al. (2022) proposed a stochastic word transformation skip-gram algorithm, which improves the representation of the compositional compound words by leveraging information from the contexts of their constituents. They noticed that addressing the compounding effect as a part of the word embedding objective outperforms existing compounding-specific post-transformation-based approaches on word semantics and word polarity predictions. Minixhofer et al. (2023) built

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

a dataset of 255k compound and non compound words across 56 diverse languages. They evaluated the effect of the decompounding method on the large language models (LLMs). They found that the LLMs perform poorly in different languages. Hence, they built self-supervised models that outperformed the unsupervised decompounding models and improved accuracy by 13.9%.

Ajees and Graham (2018) presented a hybrid decompounding method in Malayalam. The decompounding method integrates a rule-based and machine-learning approach. The decompounding module can be used as a preprocessing step in machine translation, question-answering, and extractive summarisation applications. Das et al. (2012) evaluated the hybrid compound splitting method in the Malayalam text. They used a machine learning approach to classify the word as being split or not. Subsequently, a rule-based sandhi splitter is used to split the word. Shree et al. (2016) presented a CRF tool to split a compound word in the Kannada language. They evaluated the model's effectiveness using five different combinations of training data. They achieved a compound splitting accuracy of 91.19%. Xue (2003) proposed a maximum entropy model for Chinese word segmentation. They found that the model can handle unknown words more efficiently than the maximum matching algorithm. They observed that the maximum entropy model provides precision and recall of 95.01% and 94.94%, respectively.

2.3 Deep learning-based decompounding methods

Chen et al. (2015) proposed an LSTM architecture for Chinese word segmentation. They experimented with different LSTM layers and dropout rates. They observed that the LSTM model provides a precision, recall, and f-score of 97.5%, 97.3%, and 97.4%, respectively. Kitagawa and Komachi (2018) also presented an LSTM architecture for Japanese word segmentation. They used a character-based n-gram embedding and a gold dictionary created from the test corpus. They noticed that the model achieves an F1-score of 98.67%. Premjith et al. (2018) investigated different deep learning-based decompounding methods in Malayalam. They found that the recurrent neural network (RNN), long short-term memory (LSTM), and gated recurrent units (GRU) provide an accuracy of 98.08%, 97.88%, and 98.16%, respectively. Hellwig (2015) presented a neural network-based decompounding model in Sanskrit. The neural network comprises an input layer and a hidden forward and backward layer. They used LSTM as a hidden layer to avoid the vanishing gradient problem. They observed that the decompounding model improves precision, recall, and accuracy. Reddy et al. (2018b) introduced a deep sequence to sequence (seq2seq) model that takes the sandhied string as the input and predicts the unsandhied string as output. The model comprises multiple layers of LSTM cells with attention. They achieved an improvement of the F-score of 16.79%.

Hellwig and Nehrdich (2018) developed a Sanskrit word segmentation model using character-level convolutional and RRNs. The model achieves a sentence splitting and string splitting accuracy of 85.2% and 96.7%, respectively. Aralikatte et al. (2018a) presented a double decoder (DD-RNN) with an attention model for the word decompounding in Sanskrit. They found that the model provides the splitting location and prediction accuracy of 95% and 79.5%, respectively, which outperforms the state of art by 20%. They also demonstrate the model's generalisation capability by applying the word segmentation method in Chinese. Dave et al. (2021) proposed a two-stage deep learning-based decompounding model in Sanskrit. In the first stage, they predict the sandhi window using the RNN model. In the second stage, the sandhi window is split into two words using the seq2seq model. They observed a location prediction and split prediction accuracy of 92.3% and 86.8%, respectively.

Few researchers developed software tools for decompounding methods in morphologically rich European and Asian languages. Biemann et al. (2008) developed a modular ASV toolbox for scientific and educational purposes. The tool supports different sub-modules such as linguistic annotation, classification, clustering, language detection, POS–tagging, named entity recognition, German noun compound splitting, and base form reduction for German, English, and Norwegian.

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

The tool split the compound word recursively using a trie-based data structure. In addition to the above, a prominent Sanskrit sandhi splitter is available on the web. Sachin (2007) proposed a JNU sandhi splitter.[c] The sandhi splitting tool is also called a Sanskrit sandhi recogniser and analyser. The tool provides an accuracy of 8.1% in Sanskrit split prediction. Similarly, UoH sandhi splitter[d] and INRIA sandhi splitter[e] provide a split prediction accuracy of 47.2% and 59.9%, respectively.

Based on the above findings, decompounding models play an important role in text analysis tasks. The effect of the decompounding method is thoroughly investigated in European and a few Asian languages. However, no previous studies exist on decompounding methods in Indian languages like Marathi, Hindi, and Sanskrit from an IR perspective. Hence, we evaluated different decompounding techniques in Indian languages in the IR domain. Our work is motivated by the earlier work of Koehn and Knight (2003), Ganguly et al. (2013), Ajees and Graham (2018), Aralikatte et al. (2018a).

3. 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*[f] 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. Three types of *Sandhi*s are defined in azwADyAyI[g] 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 and the first character of the second word. For example, 'jana' and 'adesha' are combined to form '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, 'sat' and 'jana' are combined to form '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, 'antaH' and 'gata' are combined to form '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 language compound word can be split into multiple ways. Hence, detecting the splitting point location is a

[c] http://sanskrit.jnu.ac.in/sandhi/viccheda.jsp
[d] https://sanskrit.uohyd.ac.in/scl/
[e] https://sanskrit.inria.fr/DICO/sandhi.fr.html
[f] The word Sandhi means placing together.
[g] https://www.britannica.com/topic/Ashtadhyayi

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

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.

4. Different decompounding methods

Decompounding is used as an essential preprocessing step 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.

4.1 Corpus-based decompounding approaches

We evaluate here some of the existing decompounding techniques that have provenly yielded high effectiveness in different morphologically rich European and Asian languages Koehn and Knight (2003), Deepa et al. (2004), Ganguly et al. (2013). 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.

4.1.1 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 (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 Koehn and Knight (2003).

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

4.1.2 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 Adda-Decker (2003).

4.1.3 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

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

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 A.1) and apply sandhi[h] rules to deal with the morphological features of Indian languages. However, the previously evaluated decompounding in Hindi speech synthesis Deepa et al. (2004) does not consider the morphological characteristics of Indian languages.

4.1.4 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 Ganguly et al. (2013). 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 (2). This approach aligns with the earlier work of the compound splitting method evaluated in Bengali IR Ganguly et al. (2013).

$$Overlap(w, c) = \frac{\left|D(w) \bigcap D(c)\right|}{min\{|D(w)|, |D(c)|\}} \tag{2}$$

4.2 Machine learning-based decompounding methods

Basic machine learning models such as support vector machine (SVM), decision trees, CRF, and random forests are widely used in different computational areas like text classification, part-of-speech (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 3. Post-training, the models are checked using compound words from the test dataset and evaluation is done by looking at the change in the retrieval effectiveness of different IR techniques. Here, we briefly describe different ML techniques used in this context, followed by the general steps adopted for these machine learning-based decompounding models.

[h] as described in *Ashtadhyayi* (azwADyAyI).

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

4.2.1 Decision tree approach

The decision tree Quinlan (1986) 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.

4.2.2 Random forest approach

Random forest Ho (1995) 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 of votes for the 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.

4.2.3 k-nearest neighbours approach

The k-nearest neighbours algorithm Cover and Hart (1967) 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, normalising 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.

4.2.4 Conditional random field approach

Machine learning models are broadly categorised into two types: the generative models and the discriminative models. A conditional random field Lafferty et al. (2001) 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.

4.2.5 Support vector machine approach

SVM Cortes and Vapnik (1995) is a supervised learning algorithm primarily used for classification tasks. Suppose we have a set of training samples $D = (x_1, y_1), \ldots, (x_m, y_m)$, where x_i is the feature vector of ith training sample, y_i is the output class of i-th training sample, and 'm' being the number of training samples. The main idea behind SVM is to classify the samples into two or more classes by a hyperplane. SVM finds a hyperplane that maximises its margin. SVM chooses the extreme points near the boundaries that help create a hyperplane. The extreme points are called support vectors; hence, the algorithm is called a support vector machine.

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

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 12) (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), 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 datasets. The training dataset 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 dataset.
(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.

4.3 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 Chen et al. (2015), Japanese Kitagawa and Komachi (2018), and Sanskrit Hellwig (2015), Hellwig and Nehrdich (2018), Reddy et al. (2018a). 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. (2014) 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

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

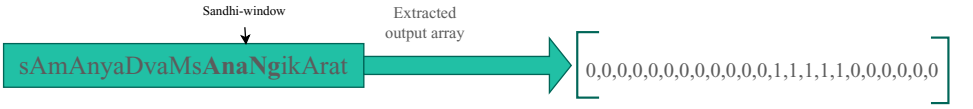

Figure 1. Sandhi-window (AnaNg) as the prediction target in a compound word.

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.[i]

4.3.1 Encoder-decoder model

The proposed encoder–decoder model is inspired by the character level sequence to sequence model Sutskever et al. (2014). 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. (2016) and Sanskrit[j] 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 Dave et al. (2021), Aralikatte et al. (2018b). 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 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 2. The model is trained by the Adam optimiser Kingma and Ba (2015) 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 setting is shown in Table 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.

4.3.2 Recurrent neural network (RNN)

Recurrent neural network (RNN) Medsker and Jain (2001) 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

[i] https://github.com/cse-iitbhu/Decompounding-code-and-Training-dataset
[j] http//sanskrit.uohyd.ac.in/Corpus/

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

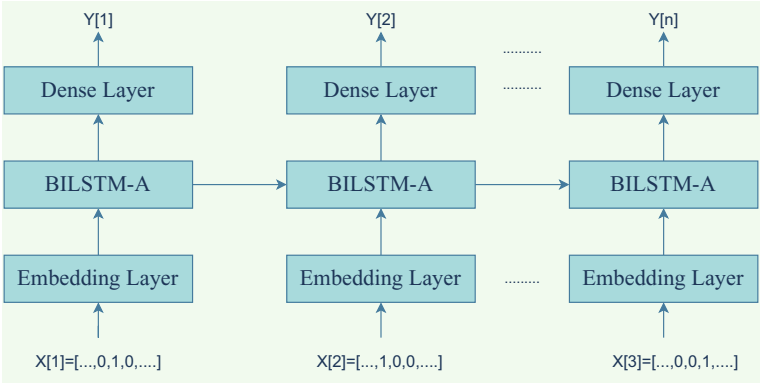

Figure 2. Model architecture for Sandhi Split – Stage 1.

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 (3). A basic RNN architecture is shown in Figure 3.

$$h(t) = f(h(t-1), x_t) \tag{3}$$

where f represents the nonlinear 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, which causes the vanishing gradient problem. We use other sequential models like LSTM and GRU to overcome the vanishing gradient problem.

4.3.3 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 Schmidhuber (1997) 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.

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

Table 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

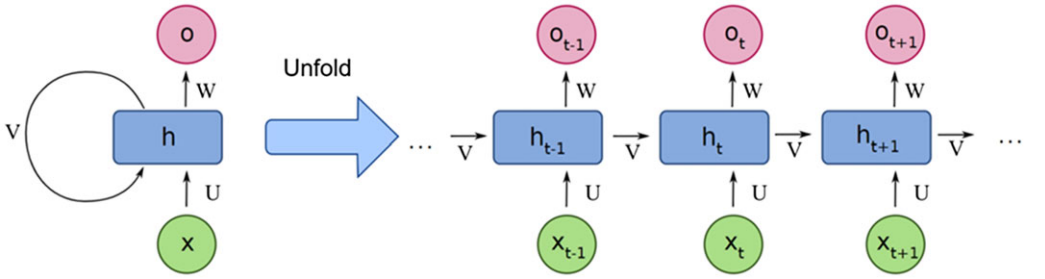

Figure 3. A basic RNN architecture.

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

(5) Mathematical expressions of different gates are as follows.

$$i_t = \sigma(W_{xi} \cdot x_t + W_{hi} \cdot h_{t-1} + W_{ci} \cdot c_{t-1} + b_i) \tag{4}$$

$$f_t = \sigma(W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1} + W_{cf} \cdot c_{t-1} + b_f) \tag{5}$$

$$o_t = \sigma(W_{xo} \cdot x_t + W_{ho} \cdot h_{t-1} + W_{co} \cdot c_{t-1} + b_o) \tag{6}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tanh(W_{xc} \cdot x_t + W_{hc} \cdot h_{t-1} + b_c) \tag{7}$$

A hidden state is represented as follows:

$$h_t = o_t \cdot \tanh(c_t) \tag{8}$$

$W_{xi}, W_{xf}, W_{xo}, W_{xc}$ 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 4.

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

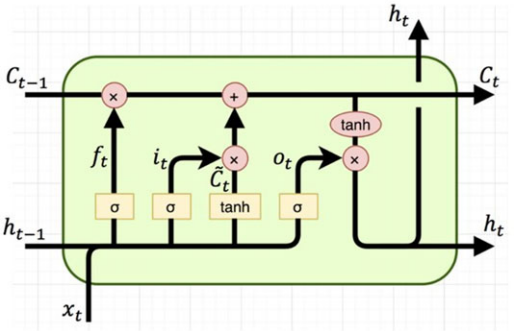

Figure 4. A single cell LSTM architecture.

4.3.4 Gated recurrent unit (GRU)

A gated recurrent unit is a variant of RNNs like RNN and LSTM introduced by Cho et al. (2014). GRU also solves the problem of vanishing gradient. GRU is like an LSTM network but is comprised of 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.

$$z_t = \sigma(W_z \cdot x_t + U_z \cdot h_{t-1} + b_z) \tag{9}$$

$$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1} + b_r) \tag{10}$$

$$h_t = (1 - z_t) \cdot \tanh(r_t \cdot U \cdot h_{t-1} + W \cdot x_t) + z_t \cdot h_{t-1} \tag{11}$$

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 5.

5. 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.[k]

[k]https://keras.io/

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

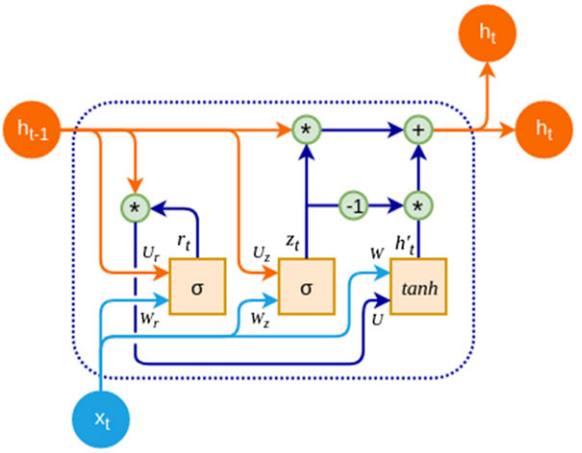

Figure 5. A single cell GRU architecture.

(2) During compound splitting, the prefix-based compound word (Table A1) (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[l] 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) 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).

6. Information retrieval framework

To evaluate the effectiveness of different decompounding models, we use an open-source search engine called Terrier[m] IR system for indexing and retrieving the document collection. We primarily used the following retrieval models: probabilistic retrieval models (BM25 and TF-IDF), DFR-based retrieval models (In_expC2, BB2, InL2), and the Language model. The mathematical expression of different retrieval models is given below. Each mathematical expression presents the weight of the term (t) in document (d), that is, w(t,d).

6.1 BM25 model

Okapi BM25 Robertson et al. (2000) is a popular model where BM stands for 'best matching'. The term weight of a query term in a document is given by Equation 12.

[l]http://fire.irsi.res.in
[m]http://terrier.org/

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

$$w(t,d) = tf_{\mathrm{d}} \cdot \left(\frac{\log\left(\frac{N-n+0.5}{n+0.5}\right)}{k_1 \cdot ((1-b) + b \cdot \frac{dl}{avdl} + tf_{\mathrm{d}})}\right) \tag{12}$$

6.2 TF-IDF model

In this model, the importance of a query term in a document is determined by the combination of term frequency (*tf*) and inverse document frequency (*idf*). Here, the *tf* represents the frequency of a term present in a given document, and *idf* is the inverse of document frequency (*df*) where *df* represents the number of documents that contain the given term. We call this quantity as term-weight ($w(t,d)$) of term t in document d. The scalar combination of all such $w(t,d)$s in the query terms is used in cosine-similarity. The TF-IDF model within the Terrier retrieval system uses Robertson's *tf* and Sparck Jones *idf* Sparck Jones (1972).

$$w(t,d) = Robertson\_tf \cdot idf \tag{13}$$

where,

$$Robertson\_tf = \frac{tf_{\mathrm{d}}}{k_1 \cdot ((1-b) + b \cdot \frac{dl}{avdl}) + tf_{\mathrm{d}}} \quad idf = \log\left(N/d_{\mathrm{f}} + 1\right) \quad tfn = tf \cdot \log_2\left(1 + c \cdot \frac{avdl}{l}\right)$$

dl : document length in number of terms

$avdl$: average document length

d_{f} : document frequency of term {t}

N : total number of documents in the collection

n : number of documents containing at least one term {t}

k_1 : term-frequency parameter, a constant

b : document length normalization parameter

n_{t} : document frequency of term {t}

tfn : normalised term frequency

c : free parameter

F : frequency of term {t} in the collection

We also consider other probabilistic models such as In_expC2, BB2, and InL2. These models come from the divergence from randomness (DFR) family Amati and Van Rijsbergen (2002). The DFR models are based on the idea that the more the divergence of the within-document term frequency from its frequency within the collection, the more information is carried by the term (t) in the document (d) Robertson et al. (1980).

6.3 In_expC2 model

In this model, the relevance score of a document is calculated by Equation (14).

$$w(t,d) = \frac{F+1}{n_t(tfn_e+1)}\left(tfn_e \cdot \log_2 \frac{N+1}{n_e+0.5}\right) \tag{14}$$

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

tfn_e denotes the normalised term frequency. A modified version of normalisation two is given by

$$tfn_e = tf \cdot \log_e (1 + c\frac{avdl}{l}) \tag{15}$$

6.4 BB2 model

In the Bose–Einstein model for randomness, the weight of a query term in a given document is given by the ratio of two Bernoulli's processes (first normalisation and second normalisation) for term frequency normalisation as shown in Equation (16).

$$w(t,d) = \frac{F+1}{n_t \cdot (tfn+1)}\Big[(-\log_2 (N-1) - \log_2 (e) + f(N+F-1, N+F-tfn-2) - f(F, F-tfn))\Big] \tag{16}$$

$$tfn = tf \cdot \log_2 (1 + c \cdot \frac{avdl}{l}) \tag{17}$$

where,

F : frequency of term t in the collection
n_t : document frequency of term t
tfn : normalised term frequency
c : free parameter

6.5 InL2 model

In this model, the weight of a query term in a document is given by the ratio of two Laplace processes (first normalisation and normalisation 2) for term frequency normalisation as shown in equation (18).

$$w(t,d) = \frac{1}{tfn+1}\Big(tfn \cdot \log_2 \frac{N+1}{n_t + 0.5}\Big) \tag{18}$$

6.6 Hiemstra language model

We also evaluate using a non-parametric probabilistic or language model proposed by Hiemstra (2001). The probability of a query term t in a document d depends upon the term frequency of t in both d and the entire corpus. This model uses a smoothing parameter λ to tune their contribution (a popular default value of λ is 0.15). The similarity between a query term t and document d is given by Equation 19.

$$P(D, T_1, T_2, T_3, \cdots T_n) = P(D) \prod_{i=1}^{n} ((1-\lambda_i)P(T_i) + \lambda_i P(T_i|D)) \tag{19}$$

where D is a document and T_i are query terms.

7. Evaluation metrics

We used the following metrics to investigate the effectiveness of decompounding models in the IR domain.

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

7.1 Precision

It is the fraction of relevant documents retrieved among the retrieved documents.

$$Precision = \frac{\left|\text{Relevant documents} \bigcap \text{Retrieved documents}\right|}{\text{Total number of retrieved documents}} \tag{20}$$

7.2 Recall

It is the fraction of relevant documents retrieved from a set of relevant documents.

$$Recall = \frac{\left|\text{Relevant documents} \bigcap \text{Retrieved documents}\right|}{\text{Number of relevant documents in the collection}} \tag{21}$$

In both, the numerator is the number of relevant and retrieved documents. To improve the performance of any system, we improve precision and recall everywhere – which essentially requires an increase in the number of relevant retrieves (Rel.Ret.).

7.3 AP

It is the average of the precision values whenever a relevant document is retrieved. R_d stands for the d-th relevant document among a ranked list of retrieved documents. The total number of relevant documents for a given query is 'R'.

$$averageprecision(AP) = \frac{1}{R}\sum_{d=1}^{R} precision(R_d) \tag{22}$$

7.4 MAP

AP corresponds to a single query. However, the comparison between two IR models should not be based on a single query but on a set of queries so that variation in retrieval effectiveness over individual queries is averaged out. Hence, MAP is defined for a set of queries and is computed as the simple arithmetic mean of AP scores of all such queries.

$$meanaverageprecision(MAP) = \frac{1}{|Q|}\sum_{t=1}^{|Q|} AP(t) \tag{23}$$

$|Q|$: Number of queries

8. Test collection

In this work, we experimented with the Marathi, Hindi, and Sanskrit test collections. The Marathi and Hindi test collection downloaded from FIRE[n] evaluation campaign. We built a small test collection in Sanskrit and experimented with it. These corpora primarily comprise news articles extracted from different domains. The news article includes politics, business, sports, science, and other current affairs covering almost all walks of life. Each collection comprises a set of documents and queries. The statistic of different test collections is shown in Table 2. Each query consists of three logical sections, i.e., a brief title (under <TITLE> tag), followed by a description tag (<DESC> tag) and a narrative tag (<NARR> tag). This experiment considers only a query's Title (T) section. In the collections, both topics and documents use the UTF-8 encoding system.

[n] http://fire.irsi.res.in/fire/static/data

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

Table 2. Statistics of text collection

Collection	Size	Number of documents	Number of terms in the lexicon	Number of queries
Marathi	514.9 MB	99276	854027	39
Hindi	1.3 GB	331608	443243	50
Sanskrit	11 MB	7057	376713	50

9. Experimental setup

We implemented different decompounding models to split the Indian language compound words. Moreover, we evaluated the effectiveness of decompounding models in the IR domain by the following steps.

1. Apply different decompounding models in Indian languages described in Section 4.
2. Perform retrieval experiments using different retrieval models and calculate the MAP score and the number of relevant retrieved documents.
3. Compared the MAP scores and a number of relevant retrieved documents with no decompounding approach (none) as the baseline.
4. Compared the MAP score and a number of relevant retrieved of different decompounding approaches (corpus-based, hybrid machine learning-based, and deep learning-based) in different Indian languages IR.

Primarily, we explore the following research questions in the decompounding methods in the Indian language IR.

1. *Does word decompounding impact the Indian language IR? If yes, to what extent?* In this experiment, we evaluate the impact of different decompounding techniques in the Indian language IR.
2. *Can corpus-based, machine learning based, and deep learning-based decompounding models be used in the Indian language IR? If yes, how?* In this experiment, we presented different corpus-based, hybrid machine learning-based and deep learning-based decompounding models and evaluated their effectiveness in Indian languages IR.
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?* In this experiment, we compared the MAP score of different decompounding models and suggested the best-performing model for a language.
4. *Among the different IR models, which provides the best effectiveness from the IR perspective?* In this experiment, we evaluated different IR models and suggested the best-performing model for a language.

10. Experimental results

To address the **research questions (RQs)** described in Section 1, we detail the results and our observations below.

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

Table 3. 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 4. MAP scores of different corpus-based decompounding evaluation in 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

10.1 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 Tables 3, 4, and 5, 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 an 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

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

Table 5. MAP scores of different corpus-based decompounding evaluation in 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

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 Monz and Rijke (2001).

10.2 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 Tables 6, 7 and 8, 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 Monz and Rijke (2001), Ganguly et al. (2013). The fundamental reason behind this is that the machine learning-based decompounding models can efficiently split the simple concatenation and *sandhi*-ied compound words.

10.3 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 learning-based decompounding models for Marathi, Hindi, and Sanskrit languages is shown in Tables 9, 10, and 11. 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

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

Table 6. 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. 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

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

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

Table 8. 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

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.

10.4 Retrieval effectiveness analysis: some insights

In this work, the statistically significant differences are detected by a two-sided t-test (significance level $\alpha = 5\%$) with Bonferroni correction Simes (1986). We use without decompounding as a baseline (Tables 9, 10 and 11). We also observe that the decompounding methods typically improve MAP scores in different retrieval models across the Indian languages, but the changes in effectiveness are not statistically significant. 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 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 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 Figures 7 and 8. In Sanskrit, some queries provide phenomenal retrieval effectiveness. For example, in Topic 47, BJP party's national executive meeting held in New Delhi, the decompounding model improves the average precision score by

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

Table 9. 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

102.93%. A similar observation is found in Topic 34, corruption in P.N.B. bank (improvement of 86.59%).

11. Discussion

In the first set of experiments, we evaluate the effect of different corpus-based decompounding models (shown in Tables 3, 4, and 5) 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 terms of 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. (2013) and Monz and Rijke (2001) reported that the corpus-based decompounding model improved an 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 Koehn and Knight (2003) 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

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

Table 10. 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

cases in non-Indian languages. For example, the compound word 'underworld' is split into 'under' and 'world' due to the high-frequency constituents in the collection. Here, the compound splitting is syntactically correct but changes the semantic meaning of the word. Hence, compound splitting potentially reduces the effectiveness of an IR system. The characteristics of the Indian language compound words differ from those of the European ones, so their 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 6, 7, and 8) 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% Monz and Rijke (2001). 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

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

Table 11. 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

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 Monz and Rijke (2001).

In the third set of experiments, we evaluate the effect of different deep learning-based decompounding models (shown in Tables 9, 10, and 11) in Indian language IR. We find that different deep learning-based word-decompounding models improve the effectiveness of IR systems even further. 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 an MAP score by 6.1%. The deep learning-based models provide better effectiveness in Marathi compared to 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. (2016). b) The number of relevant documents in Sanskrit is less than that of other two languages, as the Sanskrit dataset is the smallest. There are 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% Monz and Rijke (2001). Deep learning-based models outperform the

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

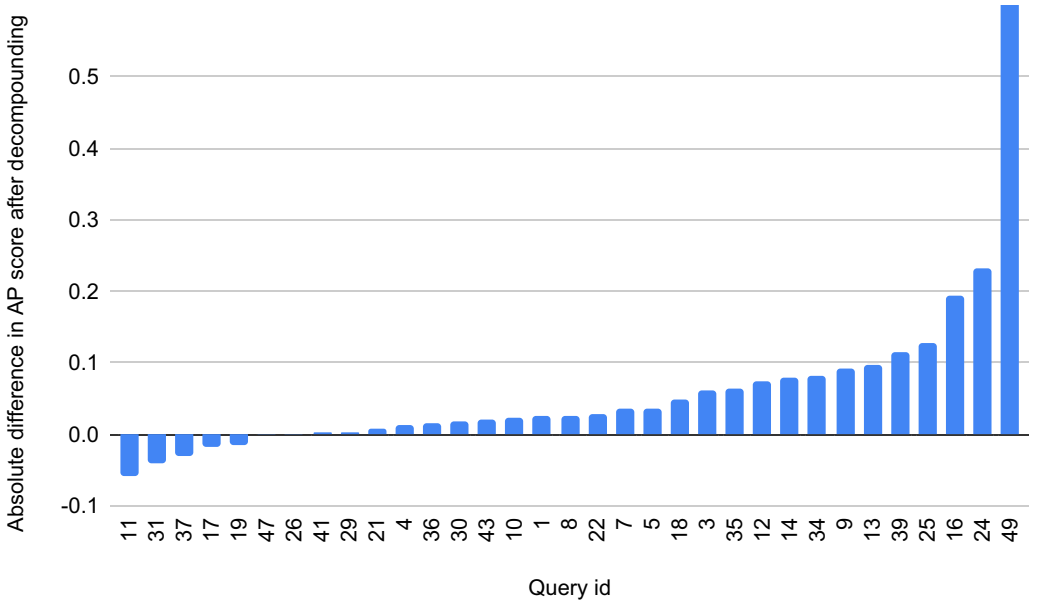

Figure 6. A query by query evaluation in the Marathi by In_expC2 model.

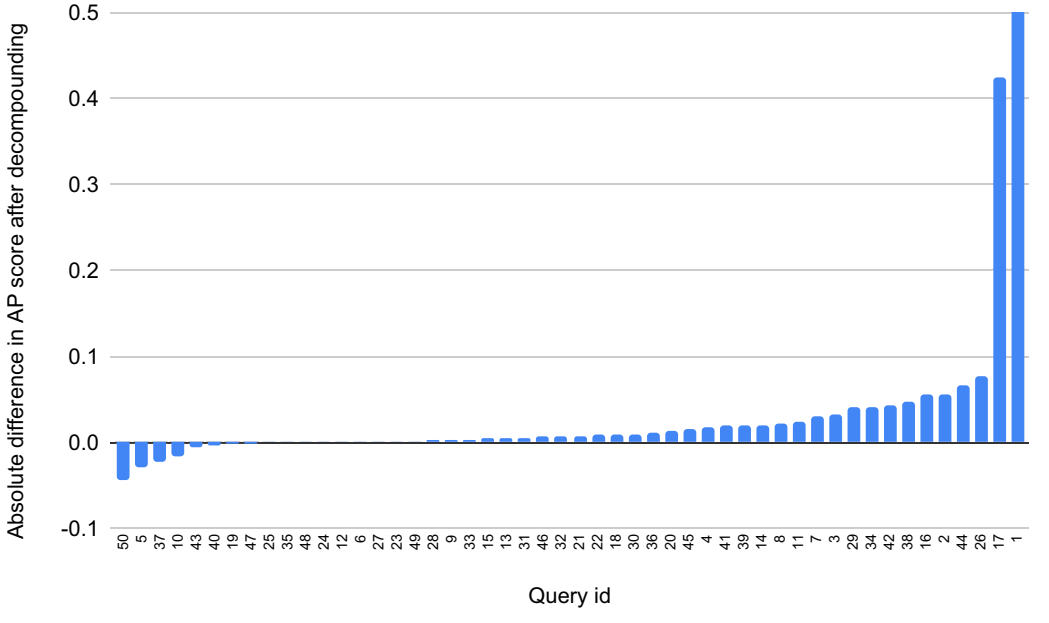

Figure 7. A query by query evaluation in the Hindi by In_expC2 model.

corpus-based and hybrid machine learning-based models in Indian languages. The fundamental reasons behind it 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 moderate

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

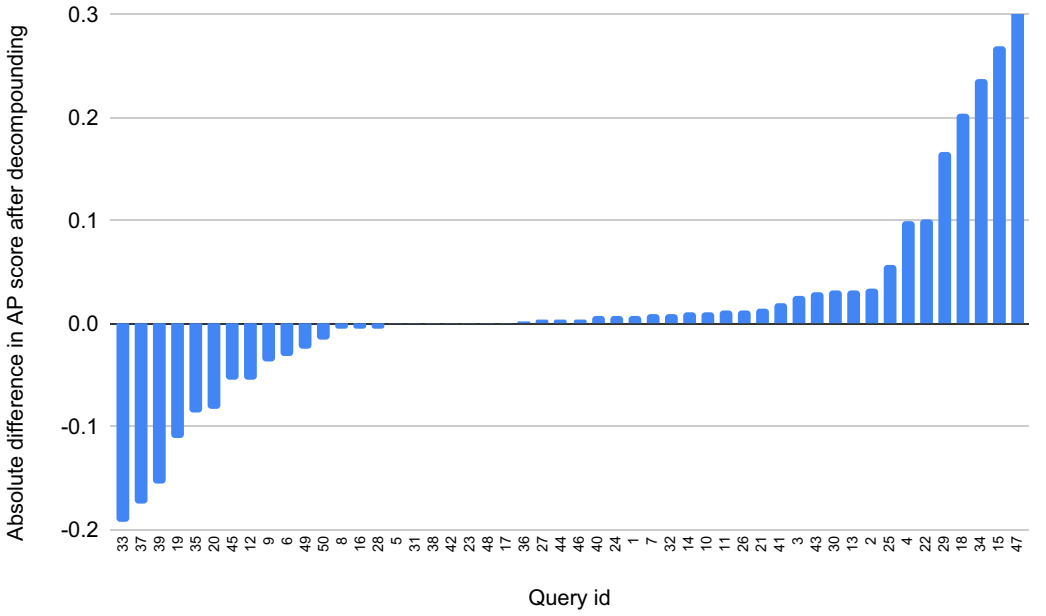

Figure 8. A query by query evaluation in the Sanskrit by BB2 model.

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.

12. Conclusion

Decompounding is an important preprocessing 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 not that effective 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.

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

Acknowledgements. This work was supported by IIT (B.H.U), Varanasi. Moreover, the support and the resources provided by PARAM Shivay Facility under the National Supercomputing Mission, Government of India at the IIT (B.H.U), Varanasi, are gratefully acknowledged.

References

Adda-Decker M. (2003). A corpus-based decompounding algorithm for German lexical modeling in LVCSR. In *Proc. 8th European Conference on Speech Communication and Technology (Eurospeech 2003)*, pp. 257–260.

Airio E. (2006). Word normalization and decompounding in mono-and bilingual IR. *Information Retrieval* **9**(3), 249–271.

Ajees A. and Graham G. (2018). A hybrid approach for suffix separation in malayalam. In *2018 International Conference on Emerging Trends and Innovations In Engineering And Technological Research (ICETIETR)*, IEEE, pp. 1–4.

Alfonseca E., **Bilac S. and Pharies S.** (2008). Decompounding query keywords from compounding languages. In *Proceedings of ACL-08: HLT, Short Papers*, pp. 253–256.

Amati G. and Van Rijsbergen C. J. (2002). Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Transactions On Information Systems (TOIS)* **20**(4), 357–389.

Aralikatte R., **Gantayat N.**, **Panwar N.**, **Sankaran A. and Mani S.** (2018a). Sanskrit sandhi splitting using seq2(seq)ˆ 2. arXiv preprint arXiv: 1801.00428.

Aralikatte R., **Gantayat N.**, **Panwar N.**, **Sankaran A. and Mani S.** (2018b). Sanskrit sandhi splitting using seq2(seq)2. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium: Association for Computational Linguistics, pp. 4909–4914.

Bhardwaj S., **Gantayat N.**, **Chaturvedi N.**, **Garg R. and Agarwal S.** (2018). Sandhikosh: A benchmark corpus for evaluating sanskrit sandhi tools. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Biemann C., **Quasthoff U.**, **Heyer G. and Holz F.** (2008). ASV toolbox: A modular collection of language exploration tools. In *LREC. Citeseer*.

Braschler M. and Ripplinger B. (2004). How effective is stemming and decompounding for german text retrieval? *Information Retrieval* 7(3-4), 291–316.

Chen X., **Qiu X.**, **Zhu C.**, **Liu P. and Huang X.-J.** (2015). Long short-term memory neural networks for Chinese word segmentation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1197–1206.

Cho K., **van Merriënboer B.**, **Gulcehre C.**, **Bahdanau D.**, **Bougares F.**, **Schwenk H. and Bengio Y.** (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734.

Cortes C. and Vapnik V. (1995). Support-vector networks. *Machine Learning* **20**(3), 273–297.

Cover T. and Hart P. (1967). Nearest neighbor pattern classification. *IEEE Transactions On Information Theory* **13**(1), 21–27.

Daiber J., **Quiroz L.**, **Wechsler R. and Frank S.** (2015). Splitting compounds by semantic analogy. In *Proceedings of the 1st Deep Machine Translation Workshop*. Praha, Czechia: ÚFAL MFF UK, pp. 20–28.

Das D , **Radhika K.**, **Rajeev R. and Raj R.** (2012). Hybrid sandhi-splitter for Malayalam using unicode. In *In proceedings of National Seminar on Relevance of Malayalam in Information Technology*, pp. 66–78.

Dave S., **Singh A. K.**, **D. P. A. P. and Lall P. B.** (2021). Neural compound-word (sandhi) generation and splitting in Sanskrit language. In *8th ACM IKDD CODS and 26th COMAD*, pp. 171–177.

Deepa S., **Bali K.**, **Ramakrishnan A. G. and Talukdar P.** (2004). Automatic generation of compound word lexicon for Hindi speech synthesis. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, pp. 140–146.

Devadath V., **Kurisinkel L. J.**, **Sharma D. M. and Varma V.** (2014). A sandhi splitter for Malayalam. In *Proceedings of the 11th International Conference on Natural Language Processing*, pp. 156–161.

Erbs N., **Santos P. B.**, **Zesch T. and Gurevych I.** (2015). Counting what counts: Decompounding for keyphrase extraction. In *Proceedings of the ACL. 2015 Workshop on Novel Computational Approaches to Keyphrase Extraction*, pp. 10–17.

Ganguly D., **Bhat S. and Biswas C.** (2022). if you can't beat them, join them": A word transformation based generalized skip-gram for embedding compound words. In *Proceedings of the 14th Annual Meeting of the Forum for Information Retrieval Evaluation*, pp. 34–42.

Ganguly D., **Leveling J. and Jones G. J.** (2013). A case study in decompounding for Bengali information retrieval. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, Springer, pp. 108–119.

Haruechaiyasak C., **Kongyoung S. and Dailey M.** (2008). A comparative study on thai word segmentation approaches. In *2008 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, IEEE, vol **1**, pp. 125–128.

Hellwig O. (2015). Using recurrent neural networks for joint compound splitting and sandhi resolution in Sanskrit. In *4th Biennial Workshop On Less-Resourced Languages*, pp. 2754–2766.

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

Hellwig O. and Nehrdich S. (2018). Sanskrit word segmentation using character-level recurrent and convolutional neural networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2754–2763.

Hiemstra D. (2001). *Using Language Models for Information Retrieval*. Univ. Twente, pp. 22–26.

Ho T. K. (1995). Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol **1**, IEEE, pp. 278–282.

Hochreiter S. and Schmidhuber J. (1997). Long short-term memory. *Neural Computation* **9**(8), 1735–1780.

Hollink V., Kamps J., Monz C. and De Rijke M. (2004). Monolingual document retrieval for European languages. *Information Retrieval* **7**(1), 33–52.

Kingma D. and Ba J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA.

Kitagawa Y. and Komachi M. (2018). Long short-term memory for Japanese word segmentation. In *Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation*, Hong Kong, Association for Computational Linguistics. https://aclanthology.org/Y18-1033

Koehn P. and Knight K. (2003). Empirical methods for compound splitting. In *10th Conference of the European Chapter of the Association for Computational Linguistics*, Budapest, Hungary, Association for Computational Linguistics.

Lafferty J., McCallum A. and Pereira F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Laureys T., Vandeghinste V. and Duchateau J. (2002). A hybrid approach to compounds in LVCSR. In *Seventh International Conference on Spoken Language Processing*.

Leveling J., Magdy W. and Jones G. J. (2011). An investigation of decompounding for cross-language patent search. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pp. 1169–1170.

Medsker L. R. and Jain L. (2001). Recurrent neural networks. *Design and Applications* **5**(64-67), 2.

Minixhofer B., Pfeiffer J. and Vulić I. (2023). Compoundpiece: Evaluating and improving decompounding performance of language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 343–359.

Monz C. and Rijke M.d (2001). Shallow morphological analysis in monolingual information retrieval for Dutch, German, and Italian. In *Workshop of the Cross-Language Evaluation Forum for European Languages*. Springer, pp. 262–277.

Pellegrini T. and Lamel L. (2009). Automatic word decompounding for ASR in a morphologically rich language: Application to amharic. *IEEE Transactions On Audio, Speech, and Language Processing* **17**(5), 863–873.

Premjith B., Soman K. and Kumar M. A. (2018). A deep learning approach for Malayalam morphological analysis at character level. *Procedia Computer Science* **132**, 47–54.

Quinlan J. R. (1986). Induction of decision trees. *Machine Learning* **1**(1), 81–106.

Reddy V., Krishna A., Sharma V., Gupta P., M R V. and Goyal P. (2018a). Building a word segmenter for Sanskrit overnight. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan: European Language Resources Association (ELRA).

Reddy V., Krishna A., Sharma V. D., Gupta P., Goyal P., et al. (2018b). Building a word segmenter for Sanskrit overnight. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC) 2018*. Miyazaki, Japan, European Language Resources Association (ELRA). https://aclanthology.org/L18-1264

Riedl M. and Biemann C. (2016). Unsupervised compound splitting with distributional semantics rivals supervised methods. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 617–622.

Rigouts Terryn A., Macken L. and Lefever E. (2016). Dutch hypernym detection: Does decompounding help?. In *Joint Second Workshop on Language and Ontology & Terminology and Knowledge Structures (LangOnto2+ TermiKS)*. European Language Resources Association (ELRA), pp. 74–78.

Robertson S. E., van Rijsbergen C. J. and Porter M. F. (1980). Probabilistic models of indexing and searching. In *SIGIR*, vol. **80**, 35–56.

Robertson S. E., Walker S. and Beaulieu M. (2000). Experimentation as a way of life: Okapi at trec. *Information Processing & Management* **36**(1), 95–108.

Sachin K. (2019). Appropriate dependency tagset for Sanskrit analysis and generation. *Orientales Danica Fennica Norvegia Svecia* **80**, 401–425.

Sahu S. S. and Mamgain P. (2019). A corpus-based decompounding in Sanskrit. In *FDIA@ ESSIR*, pp. 110–114.

Shree M. R., Lakshmi S. and Shambhavi B. (2016). A novel approach to sandhi splitting at character level for Kannada language. In *2016 International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, IEEE, pp. 17–20.

Simes R. J. (1986). An improved bonferroni procedure for multiple tests of significance. *Biometrika* **73**(3), 751–754.

Singh D., Bhingardive S. and Bhattacharyya P. (2016). Multiword expressions dataset for Indian languages. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pp. 2331–2335.

Sparck Jones K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation* **28**(1), 11–21.

https://doi.org/10.1017/nlp.2024.16 Published online by Cambridge University Press

Sutskever I., Vinyals O. and Le Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pp. 3104–3112.

Xue N. (2003). Chinese word segmentation as character tagging. *International Journal of Computational Linguistics & Chinese Language Processing, 8(1): February 2003: Special Issue on Word Formation and Chinese Language Processing*, **1**, 29–48.

Appendix

Table A.1. List of prefixes used in different Indian languages IR

apa	abhi	ati
na	aka	mukhya
dura	para	ucha
nira	maha	purba
upa	pari	pradhan
prati	tri	pramukha

Cite this article: Sahu S. S. and Pal S. A case study on decompounding in Indian language IR. *Natural Language Processing* https://doi.org/10.1017/nlp.2024.16