# PREPROCESSING OF NEPALI NEWS CORPUS FOR DOWNSTREAM TASKS

*Sushil Awale, Suraj Prasai, Birodh Rijal & Santa B. Basnet*

*Text collected from online resources introduce a lot of errors which results in incorrect learning outcomes in automatic language learning tasks. In this paper, we discuss a Nepali text preprocessing pipeline to generate clean corpus. This pipeline is tested using a language model to observe impact of each steps in learning task. The relevancy of this work lies in systematizing the procedure in the development of standard Nepali corpus.*

Keywords: Text processing, conjuncts, language models, glyphs, Nepali corpus

## 1. Introduction

Text preprocessing is the first step in a Natural Language Processing (NLP) pipeline and it is a very important step. Text preprocessing is the process of transforming raw textual data into a normalized form suitable for numerical and statistical operations in order to perform various analyses.

In training Nepali NLP systems, text data is often scraped from Nepali news sites, eBooks, pdfs, blogs, and other online sources. The data from these sources often contain words with glyph errors, misspelled words, Non-Devanagari words, typos. Glyph errors are mostly caused by poorly built font conversion tools that are easily available online. A common glyph error is in converting फ from True Type Font (TTF) to Unicode. For example, फेवा is translated as पेmवा. Typos and misspelled words are also quite common in online texts. Publishers make these mistakes due to various reasons such as inability to write the word correctly with available keyboards or lack of Nepali spell knowledge. In NLP models, different spellings for the same word result in different representations of the words in the vector space. As a result, the learning outcomes prove to be inefficient.

In this paper, we introduce a text preprocessing pipeline that addresses the above-mentioned issues. We designed an NLP pipeline that filters Non-Devanagari words using a simple heuristic-based algorithm, repairs glyph errors, and corrects misspelled words. We then run our text corpus through the pipeline and train a language model. Finally, we evaluated the language model for sentence completion tasks. The goal of our evaluation study is two-fold:

i. Evaluate the impact of each individual preprocessing step in the result of the trained model,

ii. Devise the best preprocessing pipeline for cleaning large Nepali corpus.

## 2. Related work

Text content collected from various sources contains different types of errors. Preprocessing the text is the first step in the pipeline of natural language processing (NLP) systems. The basic text preprocessing steps that involves tokenization, stop word removal, text normalization, stemming etc. (Camacho-Collados and Taher, 2018) have used lowercasing, lemmatizing, multiword grouping etc. tasks in text categorization and sentiment analysis which has better performance on preprocessed text. Improved performance is achieved while using text preprocessing and dimension reduction techniques are applied in text clustering systems (Kadhim et al., 2014). Deep learning system often ignore the preprocessing of collected text which has an impact on the performance of the system.

There are chances of additional type of error in the collected text for languages like Nepali. Most of the publications and news houses use traditional TTF fonts to produce the significant amount of text on a daily basis. Internet browsers do not render TTF fonts unless the font resources are provided in the user's machine. Before publishing the content in the web most of the publisher use automatic font conversion program to convert text in TTF format to Unicode. Due to limitation of these automatic converter, the converted text is erroneous. There is

high possibility of broken glyph and character ordering in forming the character conjunct. Basnet et al. (2017) have proposed finite state approach to understand text encoding and utilize the character mapping table, run time cache and the glyph re-arrangement rules during conversion which shows better performance in font conversion.

## 3. Preprocessing pipeline

The architecture of our pipeline, shown in Figure 1, is simple. We merge the text files into a single large file and then feed it into the pipeline.

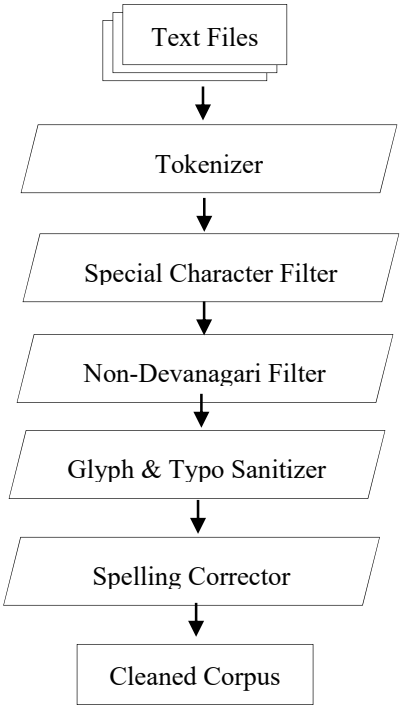


Figure 1: Text preprocessing pipeline architecture

### 3.1 Tokenizer

We tokenize the text based on Devanagari end-of-sentence markers - the *purnabiram* (।), the question mark (?), and the exclamation mark (!). Here, we preserve the end-of-sentence markers.

### 3.2 Special character filter

Next, we filter out special characters that are not used in the Nepali language but can be present in a corpus scraped from online sources. However, we do not remove punctuation and special characters that are used in the Nepali language. Figure 2, shows all the special characters that are filtered out.

←�…¬ = > < @ # $ ^ & * | \ / ` ~ _ { } [ ]

Figure 2: Filtered special characters

### 3.3 Non-Devanagari filter

We filter non-Devanagari tokens using a heuristics-based algorithm. The algorithm, shown in Figure 3, categorizes a token as non-Devanagari if the majority of the characters do not belong to the Devanagari script and discards them.

```
for token in sentence
  if majority of characters ϵ Non-Devanagari
    token is not in Devanagari script
else
  token is in Devanagari script
```

Figure 3: Algorithm to filter non-Devanagari tokens

### 3.4 Glyph and typo sanitizer

We repair the broken glyphs and typos in the text by using a glyph and typo mapping table, shown in Table 1. The broken glyphs and typos are identified using regular expressions and replaced with the corresponding replacement. We created the glyph and typo mapping table by observation of the corpus. Finally, we pass the repaired text into a Unicode Character Tokenizer (UCT) (Singh and Basnet, 2017). The UCT tokenizes a given word into Unicode characters and returns glyph groups, as shown in Figure 4. Glyph groups are groups of characters that are manually categorized. The grouping is based either on the position the characters could appear during conjunct formation, or whether the character could appear alone. We then filter out tokens that do not belong to any

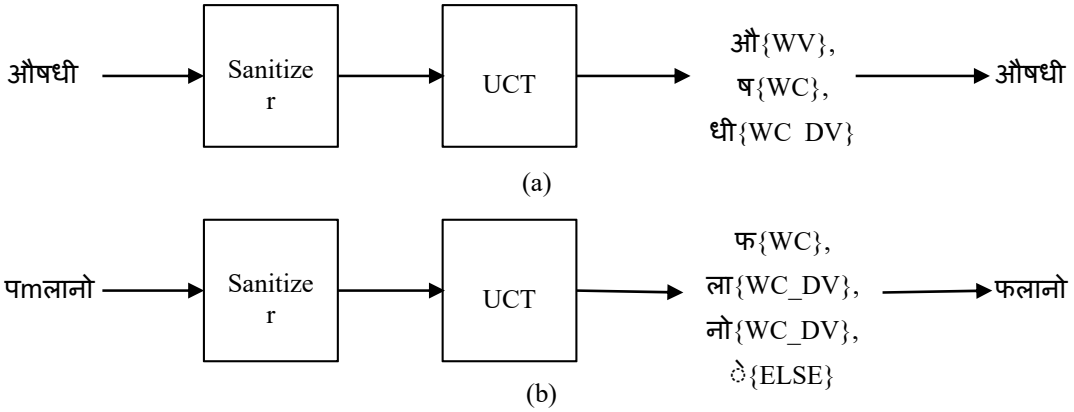


Figure 4: Working of Unicode Character Tokenizer

*In Figure 4(a), the word 'औषधी' has no typo errors or glyph errors and hence no replacement takes place. Also, the output from the UCT assigns each token into valid groups. Hence, no tokens are removed. In Figure 4(b), the word 'पmलानो' is a combination of प + m + ल + ा + न + ा + े + े characters. It consists of both broken glyph (फ is shown as पm) and typo (नो is typed as न + ा + े + े). The errors are identified using regular expressions and replaced by referring to the Glyph and Typo Mapping table. Also, the output from the UCT assigns each token into valid groups except for े which is discarded, and the remaining tokens are concatenated to give the final output 'फलानो'.*

glyph groups and concatenate all the remaining tokens.

Table 1: Glyph and typo mapping table

| Rule | Pattern | Replacement |
|---|---|---|
| R1 | तm / पm / भm | क्त / फ / झ |
| R2 | ¥ / ‰ / Œ / œ / ÷ | र् / झ / त् / त्र / / |
| R3 | (C)« | (C)ि र |
| R4 | (C)+ | C |
| R5 | t | ं(C) /ँ(C) |
| R6 | अ + ा / ौ / ो | आ / औ / ओ |
| R7 | आ + े/ै | ओ / औ |
| R8 | ा + े / ै | ो/ ौ |

Here, C refers to the set of consonants in the Nepali language and c refers to the set of vowel conjuncts in the Nepali language.

### 3.5 Spelling corrector

We correct the spelling in our corpus using a vector distance based spell checking system (Rijal and Basnet, 2017). It is currently the state-of-the-art spell checking system for Nepali. The spell checking system uses a font unification engine, vector distance metric based on the tf-idf weighting of character n-grams and language dependent conjunct replacement list to generate candidate suggestions. These suggestions are then ranked using Levenshtein distance to provide the final list of suggestions. Finally, the top most suggestions from the list is selected as the correct replacement.

## 4. Evaluation

We evaluate 5 different corpora obtained from the pipeline. The simplest way to perform this evaluation is through use case of language

modeling. Language model is built using the cleaned corpus and the model is evaluated on a sentence completion task for our experiment. In the following sections, we discuss the dataset preparation, our experimental setup, and the evaluation results.

### 4.1 Dataset

For our dataset, we use 86,314 news articles collected from various Nepalese news portals. The corpus comprises of news articles in different categories like society, politics, sports, education, technology and other categories. We split the corpus into two sets viz. training and test set, in the ratio of 9:1. We passed the training set through the pipeline and trained the language model using the cleaned corpus while we build the sentence completion task dataset using the cleaned test corpus. We apply different settings in our pipeline to produce five corpora using the same train corpus:

Corpus I: using pipeline up to Special Character Filter,

Corpus II: using pipeline up to non-Devanagari filter,

Corpus III**:** using pipeline up to Glyph and Typo Sanitizer,

Corpus IV**:** using pipeline up to Spelling Corrector skipping Glyph and Typo Sanitizer,

Corpus V: using the complete pipeline.

### 4.2 Language model

In NLP, language modeling is the task of predicting the $n^{th}$ word given a sequence of n-1 previous words. It can be extended to predicting characters, sentences, and phrases. We built a classical n-gram language models for our evaluation using the popular KenLM tool (Kenneth et al., 2013). KenLM uses Kneser-Ney smoothing which is a popular choice for language modeling due to its relatively low perplexity. The KenLM tool also allows for faster querying and low memory consumption by using merge sort and interpolation.

We trained five 5-gram language models using the five corpora. We then evaluated the trained language models by performing extrinsic evaluation on sentence completion tasks.

### 4.3 Sentence completion task

In a sentence completion task, sentences consist of one or more blank spaces that need to be filled. Options for the answer may or may not be given. A suitable machine learning model is used to fill in the blanks, and the model is evaluated on the performance of the task. A study similar to ours was conducted by Geoffrey et al. (2012), where they used different language models on a Scholastic Aptitude Test (SAT) sentence completion task.

For our evaluation, we generate a test set $S$ consisting of 1700 sentences (~1% of test corpus) which are selected from the test corpus using stratified sampling where four strata were created based on sentence length. We randomly remove a word $k_i$ between the first and the last word position in the sentence $S_i$. Next, we build a bigram frequency distribution using the train corpus and collect all the bigrams having the keyword $(k-1)_i$ where $(k-1)_i$ is the word that appears one position before $k_i$ in $S_i$. We use this collection to generate a set of candidates $C_i$ for $k_i$. We also build test bigram set $B$ from the bigram frequency distribution using the test corpus, whose use case is described below.

### 4.4 Evaluation

For evaluation of the 5 corpora obtained from pipeline, we implement a sentence completion task to form complete sentences with each candidate in turn, and evaluate its probability under a language model. The probability of a sentence in a language model is calculated using the following formula

$$P(W^n) = \prod_{j=1}^{n} P\left(w_j \middle| w^{j-1}\right) \qquad (4.4.1)$$

In 4.1, we compute the joint probability by calculating the conditional probability of a word $w_j$ given previous words $w^{j-1}$.

For each sentence $S_i$, we create a set of top five candidates $(C_i^5)$ from $C_i$ based on the probability score given by the language model. We also create a set of top five bigrams $(B_{(k-1)_i}^5)$ from $B$ for $(k-1)_i$. Finally, we evaluate the performance of the language models on the sentence completion

task using Precision (*P*), Recall (*R*), and F-measure (*F*). They are calculated using the following formulas

$$P = \frac{H}{H + I} \quad (4.4.2)$$

$$R = \frac{H}{H + D} \quad (4.4.3)$$

$$F = \frac{2H}{2H + I + D} \quad (4.4.4)$$

Here, a hit, $H$ is the total number of words in $C_i^5$ having the same position as in $B_{(k-1)_i}^5$. Similarly, an insert, $I$ is the total number of words in $C_i^5$ that are also in $B_{(k-1)_i}^5$ but not in the same position. Finally, a delete, $D$ is the total number of words in $C_i^5$ that is not in $B_{(k-1)_i}^5$.

## 5. Results

The results from the evaluation are shown in Table 2.

Table 2: Precision, Recall and F-Measure from model evaluation

| Models | Precision (P) | Recall (R) | F-Measure (F) |
|---|---|---|---|
| Corpus I | 43.94 | 27.92 | 31.15 |
| Corpus II | 43.36 | 27.56 | 33.7 |
| Corpus III | 44.62 | 28.68 | 34.92 |
| Corpus IV | **48.73** | **41.06** | **44.57** |
| Corpus V | **51.22** | **42.57** | **46.49** |

Our result shows the Corpus IV model and Corpus V model significantly outperformed the other models on the evaluation task. The addition of spelling correction in the preprocessing pipeline shows the biggest jump in recall. Here, the same words having different syntactic representations were converged by the spelling corrector explaining the increase in recall. Interestingly, adding the glyph and typo sanitizer in the pipeline did not show huge improvements, highlighting the importance of spelling correction in preprocessing. On the other hand, combining the glyph and typo sanitizer and the spelling corrector in the pipeline, the model achieves the highest score and improves on the score having only the spelling corrector.

## 6. Conclusion

In this paper, we have presented a text preprocessing pipeline to clean online scrapped Nepali text corpus. We have shown how each preprocessing step affects the learning outcome of an n-gram language model on the sentence completion task. The study shows the importance of fixing broken glyphs, correcting typos, and replacing misspelled words in the preprocessing phase. However, our evaluation is not complete as there are many preprocessing steps to consider. But, we hope that our findings will encourage future researchers to carefully select and report these preprocessing decisions when evaluating or comparing different models.

## References

Trishna Singh and Santa B. Basnet. 2017. Unification of fonts encoding system of devanagari writing in Nepali. *Nepalese Linguistics, Vol. 32, 2017, pp. 130-136.*

Birodh Rijal and Santa B. Basnet. (2018, August 3–5). Vector distance based spelling checking system in Nepali with language-dependent heuristics. *Fifth International Conference on IT4D*, Kathmandu, Nepal.

Kenneth H. et al. 2013. Scalable modified Kneser-Ney language model estimation. *In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2, pp. 690–696.*

Geoffrey Z. et al. 2012. Computational approaches to sentence completion. *In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1, pp. 601–610.*

A. I. Kadhim, Y. Cheah and N. H. Ahamed. 2014 Text Document Preprocessing and Dimension

Reduction Techniques for Text Document Clustering. *4th International Conference on Artificial Intelligence with Applications in Engineering and Technology*, 2014, pp. 69-73.

J. Camacho-Collados and P. Mohammad Taher. 2018. On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis *Proceedings of the 2018 (EMNLP) Workshop (B) lackbox(NLP): Analyzing and Interpreting Neural Networks for (NLP)*. pp. 40-46