NEPALI UNICODE CONVERTER WITH PERSONALIZED TEXT PREDICTOR USING N-GRAM MODEL

Nirajan Bist
Department of Computer and Electronics Engineering
Kantipur Engineering College
Lalitpur, Nepal
neonbest12@gmail.com

Abhishek Karki
Department of Computer and Electronics Engineering
Kantipur Engineering College
Lalitpur, Nepal
abhishekkarki17@gmail.com

Saugat Paudel
Department of Computer and Electronics Engineering
Kantipur Engineering College
Lalitpur, Nepal
saugatpaudel6604@gmail.com

Prakash Thapa
Department of Computer and Electronics Engineering
Kantipur Engineering College
Lalitpur, Nepal
prakashthapa617@gmail.com

***Abstract*—Nepali Unicode Converter is the simplest and easiest way to type in Nepali Unicode. It automatically converts Roman Nepali text into Nepali Unicode in any of the desktop applications supporting Unicode fonts. This Nepali Unicode is widely used in any media, machine, or browser to support various languages. This can be used in chatting, emailing, messaging, and many other applications. Furthermore with the growth in technologies and the internet, socializing has become much easier. People around the world spend more and more time on their electronic devices like PCs, laptops, mobiles for social networking, email, banking and a variety of other activities. Due to fast paced nature of such conversation saving as much as time possible while typing is necessary. Hence an application that predicts the next possible words is necessary. Predicting the most probable word for immediate selection is useful technique for enhancing the communication experience. The objective of this work is to design and implement a word predictor algorithm that suggests Nepali words that are being used more in combination with other words of the users, with a lower load for system and significantly reduce the amount of keystrokes required by users. The predictor uses methodology of the N-grams for text prediction. This research uses Maximum Likelihood Estimation method for making prediction table of most probable words after each N-gram. Stupid back-off method is used for prediction if Out of Vocabulary sequences encountered. The training data was scraped from various news portals and mixed into final training data by random sampling. About 80% of the total sentences were used for training and remaining 20% were used for testing the model. Vocabulary Size was 46,000. Accuracy of the model was about 48% for 4-Gram model. Perplexity of the 4-Gram model reaches down to 237.**

***Index Terms*—Roman Nepali, Nepali Unicode, Nepali Text Predictor, N-grams, Stupid-Backoff**

I. Introduction

Unicode is the universally used text encoding system that provides a unique code to every character and symbol regardless of the platform, program and language [1]. On the other hand, ASCII fonts are machine dependent character encoding standard. Since ASCII and Unicode play the identical role for English character encoding system, document and website with English words using either one of them does not make different sense. However, for many languages (such as Nepali) Unicode font is way more beneficial over ASCII.

People cannot type as fast as they think. As a result, they have been forced to go through frustration of slow communication. In the case of text entry people have to press more keys for less number of letters. Predictive typing applications have shown some success. Past approaches to predictive text entry have applied text compression methods (e.g., [2]), taking advantage of the high level of repetition in language [3]. Other techniques for predicting text includes Artificial Neural Networks, Long Short-Term Memory, Support Vector Machine, Machine Learning, etc. These techniques appear to be heavy on the computer for training the model frequently because the requirement is adaptive model with the growth of user data. So for this research we have used N-Gram model which is a statistical probabilistic model which can be trained fast according to the need of this research project.

In Natural Language Processing, the major focus is on understanding and determining how the interaction between human and a computer can be optimized. As humans, we tend to use language based upon the situation we are presented with. Selection of our next word depends upon a set of previous words. This

research aims to replicate a similar behavior of human word selection into a natural language processing model. Suggestions of new words that might be used are generated based upon the previous set of words. To achieve this goal, the N-grams model is used. We assign probability to each word and select the next word with highest probability values. This probability values are generated based upon the sequence of previous words. These sequences are known as N-grams where N is the value which may be a unigram, bigram, trigram and quad gram.

II. Literature Review

Unicode is an information technology standard for the encoding, representation, and handling of text and symbol regardless of the platform, program and language. There are a lot of Nepali Unicode Converters easily available on the internet. However, the systems that converts Nepali Romanized words into Unicode Nepali are not always accurate and does not work for every Nepali words. Also, one of the biggest hindrance is the absence of standardized Nepali Keyboard Layout. The existing keyboard layouts are not scientific and statistically optimized. These layouts do not consider the basic factors like distribution of frequency load among the keys, hand alternation, and many other factors due to which they put excessive and disproportionate stress on the fingers, which on long term can cause several adverse effects. The paper by Prajapati, Shrestha and Jha (2008) [4] analyses how the traditional Nepali Unicode keyboard layout requires a lot of typing effort and reduces typing speed.

Predicting the next word has been an important technique for better communication for more than a decade. One of the first predictive typing assistance was the Reactive Keyboard [5], which made use of text compression methods to suggest completions. This approach used statistical methods. Statistical methods generally suggest words based on word frequency lists to complete the words already spelled out by the user. The benefits of increased accuracy of prediction precision cannot be confined to keystrokes saved by the predictions. An efficient word prediction model can improve the quality of text generated for persons with language impairments, and those with learning disabilities . Word prediction techniques can also be used in order to correct typing errors, separate ambiguous key pad sequences and provide more accurate scanning interface features forecasts. The prediction of letter sequences was analyzed by Shannon (1951) [6]. He discovered that written English has high level of repetitions. Based upon this research an obvious question was whether users can be supported by systems which forecast the next keystrokes, words, or phrases while writing text. A variety of typing tools for apraxia persons and dyslexic persons were developed within natural language by Magnuson and Hunnicutt [7]. These tools provide a possible list of words from which the user could select the required or most approximate word. For these users it is usually more efficient to scan and choose from lists of proposed words than to type. Scanning and selecting skilled authors from several displayed options may, in contrast, slow down (Magnuson and Hunnicutt 2002). A prediction system can make more suitable word choices for the user by exploiting the present sentence context using statistical techniques. Ngrams are Markov models that estimate words from a set of previous words. Ngram probabilities is estimated by counting in a corpus and calculating maximum likelihood estimation. The previous N1 word is used in predicting the current (Nth) term in the Ngrams word prediction methods. In a large corpus, known as the training text, the Ngrams data is collected by counting each single N word sequence. In case of Augmented communications usage Ngrams techniques were limited to unigram and bigram word prediction, but in many other areas related to natural language processing such as speech recognition and machine translation trigram and higher Ngrams orders were often used. [8].

Gregory W. Lesher (2001) found that the impact of using higher-order N-grams and larger database can decrease the number of keystrokes by 7.5 percentage points [9].For a training text size containing 3 million words, he found that keystroke savings increased steadily with higher ngram orders i.e when moving from unigram to bigram word prediction keystroke were decreased by 6.4 percentage points and 7.5 percentage points when moving from unigram to trigram, which suggests increased context-sensitivity in prediction.

III. Problem On hand

Typing in Unicode Nepali text should be quick and easy. It should be effortless and explicit. However, there are only two methods that can be used to type in Unicode Nepali. We can either use keyboard layouts by Madan Puraskar Pustakalaya or use Roman Nepali to Unicode Nepali converters. But none of these methods is good enough. People need to remember one of the keyboard layouts for typing in Nepali Unicode which is difficult for normal users. Using Unicode converter solves the problem of remembering the layouts but currently available converters are not always accurate and does not work for every Nepali words. Moreover, none of those methods is capable of suggesting the next probable words for the user which can make typing quick and easy. Increase in typing speed saves time and enhance the communication experience. So, to solve these issues a better Nepali Unicode Converter that can also suggest next probable words is necessary. In this research, we focus on developing such kind of system that can not only convert Roman Nepali to Nepali Unicode but also suggests the next probable words to the user.

IV. Proposed Method

The operation of the this research is divided into two sections, conversion and prediction. The conversion section deals with conversion of Romanized Nepali text into Nepali Unicode either using the predefined map or user defined map. If the text is not found in any of those models, hard-coded rules are used for conversion. The prediction section deals with suggesting next word from user-defined or pre-defined N-gram model.

A. Conversion

The Roman Nepali to Nepali Unicode conversion takes place either using the user defined model or using predefined model. The development and operation of these models are explained below.

1) User defined Map: For the conversion of words that a user frequently uses with their own notation, a user defined map is created. For example: If a user has a habit of writing "छ" as cha, xa or 6, a separate personal user defined map is created for him/her. This map is used to assist user by increasing the typing speed and productivity.

2) Predefined Map: Almost 40,000 Nepali words are collected from Nepali Brihat Sabdakosh-2075 provided by Nepal Academy. Thus collected words are then converted into English equivalent words using reverse mapping rules. The mapped words are then stored in a JSON file. For Example:

नेपाली - nepali
पानी - pani
साथी - sathi

JSON file format: { "nepali":"नेपाली", "pani":"पानी", "साथी":"sathi",..... }

Also, the reverse dictionary for stop-words like ले, बाट, द्वारा, etc., and loanwords like कम्प्युटर, ईन्जिनियर, क्याल्कुलेटर, etc., is created and stored in a JSON file.

3) Static Rules: If any of the maps stated above does not work for a text, the last option for the Nepali Unicode conversion is hard-coded rules. It uses predefined static rules to convert Roman Nepali into Nepali Unicode. The static rules used are:

a : ' '	au : ौ	i : इ	k : क
A : ा	^: ृ	I : ई	kh : ख
aa : ा	\\: ॄ	ee : ई	g : ग
i : ि	* : ॅ	u : उ	gh : घ
I : ी	** : ँ	U : ऊ	ng : ङ
ee : ी	| : ''	oo : ऊ	ch : च
u : ु	|| : ''	e : ए	x : छ
U : ू	: : ः	ai : ऐ	chh : छ
oo : ू	. : ।	o : ओ	j : ज
e : े	a : अ	au : औ	z : ज
ai : ै	A: आ	c : क	
o : ो	aa : आ	q : क	
jh : झ	Th : थ	v : भ	Sh : ष
yn : ञ	D : द	bh : भ	s : स
t : ट	Dh : ध	m : म	h : ह
th : ठ	n : न	y : य	ksh : क्ष
d : ड	p : प	r : र	jn : ज्ञ
dh : ढ	f : फ	l : ल	gny : ज्ञ
N : ण	ph : फ	w : व	gy : ज्ञ
T : त	b : ब	sh : श	

Note: '|' is zero width joiner and '||' is zero width non joiner.

Letters other than mentioned above (e.g. ., ", ?, / ,etc.) are inserted as it is.

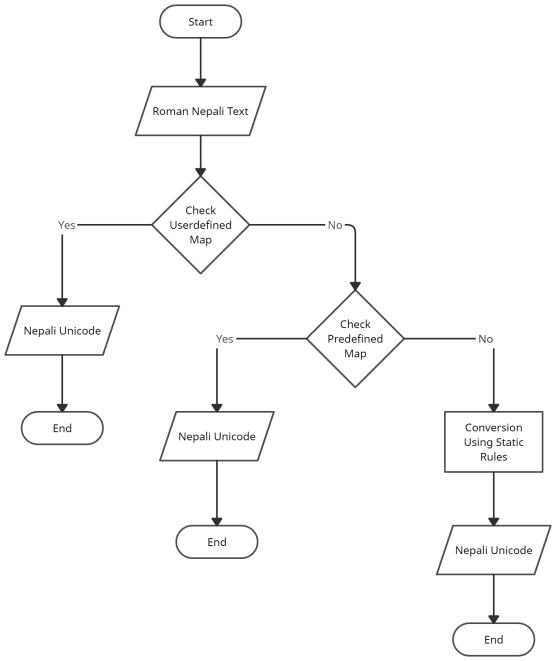

Fig. 1: Flowchart for Roman Nepali to Nepali Unicode conversion

B. Prediction

The prediction section deals with suggesting next word from N-gram model based on previous text inputs by the user and pre-trained map. This section is further divided into two sub-sections. Each of those sections are explained below.

1) Predefined Model: To create a predefined model, corpus was created using the data scrapped from Ghumphir, Kinmel, Opinion, Politics, Social and Sports section of Nepali News portal Setopati.com. Data collected are cleaned before making the model. Here are the steps involved to create predefined model:

I. Data Collection.
II. Data Cleaning and Pre-processing.
   a) Load the data and remove the unwanted characters(noise) from the data.
   b) Tokenize the data i.e. make tokenized sentences.

c) Split the tokenized sentences into train and test sets.
d) Replace words with a low frequency by an unknown marker <unk>.
e) Build a closed vocabulary which consists of unique words from the training data set.

III. N-gram Model Generation using Maximum Likelihood Estimation.
a) Compute the count of n-grams from a training data set.
b) Estimate the conditional probability of a next word with k-smoothing.
c) Use Maximum Likelihood Estimation model to suggest next word.

IV. Evaluate the N-gram models by computing the accuracy and perplexity score.

Data Collection: The data which is a corpus of Nepali Unicode sentences scraped from Ghumphir, Kinmel, Opinion, Politics, Social and Sports section of Nepali News portal Setopati.com using Beautiful Soup.

File Name	File Size	Word Count
Predefine	26.5 MB	100307
Sabdakosh	706 KB	35000
Stopwords	5 KB	350
Loanwords	149 KB	560

Data Cleaning and Pre-processing: The data collected is cleaned before using it further. The data cleaning process include following steps:

I. Removing non-Nepali text
II. Removing numbers
III. Removing punctuation, marking end of sentences
IV. Removing nonfunctional marks
V. Normalizing white space
VI. Removing links

Then the data made into tokenized sentences. Thus obtained data is split into two separate training and testing data sets. 80% of the data was allocated for training the model and remaining 20% data was made for testing the model. From the training data, vocabulary, a list of unique words reaching the threshold count is made. For example, for the threshold is kept 2, the words whose count is less than 2 are not included in the vocabulary. This helps in reducing the number of n-grams.

Handling 'Out of Vocabulary' Words: If the model encounters a word that it never saw during training, it won't have an input word to help it determine the next word to suggest. The model will not be able to predict the next word for the user because there are no counts for the current word. This 'new' word is called an 'unknown word', or Out Of Vocabulary words(OOV). To handle unknown words during prediction, we use a special token '<unk>' to represent all unknown words.

Using the vocabulary, the training and testing data is curated with the <unk> token for each OOV word.

N-gram model Generation: An N-gram is a connected string of N items from a sample of text or speech. The N-gram consists of large blocks of words, or smaller sets of syllables.

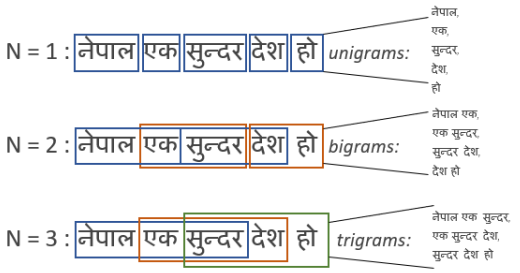

Fig. 2: Example of N-gram Generation

The above example shows how the N-grams are formed. The frequency of each N-gram is kept in their separate frequency table or dictionary. Thus we get frequency table of unigrams, bigrams, trigrams, quadgrams and pentagrams.

The probability of nth word depends on the n-1 words. For a trigram model (n = 3), for example, the probability of each words depends on the two words immediately before it.

Training the n-gram model is done by calculating conditional probabilities from the training data.

The conditional probability for the word at position 't' in the sentence, if the words before it are $w_{t-1}, w_{t-2}...w_{t-n}$ is:

$$P(w_t|w_{t-1}...w_{t-n}) \quad (1)$$

We can estimate this probability by counting the occurrences of these series of words in the training data.

$$\hat{P}(w_t|w_{t-1}...w_{t-n}) = \frac{C(w_{t-1}...w_{t-n}, w_n)}{C(w_{t-1}...w_{t-n})} \quad (2)$$

where:

The function C(...) denotes the number of occurrence of the given sequence and is retrieved from the previously made frequency tables for the given sequence(n-gram).

$\hat{P}$ means the estimation of P.

The following example shows the probability calculation of a word in a trigram model: Given sentence: नेपाल एक सुन्दर देश हो

P(देश / एक सुन्दर) = C(एक सुन्दर देश) / C(एक सुन्दर)

Maximum Likelihood Estimation: MLE is an estimation of the parameters of a probability distribution by maximizing a likelihood function, so that the assumed statistical model of the observed data is most probable.

It estimates the model parameters such that the probability is maximized. In practice, we simply count the occurrence of word

patterns to calculate the maximum likelihood estimation of $P(w_t|w_{t-1}, w_{t-2}...w_{t-n})$. Here, $w_t|w_{t-1}, w_{t-2}...w_{t-n}$ is the sequence of words and 't' is the position of the word in the sentence.

Unigram Model:

$$P(w_t) = \frac{C(w_t)}{\sum_w C(w)} \quad (3)$$

Bigram Model:

$$P(w_t|w_{t-1}) = \frac{C(w_{t-1}, w_t)}{\sum_w C(w_{t-1}, w)} \quad (4)$$

Trigram Model:

$$P(w_t|w_{t-1}, w_{t-2}) = \frac{C(w_{t-2}, w_{t-1}, w_t)}{\sum_w C(w_{t-2}, w_{t-1}, w)} \quad (5)$$

Prediction Table for Suggestions: Using MLE formulas, probability tables are created for each of the unigram, bigram, trigram and quadgram models. They are used to evaluate the models and to find which word is most likely to appear next. The suggestions for each N-gram are sorted in descending order based on their conditional probability.

Stupid Backoff Algorithm in Use: When the N-gram is encountered in the model in course of suggestion, the word with the highest probability for the encountered N-gram is the best next word and hence it is suggested. But if the N-gram is not found in the current model then the lower order N-gram model is used for suggestion. This is called Stupid back-off algorithm. That is, if the given sequence after which a word is to be predicted is not present in a higher-order N-grams, the first word is removed and we back off to a lower-order N-gram.

The figure above is the block diagram showing the steps followed to generate prediction tables. N-Grams model is generated using the train corpus. Using unigrams, bigrams, trigrams and quad grams in N-Gram model, prediction tables are generated for Maximum Likelihood Estimation.

Sample Ngram Model Prediction Tables:
For unigram: 'नमस्ते ' : [('के ', 0.085), ("छ', 0.046),..]
For bigram: ('नमस्ते ', 'के '), : [('छ', 0.051), ('यहाँलाई'), 0.025),..]
For trigram: ('नमस्ते ', 'के ', 'छ') : [('खबर', 0.056), ('यहाँलाई', 0.035),..]
For quadgram: ('नमस्ते ', 'के ', 'छ', 'खबर') : [('हजुरको', 0.021), ('यहाँलाई', 0.015),..]
Atmost 3 probable next words for each gram are included in the prediction table.

2) User defined Model: During the period of using the program, user can manually add the words if not predicted by the predefined model. Everything the user types, it is stored in a user's personalized corpus. This corpus is used to train another personalized N-gram model following the same steps as pre-defined model except the data collection step, as the corpus for training this model is readily available from the previously stored user data. The more user uses the program, the better it knows about the sequence or patterns of the users text and hence gives better predictions in long run.

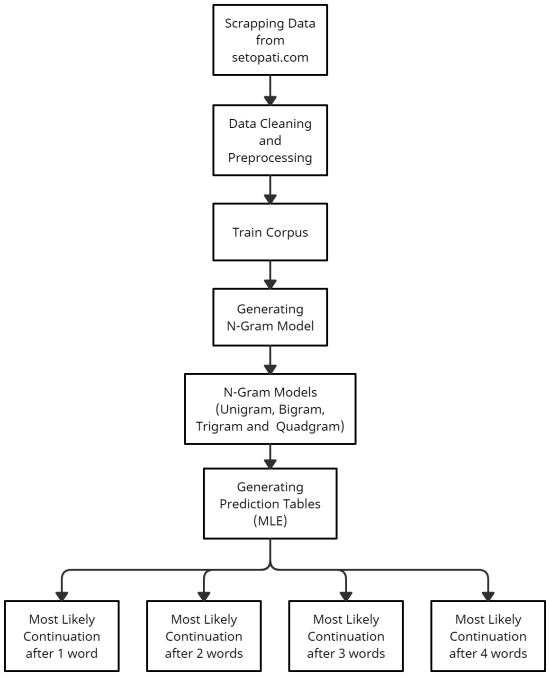

Fig. 3: Generating prediction tables by applying NGrams model

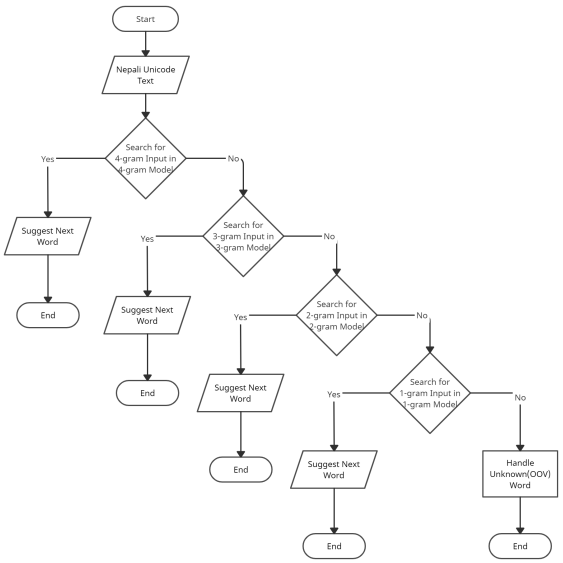

Fig. 4: Generating prediction tables by applying N-Grams model

V. Perplexity

The equation 2 doesn't work when a count of an n-gram is zero. Let's assume that the application encounters an n-gram that did not arise in the training data. Then, the equation 2 cannot be evaluated (it becomes zero divided by zero). To handle zero counts k-smoothing is performed.

K-smoothing adds a positive constant k to each numerator and $k*|V|$ in the denominator, where $|V|$ is the number of words in the vocabulary.

$$\hat{P}(w_t|w_{t-1}...w_{t-n}) = \frac{C(w_{t-1}...w_{t-n}, w_n) + k}{C(w_{t-1}...w_{t-n}) + k|V|} \quad (6)$$

For n-grams that have a zero count, the equation 6 becomes $\frac{1}{|V|}$. This shows that any n-gram with zero count has the probability of $\frac{1}{|V|}$.

In practice we do not use raw probability as our metric for evaluating language models, but a variant called perplexity, a measurement of how well a probability distribution or probability model predicts a sample. If a language model can predict the words that did not occur in the test set, i.e., the P(a sentence from a test set) is highest; then such a language model is more accurate [10]. For a test set $w_{t-1}, w_{t-2}...w_{t-n}$:

$$PP(W) = \sqrt[N]{\prod_{t=n+1}^{N} \frac{1}{P(w_t|w_{t-n}...w_{t-1})}} \quad (7)$$

where; N is the length of the sentence.
n is the number of words in the n-gram (e.g. 2 for a bigram).

The Equation 7 easily creates numerical underflow. The product of small probabilities quickly rounds off to zero. This formula below is equivalent to the one above, and solves the issue.

$$PP(W) = exp(-\frac{\sum_{t=n+1}^{N} log(P(w_t|w_{t-n}...w_{t-1}))}{N}) \quad (8$$

Note that because of the inverse in Equation (3.7), the higher the conditional probability of the word sequence, the lower the perplexity. Thus, minimizing perplexity is means maximizing the test set probability of the language model.

VI. Accuracy

Accuracy is the closeness of the measured value to a standard or true value. Accuracy of N-Gram model is measured using following equation.

$$Accuracy = \frac{NumberOfCorrectPrediction}{TotalNumberOfPrediction} \quad (9)$$

Higher the accuracy, better the model predicts the next probable word.

VII. Result

From 100,000 total words, about 80% (80,000) words were used as training data where remaining 20% (20,000) sentences were used for evaluating the models in terms of accuracy and perplexity.

Grams	Accuracy
1-Gram	0.299
2-Gram	0.384
3-Gram	0.457
4-Gram	0.486

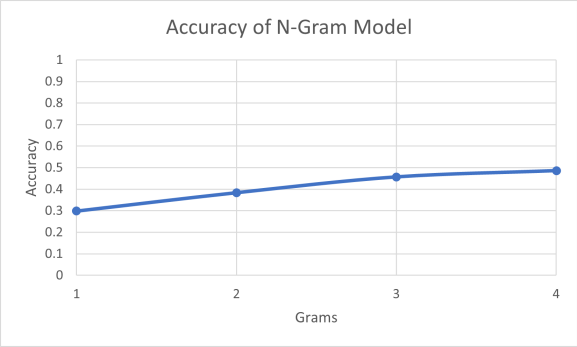

Fig. 5: Plot of accuracy of model

With increase in length of N-gram the accuracy of the model increases. The accuracy of 1-gram, 2-gram, 3-gram and 4-gram model was evaluated to 29.9%, 38.4%, 45.7% and 48.6% respectively. The accuracy increases with increase in length of N-gram as expected because of the increase in available knowledge of previous text which helps in predicting next word which is more probable in the local context.

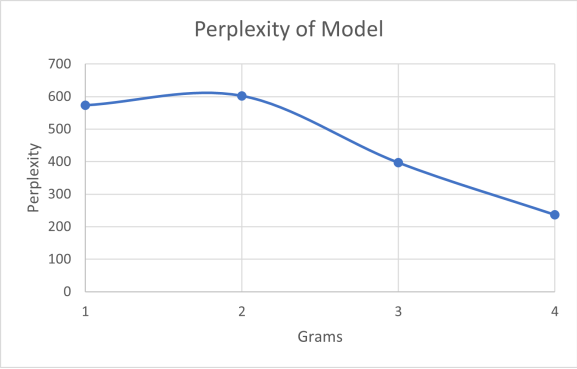

Fig. 6: Plot of perplexity of model

With increase in length of N-gram the perplexity of the model decreases as expected. The perplexity of 1-gram, 2-gram, 3-gram and 4-gram model was evaluated to 573, 602, 397 and 237 respectively.

VIII. Conclusion

A quick, simple and flexible Unicode Converter is very essential and productive. This type of application is very useful in saving human efforts. It is widely usable in any media, machine or browser for chatting,

emailing, messaging, writing documents and many others.

N-Grams being a fairly simple language model turns out to be pretty straightforward and useful for real time prediction and model formation. The accuracy of the model can be seen improving as the value of N-Grams increases. The accuracy of our trained model reaches upto 48%. Similarly perplexity of the trained model decreases with the increase of length of N-gram. Building a 5-grams model; avoiding pruning when while training the models; or more powerful smoothing algorithms than 'stupid back-off' can be applied for increase the accuracy of the model. However, with the increase in size of model also increases the prediction time. Hence the trade-off between accuracy and speed needs to be considered while enhancing the model.

References

[1] M. Reading and A. Wesley, "The unicode consortium. The unicode standard, version 4.0," 2003.
[2] Ian H. Witten, A. Moffat, and T. C. Bell, "Compressing and indexing documents and images," in Managing Gigabytes, 1999.
[3] T Stocky, A. Faaborg, and H. Lieberman, "A commonsense approach to predictive text entry," in MIT Media Laboratory, 2004.
[4] C. Prajapati, J. D. Shrestha, and S. Jha, "Nepali unicode keyboard layout standarization based on genetic algorithm," p. 2, 2008.
[5] I. H. Witten. John J. Darragh, and M. L. James, "The reactive keyboard: A predictive typing aid.," pp. 41–49, 1990.
[6] C. Shannon, "Prediction and entropy of printed english," in In Bell Systems Technical Journal, pp. 50–64, 1951.
[7] T. Magnuson and S. Hunnicutt, "Measuring the effectiveness of word prediction: The advantage of long term use," in Technical Report TMHQPSR, p. 43, 2002.
[8] J. Dumbali and N. Rao, "Real time word prediction using ngrams model," in International Journal of Innovative Technology and Exploring Engineering, vol. 8, 2019.
[9] G. Lesher, B. Moulton, and J. Higginbotham, "Effects of ngram order and training text size on word prediction," 12 2001.
[10] D. Jurafsky and J. H. Martin, "Ngram language models," in Speech and Language Processing, 2019.