Understanding Markets with Data Science - MGT 8803

Jai RaMG (जय राम जी) - Jiffy and intuitive Ramayan and Mahabharat Generator

Kunaal Ahuja - 903296135
Abhishek Patria - 903511102

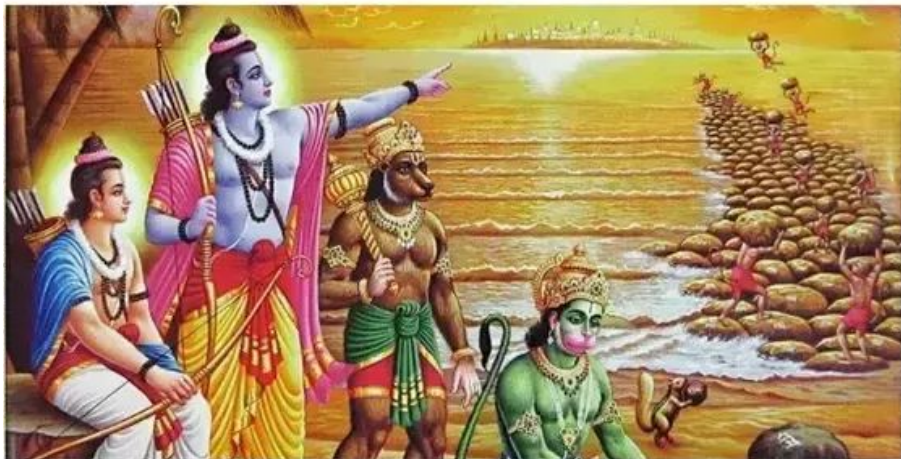


Table of contents

Introduction 3

Motivation 3

Distinguishing features of Devanagari 4

Goals 4

Data Munging 5
- **Data Acquisition** 5
- **Pre-processing** 5
- **Transliteration from Devanagari script** 5
- **Transliteration back to Devanagari Script** 5

Modeling 5
- **N-gram** 5
- **Long Short Term Memory (LSTM)** 6
- **Hyperparameters** 7

Data Source 8

Coding Environment 8

Code Overview and Snippets 8
- **Mahabharat Web scraping** 8
- **Data Collection** 9
- **Data Cleaning** 10
- **One Hot Encoding** 11
- **Preparing Dataset** 11
- **Model** 11

Results 12
- **Mahabharat** 12
- **Ramayana** 13
- **Hindi Literature** 15

Impact and Benefits 16
- **Efficiency** 16
- **Accuracy** 16
- **Volume** 16
- **Transliteration** 16

Applications 17

References 17

Introduction

Natural Language Generation, as defined by Artificial Intelligence: Natural Language Processing Fundamentals, is the “process of producing meaningful phrases and sentences in the form of natural language.” In its essence, it automatically generates narratives that describe, summarize or explain input structured data in a human-like manner at the speed of thousands of pages per second.

Motivation

Content generation systems assist human writers and make the writing process more efficient and effective. These systems comprise a high-level structure of human-authored text used to automatically build a template for a new topic for automatically written text.

Natural language generation enables you to generate complex personalization at scale, creating improved customer communication and experience with your organization.

Upon literature survey, we found that most of prior work in natural language content generation has been done to reproduce the style of Ancient English.

Devanagari (देवनागरी), is a left-to-right alphasyllabary based on the ancient Brāhmī script used in the Indian subcontinent. It was developed in ancient India from the 1st to the 4th century CE and has been in regular use by the 7th century CE. The Devanagari script, composed of 47 primary characters including 14 vowels and 33 consonants, is the fourth most widely adopted writing system in the world, being used for over 120 languages.

India has a population of 1.353 billion out of which approximately 700 million are Hindi speakers. The Natural Language Processing Market size is expected to grow from USD 10.2 billion in 2019 to USD 26.4 billion by 2024, at a Compound Annual Growth Rate (CAGR) of 21.0% during the forecast period. This provides an opportunity to work for a huge untapped market.

Distinguishing features of Devanagari

Devanagari does not have the concept of a letter case. It is written from left to right and has a strong preference for symmetrical rounded shapes within squared outlines. A characteristic feature is that it is recognizable by a horizontal line that runs along the top of full letters.

For example, Professor Chris Gu is written as प्रोफेसर क्रिस गू. Note the horizontal line above each word and the rounded shapes.

Goals

In this project we have tried to build a Natural language model generator using a mix of Natural Language Processing and Deep Learning to automatically generate verses of the two major Sanskrit epics of ancient India, Ramayan and Mahābhārat, which forms the Hindu Itihasa.

The Ramayana is one of the largest ancient epics in world literature. It consists of nearly 24,000 verses. It is divided into six Kands (Adi (Bala) Kand, Ayodhya Kand, Aranya Kand, Kishkindha Kand, Sundara Kand, Lanka Kand) and about 500 sargas (chapters).

The Mahābhārata is the longest epic poem known and has been described as "the longest poem ever written". It consists of over 100,000 śloka or over 200,000 individual verse lines (each sloka is a couplet), and long prose passages. At about 1.8 million words in total, the Mahābhārata is roughly ten times the length of the Iliad and the Odyssey combined, or about four times the length of the Rāmāyaṇa.

In this project we have trained our model on the Devanagari version of the Ramayana and Mahabarat and on the other works of hindi literature by famous poets and writers like Kabir, Meera, Rahim, Surdas, Tulsidas, etc.

Data Munging

Data Acquisition

The Ramayana, Mahabharata and other ancient Indian texts were collected through Web Scraping. It took almost one day to download all of the parts into flat files that could be utilised for input to the model.

Pre-processing

The collected files were then processed for cleaning of unnecessary whitespaces, tabs, etc. Thereafter, the data was split into a set of characters for the model and a vocabulary was created for the corpus dataset. A one-hot vector representation for each character of the vocabulary was created.

Transliteration from Devanagari script

Hindi literature is written in the Devanagari script, which has its own set of characters and a set of rules for combining them, hence the text was translated into English corpus.

Transliteration back to Devanagari Script

Once the model was trained and the corresponding language verses were generated, it was translated back to Devanagari script.

Modeling

We used a hybrid model consisting of a custom character level N-gram natural language model and a Long Short Term Memory (LSTM) network to train our Jai RaMG language generator.

N-gram

An n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words or base pairs.

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)\dots P(w_n|w_1^{n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

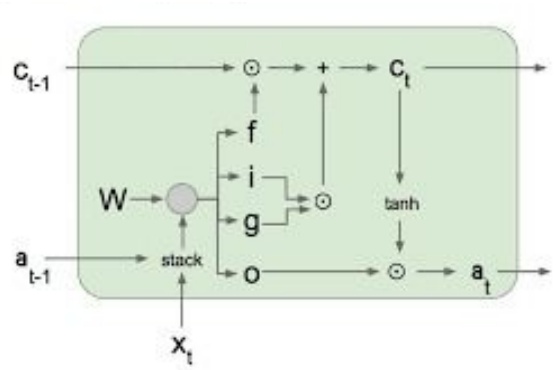
Where $P(w_i)$ is the probability of the i^{th} word in a given sentence

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} w_n)}{C(w_{n-N+1}^{n-1})}$$

The n-grams for our model were generated using the Mahabharat, Ramayana and other Indian text corpus. The collection of the N-grams was then used to learn the LSTM network.

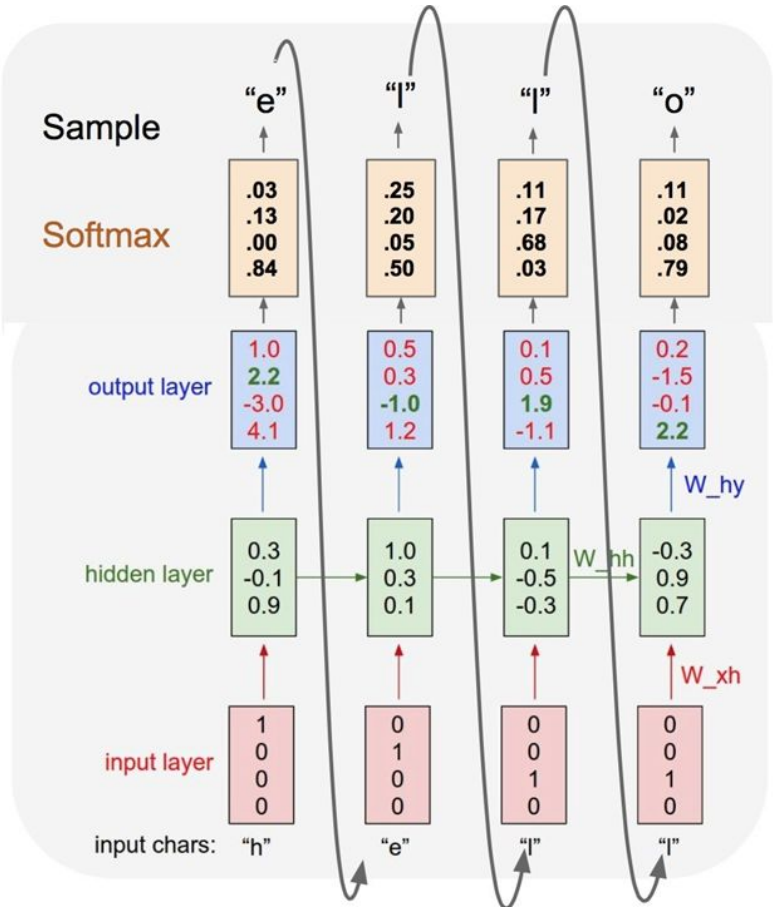
Long Short Term Memory (LSTM)

LSTM is a type of Recurrent Neural Network (RNN) that has feedback connections consisting of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

For a random sample in our corpus, an LSTM consists of an input layer that would receive one hot encoding of each character. This output would then be processed through hidden layers. For each corresponding letter, we would have an output that would be passed through a Softmax node to get the best possible probability. An illustration is as follows: -



Hyperparameters

Following hyperparameters were used to train our model

TRANSLITERATE	TRUE
INPUT_LENGTH	40
OUTPUT_LENGTH	1
DATASET_SIZE	100000
NUM_EPOCHS	100
BATCH_SIZE	512
HIDDEN_SIZE	350
GEN_LENGTH	3000

Data Source

We got the electronic version of the Ramayan from https://github.com/svenkatreddy/Ramayana_Book
and Mahabharat from https://www.sacred-texts.com/hin/mbs/ .
For the other literature, we used the dataset provided by the cltk https://github.com/cltk/hindi_text_ltrc

Coding Environment

We used Google Colab to build our model in Python. Google Colab is an online code editor where users can collaborate on a project. It also provides GPU hardware to run complex models like deep learning models. This helped us with the necessary resources for this task and enabled us to collaborate in an efficient manner.

Code Overview and Snippets

Mahabharat Web scraping

The dataset for Mahabharat was scraped from the web using python libraries. Following is the code snippet to do the same.

```
for book in range(1,19):
  book_content = []
  book_num = "%02d" % book
  for chapter in range(1, 500):
    chapter_num = "%03d" % chapter
    url = "https://www.sacred-texts.com/hin/mbs/mbs" + book_num+chapter_num + ".htm"
    content = get_chapter(url)
    if type(content) == list:
      book_content.extend(get_chapter(url))
  filename = 'book' + book_num + '.txt'
  with open(filename, 'w') as f:
    for item in book_content:
        f.write("%s\n" % item)
```

Data Collection

We collected data from various web sources. In the case of Mahabharat, we scraped the textual data from the web. For this project, we loaded the data on the Google Colab environment.

```
def get_mahabharat_text():
  text = ""
  nums = list(range(1, 18))
  for num in nums:
    file_name = "book{:02d}.txt".format(11)
    text_file = open(file_name, "r")
    lines = text_file.readlines()
    data = " ".join(lines)
    text = text + data

  return text
```

```
def get_ramayan_data():
  kands = ['balakanda', 'ayodhyakanda', 'aranyakanda', 'kishkindhakanda', 'sundarakanda', 'uttarakanda']
  data_list = []
  for kand in kands:
    file_name = kand + '_all.json'
    with open(file_name) as f:
      data = json.load(f)
      data_list.append(data[kand]['chapters'])
      # print(data)
  return data_list
```

```
def download_hindi_literature():
  corpus_path = "corpus"
  corpus_url = "https://github.com/cltk/hindi_text_ltrc/archive/master.zip"
  corpus_zip_path = "master.zip"
  urllib.request.urlretrieve(corpus_url, corpus_zip_path)

  # Unzip the whole git-repository to the corpus-path.
  print("\n\nUnzipping corpus...\n\n")
  zip_file = zipfile.ZipFile(corpus_zip_path, 'r')
  zip_file.extractall(corpus_path)
  zip_file.close()
```

Data Cleaning

```
def clean_text(text, alpha = True):
    if alpha:
      letters = string.ascii_letters
      for letter in letters:
        text = text.replace(letter, " ")

    text = text.replace("\t", " ")
    text = text.replace("\n", " ")
    text = text.replace("|", " ")
    text = text.replace("0", " ")
    text = text.replace("1", " ")
    text = text.replace("2", " ")
    text = text.replace("3", " ")
    text = text.replace("4", " ")
    text = text.replace("5", " ")
    text = text.replace("6", " ")
    text = text.replace("7", " ")
    text = text.replace("8", " ")
    text = text.replace("9", " ")
    text = " ".join(text.split())

    return text
```

One Hot Encoding

```
def encode_string(text, vocab_set):
    encoded_string = []
    for character in text:
        encoded_character = np.zeros((len(vocab_set),))
        one_hot_index = vocab_set.index(character)
        encoded_character[one_hot_index] = 1
        encoded_string.append(encoded_character)
    return np.array(encoded_string)
```

Preparing Dataset

```
def create_data(text, vocab_set):
    data_input = []
    data_output = []
    current_size = 0
    bar = progressbar.ProgressBar(max_value=DATASET_SIZE)
    print("\n\nGenerating data set...\n\n")
    while current_size < DATASET_SIZE:
        random_string = random_substring_of_length(text, INPUT_LENGTH + OUTPUT_LENGTH)
        random_string_encoded = encode_string(random_string, vocab_set)

        input_sequence = random_string_encoded[:INPUT_LENGTH]
        output_sequence = random_string_encoded[INPUT_LENGTH:]

        data_input.append(input_sequence)
        data_output.append(output_sequence)

        current_size += 1
        bar.update(current_size)
    bar.finish()

    train_input = np.array(data_input)
    train_output = np.array(data_output)
    return train_input, train_output
```

Model

```
def create_model(vocab_set):
    input_shape = (INPUT_LENGTH, len(vocab_set))

    model = models.Sequential()
    model.add(layers.LSTM(HIDDEN_SIZE, input_shape=input_shape, activation="relu"))
    model.add(layers.Dense(OUTPUT_LENGTH * len(vocab_set), activation="relu"))
    model.add(layers.Reshape((OUTPUT_LENGTH, len(vocab_set))))
    model.add(layers.TimeDistributed(layers.Dense(len(vocab_set), activation="softmax")))
    model.summary()

    model.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])

    return model
```

Results

The results for each of the corpus are as follows: -

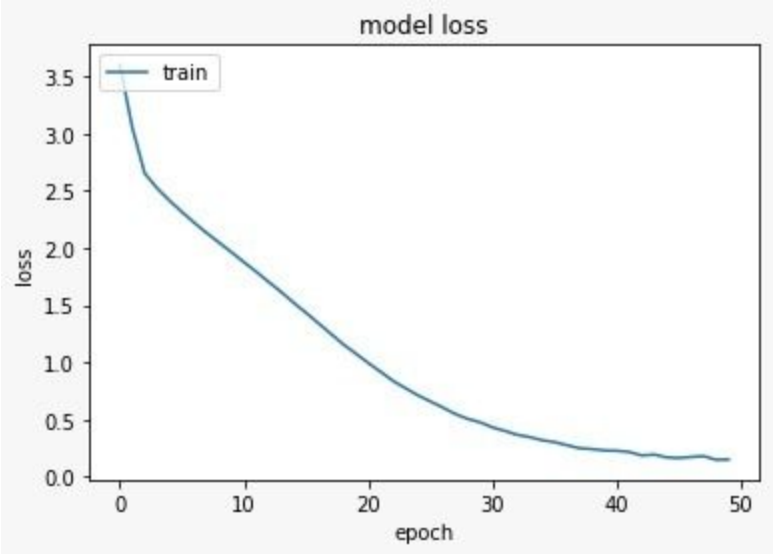
Mahabharat

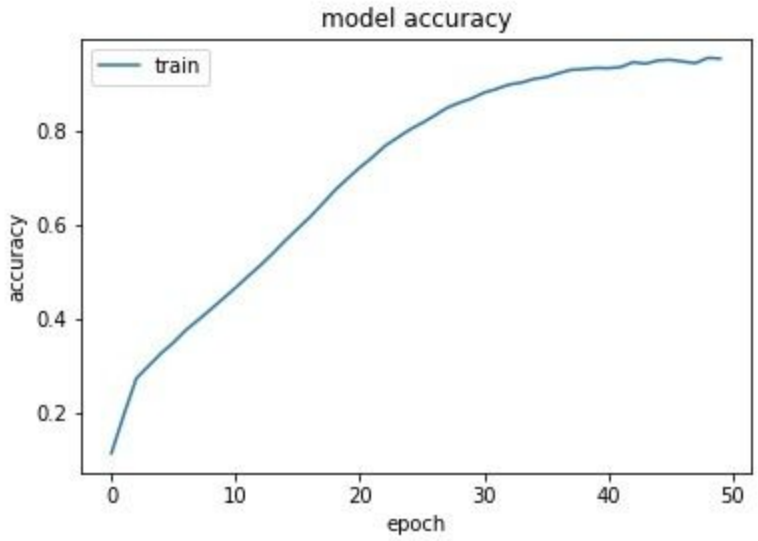
Input: यः अध्रुवे जीवलॊके च सथाने वाशाश्वते

Output: सति जीमतुजशः सर्वाः सवयापि दुःखास्त्रुशॊ sथि वीराणो कथम अधीयन्त भूरिन्यमे [व] तरास्त्रैष कृत्वा बाहा करिणाश च विमौः परिप्तेन वने सत्विताः शयानं शरुत्वा कषीटन दुःशासनिशं हृष्ठे भराताः पाप्य अपि पृष्ण सिनुं नून्यॊक्तं यथा गूर्णस्य परिश्यति संसार जञात्वं धर्मशस्यॊर इति शूरान रणे सथायुं न हि शक्यमाभिः पश्त कृष्ण निर्थयत कर्णाद ऋजस्येन नाप्य एवं सिं वीरं पात्यशस्वृन्दन्त्यः पुर्षसं यसि समेत्षे कृष्ण शयानं सूतेषु सुजमॊ ये सवितः शस्त्रैः पततः शरुत्वा कर्ण पर्यकृत्डाभॊ मानस तु धर्मम जीवसि समात्यया एव अनवे स ते विह्रः शरुसं हि परद्जते सत्येष्ठ रॊणि पृश्यशे

Characteristics:

- Ability to reproduce coherent formatting rules like whitespaces
- Maintain the semantic and lexical structure of the original text. For example some form of words in Sanskrit are supposed to end with a विसर्ग symbol ‘:’





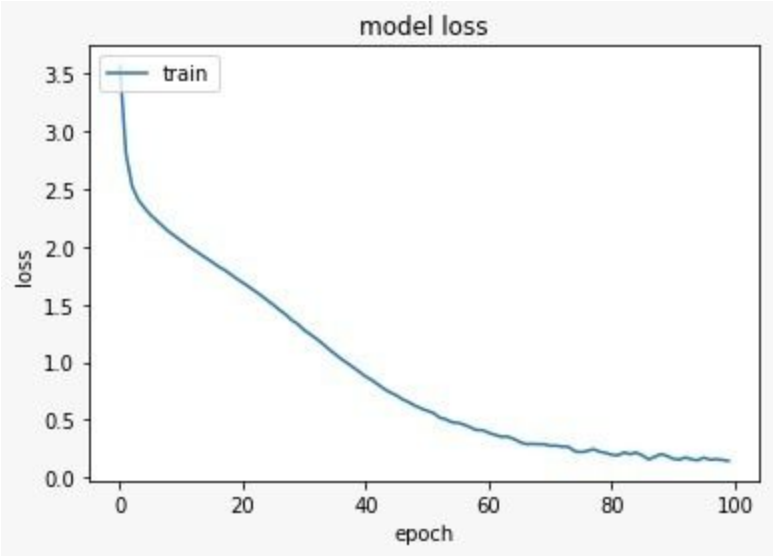
Ramayana

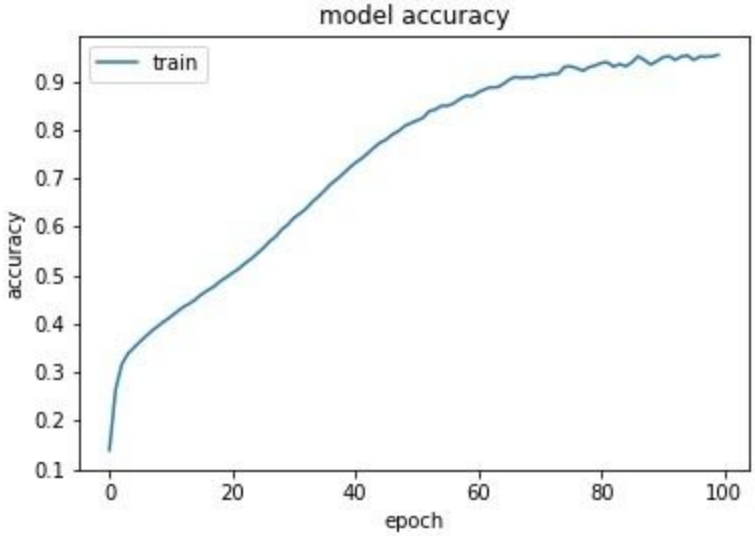
Input: सर्व शास्त्र अर्थ तत्तमम् ततः प्रास्थाया

Output: नीते जगिरेः पूर्वसम् महोर्मालम् राम निवर्पतः ॥१-१३-३२॥एवम् भवेन म्रघके मया शाक्षापे सुबदौ वापाणाम् भास्वरस्प समृत्रिवये नार वा सीताम् शशी भाषण ईव्डुराभाश्च वीराणि च पूनयन् पुनः प्रसहसा सौम्य दशकथः तवोद्धुः ॥२-७१-१॥इमाज्ञपि सानना पुष्पोश्वानराज ह् शि समाजवावणो रामा राजीञ्षितैः उव च लक्ष्मण बालकस्य इदा भगूनि दृशं तरम् ॥२-५०-२५॥

Characteristics:

- Preservation of end of verse symbol '॥'
- Appropriate numbering and tagging of the verse as per the original text
- Ability to reproduce coherent formatting rules like whitespaces
- Maintain the semantic and lexical structure of the original text


model loss
train
loss
epoch
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80
100


model accuracy
train
accuracy
epoch
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
20
40
60
80
100

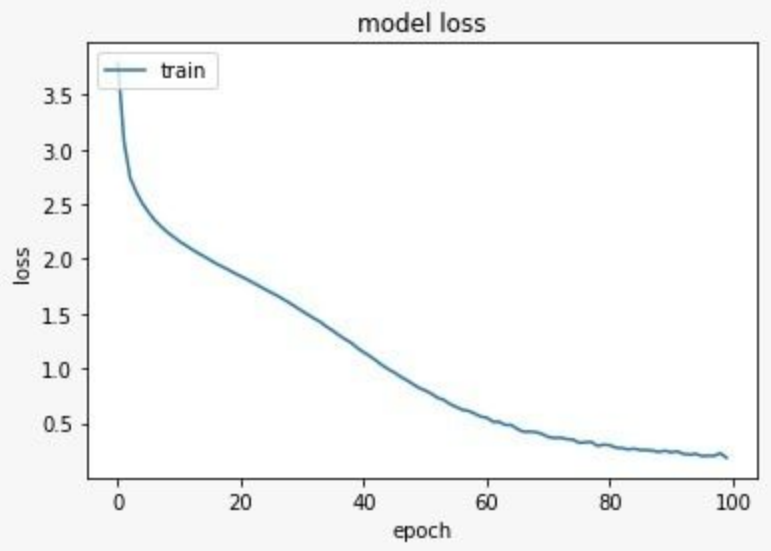
Hindi Literature

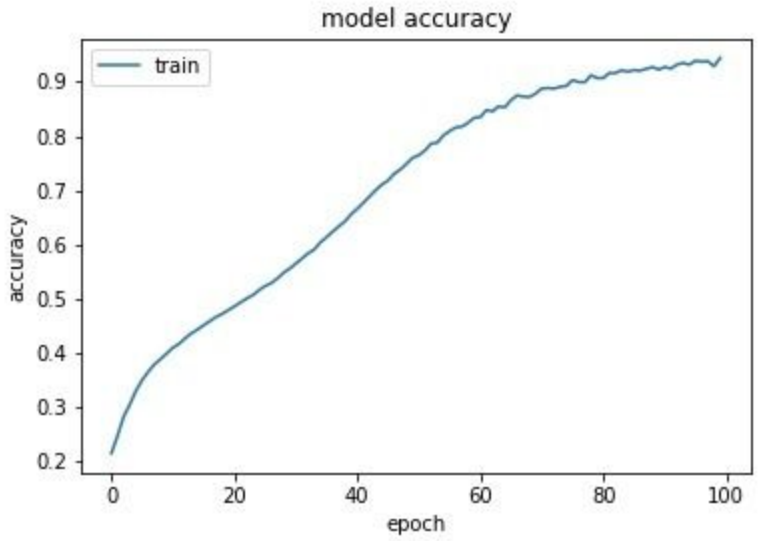
Input: एक भर पटकन भी धीरारी

Output: मेंग्ठल में देखो . यहीं तो आ हरार् सी जाते थे और रामभीमःमय का ऐसा समझते के पेदूंचक र् इन्दें अब हसको मानता है जो किसी न ठाक लिया s यह बुछा प्रशंसा अपने उस पर उम्बास को यह पलीन लगता था s अब वह मिटे दोनियाने का हृदय और प्रेम करने के समय बडहरी हो s मैं एक तब मगनों कोई एक रहक या ना आर्याॐ सम्भ के ऐसे से न गाने s यह हम अच्छा तो कहाँ विद्यारमान आपकी जिस कपस्था कर लिय ॐर्मभावता की शीरतॐ उद्धा अनंत . वह अब अप अकारथ से कुछ कितन में रोती . आज चली जाहू ऋ जग खलल महन मदां मही s

Characteristics:

- Preservation of end of sentence symbol ‘s’
- Ability to reproduce coherent formatting rules like whitespaces
- Maintain the semantic and lexical structure of the original text





Impact and Benefits

Efficiency

This model was written in a way that it can be really efficient in real-life applications. The training time was approximately 30 minutes per corpus which spanned across thousands of pages.

Accuracy

The model was not just efficient, we laid special emphasis on the accuracy of the model as well. We could achieve around 95% accuracy for 50-100 epochs or number of iterations. This can be further improved by running the model for more number of epochs and by running on better hardware.

Volume

The model was built in a way that it could handle the needs of big data systems. More than 1000 pages of text could be processed in less than 30 minutes making it a lucrative option to be used in the literature and translation industry.

Transliteration

We built a generic transliteration model which can be used to transliterate from any major language to another. In this project, we used it to convert text from Devanagari to English and vice-versa.

Applications

There are multiple applications of this project. This can be extended to the education sector where it can be used as a teaching assistant. It can also be used to translate documents written in local languages to popular languages so that the documents and texts are accessible to a larger audience.

It can also be used as a Question-Answering bot where given a starting point, the model can generate relative texts for the question and provide relevant answers.

Another application of this project would be in the Service industry where we can train this model on the customer service data to provide support to users.

References

1. Dan Jurafsky, Speech and Language Processing (3rd ed.)
2. Wang, S. and Manning, C. D. (2012). Baselines and bigrams: Simple, good sentiment and topic classification. In ACL 2012, 90–94
3. N-gram: a language independent approach to IR and NLP P Majumder et. el
4. https://www.worldatlas.com/articles/the-most-widely-spoken-languages-in-india.html
5. https://www.marketsandmarkets.com/Market-Reports/natural-language-processing-nlp-825.html