**Chaii - Hindi and Tamil Question Answering**

**Identify the answer to questions found in Indian language passages**

### India is the second most populated country in the world. In the web world, Indian languages are dominated by English. NLU (Natural Language Understanding) doing well with English but not well enough for Indian language speakers. Which has limited the web users from Indian communities. Kaggle has launched a competition to enhance the user experience in the field of NLP and NLU.

### The Kaggle data set – chaii has been chosen for this project which covers Hindi and Tamil languages without the use of translation. We will be using question-answer data pairs to answers the questions asked by the user.

### A successful model can improve the baseline performance of NLU models in Indian languages and could help to improve the web experience for the Indian community.

## **1. Data**

Google research India contributes to the unique challenges in the Indian context (such as code-mixing in Search, diversity of languages, dialects, and accents in Assistant), learning from limited resources, and advancing multilingual models. This Kaggle NLU project is a sufficient size to develop a best Question-Answering Model. Click below to know more about data and related challenges

* [Google research India](https://research.google/teams/india-research-lab/)
* [Kaggle Dataset](https://www.kaggle.com/c/chaii-hindi-and-tamil-question-answering)
* [Chaii (challenge in Al for India)](https://events.withgoogle.com/chaii2021/)

## **2. Method**

There are so many methods for the question answering systems listed below

(<https://huggingface.co/transformers/model_doc>). Few of the models are well known and widely used, few models are language-specific, and few are multilingual. These models are different in size and cover from base to large model.

* ALBERT For Question Answering
* BART Question Answering:
* BERT Question Answering:
* DEBERTA Question Answering:
* DEBERTA-V2 For Question Answering
* DistilBERT For Question Answering
* ELECTRA For Question Answering
* ROBERTA
* TFXLM For Question Answering Simple
* XLM Roberta For Question Answering
* FLAUBERT For Question Answering Simple
* IBERT For Question Answering
* LED For Question Answering
* FUNNEL For Question Answering
* LONGFORMER For Question Answering
* LXMERT For Question Answering
* MBART For Question Answering
* MEGATRONBERT For Question Answering
* MOBILEBERT For Question Answering
* MPNET For Question Answering
* REFORMER For Question Answering
* REMBERT For Question Answering
* ROFORMER For Question Answering
* SQUEEZEBERT For Question Answering
* TAPAS For Question Answering
* XLNET For Question Answering
* **ALBERT**

The ALBERT model was proposed in ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. It presents two parameter-reduction techniques to lower memory consumption and increase the training speed of BERT: Splitting the embedding matrix into two smaller matrices. Using repeating layers split among groups.

* **BART**

Bart uses a standard seq2seq/machine translation architecture with a bidirectional encoder (like BERT) and a left-to-right decoder (like GPT). The pretraining task involves randomly shuffling the order of the original sentences and a novel in-filling scheme, where spans of text are replaced with a single mask token.

BART is particularly effective when fine tuned for text generation but also works well for comprehension tasks. It matches the performance of RoBERTa with comparable training resources on GLUE and SQuAD, achieves new state-of-the-art results on a range of abstractive dialogue, question answering, and summarization tasks.

* **BERT**

The BERT model was proposed in BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. It’s a bidirectional transformer pretrained using a combination of masked language modeling objective and next sentence prediction on a large corpus comprising the Toronto Book Corpus and Wikipedia.

We introduce a new language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models, BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly BERT is conceptually simple and empirically powerful.

# DeBERTa

The DeBERTa model was proposed in DeBERTa: Decoding-enhanced BERT with Disentangled Attention. It is based on Google’s BERT model released in 2018 and Facebook’s RoBERTa model released in 2019. It builds on RoBERTa with disentangled attention and enhanced mask decoder training with half of the data used in RoBERTa.

# DeBERTa-v2

The DeBERTa model was proposed in DeBERTa: Decoding-enhanced BERT with Disentangled Attention. It is based on Google’s BERT model released in 2018 and Facebook’s RoBERTa model released in 2019. It builds on RoBERTa with disentangled attention and enhanced mask decoder training with half of the data used in RoBERTa.

* **DistilBERT**

The DistilBERT model was proposed in the blog post Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT, and the paper DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. DistilBERT is a small, fast, cheap and light Transformer model trained by distilling BERT base. It has 40% less parameters than bert-base-uncased, runs 60% faster while preserving over 95% of BERT’s performances as measured on the GLUE language understanding benchmark.

# ELECTRA

The ELECTRA model was proposed in the paper ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. ELECTRA is a new pretraining approach which trains two transformer models: the generator and the discriminator. The generator’s role is to replace tokens in a sequence and is therefore trained as a masked language model. The discriminator, which is the model we’re interested in, tries to identify which tokens were replaced by the generator in the sequence.

# RoBERTa

## Overview

The RoBERTa model was proposed in RoBERTa: A Robustly Optimized BERT Pretraining Approach. It is based on Google’s BERT model released in 2018. It builds on BERT and modifies key hyperparameters, removing the next-sentence pretraining objective and training with much larger mini-batches and learning rates.

# XLM

The XLM model was proposed in Cross-lingual Language Model Pretraining. It’s a transformer pretrained using one of the following objectives:

a causal language modeling (CLM) objective (next token prediction)

a masked language modeling (MLM) objective (BERT-like), or

a Translation Language Modeling (TLM) object (extension of BERT’s MLM to multiple language inputs)

# XLM-RoBERTa

The XLM-RoBERTa model was proposed in Unsupervised Cross-lingual Representation Learning at Scale. It is based on Facebook’s RoBERTa model released in 2019. It is a large **multi-lingual language model**, trained on 2.5TB of filtered Common Crawl data.

For our current project, we will use **DistilBERT** because it is a small, fast, cheap, and light Transformer model trained by distilling BERT base.

**3. Data Cleaning & preprocessing**

Data Cleaning & preprocessing Report

The Chaii- a question answering dataset has a set of contexts, questions, answer text, and position of start index of answer started. The training dataset contains '1114' rows and '6' columns. The test dataset contains '5' rows and '4' columns. There are no missing or non-null values are present. Dataset consists of context and questions in two languages like Hindi and Tamil.

**Observation 1:** There are few things to notice to take a general idea about the data, for example, length and word counts (min, max, and mean) of contexts and questions.

**Solution1:** In our data set we have

max text length of context- 49815

min text length of context- 176

mean text length of context- 10999.16

max text length of the question- 121

min text length of the question- 19

mean text length of the question- 41.65

max context word count of context- 10259

min context word count of context- 24

mean context word count of context- 1694.25

max question word count question - 22

min question word count question - 3

mean question word count question - 7.1

**Observation 2:** The data set contains more than one language Hindi and Tamil**. We need to deal it separately.**

**Solution:** Tokenized all the contexts and questions to char\_to\_Token.

**Observation 3:** This data set contains a start index of the answer text. To feed it into the model, we need to find the exact end position of the answer text.

**Solution 3:** We already have the answer text column and first index of the answer text. We have calculated the end index by taking the length of the answer text and adding it to the start index that generated the end index of the answer text. Later, it has converted to the token for further processing.

## **4. EDA**

EDA Report

* The language frequency in the chaii dataset is presented by the fig 4.1. This data set contains larger number of Hindi language text than Tamil

A picture containing bar chart

Description automatically generated

Fig 4.1 Frequency of language

* Contexts length frequency can be seen in the fig 4.2. There are contexts with maximum length of ~49000 and minimum length of 176.

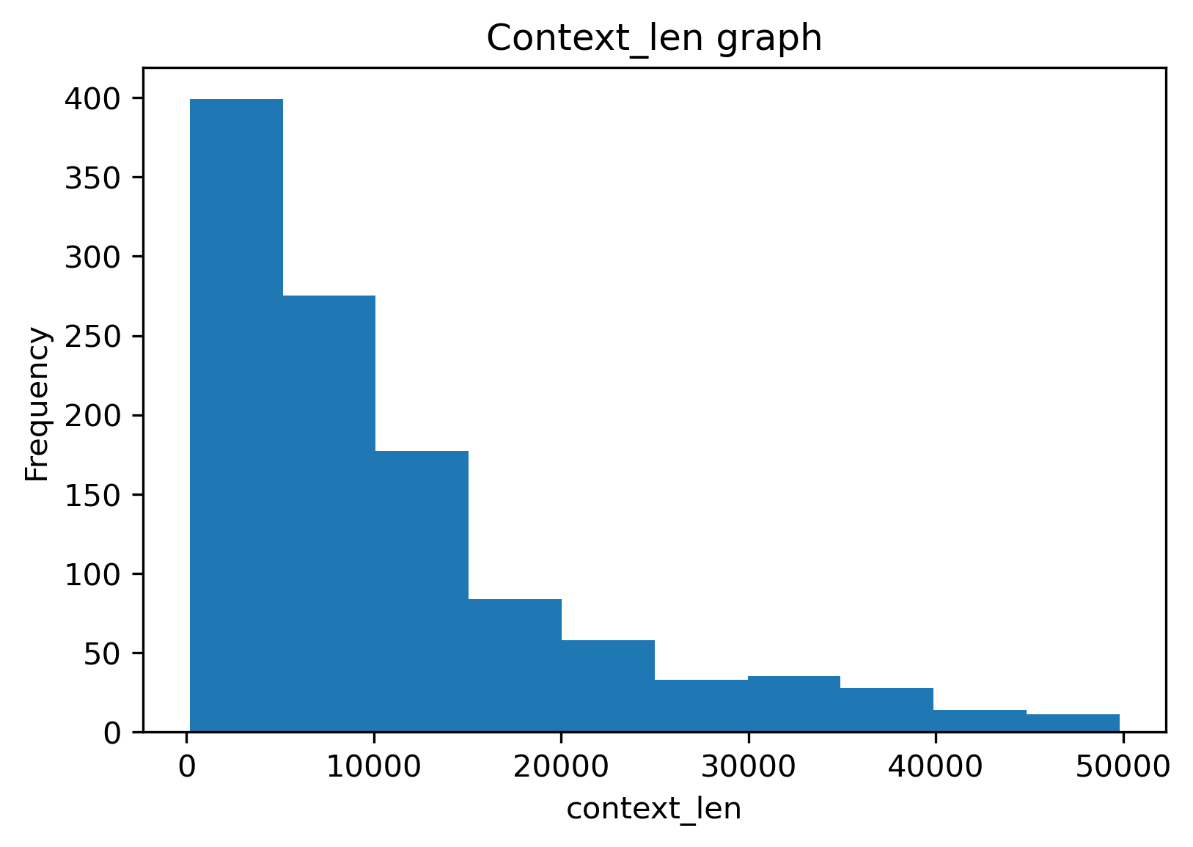


Fig4.2 Frequency of contexts length

* Questions length frequency can be seen in the fig 4.3. There are questions with maximum length of 121 and minimum length of 19.

Chart, histogram

Description automatically generated

Fig 4.3 Frequency of question length

* The word count for contexts can be seen in the fig 4.4. The maximum word count for contexts ~ 10000 is and minimum is 24.

Chart, histogram

Description automatically generated

Fig 4.4 Total word count in context

* The word count for contexts can be seen in the fig 4.5. The maximum word count for contexts 22 is and minimum is 3.

Chart, histogram

Description automatically generated

Fig 4.5 Total word count in questions

## **5. NLP - Question Answering Model**

We chose to work with the DistilBERT question answering model. As our system requirements, we needed a faster and smaller model. This model takes less time to train and less parameter to tune in. It is based on Bert question answering model but smaller and faster than base model. Some of the basic requirements for this model are explained below.

**Encoder: DistilBertTokenizerFast:**

DistilBertTokenizerFast is our tokenizer for this model. Which creates token for distilbert-base-uncased model. DistilBertTokenizerFast is identical to BertTokenizerFast and runs end-to-end tokenization: punctuation splitting and wordpiece. It has following Parameters

**vocab\_file**: Files of string types

**do\_lower\_case**: It is an optional Boolean type of parameter with default value True.

**unk\_token**: It is an optional str type of parameter with default value UNK. The unknown token. A token that is not in the vocabulary cannot be converted to an ID and is set to be this token instead.

**sep\_token**: It is an optional str type of parameter with default value SEP. The separator token, which is used when building a sequence from multiple sequences, e.g., two sequences for sequence classification or for a text and a question for question answering. It is also used as the last token of a sequence built with special tokens.

**pad\_token**; It is an optional str type of parameter with default value PAD. The token used for padding, for example when batching sequences of different lengths.

**cls\_token**: It is an optional str type of parameter with default value CLS. The classifier token which is used when doing sequence classification (classification of the whole sequence instead of per-token classification). It is the first token of the sequence when built with special tokens.

**mask\_token**: It is an optional str type of parameter with default value MASK. The token used for masking values. This is the token used when training this model with masked language modeling. This is the token which the model will try to predict.

**DistilBert For Question Answering Model**

DistilBert Model with a span classification head on top for extractive question-answering tasks like SQuAD (a linear layers on top of the hidden-states output to compute *span start logits* and *span end logits*). It our project it was trained on the Chaii dataset instead of SQuAD. It has following parameters.

**input\_ids** (torch.LongTensor of shape (batch\_size, num\_choices)) – Indices of input sequence tokens in the vocabulary.

**attention\_mask** (torch.FloatTensor of shape (batch\_size, num\_choices), *optional*) – Mask to avoid performing attention on padding token indices. Mask values selected in [0, 1]

1 for tokens that are **not masked**,

0 for tokens that are **masked**.

**head\_mask** (torch.FloatTensor of shape (num\_layers, num\_heads), *optional*) – Mask to nullify selected heads of the self-attention modules. Mask values selected in [0, 1]

1 indicates the head is **not masked**,

0 indicates the head is **masked**.

**start\_positions** (torch.LongTensor of shape (batch\_size,), *optional*) – Labels for position (index) of the start of the labelled span for computing the token classification loss. Positions are clamped to the length of the sequence (sequence\_length).

**end\_positions** (torch.LongTensor of shape (batch\_size,), *optional*) – Labels for position (index) of the end of the labelled span for computing the token classification loss. Positions are clamped to the length of the sequence (sequence\_length).

**Returns**

A [QuestionAnsweringModelOutput](https://huggingface.co/transformers/main_classes/output.html#transformers.modeling_outputs.QuestionAnsweringModelOutput) or a tuple of torch.FloatTensor

**loss** (torch.FloatTensor of shape (1,), *optional*, returned when labels is provided) – Total span extraction loss is the sum of a Cross-Entropy for the start and end positions.

**start\_logits** (torch.FloatTensor of shape (batch\_size, sequence\_length)) – Span-start scores

**end\_logits** (torch.FloatTensor of shape (batch\_size, sequence\_length)) – Span-end scores

## **6. Results**

After training our model, we can start predictions and asking questions about our mode. We can extract the start-end token range from our model. By access the start\_logits and end\_logits tensors we can perform the argmax functions like so:

**start\_pred = torch.argmax(outputs[‘start\_logits], dim = 1)**

**end\_pred = torch.argmax(outputs['end\_logits'], dim=1)**

The model’s prediction of start-end answer positions is given below.

**tensor ([ 1, 56, 89, 1, 2, 70, 1, 1, 410, 1])**

**tensor ([491, 504, 495, 468, 468, 489, 14, 8, 485, 8])**

## **7. Performance**

## We can measure the performance of the model by calculating the accuracy or exact match (EM). It will give an idea about the model accuracy that how close the predicted start, and the end position are with the exact start and end position.

## To calculate the EM of each batch, we take the sum of the number of matches per batch — and divide it by the total. We do this with PyTorch like so:

## acc = ((start\_pred = start\_true). sum() / len(start\_pred) ).item()

## The final .item() extracts the tensor value as a plain and simple Python int.

## In our case, Model accuracy is 50%. Means 50% of the time, the model can get the exact span of our correct answer. 50% is definitely not a good number. We can improve the model performance by finetuning the model and tunning model with different dataset.

## **8. Future Improvements**

We have used a highly optimized base model, but we could still experiment with other different question answering models.

There are so many rooms to improve the model performance by modifying the hyperparameters.

Distilbert model performs very well with the English dataset. We have used the Hindi & Tamil dataset for our current model. We can check or improve the accuracy of the model by changing the dataset or by modifying some parameters in our case.

For the multilingual dataset, we can experiment with other models listed in the method sections to see if the model gives better accuracy than this model.

**9. Credits**

Thanks, James Briggs, for his great articles “How-to Fine-Tune a Q&A Transformer” and Vivek Kumar for being an amazing Springboard mentor.

**10. references**

<https://huggingface.co/transformers/model_doc/distilbert.html>

<https://towardsdatascience.com/how-to-fine-tune-a-q-a-transformer-86f91ec92997>

<https://advertools.readthedocs.io/en/master/advertools.stopwords.html> <https://www.youtube.com/watch?v=HVBk2Ge_Q98>

<https://advertools.readthedocs.io/en/master/advertools.word_frequency.html#absolute-and-weighted-word-count>

<https://www.analyticsvidhya.com/blog/2019/02/stanfordnlp-nlp-library-python/>

[Question Answering with a fine-tuned BERT | Chetna | Medium | Towards Data Science](https://towardsdatascience.com/question-answering-with-a-fine-tuned-bert-bc4dafd45626)