Punctuation and case restoration in code mixed Indian languages

Subhashree Tripathy
Reverie Language Technologies,
Bengaluru
subhashree.tripathy
@reverieinc.com

Ashis Samal
Reverie Language Technologies,
Bengaluru
ashis.samal
@reverieinc.com

Abstract

Automatic Speech Recognition (ASR) systems are taking over in different industries starting from producing video subtitles to interactive digital assistants. ASR output can be used in automatic indexing, categorizing, searching along with normal human readability. Raw transcripts from ASR systems are difficult to interpret since it usually produces text without punctuation and case information (all lower, all upper, camel case etc.), thus limiting the performance of downstream NLP tasks. We proposed an approach to restore the punctuation and case for both English and Hinglish (i.e Hindi vocabulary in Latin script) languages. We have performed a classification task using encoder-based transformers which is a mini BERT consisting of 4 encoder layers for punctuation and case restoration instead of the traditional Seq2Seq model considering the latency constraint in real world use cases. It consists of a total number of 15 distinct classes for the model which includes 5 punctuations i.e Period(.), Comma(,), Single Quote(‘), Double Quote(“) & Question Mark(?) with different combinations of casing. The model is benchmarked on an internal dataset which was based on user conversation with the voice assistant and it achieves a F1(macro) score of 91.52% on the test set.

1 Introduction

Raw transcripts from ASR systems are difficult to interpret and not very user friendly for display purposes.To make the ASR transcripts more readable and interpretable, we need to include appropriate punctuation and segmentation at word and sentence level. We have experimented and pivoted to a unique word level classification approach with certain techniques of model optimizations making it useful in real time.

Punctuation are marks used in printed and written documents to separate sentences and clauses and to help make the meaning of sentences more clear.The standard English punctuation is as follows: period, comma, apostrophe, quotation, question, exclamation, brackets, braces, parenthesis, dash, hyphen, ellipsis, colon, semicolon.

Auto punctuation and capitalization is a way to automatically add punctuation and restore casing to a sentence thereby making it suitable to read for users.

Example 1:
Raw text : lets eat shyam
Converted text : Let’s eat, Shyam.
Example 2:
Raw text : shyam khaane chalein
Converted text : Shyam, khaane chalein?
Example 3:
Raw text : hello astor how are you
Converted text : Hello Astor, how are you?

Implementing auto-punctuation and capitalization on ASR output can improve its readability, have better display and help improve several downstream NLP tasks such as,

* Neural Machine Translation
* Sentimental Analysis
* Text summarization
* Named Entity Recognition

2 Motivation

In recent years, studies on ASR have shown outstanding results but there are still difficulties in standardizing the output of ASR[1] such as capitalization and punctuation restoration for speech transcriptions. The problems restrict readers to understand the ASR output semantically and also cause difficulties for natural language processing models such as NER, POS and semantic parsing. In this paper, we propose a method to restore the punctuation and case for ASR transcription.

Most of the punctuation and case restoration models work in Seq2Seq (Encoder-Decoder) neural

Proceedings of the Workshop on Unimodal and Multimodal Induction of Linguistic Structures (UM-IoS), pages 82 - 86

network architectures like T5, BART, GPT etc. Although these models are very good at generating long text sequences based on the task they are trained and fine tuned for, this comes with a challenge of high latency, which is a bottleneck for real time ASR systems. To address this challenge, we have framed the task as a classification problem.

3 Experiment details

We have approached the punctuation and case restoration task as a text classification problem where there are a total of 18 possible combinations of punctuation and casing, out of which we have considered 15 unique classes. Currently, our model supports 5 types of punctuation i.e period (represented as P), comma (C), question mark (?), single quote (SQ) and double quote (DQ) & 3 types of casing i.e lower cased (represented as OTHERS), upper cased (ALL_CAP) and sentence cased (CAP_INIT).
We have given a few examples of each category as mentioned in Table 1.

4 Model Architecture

Our base model is a pre-trained bert-mini model which has 4 bert encoder layers. We have wired two linear layers on top of it as a classification head for the word level text classification. This could process a maximum 256 tokens in one sequence of text[2].
The main purpose of using BERT-encoder is, it is faster in comparison to any Seq2Seq model and the context of words is learnt better which helps us understand the patterns of the language. A glimpse of how BERT[3] works is shown in Figure 1.

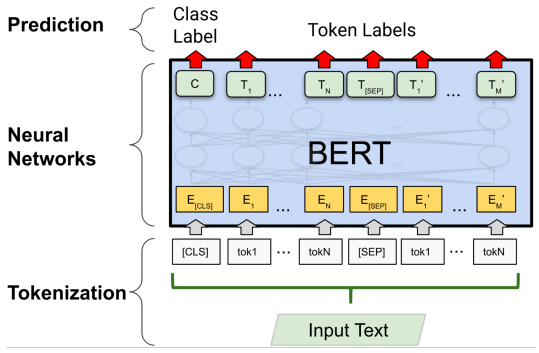

Figure 1: Bert architecture

The training of the task is done in two phases : -

1. **Pre-training :** The original sentence is usually passed to BERT and then tokenized using the word piece encoder, which generates contextual - embeddings i.e the embeddings depend on the context . Transformer reads the entire sequence of words based on its surroundings from both directions simultaneously instead on left to right/ right to left[4].

2. **Fine-tuning :** In order to fine tune the pre-trained BERT, we added a few layers at the end as well where the model learns to perform downstream tasks. The proposed methodology to our problem statement is a token classification approach where it predicts the punctuation mark associated with the given word. just as shown below.

   E.g : For the sentence, "i have a pen do you", the corresponding punctuation labels for it is predicted as, "CAP_INIT, OTHERS, OTHERS, P, CAP_INIT, Q" respectively.

5 Training Details

For model training purposes, textual data from publicly available NCERT textbooks along with prepared in-house data was used. Approximately 500000 sentences were used as training data which were cleaned and formatted to get rid of noisy data and make it suitable for a machine learning model. It consists of 15 unique labels i.e 'ALL_CAP', 'ALL_CAP_C', 'ALL_CAP_P', 'ALL_CAP_Q', 'ALL_CAP_SQ', 'C', 'CAP_INIT', 'CAP_INIT_C', 'CAP_INIT_P', 'CAP_INIT_Q', 'CAP_INIT_SQ', 'OTHERS', 'P', 'Q' & 'SQ'.
Since we are using a supervised learning technique, input data (lower case with removed punctuation) and their corresponding labeled data were fed to the model. We have performed the complete experiment in one Tesla V100 GPU system, which got 16 GB of memory.
Some of the hyper-parameters used in the training are as follows:

* Epochs : 15
* Warmup_steps : 500
* Train_batch_size: 128
* Learning_rate: 0.0001

It took around 4 hours of time to complete the training process. The accuracy of the model improves significantly with consistent training. Class wise distribution of different labels is in the Figure 2

Category	Example	Category	Example
CAP_INIT	Dial, What, Hey	DQ	“he, “said”
OTHERS	possible, there, hot	Q	person?
ALL_CAP	IPL, SBI, FM	C	here,
P	done., here.	ALL_CAP_P	JIO.
SQ	teacher’s/ ’teacher	ALL_CAP_SQ	CSK’s/ ’CSK
ALL_CAP_DQ	“JIO / ”JIO”	CAP_INIT_SQ	Jio’s/ ’Jio
ALL_CAP_Q	JIO?, ICICI?	CAP_INIT_DQ	“Jio
ALL_CAP_C	JIO,	CAP_INIT_Q	Jio?
CAP_INIT_P	Hello., Fine.	CAP_INIT_C	Jio,

Table 1: Labels with their examples

below. The y and x axis represent the labels and count of the labels respectively. The objective is

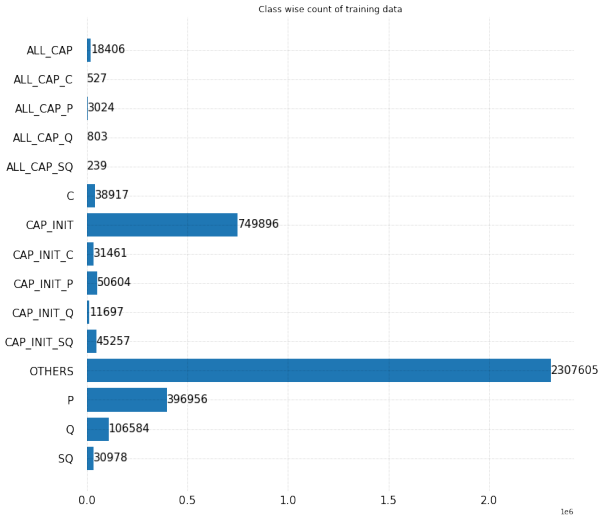

Figure 2: Class wise count of training data

to make the model output be as close as possible to the desired output or ground truth values. During model training, the model weights are adjusted iteratively to minimize the loss.

Cross entropy loss is popularly used in classification tasks both in machine learning and deep learning[5]. Cross-entropy is defined in Figure 3.

$$L_{\text{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \text{ for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

Figure 3: Cross entropy loss function

6 Model optimization

Model optimization helps us in achieving below objectives.

- **Smaller storage size :** Smaller models occupy less storage space on the deployed devices
- **Less memory usage :** Smaller models use less memory when they are running during inference
- **Latency reduction :** Latency is the time it takes to run a single inference with a given model. Some forms of optimization can reduce the amount of computation required to run inference using a model, resulting in lower latency. Latency can also have an impact on power consumption. Latency reduction is a major concern for us since we are integrating this with STT (Speech-To-Text) output and the overall result should not add more than 50ms latency to speech transcriptions. We have leveraged PyTorch JIT Compiler[6], which performs run-time optimization on model’s computation. TorchScript is the recommended model format for doing scaled inference with PyTorch models. We use torch.jit.trace and provide model and sample input as arguments. The input will be fed through the model as in regular inference and the executed operations will be traced and recorded into TorchScript.

7 Results

We prepared an internal testing dataset with 2050 data which was based on user conversation with the voice assistant. It consists of 15 different classes with macro-averaged F1- score[7] achieved is 91.527%.
Class wise precision score, recall score and F1-score is illustrated below in Table 2.
The class level confusion matrix from the test set

Class	Precision	Recall	F1-Score
ALL_CAP	0.996	0.963	0.979
ALL_CAP_C	1	1	1
ALL_CAP_P	0.857	0.909	0.882
ALL_CAP_Q	0.947	0.947	0.947
ALL_CAP_SQ	1	1	1
C	0.181	0.5	0.27
CAP_INIT	0.966	0.992	0.979
CAP_INIT_C	0.571	0.727	0.64
CAP_INIT_P	0.857	0.947	0.9
CAP_INIT_Q	0.883	0.892	0.887
CAP_INIT_SQ	0.987	1	0.993
OTHERS	0.996	0.983	0.990
P	0.944	0.978	0.961
Q	0.984	0.964	0.974
SQ	1	0.947	0.972

Table 2: Class level evaluation

performance is shown in Figure 4 below. The x-axis represents different classes of punctuation and the y-axis represents the predicted labels by the classifier. The blue diagonal denotes the percentage of true positives, i.e accurately detected classes which have a mean of 93.9%. The remaining yellow cells in the confusion matrix are false positives with respect to the predicted labels.

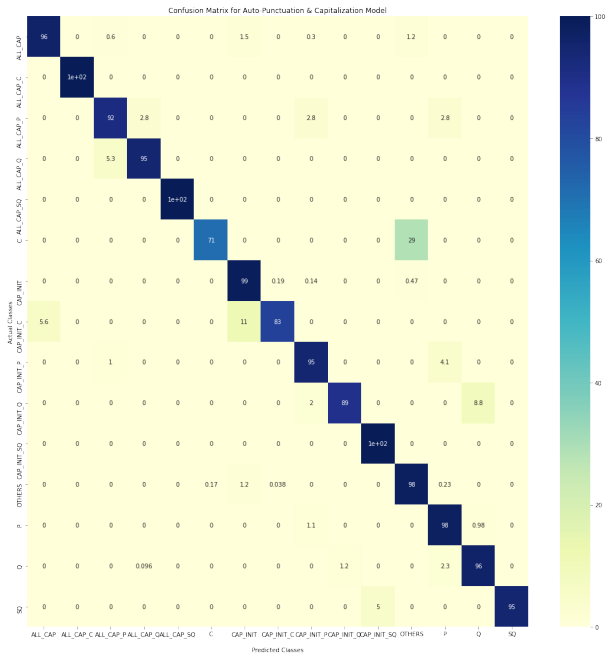

Figure 4: Class level confusion matrix of testing dataset

On a sentence level evaluation, the performance of our model on the test set is shown in Table 3.

In the table, the correctly predicted sentences is referred as True and the incorrectly predicted sentences is referred as False which has an accuracy score of 82%.

Sentences	True	False
2034	1664	370

Table 3: Sentence level count

8 Observations

Since our problem statement is framed as a classification task we have only used the encoders. We were able to reduce the computational power to half and reduce the latency significantly. Considering our model is trained on both English and romanized Hindi, there are some words which are spelled the same but mean completely different in different sentences which could cause ambiguity. Here's an example below.

Sentence 1 : Do you know me?

Sentence 2 : Do apple chahiye.

Although both sentences start with "Do", sentence 1 should end with a question mark ('?') while sentence 2 should end with a period('.'). We have trained the model with sentences using maximum possible ambiguous words in different contexts to handle these challenges due to the code mix. After benchmarking our test dataset, we observed that out of all the labels used, 'C' seems to be difficult to predict and place in the right position which could be due to less training data with commas. We could revisit the data preparation phase and include more sentences with "," in different positions and evaluate the model.

9 Limitation

There are a few limitations to our model. First being, not able to evaluate our model on any public dataset due to lack of resources in Hinglish data for auto-punctuation domain. Due to lack of hardware resources, our current model is limited to 32 tokens which is approximately 25 words in Hindi.

10 Future work

We would improve our existing model through the following steps.

– For better accuracy, we would add quality and diverse data to our training and validate our model on a public domain dataset and release

our Hinglish testset for more research and collaboration.

- We would optimize our model by further reducing the latency.
- We would include more punctuation types like exclamation marks, brackets (braces, parenthesis, square), dash, hyphen, ellipsis, colon, semicolon in further training.
- We would extend our language domain as well by including native and romanized versions of different Indian languages.
- In future, we plan to overcome the token count limitation so we can extend our model for longer sentences as well.

11 Conclusion

We present an approach to restore punctuation and case of the raw output obtained from the ASR system with significantly reduced latency. Currently our model is trained on English and Hinglish (i.e Hindi vocabulary in English script) data and achieves expected performance under different conditions.

Acknowledgements

Our research would not have been possible without the exceptional support from our colleagues from Speech team who helped us with suitable in-house resources to train and test on auto-punctuation and capitalization. Their valuable insights assited our research significantly and is highly appreciated.

References

[1]Attila Nagy, Bence Bial, Judit Ács, Automatic punctuation restoration with BERT-models, January 2021, URL: https://www.researchgate.net/publication/348618580_Automatic_punctuation_restoration_with_BERT_models

[2]Hugging Face : Bert-mini. https://huggingface.co/prajjwal1/bert-mini

[3]Google AI Blog : A Fast Word-Piece Tokenization System https://ai.googleblog.com/2021/12/a-fast-wordpiece-tokenization-system.html

[4]Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, October 2018, URL: https://arxiv.org/abs/1810.04805

[5]Machinelearningmastery : A Gentle Introduction to Cross-Entropy for Machine Learning https://machinelearningmastery.com/cross-entropy-for-machine-learning/

[6]PyTorch Tutorial : TORCHSCRIPT FOR DEPLOYMENT https://pytorch.org/tutorials/recipes/torchscript_inference.html

[7]Towardsdatascience : Multi-Class Metrics Made Simple, Part II: the F1-score https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-