Project Abstract: Multi-language Multi-aspect Review Sentiment Analysis for the hospitality sector

Project Aim: Semantic analysis

Project Approach: Concept-based Semantic analysis.

Limitations: language-specific differences in grammar and vocabulary, as well as cultural differences in sentiment expression.

Assumptions: Raw data is in text form.

Planning: Transform text into vector-> Pattern Recognition -> Predictive Analysis

Design Architecture:

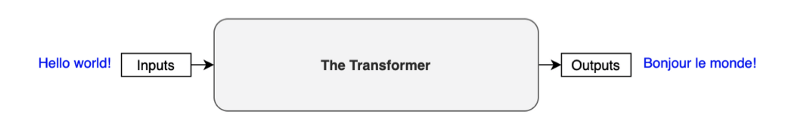
1)Transformers: [Het Shah](mailto:het.shah@htree.plus)

We can achieve this in two ways:

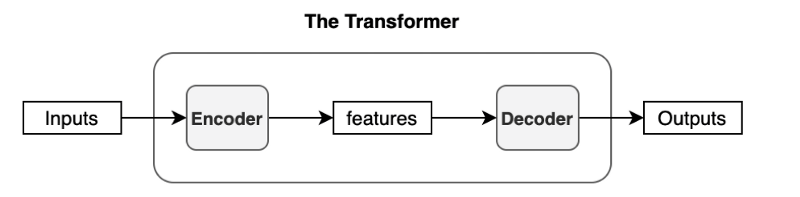
1. Creating an individual transformer(encoding-decoding model) for the language we have to translate.

**Encoder-Decoder Model:**

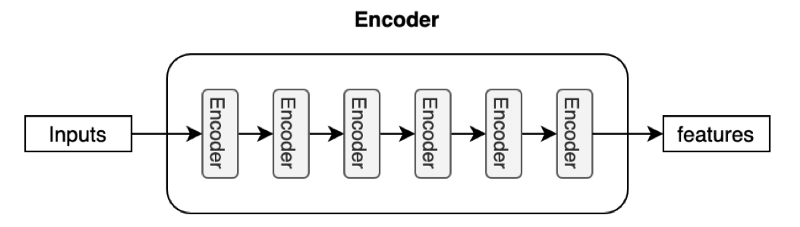
The original transformer published in the paper is a neural machine translation model. For example, we can train it to translate an English sentence into a French sentence.



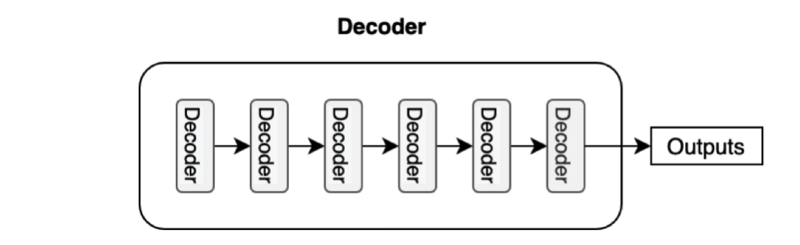
The transformer uses an encoder-decoder architecture. The encoder extracts features from an input sentence, and the decoder uses the features to produce an output sentence (translation).



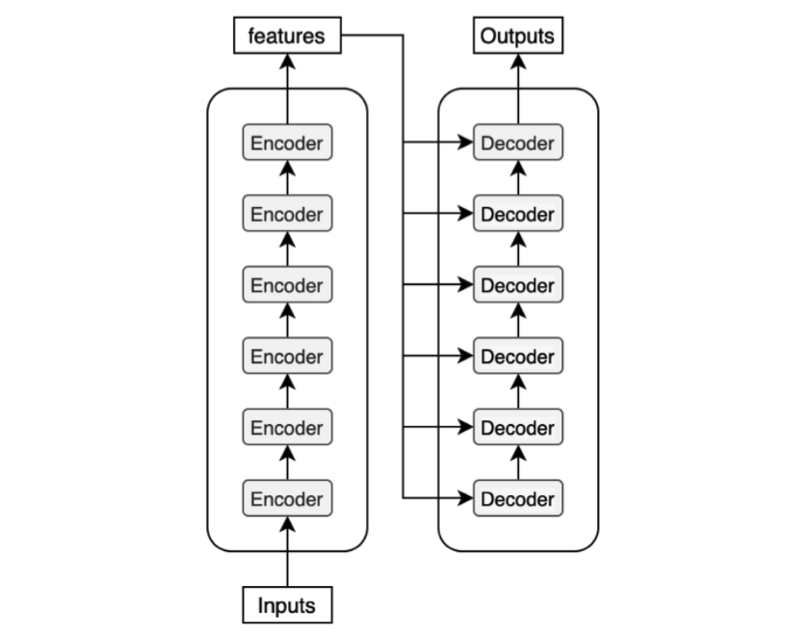
The encoder in the transformer consists of multiple encoder blocks. An input sentence goes through the encoder blocks, and the output of the last encoder block becomes the input features to the decoder.



The decoder also consists of multiple decoder blocks.



Each decoder block receives the features from the encoder. If we draw the encoder and the decoder vertically, the whole picture looks like the diagram from the paper.



**Input Embedding and Positional Encoding:**

We tokenize an input sentence into distinct elements (tokens) as we often do for any neural translation model. A tokenized sentence is a fixed-length sequence. For instance, if the maximum length is 200, every sentence will have 200 tokens with trailing paddings, which, if intuitively denoted, would look like below:

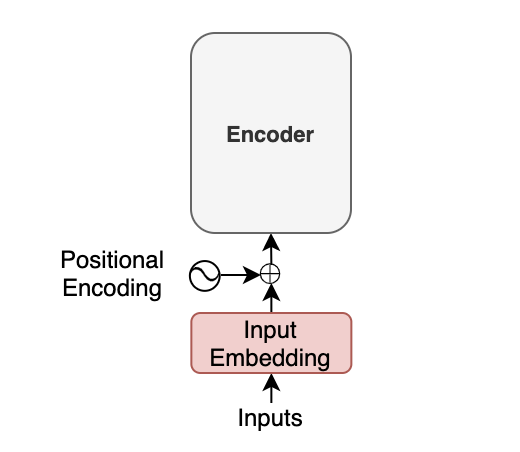
(‘Hello’, ‘world’, ‘!’, <pad>, <pad>, …, <pad>)

These tokens are typically integer indices in a vocabulary dataset. So, it may be a sequence of numbers like the below:

(8667, 1362, 106, 0, 0, …, 0)

The number 8667 corresponds to the token “Hello” in this example. It may also contain special characters like EOS(end-of-sentence marker). It depends on your tokenizer and vocabulary dataset.

To feed those tokens into the neural network, we convert each token into an embedding vector, a common practice in neural machine translation and other natural language models. For e.g, if we use a 512-dimensional vector for such embedding. So, if the maximum length of a sentence is 200, the shape of every sentence will be (200, 512). The transformer learns those embeddings from scratch during training.



Moreover, with the transformer, we inject positional encoding into each embedding so that the model can know word positions without recurrence.

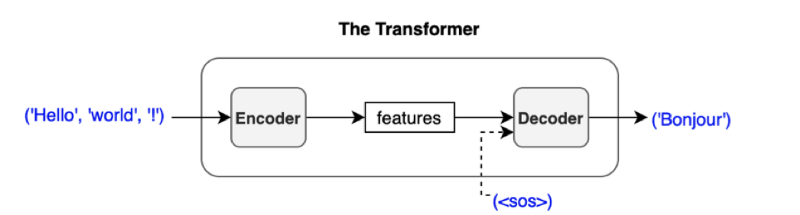
**Softmax and Output Probabilities:**

The decoder uses input features from the encoder to generate an output sentence. The input features are nothing but enriched embedding vectors.

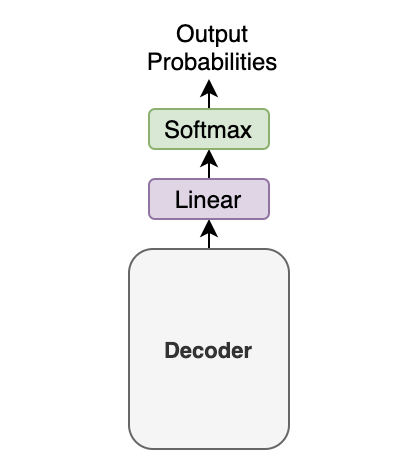
For simplicity, if we express a sentence like ('Hello', 'world', '!'), but the actual inputs to the encoder are input embeddings with positional encodings.

The decoder outputs one token at a time. An output token becomes the subsequent input to the decoder. In other words, a previous output from the decoder becomes the last part of the next input to the decoder. This kind of processing is called “auto-regressive” — a typical pattern for generating sequential outputs and not specific to the transformer. It allows a model to generate an output sentence of different lengths than the input.

However, we have no previous output at the beginning of a translation. So, we pass the start-of-sentence marker <SOS> (as known as the beginning-of-sentence marker <BOS>) to the decoder to initiate the translation.



Then, the output vector from the decoder goes through a linear transformation that changes the dimension of the vector from the embedding vector size (512) into the size of vocabulary (say, 10,000). The softmax layer further converts the vector into 10,000 probabilities.



**Generative Pre-trained Transformer 3 (GPT-3)**

GPT-3 is a language generation model developed by OpenAI. It is one of the largest and most advanced language models to date, with over 175 billion parameters. It is trained on a diverse range of internet text, allowing it to generate translated languages, and perform other language-related tasks with remarkable accuracy.

GPT-3 has demonstrated impressive accuracy in various language tasks such as text generation, question answering, and summarization. However, like all AI models, it is not perfect and may make mistakes, provide irrelevant or incorrect answers, or produce biased results. The accuracy of GPT-3 largely depends on the quality of the input data and the specific task it is trained for. Overall, GPT-3 has received praise for its ability to perform a wide range of language tasks with high accuracy and fluency

**MT5 Model**

mT5 is a multilingual Transformer model pre-trained on a dataset (mC4) containing text from 101 different languages. The architecture of the mT5 model (based on T5) is designed to support any Natural Language Processing task (classification, NER, question answering, etc.) by reframing the required task as a sequence-to-sequence task.

In other words — text goes in, and text comes out. For example, in a classification task, the input to the model can be the text sequence to be classified, and the output from the model will be the class label for the sequence. For translation, this is even more straightforward. The text that goes in is in one language, and the text that comes out is in another.

**RNN Model**:

RNN Model can also translate the language but the main difference between RNN and Transformer is the attention layer. Transformer makes use of multi-attention layer in order to translate the whole word at a time whereas RNN model doesn't have any attention layer

So there might be replace of words at the Translation Time.

2) Concept/Aspect-Based Analysis(HLD): [Devansh Mistry](mailto:devansh.mistry@htree.plus)

I have clearly mentioned data requirement. Change in data will further require more time.

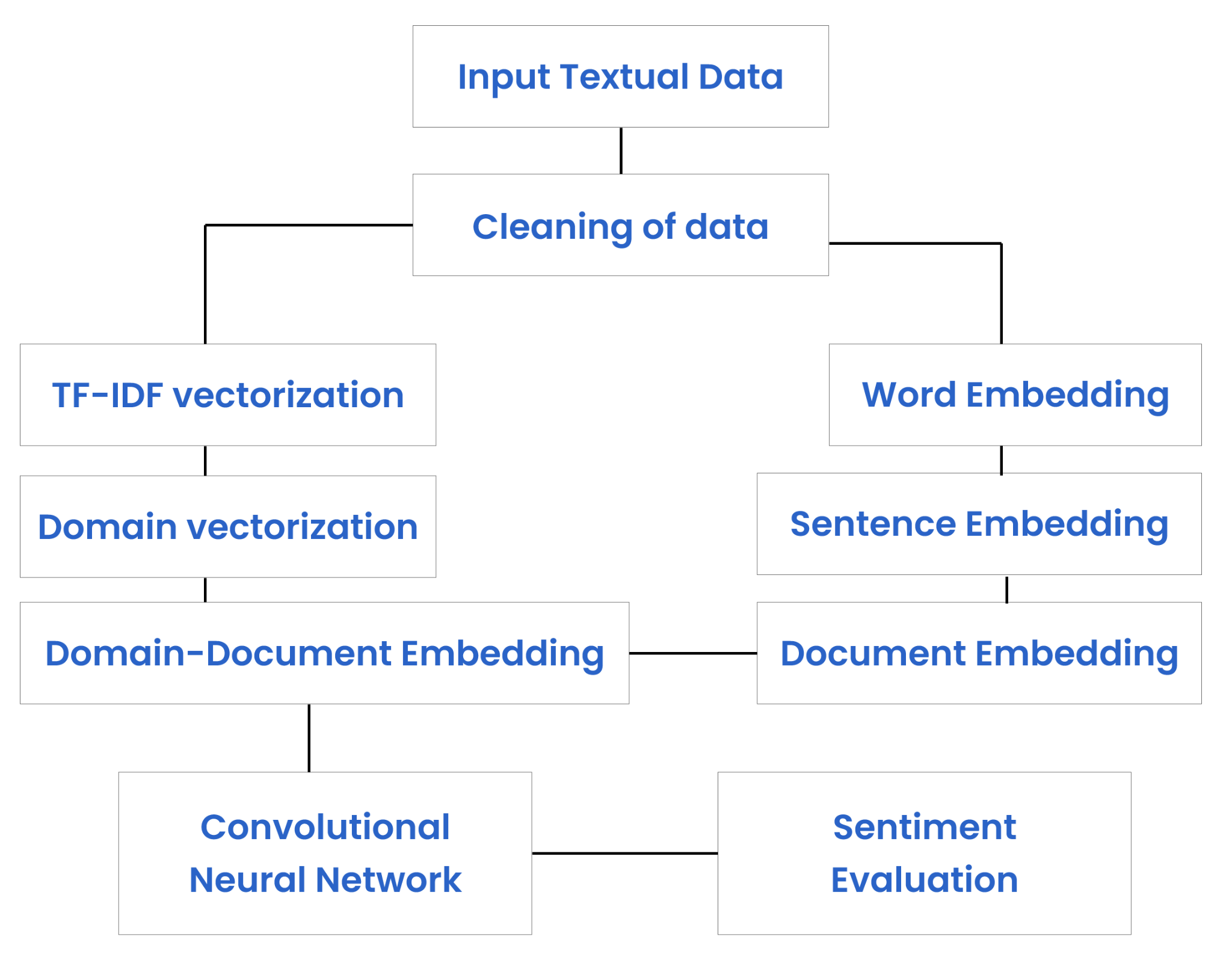
Abstract: Semantic analysis is the process of understanding the meaning of a piece of text. It can be done using various coding approaches, such as rule-based, statistical, or machine-learning methods.

* Rule-based approach: This approach uses a set of hand-written rules to identify the meaning of the text. It is simple to implement but has limited accuracy and scalability.
* Statistical approach: This approach uses statistical models, such as n-grams and hidden Markov models, to identify patterns in the text and determine its meaning. It is more accurate than the rule-based approach but requires large amounts of annotated training data.
* Machine learning approach: This approach uses algorithms, such as decision trees, random forests, and deep neural networks, to learn patterns in the text and determine their meaning. It is more scalable and accurate than the other two approaches but requires significant computational resources and large annotated training data.

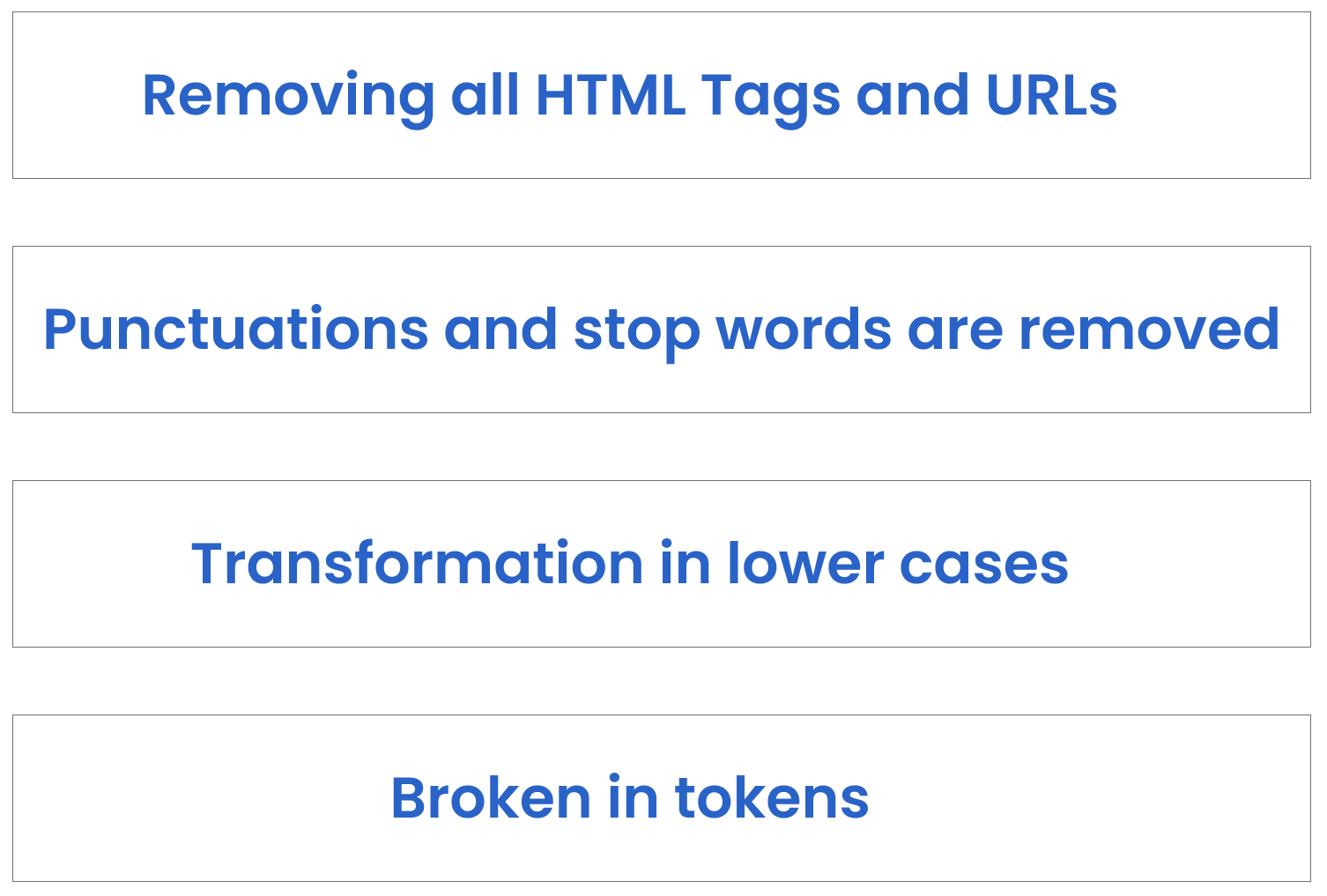
The choice of approach depends on various factors such as the size of the text data, the desired level of accuracy, computational resources, and the availability of annotated training data. In the ML approach, we have options to choose:

1. We will achieve this by CNN-TDIDF family models, which combine CNN and TF-IDF in parallel using deep learning. The same figure follows 4th option.

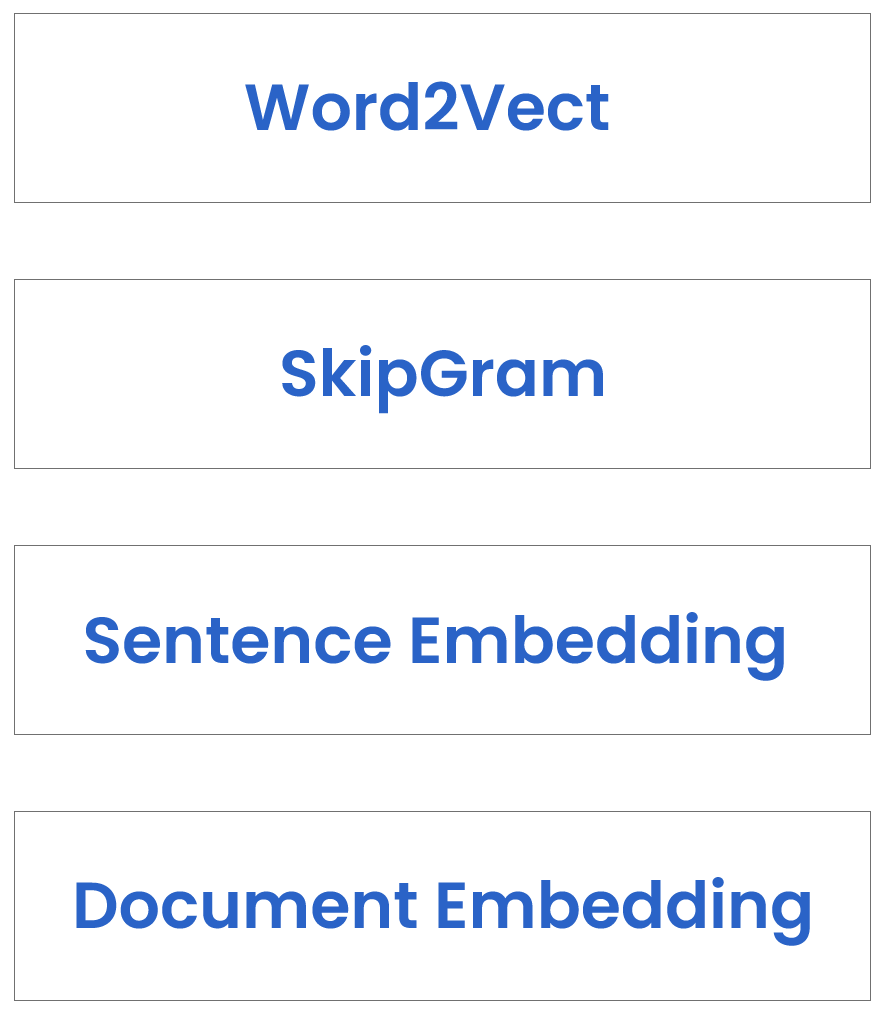
Overall Structure:



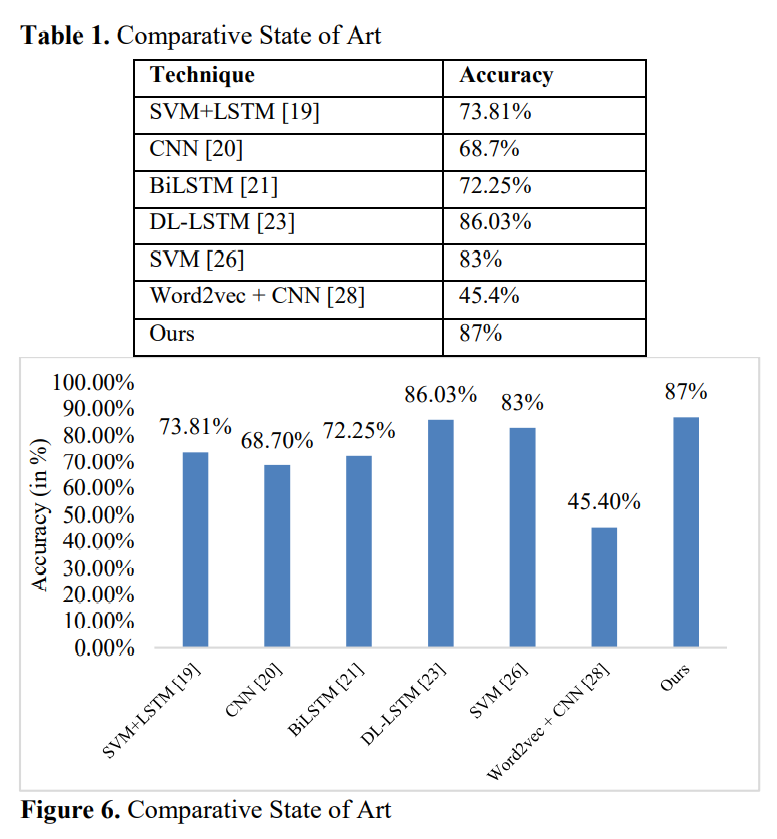
Data Cleaning:



Final HLD Pipeline:



Comparison of Approaches:



A simple model like the linear TF-IDF model already provides outstanding accuracy. Using more complex models does not improve accuracy, but costs much more time: the RNN model needs 20 times more time than the TF-IDF. The BERT model needs more than 1000 times more time than the TF-IDF! Again Data From clients is going to play a bigger role.

1. PyABSA:

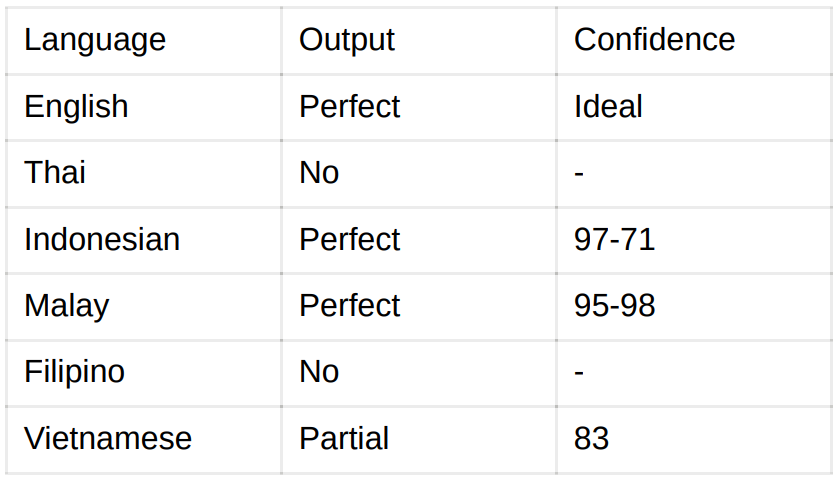
Requirement: Labeled Data

Sample Csv: <https://drive.google.com/file/d/16X16n4lXG9totiEKnnFLBGuuBo_18oqe/view?usp=sharing>

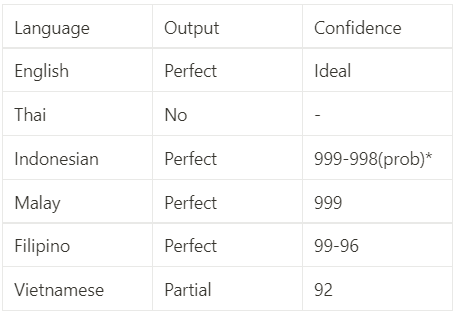
Idea:

In this library, we can directly use the model for English, Indonesian, Malay, and Filipino. We have a good output for Thai and Vietnamese, but not satisfactory. Here is the result on sample data:

Experiment Details(“multilingual”)



Experiment Details(“multilingual-256-2”)



Here I have used this statement for evaluation:

Camera quality is very good but battery drains fast

Perfect Means: Camera with positive, Battery with negative

Partial Means: Camera with positive

No Means: no output

So we will fine-tune the model. If the output will be not as expected, we have to work for separate models for particular languages. Let’s see how this library works:

* “No Translating”
* Aspect Extraction
* Classify polar sentiments
* Sentiment Connection-Output Building

This library is pre-trained with serval good datasets, but for higher accuracy, we will try to fine-tune with user data and analyze the impact on accuracy.

Limitation: this approach will require a proper environment and dependencies.

Links for Library & Experimental Notebook:

<https://pyabsa.readthedocs.io/en/v2/>

<https://github.com/yangheng95/PyABSA>

<https://colab.research.google.com/drive/1k5o8LpIXQ35ZVRn7bRqjsHL0T9ij_RQm?usp=sharing>

<https://github.com/yangheng95/PyABSA/blob/release/demos/aspect_term_extraction/train_atepc_english.py>

<https://github.com/yangheng95/ABSADatasets>

Code for tuning in this:

“# Import required libraries

import pyabsa

import pandas as pd

# Load the annotated data into a Pandas DataFrame

df = pd.read\_csv('annotated\_data.csv')

# Split the data into training and testing sets

train\_data = df[:int(len(df) \* 0.8)]

test\_data = df[int(len(df) \* 0.8):]

# Initialize a PyABSA sentiment analysis model

model = pyabsa.SentimentAnalysisModel()

# Train the model on the annotated data

model.fit(train\_data['review'].tolist(), train\_data['sentiment'].tolist())

# Evaluate the performance of the model on the testing set

predictions = model.predict(test\_data['review'].tolist())

accuracy = pyabsa.metrics.accuracy(predictions, test\_data['sentiment'].tolist())

print("Accuracy:", accuracy)

# Save the model to disk

model.save('sentiment\_analysis\_model.pkl')”

1. GPT-3 API solution:

This solution will use gpt-3 model and do ABSA on API calls, here is an example:

<https://colab.research.google.com/drive/1agHC6wiGCYSgdtvNhRJTVW2KuCXyGgJJ?usp=sharing>

Limitation:

* API key will take cost

1. Make everything by our side. Making separate transformers for each language and making the ASBA model with the process mentioned above in 1st approach, going to be time-consuming, and the final deliverable maybe not be achieved on time. Domain expertise will be required.
2. Hugging Face & other Pre-trained Models/Transformers, using Transfer Learning.