Rapport

Salah eddine teffahi

Abdel malek slamani

Imad eddine boukader

Introduction

In the realm of social media analysis, understanding the semantic similarity between tweets holds significant importance, particularly in discerning whether multiple tweets originate from the same user. To address this challenge, our project aims to develop a sophisticated model capable of analyzing the semantic resemblance of two tweets and assigning a similarity score, ranging between 0 and 1, denoting the probability that they were authored by the same individual. Leveraging cutting edge transformer architectures for text representation and employing advanced distance calculation techniques, this endeavor seeks to provide a robust solution for discerning tweet authorship, thereby facilitating various applications in user profiling, content moderation, and information verification. The motivation behind this project lies in enhancing the accuracy and efficiency of social media analysis, contributing to improved user experience and platform integrity. Our primary goal is to create a versatile tool that not only aids in identifying tweet similarities but also fosters a deeper understanding of user behavior and communication patterns across digital platforms.

This Python code is used to create a DataFrame of pairs of tweets, labeled as either positive or negative, based on whether they are from the same user or not.

Data Preparation

Firstly, we created an empty DataFrame `pairs_df` with columns 'tweet1', 'tweet2', and 'label'.

(we wanted to try concatenating both tweets in one column and at the beginning of every tweets we put tweet 1 :… tweet2: … to experiment if it better then separating them and giving two different features to the model)

We use we then iterates over each unique user in the DataFrame `train_dropped`. For each user, we retrieves all tweets from that user and generates all possible combinations of pairs of these tweets,The number of pairs to be sampled is determined by the smaller of the number of tweet pairs for the current user and the number of tweets from other users. This is done to ensure an equal number of positive and negative pairs.

Positive pairs are randomly sampled from the tweet pairs of the current user, and negative pairs are randomly sampled from the tweet pairs of all other users. These pairs are then added to the `pairs_df` DataFrame. Positive pairs are labeled with '1' and negative pairs are labeled with '0'.

The resulting DataFrame contains pairs of tweets along with a label indicating whether they are from the same user or not.

we lowercase all text, remove punctuation and symbols (except hashtags and mentions), and utilize BERT Base and DistilBERT tokenizers for further refinement.

The BERT tokenizer is used to tokenize input text into tokens suitable for processing by the BERT model. It employs a technique called WordPiece tokenization, which breaks down words into smaller subword units. This allows BERT to handle out-of-vocabulary words by breaking them down into subword units that it has been trained on.

In the training step we couldn't try a large number of epochs or big data because every time we tried with more then 3 epochs and 15k row the training in kaggle stops because of connection lost, we had several problems creating a dataloader but we abandoned the idea

Model architecture

Our architecture is a PyTorch model that uses a transformer model (like BERT) as its base. The model is designed to take two sets of inputs, process them separately through the transformer, and then compare the results.

1. `__init__(self, transformer_model)`: This is the constructor of the class. It takes a transformer model as input and initializes two linear layers (`fc` and `fc2`). The first linear layer (`fc`) has 768 input features and 1 output feature, which matches the output dimension of a BERT model. The second linear layer (`fc2`) is a simple 1-to-1 mapping.

2. `forward(self, input_ids1, attention_mask1,input_ids2, attention_mask2)`: This is the forward pass of the model. It takes two sets of inputs: `input_ids1` with `attention_mask1` and `input_ids2` with `attention_mask2`. Each set of inputs is processed separately: - The inputs are passed through the transformer model. The `[0]` indexing is used to get the output of the last layer of the transformer model. - The mean of the transformer outputs is calculated across the sequence length dimension (`dim=1`). - The mean output is passed through the `fc` linear layer. - The output of the linear layer is passed through a sigmoid activation function.

3. After processing both sets of inputs, the absolute difference between the two outputs is calculated. This "distance" is then passed through the second linear layer (`fc2`), and the final output of the model is obtained by applying a sigmoid activation to the output of `fc2`.

The model is designed for tasks where the relationship between two inputs needs to be determined, such as sentence similarity or paraphrase detection. The final output is a single value between 0 and 1, which could represent the probability that the two inputs are similar or the degree of similarity between them.