Project Report

This report provides an in-depth analysis of the project, including the objectives, methodology, model architecture, and results of Tweet Similarity Analysis with Transformer Embeddings

Realised by : -Amrouche Nassiba Master 1 IA

-Mohamedmahmoud ikram

Project Objectives

1

Data Preparation:

Create Tweet Pairs: -Randomly sample pairs of tweets from the dataset.

Labeling: -Assign a similarity label to each pair: -Same-user pairs: Label 1 (high similarity), Different-user pairs: Label 0 (low similarity).

2

Data Preprocessing:

Text Cleaning: - Lowercase all text. -Remove punctuation, symbols, and other special characters (consider keeping hashtags and mentions). - Tokenize tweets into individual words. - Explore options like stop word removal and stemming/lemmatization, analyzing their impact on model performance.

3

Model Architecture:

Embedding Layer. -Transformer Encoder. -Feature Extraction. -Manhattan Distance.

Dense Layer.

4

Evaluation:

Evaluation Metrics: -Precision, Recall, F1 Score.

Methodology

1. **Data Loading:**
   - Load the training and test data from Excel files (`train.xlsx` and `test.xlsx`).
2. **Random Tweet Pair Generation:**
   - Define a function `create_random_tweet_pairs` to generate pairs of tweets randomly.
   - Separate tweets by user.
   - Randomly select two users and a tweet from each user to form a tweet pair.
   - Label the pairs as same-user or different-user pairs based on the users.
3. **Data Preprocessing and Labeling:**
   - Clean the text of each tweet using the `clean_text` function:
     - Convert to lowercase.
     - Remove punctuation, symbols, and special characters.
     - Tokenize tweets into individual words.
     - Remove stopwords.
     - Lemmatize words.
   - Label the tweet pairs based on whether they are from the same user or not using the `label_tweet_pairs` function.

4. Text Encoding with BERT:

- Load the pre-trained BERT tokenizer and model.
- Define a function `encode_tweets` to encode the tweet pairs using BERT embeddings.
- Tokenize and encode tweets using BERT tokenizer.
- Pass the encoded inputs through the BERT model to get the contextual embeddings.
- Use the mean of the last hidden state as the representation of each tweet.
- Return the encoded tweet pairs with their labels.

5.Model Definition and Training:

- Define a neural network model `SimilarityModel` using PyTorch.
- Initialize the model with appropriate input dimension.
- Define the loss function (`BCELoss`) and optimizer (Adam).
- Train the model for a specified number of epochs:
  - Compute the Manhattan distance between tweet embeddings.
  - Pass the distance through a dense layer to get the similarity score.
  - Calculate the loss between predicted and true labels.
  - Backpropagate the loss and update model parameters.

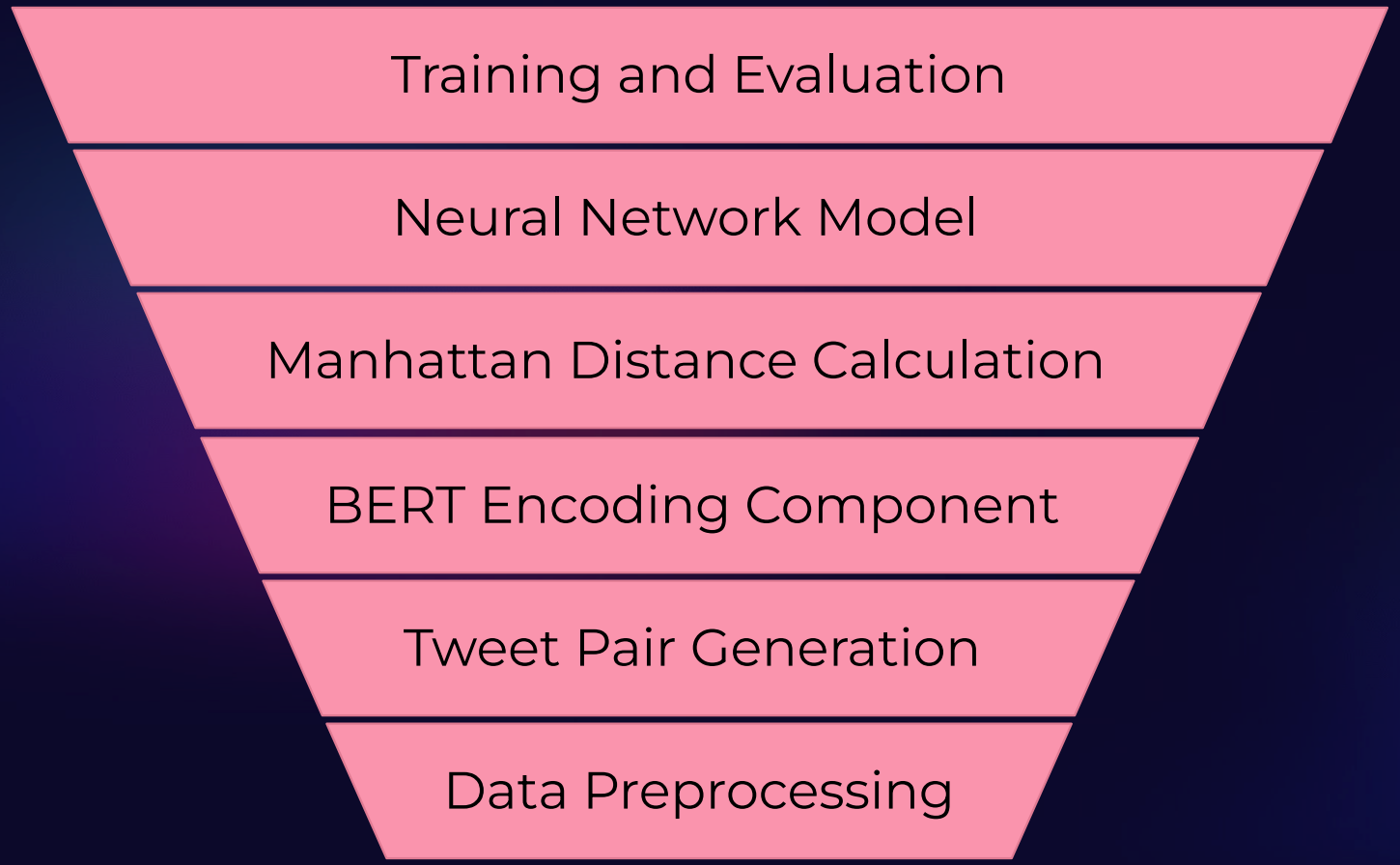
6.Evaluation:

- Evaluate the trained model on the test set:
  - Pass the Manhattan distance through the trained model to get the similarity score.
  - Convert the score to a binary prediction based on a threshold.
  - Compute precision, recall, and F1 score using scikit-learn metrics.

Model Architecture

1. **Data Preprocessing Component:**
    - This component includes functions for cleaning and preprocessing the raw tweet text data. It performs tasks such as converting text to lowercase, removing punctuation, tokenizing text into words, removing stopwords, and lemmatizing words.
2. **Tweet Pair Generation Component:**
    - This component generates pairs of tweets for training and testing the model. It randomly selects tweets from the dataset and forms pairs, ensuring a balance between pairs from the same user and pairs from different users.
3. **BERT Encoding Component:**
    - This component utilizes a pre-trained BERT model and tokenizer to encode the tweet pairs into contextual embeddings. It tokenizes the tweet text, passes the tokenized inputs through the BERT model, and extracts the contextual embeddings for each tweet.
4. **Manhattan Distance Calculation Component:**
    - After obtaining the contextual embeddings for tweet pairs, this component calculates the Manhattan distance between the embeddings of corresponding tweets in each pair. The Manhattan distance is a measure of similarity between two vectors and is computed as the sum of absolute differences between corresponding coordinates.
5. **Neural Network Model Component:**
    - The neural network model component defines the architecture of the similarity prediction model. It consists of a single dense layer followed by a sigmoid activation function. The input dimension of the dense layer corresponds to the dimension of the Manhattan distance output.
6. **Training and Evaluation Component:**
    - This component handles the training and evaluation of the neural network model. It utilizes a binary cross-entropy loss function for training and optimizes the model parameters using the Adam optimizer. After training, it evaluates the model's performance on a separate test dataset using metrics such as precision, recall, and F1 score.

Model Architecture



This diagram illustrates how data flows through various components of the model, from initial preprocessing to final evaluation, with each component performing specific tasks in the process of tweet similarity .

Results

1 Precision

The precision score indicates the proportion of correctly identified similar tweet pairs out of all tweet pairs predicted as similar. A precision score of 0 means that none of the predicted similar tweet pairs were actually similar. This could occur if the model is too conservative in predicting similarities, resulting in a high number of false positives.

2 Recall

Recall: The recall score indicates the proportion of correctly identified similar tweet pairs out of all actual similar tweet pairs in the testing set. A recall score of 0 means that the model failed to identify any of the actual similar tweet pairs. This could happen if the model is too selective and misses many true positives.

3 F1 score

The F1 score combines precision and recall into a single metric. A low F1 score indicates that the model either has low precision, low recall, or both. It is important to strike a balance between precision and recall to achieve a high F1 score.