Codalab username : nagasai3810

EmoInt

Inetially I have tried multiple models and evaluated the results. Then based on the evaluation result I've selected the best performing model and fine tune it for further.
For static approach I've tried with linear regression, SVM, word embeddings and random forest. Among those I found that the random forest giving the best result.

1.Static Method:

Using Randomforestregressor

Library Imports:

The code begins by importing necessary libraries, including scikit-learn for machine learning, pandas for data manipulation, regular expressions for text processing, NLTK for natural language processing, and emot for handling emojis.

Data Loading and Preprocessing:

Three datasets are loaded from CSV files, representing training, development, and test data.
The datasets are concatenated to form the training set.
The tweet texts and corresponding fear scores are extracted as features (X) and labels (y).
The training set is further split into training and validation sets using a 80-20 split ratio.

Text Preprocessing Functions:

Two functions are defined for text preprocessing:
convert_emojis: Replaces emojis in text with corresponding text representations.
preprocess_text: Applies various text cleaning steps, including removing HTML tags, special characters, URLs, and performing tokenization, lowercase conversion, stop word removal, and stemming.

Pipeline Definition:

A scikit-learn pipeline is defined with three steps:
preprocess: Applies the preprocess_text function to each tweet.
tfidf: Utilizes TF-IDF vectorization with a maximum of 5000 features.
model: Uses a RandomForestRegressor model with 100 estimators.

Model Training:

The pipeline is trained on the entire training dataset (X and y).

Making Predictions:

The trained model is used to make predictions on the test set (X_test).
The predicted scores are added to the 'score' column of the test dataframe.

Saving Results:

The final predictions, including tweet IDs, texts, emotions, and predicted scores, are saved to a CSV file named 'fear-prediction.txt'.
In summary, the code constructs and trains a machine learning pipeline to predict fear scores for tweets. It incorporates extensive text preprocessing, TF-IDF feature extraction, and employs a RandomForestRegressor model. The predictions are saved to a file for further analysis or evaluation.

2.Deep Learning Method:

Library Imports:

Import necessary libraries, including NumPy for numerical operations, Pandas for data manipulation, scikit-learn for train-test splitting, TensorFlow's Keras for building and training the neural network, and Hugging Face's Transformers for BERT model and tokenizer.

Load and Preprocess Data:

Load a dataset from a CSV file containing tweet text and emotion intensity scores into a Pandas DataFrame.
Preprocess the data by extracting tweet texts (X) and intensity scores (y).

Tokenization and BERT Embeddings:

Use the BERT tokenizer to tokenize the tweet texts and pad sequences to a maximum length of 128.
Load the BERT model and extract BERT embeddings for each tokenized sequence.

Train-Test Split for Embeddings:

Split the BERT embeddings and intensity scores into training and testing sets using scikit-learn's train_test_split.

Build Neural Network Model:

Create a simple feedforward neural network using TensorFlow's Keras Sequential API.
The model consists of a hidden layer with 64 units and ReLU activation, followed by an output layer with 1 unit and linear activation.

Compile and Train Model:

Compile the model using the Adam optimizer and mean squared error loss function.
Train the model on the training set for 100 epochs with a batch size of 32, using the testing set for validation.

Evaluate and Print Correlation:

Use the trained model to predict intensity scores for the entire dataset.
Calculate both Pearson and Spearman correlation coefficients between the true intensity scores and predicted scores.
Print the overall Pearson and Spearman correlations.