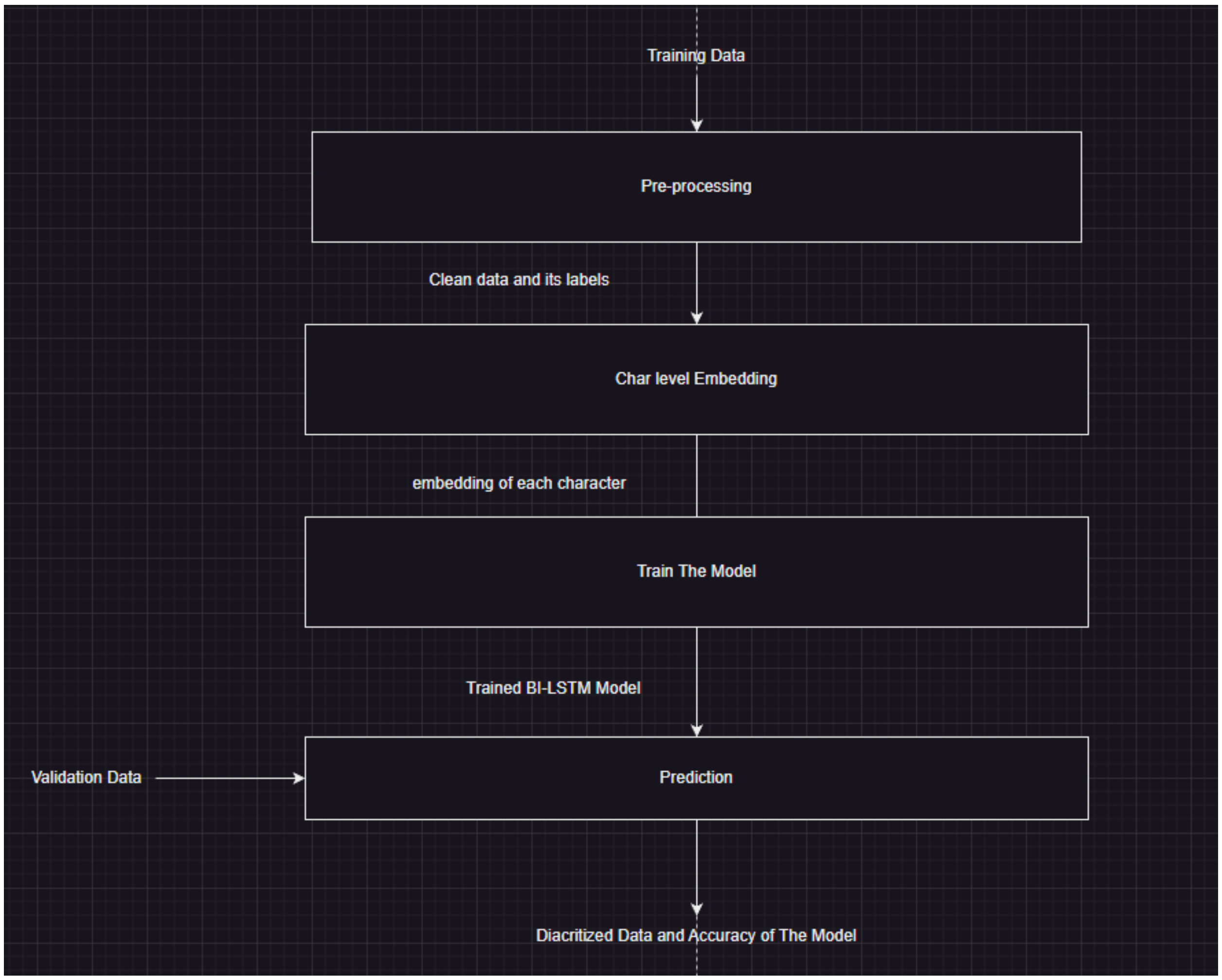
NLP Project

Arabic Text Diacritization

Name	Sec	Bn
Moaz Mohamed Hassan	2	25
Youssef Elsayed Elwaer	2	38
Karim Mahmoud Kamal	2	12
Mohamed Nabil Abdelfattah	2	19

Pipeline


Training Data
Pre-processing
Clean data and its labels
Char level Embedding
embedding of each character
Train The Model
Trained BI-LSTM Model
Validation Data
Prediction
Diacritized Data and Accuracy of The Model

Preprocessing

In data preprocessing we do the following:

1. Clean the data:
    - Remove numbers and un wanted characters like that (brackets, slashes, some signs like that “,»–';«*~”)
    - Remove brackets.
    - Remove extra spaces.
2. Remove diacritics form training data.
3. Save the cleaned data with diacritics to extract labels and without to train the model.

Feature Extraction

We have tried multiple of features and compare between accuracy of them:

1. Bigram based features

    We have used sickit-learn function of ‘CountVectorizer’ with these arguments:

    - ngram_range=(2, 2): This parameter specifies the range of n-grams to extract. In this case, it's set to (2, 2), meaning it extracts only bigrams (pairs of consecutive characters).
    - analyzer='char_wb': This parameter specifies the type of analyzer to be used. 'char_wb' stands for character n-grams with word boundaries. It means that the vectorizer considers word boundaries when forming n-grams.
2. TF-IDF based features
    - We have used sickit-learn function of ‘TfidfVectorizer’ with the same parameters that means we care about the importance of bigram on character level ‘two consecutive characters’
3. Char level embeddings
    - Inside our CharLSTM class, we declare Emedding layer, that takes a char index and returns a 200 float number vector, representing this char.
    - It is then used as input to BiLSTM network.

Model Training

We have tried simple RNN and Bi-LSTM and compared between accuracy of them:

1. **RNN with Bigram**
    - We have created a data loader of batch size of 32 and have passed features of each character to the model by batches with the size of [32,1,72] that refers to the following:
        - 32: batch size
        - 1: features of one character
        - 72: dimension of the feature
    - The model was consisting of simple RNN layer with dimension of [72, 64] (features_dim*hidden_layer_size)
2. **RNN with TF_IDF**

    As same as the previous model but using TF-IDF instead of Bigram
3. **B-LSTM with character level embedding**
    - We create array of arrays, that represents char index of every char in every sentence, and create the same array but with labels indexes for every char, and pad the small sentences until every array reaches the max_len size.
    - The model layers are:
        - Embedding layer [unique_chars_size, embedding_dimension].
        - Bi-LSTM network, with input size = embedding_dimension.
        - Linear layer (output layer) with input size = hidden size * 2 (because of Bi) and output size = labels size.
    - Bi-LSTM processes the input sequence in both the forward and backward directions, final_output = W * concatenated_hidden_states + bias.
    - In the training loop, we pass every batch of sentences sequences to the forward pass, then we calculate loss and optimize weights using gradient decent.

- In validation, the same but don't calculate loss, instead calculate accuracy, using correct_predictions / total_predictions, ignoring padding chars and non-Arabic chars.

4. **LSTM with character level embedding**
    - Same as above but not bidirectional, changing output linear layer input size to only hidden size not hidden size * 2.
    - It had less accuracy than Bi-LSTm.

Evaluation

Model and Parameters	**DER**
RNN with TF-IDF	(100% - 61%) = 39%
RNN with Bi-gram	(100% - 61%) = 39%
LSTM with char embeddings (embedding_size=200, hidden_size=256, lr=0.01, num_layers=2, num_epochs=10, max_len=500, batch_size=256)	(100% - 65%) = 35%
Bi-LSTM with char embeddings (embedding_size=200, hidden_size=256, lr=0.001, num_layers=3, num_epochs=12, max_len=100, batch_size=256)	(100% - 96.56%) = 3.44%
Bi-LSTM with char embeddings (embedding_size=300, hidden_size=256, lr=0.001, num_layers=5, num_epochs=19, max_len=600, batch_size=32, dropout_rate=0.2, lr_step=5, lr_gamma=0.1)	(100% - 97.778%) = 2.222%