Bangla POS and NER Tagging: Development Documentation

Overview

This project involves developing an end-to-end machine learning system for performing Named Entity Recognition (NER) and Parts of Speech (POS) tagging on Bangla text. The system is implemented using a neural network model trained on synthetically generated data and deployed via a FastAPI-based inference API.

1. Data Analysis and Preprocessing

1.1 Dataset Structure

The dataset comprises Bangla sentences, each followed by its tokens with corresponding POS and NER tags. The dataset is stored in a tab-separated values (TSV) format, where each line contains a token, its POS tag, and its NER tag.

1.2 Preprocessing Steps

The preprocessing steps are crucial for preparing the raw text data for model training. The following steps were implemented in the data_preprocessing.py script:

- **Loading the Data:**
    - The TSV file is read, and sentences along with their respective POS and NER tags are extracted.
    - Sentences and their associated tags are stored in lists for further processing.
- **Tokenization:**
    - Three tokenizers (word_tokenizer, pos_tokenizer, and ner_tokenizer) are created using TensorFlow's Tokenizer. The word_tokenizer is used to convert sentences into sequences of integers, while the pos_tokenizer and ner_tokenizer do the same for POS and NER tags, respectively.
    - These tokenizers map each unique word/tag to a unique integer.
- **Padding Sequences:**
    - All sequences (words, POS tags, and NER tags) are padded to ensure uniform length across the dataset. The maximum sequence length (max_len) is determined based on the longest sentence in the dataset.

- **Saving Preprocessed Data:**
    - The preprocessed data, including tokenizers and max_len, are saved into a .npz file for easy access during model training. Additionally, the tokenizers and max_len are saved in a single JSON file (tokenizer.json) for use in the deployment phase.

1.3 Decisions Made

- **Tokenizer Configuration:**
    - The tokenizers are configured with lower=False to preserve the case of the text, which can be important for distinguishing proper nouns in NER tasks.
    - Padding is applied post-sequence to align with the natural reading order.

- Single JSON for Tokenizers:
    - Storing all tokenizers and max_len in a single JSON file simplifies the deployment process and ensures consistency between training and inference.

2. Model Development

2.1 Model Architecture

The neural network model was implemented using TensorFlow/Keras. The chosen architecture is a Bidirectional LSTM (Long Short-Term Memory) network, which is well-suited for sequence-based tasks like POS tagging and NER.

- **Embedding Layer:**
    - The first layer of the model is an Embedding layer, which converts integer sequences into dense vectors of fixed size (128 in this case). This layer helps in capturing the semantic meaning of words based on their context.

- **Bidirectional LSTM:**
    - A Bidirectional LSTM layer with 256 units follows the Embedding layer. The bidirectional nature allows the model to capture dependencies from both past and future contexts in the sentence, improving performance on sequence labeling tasks.

- **TimeDistributed Dense Layers:**
  - Two TimeDistributed Dense layers are used to produce the final POS and NER predictions. Each TimeDistributed layer applies a Dense layer to each time step of the LSTM's output, enabling the model to predict a tag for each word in the sequence.
  - Softmax activation is applied to obtain a probability distribution over the possible tags.

2.2 Training and Validation

- **Data Splitting:**
  - The dataset is split into training, validation, and test sets using an 80-10-10 split. This ensures that the model is trained on a majority of the data while being validated on a separate portion to avoid overfitting.

- **Model Compilation:**
  - The model is compiled with the Adam optimizer and categorical cross-entropy loss for both POS and NER outputs. Accuracy is used as the primary metric for evaluation during training.

- **Model Training:**
  - The model is trained for 10 epochs with a batch size of 64. The training history, including accuracy and loss for both POS and NER tasks, is recorded for analysis.

2.3 Decisions Made

- **Bidirectional LSTM:**
  - A Bidirectional LSTM was chosen for its effectiveness in capturing long-range dependencies in sequential data, which is crucial for both POS tagging and NER tasks.

- **Shared Architecture:**

    - A shared LSTM layer for both tasks helps the model leverage common features between POS and NER, potentially improving performance on both tasks.

3. Model Evaluation

3.1 Evaluation Metrics

The model is evaluated on the test set using the following metrics:

- **Accuracy:** Measures the percentage of correctly predicted tags.
- **Precision, Recall, F1-Score:** These metrics are calculated for both POS and NER tasks using a macro-average, which treats all classes equally regardless of their frequency.

3.2 Challenges Faced

- **Imbalanced Classes:**
    - One challenge was handling imbalanced classes, particularly for the NER task where some entity types are rare. The use of macro-averaging for precision, recall, and F1-score helped mitigate this issue.

- **Padding and Masking:**
    - Another challenge was ensuring that padding tokens did not affect the performance metrics. This was handled by masking the padding tokens before calculating metrics.

4. Model Deployment

4.1 FastAPI Inference API

A FastAPI-based inference API was developed to serve the trained model. The API provides two endpoints:

- **Health Check Endpoint:**
    - A simple /health endpoint that returns the health status of the API.

- **Prediction Endpoint:**
    - The /predict endpoint accepts a list of sentences and returns the predicted POS and NER tags.

4.2 Loading Preprocessed Data

The tokenizers and max_len are loaded from the tokenizer.json file during the API's initialization. This ensures that the preprocessing during inference is consistent with the preprocessing during training.

4.3 Inference Workflow

The input sentences are tokenized and padded, and the model makes predictions on the processed data. The predictions are then converted from index form back to their respective tags using the index_word mapping from the tokenizers.

4.4 Challenges Faced

- **Consistency Between Training and Inference:**
    - Ensuring that the tokenization and padding during inference matched the training setup was a key challenge. This was addressed by saving the tokenizers and max_len to a JSON file during training and loading them in the API.

5. Conclusion

This document outlines the development process for a Bangla POS and NER tagging system, from data preprocessing to model training and deployment. The use of a Bidirectional LSTM model, combined with careful preprocessing and deployment considerations, allowed for the creation of an effective and scalable tagging system. The FastAPI-based API ensures that the model can be easily integrated into real-world applications, providing accurate and efficient tagging of Bangla text.