Deep Learning Project Report

Title: Persian Sentiment Analysis on Snappfood Service Reviews
Prof: Hadi Farahani
Author: Sajjad Zangiabadi
Date: February 2024

Abstract:
This project focuses on sentiment analysis of Persian language customer reviews regarding the service provided by Snappfood, an online food delivery platform. Deep learning techniques are employed to classify the sentiment of the reviews into positive or negative categories. The approach involves preprocessing the text data, building a vocabulary, and training a recurrent neural network (RNN) model for sequence classification. The model achieves promising results in accurately predicting the sentiment of customer reviews.

1. Introduction:
The proliferation of online platforms such as Snappfood has led to an increasing volume of customer reviews, making sentiment analysis essential for understanding customer feedback. This project aims to analyze Persian language reviews of Snappfood's service to classify them as positive or negative sentiments using deep learning techniques.

2. Vocabulary Class:
2.1 Definition:
The Vocabulary class is designed to handle text preprocessing tasks such as tokenization, building vocabulary dictionaries, and converting text data into numerical representations. It provides methods to tokenize input text, build a vocabulary based on frequency thresholds, and convert text into sequences of numerical indices.

2.2 Functionality:

1. Tokenizer:
    - The tokenizer function breaks down input text into individual words or tokens. It employs regular expressions (regex) to identify and tokenize various elements within the text, such as words, numbers, punctuation marks, and special characters. The tokenizer also handles Persian and English text, ensuring robust tokenization across multiple languages.
2. Build Vocabulary:
    - The build_vocabulary method constructs a vocabulary based on the frequency of words in a list of input sentences. It calculates the frequency of each word, filters out infrequent words based on a frequency threshold, and selects the top words up to a

maximum vocabulary size. The vocabulary is represented by dictionaries mapping words to numerical indices (stoi) and vice versa (itos).

3. Numericalize:

    - The numericalize method converts input text into sequences of numerical indices based on the constructed vocabulary. It tokenizes the input text using the tokenizer function and then maps each token to its corresponding numerical index in the vocabulary. Unknown words are assigned a special '<UNK>' token.

2.3 Usage:

- Instantiate a Vocabulary object with specified frequency threshold and maximum vocabulary size.
- Call the build_vocabulary method with a list of sentences to construct the vocabulary.
- Use the numericalize method to convert text data into numerical representations suitable for input to neural networks.
- Optionally, retrieve the constructed vocabulary dictionaries using the get_vocabularies method for reference or further analysis.

2.4 Explanations:

Explanation of Tokenizer: The tokenizer function employs a series of regex patterns to tokenize the input text. Each regex pattern identifies a specific element within the text and applies a replacement to facilitate tokenization. For example:

- (r'([a-zA-Z]+)', r' \1 '): Identifies English alphabetic characters and adds spaces around them.
- (r'([\u0600-\u06FF]+)', r' \1 '): Identifies Persian alphabetic characters and adds spaces around them.
- (r'(?:[\d۰-۹]+(?:[.,/][\d۰-۹]+)?)', r' NUM '): Identifies numbers and replaces them with the token 'NUM'.
- (r'([؟؛،»«!"#$%&()+,-./:;<=>?@[\]^_{|}~\t\n])\1', r' \1 '): Identifies punctuation marks and special characters and adds spaces around them.
- (r'(\u200c)+', r' \1 '): Identifies zero-width non-joiner characters (used in Persian text) and adds spaces around them.

Explanation of Numericalize: The numericalize method first tokenizes the input text using the tokenizer function. It then iterates over each token and maps it to its corresponding numerical index in the vocabulary. If a token is not present in the vocabulary, it is replaced with the '<UNK>' token. The resulting sequence of numerical indices represents the input text in a format suitable for processing by machine learning models.

3. Dataset Preparation and Utilization

3.1 Custom Dataset Class

3.1.1 Definition:

The custom dataset class, named CustomDataset, is a PyTorch-compatible class designed to facilitate the integration of structured data, particularly from a Pandas DataFrame, into the deep learning pipeline.

3.1.2 Purpose:

The purpose of the CustomDataset class is to provide a standardized interface for accessing and processing dataset samples, ensuring compatibility with PyTorch's data loading utilities and enabling seamless integration with deep learning models.

3.1.3 Functionality:

The CustomDataset class encapsulates functionalities such as initializing the dataset with a Pandas DataFrame, retrieving the length of the dataset, and fetching individual samples. It allows optional transformations to be applied to samples, enhancing flexibility in data preprocessing.

3.2 Collate_batch Function.

3.2.1 Definition:

The collate_batch function is responsible for collating a batch of samples into padded sequences of numericalized text, along with their respective labels and lengths. It is integral for batch processing during model training and inference.

3.2.2 Purpose:

The primary purpose of the collate_batch function is to prepare batches of data that adhere to the input requirements of deep learning models. It ensures uniformity in batch size and facilitates efficient computation by padding sequences to equal lengths.

3.2.3 Functionality:

The collate_batch function takes a list of samples, where each sample is a dictionary containing text and label information and processes them into padded sequences of numericalized text. It handles padding and converts labels to the appropriate data type, preparing the batch for consumption by the deep learning model.

3.3 Usage

The utilization of the custom dataset class and collation function is straightforward and crucial for the seamless integration of data into the deep learning pipeline. The typical usage involves the following steps:

1. Dataset Initialization: Instantiate the CustomDataset class, passing the relevant Pandas DataFrame and optional parameters such as text and label column names.
2. Data Loading: Utilize PyTorch's data loading utilities (e.g., DataLoader) in conjunction with the custom dataset to load batches of data during model training or evaluation.
3. Batch Processing: Apply the collate_batch function to collate batches of data, ensuring uniformity in batch size and format.

4. Recurrent Neural Network (RNN)

4.1 Definition:

The provided Python class defines an RNN model tailored for sequence classification tasks. It employs an embedding layer, followed by a bidirectional Long Short-Term Memory (LSTM) layer and fully connected layers for classification. The model utilizes Rectified Linear Unit (ReLU) activation for feature transformation and Sigmoid activation for binary classification outputs.

4.2 Architecture:

The architecture of the RNN model comprises several key components:

1. Embedding Layer: The first layer in the network, responsible for converting input token indices into dense vectors of fixed size (embedding dimension).
2. LSTM Layer: The core recurrent layer of the model, consisting of bidirectional LSTM cells. The LSTM layer processes the embedded sequences to capture temporal dependencies and generate hidden states. The bidirectional nature allows the network to consider both past and future contexts.
3. Fully Connected (FC) Layers: The hidden states produced by the LSTM layer are flattened and passed through one or more fully connected layers. These layers perform nonlinear transformations to extract higher-level features from the sequential input.
4. ReLU Activation: Applied after the fully connected layers, ReLU activation introduces nonlinearities to the network, enabling it to learn complex mappings between input and output.
5. Sigmoid Activation: The final layer of the network, which applies the sigmoid function to produce binary classification probabilities. It squashes the output values between 0 and 1, representing the likelihood of a given input sequence belonging to the positive class (e.g., positive sentiment).

4.3 Functionality:

- Forward Pass: The forward method performs the forward pass of the RNN model. It takes input sequences (text) represented as token indices along with their lengths (lengths). These sequences are embedded, packed into padded sequences to handle variable lengths efficiently, and passed through the LSTM layer. The final hidden state(s) are extracted and passed through fully connected layers followed by ReLU activation. The output is then fed into a final fully connected layer with a sigmoid activation to obtain binary classification probabilities.

4.4 Usage:

- To utilize this RNN model for sequence classification tasks, one must first instantiate the RNN class, specifying the desired parameters such as vocabulary size, embedding dimension, RNN hidden state dimension, fully connected layer dimension, and optionally, the number of recurrent layers, bidirectionality, and dropout probability. Once instantiated, the model can be trained using appropriate data and loss functions, and then evaluated or used for making predictions on new sequences.

4.4 Component Analysis:

1. Embedding Layer:
    - Purpose: Converts input token indices into dense embedding vectors.
    - Functionality: Maps discrete token indices to continuous vectors in an embedding space, allowing the network to learn representations of words or tokens.
    - Usage: Essential for transforming discrete inputs into a continuous representation suitable for neural network processing.
2. LSTM Layer:
    - Purpose: Captures sequential dependencies in the input data.
    - Functionality: Processes input sequences and maintains hidden states across time steps, enabling the model to retain information over long sequences.
    - Properties: Bidirectionality allows the model to consider both past and future contexts simultaneously. Number of layers (num_layers) determines the depth of the LSTM stack.
    - Usage: Integral for modeling sequential data such as text, speech, or time series.
3. Fully Connected Layers (FC):
    - Purpose: Extract higher-level features from the LSTM hidden states.
    - Functionality: Applies linear transformations followed by nonlinear activations to the input data, enabling the model to learn complex mappings.
    - Usage: Facilitates the extraction of abstract features from sequential inputs, leading to improved classification performance.
4. ReLU Activation:
    - Purpose: Introduces nonlinearity into the network.
    - Functionality: Applies the rectified linear function elementwise to the input, effectively removing negative values and promoting sparsity.
    - Usage: Enables the model to learn nonlinear relationships between features, enhancing its capacity to model complex data.
5. Sigmoid Activation:
    - Purpose: Produces binary classification probabilities.
    - Functionality: Squashes the input values between 0 and 1, interpreting them as probabilities of belonging to a particular class.

  - Usage: Commonly employed in binary classification tasks, such as sentiment analysis, where the goal is to predict one of two classes (e.g., positive, or negative sentiment).

5. Training and Evaluation:
The provided train and evaluate functions are crucial components of your deep learning project, responsible for training and evaluating your RNN model, respectively. These functions interact with your data through a dataloader and utilize various components such as the model itself, optimizer, loss function, and evaluation metrics to optimize and assess the model's performance.

5.1 Train Function:

- Purpose: Trains the RNN model using the provided dataloader by iterating over batches of data, computing predictions, calculating loss, and updating model parameters.
- Components:
  1. Model Training: Sets the model to training mode (model.train()).
  2. Batch Iteration: Loops over batches of data from the dataloader.
  3. Forward Pass: Passes input sequences through the model to compute predictions.
  4. Loss Calculation: Computes the loss between the model predictions and ground truth labels using a specified loss function (criterion).
  5. Backpropagation: Propagates gradients backward and updates model parameters using the optimizer (optimizer).
  6. Metrics Tracking: Tracks total accuracy and loss for monitoring training progress.
- Usage: Call this function during the training phase of your deep learning project to iteratively update the model parameters using your training data.

5.2 Evaluate Function:

- Purpose: Evaluates the performance of the trained model on a separate validation dataset using the provided dataloader.
- Components:
  1. Model Evaluation: Sets the model to evaluation mode (model.eval()).
  2. Batch Iteration: Loops over batches of data from the validation dataloader.
  3. Forward Pass: Passes input sequences through the model to compute predictions without gradient calculation.
  4. Loss Calculation: Computes the loss between the model predictions and ground truth labels using the same loss function (criterion) as in the training phase.

5. Metrics Calculation: Computes evaluation metrics such as accuracy and F1 score for model performance assessment.

- Usage: Utilize this function after training to assess the generalization ability of your model on unseen data.

5.3 Components:

1. Optimizer:
    - Purpose: Optimizes the model parameters during training by adjusting them based on computed gradients.
    - Usage: Common optimizers include Adam, SGD, and RMSprop, which can be instantiated with specified learning rates and other hyperparameters.
2. Loss Function (Criterion):
    - Purpose: Computes the discrepancy between model predictions and ground truth labels, providing a measure of how well the model is performing.
    - Usage: Common loss functions for binary classification tasks include Binary Cross-Entropy Loss and Hinge Loss, which penalize incorrect predictions differently based on their confidence levels.
3. Evaluation Metrics (Accuracy, F1 Score):
    - Accuracy: Measures the proportion of correctly classified samples out of the total number of samples.
    - F1 Score: Harmonic mean of precision and recall, providing a balanced measure of the model's performance in binary classification tasks.
    - Usage: These metrics help quantify the model's performance during both training and evaluation phases, enabling comparison and decision-making regarding model selection and hyperparameter tuning.

6. Results:
We evaluate the performance of the model based on various metrics including accuracy, loss, and F1-score. Additionally, we utilize visualizations such as learning curves to provide a comprehensive overview of the model's performance over training epochs.

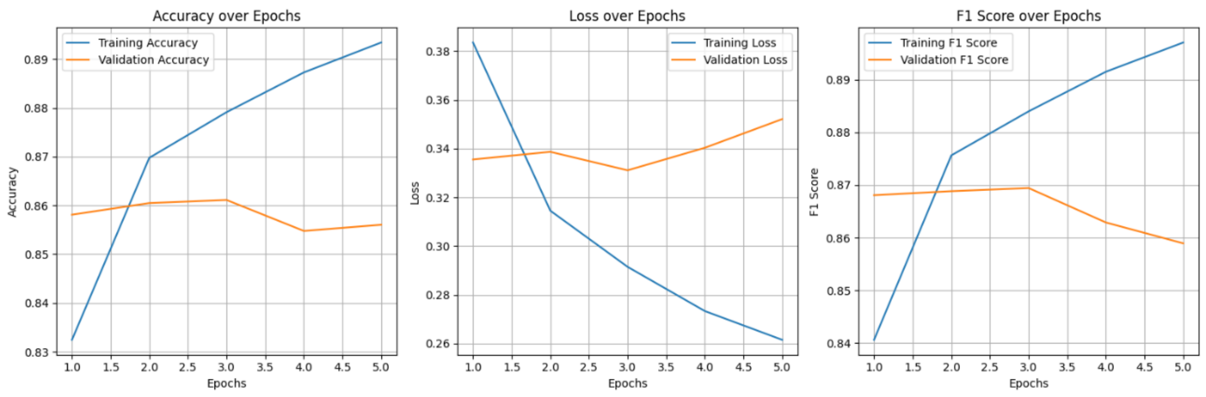
6.1 Performance Metrics:

1. Accuracy: The proportion of correctly classified samples out of the total number of samples.

2. Loss: The discrepancy between model predictions and ground truth labels, providing a measure of how well the model is performing.

3. F1-score: The harmonic mean of precision and recall, offering a balanced measure of the model's performance in binary classification tasks.

6.2 Visualizations:
We include learning curves to illustrate the changes in model performance metrics (accuracy, loss, and F1-score) over training epochs. These visualizations allow us to track the progress of the model's training process and identify any trends or patterns in its performance.



7. Discussion:
In the discussion section, we dive into understanding the performance of our RNN model for Persian sentiment analysis in more detail.

7.1 Interpretation of Results:

- We analyze the trends depicted in the learning curves, focusing on how the model's accuracy, loss, and F1-score change over training epochs. Understanding these trends helps us gauge the model's learning progress and its ability to capture the complexities of sentiment in Persian text.

7.2 Challenges and Limitations:
Throughout our project, we encountered several challenges that influenced the performance and interpretation of our RNN model for Persian sentiment analysis. These challenges highlight the complexity and nuances inherent in analyzing sentiment in informal Persian text.

1. Mixed Language Content:

    - One significant challenge was the presence of mixed language content within the text data. We observed instances where English words were interspersed with Persian text, as well as non-standard Persian spellings, such as elongated letters (e.g., "خوب" written as "khoooooooob"). This heterogeneity in language usage posed a challenge for the model, as it needed to accurately interpret and classify sentiment across multiple languages and unconventional spellings.

2. Informal Language Style:
    - Another challenge stemmed from the informal nature of the text data. The language used in the text was highly informal, characterized by colloquialisms, slang, and unconventional expressions. This informal style of communication posed a challenge for the model, as it needed to understand and classify sentiment in text written in a manner that deviated from formal language conventions. People tend to write informally as they like, which further complicates the task of sentiment analysis.
3. Lexical Variation and Ambiguity:
    - The informal nature of the text also contributed to lexical variation and ambiguity. Words and phrases may have multiple meanings or interpretations in informal contexts, making sentiment analysis more challenging. Additionally, slang terms and expressions may not have standardized definitions, further complicating the task of accurately capturing sentiment.
4. Data Quality and Annotation:
    - Ensuring the quality and consistency of the annotated data posed another challenge. Annotating sentiment in informal text can be subjective and prone to errors, leading to inconsistencies in the labeled data. Addressing these challenges required careful data preprocessing and annotation strategies to mitigate biases and ensure the reliability of the training data.

7.3 Implications for Real-World Applications:

- We discuss how accurately classifying sentiment in Persian text can benefit platforms like Snappfood, enabling them to better understand customer feedback and preferences. This understanding can lead to improved user experiences and more effective business strategies tailored to the needs of Persian-speaking users.

This report provides a comprehensive overview of the deep learning project on Persian sentiment analysis, outlining methodologies, results, and discussions to facilitate further understanding and application in real-world scenarios.