CP322 Machine Learning - Assignment 3
Due Date: Dec 4, 2024 at 11:59 PM

About Submission

When writing and submitting your assignments, follow these requirements:

- You are expected to submit a single Notebook file for this assignment.
- Extensions for assignment are only granted for medical reasons with a doctor's note. Assignments submitted within 48 hours after the deadline will have their grade reduced by 50%. Submissions beyond 48 hours post-deadline cannot be accepted and will receive a grade of 0.
- Your assignment should be submitted online through the MyLearningSpace website. Email submission is not accepted.
- Please document your program carefully.
- This assignment consists of two parts:
  - Part 1: Worth 15 points, which contributes directly to your assignment grade.
  - Part 2: Worth 5 bonus points, which can be added to your score if you complete it correctly. The maximum score you can achieve for this assignment is 15 points, including bonus points. For example, if you lose 5 points in Part 1 due to deductions but complete Part 2 correctly, you can still receive the full 15 points. This allows you to offset deductions in Part 1 by excelling in Part 2.

  Please ensure you review each part carefully to maximize your score!

Objective and Introduction

Sentiment classification is a fundamental task in natural language processing (NLP), where the goal is to determine the sentiment expressed in a piece of text. For this assignment, the task is to classify movie reviews from the IMDB dataset into positive or negative sentiments.

The IMDB dataset is a widely used benchmark for sentiment analysis tasks. It consists of 50,000 movie reviews, with each review labeled as either positive or negative. The dataset is divided equally into 25,000 training and 25,000 testing samples, providing a balanced distribution of sentiments. This dataset is ideal for building and evaluating sentiment classification models due to its size and relevance. Please download this dataset from Kaggle at `https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews`.

The objective of this assignment is to provide hands-on experience with two popular machine learning frameworks: PyTorch for implementing a Multi-Layer Perceptron (MLP) and Hugging Face's Transformers library for using a pre-trained BERT model. The assignment focuses on performing sentiment analysis on movie reviews using the IMDB dataset, aiming to classify reviews as positive or negative. By completing this assignment, you will gain practical skills in data preprocessing, model implementation, and evaluation.

1 Implementing Multi-Layer Perceptron using PyTorch: 15 points

In this part, you will: 1) prepare the IMDB dataset for training an MLP model; 2) define and implement the MLP model architecture using PyTorch; 3) train the MLP model and validate its performance; evaluate the trained model on the test data and visualize the training process.

Step-by-Step Implementation

1. Prepare the IMDB dataset for training the MLP model by preprocessing the text data and splitting it into training, validation, and test sets.

   (a) Load the IMDB Dataset: use pandas to load the IMDB dataset CSV file into a DataFrame.

   (b) Preprocess the Text Data: use 'TfidfVectorizer' from scikit-learn to convert the text reviews into TF-IDF features. Limit the number of features to a manageable number (e.g., 5000).

   (c) Encode the Labels: convert the sentiment labels ('positive' and 'negative') into numerical values using 'LabelEncoder' from scikit-learn.

   (d) Split the Data: split the dataset into training, validation, and test sets using 'train_test_split' from scikit-learn. Use 60% of the data for training, 20% for validation, and 20% for testing.

   (e) Create Dataset Class: define a custom 'Dataset' class for PyTorch to handle the data. This class should return the features and labels as tensors.

   (f) Create DataLoaders: use 'DataLoader' from PyTorch to create iterators for the training, validation, and test sets.

   At the end of this step, you should have three DataLoaders (train_loader, val_loader, and test_loader) ready for training, validation, and testing the MLP model.

2. Define the architecture of the MLP model using PyTorch.

   (a) Define the MLP Class: create a class MLP that inherits from nn.Module. Define the network layers in the constructor. For this assignment, you have the flexibility to design your MLP with a single hidden layer and use ReLU as the activation function.

   (b) Initialize the Model: specify the input size (number of features), hidden layer size (e.g., 512), and number of classes (2 for binary classification). Create an instance of the MLP model.

By the end of this step, you should have an MLP model defined and ready to be trained.

3. Train the MLP model using the training data and validate its performance on the validation data.

   (a) Define Loss and Optimizer: use 'CrossEntropyLoss' as the loss function and 'Adam' as the optimizer.

   (b) Initialize Lists for Storing Metrics: create lists to store training and validation losses and accuracies for each epoch.

   (c) Training Loop: implement the training loop to train the model for a specified number of epochs (e.g., 10). For each epoch:

      i. Set the model to training mode.
      ii. Iterate over batches of data from the training DataLoader.
      iii. Perform a forward pass, compute the loss, perform a backward pass, and update the weights.
      iv. Track the running loss and accuracy for each batch.

   (d) Validation Loop: after each epoch, evaluate the model on the validation data.

      i. Set the model to evaluation mode.
      ii. Iterate over batches of data from the validation DataLoader.
      iii. Perform a forward pass and compute the loss.
      iv. Track the running loss and accuracy for each batch.

   At the end of this step, you should have a trained MLP model and recorded training/validation losses and accuracies for each epoch.

4. Evaluate the trained MLP model on the test data and visualize the training process.

   (a) Evaluate on Test Set:

      i. Set the model to evaluation mode.
      ii. Iterate over batches of data from the test DataLoader.
      iii. Perform a forward pass and compute predictions.
      iv. Track the accuracy and generate a classification report.

   (b) Plot Training and Validation Losses/Accuracies: use matplotlib to visualize the training and validation losses and accuracies over epochs.

   By the end of this step, you should have evaluated the model's performance on the test set and visualized the training process through loss and accuracy plots.

2 Using Pre-trained BERT for Sentiment Analysis with Hugging Face: 5 bonus points

Let's use the 'nlptown/bert-base-multilingual-uncased-sentiment' model directly with Hugging Face's pipeline. This model is designed to work out of the box without needing fine-tuning.

Step-by-Step Implementation

1. Data Preparation:
   (a) The dataset is loaded, and labels are encoded using ‘LabelEncoder’.
   (b) The data is split into training, validation, and test sets using train_test_split.
2. Pipeline Usage:
   (a) The ‘pipeline’ API from Hugging Face is used to initialize a sentiment analysis model.
   (b) The pre-trained model ‘nlptown/bert-base-multilingual-uncased-sentiment’ is used directly without further fine-tuning.
3. Text Truncation:
   (a) The truncate_texts function ensures that no input text exceeds the maximum length of 512 tokens. This avoids the error due to exceeding the maximum sequence length.
   (b) The validation and test texts are truncated accordingly.
4. Prediction Conversion:
   (a) The convert_sentiment_label function converts the sentiment labels from the pipeline to binary labels (0 for negative, 1 for positive).
   (b) Convert the sentiment labels to binary values: if the label is ’4 stars’ or ’5 stars’, change it to ‘1‘ (positive sentiment). For all other labels, change it to ‘0‘ (negative sentiment).
5. Evaluation:
   (a) The accuracy and classification report are calculated for both the validation and test sets to evaluate the model’s performance.