Assignment 2. Product Category Classification 2

DATA304: Big Data Analysis

Due Date: October 20, 2025

General Instructions

The programming assignments in this course are designed to foster your ability to implement and apply the concepts introduced during class. All datasets and resources for the assignments are provided at the following link: **Link**. You do not need to download new data for each assignment. Instead, simply place the newly provided Jupyter notebook files for each assignment into your existing `assignment_release` directory (or into the renamed directory, if you have changed it).

- **How to submit:**
  - You need to submit your code via LMS for all assignments. For each assignment, submit **All** the released `.ipynb` files, including all modifications you have made.
  - Assignments #1, #2, and #3 also require Kaggle submissions.
  - Assignments #4 and #5 do not require Kaggle submissions.

  Make sure you submit both LMS and Kaggle on time to receive the credit. All submissions are due by midnight (23:59 KST) on the deadline day.

- **How the assignments are graded:**
  - For Assignments #1, #2, and #3, you will receive full credit as long as you outperform our baseline code.
  - For Assignments #4 and #5, you will receive full credit as long as you submit your code on time.

  This policy is intended to encourage you to solve the problem by yourself and to compare your results with others. Use the Q&A board on LMS for all assignment-related questions.

Assignment Instruction

This assignment consists of four Jupyter notebooks: `P2a.guide-to-using-BERT.ipynb`, `P2b.train-with-BERT-embedding.ipynb`, `P2c.train-with-BERT-embedding-limited-label.ipynb`, and `P2d.fine-tuning-BERT (reference).ipynb`.

P2a: Guide to using BERT

We explore the fundamental ways of using BERT (Bidirectional Encoder Representations from Transformers). The goal is to understand how text is processed by BERT, how outputs are represented, and how to use BERT embeddings for downstream tasks. The exercise is divided into three parts:

1. [**Part A**]: Understanding BERT input
2. [**Part B**]: Understanding BERT output
3. [**Part C**]: Using BERT mean-pooled embeddings

In this assignment, we mainly use fixed BERT mean-pooled embeddings for efficiency (i.e., partial fine-tuning with fixed encoder). In Part C, you will learn how to extract and save these embeddings for your assignment.

⌨ **Your Task:** Carefully read and understand the provided code. Examine intermediate values by printing them out, and make sure you clearly understand what operations are being performed at each step and how they fit together in the overall process. No submission required.

P2b: Train with BERT embedding

We perform product category classification using precomputed BERT mean-pooled embeddings. Unlike the previous assignment where we used TF-IDF features as input to a linear classifier, here we replace TF-IDF with semantic embeddings extracted from BERT. The exercise is divided into two parts:

1. [**Part A**]: Training with default labeled set
2. [**Part B**]: Training with limited labeled set

⌨ **Your Task:** Carefully read and understand the provided code. Examine intermediate values by printing them out, and make sure you clearly understand what operations are being performed at each step and how they fit together in the overall process. No submission required.

P2c: Train with BERT embedding (limited labels)

We explore methods to improve classification performance in limited label scenarios. Here, only a small subset of dataset is labeled. This mimics realistic situations in industry where annotations are expensive and only partially available. We will attempt two approaches to handle this challenge:

1. [**Part A**]: Label Embedding-based classifier
2. [**Part B**]: Self-Training (Pseudo-Labeling)

[Part A]: Label Embedding-based classifier

Instead of treating category labels as discrete IDs, we represent each class label with its **semantic embedding** (i.e., using BERT embeddings of category names). This allows the model to leverage semantic similarity between categories (e.g., *"knitting hooks"* and *"crochet hooks"* are closer than *"knitting hooks"* and *"laptops"*). This approach can improve generalization, especially when labeled data is sparse.

⌨ **Your Task:** Carefully read and understand the provided code. Examine intermediate values by printing them out, and make sure you clearly understand what operations are being performed at each step and how they fit together in the overall process. No submission required.

[Part B]: Self-Training (Pseudo-Labeling)

Here, we extend supervised training with self-training: The model first learns on the labeled training set. Then, it generates predictions for unlabeled data. If the model is confident enough (probability above a threshold), those predictions are added as pseudo-labels and used for further training. This approach can effectively utilize the large pool of unlabeled data to boost classification accuracy.

Refer to the following algorithm that we studied in class.

Algorithm 1 Self-training with Pseudo-Labeling

```
1: Input: Labeled dataset $\mathcal{D}_l$, Unlabeled dataset $\mathcal{D}_u$, Confidence threshold $\tau$
2: Output: Trained model $f$
3: Train base model $f$ on $\mathcal{D}_l$
4: while not converged do
5:     Predict pseudo-labels for samples in $\mathcal{D}_u$ using $f$
6:     Select high-confidence predictions: $\mathcal{D}_p = \{(x, \hat{y}) \mid \max f(x) \geq \tau\}$
7:     Update training set: $\mathcal{D}_l \leftarrow \mathcal{D}_l \cup \mathcal{D}_p$
8:     Update $f$ on the expanded $\mathcal{D}_l$
9: end while
10: return $f$
```

⌨ **Your Task:** Now, your task is to implement training loop with self-training, and submit your best results to Kaggle.

Tip 1: `torch.utils.data.ConcatDataset` may be helpful for combining datasets.

Tip 2: The details can be adjusted flexibly. For example, instead of using a fixed threshold, you may gradually increase it during training. In addition, you may periodically re-check the pseudo-labels that have already been added.

P2d: Fine-tuning BERT

We use BERT to obtain contextual representations of product texts, taking the [CLS] token embedding as a compact summary of the input. On top of these embeddings, a simple linear classifier is trained to predict product categories. In this part, rather than relying on pre-extracted fixed BERT embeddings, each text is encoded on-the-fly. You will also explore how to control the degree of fine-tuning (from full fine-tuning to partial fine-tuning). This section is provided mainly for reference, so focus on understanding the workflow rather than executing every single cell.

Submission Guidelines

Kaggle Submission

- Submit your prediction file in `.csv` format to Kaggle: **Link**
- Ensure that the filename of your submission follows the format: `studentID_assignment1.csv`
- Set your **Kaggle team name** to your AWS account ID (e.g., `korea-dt-xx`). **This is mandatory in order to receive credit in our grading system. Please be careful!**
- There is **no limitation on the number of Kaggle submissions**.

LMS Submission

- Submit your source code (all `.ipynb` files) to the LMS.
- Make sure the submitted code can be executed without errors.