Navigating Modern Text Classification: A Conceptual, Computational, and Empirical Guide to Pre-trained Large Language Models

Martin Juan José Bucher
Marco Martini

Step-by-step documentation

Note: This document is intended as a guide to our text classification pipeline as contained in the folder '02-pipeline'. To allow users to train models based on their own input data, our pipeline has a slightly different setup than the pipeline containing the reproduction materials for the paper, which are contained in the folder '01-reproduction-material'. However, both pipelines are closely related. Consequently, the following steps should allow users to run both the custom pipeline as well as our replication code.

Training a Language Model for Your Classification Task

To conveniently use the fine-tuning pipeline presented in our paper, we provide our code as a Jupyter Notebook file called ***pipeline-finetuning.ipynb***.

If you do not have a machine with a GPU, we recommend setting up the pipeline on a Google Colab instance. This speeds up training significantly compared to a machine without a dedicated GPU. There are free options as well as possibilities to upgrade on Memory and compute power. We recommend testing the pipeline locally first. If cell F starts training successfully but runs for hours even with small datasets (~ 1k observations), setting up Google Colab is likely going to be worthwhile.

In case of any questions regarding the pipeline, please refer to the Issue Tracker under our Github Repository [to be released upon publication] and create a new Issue.

Preparation

To be able to run the pipeline:

i) the annotated dataset should be provided in the right format (see Step D below as well as the instructions at the beginning of the ***pipeline-finetuning.ipynb*** notebook) as two separate CSV files (one for the labels and one for the text samples).
ii) you need to have Jupyter Notebook running on your local machine (or use Google Colab).

Installing Jupyter Notebook for Windows

On a Windows machine, the easiest way to get the notebook is to install the Anaconda library, an open-source software containing Jupyter, Python, and other relevant packages (https://www.anaconda.com). There are plenty of tutorials online on how to install and launch Jupyter Notebook on Windows.

Installing Jupyter Notebook for macOS

Install Jupyter through the pip package manager: "pip install jupyter"

Start Jupyter as follows: “jupyter lab”

This will open Jupyter under the *patah* in which you ran the command. To simplify the browsing of the files, navigate to the folder location of this source code and run the above command at that location.

It might be helpful to create a virtual environment using “venv” or “conda” to avoid conflicting package dependencies with other python projects.

The notebook

The following sections describe the individual cells in the ***pipeline-finetuning.ipynb*** notebook.

After installing Jupyter Notebook, launch the ***pipeline-finetuning.ipynb*** file. Code cells in Jupyter Notebook can be executed via Menu > Run > Run Selected Cell (or simply with Shift + Enter).

A:

The first code block installs and imports required packages. You may need to install some additional packages (depending on your environment).

B:

Next, define required parameters. In particular, set the following variables:

- “PROJECT_NAME”: This should be a unique name describing your project.
- “DATASET”: The name of the subfolder where your dataset is stored inside the “data” folder. If the dataset is named “elections”, then a folder “elections” should exist within the “data” folder. Inside this folder, place your CSVs (see Step D below).
- “LANGUAGE_FOR_MODEL”: Select the language of your dataset/corpus.
- “LANGUAGE_MODEL”: Select the LLM you want to fine-tune.

C (Optional):

This step allows setting optional parameters that require some knowledge on how to optimize deep neural networks. Skip this section to go with the default parameters. As shown in our paper, the default parameters yield strong performance.

IMPORTANT: Even if using the default parameters, you need to run this cell to define all variables.

D:

Import the dataset as two different CSVs.

- The first CSV file contains all sentences (i.e. the text data), where each row contains one sample (e.g. a tweet). A row does not necessarily need to contain a single sentence, but should not be overly long as the language models have a maximum capacity of reading 512 tokens (1 token is not exactly 1 word).
- The second CSV file contains the labels, where each row contains one integer number corresponding to the class of the texts in the first file (the label in row 1 in the second file must indicate the class of the text in row 1 of the first file, and so on). Thus, the order of the rows from the first file and the second file should match exactly and the files should have the same number of rows.

E (Optional):

It is possible to provide an additional logging instance using the `Weights & Biases` platform. To do so, you need to create an account and a corresponding project and link the notebook with your API key.

Having an external logging platform makes it possible to track and compare different experiments across time and to compare the performance of different hyperparameter configurations directly in the Cloud. If the logging function is turned off, it will simply print out the metrics directly into the Jupyter Notebook. Once the notebook is closed, these results are gone.

F:

This code block implements the fine-tuning of the model using the provided training data.

G:

Running this code block, will auto-label your data based on the fine-tuned model. The output is stored in the data folder under your project name.

It is possible to store the model weights for later usage. A folder `ckpt` will be created and within this folder another folder with the name of your RUN_ID. Inside the latter folder, will be a file with the ending `....pth.tar`. These are the model weights. Downloading this file with the corresponding folder hierarchy (i.e. `ckpt/YOUR_RUN_ID/ckpt_last_seed_1234.pth.tar`), makes it possible to re-use this model.

Additional Instructions to Run the Pipeline on Google Colab with GPU Support

Go to `www.colab.research.google.com`, login with your Google credentials and create a new notebook.

Next, go to the menu and click "Runtime" > "Change Runtime Type". Under "Hardware Accelerator", which defaults to "None" (or "CPU"), choose one of the fields with "GPU" in the title (names may differ). This should successfully connect to one of Google's GPUs.

Upload your dataset(s). This can be done in several ways.

The easiest way is to simply mount your google drive. This way, you can directly load the entire folder structure into your google drive and have the script store the results there. This this end, upload the entire "02-pipeline" folder into Google drive. Then open ***pipeline-finetuning.ipynb*** by right-clicking and selecting "Open with … Google Colab".

Then, to mount the Google drive, insert the following code at the beginning of your notebook (and run it before running the rest of the notebook):

- from google.colab import drive
- drive.mount('/content/drive')

Then navigate to the main folder:

- %cd /content/drive/MyDrive/…/02-pipeline

Verify the folder content with:

- !ls

Lastly, execute the remaining cells of the ***pipeline-finetuning.ipynb*** in the same way as via Jupyter.

Step F should now run noticeably faster.