University of Zurich[UZH]

uyWhisper

Finetuning Whisper on Portunhol Data

Submission date: July 26th, 2024

by

Dominik Martínez
dominik.martinez@uzh.ch

Report submitted to

Noëmi Aepli

as the examination basis for the course

Programming Project

Department of Computational Linguistics
University of Zurich

Contents

1 Introduction

Modern large-scale Automatic Speech Recognition (ASR) models like OpenAI's Whisper perform astoundingly well on audios of different quality, even on input in unstandardised languages they have not been trained on. Whisper, for example, has not many problems in understanding Portunhol [1] data. However, the models' outputs usually follow the standardised languages they have seen during training. In the case of Whisper and Portunhol, it is not unusual to see output that has not been transcribed but translated to one of both languages. A simple example is the Spanish-based word *pero* 'but' being translated to Portuguese *mas*, although the audio data clearly contains *pero*.

The goal of this programming project was, therefore, the fine-tuning of an existing Whisper model on Portunhol data. The data to be used proceeds from Bárbara Garrido Sánchez' doctoral project. During her data collection phase, she recorded both interviews and dialogues of Northern Uruguayan Portunhol speakers, which she later transcribed manually. Her corpus is, although not the biggest, highly accurate and would serve as high quality finetuning data for this project.

Finetuning a Whisper model can be considered feasible without any low-level programming effort, and there are even step-by-step tutorials[2]. The most crucial challenge during the project was, nevertheless, the fact that the data was unaligned. This would have been an important feature of the dataset, as Whisper is trained on data of a maximum length of thirty seconds, and any data to be used needs to be split into chunks of this length. I intended to use Timething both for aligning and splitting the data. Unfortunately, and despite being able to run the tool on our data, I was not able to obtain splits of the audio data with their corresponding transcription. Given the time budget of the programming project and many hours wasted for installation and debugging, I was not able to test other tools and finetune a model.

Therefore, this report shall serve as a recapitulation of the current state and, more importantly, as inspiration for future attempts.

[1] Portunhol or Portuñol denominates various language varieties between Portuguese and Spanish.

[2] A model for this programming project was the following, well-explained tutorial: `https://huggingface.co/blog/fine-tune-whisper`.

2 Data

The data was collected in Artigas, the northernmost department of Uruguay, with the goal of investigating the local variety. Given its geographic proximity to Brazil (Artigas is the only Uruguayan department bordering two countries, namely, Argentina and Brazil), the local dialect is heavily influenced by Brazilian Portuguese. The Artigas variety exhibits features of both languages and can therefore be characterised as Portunhol. Common denominations are *Portuñol riverense*, *Portuñol fronterizo*, and *Bayano*, while the official designation is *Dialectos Portugueses del Uruguay* (DPU, Portuguese dialects of Uruguay).

The audio data was collected by Bárbara Garrido in line with her doctoral thesis. Audio recording was performed in two different settings: In *interviews*, the conversation is between an interviewer – Bárbara Garrido – and a Portunhol speaker. The utterances by the interviewer are mainly in Portuguese or Spanish, with a certain influence of the two languages on each other, but not in the local variety. In contrast, *dialogues* are conversations between two Portunhol speakers. The corpus consists of 31 interviews and 16 dialogues of varying length (between 30 and 90 minutes).

The parallel textual data are transcriptions of the audio data by Bárbara Garrido. As it was not necessary for her doctoral thesis to transcribe the audio files in their entirety, she selected excerpts of around 30-45 minutes. Therefore, only a subpart of the corpus can be used as parallel data for model training. The transcriptions were further enriched by language annotations: Each word was tagged either as *Spanish*, *Portuguese*, or *hybrid* (with the hybrid label being removed further into the process) . The language annotations are not relevant for model training and removed during preprocessing. However, to gain knowledge about the representation of the two labels per speaker, a script counts each label and outputs a table with the absolute and relative quantity of each label (*quantify_variants.py*, see README).

3 Preprocessing

This section describes the main effort of the entire project. All code described in this and further sections is available on GitHub.

3.1 Audio preprocessing

As for the audio files, a first step included the download from a shared folder and upload to the server. A second step required by the nature of the data was cutting the audio files such that the split to be used only include transcribed parts. For this step, I leveraged the time stamps contained in the meta header of each text file. Audio cutting is carried out by a script called *cut_audios.py*, which processes pairs of audio and text files sequentally. For each file pair, the time window is extracted and compared against the audio file. If needed (some of the shorter files were transcribed in their entirety), the script calls FFmpeg, which cuts the audio file and places the output in an output directory. If the time window in the meta data is consistent with the audio length, the file remains unchanged and is copied to the output directory.

3.2 Text preprocessing

In a first step, the raw textual data needed to be adapted manually. This included downloading them from a shared folder, ensuring consistent file naming, getting familiar with a few special cases (e.g., an interview recorded in two separate audio files but transcribed in one single text file), and uploading the files to the server.

The remaining steps of text preprocessing are handled by the global script *preprocess_text.sh*. The script handles some steps itself, while it calls Python scripts for others. This will be indicated in each of the following paragraphs.

A second step involved encoding conversion. Because the transcriptions where created on Windows and without explicit UTF encoding, the text files are encoded with CP-1252. In a first try, I used the UNIX utility *iconv*. However, the resulting UTF-8 files showed unexpected behaviour with regards to newlines, which heavily compromised the preparation of the

data for alignment. Therefore, I wrote a basic Python script called *convert_cp1252_to_utf8.py* to carry out the conversion. This Python script is called by the global text preprocessing bash script.

The third step of text preparation was to select the actual data from the text files, i.e., to remove non-transcription data. This rather easy step only removes the meta header (containing names and time stamps) and the first of two columns, which contains a speaker ID. Both operations are executed directly in the global bash script: the first by *sed*, and the second by *cut*.

In a fourth step, the language labels and meta comments contained in the raw data need to be removed. Both steps are resolved by the deletion of matches of a certain regular expression. The first expression catches everything contained in parentheses and square brackets, including the symbols themselves. By doing so, meta comments like '(risa)' and '[dirigiéndose a otra persona]' are removed. The second expression matches the symbols used as language labels ('{}' for Portuguese, '<>' for Spanish, and '//' for hybrid where labels still were present in the data) but not their content. Regex matching and deletion is contained in *clean_meta.py*, which is called by the global script.

3.3 Alignment and splitting

As Whisper expects fragments of maximum 30 seconds for its training (and finetuning), the data needs to be preprocessed accordingly. There are multiple tools to align audio and textual data, even for many languages (of which Portuguese and Spanish are crucial). Nevertheless, Timething stood out for its ability to directly cut the audio in accordance with its own alignment.

Timething knows two types of data: Long form and short form, depending on the audio length. For alignment, there exist two different types of data input. For long form alignment, which is adequate for our data, Timething expects the audio and text files and outputs a JSON file with the alignment information. For short form alignment, the transcriptions are expected in one single file, distributed onto one line per partial transcription. After aligning, Timething provides a second tool that recuts the audio files and transcriptions such that, on the one hand, no audio file exceeds a length threshold defined by the user, and, on the other hand, the new short transcriptions

correspond to the audio files. According to Timething's README, this cutting tool expects the same input as the short form alignment. Our code therefore provides both text formats: For long form alignment, the global script writes the entire transcription onto one single line of a text file. For audio cutting, the global script prepends the path to the audio file to the transcription, resulting in a CSV file with the pipe symbol as separator and the audio path as well as the transcription as the two only columns.

After having prepared the data for alignment and cutting by Timething, *run_timething.sh* calls the alignment tool. For it to work, I had to adapt a few details in the Timething codebase. Specifically, in *dataset.py*, I needed to explicitly convert two Path objects (*record.file* and *filename*) to a string (to prevent an error from stopping alignment), and in *utils.py*, I had to remove the call of the deprecated function *torch.cuda.is_built()*. Furthermore, in *cli.py*, I implemented the ability to read in different encodings (however, with the conversion to UTF-8 in the preprocessing, this change became obsolete). Lastly, I needed to change the *batch_size* argument (which I specify in *run_timething.sh*) to 1. If I had used a batch size of 10 (as proposed by the README), I would have had to deal with errors regarding tensor shapes. With these changes, the alignment tool ran fine and produced a sensible output in JSON format.

With the alignment information stored to a JSON file and in order to finetune the Whisper model, the Timething cutting tool needed to be run. Again, the codebase required adjustments in order to run without any errors. Specifically, the CSV input (i.e., the transcription contained in it) seemed to be too large for the standard engine of *pandas.read_csv*. I therefore had to set the *engine* argument to *Python*. With this change, the cutting tool ran through without any errors. Unfortunately, it did not produce the expected cut files. Instead, the output was simply the uncut input audio file and the uncut transcription. Different tests (e.g., changing the *cut-threshold-seconds* argument to the README value 8.0 or, just to be sure, the value of 0.5 minutes) did not change the outcome. With this setback (cutting the data with Timething now seemed hardly possible) and the time investment being already high, I decided to use the time allocated for documentation to finish the report and write some minimal documentation.

4 Finetuning

After preprocessing, the data should have been ready for finetuning. However, as explained before, splitting the audio into 30 second chunks could not be achieved successfully. Still, I had set up a script (*finetune.py*, called by *run_finetuning.sh*) to finetune the Whisper model. The script is in an unfinished state and contains only argument parsing, data loading, and dataset creation. The latter two are implemented as suggested in the blog post by Sanchit Ghandi and the corresponding Jupiter notebook. With the properly preprocessed data at hand, this script could be further written and adapted to our project.

5 Challenges

The challenges of this project can be grouped into two main groups: On the one hand, the installation of some tools took unexpectedly long. On the other hand, working with Timething proved difficult and did not yield the splitted audio required for training.

One challenging installation was the one of FFmpeg. For preprocessing, I had worked on a local machine and everything worked fine. However, the Timething alignment tool needed to be run on a GPU for performance reasons, and also required FFmpeg. Installing the audio-processing software on the server was not straightforward and, in hindsight, I should have asked for help earlier (see section 7). In the end, I managed to download and install a static build. Another very challenging (and time-consuming) installation was the one of Timething. Some dependencies could not be installed automatically, while others led to version conflicts. In the end, I installed all required dependencies separately and specifically had to exclude *tokenizers* from the library's config file. To make the installation less of a hassle, we provide a requirements file with a working combination of dependency versions. Timething itself, after I had run into issues, needed to be reinstalled in an editable version such that I could amend the code (see section 3.3).

The second large challenge was related to Timething itself. While the software promised to be able to align text to audio and cut both types of data accordingly, only the former seems to work without any issues. As explained in section 3.3, the cutting tool did not cut neither audio nor transcription, and instead simply returned the input. With this crucial step not working and the time budget being used up, the project had to be dismissed, at least for the moment.

6 Future Work

As a first step, if no obvious error in the handling of Timething can be found, I would advise to use other tools for alignment and cutting. Timething would have been an optimal choice as it combines both steps in one library. However, there seem to be other promising tools like aeneas and Montreal Forced Aligner.

Next, the original idea of this project could be tested: If it is possible to finetune a Whisper model on our Portunhol data and how this would improve performance on Portunhol data. First, a vanilla finetuning could be tested, and then some other techniques used for low-resourced data could be applied, such as artificial noise by random or linguistically motivated letter permutations or random and linguistically motivated data augmentation.

As an extension, more Portunhol data could be collected and integrated. This additional data must not necessarily originate in Northern Uruguay but could instead be collected from other Portunhol regions (such as border regions between Brazil and Spanish-speaking countries or the Iberian Peninsula). A finetuned model might profit from the different characteristics of each variety, and rather stable (i.e., true to the audio input) transcriptions could be generated.

7 Learnings

The personal learnings I gained during this programming project are twofold: Technical learnings include the usage and practice of UNIX commands, including their detailed behaviour, and the practice of debugging in a real-world scenario. Also, especially during earlier stages of the project, I gained a deeper understanding of Whisper's working. The second group of learnings includes those related to the project itself: First, I sharpened my project planning skills but there is still much room for improvement (especially with respect to time planning, as I underestimated the time consumed by almost any step). Second, as the rather poor commit history of the repository shows and as I realised during the documentation / reporting phase, I need to track code changes more meticulously, even during steps I would consider rather unimportant. Third, I feel that I should seek help earlier when getting stuck with certain things, e.g., the installation of software.

8 Conclusion

This programming project pursued the goal of finetuning a Whisper model on existing Portunhol data. The small but important corpus created by Bárbara Garrido can be considered an optimal resource for such task, however, it lacks alignment between the recorded audio data and the transcriptions. After rather simple preprocessing, such alignment and, followingly, cutting of the data in chunks suited for Whisper models, would have been the next big step. However, the tool we aimed to use did not yield the respective outputs, and due to time reasons, the project had to be stopped. Nevertheless, the project has greatly served for practicing working with already existing code, and I am looking forward to use the knowledge gained in the process to continue the project idea at a later point.