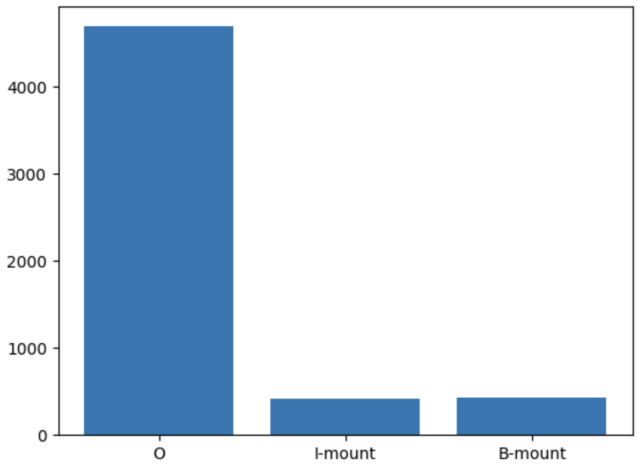
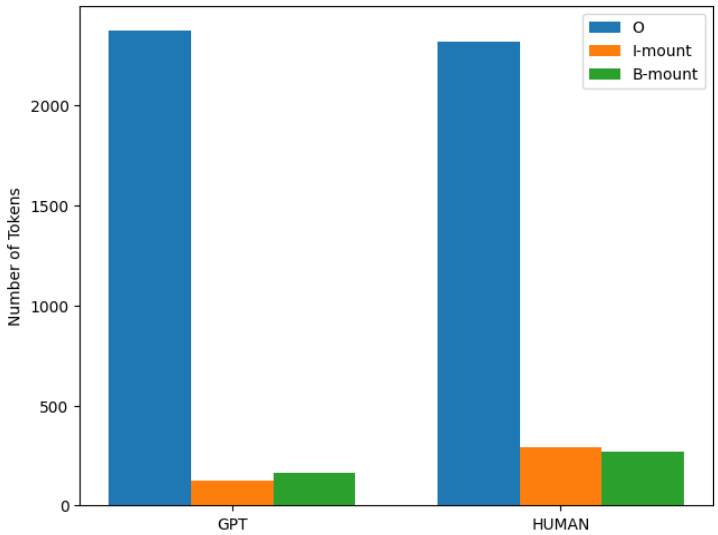
I was trying to implement all functions as reusable as possible, however, with very tight deadlines I sacrificed some best practices in order to start training as soon as possible. My codebase is divided into two parts: code for data generation and code for train/inference. I decided to write code in PyTorch as it is native framework for the HuggingFace transformers package.

Data Analysis

Let's first analyze our dataset, so we can properly understand the behavior of the model. We start by analyzing the number of tokens, which is:



As we can see there is clearly class imbalance in the dataset. If we compare the number of tokens in human sentences versus the number of tokens in GPT-generated tokens we would get the following statistics:



Thus we can make the hypothesis that natural human speech in the dataset contains more complex mountain names (name consists of more than 2 words) than in generated via gpt.

Enhancements

Generating data

For generation of data I have used OpenAI API and the ‘gpt-3.5-turbo’ model. There are three ways data can be gathered:

1. Passing to the ChatGPT desired names, so it can generate a sentence. Then the sentence is being parsed into tokens.
2. Passing to the ChatGPT a sentence that surely contains names of mountains or mountain ranges, so it can extract names from the sentence.
3. Telling ChatGPT to generate its own sentences and give them to us specifying names it has used during generation.

I used the first two methods, as intuitively I think they give me more control of what ChatGPT can actually generate. Also I did not have enough time to properly implement the full pipeline to check answers of the gpt model. The first method I would classify as ‘synthetic’, and the second one is ‘natural’. I tried to put as much ‘natural’ data as possible, because ChatGPT, as a lot of other gpt models, generates biased output.

The result of my data gathering is 271 sentences, which mostly contains at least one mountain name. I included a few sentences without names, because I did not want the model to become biased towards containing mountain names in a sentence.

After thorough review of the data and assessing results of training I thought of some ways to improve the current state of the situation. So my few improvements to the data generation are:

1) Gather more ‘natural’ data from various sources, not only scientific papers and journals. As I have almost no time for gathering data, I would take it from the easiest sources – and in my case it were articles from National Geographic, or any other website on the first Google page after searching “articles on mountains”. Due to the fact that data is mostly from scientists, Bert model performs poorly on more everyday sentences (for example, a person decided to write a friend about some mountain).

2) Make prompts better. This is a very important part of the enhancements, as prompts are being used in both ‘synthetic’ and ‘natural’ methods. It takes a lot of labor to clean all data that ChatGPT has generated, and I personally spent hours cleaning it and checking (more often ChatGPT generated sentences including not only the names I passed, but also alternative names of mountains). Enhancing prompts can be crucial in data cleaning and also can be crucial in generating more unique sentences.
3) Use the third method (only with better prompts). This improvement can increase the amount of data we will train our model on, but it requires much more sophisticated prompt engineering. It eliminates the need to scrap websites in search of unique ‘natural’ data. Also it would be better to use gpt-4 instead, as it was trained on bigger amounts of data and can be more efficient in generating unique sentences.
4) Diversify features of data. Currently I have big problems with the amount of non-mountain entities in sentences, or the variety of length of sentences. Model can be surprised to see non-mountain names with the first letter as uppercase in the middle of a sentence, because it has overfitted to the fact that there are no other names than mountains. Also model can perform poorly on long sentences.

Training

I developed an algorithm in torch, to train the model using ‘.csv’ file. This file is being passed at the start of the training loop. While training there are a number of metrics considered: loss, accuracy, mAP and recall.

So the following adjustments must be made:

1) Use weighted loss instead. There is strong evidence of large class imbalance, by that I mean that there are clearly much less ‘O’ tokens, than ‘B-mount’ or ‘I-mount’. Also there are less ‘I-mount’ tokens than ‘B-mount’ tokens. Using weighted loss tells the model to pay more attention to ‘I-mount’ and ‘B-mount’ rather than care only about ‘O’.
2) Add functional callbacks. It is more like the comfort of a training model. In my case it is not necessary, because I have 4-7 epochs to fine-tune Bert, but in other cases (maybe I will somehow change the algorithm, so it will need to train more) it can be important to save the best weights (like early stopping callback) or adapt the learning rate (learning rate schedule) to avoid getting stuck in non-optimal minimum etc.

3) Implement training through Trainer HuggingFace class. I decided to use native torch, because it is more flexible and helps imagine how things work inside. But as I use HuggingFace pretrained models it will be wise to go for a Trainer insead.