Main NLP tasks

- Token classification
- Masked language modeling (like BERT)
- Summarization
- Translation
- Causal language modeling pretraining (like GPT-2)
- Question answering

Token classification

Some popular token classification methods:

- **Named entity recognition (NER)**: Find the entities (such as persons, locations, or organizations) in a sentence. This can be formulated as attributing a label to each token by having one class per entity and one class for “no entity.”
- **Part-of-speech tagging (POS)**: Mark each word in a sentence as corresponding to a particular part of speech (such as noun, verb, adjective, etc.).
- **Chunking**: Find the tokens that belong to the same entity. This task (which can be combined with POS or NER) can be formulated as attributing one label (usually `B-`) to any tokens that are at the beginning of a chunk, another label (usually `I-`) to tokens that are inside a chunk, and a third label (usually `O`) to tokens that don’t belong to any chunk.

To build a token classifier we are going to be using a dataset named **CoNLL-2003 dataset**
, which contains news stories from Reuters. We will be building a NER classifier.

NER classes: `['O', 'B-PER', 'I-PER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-`

`MISC']`

O means the word doesn’t correspond to any entity.
B-PER/I-PER means the word corresponds to the beginning of/is inside a person entity.
B-ORG/I-ORG means the word corresponds to the beginning of/is inside an organization entity.
B-LOC/I-LOC means the word corresponds to the beginning of/is inside a location entity.
B-MISC/I-MISC means the word corresponds to the beginning of/is inside a miscellaneous entity.

Note:

To tokenize a pre-tokenized input, we can use our `tokenizer` as usual and just add `is_split_into_words=True`:

The metric for NER Classification is using the library SEQEVAL

Seqeval is a Python framework for sequence labeling evaluation. seqeval can evaluate the performance of chunking tasks such as named-entity recognition, part-of-speech tagging, semantic role labeling and so on.

Defining the model

Ofcourse the transformers have a `AutoModelForTokenClassification` to build a model from checkpoint (bert-cased)

Fine tuning the Model

Masked Language Modelling

For many NLP application we can take a pretrained model from hub and fine-tune it on our data for the task at hand. One important consedration before doing it is to make sure that the corpus on which the pretraining was performed is not too different from our corpus. If this consderation holds good then we can use the pretrained model and get good results.

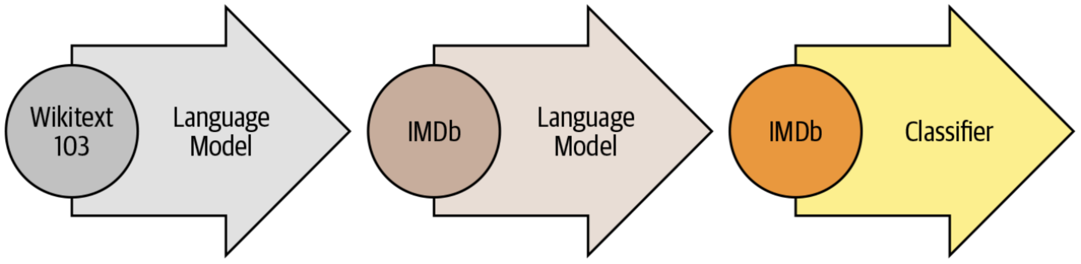
But if the case is that pretrained model is worked on a very different coprus from the one we want for our work then it is better to first fine-tune the Language Model on our data.

For example, if your dataset contains legal contracts or scientific articles, a vanilla Transformer model like BERT will typically treat the domain-specific words in your corpus as rare tokens, and the resulting performance may be less than satisfactory.

Fine-tuning the LM in such case will be a better way to proced. This process of fine-tuning a pretrained language model on in-domain data is usually called *domain adaptation*

. It was popularized in 2018 by **ULMFiT** which was one of the first neural architectures (based on LSTMs) to make transfer learning really work for NLP.

An example of Domain Adaption with ULMFiT



We use a the **DistilBERT model** that can be trained much faster with little to no loss in downstream performance. This model was trained using a special technique called ***knowledge distillation*** where a large "teacher model" like BERT is used to guide the training of a "student model" that has far fewer parameters.