Photo by Sigmund on Unsplash

Feature Extraction with BERT for Text Classification

Extract information from a pretrained model using Pytorch and Hugging Face

Marcello Politi · Follow
Published in Towards Data Science
4 min read · Jun 27, 2022

Goal

Let's begin by defining what our purpose is for this hands-on article. We want to build a model that takes as input one or more documents, and manages to classify them by their content.
Some categories are for example : politics, travel , sports, etc.
To do this we want to use a pretrained model such as BERT.

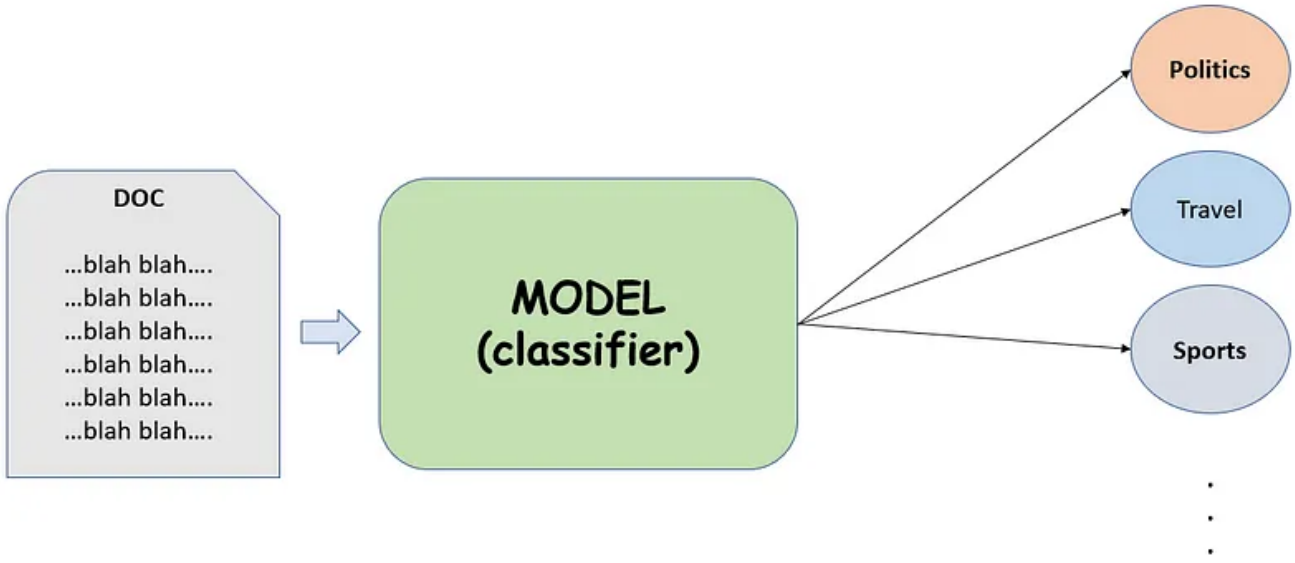

Image By Author

Background

BERT is a language model based heavily on the Transformer encoder. If you are unfamiliar with Transformers I recommend reading this amazing article.

Bert in a nutshell :

- It takes as input the embedding tokens of one or more sentences.
- The first token is always a special token called **[CLS]**.
- The sentences are separated by another special token called **[SEP]**.
- For each token BERT outputs an embedding called **hidden state**.
- Bert was trained on the **masked language model** and **next sentence prediction** tasks.

In the **masked language model (MLM)**, an input word (or token) is masked and BERT has to try to figure out what the masked word is. For the **next sentence prediction (NSP)** task, two sentences are given in input to BERT, and he has to figure out whether the second sentence follows semantically from the first one.

For classification problems such as ours, **we are only interested in the hidden state associated with the initial token [CLS]**, which somehow **captures the semantics of the entire sentence** better than the others. So we can use this embedding as input of a classifier that webuild on top of it.

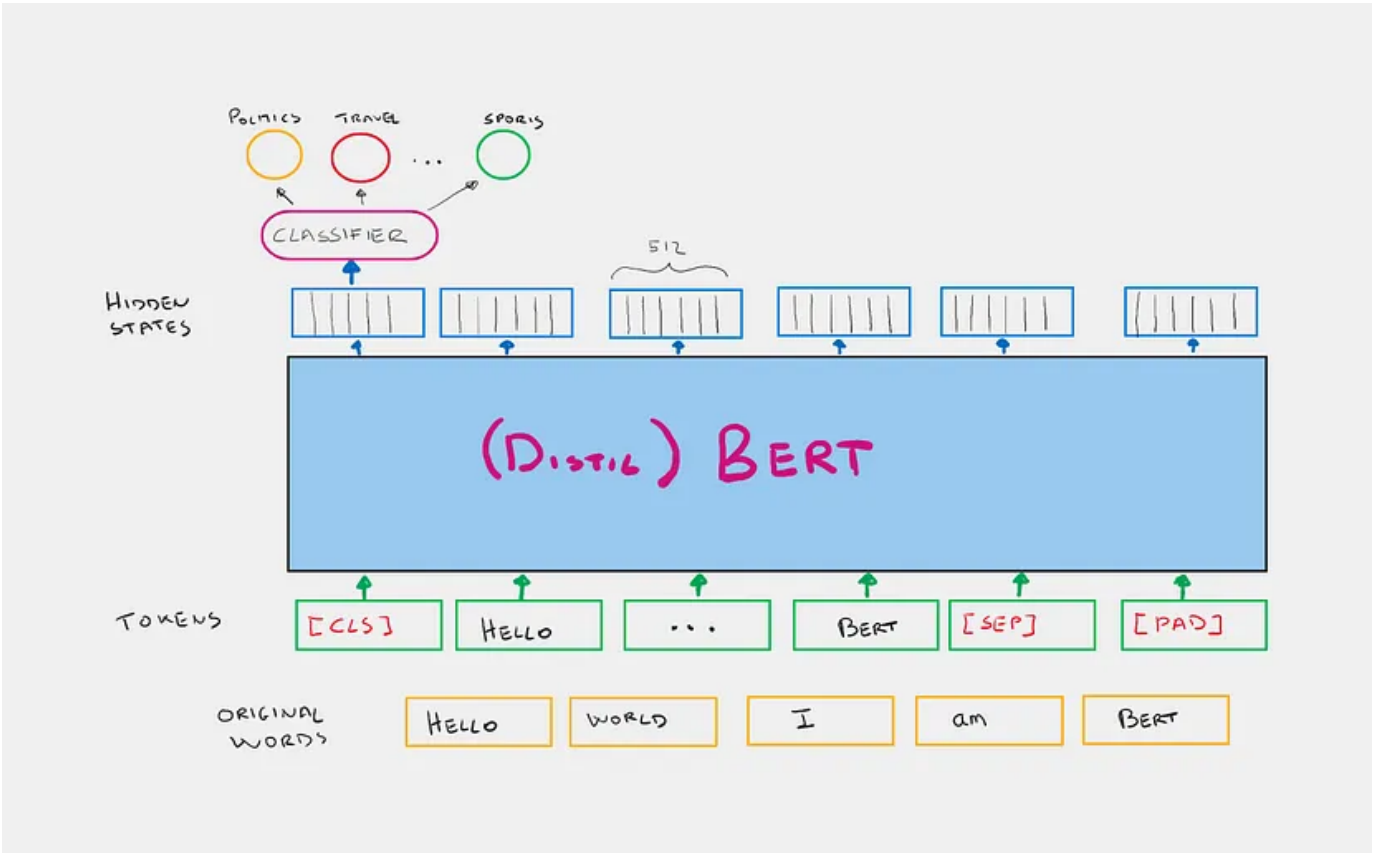

Image By Author

From the image above you can see that we will be using a lighter version of BERT called **DistilBERT**. This distilled model is **40% smaller** than the original but still maintains about 97% performance on the various NLP tasks.
Another thing you can notice, is that BERT's input are not the original words but the tokens. Simply BERT has associated a tokenizer that preprocess the text so that it is appealing for the model. Words are often split into subwords and in addition special tokens are added **[CLS]** to indicate the beginning of the sentence, **[SEP]** to separate multiple sentences, and **[PAD]** to make each sentence have the same number of tokens.

If you want to learn more about Bert or his wordpiece tokenizer check out these resources:

- https://jalammar.github.io/illustrated-bert/
- https://huggingface.co/blog/bert-101
- https://towardsdatascience.com/wordpiece-subword-based-tokenization-algorithm-1fbd14394ed7
- https://huggingface.co/docs/transformers/tokenizer_summary

Let's code!

Dataset

The dataset we are going to use is called *BBC Full Text Document Classification* and it is available on Kaggle with public access.

> *This dataset contains 2225 records, which consists of 5 categories in total. The five categories we want to identify are Sports, Business, Politics, Tech, and Entertainment.*

Download kaggle dataset directly from colab. (Remember to upload you personal kaggle key)

```
!pip install -q kaggle

from google.colab import files
files.upload()

!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d 'shivamkushwaha/bbc-full-text-document-classification'
!unzip DIRECTORY_NAME
```

download_data.py hosted with ❤ by GitHub view raw

I won't go into the details of how to read this data and turn it into a dataframe, but I assume you can create a daframe with two columns : *short_description* and *category*.

Encoding Labels

The dataset has labels such as : *Politics, Sports, etc.*...
We need to **transform these labels into numbers** by simply using a label encoder.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder

#for dimostration pourposes I inly use 2k data
df = pd.read_json("/content/dataset.json", lines = True).sample(frac=1)[:2_000]
LE = LabelEncoder()
df['label'] = LE.fit_transform(df['category'])
df.head()
```

label_encoder.py hosted with ❤ by GitHub view raw

Generate Text Embeddings

For each text generate an embedding vector, that can be used as input to our final classifier. The **vector embedding associated to each text is simply the hidden state that Bert outputs for the [CLS] token.**

Let's start by importing the model and tokenizer from **HuggingFace**.

```
!pip install transformers

import torch
from transformers import AutoTokenizer, AutoModel

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
model = AutoModel.from_pretrained("distilbert-base-uncased").to(device)
```

download_pretrained_model.py hosted with ❤ by GitHub view raw

Now we have to tokenize the text. Remember to define the ***padding*** in this way each tokenized sentence will have the same length, and the ***truncation*** so if the sentence is too long it will be cut off. The last argument is to return a PyTorch tensor.

The result of tokenizing a text will be a dictionary that contains the **input_ids** , that are the tokens expressed in numbers, and the **attention_mask** that tells us if the token is or is not a [PAD].

```
tokenized_train = tokenizer(df_train["short_description"].values.tolist(), padding = True, truncation = True, return_tensors="pt")
tokenized_val = tokenizer(df_val["short_description"].values.tolist() , padding = True, truncation = True,  return_tensors="pt")

print(tokenized_train.keys())

#move on device (GPU)
tokenized_train = {k:torch.tensor(v).to(device) for k,v in tokenized_train.items()}
tokenized_val = {k:torch.tensor(v).to(device) for k,v in tokenized_val.items()}
```

tokenization.py hosted with ❤ by GitHub view raw

Get the texts ([CLS]) hidden states by running the model.

```
with torch.no_grad():
  hidden_train = model(**tokenized_train) #dim : [batch_size(nr_sentences), tokens, emb_dim]
  hidden_val = model(**tokenized_val)

#get only the [CLS] hidden states
cls_train = hidden_train.last_hidden_state[:,0,:]
cls_val = hidden_val.last_hidden_state[:,0,:]
```

hidden_states.py hosted with ❤ by GitHub view raw

Build the classifier

Having reached this point you can use the classifier you like best by giving it as input the hidden states and asking it to predict the labels. In this case I will use a RandomForest.

```
x_train = cls_train.to("cpu")
y_train = df_train["label"]

x_val = cls_val.to("cpu")
y_val = df_val["label"]

print(x_train.shape, y_train.shape, x_val.shape, y_val.shape)


from sklearn.ensemble import RandomForestClassifier
```


classifier.py hosted with ❤ by GitHub view raw

The performance of this classifier will not be great because we used little data and did not do much work on the classifier. But out of curiosity **I recommend that you compare it against a dummy classifier that predicts labels randomly.**

```python
from sklearn.dummy import DummyClassifier

dummy = DummyClassifier(strategy= "uniform")
dummy.fit(x_train,y_train)
dummy.score(x_val,y_val)
```

dummy_classifier.py hosted with ❤ by GitHub view raw

Final Thoughts

In this hands on article we saw how we can take advantage of the capabilities of a pretrained model to create a simple classifier in very little time.
Keep in mind that what we trained is only the final classifier, i.e., the random forest.
Bert, on the other hand, was used only in inference to generate the embeddings that somehow capture the main features of the texts, which is why we say we used a Feature Extraction methodology.
But can we train Bert himself to teach him how to create better text embeddings in our specific case? Sure! In the next article I will show you how to do fine tuning of (Distil) BERT!

The End

Marcello Politi

Linkedin, Twitter, CV

NLP Deep Learning Machine Learning Data Science Artificial Intelligence

Follow

Written by Marcello Politi

1.2K Followers · Writer for Towards Data Science

Machine Learning Scientist @ PiSchool | Ex-ESA

More from Marcello Politi and Towards Data Science