Better Sentiment Analysis with BERT

Imagine you have a bot answering your clients, and you want to make it sound a little bit more natural, more human.

Marion Valette Follow
Apr 29 · 4 min read ★

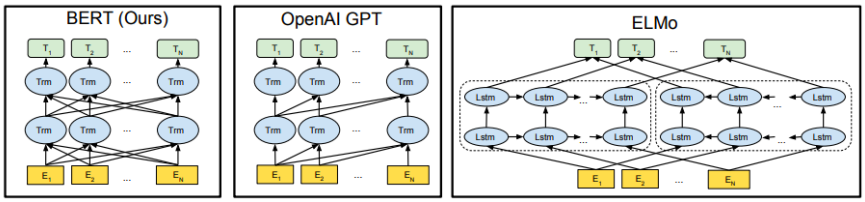
Photo by Hybrid on Unsplash

To achieve that, you have to make the answers more personalized. One way to learn more about the customers you're talking to is to analyze the polarity of their answers. By polarity here I mean detecting if the sentence (or group of sentences) is written with the intention of being perceived as a positive or negative statement. This means we are facing a binary classification problem. A lot of methods exist to solve this NLP task. I tested some and the one which really outperformed the others was BERT.

BERT Overview

BERT (Bidirectionnal Encoder Representations for Transformers) is a "new method of pre-training language representations" developed by Google and released in late 2018 (you can read more about it here). As it is pre-trained on generic datasets (from Wikipedia and BooksCorpus), it can be used to solve different NLP tasks. This includes sentence level classification (as we do here), question answering or

token level classification (eg. part of speech tagging), and BERT is able to achieve state-of-the-art performances in many of these tasks.



BERT architecture compared to two other state-of-the-art models (source: Devlin et al.)

In practice, BERT provides pre-trained language models for English and 103 other languages that you can fine-tune to fit your needs. Here, we'll see how to fine-tune the English model to do sentiment analysis.

Fine-Tuning with BERT

BERT recently provided a tutorial notebook in Python to illustrate how to make sentiment detection in movie reviews. The tutorial notebook is well made and clear, so I won't go through it in detail—here are just a few thoughts on it. First, the notebook uses the IMDb dataset, that can be downloaded directly from Keras. This dataset contains 50000 movie reviews split in two equal parts, one for training and one for testing. Each dataset is balanced, with 12500 positive reviews and 12500 negative ones.

Examples	Polarity
I think that movie can't be a Scott's film. That is impossible. Do you remember Blade Runner? And Alien? Two greats movies versus a one. I hope didn't see ever it. good bye	0
This film is a wonderfully simplistic work. Enjoyable from start to end it is both sad yet uplifting at the same time.	1

Examples of reviews from the IMDb dataset. Polarity to zero means that the sentence expresses negative feelings, whereas one means that it is positive.

To fine-tune, they apply a single new layer and softmax on top of the pre-trained model but you can customize it. It uses Tensorflow estimators structure to train and predict the results, and they require some functions like run_config or model_fn, either coded in the notebook or imported from the run_classifier.py file present in the GitHub, so you don't have to worry about them.

Model Evaluation

To see how well BERT performs, I compared it to two other models. The first one is a logistic regression with TF-IDF vectorization. The second one is inspired from Rezaeinia et al. (git). It uses Word2Vec embeddings, along with Part of Speech tagging, and passes the concatenation of both into a 1D convolutional network.

Although the logistic regression works surprisingly well, outperforming the neural based model, BERT yields even better scores. Moreover, BERT results improve significantly when the model is trained on a larger dataset. What can be a drawback is that it takes quite a long time to train, even with GPU. The logistic regression completes training within seconds, when BERT needs around 20 minutes to do so (with GPU and 25000 training reviews).

Models	Accuracy	F1 score
BERT	0.94	0.94
TF-IDF + logistic regression	0.91	0.91
Word2Vec + POS Tag + Conv1D	0.84	0.85

Results of the different models

Serving with Docker and Tensorflow

Once you have saved your model as a saved_model.pb (obtained for example with the export_savedmodel method), you can set up a server which will run the model and make predictions. First, we create a Docker container from tensorflow serving (you will have to install Docker first) and we add our model into it. Model_en should be a folder containing a folder named 1 (necessary for tensorflow) which itself contains the model you have exported. This is achieved by the command lines below.

```
# Recover the container provided by tensorflow serving
docker pull tensorflow/serving
docker run -d --name serving_base tensorflow/serving

# Add the model
docker cp model_en serving_base:/models/model_en
docker commit --change "ENV MODEL_NAME model_en" servi

# Check if the image has been correctly instantiated
docker images
```

Then we write a simple client, which will take a sentence as input, transform it into BERT input features (just like in the tutorial notebook) and call the running Docker container which will make the prediction. You can turn the client into a simple API using Flask. The python code below shows those different steps.

```
# Get sentence (requires Flask)
req = request.get_json()
raw = req["raw"]

# Transformation into BERT input features
input_examples = [run_classifier.InputExample(guid="",
input_features = run_classifier.convert_examples_to_fe
# see the notebook for the definition of the tokenizer
input_ids = np.array(input_features[0].input_ids) # Ad

# Right input format for tensorflow serving (see https
tensor_dict = {"label_ids": [label_ids], "segment_ids"
```

You can then do predictions by sending string requests, using Postman for example.

Voilà! You're now ready to do some fine sentiment analysis with BERT, and to use it in a real-world application.