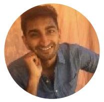
# Transformers for Multi-Label Classification made simple.

BERT, XLNet, RoBERTa, etc. for multilabel classification — a step by step guide



Ronak Patel Follow
May 27 · 4 min read ★

As a data scientist who has been learning the state of the art for text classification, I found that there are not many easy examples to adapt transformers (BERT, XLNet, etc.) for **multilabel classification**…so I decided to try for myself and here it is!

As
us



This post is accompanied by an interactive Google Colab notebook so you can try this yourself. All you have to do is upload the *train.csv, test.csv, and test_labels.csv* files into the instance. Let's get started.

In this tutorial I will be using Hugging Face's transformers library along with **PyTorch (with GPU)**, although this can easily be adapted to TensorFlow — *I may write a seperate tutorial for this later if this picks up traction along with tutorials for multiclass classification*. Below I will be training a BERT model but **I will show you how easy it is to adapt this code for other transformer models along the way.**

## Import Libraries

```
Sampler, SequentialSampler
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.metrics import classification_report, confusion_m
atrix, multilabel_confusion_matrix, f1_score, accuracy_score
import pickle
from transformers import *
from tqdm import tqdm, trange
from ast import literal_eval
```

```
In [0]: # from google.colab import drive
        # drive.mount('/content/drive')
```

```
Found GPU at: /device:GPU:0

In [4]: device = torch.device("cuda" if torch.cuda.is_available() else
        "cpu")
        n_gpu = torch.cuda.device_count()
        torch.cuda.get_device_name(0)

Out[4]: 'Tesla P100-PCIE-16GB'
```

import_libs_gist.ipynb hosted with ❤ by GitHub view raw

## Load & Preprocess Training Data

The toxic dataset is already cleaned and separated into train and test sets, so we can load the train set and use it directly.

Each transformer model requires different tokenization encodings — meaning the way that the sentence is tokenized and attention masks are used may differ depending on the transformer model you use. Thankfully, HuggingFace's transformers library makes it extremely easy to implement for each model. In the code below we load a pretrained BERT tokenizer and use the method "batch_encode_plus" to get tokens, token types, and

```
BERT:
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',
do_lower_case=True)

XLNet:
tokenizer = XLNetTokenizer.from_pretrained('xlnet-base-cased',
do_lower_case=False)

RoBERTa:
tokenizer = RobertaTokenizer.from_pretrained('roberta-base',
do_lower_case=False)
```

```
insult            7877
identity_hate     1405
dtype: int64

Count of 0 per label:
 toxic            144277
severe_toxic     157976
obscene          151122
threat           159093
insult           151694
identity_hate    158166
dtype: int64
```

```
In [0]: df = df.sample(frac=1).reset_index(drop=True) #shuffle rows
```

```
In [11]: df['one_hot_labels'] = list(df[label_cols].values)
```

| | | | | | |
|---|---|---|---|---|---|
| **0** | 5a7ca836c0f0ad72 | "\nIt should be obvious that that it is not th... | 0 | 0 | 0 |
| **1** | ebdf69f2b2f08828 | I see you back from Madison | 0 | 0 | 0 |
| **2** | a492014f9b9f739f | The topic appears to meet the criteria for spe... | 0 | 0 | 0 |

Load&preprocess_tokenizer_gist.ipynb hosted with ❤ by GitHub view raw

Next, we will use 10% of our training inputs as a validation set so we can monitor our classifier's performance as it is training. Here we want to make sure we utilize the "stratify" parameter so no unseen labels appear in the validation set. In order to stratify appropriately we will take all labels that only appear once in the dataset and force them into the training set. We will also need to create PyTorch data loaders to load the data for training/prediction.

```
In [13]: # Identifying indices of 'one_hot_labels' entries that only occur once - this will allow us to stratify split our training data later
label_counts = df.one_hot_labels.astype(str).value_counts()
one_freq = label_counts[label_counts==1].keys()
one_freq_idxs = sorted(list(df[df.one_hot_labels.astype(str).isin(one_freq)].index), reverse=True)
print('df label indices with only one instance: ', one_freq_idxs)
```

```
 set after stratified split
one_freq_input_ids = [input_ids.pop(i) for i in one_freq_idxs]
one_freq_token_types = [token_type_ids.pop(i) for i in one_fre
q_idxs]
one_freq_attention_masks = [attention_masks.pop(i) for i in on
e_freq_idxs]
one_freq_labels = [labels.pop(i) for i in one_freq_idxs]
```

Be sure to handle all classes during validation using "stratify" during train/validation split:

```
In [0]: # Use train_test_split to split our data into train and valida
tion sets
```

Load&preprocess_torch_tensor.ipynb hosted with ❤ by GitHub view raw

## Load Model & Set Params

Loading the appropriate model can be done as shown below, each model already contains a single dense layer for classification on top.

```
BERT:
model = BertForSequenceClassification.from_pretrained("bert-base-
uncased", num_labels=num_labels)

XLNet:
model = XLNetForSequenceClassification.from_pretrained("xlnet-base-
```

```
model = RobertaForSequenceClassification.from_pretrained('roberta-
base', num_labels=num_labels)
```

Optimizer params can configured in a few ways. Here we are using a customized optimization parameters (which I've had more success with), however, you could just pass "model.parameters( )" as shown in the comments.

load_params.ipynb hosted with ❤ by GitHub view raw

## Train Model

The HuggingFace library is configured for multiclass classification out of the box using **"Categorical Cross Entropy"** as the loss function. Therefore, the output of a transformer model would be akin to:

```
outputs = model(batch_input_ids, token_type_ids=None,
attention_mask=batch_input_mask, labels=batch_labels)

loss, logits = outputs[0], outputs[1]
```

**However, if we avoid passing in a labels parameter, the model will only output logits, which we can use to calculate our own loss for multilabel classification.**

```
logits = outputs[0]
```

Below is the code snippet of doing exactly that. Here we use **"Binary Cross Entropy With Logits"** as our loss function. We could have just as easily used standard "Binary Cross Entropy", "Hamming Loss", etc.

For validation, we will use micro F1 accuracy to monitor training performance across epochs. To do so we will have to utilize our logits from our model output, pass them through a sigmoid function (giving us outputs between [0, 1], and threshold them (at 0.50) to generate predictions. These predictions can then be used to calculate accuracy against the true labels.

## Train Model

```
In [23]: # Store our loss and accuracy for plotting
train_loss_set = []

# Number of training epochs (authors recommend between 2 and
 4)
epochs = 3

# trange is a tqdm wrapper around the normal python range
for _ in trange(epochs, desc="Epoch"):
```

```
  # Set our model to training mode (as opposed to evaluation m
ode)
  model.train()

  # Tracking variables
  tr_loss = 0 #running loss
  nb_tr_examples, nb_tr_steps = 0, 0

  # Train the data for one epoch
  for step, batch in enumerate(train_dataloader):
    # Add batch to GPU
```

train_model_gist.ipynb hosted with ❤ by GitHub view raw

Viola! We're ready for training, now run it…my train times ranged between 20–40 min per epoch depending on the max token length and the GPU at use.

## Prediction & Metrics

Prediction for our test set is similar to our validation set. Here we will be loading, preprocessing, and predicting with the test data.

### Load and Preprocess Test Data

```
test_label_cols = list(test_df.columns[2:])
print('Null values: ', test_df.isnull().values.any()) #should
 not be any null sentences or labels
print('Same columns between train and test: ', label_cols == t
est_label_cols) #columns should be the same
test_df.head()
```

```
Null values:  False
Same columns between train and test:  True
```

Out[25]:

| | id | comment_text | toxic | severe_toxic | obscene | threat | insu |
|---|---|---|---|---|---|---|---|
| **0** | 00001cee341fdb12 | Yo bitch Ja Rule is more succesful then you'll... | -1 | -1 | -1 | -1 | -1 |
| **1** | 0000247867823ef7 | == From RfC == \n\n The title is fine as it | -1 | -1 | -1 | -1 | -1 |

test_data_load_preprocess_predict.ipynb hosted with ❤ by GitHub view raw

## Output DataFrame

Creating a dataframe of outputs that show sentences and their classification.

output_df.ipynb hosted with ❤ by GitHub view raw

## Bonus — Optimizing Threshold for Micro F1 Accuracy

Iterating through threshold values to maximize Micro F1 Accuracy.

Bonus_thresholding.ipynb hosted with ❤ by GitHub view raw

That's it! Please comment if you have any questions. Here is the link to the Google Colab notebook again in case you missed it. If you have any personal inquiries feel free to contact me on LinkedIn or Twitter.

# R



https://electricenjin.com/blog/look-out-google-bert-is-here-to-shake-up-search-queries

https://github.com/google-research/bert

https://github.com/huggingface/transformers

https://pytorch.org/docs/stable/nn.html#bcewithlogitsloss

https://arxiv.org/abs/1706.03762

Bert Xlnet NLP Data Science Machine Learning