project-source-code

September 7, 2020

1 Project Overview

- This project will focus on addressing the problem of sentiment analysis using BERT Deep Learning technique.
- BERT is a large-scale transformer-based Language Model that can be fine-tuned for different tasks.
- Original paper of BERT can be found here.
- For the experiments, we will use SMILE Twitter Emotion dataset.

[1]:
```
import torch
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot
from tqdm.notebook import tqdm
%matplotlib inline
```

2 Data Exploration

[2]:
```
df = pd.read_csv("smile-annotations-final.csv", names=['id', 'text',␣
↪'category'])
df.head()
```

[2]:
```
                   id                                               text  \
0  611857364396965889  @aandraous @britishmuseum @AndrewsAntonio Merc...
1  614484565059596288  Dorian Gray with Rainbow Scarf #LoveWins (from...
2  614746522043973632  @SelectShowcase @Tate_StIves ... Replace with ...
3  614877582664835073  @Sofabsports thank you for following me back. ...
4  611932373039644672  @britishmuseum @TudorHistory What a beautiful ...

  category
0   nocode
1    happy
2    happy
3    happy
4    happy
```

As we know that id is unique for every tweet, we would like to set index based on id number of every sample.

[3]:
```
df.set_index('id', inplace=True)
df.head()
```

[3]:
```
                                                                 text category
id
611857364396965889  @aandraous @britishmuseum @AndrewsAntonio Merc...   nocode
614484565059596288  Dorian Gray with Rainbow Scarf #LoveWins (from...    happy
614746522043973632  @SelectShowcase @Tate_StIves ... Replace with ...    happy
614877582664835073  @Sofabsports thank you for following me back. ...    happy
611932373039644672  @britishmuseum @TudorHistory What a beautiful ...    happy
```

[4]:
```
df.text.iloc[100]
```

[4]:
```
"* @hist_astro @britishmuseum It looks like there's some #ArtisticLicence
involved in that sketch of #SummerSolstice #sunrise at #Stonehenge."
```

[5]:
```
df.category.value_counts()
```

[5]:
```
nocode                 1572
happy                  1137
not-relevant            214
angry                    57
surprise                 35
sad                      32
happy|surprise           11
happy|sad                 9
disgust|angry             7
disgust                   6
sad|disgust               2
sad|angry                 2
sad|disgust|angry         1
Name: category, dtype: int64
```

[6]:
```
df = df[~df.category.str.contains('\|')]
df.category.value_counts()
```

[6]:
```
nocode          1572
happy           1137
not-relevant     214
angry             57
surprise          35
sad               32
disgust            6
Name: category, dtype: int64
```

[7]:
```
df = df[df.category!='nocode']
df.category.value_counts()
```

[7]:
```
happy           1137
not-relevant     214
angry             57
surprise          35
sad               32
disgust            6
Name: category, dtype: int64
```

[8]:
```python
labels = df.category.unique()
labels_di = {}

for index, label in enumerate(labels):
    labels_di[label] = index
print(labels_di)
```

```
{'happy': 0, 'not-relevant': 1, 'angry': 2, 'disgust': 3, 'sad': 4, 'surprise':
5}
```

[9]:
```python
df['label'] = df.category.replace(labels_di)
print(df.label.value_counts())
print(df.category.value_counts())
```

```
0    1137
1     214
2      57
5      35
4      32
3       6
Name: label, dtype: int64
happy           1137
not-relevant     214
angry             57
surprise          35
sad               32
disgust            6
Name: category, dtype: int64
```

3 Data Separation

Since the dataset for the experiment is not balanced, we do not want to use simple train-test-split for this data. Because, a class with limited datasamples might not be represented in training or validation sets, which will cause problem of generalizability of the model. Therefore, we would like to use stratified split that ensures a certain portion of the examples will be in training and test splits for each class.

[10]:
```python
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(df.index.values, df.label.
 ↪values,
```

```
                                                    stratify=df.label.values,␣
↪test_size=0.15,
                                                    shuffle=True,␣
↪random_state=2020)
```

Now, we can check wheter the samples in the data were properly distributed into train and validation sets. For this, we can create another column in the dataframe, called 'data_type', which shows wheter the sample is in training or validation set.

[11]:
```
df['data_type'] = ['not_set'] * df.shape[0]
df.head()
```

[11]:
```
                                                                 text  \
id
614484565059596288  Dorian Gray with Rainbow Scarf #LoveWins (from...
614746522043973632  @SelectShowcase @Tate_StIves ... Replace with ...
614877582664835073  @Sofabsports thank you for following me back. ...
611932373039644672  @britishmuseum @TudorHistory What a beautiful ...
611570404268883969  @NationalGallery @ThePoldarkian I have always ...

                   category  label data_type
id
614484565059596288    happy      0   not_set
614746522043973632    happy      0   not_set
614877582664835073    happy      0   not_set
611932373039644672    happy      0   not_set
611570404268883969    happy      0   not_set
```

[12]:
```
df.loc[X_train, 'data_type'] = 'train'
df.loc[X_val, 'data_type'] = 'val'
df.head()
```

[12]:
```
                                                                 text  \
id
614484565059596288  Dorian Gray with Rainbow Scarf #LoveWins (from...
614746522043973632  @SelectShowcase @Tate_StIves ... Replace with ...
614877582664835073  @Sofabsports thank you for following me back. ...
611932373039644672  @britishmuseum @TudorHistory What a beautiful ...
611570404268883969  @NationalGallery @ThePoldarkian I have always ...

                   category  label data_type
id
614484565059596288    happy      0       val
614746522043973632    happy      0     train
614877582664835073    happy      0     train
611932373039644672    happy      0     train
611570404268883969    happy      0     train
```

[13]:
```
df.groupby(['category', 'label', 'data_type']).count()
```

[13]:
```
                              text
category      label data_type
angry         2     train        48
                    val           9
disgust       3     train         5
                    val           1
happy         0     train       966
                    val         171
not-relevant  1     train       182
                    val          32
sad           4     train        27
                    val           5
surprise      5     train        30
                    val           5
```

4 Tokenizing and Encoding the data

[14]:
```python
from transformers import BertTokenizer
from torch.utils.data import TensorDataset
```

```
D:\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of
the second argument of issubdtype from `float` to `np.floating` is deprecated.
In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

[15]:
```python
tokenizer = BertTokenizer.from_pretrained(
            'bert-base-uncased',
            do_lower_case=True)
```

[16]:
```python
encoded_train = tokenizer.batch_encode_plus(
                df[df['data_type']=='train'].text.values,
                add_special_tokens=True,
                return_attention_mask=True,
                pad_to_max_length=True,
                max_length=256, return_tensors='pt')

encoded_val = tokenizer.batch_encode_plus(
                df[df['data_type']=='val'].text.values,
                add_special_tokens=True,
                return_attention_mask=True,
                pad_to_max_length=True,
                max_length=256, return_tensors='pt')
print(f"{encoded_train.keys()}\n{encoded_val.keys()}")
```

```
Truncation was not explicitely activated but `max_length` is provided a specific
value, please use `truncation=True` to explicitely truncate examples to max
```

```
length. Defaulting to 'longest_first' truncation strategy. If you encode pairs
of sequences (GLUE-style) with the tokenizer you can select this strategy more
precisely by providing a specific strategy to `truncation`.
D:\Anaconda3\lib\site-packages\transformers\tokenization_utils_base.py:1770:
FutureWarning: The `pad_to_max_length` argument is deprecated and will be
removed in a future version, use `padding=True` or `padding='longest'` to pad to
the longest sequence in the batch, or use `padding='max_length'` to pad to a max
length. In this case, you can give a specific length with `max_length` (e.g.
`max_length=45`) or leave max_length to None to pad to the maximal input size of
the model (e.g. 512 for Bert).
  FutureWarning,
Truncation was not explicitely activated but `max_length` is provided a specific
value, please use `truncation=True` to explicitely truncate examples to max
length. Defaulting to 'longest_first' truncation strategy. If you encode pairs
of sequences (GLUE-style) with the tokenizer you can select this strategy more
precisely by providing a specific strategy to `truncation`.
```

```
dict_keys(['input_ids', 'token_type_ids', 'attention_mask'])
dict_keys(['input_ids', 'token_type_ids', 'attention_mask'])
```

[17]:
```
input_ids_train = encoded_train['input_ids']
attention_masks_train = encoded_train['attention_mask']
labels_train = torch.tensor(df[df['data_type'] == 'train'].label.values)

input_ids_val = encoded_val['input_ids']
attention_masks_val = encoded_val['attention_mask']
labels_val = torch.tensor(df[df['data_type'] == 'val'].label.values)
```

[18]:
```
dataset_train = TensorDataset(input_ids_train,
                              attention_masks_train,
                              labels_train)

dataset_val = TensorDataset(input_ids_val,
                            attention_masks_val,
                            labels_val)
print(f"There are {len(dataset_train)} samples in the training data!")
print(f"There are {len(dataset_val)} samples in the validation data!")
```

```
There are 1258 samples in the training data!
There are 223 samples in the validation data!
```

5 Formulating a Model

[19]:
```
from transformers import BertForSequenceClassification
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
                                                      num_labels = len(labels_di),
                                                      output_attentions=False,
```

```
                                                      output_hidden_states=False)
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when
initializing BertForSequenceClassification: ['cls.predictions.bias',
'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from
the checkpoint of a model trained on another task or with another architecture
(e.g. initializing a BertForSequenceClassification model from a
BertForPretraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification
from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

6 Creating DataLoaders

[20]:
```
from torch.utils.data import DataLoader, RandomSampler, SequentialSampler

bs = 8
dataloader_train = DataLoader(dataset_train,
                              sampler=RandomSampler(dataset_train),
                              batch_size=bs)
dataloader_val = DataLoader(dataset_val,
                            sampler=RandomSampler(dataset_val),
                            batch_size=bs*4)
```

7 Setting an optimizer and scheduler

[21]:
```
from transformers import AdamW, get_linear_schedule_with_warmup

epochs = 10
optimizer = AdamW(model.parameters(),
                  lr=1e-5, eps=1e-8)
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=0,
```

```
↪num_training_steps=len(dataloader_train)*epochs)
```

8 Defining evaluation metrics

Since we have imbalanced dataset, we may be interested in using f1-score, which is one of the most appropriate evaluation metrics out there to be used for inbalanced data.

[22]:
```
from sklearn.metrics import f1_score

def f1_score_func(preds, targs):
    preds_flat = np.argmax(preds, axis=1).flatten()
    targs_flat = targs.flatten()
    return f1_score(targs_flat, preds_flat, average='weighted')

def accuracy_per_class(preds, labels):

    label_di_inverse = {v:k for k, v in labels_di.items()}
    preds_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()

    for label in np.unique(labels_flat):
        y_preds = preds_flat[labels_flat==label]
        y_true = labels_flat[labels_flat==label]
        print(f"Class: {label_di_inverse[label]}")
        print(f"Accuracy: {len(y_preds[y_preds==label])}/{len(y_true)}\n")
```

9 Training a BERT model

[23]:
```
import random

seed_val = 17
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
model.to(device)
```

```
cuda
```

[23]:
```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
```

```
  (word_embeddings): Embedding(30522, 768, padding_idx=0)
  (position_embeddings): Embedding(512, 768)
  (token_type_embeddings): Embedding(2, 768)
  (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (dropout): Dropout(p=0.1, inplace=False)
)
(encoder): BertEncoder(
  (layer): ModuleList(
    (0): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (1): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
```

```
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (2): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (3): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
        )
```

```
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (4): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (5): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
```

```
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (6): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (7): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
```

```
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (8): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (9): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
```

```
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(10): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(11): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
```

```
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=6, bias=True)
)
```

[24]:
```
def evaluate(dataloader_val):

    model.eval()

    loss_val_total = 0
    preds, true_vals = [], []

    for batch in tqdm(dataloader_val):

        batch = tuple(b.to(device) for b in batch)

        inputs = {'input_ids'      : batch[0],
                  'attention_mask': batch[1],
                  'labels'        : batch[2]}

        with torch.no_grad():

            outputs = model(**inputs)
        loss = outputs[0]
        logits = outputs[1]
        loss_val_total += loss.item()

        logits = logits.detach().cpu().numpy()
        label_ids = inputs['labels'].cpu().numpy()
        preds.append(logits)
        true_vals.append(label_ids)

    loss_val_avg = loss_val_total / len(dataloader_val)

    preds = np.concatenate(preds, axis=0)
    true_vals = np.concatenate(true_vals, axis=0)

    return loss_val_avg, preds, true_vals
```

[25]:
```
for epoch in tqdm(range(1, epochs+1)):

    model.train()
    loss_train_total = 0
    progress_bar = tqdm(dataloader_train,
                        desc='Epoch {:1d}'.format(epoch),
                        leave=False, disable=False)

    for batch in progress_bar:
        model.zero_grad()
        batch = tuple(b.to(device) for b in batch)

        inputs = {'input_ids'     : batch[0],
                  'attention_mask': batch[1],
                  'labels'        : batch[2]}

        outputs = model(**inputs)

        loss = outputs[0]
        loss_train_total += loss.item()
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()

        progress_bar.set_postfix({'Training loss': '{:.3f}'.format(loss.item()/
↪len(batch))})

#     torch.save(model.state_dict(), f'Models/BERT_ft_epoch{epoch}.model')
    tqdm.write(f'\nEpoch {epoch}')
    loss_train_avg = loss_train_total/len(dataloader_train)
    tqdm.write(f"Training loss: {loss_train_avg}")

    val_loss, preds, true_vals = evaluate(dataloader_val)
    val_f1 = f1_score_func(preds, true_vals)
    tqdm.write(f"Validation loss: {val_loss}")
    tqdm.write(f"F1 Score: {val_f1}")
```

```
HBox(children=(FloatProgress(value=0.0, max=10.0), HTML(value='')))


HBox(children=(FloatProgress(value=0.0, description='Epoch 1', max=158.0, style=ProgressStyle(d



Epoch 1
Training loss: 0.8713658031406282
```

```
HBox(children=(FloatProgress(value=0.0, max=7.0), HTML(value='')))
```

```
Validation loss: 0.6209014100687844
F1 Score: 0.6953185953656175
```

```
D:\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels
with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
HBox(children=(FloatProgress(value=0.0, description='Epoch 2', max=158.0, style=ProgressStyle(c
```

```
Epoch 2
Training loss: 0.5248276907243307
```

```
HBox(children=(FloatProgress(value=0.0, max=7.0), HTML(value='')))
```

```
Validation loss: 0.4816068027700697
F1 Score: 0.8275480263522117
```

```
D:\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels
with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
HBox(children=(FloatProgress(value=0.0, description='Epoch 3', max=158.0, style=ProgressStyle(c
```

```
Epoch 3
Training loss: 0.38210659294943267
```

```
HBox(children=(FloatProgress(value=0.0, max=7.0), HTML(value='')))
```

```
Validation loss: 0.39346700268132345
F1 Score: 0.8527610995571987
```

```
D:\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels
with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
HBox(children=(FloatProgress(value=0.0, description='Epoch 4', max=158.0, style=ProgressStyle(d

Epoch 4
Training loss: 0.25785991115660606

HBox(children=(FloatProgress(value=0.0, max=7.0), HTML(value='')))

Validation loss: 0.3448405755417688
F1 Score: 0.8841925591434728

D:\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels
with no predicted samples.
  'precision', 'predicted', average, warn_for)

HBox(children=(FloatProgress(value=0.0, description='Epoch 5', max=158.0, style=ProgressStyle(d

Epoch 5
Training loss: 0.18120793347494513

HBox(children=(FloatProgress(value=0.0, max=7.0), HTML(value='')))

Validation loss: 0.46844774058886934
F1 Score: 0.8689643477616807

D:\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels
with no predicted samples.
  'precision', 'predicted', average, warn_for)

HBox(children=(FloatProgress(value=0.0, description='Epoch 6', max=158.0, style=ProgressStyle(d

Epoch 6
Training loss: 0.12532829719630978

HBox(children=(FloatProgress(value=0.0, max=7.0), HTML(value='')))

Validation loss: 0.513486019202641
F1 Score: 0.8811608118805235
```

```
D:\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels
with no predicted samples.
  'precision', 'predicted', average, warn_for)

HBox(children=(FloatProgress(value=0.0, description='Epoch 7', max=158.0, style=ProgressStyle(d



Epoch 7
Training loss: 0.08559641675858558

HBox(children=(FloatProgress(value=0.0, max=7.0), HTML(value='')))



Validation loss: 0.5168239900044033
F1 Score: 0.8747685050375633

D:\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels
with no predicted samples.
  'precision', 'predicted', average, warn_for)

HBox(children=(FloatProgress(value=0.0, description='Epoch 8', max=158.0, style=ProgressStyle(d



Epoch 8
Training loss: 0.07162901240436337

HBox(children=(FloatProgress(value=0.0, max=7.0), HTML(value='')))



Validation loss: 0.4332962855696678
F1 Score: 0.9068151300955131

D:\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels
with no predicted samples.
  'precision', 'predicted', average, warn_for)

HBox(children=(FloatProgress(value=0.0, description='Epoch 9', max=158.0, style=ProgressStyle(d



Epoch 9
Training loss: 0.05525129703404028
```

```
HBox(children=(FloatProgress(value=0.0, max=7.0), HTML(value='')))
```

```
Validation loss: 0.44452917150088717
F1 Score: 0.9045074396227775
```

```
D:\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels
with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
HBox(children=(FloatProgress(value=0.0, description='Epoch 10', max=158.0, style=ProgressStyle
```

```
Epoch 10
Training loss: 0.04888481118633777
```

```
HBox(children=(FloatProgress(value=0.0, max=7.0), HTML(value='')))
```

```
Validation loss: 0.4590662739106587
F1 Score: 0.9074067359526623
```

```
D:\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels
with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

10 Model Evaluation

[26]:

```
avg_loss, preds, gts = evaluate(dataloader_val)
accuracy_per_class(preds, gts)
```

```
HBox(children=(FloatProgress(value=0.0, max=7.0), HTML(value='')))
```

```
Class: happy
Accuracy: 170/171

Class: not-relevant
Accuracy: 23/32

Class: angry
```

```
Accuracy: 6/9

Class: disgust
Accuracy: 0/1

Class: sad
Accuracy: 3/5

Class: surprise
Accuracy: 2/5
```

As it can bee seen, the model did very good job in identifying happy sentences. However, in other cases its performance was significantly lower, such as 66% in angry and 40% in sad and surprise, respectively. Regarding disgust class, there was only one example for testing and the model could not predict it correctly.

That's it for this project! Thank you for following!