Transformer for MultiClass Text Classification

We decided to train a BERT (specifically a DistilBert) model to classify articles into 8 different groups.

Libraries

In [ ]:

```
!pip install transformers pytorch_pretrained_bert
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
from os.path import isfile, join
import string
import time
import re
from string import punctuation
import sys

import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import spacy
nlp = spacy.load('en_core_web_sm')
from nltk.tokenize import RegexpTokenizer

from sklearn.datasets import fetch_20newsgroups
from sklearn.preprocessing import LabelEncoder , StandardScaler , MaxAbsScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import FunctionTransformer
from sklearn.base import TransformerMixin
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
from sklearn.feature_extraction.text import CountVectorizer ,TfidfVectorizer
import itertools

import tensorflow as tf
from keras.wrappers.scikit_learn import KerasClassifier
from keras.layers import Dense, Input, Dropout
from keras import Sequential
from keras import metrics

import torch
import transformers
from torch.utils.data import Dataset, DataLoader
from transformers import DistilBertModel, DistilBertTokenizer
```

```
Collecting transformers
  Downloading transformers-4.12.5-py3-none-any.whl (3.1 MB)
     |████████████████████████████████| 3.1 MB 5.4 MB/s
Collecting pytorch_pretrained_bert
  Downloading pytorch_pretrained_bert-0.6.2-py3-none-any.whl (123 kB)
     |████████████████████████████████| 123 kB 22.9 MB/s
Collecting sacremoses
  Downloading sacremoses-0.0.46-py3-none-any.whl (895 kB)
     |████████████████████████████████| 895 kB 38.4 MB/s
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (fro
m transformers) (1.19.5)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from t
ransformers) (2.23.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from t
ransformers) (3.4.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages
```

```
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages
(from transformers) (21.3)
Collecting huggingface-hub<1.0,>=0.1.0
  Downloading huggingface_hub-0.1.2-py3-none-any.whl (59 kB)
     |████████████████████████████████| 59 kB 3.4 MB/s
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-package
s (from transformers) (2019.12.20)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packag
es (from transformers) (4.8.2)
Collecting pyyaml>=5.1
  Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12
_x86_64.manylinux2010_x86_64.whl (596 kB)
     |████████████████████████████████| 596 kB 35.0 MB/s
Collecting tokenizers<0.11,>=0.10.1
  Downloading tokenizers-0.10.3-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylin
ux_2_12_x86_64.manylinux2010_x86_64.whl (3.3 MB)
     |████████████████████████████████| 3.3 MB 31.3 MB/s
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages (from
transformers) (4.62.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.7/dis
t-packages (from huggingface-hub<1.0,>=0.1.0->transformers) (3.10.0.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-
packages (from packaging>=20.0->transformers) (3.0.6)
Collecting boto3
  Downloading boto3-1.20.17-py3-none-any.whl (131 kB)
     |████████████████████████████████| 131 kB 34.1 MB/s
Requirement already satisfied: torch>=0.4.1 in /usr/local/lib/python3.7/dist-packages (fr
om pytorch_pretrained_bert) (1.10.0+cu111)
Collecting botocore<1.24.0,>=1.23.17
  Downloading botocore-1.23.17-py3-none-any.whl (8.4 MB)
     |████████████████████████████████| 8.4 MB 46.4 MB/s
Collecting s3transfer<0.6.0,>=0.5.0
  Downloading s3transfer-0.5.0-py3-none-any.whl (79 kB)
     |████████████████████████████████| 79 kB 7.5 MB/s
Collecting jmespath<1.0.0,>=0.7.1
  Downloading jmespath-0.10.0-py2.py3-none-any.whl (24 kB)
Collecting urllib3<1.27,>=1.25.4
  Downloading urllib3-1.26.7-py2.py3-none-any.whl (138 kB)
     |████████████████████████████████| 138 kB 48.8 MB/s
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/local/lib/python3.7/di
st-packages (from botocore<1.24.0,>=1.23.17->boto3->pytorch_pretrained_bert) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from p
ython-dateutil<3.0.0,>=2.1->botocore<1.24.0,>=1.23.17->boto3->pytorch_pretrained_bert) (1
.15.0)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from
importlib-metadata->transformers) (3.6.0)
  Downloading urllib3-1.25.11-py2.py3-none-any.whl (127 kB)
     |████████████████████████████████| 127 kB 46.4 MB/s
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packag
es (from requests->transformers) (2021.10.8)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-package
s (from requests->transformers) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (fr
om requests->transformers) (2.10)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from sacr
emoses->transformers) (7.1.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from sac
remoses->transformers) (1.1.0)
Installing collected packages: urllib3, jmespath, botocore, s3transfer, pyyaml, tokenizer
s, sacremoses, huggingface-hub, boto3, transformers, pytorch-pretrained-bert
  Attempting uninstall: urllib3
    Found existing installation: urllib3 1.24.3
    Uninstalling urllib3-1.24.3:
      Successfully uninstalled urllib3-1.24.3
  Attempting uninstall: pyyaml
    Found existing installation: PyYAML 3.13
    Uninstalling PyYAML-3.13:
      Successfully uninstalled PyYAML-3.13
ERROR: pip's dependency resolver does not currently take into account all the packages th
at are installed. This behaviour is the source of the following dependency conflicts.
datascience 0.10.6 requires folium==0.2.1, but you have folium 0.8.3 which is incompatibl
e.
Successfully installed boto3-1.20.17 botocore-1.23.17 huggingface-hub-0.1.2 jmespath-0.10
```

```
Successfully installed boto3-1.20.17 botocore-1.23.17 huggingface-hub-0.1.2 jmespath-0.10
.0 pytorch-pretrained-bert-0.6.2 pyyaml-6.0 s3transfer-0.5.0 sacremoses-0.0.46 tokenizers
-0.10.3 transformers-4.12.5 urllib3-1.25.11
```

In [ ]:

```
# Setting up the device for GPU usage

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

Without GPU as Accellerator, the time spent for training the neural network with a single epoch is approximately 4 hours.

Loading the training data and preprocessing

This part of the notebook is almost identical to the one for the first and second point of the assignment (due to compatibility reasons).

We decided to use only 9 types of articles (from the orignal 20). In the code we used the name of the folder in which the article is contained: the name is the category of the article.

In [ ]:

```
# Loading the dataset
dataset = fetch_20newsgroups(subset='train',remove=('headers', 'footers', 'quotes'), shu
ffle=True, random_state=42)
df = pd.DataFrame()
df['text'] = dataset.data
df['source'] = dataset.target

#creation of the label column (type of article)
label=[]
for i in df['source']:
    label.append(dataset.target_names[i])
df['label']=label
df.drop(['source'],axis=1,inplace=True)


# Dictionary to go from 20 categories to 8 macros
key_categories = ['politics','sport','religion','computer','sales','automobile','science
','medicine']
cat_dict = {
**dict.fromkeys(['talk.politics.misc','talk.politics.guns','talk.politics.mideast'],'poli
tics'),
**dict.fromkeys( ['rec.sport.hockey','rec.sport.baseball'],'sport'),
**dict.fromkeys( ['soc.religion.christian','talk.religion.misc'],'religion'),
**dict.fromkeys(['comp.windows.x','comp.sys.ibm.pc.hardware','comp.os.ms-windows.misc','c
omp.graphics','comp.sys.mac.hardware'],'computer'),
**dict.fromkeys( ['misc.forsale'],'sales'),
**dict.fromkeys( ['rec.autos','rec.motorcycles'],'automobile'),
**dict.fromkeys( ['sci.crypt','sci.electronics','sci.space'],'science'),
**dict.fromkeys( ['sci.med'],'medicine')
}
df['label']=df['label'].map(cat_dict)

# Encoding
label_encoder = LabelEncoder()
df['target']= label_encoder.fit_transform(df['label'])

# How many articles do we have for each category?
df['label'].value_counts()
```

Out[ ]:

```
computer      2936
science       1779
politics      1575
sport         1197
```

```
automobile    1192
religion       976
medicine       594
sales          585
Name: label, dtype: int64
```

In [ ]:

```
df.head()
```

Out[ ]:

	text	label	target
0	I was wondering if anyone out there could enli...	automobile	0
1	A fair number of brave souls who upgraded thei...	computer	1
2	well folks, my mac plus finally gave up the gh...	computer	1
3	\nDo you have Weitek's address/phone number? ...	computer	1
4	From article <C5owCB.n3p@world.std.com>, by to...	science	6

Now some data cleaning. First, we imported the libraries. The objects re_url and re_email contain lists of special characters and expressions used almost anywhere.

In [ ]:

```
import re
import string
import pandas as pd
import nltk
nltk.download('stopwords')

from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

re_url = re.compile(r'(?:http|ftp|https)://(?:[\w_-]+(?:(?:\.[\w_-]+)+))(?:[\w.,@?^=%&:/
~+#-]*[\w@?^=%&/~+#-])?')
re_email = re.compile('(?:[a-z0-9!#$%&\'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&\'*+/=?^_`{|}~-]+
)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])
*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:(2(5
[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-9]))\.){3}(?:(2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-
9]?[0-9])|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x
09\x0b\x0c\x0e-\x7f])+)\])')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

This function will remove common headers and espressions with little to none meaning (for example, the last time the article was modified or where it comes from), simplifying the text overall.

In [ ]:

```
def clean_header(text):
    text = re.sub(r'(From:\s+[^\n]+\n)', '', text)
    text = re.sub(r'(Subject:[^\n]+\n)', '', text)
    text = re.sub(r'(([\sA-Za-z0-9\-]+)?[A|a]rchive-name:[^\n]+\n)', '', text)
    text = re.sub(r'(Last-modified:[^\n]+\n)', '', text)
    text = re.sub(r'(Version:[^\n]+\n)', '', text)

    return text
```

clean_text is designed to standardize the text, removing capital letters, unnecessary spaces, replacing "url" etc.

In [ ]:

```
def clean_text(text):
    text = text.lower()
```

```
    text = text.strip()
    text = re.sub(re_url, '', text)
    text = re.sub(re_email, '', text)
    text = re.sub(f'[{re.escape(string.punctuation)}]', '', text)
    text = re.sub(r'(\d+)', ' ', text)
    text = re.sub(r'(\s+)', ' ', text)

    return text
```

Then we applied the functions to the dataset.

In [ ]:

```
df['text'] = df['text'].apply(clean_header)

df['text'] = df['text'].apply(clean_text)

stop_words = stopwords.words('english')

df['text'] = df['text'].str.split() \
    .apply(lambda x: ' '.join([word for word in x if word not in stop_words]))

df=df.dropna() #this function eliminates empty lines
```

Now we can see some improvements:

In [ ]:

```
df.head()
```

Out[ ]:

	text	label	target
0	wondering anyone could enlighten car saw day d...	automobile	0
1	fair number brave souls upgraded si clock osci...	computer	1
2	well folks mac plus finally gave ghost weekend...	computer	1
3	weiteks addressphone number id like get inform...	computer	1
4	article tom baker understanding expected error...	science	6

Preparing the Dataset and Dataloader

Let's define some variables the will be used later during the training phase. We decided to use a BERT model.

- **max_len and batch_size (train and test) are control parameters for the dataloader.**
- **the Triage class is the tokenization of the dataset, required for the dataloader**
- **DataLoader will create training and test (validation) dataloaders, that will give the neural network the data in a controlled way.**

In [ ]:

```
# Defining some key variables that will be used later on in the training
MAX_LEN = 150
TRAIN_BATCH_SIZE = 12
VALID_BATCH_SIZE = 6
EPOCHS = 3
LEARNING_RATE = 5e-05
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-cased')
```

In [ ]:

```
class Triage(Dataset):
```

```
class Triage(Dataset):
    def __init__(self, dataframe, tokenizer, max_len):
        self.len = len(dataframe)
        self.data = dataframe
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __getitem__(self, index):
        title = str(self.data.text[index])
        title = " ".join(title.split())
        inputs = self.tokenizer.encode_plus(
            title,
            None,
            add_special_tokens=True,
            max_length=self.max_len,
            pad_to_max_length=True,
            return_token_type_ids=True,
            truncation=True
        )
        ids = inputs['input_ids']
        mask = inputs['attention_mask']

        return {
            'ids': torch.tensor(ids, dtype=torch.long),
            'mask': torch.tensor(mask, dtype=torch.long),
            'targets': torch.tensor(self.data.target[index], dtype=torch.long)
        }

    def __len__(self):
        return self.len
```

In [ ]:

```
# Creating the dataset and dataloader for the neural network

train_size = 1
train_dataset=df.sample(frac=train_size,random_state=200)
train_dataset = train_dataset.reset_index(drop=True)

print("TRAIN Dataset: {}".format(train_dataset.shape))

training_set = Triage(train_dataset, tokenizer, MAX_LEN)
```

```
TRAIN Dataset: (10834, 3)
```

Finally, the dataloader:

In [ ]:

```
train_params = {'batch_size': TRAIN_BATCH_SIZE,
                'shuffle': True,
                'num_workers': 0
                }

training_loader = DataLoader(training_set, **train_params)
```

Creating the Neural Network for Fine Tuning

Now, let's create a Neural Network for the DistilBert. We have 8 types of articles, a dropout of 30% and a linear classifier.

In [ ]:

```
# Creating the customized model, by adding a drop out and a dense layer on top of distil
bert to get the final output for the model.

class DistillBERTClass(torch.nn.Module):
    def __init__(self):
```

```
        super(DistillBERTClass, self).__init__()
        self.l1 = DistilBertModel.from_pretrained("distilbert-base-uncased")
        self.pre_classifier = torch.nn.Linear(768, 768)
        self.dropout = torch.nn.Dropout(0.3)
        self.classifier = torch.nn.Linear(768, 9)

    def forward(self, input_ids, attention_mask):
        output_1 = self.l1(input_ids=input_ids, attention_mask=attention_mask)
        hidden_state = output_1[0]
        pooler = hidden_state[:, 0]
        pooler = self.pre_classifier(pooler)
        pooler = torch.nn.ReLU()(pooler)
        pooler = self.dropout(pooler)
        output = self.classifier(pooler)
        return output
```

In [ ]:

```
model = DistillBERTClass()
model.to(device)
```

```
Some weights of the model checkpoint at distilbert-base-uncased were not used when initia
lizing DistilBertModel: ['vocab_transform.weight', 'vocab_layer_norm.weight', 'vocab_proj
ector.bias', 'vocab_projector.weight', 'vocab_layer_norm.bias', 'vocab_transform.bias']
- This IS expected if you are initializing DistilBertModel from the checkpoint of a model
trained on another task or with another architecture (e.g. initializing a BertForSequence
Classification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing DistilBertModel from the checkpoint of a m
odel that you expect to be exactly identical (initializing a BertForSequenceClassificatio
n model from a BertForSequenceClassification model).
```

Out[ ]:

```
DistillBERTClass(
  (l1): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0): TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
        (1): TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
```

```
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      )
      (2): TransformerBlock(
        (attention): MultiHeadSelfAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      )
      (3): TransformerBlock(
        (attention): MultiHeadSelfAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      )
      (4): TransformerBlock(
        (attention): MultiHeadSelfAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      )
      (5): TransformerBlock(
        (attention): MultiHeadSelfAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn): FFN(
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      )
    )
  )
)
```

```
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (dropout): Dropout(p=0.3, inplace=False)
  (classifier): Linear(in_features=768, out_features=9, bias=True)
)
```

Then loss and optimizer function (minimizes the loss)

In [ ]:

```
# Creating the loss function and optimizer
loss_function = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params =  model.parameters(), lr=LEARNING_RATE)
```

Fine Tuning the Model

In [ ]:

```
# Function to calcuate the accuracy of the model

def calcuate_accu(big_idx, targets):
    n_correct = (big_idx==targets).sum().item()
    return n_correct
```

In [ ]:

```
# Defining the training function on the 80% of the dataset for tuning the distilbert mode
l

def train(epoch):
    tr_loss = 0
    n_correct = 0
    nb_tr_steps = 0
    nb_tr_examples = 0
    model.train()
    for _,data in enumerate(training_loader, 0):
        ids = data['ids'].to(device, dtype = torch.long)
        mask = data['mask'].to(device, dtype = torch.long)
        targets = data['targets'].to(device, dtype = torch.long)

        outputs = model(ids, mask)
        loss = loss_function(outputs, targets)
        tr_loss += loss.item()
        big_val, big_idx = torch.max(outputs.data, dim=1)
        n_correct += calcuate_accu(big_idx, targets)

        nb_tr_steps += 1
        nb_tr_examples+=targets.size(0)

        if _%100==0:
          loss_step = tr_loss/nb_tr_steps
          accu_step = (n_correct*100)/nb_tr_examples
          #print(f"Training Loss per step {nb_tr_steps}: {loss_step}")
          print(f"Training Accuracy per step {nb_tr_steps}: {accu_step}")

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f'The Total Accuracy for Epoch {epoch}: {(n_correct*100)/nb_tr_examples}')
    epoch_loss = tr_loss/nb_tr_steps
    epoch_accu = (n_correct*100)/nb_tr_examples
    print(f"Training Loss Epoch: {epoch_loss}")
    print(f"Training Accuracy Epoch: {epoch_accu}")

    return
```

In [ ]:

```
for epoch in range(EPOCHS):
    train(epoch)
```

```
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2218: 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,
```

```
Training Accuracy per step 1: 16.666666666666668
Training Accuracy per step 101: 25.0
Training Accuracy per step 201: 26.65837479270315
Training Accuracy per step 301: 30.564784053156146
Training Accuracy per step 401: 33.85286783042394
Training Accuracy per step 501: 37.82435129740519
Training Accuracy per step 601: 41.098169717138106
Training Accuracy per step 701: 43.99667142177841
Training Accuracy per step 801: 46.36912193091968
Training Accuracy per step 901: 48.816130225675174
The Total Accuracy for Epoch 0: 48.86468525013845
Training Loss Epoch: 1.4331750190535257
Training Accuracy Epoch: 48.86468525013845
Training Accuracy per step 1: 83.33333333333333
Training Accuracy per step 101: 70.95709570957095
Training Accuracy per step 201: 72.30514096185738
Training Accuracy per step 301: 72.92358803986711
Training Accuracy per step 401: 74.02327514546965
Training Accuracy per step 501: 74.46773120425814
Training Accuracy per step 601: 74.88907376594564
Training Accuracy per step 701: 75.21398002853067
Training Accuracy per step 801: 75.24968789013732
Training Accuracy per step 901: 75.54568997410284
The Total Accuracy for Epoch 1: 75.53073657005723
Training Loss Epoch: 0.733214638566192
Training Accuracy Epoch: 75.53073657005723
Training Accuracy per step 1: 100.0
Training Accuracy per step 101: 85.31353135313532
Training Accuracy per step 201: 84.78441127694859
Training Accuracy per step 301: 85.8250276854928
Training Accuracy per step 401: 85.88944305901911
Training Accuracy per step 501: 85.89487691284099
Training Accuracy per step 601: 85.8014420410427
Training Accuracy per step 701: 86.04374702805517
Training Accuracy per step 801: 86.090303786933
Training Accuracy per step 901: 86.16352201257861
The Total Accuracy for Epoch 2: 86.17315857485693
Training Loss Epoch: 0.41437769726389073
Training Accuracy Epoch: 86.17315857485693
```

Validating the Model

Now let's test the data on the test subset (which was not used during training).

Let's download the data first:

In [ ]:

```
# Loading the dataset
dataset = fetch_20newsgroups(subset='test',remove=('headers', 'footers', 'quotes'), shuffle=True, random_state=42)
df_test= pd.DataFrame()
df_test['text'] = dataset.data
df_test['source'] = dataset.target

#creation of the label column (type of article)
label=[]
for i in df_test['source']:
    label.append(dataset.target_names[i])
```

```
df_test['label']=label
df_test.drop(['source'],axis=1,inplace=True)


# Dictionary to go from 20 categories to 9 macros
key_categories = ['politics','sport','religion','computer','sales','automobile','science
','medicine']
cat_dict = {
**dict.fromkeys(['talk.politics.misc','talk.politics.guns','talk.politics.mideast'],'poli
tics'),
**dict.fromkeys( ['rec.sport.hockey','rec.sport.baseball'],'sport'),
**dict.fromkeys( ['soc.religion.christian','talk.religion.misc'],'religion'),
**dict.fromkeys(['comp.windows.x','comp.sys.ibm.pc.hardware','comp.os.ms-windows.misc','c
omp.graphics','comp.sys.mac.hardware'],'computer'),
**dict.fromkeys( ['misc.forsale'],'sales'),
**dict.fromkeys( ['rec.autos','rec.motorcycles'],'automobile'),
**dict.fromkeys( ['sci.crypt','sci.electronics','sci.space'],'science'),
**dict.fromkeys( ['sci.med'],'medicine')
}
df_test['label']=df_test['label'].map(cat_dict)

# Encoding
label_encoder = LabelEncoder()
df_test['target']= label_encoder.fit_transform(df_test['label'])

# How many articles do we have for each category?
df_test['label'].value_counts()
```

Out[ ]:

```
computer      1955
science       1183
politics      1050
sport          796
automobile     794
religion       649
medicine       396
sales          390
Name: label, dtype: int64
```

We decided to balance the test dataset in order to use the accuracy as an estimate of the goodness the model.

In [ ]:

```
def downsample(df):
    minority_frequency  = df['label'].value_counts()[-1]
    minority_label = df['label'].value_counts().index[-1]

    df_balanced = df.loc[df['label'] == minority_label , : ].sample(minority_frequency).
copy()
    df_balanced = df_balanced.reset_index(drop = True)

    label_list = df['label'].value_counts().index.tolist()
    #Sample and concat
    for label in label_list:
        if label != minority_label:
            sample_df = df.loc[df['label'] == label , : ].sample(minority_frequency).cop
y()
            df_balanced = pd.concat([ df_balanced , sample_df],axis = 0 , ignore_index=T
rue)
    # Shuffle data
    df_balanced = df_balanced.sample(frac = 1).reset_index(drop = True)

    return df_balanced
```

Now, let's balance and pre-process the dataset and then let's create the appropriate dataloader.

In [ ]:

```
df_test = downsample(df_test) #balancing
```

```
df_test['text'] = df_test['text'].apply(clean_header)
df_test['text'] = df_test['text'].apply(clean_text)
df_test['text'] = df_test['text'].str.split() \
    .apply(lambda x: ' '.join([word for word in x if word not in stop_words]))

df_test=df_test.dropna()

test_dataset=df_test.reset_index(drop=True)

testing_set = Triage(test_dataset, tokenizer, MAX_LEN)

test_params = {'batch_size': VALID_BATCH_SIZE,
               'shuffle': True,
               'num_workers': 0
               }

testing_loader = DataLoader(testing_set, **test_params)
```

The testing function:

In [ ]:

```
def valid(model, testing_loader):
    model.eval()
    n_correct = 0; n_wrong = 0; total = 0;
    y_pred= []; y=[]
    with torch.no_grad():
        for _, data in enumerate(testing_loader, 0):
            ids = data['ids'].to(device, dtype = torch.long)
            mask = data['mask'].to(device, dtype = torch.long)
            targets = data['targets'].to(device, dtype = torch.long)
            outputs = model(ids, mask).squeeze().to(device, dtype = torch.long)
            big_val, big_idx = torch.max(outputs.data, dim=1)
            total+=targets.size(0)
            n_correct+=(big_idx==targets).sum().item()
    return (n_correct*100.0)/total
```

At last, the results: almost 70% of accuracy.

In [ ]:

```
acc = valid(model, testing_loader)
print("Accuracy on test data = %0.2f%%" % acc)
```

```
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2218: 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,
```

```
Accuracy on test data = 67.53%
```