Introduction

I love Asana, I'm PM and they are product company. I like data science. I wanna explore Asana and it's competitors. I also want to play with and learna about the BERT model. This will be a curiosity-driven journey, not sure where it'll end.

Packages

```
# Package to store the versions of packages used
!pip install -q watermark
```

```
# Package to download the BERT models and process data
!pip install -q transformers
```

```
# Package for scrapping data on Google Store
# https://pypi.org/project/google-play-scraper/
!pip install -q google_play_scraper
```

```
# File manipulation imports for Google Colab
from google.colab import drive
drive.mount('/content/drive')
import os
os.chdir("/content/drive/My Drive/Colab Notebooks/BERT_App_Sentiment_Analysis")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

```
# Imports

# Data manipulation and visualization
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
from pylab import rcParams
from matplotlib import rc
from tqdm import tqdm
import datetime


# Deep Learning, NLP and metrics
```

```
import sklearn
import torch
import transformers
from textwrap import wrap
from torch import nn, optim
from torch.utils import data
from collections import defaultdict
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from transformers import BertModel
from transformers import BertTokenizer
from transformers import AdamW
from transformers import get_linear_schedule_with_warmup

# Web Scrapping Imports
# https://pypi.org/project/Pygments/
import json
import pygments
import google_play_scraper
from pygments import highlight
from pygments.lexers import JsonLexer
from pygments.formatters import TerminalFormatter

# Random Seed
#RANDOM_SEED = 99
#np.random.seed(RANDOM_SEED)
#torch.manual_seed(RANDOM_SEED)

%matplotlib inline
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
  import pandas.util.testing as tm
```

```
# Package versions
%reload_ext watermark
%watermark -v -iv
```

```
pygments            2.1.3
torch               1.5.1+cu101
google_play_scraper 0.0.3.0
pandas              1.0.5
numpy               1.18.5
sklearn             0.22.2.post1
seaborn             0.10.1
json                2.0.9
transformers        3.0.2
matplotlib          3.2.2
CPython 3.6.9
IPython 5.5.0
```

Web Scrapping

```
# Listing apps I want to gather data on
# They'll all be Asana's competitors on task management
# Took the apps from Asana's comparison page, plus a few other alternatives the app store rec
# https://asana.com/compare
# Asana, Airtable, Basecamp, Jira, Microsoft To Do
# Monday.com, Smartsheet, Taskade, Trello, Wrike
# The google_play_scrapper documentations details how to get the url for each app
# https://github.com/facundoolano/google-play-scraper
apps_list = ['com.asana.app',
             'com.formagrid.airtable',
             'com.basecamp.bc3',
             'com.atlassian.android.jira.core',
             'com.microsoft.todos',
             'com.monday.monday',
             'com.smartsheet.android',
             'com.taskade.mobile',
             'com.trello',
             'com.wrike']
```

```
# List to store details from the apps
app_details = []

# Loop through the app list and retrieve details of each app
for ap in tqdm(apps_list):

    # Retrieve app details
    info = google_play_scraper.app(ap, lang='en', country='us')

    # Store the details
    app_details.append(info)
```

```
100%|██████████| 10/10 [00:01<00:00,  7.70it/s]
```

```
# Function to print a request in JSON format
def print_json(json_object):

    # Generate json format
    json_str = json.dumps(json_object,
                          indent = 2,
                          sort_keys = True,
                          default = str)

    # The highlight function from pygments highlights the output text
    # It uses different colorts to facilitate reading
    print(highlight(json_str, JsonLexer(), TerminalFormatter()))
```

```
# Check the result in JSON format
print_json(app_details[0])
```

```
  "contentRating": "Everyone",
  "contentRatingDescription": null,
  "currency": "USD",
  "description": "Asana is the work manager for teams. But better. From the small stuff
  "descriptionHTML": "Asana is the work manager for teams. But better. From the small st
  "developer": "Asana, Inc.",
  "developerAddress": null,
  "developerEmail": "support@asana.com",
  "developerId": "Asana,+Inc.",
  "developerInternalID": "9027419648812383370",
  "developerWebsite": "https://asana.com/product",
  "free": true,
  "genre": "Business",
  "genreId": "BUSINESS",
  "headerImage": "https://lh3.googleusercontent.com/4ts1ELx9Kpks2R2KWE_hCTBW63gVqR5UrSg
  "histogram": [
    1434,
    478,
    1278,
    3222,
    24885
  ],
  "icon": "https://lh3.googleusercontent.com/EJEviNAy8fAdCNMrcxaZDYLH1AnDnvficaxztxPnEF
  "installs": "1,000,000+",
  "minInstalls": 1000000,
  "offersIAP": null,
  "originalPrice": null,
  "price": 0,
  "privacyPolicy": "http://www.asana.com/privacy",
  "ratings": 31299,
  "recentChanges": "\ud83c\udfb5 Give a little bit...\r\nGive a little bit of appreciat
  "recentChangesHTML": "\ud83c\udfb5 Give a little bit...<br>Give a little bit of appre
  "released": "Feb 27, 2013",
  "reviews": 9591,
  "sale": false,
  "saleText": null,
  "saleTime": null,
  "score": 4.586184,
  "screenshots": [
    "https://lh3.googleusercontent.com/a-c_cZ7cTlTHgMmXuG-BqsN6-xm0s_koN56J9_jRhVgd81HS
    "https://lh3.googleusercontent.com/ZBeNcL0KBzHLkZvmN9TGohTZ1EBdPoQ0BEnBs4eEiAtpZcgR
    "https://lh3.googleusercontent.com/YQzlPY-Gf0IZcJ23dmX-2WZRt1Sf-xh7d8QteyxVXuUTBXA
    "https://lh3.googleusercontent.com/Od0HDyc248fg5ya7y3b7BcSHz8P-_eQVGqvnln3KJxXRwoSB
    "https://lh3.googleusercontent.com/RIEE8eAwXQLotr3jRF0bim47WoGYJ_Iu3W8alWSOnEiImvQe
    "https://lh3.googleusercontent.com/WDGYFWrU3MMIwoaHqggYVQT2bCTH4OitaL94oZZcY6pO3CfN
    "https://lh3.googleusercontent.com/k8TYhLxOalU6RPSQFt_8QnUDwBZTcE2UD1CkmQ09T3QYmH7b
  ],
  "size": "14M",
  "summary": "Organize. Plan. Get work done. #withAsana",
  "summaryHTML": "Organize. Plan. Get work done. #withAsana",
  "title": "Asana: Your work manager",
  "updated": 1595265487,
  "url": "https://play.google.com/store/apps/details?id=com.asana.app&hl=en&gl=us",
  "version": "6.50.8",
  "video": "https://www.youtube.com/embed/jY0-gsNImlk?ps=play&vq=large&rel=0&autohide=1
```

```
# Put the retrieved information into a dataframe
```

```
df_app_details = pd.DataFrame(app_details)
```

```
# Save the dataframe to disk

# Retrieve datetime to stamp the file
now = datetime.datetime.now().strftime("%Y-%m-%d-%H-%M-%S")

# Save with current datetime
df_app_details.to_csv(f'data/app_details_{now}.csv', header=True, index=None)
```

```
df_app_details.head(3)
```

	title	description	descriptionHTML	summary	summaryHTML	installs	minInsta
0	Asana: Your work manager	Asana is the work manager for teams. But bette...	Asana is the work manager for teams. But bette...	Organize. Plan. Get work done. #withAsana	Organize. Plan. Get work done. #withAsana	1,000,000+	1000
1	Airtable	Organize anything you can imagine with Airtabl...	Organize anything you can imagine with Airtabl...	Organize anything you can imagine with a moder...	Organize anything you can imagine with a moder...	100,000+	100
2	Basecamp 3	<b>Use your company's Basecamp 3 account on-th...	<b>Use your company&#39;s Basecamp 3 account o...	Basecamp 3, official Android version for the w...	Basecamp 3, official Android version for the w...	500,000+	500

```
# List to store app reviews
app_reviews = []

# Loop to retrieve and store app reviews
for ap in tqdm(apps_list):

    # Extract sample reviews from reviews with different stars given
    for star in list(range(1, 6)):

        # Extract the most relevant and the most recent reviews
        for sort_order in [google_play_scraper.Sort.MOST_RELEVANT, google_play_scraper.Sort.N
            rvws, _ = google_play_scraper.reviews(ap,
                                                  lang='en',
                                                  country='us',
                                                  sort=sort_order,
                                                    count = 100 if star == 3 else 50,
                                                    filter_score_with = star)

            for r in rvws:
                r['sortOrder'] = 'most_relevant' if sort_order == google_play_scraper.Sort.MO
                r['appId'] = ap

            # Save reviews
            app_reviews.extend(rvws)
```

```
100%|██████████| 10/10 [00:09<00:00,  1.03it/s]
```

```
# Create a dataframe with the reviews
df_app_reviews = pd.DataFrame(app_reviews)
```

```
# Save the dataframe to disk

# Retrieve datetime to stamp the file
now = datetime.datetime.now().strftime("%Y-%m-%d-%H-%M-%S")

# Save with current datetime
df_app_reviews.to_csv(f'data/app_reviews_{now}.csv', header = True, index = None)
```

```
# Loading the csv with app reviews
df_reviews = pd.read_csv(f'data/app_reviews_{now}.csv')
df_reviews.head(3)
```

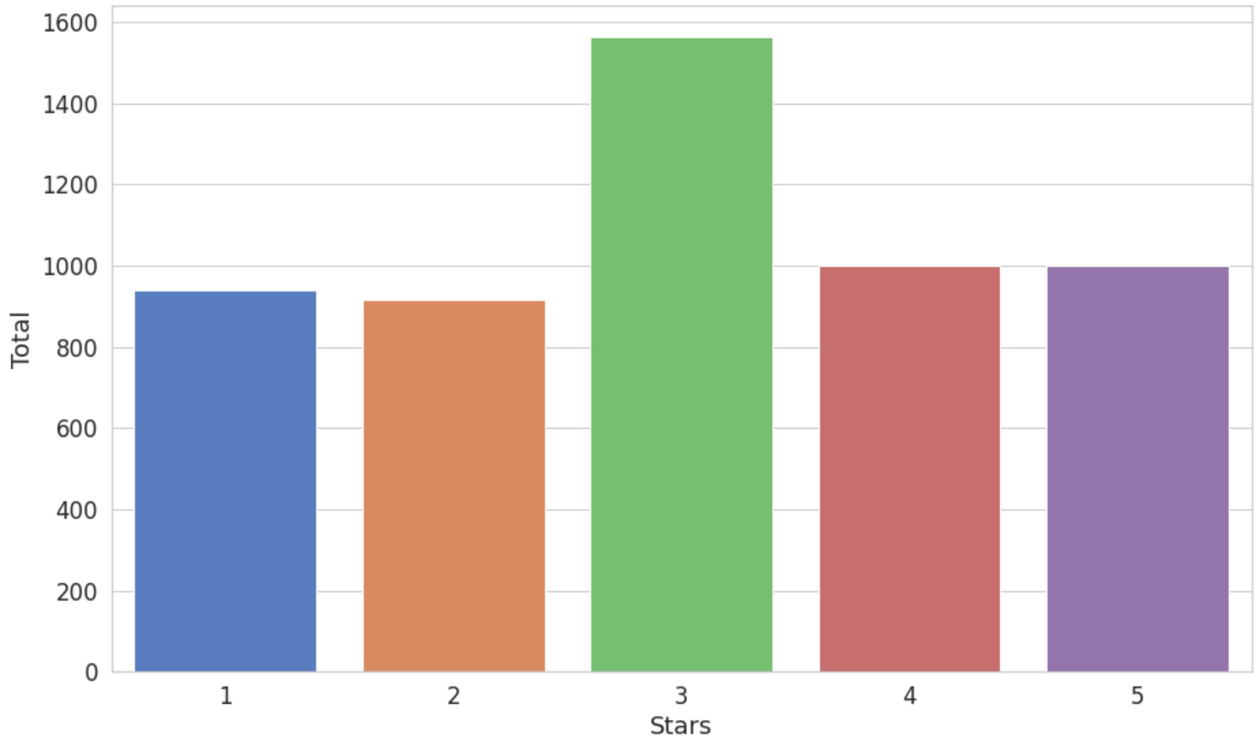
	reviewId	userName	
0	gp:AOqpTOFrr-TuGFhU9WiSFNSqYKYrn8ZvUiUZ-ljdC8O...	Erlang P	https://lh3.google
1	gp:AOqpTOGjmtD5IJRa-8Rk7hxS02RFs1oyJgdDwFOXbsj...	mrk 1	https://lh3.googleuse

```
df_reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5418 entries, 0 to 5417
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   reviewId              5418 non-null   object
 1   userName              5418 non-null   object
 2   userImage             5418 non-null   object
 3   content               5417 non-null   object
 4   score                 5418 non-null   int64 
 5   thumbsUpCount         5418 non-null   int64 
 6   reviewCreatedVersion  4852 non-null   object
```

```
# Plot stars
sns.set(style = 'whitegrid', palette = 'muted', font_scale = 1.5)
rcParams['figure.figsize'] = 15, 9
sns.countplot(df_reviews.score)
plt.xlabel('Stars')
plt.ylabel('Total')
```

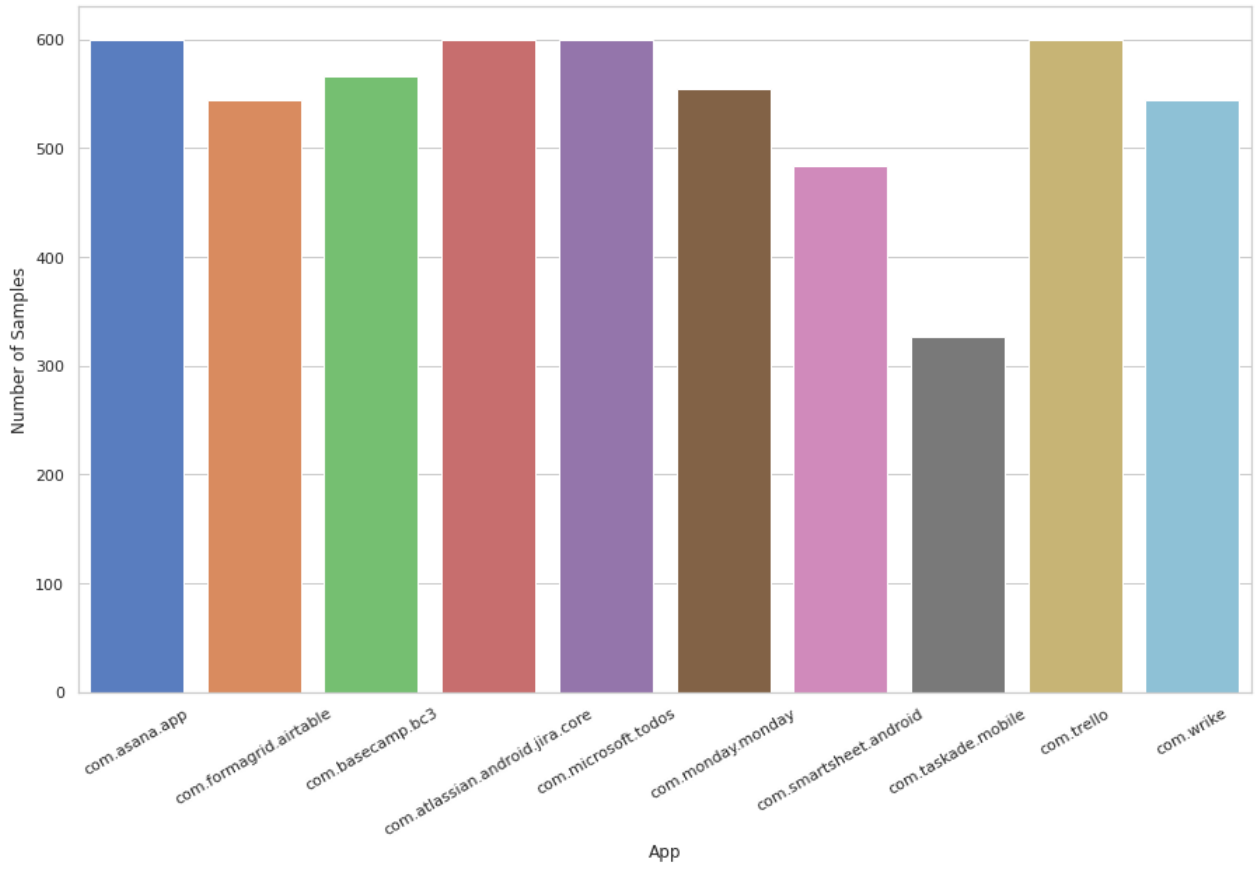
```
Text(0, 0.5, 'Total')
```



```
# Plot appId
```

```
sns.set(style = 'whitegrid', palette = 'muted', font_scale = 1)
rcParams['figure.figsize'] = 15, 9
ax = sns.countplot(df_reviews.appId)
ax.set_xticklabels(ax.get_xticklabels(),rotation=30)
plt.xlabel('App')
plt.ylabel('Number of Samples')
```

```
Text(0, 0.5, 'Number of Samples')
```

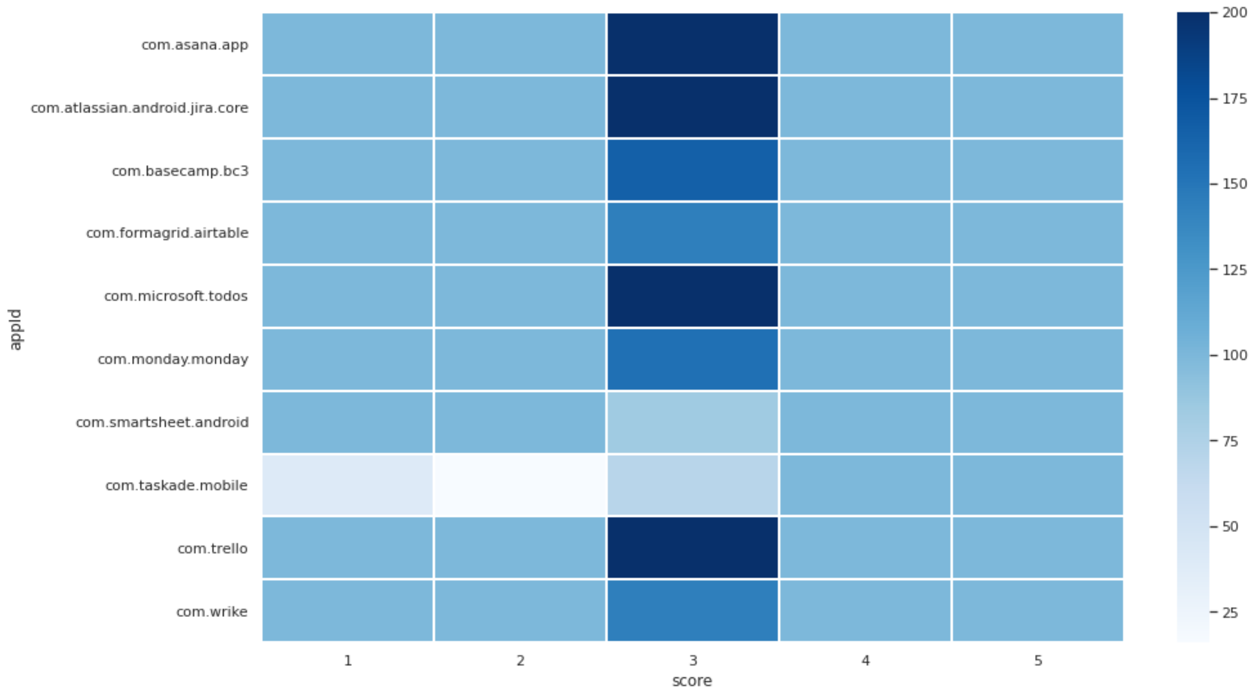


```
# Creating a pivot table to see which app x star combination didn't retrieve the desired amou
app_x_stars = df_reviews.groupby(['appId', 'score']).size().unstack()
app_x_stars
```

score	1	2	3	4	5
appId					
com.asana.app	100	100	200	100	100
com.atlassian.android.jira.core	100	100	200	100	100
com.basecamp.bc3	100	100	166	100	100
com.formagrid.airtable	100	100	144	100	100
com.microsoft.todos	100	100	200	100	100
com.monday.monday	100	100	154	100	100
com.smartsheet.android	100	100	84	100	100

```
# Plotting app x stars as a heatmap
sns.heatmap(app_x_stars, linewidths=1, linecolor='white', cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd3929e63c8>
```



Preprocessing

```
# Grouping function
# This will convert range of 1-5 star reviews into negative(0), neutral(1) and positive(2)
# This is why I've gathered twice as much data for 3 star reviews
def group_rating(rating):

    # initialize groups on -1 to catch any bugs
    grp_rating = -1

    # Convert ratings to integers
    rating = int(rating)

    # If the rating is above 3, then positive (2)
    if rating > 3:
        grp_rating = 2

    # If rating is 3, then neutral (1)
    elif rating == 3:
        grp_rating = 1

    # If rating is below 3, then negative (0)
    else:
        grp_rating = 0

    return grp_rating
```

```
# Apply the function to the dataset and create a 'sentiment' column with the output
df_reviews['sentiment'] = df_reviews.score.apply(group_rating)
```

```
df_reviews.head(3)
```

reviewId **userName**

```
# Shuffling the dataframe to avoid biasing the model later on
df_reviews = df_reviews.sample(frac=1).reset_index(drop=True)
```

https://lh3.google

```
# List with class names
class_names = ['negative', 'neutral', 'positive']
```

```
print(f'Negative: {(len(df_reviews[df_reviews.sentiment == 0])/len(df_reviews))}')
print(f'Neutral: {(len(df_reviews[df_reviews.sentiment == 1])/len(df_reviews))}')
print(f'Positive: {(len(df_reviews[df_reviews.sentiment == 2])/len(df_reviews))}')
```

```
Negative: 0.34256183093392395
Neutral: 0.2882982650424511
Positive: 0.36913990402362495
```

```
# Plot class distribution
sns.set(style = 'whitegrid', palette = 'muted', font_scale = 1.5)
rcParams['figure.figsize'] = 15, 9
sns.countplot(df_reviews.sentiment)
plt.xlabel('Class')
plt.ylabel('Total')
```

```
Text(0, 0.5, 'Total')
```

Downloading the pre-treined BERT model.

List of available models: https://github.com/google-research/bert

```
# Model download
tokenizer = transformers.BertTokenizer.from_pretrained('bert-base-cased')
```

```
# Test text
test_text = 'Just a test sentence. Test 2.'
test_text
```

```
'Just a test sentence. Test 2.'
```

```
# Tokenize
tokens = tokenizer.tokenize(test_text)
tokens
```

```
['Just', 'a', 'test', 'sentence', '.', 'Test', '2', '.']
```

```
# Extract the token_ids
token_ids = tokenizer.convert_tokens_to_ids(tokens)
token_ids
```

```
[2066, 170, 2774, 5650, 119, 5960, 123, 119]
```

```
# Create the encoding object to format the data for the BERT model
encoding = tokenizer.encode_plus(test_text,
                                 max_length = 32,
                                 add_special_tokens = True,
                                 pad_to_max_length = True,
                                 return_attention_mask = True,
                                 return_token_type_ids = False,
                                 return_tensors = 'pt')
```

```
Truncation was not explicitely activated but `max_length` is provided a specific value,
```

```
# Print
encoding
```

```
{'input_ids': tensor([[ 101, 2066,  170, 2774, 5650,  119, 5960,  123,  119,  102,    0,
```

Applying the BERT tokenizer to the dataset

```
# List for the tokens
token_length = []
```

```
# Drop NaN values before tokenizing
df_reviews = df_reviews.dropna(subset=['content'], how='all')
df_reviews.reset_index(inplace = True, drop=True)
df_reviews.shape
```

```
(5417, 13)
```

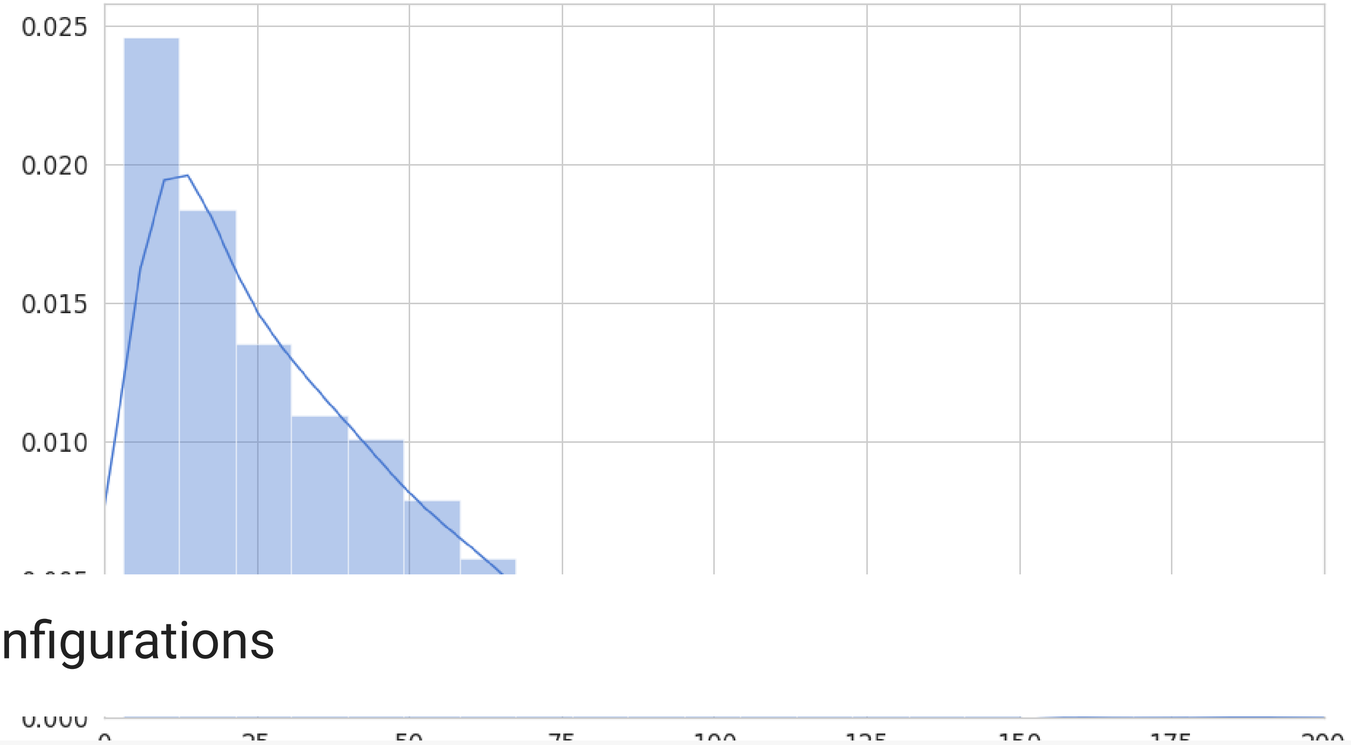
```
# Loop through the dataset content applying the tokenizer
for content in df_reviews.content:
    tokens = tokenizer.encode(content)
    token_length.append(len(tokens))
```

```
# Sample of contents
df_reviews.content.tail(5)
```

```
5412    My office just signed up for the website versi...
5413    I like the app, but I have to give it one star...
5414    While the interface is very clean. I have few ...
5415                                 Still learning this one
5416    I like to have all tge features its website ha...
Name: content, dtype: object
```

```
# Plot
ax = sns.distplot(token_length)
plt.xlim([0, 200])
plt.xlabel('Token Length')
```

```
Text(0.5, 0, 'Token Length')
```



Configurations

```
# Model Hyperparameters
EPOCHS = 10
BATCH_SIZE = 16
MAX_LENGTH = 150
LEARNING_RATE = 0.00002
'''
pent about 7 hours debugging this model to find out that the learning rate
has to be precisely 2e^-5 as anything else was cause the model not to learn at all
'''
```

```
'\npent about 7 hours debugging this model to find out that the learning rate\nhas to b
e precisely 2e^-5 as anything else was cause the model not to learn at all\n'
```

Data Batching

```
class DataBatcher(data.Dataset):

    # Constructor
    def __init__(self, review, targets, tokenizer, max_len):

        # Initialize class atributes
        self.review = review
        self.targets = targets
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.review)
    # Method to obtain each review
    def __getitem__(self, item):

        # Load a review
        review = str(self.review[item])

        # Create the review embedding
        encoding = tokenizer.encode_plus(review,
                                         max_length = self.max_len,
                                         truncation=True,
                                         add_special_tokens = True,
                                         pad_to_max_length = True,
                                         return_attention_mask = True,
                                         return_token_type_ids = False,
                                         return_tensors = 'pt')

        # Among the methods returns, there is the attention mask
        return {'review_text': review,
                'input_ids': encoding['input_ids'].flatten(),
                'attention_mask': encoding['attention_mask'].flatten(),
                'targets': torch.tensor(self.targets[item], dtype = torch.long)}
```

```
# This function creates a data loader to convert the dataset to the BERT format
# torch.utils.data.dataloader.DataLoader
def create_data_loader(df, tokenizer, max_len, batch_size):
    ds = DataBatcher(review = df.content.to_numpy(),
                     targets = df.sentiment.to_numpy(),
                     tokenizer = tokenizer,
                     max_len = max_len)

    return data.DataLoader(ds, batch_size = batch_size, num_workers = 4)
```

```
# Train test split
df_train, df_test = train_test_split(df_reviews, test_size = 0.2) #, random_state = RANDOM_SE
```

```
# Test validation split
df_valid, df_test = train_test_split(df_test, test_size = 0.5) #, random_state = RANDOM_SEED
```

```
print(f'df_train.shape: {df_train.shape}')
print(f'df_test.shape: {df_test.shape}')
print(f'df_valid.shape: {df_valid.shape}')
```

```
df_train.shape: (4333, 13)
df_test.shape: (542, 13)
df_valid.shape: (542, 13)
```

```
# Load the data_loaders
train_data_loader = create_data_loader(df_train, tokenizer, MAX_LENGTH, BATCH_SIZE)
```

```
test_data_loader = create_data_loader(df_test, tokenizer, MAX_LENGTH, BATCH_SIZE)
valid_data_loader = create_data_loader(df_valid, tokenizer, MAX_LENGTH, BATCH_SIZE)
```

```
# Visualize a sample on the training data
sample = next(iter(train_data_loader))
print(sample['input_ids'].shape)
print(sample['attention_mask'].shape)
print(sample['targets'].shape)
```

```
torch.Size([16, 150])
torch.Size([16, 150])
torch.Size([16])
```

```
# Single review sample already on BERT format
print(sample)
```

```
{'review_text': ['I would literally be lost without this. I manage a busy student clinic
        [  101,   113,   122,  ...,     0,     0,     0],
        [  101,  2156,  1136,  ...,     0,     0,     0],
        ...,
        [  101,  8762, 25764,  ...,     0,     0,     0],
        [  101,  1135,   112,  ...,     0,     0,     0],
        [  101,  2038, 12647,  ...,     0,     0,     0]]), 'attention_mask': tensor([[1
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        ...,
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0]]), 'targets': tensor([2, 0, 1, 2, 2, 2, 0, 2, 1, 2, 1,
```

Model

```
# Loading the pre-trained BERT model
model_bert = BertModel.from_pretrained('bert-base-cased')
```

```
# Model
model_bert
```

```
BertModel(
  (embeddings): BertEmbeddings(
    (word_embeddings): Embedding(28996, 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)
```

```
# Visualize the shape of the last dense layer and the last pooling layer
last_hidden_state, pooled_output = model_bert(input_ids = encoding['input_ids'], attention_ma
```

```
    )
```

```
last_hidden_state.shape
```

```
torch.Size([1, 32, 768])
)
```

```
pooled_output.shape
```

```
torch.Size([1, 768])
```

Adding the layers relative to my specific model.

Only those get trained in practice.

```
class SentimentClassifier(nn.Module):

    # Constructor
    def __init__ (self, n_classes):

        # Initialize atributes
        super(SentimentClassifier, self).__init__()

        # Define the pre-trained BERT model
        self.bert = BertModel.from_pretrained('bert-base-cased')

        # Add a dropout layer
        self.drop1 = nn.Dropout()

        # Add a hidden layer
        self.fc1 = nn.Linear(self.bert.config.hidden_size, 100)

        # Add a dense layer
        self.fc2 = nn.Linear(100, n_classes)

        # Final classification with softmax
        self.softmax = nn.Softmax(dim = 1)

    # Forward method
    def forward(self, input_ids, attention_mask):

        # Load the pooling layer from BERT
        _, pooled_output = self.bert(input_ids = input_ids, attention_mask = attention_mask)

        # Define the outputs from the created layers
        output = self.drop1(pooled_output)
        output = self.fc1(output)
        output = self.fc2(output)

        # Return
        return self.softmax(output)
```

```
# Setting the device to GPU
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
device
```

```
device(type='cuda', index=0)
```

```
# Create instance of the model
model_sentiment_classifier = SentimentClassifier(len(class_names))
```

```
# Send model to the device
model_sentiment_classifier = model_sentiment_classifier.to(device)
```

```
# Load the inputs and attention mask
input_ids = sample['input_ids'].to(device)
attention_mask = sample['attention_mask'].to(device)
```

```
# Print
print(input_ids.shape)
print(attention_mask.shape)
```

```
torch.Size([16, 150])
torch.Size([16, 150])
```

```
# Load the inputs and attention mask onto the model
model_sentiment_classifier(input_ids, attention_mask)
```

```
tensor([[0.3765, 0.3467, 0.2767],
        [0.2645, 0.3171, 0.4184],
        [0.2249, 0.2132, 0.5619],
        [0.2705, 0.2744, 0.4551],
        [0.3308, 0.3413, 0.3279],
        [0.2804, 0.3767, 0.3429],
        [0.4921, 0.2083, 0.2996],
        [0.3243, 0.3499, 0.3258],
        [0.2173, 0.3410, 0.4417],
        [0.3276, 0.3341, 0.3383],
        [0.3849, 0.3434, 0.2717],
        [0.4063, 0.2403, 0.3533],
        [0.4087, 0.3232, 0.2681],
        [0.3955, 0.1935, 0.4110],
        [0.1764, 0.2537, 0.5699],
        [0.3228, 0.2547, 0.4225]], device='cuda:0', grad_fn=<SoftmaxBackward>)
```

```
# The original BERT model uses AdamW: algorithm with fixed decay weight
optimizer = AdamW(model_sentiment_classifier.parameters(), lr = LEARNING_RATE, correct_bias =
```

```
# Defining the total number of steps
total_step = len(train_data_loader) * EPOCHS
```

```
# Adjust the learning rate
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps = 0, num_training_ste
```

```
# Loss function
loss_fn = nn.CrossEntropyLoss().to(device)
#loss_fn = nn.NLLLoss().to(device)
#loss_fn = nn.MultiMarginLoss().to(device)
```

```
# Train function
def train_model(model, data_loader, loss_fn, optimizer, device, scheduler, n_examples):

    # Prepare for training
    model = model.train()
    losses = []
    correct_prediction = 0

    # Loop through the data samples
    # Complete Deep Learing cicle
    for d in data_loader:
        input_ids = d['input_ids'].to(device)
        attention_mask = d['attention_mask'].to(device)
        targets = d['targets'].to(device)
        outputs = model(input_ids = input_ids, attention_mask = attention_mask)

        _, preds = torch.max(outputs, dim = 1)
        loss = loss_fn(outputs, targets)

        correct_prediction += torch.sum(preds == targets)
        losses.append(loss.item())

        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), max_norm = 1.0)
        optimizer.step()
        scheduler.step()
        optimizer.zero_grad()

    return correct_prediction.double() / n_examples, np.mean(losses)
```

```
# Evaluate function
def evaluate_model(model, data_loader, loss_fn, device, n_examples):

    model.eval()
    losses = []
    correct_prediction = 0

    with torch.no_grad():
        for d in data_loader:
            input_ids = d['input_ids'].to(device)
            attention_mask = d['attention_mask'].to(device)
            targets = d['targets'].to(device)
            outputs = model(input_ids = input_ids, attention_mask = attention_mask)

            _, preds = torch.max(outputs, dim = 1)
            loss = loss_fn(outputs, targets)

            correct_prediction += torch.sum(preds == targets)
            losses.append(loss.item())

    return correct_prediction.double() / n_examples, np.mean(losses)
```

Training

```
%%time

# Store the train history
history = defaultdict(list)

# Control the best accuracy
now = datetime.datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
best_accuracy = 0

# Loop
for epoch in range(EPOCHS):

    print(f'Epoch {epoch+1}/{EPOCHS}')
    print('-' * 10)
    train_acc, train_loss = train_model(model_sentiment_classifier,
                                        train_data_loader,
                                        loss_fn,
                                        optimizer,
                                        device,
                                        scheduler,
                                        len(df_train))

    print(f'Train error: {train_loss} Train accuracy: {train_acc}')

    valid_acc, valid_loss = evaluate_model(model_sentiment_classifier,
                                           valid_data_loader,
                                           loss_fn,
                                           device,
                                           len(df_valid))

    print(f'Validation error: {valid_loss} Validation accuracy: {valid_acc}')
    print()

    history['train_acc'].append(train_acc)
    history['train_loss'].append(train_loss)
    history['valid_acc'].append(valid_acc)
    history['valid_loss'].append(valid_loss)

    if valid_acc > best_accuracy:
        torch.save(model_sentiment_classifier.state_dict(), f'models/model_sentiment_classifi
        best_accuracy = valid_acc
```

```
Epoch 1/20
----------
Train error: 0.9689015297432227 Train accuracy: 0.5550426955919686
Validation error: 0.9876737997812384 Validation accuracy: 0.5535055350553505

Epoch 2/20
----------
Train error: 0.8720542777508388 Train accuracy: 0.6665128086775907
Validation error: 0.8889718458932989 Validation accuracy: 0.6494464944649446

Epoch 3/20
----------
Train error: 0.7904759377131163 Train accuracy: 0.7569813062543272
Validation error: 0.8591430309940787 Validation accuracy: 0.6900369003690037

Epoch 4/20
----------
Train error: 0.7303455419206092 Train accuracy: 0.8186014308792985
Validation error: 0.8374723932322334 Validation accuracy: 0.7084870848708487

Epoch 5/20
----------
Train error: 0.7050722646097416 Train accuracy: 0.8453727209785369
Validation error: 0.8022963913048015 Validation accuracy: 0.7472324723247232

Epoch 6/20
----------
Train error: 0.6873861494099522 Train accuracy: 0.8636048926840526
Validation error: 0.8048255741596222 Validation accuracy: 0.7435424354243543

Epoch 7/20
----------
Train error: 0.6782636818410726 Train accuracy: 0.8723747980613894
Validation error: 0.7946215657626882 Validation accuracy: 0.7546125461254612

Epoch 8/20
----------
Train error: 0.6710077627558549 Train accuracy: 0.8802215555042696
Validation error: 0.8055428073686712 Validation accuracy: 0.7453874538745388

Epoch 9/20
----------
Train error: 0.6633158882605633 Train accuracy: 0.8876067389799216
Validation error: 0.7873189747333527 Validation accuracy: 0.7619926199261993

Epoch 10/20
----------
Train error: 0.6607069258760262 Train accuracy: 0.8906069697669052
Validation error: 0.7916684448719025 Validation accuracy: 0.7601476014760148

Epoch 11/20
----------
Train error: 0.6612049781088459 Train accuracy: 0.8896838218324487
Validation error: 0.7964956374729381 Validation accuracy: 0.7509225092250922
```

Model trained and saved to disk!

```
history
```