BERT

Loading the Data

```
!gdown --id 15LVlv7K31SzBs_IyZE_md0u6qEg6rby6
!unzip dataminingmtl782.zip
```

Importing Required Libraries

```
!pip install -q wordcloud
import wordcloud

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.stem.wordnet import WordNetLemmatizer
import pickle
import cv2
import re
from pylab import rcParams
from matplotlib import rc
from nltk.tokenize import word_tokenize
from textblob import TextBlob
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, plot_confusion_matrix
from tqdm import tqdm
import torch
from torch import nn, optim
from torch.utils import data
%matplotlib inline
%config InlineBackend.figure_formate = 'retina'
sns.set(style = 'whitegrid', palette = 'muted', font_scale = 1.2)
HAPPY_COLOURS_PALETTE = ['#01BEFE', '#FFDD00', '#FF7D00', '#FF006D', '#ADFF02', '#8F00FF']
sns.set_palette(sns.color_palette(HAPPY_COLOURS_PALETTE))
rcParams['figure.figsize'] = 8,8
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)
```

Loading the file Data-frames

```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
train_df
```

Setting Hyper Parameters

```
IMG_SCALING = (3, 3)
BATCH_SIZE = 48
GAUSSIAN_NOISE = 0.1
N_CLASSES = 3
dim = (192,192)
```

Exploratory Data Analysis and Text pre-processing

```
print(train_df['image id'].duplicated().any()) # Check if there are any duplicated images in the data
train_df['label'].value_counts().plot(kind = 'barh')# Plot the counts of each label
train_df['label_num'].value_counts().plot(kind = 'barh') # Plot the counts of each label_num
```

```
sns.countplot(train_df.label_num)
plt.xlabel('Sentiment')
```

```
train_df['text'] = train_df['text'].apply(lambda x:x.lower())
test_df['text'] = test_df['text'].apply(lambda x: x.lower())# Convert the text in lower case
print(train_df[train_df['text']=='#name?']) # Check if there are any empty text images in the dataset
print(test_df[test_df['text'] == '#name?'])
train_df['text']
```

```
train_df.isnull().sum() #Check number of null entries in the dataframe
```

```
train_df.drop(858,inplace=True)# Remove the corrupted Image
train_df[ train_df['image id']=='image_6357.jpg' ]
```

```
num = 858#Check that the corrupted image has been removed
train_df['label'].iloc[num]
img = cv2.imread(MAIN_PATH +'train_images/train_images/'+train_df['image id'].iloc[num])
plt.figure(figsize = (12*1.2,8*1.2))
plt.imshow(img)
plt.axis('off')
```

```
for i,id in enumerate(train_df['image id']):
    try:
        img = cv2.imread(MAIN_PATH +'train_images/train_images/'+id)
        if(img.shape[2]!=3):
            print(id + ' ' + img.shape)
    except:
        print(id+' '+ str(i))
# Verify that all the images that are read are of dimension(None,None,3)
```

```
for text,i in zip(train_df['text'],train_df['ID']):
    print(text)
# Go through the text and see what kind of processing is needed
```

```
# Text cleaning very important
def clean_str(string):
    string = re.sub(r"\n", " ", string)
    string = re.sub(r"\r", " ", string)
    string = re.sub(r"[0-9]", "digit", string)# Replace digits with the word digit
    string = re.sub(r"\'", " ", string)
    string = re.sub(r"\"", " ", string)
    string = re.sub(r"\?", " ", string)
    string = re.sub(r"\!", " ", string)
    string = re.sub(r"\/", " ", string)
    string = re.sub(r"\\", " ", string)
    string = re.sub(r"\.", " ", string)
    sample = re.sub(r'''(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()
<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^\s()<>]+|(\([^\s()<>]+\)))*\)|[^\s`!()\[\]
{};:'".,<>?«»“”‘’]))''', " ", string)# Remove hyperlinks
    sample = re.sub(r"http\S+", "", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+.COM","", sample)# Remove the hyperlinks with www and
all the possible missing combintations of .
    sample = re.sub(r"WWW.[a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"WWW.[a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"www [a-zA-Z0-9_@]+ COM","", sample)
    sample = re.sub(r"WWW [a-zA-Z0-9_@]+ com","", sample)
    sample = re.sub(r"www [a-zA-Z0-9_@]+ com","", sample)
    sample = re.sub(r"WWW [a-zA-Z0-9_@]+ COM","", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+ COM","", sample)
    sample = re.sub(r"WWW.[a-zA-Z0-9_@]+ com","", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+ com","", sample)
    sample = re.sub(r"WWW.[a-zA-Z0-9_@]+ COM","", sample)
    sample = re.sub(r"www [a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"WWW [a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"www [a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"WWW [a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"WWW[a-zA-Z0-9_@]+com","", sample)
    sample = re.sub(r"www[a-zA-Z0-9_@]+com","", sample)
    sample = re.sub(r"WWW[a-zA-Z0-9_@]+COM","", sample)
    sample = re.sub(r"www[a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"WWW[a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"www[a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"WWW[a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+COM","", sample)
    sample = re.sub(r"WWW.[a-zA-Z0-9_@]+com","", sample)
    sample = re.sub(r"www.[a-zA-Z0-9_@]+com","", sample)
    sample = re.sub(r"WWW.[a-zA-Z0-9_@]+COM","", sample)
    sample = re.sub(r"http\S+", "", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+.com","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+.COM","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+ COM","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+ com","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+ com","", sample)
    sample = re.sub(r"[a-zA-Z0-9_@]+ COM","", sample)
    return sample
    # string = re.sub(r'^https?:\/\/.*[\r\n]*', '', string, flags=re.MULTILINE) # WILL REMOVE
HYPERLINKS!!!!
    return string.strip().lower()


train_df['text'] = train_df['text'].apply(clean_str) # Should be very very clean
test_df['text'] = test_df['text'].apply(clean_str)
```

```
import nltk
nltk.download('punkt')
def TOK(text):
    return word_tokenize(text.lower())
train_df['tokens'] = train_df.text.apply(TOK)

train_df
```

```
test_df['tokens'] = test_df.text.apply(TOK)

test_df
```

```
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = [word.lower() for word in stopwords.words('english')]
def remove_stopwords(tokens):
  ans = []
  for tok in tokens:
    if tok in stop_words:
      continue
    pas = True
    for s in '@!`~\'\"\\*&^%$#-=+[].,<>?':
      if s in tok:
        pas = False
        break
    if not pas:
      continue
    if 'meme' in tok:
      continue
    ans.append(tok)
  return ans
# print(train_df.tokens.iloc[599],remove_stopwords(train_df.tokens.iloc[599]))
```

```
train_df.tokens = train_df.tokens.apply(remove_stopwords)
test_df.tokens = test_df.tokens.apply(remove_stopwords)
```

```
def Tokens_to_text(tokens):
    s = ''
    for token in tokens:
        s+=token+' '
    return s.lower().strip()
def maxLen(df):
    x = 0
    for i, tokens in enumerate(df.tokens):
        x = max(x, len(tokens))
    return x
max_words = maxLen(train_df)
```

```
train_df.text = train_df.tokens.apply(Tokens_to_text)
test_df.text = test_df.tokens.apply(Tokens_to_text)
```

Before processing vs after processing

```
Before: 3...2...1..dont do it timmy!
After: dont do it timmy
Before: a vote for trump is a vote for putin 2009 www.protectourelections.com
After a vote for trump is a vote for putin 2009

Before: “i start where the last man left off.” thomas edison visit: www.cettechnology.com/memes
for more quotes © techsolmarketing.com - free for use without modification
After: i start where the last man left off thomas edison visit: memes for more quotes © free for
use without modification
```

Loading Model

```
import transformers
tokenizer = transformers.BertTokenizer.from_pretrained('bert-base-uncased')
sample_text = train_df.text.iloc[4]
sample_text
```

```
tokens = tokenizer.tokenize(sample_text)
print(len(tokens))
print(tokens)
```

```
token_ids = tokenizer.convert_tokens_to_ids(tokens)
print(len(token_ids))
print(token_ids)
```

```
encoding = tokenizer.encode_plus(
    sample_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'
)
```

```
token_lens = []
for text in train_df.text:
  tokens = tokenizer.encode(text, max_length= 512)
  token_lens.append(len(tokens))
for text in test_df.text:
  tokens = tokenizer.encode(text, max_length= 512)
  token_lens.append(len(tokens))
```

```
sns.distplot(token_lens)
```

Creating Data Set

```
class MemeClassificationDataset(data.Dataset):
  def __init__(self, text,label_num, tokenizer, max_len ):
    self.text = text
    self.label_num = label_num
    self.tokenizer  = tokenizer
    self.max_len = max_len
  def __len__(self):
    return len(self.text)
  def __getitem__(self, index):
    review = str(self.text[index])
    encoding = tokenizer.encode_plus(
      review,
      max_length = self.max_len,
      add_special_tokens = True,
      pad_to_max_length = True,
      return_attention_mask = True,
      return_token_type_ids = False,
      return_tensors = 'pt'
    )
    return {
        'review_text':text,
        'input_ids':encoding['input_ids'].flatten(),
        'attention_mask':encoding['attention_mask'].flatten(),
        'targets':torch.tensor(self.label_num[index], dtype = torch.long)
    }
```

```
MAX_LEN = 64
BATCH_SIZE = 16
EPOCHS = 20
```

Splitting into training, testing and Validation sets

```
df_train, df_val = train_test_split(train_df, test_size = 0.1, random_state = RANDOM_SEED)
df_test, df_val = train_test_split(df_val, test_size = 0.5, random_state = RANDOM_SEED)
```

Creating Data Loader

```
def create_data_loader(df, tokenizer, batch_size, max_len):
  ds = MemeClassificationDataset(
      text = df.text.to_numpy(),
      label_num = df.label_num.to_numpy(),
      tokenizer = tokenizer,
      max_len = max_len
  )
  return data.DataLoader(
      ds,
      batch_size = batch_size,
      num_workers = 4
  )
train_data_loader = create_data_loader(df_train, tokenizer, BATCH_SIZE, MAX_LEN)
test_train_data_loader = create_data_loader(df_test, tokenizer, BATCH_SIZE, MAX_LEN)
val_data_loader = create_data_loader(df_val, tokenizer, BATCH_SIZE, MAX_LEN)
```

Creating the sentiment Classifier model

```
PRE_TRAINED_MODEL_NAME = 'bert-base-cased'
class SentimentClassifier(nn.Module):
  def __init__(self, n_classes):
    super(SentimentClassifier,self).__init__()
    self.bert = transformers.BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
    self.drop = nn.Dropout(0.2)
    self.out = nn.Linear(self.bert.config.hidden_size,n_classes)
    self.softmax = nn.Softmax(dim = 1)


  def forward(self, input_ids, attention_mask):
    ext_put = self.bert(
        input_ids = input_ids,
        attention_mask = attention_mask,

    )
    pooled_output = ext_put[1]
    output = self.drop(pooled_output)
    output = self.out(output)
    output = self.softmax(output)
    return output
```

```
torch.cuda.get_device_name(0)#Checking Cuda is enabled
```

```
device = torch.device('cuda:0')
model = SentimentClassifier(3)
model = model.to(device)
input_ids = t_data['input_ids'].to(device)
attention_mask = t_data['attention_mask'].to(device)
t_data = next(iter(train_data_loader))
t_data.keys()
print(input_ids.shape, attention_mask.shape)
```

Training The model

```
optimizer = transformers.AdamW(model.parameters(),lr = 2e-5, correct_bias = False)

total_steps = len(train_data_loader)*EPOCHS
scheduler= transformers.get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps = 0,
    num_training_steps = total_steps
)


loss_fn = nn.CrossEntropyLoss().to(device)
```

```
# one training Epoch
def train_epoch(
    model,
    data_loader,
    loss_fn,
    optimizer,
    device,
    scheduler,
    n_examples
):
  model = model.train()
  losses = []
  correct_predictions = 0
  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_predictions+=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_predictions.double()/n_examples,np.mean(losses)
```

```
#Evaluate the model performance
def eval_model(model, data_loader, loss_fn, device, n_examples):
  model = model.eval()
  losses = []
  correct_predictions = 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_predictions+=torch.sum(preds == targets)
      losses.append(loss)
  return correct_predictions.double()/n_examples,np.mean(losses)
```

```
import collections
%%time

history = collections.defaultdict(list)
best_acc = 0
for epoch in range(EPOCHS):
  print(f'Epoch {epoch+1}/{EPOCHS}')
  print('-'*10)
  train_acc, train_loss = train_epoch(
      model,
      train_data_loader,
      loss_fn,
      optimizer,
      device,
      scheduler,
      len(df_train)
  )
  print(f'Train Loss: {train_loss}, Train Accuracy: {train_acc}')

  val_acc, val_loss = eval_model(
      model,
      val_data_loader,
      loss_fn,
      device,
      len(df_val)
  )
  print(f'Validation Loss: {val_loss}, Validation Accuracy: {val_acc}')
  history['train acc'].append(train_acc)
  history['train loss'].append(train_loss)

  history['val acc'].append(val_acc)
  histroy('val loss').append(val_loss)
  if val_acc >best_acc:
    best_acc = val_acc
    torch.save(model, 'model.pth')
```

Prediction

```
def get_predictions(model, data_loader):
  model = model.eval()
  review_texts = []
  predictions = []
  prediction_probs = []
  real_values = []
  with torch.no_grad():
    for d in data_loader:
      texts = d["review_text"]
      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)
      review_texts.extend(texts)
      predictions.extend(preds)
      prediction_probs.extend(outputs)
      real_values.extend(targets)
  predictions = torch.stack(predictions).cpu()
  prediction_probs = torch.stack(prediction_probs).cpu()
  real_values = torch.stack(real_values).cpu()
  return review_texts, predictions, prediction_probs, real_values
```

```
y_review_texts, y_pred, y_pred_probs, y_test = get_predictions(
  model,
  test_data_loader
)
```

Predicting on test Set

```
test_df['label_num'] = np.zeros(shape = (len(test_df,)))
test_train_data_loader = create_data_loader(test_df, tokenizer, BATCH_SIZE, MAX_LEN)
y_review_texts, y_pred, y_pred_probs, y_test = get_predictions(
  model,
  test_data_loader
)
```

Submission

```
submit = pd.read_csv('Samplesubmission.csv')
submit.label_num = y_pred
submit.to_csv("Submission.csv", index = False)
```