This Notebook trains a sentiment classification model on a book review dataset

The model is specially designed for reviews with word count lenght less than 250.

Here we have used Transformers library and will be usin XLNET transformner with fine tuning

```
pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.7/dist-packages (4
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: tokenizers<0.11,>=0.10.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: sacremoses in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: huggingface-hub==0.0.12 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from sacre
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from sac
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from sa
```

```
pip install sentencepiece
```

```
Requirement already satisfied: sentencepiece in /usr/local/lib/python3.7/dist-packages (
```

```
# install the required libraries
import os
import math

import torch
from torch.nn import BCEWithLogitsLoss
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler
from transformers import AdamW, XLNetTokenizer, XLNetModel, XLNetLMHeadModel, XLNetConfig
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
from tqdm import tqdm, trange
import matplotlib.pyplot as plt
```

```
%matplotlib inline
```

```
# check the GPU availability
print("GPU Available: {}".format(torch.cuda.is_available()))
n_gpu = torch.cuda.device_count()
print("Number of GPU Available: {}".format(n_gpu))
print("GPU: {}".format(torch.cuda.get_device_name(0)))
```

```
GPU Available: True
Number of GPU Available: 1
GPU: Tesla P100-PCIE-16GB
```

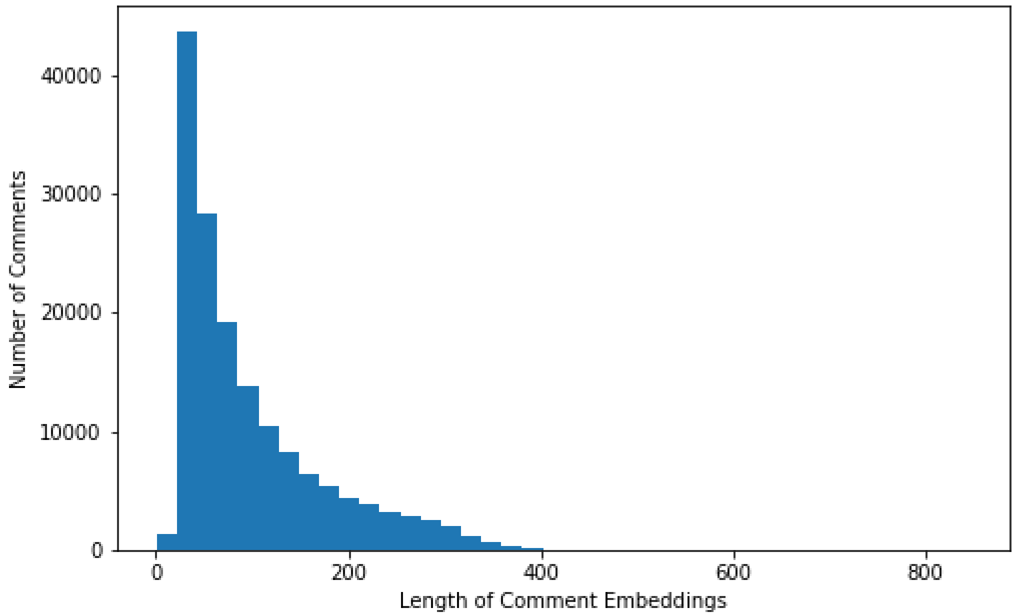
```
# mount the google drive
from google.colab import drive
drive.mount("/content/gdrive")
```

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

```
def lenCategory(x):
    if x == 1:
        return "1"

    if x == 2:
        return "2"

    if x == 3:
        return "3"

    if x <= 5:
        return "4-5"

    if x <= 10:
        return "6-10"

    if x <= 25:
        return "11-25"

    if x <= 50:
        return "26-50"

    if x <= 100:
        return "51-100"

    if x <= 250:
        return "101-250"
```

```
    if x <= 500:
        return "251-500"

    if x <= 1000:
        return "501-1000"

    if x <= 2500:
        return "1001-2500"

    if x <= 5000:
        return "2501-5000"

    return '5000+'



# custom function to load the data
def loadData(path):
    data = pd.read_csv(path)
    data = data.drop_duplicates("review")
    data["len"] = data.apply(lambda x : len(str(x["review"]).split()) , axis = 1)
    data["lencat"] = data.apply(lambda x : lenCategory(x["len"]) , axis = 1)
    return data


train = loadData('/content/gdrive/MyDrive/Toptal/train_data.csv')
holdout = loadData('/content/gdrive/MyDrive/Toptal/holdout_data.csv')

train = train.dropna()
holdout = holdout.dropna()


# select the data with wordcount less than equal to 250
train = train[~train.lencat.isin(['501-1000','1001-2500', '2501-5000',"251-500"])]
holdout = holdout[~holdout.lencat.isin(['501-1000','1001-2500', '2501-5000',"251-500"])]


def plot_sentence_embeddings_length(text_list, tokenizer):
    tokenized_texts = list(map(lambda t: tokenizer.tokenize(t), text_list))
    tokenized_texts_len = list(map(lambda t: len(t), tokenized_texts))
    fig, ax = plt.subplots(figsize=(8, 5));
    ax.hist(tokenized_texts_len, bins=40);
    ax.set_xlabel("Length of Comment Embeddings");
    ax.set_ylabel("Number of Comments");
    return

# load the XLNET tokenizer based on XLNET LARGE CASED
tokenizer = XLNetTokenizer.from_pretrained('xlnet-large-cased', do_lower_case=True)

# get the review text
train_text_list = train["review"].values
test_text_list = holdout["review"].values
```

```
plot_sentence_embeddings_length(train_text_list, tokenizer)

plot_sentence_embeddings_length(test_text_list, tokenizer)
```




```
def tokenize_inputs(text_list, tokenizer, num_embeddings=512):
    """
    Tokenizes the input text input into ids. Appends the appropriate special
    characters to the end of the text to denote end of sentence. Truncate or pad
    the appropriate sequence length.
    """
    # tokenize the text, then truncate sequence to the desired length minus 2 for
    # the 2 special characters
    tokenized_texts = list(map(lambda t: tokenizer.tokenize(t)[:num_embeddings-2], text_list)
    # convert tokenized text into numeric ids for the appropriate LM
    input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_texts]
    # append special token "<s>" and </s> to end of sentence
    input_ids = [tokenizer.build_inputs_with_special_tokens(x) for x in input_ids]
    # pad sequences
    input_ids = pad_sequences(input_ids, maxlen=num_embeddings, dtype="long", truncating="pos
    return input_ids


# create input id tokens

train_input_ids = tokenize_inputs(train_text_list, tokenizer)
train_input_ids.shape


test_input_ids = tokenize_inputs(test_text_list, tokenizer)
test_input_ids.shape
```

```
(157565, 512)
```

```
def create_attn_masks(input_ids):
    """
    Create attention masks to tell model whether attention should be applied to
    the input id tokens. Do not want to perform attention on padding tokens.
    """
    # Create attention masks
    attention_masks = []

    # Create a mask of 1s for each token followed by 0s for padding
    for seq in input_ids:
        seq_mask = [float(i>0) for i in seq]
        attention_masks.append(seq_mask)
    return attention_masks

train_attention_masks = create_attn_masks(train_input_ids)
test_attention_masks = create_attn_masks(test_input_ids)
```

```
# add input ids and attention masks to the dataframe
train["features"] = train_input_ids.tolist()
train["masks"] = train_attention_masks

holdout["features"] = test_input_ids.tolist()
holdout["masks"] = test_attention_masks
```

```
train.head()
```

```
holdout.head()
```

	review	summary	score	sentiment	len	lencat	features	masks
0	I enjoyed this book, although a paranormal rom...	Good read	5.0	2	20	11-25	[17, 150, 4163, 52, 522, 19, 1082, 24, 31391, ...	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...
	I thought this was					101-	[17, 150, 449, 52,	[1.0, 1.0, 1.0, 1.0, 1.0,

```
# train valid split
train, valid = train_test_split(train, test_size=0.2, random_state=42)
```

```
X_train = train["features"].values.tolist()
```

```
X_valid = valid["features"].values.tolist()

train_masks = train["masks"].values.tolist()
valid_masks = valid["masks"].values.tolist()


from sklearn.preprocessing import OneHotEncoder
one_hot = OneHotEncoder(sparse=False).fit(
  train.sentiment.to_numpy().reshape(-1, 1)
)


Y_train = one_hot.transform(train.sentiment.to_numpy().reshape(-1, 1))
Y_valid = one_hot.transform(valid.sentiment.to_numpy().reshape(-1, 1))


Y_train


# Convert all of our input ids and attention masks into
# torch tensors, the required datatype for our model

X_train = torch.tensor(X_train)
X_valid = torch.tensor(X_valid)

Y_train = torch.tensor(Y_train, dtype=torch.float32)
Y_valid = torch.tensor(Y_valid, dtype=torch.float32)

train_masks = torch.tensor(train_masks, dtype=torch.long)
valid_masks = torch.tensor(valid_masks, dtype=torch.long)


# Select a batch size for training
batch_size = 8

# Create an iterator of our data with torch DataLoader. This helps save on
# memory during training because, unlike a for loop,
# with an iterator the entire dataset does not need to be loaded into memory

train_data = TensorDataset(X_train, train_masks, Y_train)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data,\
                              sampler=train_sampler,\
                              batch_size=batch_size)

validation_data = TensorDataset(X_valid, valid_masks, Y_valid)
validation_sampler = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data,\
                                   sampler=validation_sampler,\
                                   batch_size=batch_size)


def train(model, num_epochs,\
          optimizer,\
```

```
            optimizer,\
            train_dataloader, valid_dataloader,\
            model_save_path,\
            train_loss_set=[], valid_loss_set = [],\
            lowest_eval_loss=None, start_epoch=0,\
            device="cpu"
            ):
    """
    Train the model and save the model with the lowest validation loss
    """

    model.to(device)

    # trange is a tqdm wrapper around the normal python range
    for i in trange(num_epochs, desc="Epoch"):
      # if continue training from saved model
      actual_epoch = start_epoch + i

      # Training

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

      # Tracking variables
      tr_loss = 0
      num_train_samples = 0

      # Train the data for one epoch
      for step, batch in enumerate(train_dataloader):
        # Add batch to GPU
        batch = tuple(t.to(device) for t in batch)
        # Unpack the inputs from our dataloader
        b_input_ids, b_input_mask, b_labels = batch
        # Clear out the gradients (by default they accumulate)
        optimizer.zero_grad()
        # Forward pass
        loss = model(b_input_ids, attention_mask=b_input_mask, labels=b_labels)
        # store train loss
        tr_loss += loss.item()
        num_train_samples += b_labels.size(0)
        # Backward pass
        loss.backward()
        # Update parameters and take a step using the computed gradient
        optimizer.step()
        #scheduler.step()

      # Update tracking variables
      epoch_train_loss = tr_loss/num_train_samples
      train_loss_set.append(epoch_train_loss)

      print("Train loss: {}".format(epoch_train_loss))
```

```
    # Validation

    # Put model in evaluation mode to evaluate loss on the validation set
    model.eval()

    # Tracking variables
    eval_loss = 0
    num_eval_samples = 0

    # Evaluate data for one epoch
    for batch in valid_dataloader:
      # Add batch to GPU
      batch = tuple(t.to(device) for t in batch)
      # Unpack the inputs from our dataloader
      b_input_ids, b_input_mask, b_labels = batch
      # Telling the model not to compute or store gradients,
      # saving memory and speeding up validation
      with torch.no_grad():
        # Forward pass, calculate validation loss
        loss = model(b_input_ids, attention_mask=b_input_mask, labels=b_labels)
        # store valid loss
        eval_loss += loss.item()
        num_eval_samples += b_labels.size(0)

    epoch_eval_loss = eval_loss/num_eval_samples
    valid_loss_set.append(epoch_eval_loss)

    print("Valid loss: {}".format(epoch_eval_loss))

    if lowest_eval_loss == None:
      lowest_eval_loss = epoch_eval_loss
      # save model
      save_model(model, model_save_path, actual_epoch,\
                 lowest_eval_loss, train_loss_set, valid_loss_set)
    else:
      if epoch_eval_loss < lowest_eval_loss:
        lowest_eval_loss = epoch_eval_loss
        # save model
        save_model(model, model_save_path, actual_epoch,\
                   lowest_eval_loss, train_loss_set, valid_loss_set)
    print("\n")

  return model, train_loss_set, valid_loss_set


def save_model(model, save_path, epochs, lowest_eval_loss, train_loss_hist, valid_loss_hist):
  """
  Save the model to the path directory provided
  """
  model_to_save = model.module if hasattr(model, 'module') else model
  checkpoint = {'epochs': epochs, \
                'lowest_eval_loss': lowest_eval_loss,\
```

```
              'state_dict': model_to_save.state_dict(),\
              'train_loss_hist': train_loss_hist,\
              'valid_loss_hist': valid_loss_hist
             }
  torch.save(checkpoint, save_path)
  print("Saving model at epoch {} with validation loss of {}".format(epochs,\
                                                                      lowest_eval_loss))
  return

def load_model(save_path):
  """
  Load the model from the path directory provided
  """
  checkpoint = torch.load(save_path)
  model_state_dict = checkpoint['state_dict']
  model = XLNetForMultiLabelSequenceClassification(num_labels=model_state_dict["classifier.we
  model.load_state_dict(model_state_dict)

  epochs = checkpoint["epochs"]
  lowest_eval_loss = checkpoint["lowest_eval_loss"]
  train_loss_hist = checkpoint["train_loss_hist"]
  valid_loss_hist = checkpoint["valid_loss_hist"]

  return model, epochs, lowest_eval_loss, train_loss_hist, valid_loss_hist


torch.cuda.empty_cache()


#config = XLNetConfig()

class XLNetForMultiLabelSequenceClassification(torch.nn.Module):

  def __init__(self, num_labels=2):
    super(XLNetForMultiLabelSequenceClassification, self).__init__()
    self.num_labels = num_labels
    self.xlnet = XLNetModel.from_pretrained('xlnet-base-cased')
    self.classifier = torch.nn.Linear(768, num_labels)

    torch.nn.init.xavier_normal_(self.classifier.weight)

  def forward(self, input_ids, token_type_ids=None,\
              attention_mask=None, labels=None):
    # last hidden layer
    last_hidden_state = self.xlnet(input_ids=input_ids,\
                                   attention_mask=attention_mask,\
                                   token_type_ids=token_type_ids)
    # pool the outputs into a mean vector
    mean_last_hidden_state = self.pool_hidden_state(last_hidden_state)
    logits = self.classifier(mean_last_hidden_state)

    if labels is not None:
```

```
      loss_fct = BCEWithLogitsLoss()
      loss = loss_fct(logits.view(-1, self.num_labels),\
                      labels.view(-1, self.num_labels))
      return loss
    else:
      return logits

  def freeze_xlnet_decoder(self):
    """
    Freeze XLNet weight parameters. They will not be updated during training.
    """
    for param in self.xlnet.parameters():
      param.requires_grad = False

  def unfreeze_xlnet_decoder(self):
    """
    Unfreeze XLNet weight parameters. They will be updated during training.
    """
    for param in self.xlnet.parameters():
      param.requires_grad = True

  def pool_hidden_state(self, last_hidden_state):
    """
    Pool the output vectors into a single mean vector
    """
    last_hidden_state = last_hidden_state[0]
    mean_last_hidden_state = torch.mean(last_hidden_state, 1)
    return mean_last_hidden_state

model = XLNetForMultiLabelSequenceClassification(num_labels=3)
#model = torch.nn.DataParallel(model)
#model.cuda()
```

```
Some weights of the model checkpoint at xlnet-base-cased were not used when initializing
- This IS expected if you are initializing XLNetModel from the checkpoint of a model tra
- This IS NOT expected if you are initializing XLNetModel from the checkpoint of a model
```

```
optimizer = AdamW(model.parameters(), lr=2e-5, weight_decay=0.01, correct_bias=False)
```

```
# num_epochs=10

# # cwd = os.getcwd()
# model_save_path = output_model_file = "/content/gdrive/MyDrive/Toptal/xlnet_11-25_aug_balan
# model, train_loss_set, valid_loss_set = train(model=model,\
#                                               num_epochs=num_epochs,\
#                                               optimizer=optimizer,\
#                                               train_dataloader=train_dataloader,\
#                                               valid_dataloader=validation_dataloader,\
#                                               model_save_path=model_save_path,\
```

```
#                                               model_save_path=model_save_path,\
#                                               device="cuda")


"""Epoch:   0%|          | 0/10 [00:00<?, ?it/s]Train loss: 0.019482453157987224
Valid loss: 0.017420511438132023
Epoch:  10%|█         | 1/10 [7:13:50<65:04:38, 26030.91s/it]Saving model at epoch 0 with val


Train loss: 0.01679534113878288
Epoch:  20%|██        | 2/10 [14:27:13<57:48:33, 26014.17s/it]Valid loss: 0.01758294764020690


Epoch:  20%|██        | 2/10 [16:06:24<64:25:38, 28992.35s/it]"""


def generate_predictions(model, df, num_labels, device="cpu", batch_size=32):
  num_iter = math.ceil(df.shape[0]/batch_size)

  pred_probs = np.array([]).reshape(0, num_labels)

  model.to(device)
  model.eval()

  for i in range(num_iter):
    df_subset = df.iloc[i*batch_size:(i+1)*batch_size,:]
    X = df_subset["features"].values.tolist()
    masks = df_subset["masks"].values.tolist()
    X = torch.tensor(X)
    masks = torch.tensor(masks, dtype=torch.long)
    X = X.to(device)
    masks = masks.to(device)
    with torch.no_grad():
      logits = model(input_ids=X, attention_mask=masks)
      logits = logits.sigmoid().detach().cpu().numpy()
      pred_probs = np.vstack([pred_probs, logits])

  return pred_probs


# load the saved model
model, start_epoch, lowest_eval_loss, train_loss_hist, valid_loss_hist = load_model("/content
```

```
Some weights of the model checkpoint at xlnet-base-cased were not used when initializing
- This IS expected if you are initializing XLNetModel from the checkpoint of a model tra
- This IS NOT expected if you are initializing XLNetModel from the checkpoint of a model
```

```
num_labels = 3
pred_probs = generate_predictions(model, holdout, num_labels, device="cuda", batch_size=32)
preds = np.argmax(pred_probs, axis = 1)
ytrue = holdout.sentiment.values
```

```
from sklearn.metrics import accuracy_score, recall_score, precision_score, confusion_matrix,
```

```
accuracy_score(ytrue,preds)
```

```
0.9167137371878272
```

```
print(classification_report(ytrue,preds))
```

```
              precision    recall  f1-score   support

           0       0.76      0.73      0.74      9295
           1       0.63      0.47      0.54     14963
           2       0.95      0.98      0.97    133307

    accuracy                           0.92    157565
   macro avg       0.78      0.73      0.75    157565
weighted avg       0.91      0.92      0.91    157565
```

```
'''

0s
print(classification_report(ytrue,preds))
              precision    recall  f1-score   support

           0       0.76      0.73      0.74      9295
           1       0.63      0.47      0.54     14963
           2       0.95      0.98      0.97    133307

    accuracy                           0.92    157565
   macro avg       0.78      0.73      0.75    157565
weighted avg       0.91      0.92      0.91    157565
'''
```

```
'\n\n0s\nprint(classification_report(ytrue,preds))\n              precision    recall  f
ort\n\n           0       0.76      0.73      0.74      9295\n           1       0.63
0.54     14963\n           2       0.95      0.98      0.97    133307\n\n    accuracy
0.92    157565\n   macro avg       0.78      0.73      0.75    157565\nweighted avg
```

✓ 0s completed at 3:21 AM