In [1]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

from sklearn.metrics import accuracy_score, classification_report, confusion_mat
rix

import torch
from torchtext.data import Field, TabularDataset, BucketIterator, Iterator

from transformers import RobertaTokenizer, RobertaModel, AdamW, get_linear_sched
ule_with_warmup

import warnings
warnings.filterwarnings('ignore')

import logging
logging.getLogger("transformers.tokenization_utils_base").setLevel(logging.ERROR
)
```

```
wandb: WARNING W&B installed but not logged in.  Run `wandb login` o
r set the WANDB_API_KEY env variable.
```

In [2]:

```
#Loading Libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud,STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize,sent_tokenize
from bs4 import BeautifulSoup
import re,string,unicodedata
from keras.preprocessing import text, sequence
from sklearn.metrics import classification_report,confusion_matrix,accuracy_scor
e,f1_score
from sklearn.model_selection import train_test_split
from string import punctuation
from nltk import pos_tag
from nltk.corpus import wordnet
import keras
from keras.models import Sequential
from keras.layers import Dense,Embedding,LSTM,Dropout
from keras.callbacks import ReduceLROnPlateau
import tensorflow as tf

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import  LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import learning_curve
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score
from sklearn import svm
from gensim.models.word2vec import Word2Vec
import pickle
```

```
import warnings
warnings.filterwarnings("ignore")
```

In [3]:

```
df = pd.read_csv("../input/combinedtweets/combined_tweets.csv")
print(df.shape)
df.head()
```

```
(29052, 5)
```

Out[3]:

	id	title	text	source	target
0	buzzfeed	Proof The Mainstream Media Is Manipulating The...	I woke up this morning to find a variation of ...	www.addictinginfo.org	fake
1	buzzfeed	Charity: Clinton Foundation Distributed “Water...	Former President Bill Clinton and his Clinton ...	eaglerising.com	fake
2	buzzfeed	A Hillary Clinton Administration May be Entire...	After collapsing just before trying to step in...	eaglerising.com	fake
3	buzzfeed	Trump’s Latest Campaign Promise May Be His Mos...	Donald Trump is, well, deplorable. He’s sugges...	www.addictinginfo.org	fake
4	buzzfeed	Website is Down For Maintenance	Website is Down For Maintenance	www.proudcons.com	fake

In [4]:

```
df['all'] = df['title']+ ''+ df['text']
indexfake = df[ df['target'] == "fake" ].index
indexreal = df[ df['target'] == "real" ].index
# now use df.loc to set values only to those rows
df.loc[indexfake,'is_fake'] = 1
df.loc[indexreal,'is_fake'] = 0
df = df.astype({"is_fake": int})
df.head()
```

Out[4]:

	id	title	text	source	target	all
0	buzzfeed	Proof The Mainstream Media Is Manipulating The...	I woke up this morning to find a variation of ...	www.addictinginfo.org	fake	Proof The Mainstream Media Is Manipulating The...
1	buzzfeed	Charity: Clinton Foundation Distributed “Water...	Former President Bill Clinton and his Clinton ...	eaglerising.com	fake	Charity: Clinton Foundation Distributed “Water..
2	buzzfeed	A Hillary Clinton Administration May be Entire...	After collapsing just before trying to step in...	eaglerising.com	fake	A Hillary Clinton Administration May be Entire...
3	buzzfeed	Trump’s Latest Campaign Promise May Be His Mos...	Donald Trump is, well, deplorable. He’s sugges...	www.addictinginfo.org	fake	Trump’s Latest Campaign Promise May Be His Mos...
4	buzzfeed	Website is Down For Maintenance	Website is Down For Maintenance	www.proudcons.com	fake	Website is Down For MaintenanceWebsite is Down...

In [5]:

```
import nltk
nltk.download('stopwords')
stop = stopwords.words('english')
#sw_list = ['say','said','though','asked','time','including','made','including','time','number','one','called','added','according','made','put'] #adding more stop words in the list to refine the word cloud
#stop.extend(sw_list)
punctuation = list(string.punctuation)
stop.extend(punctuation)
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [6]:

```
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

#Removing the square brackets
def remove_between_square_brackets(text):
    return re.sub('\[[^]]*\]', '', text)
# Removing URL's
def remove_between_square_brackets(text):
    return re.sub(r'http\S+', '', text)
#Removing the stopwords from text
def remove_stopwords(text):
    final_text = []
    for i in text.split():
        if i.strip().lower() not in stop:
            final_text.append(i.strip())
    return " ".join(final_text)
#Removing the noisy text
def denoise_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
    text = remove_stopwords(text)
    return text
```

In [7]:

```
df['all']=df['all'].astype('string')
df = df.replace(np.nan, '', regex=True)
df['all'].dtype
```

Out[7]:

```
StringDtype
```

In [8]:

```
#Apply function on full-text column
df['all']=df['all'].apply(denoise_text)
df.head()
```

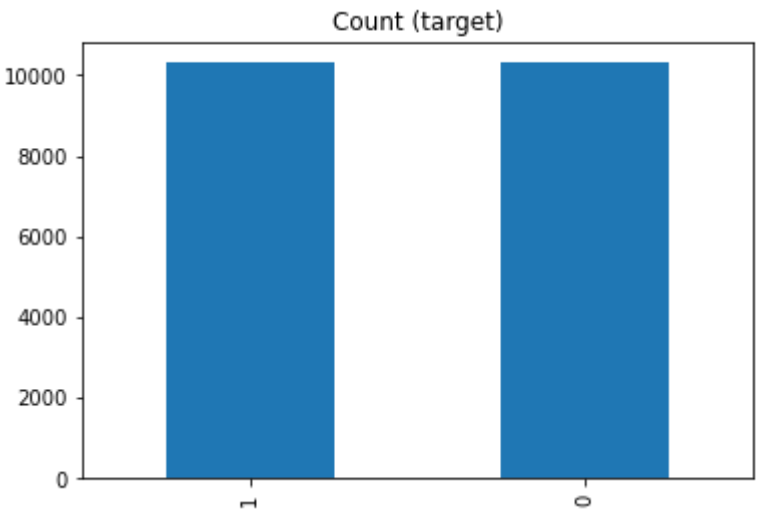
Out[8]:

	id	title	text	source	target	all
0	buzzfeed	Proof The Mainstream Media Is Manipulating The...	I woke up this morning to find a variation of ...	www.addictinginfo.org	fake	Proof Mainstream Media Manipulating Election T...
1	buzzfeed	Charity: Clinton Foundation Distributed “Water...	Former President Bill Clinton and his Clinton ...	eaglerising.com	fake	Charity: Clinton Foundation Distributed “Water..
2	buzzfeed	A Hillary Clinton Administration May be Entire...	After collapsing just before trying to step in...	eaglerising.com	fake	Hillary Clinton Administration May Entirely Ru...
3	buzzfeed	Trump’s Latest Campaign Promise May Be His Mos...	Donald Trump is, well, deplorable. He’s sugges...	www.addictinginfo.org	fake	Trump’s Latest Campaign Promise May Horrible O...
4	buzzfeed	Website is Down For Maintenance	Website is Down For Maintenance	www.proudcons.com	fake	Website MaintenanceWebsi Maintenance

In [9]:

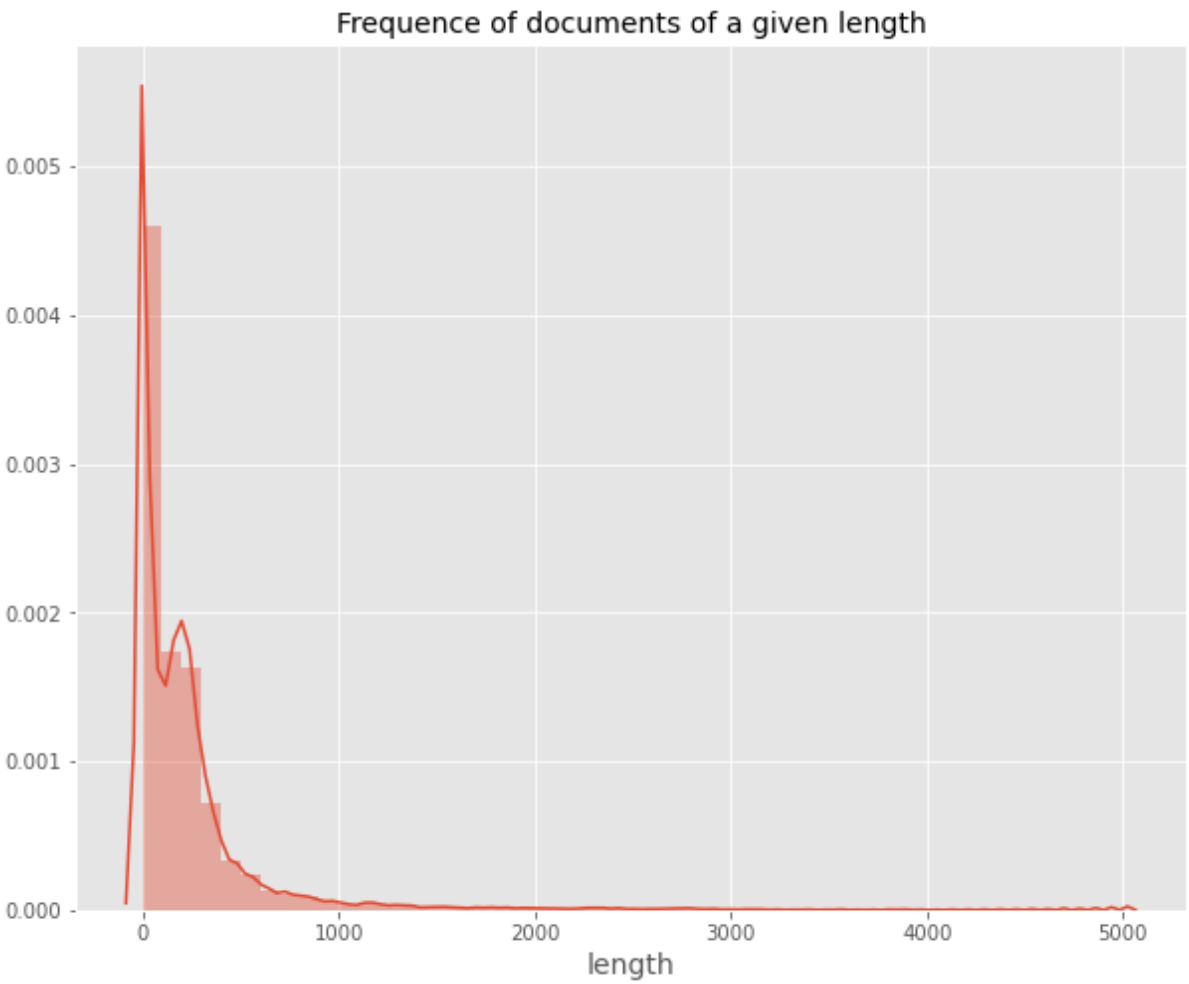
```
# Class count
count_class_real, count_class_fake = df.is_fake.value_counts()

# Divide by class
df_class_real = df[df['is_fake'] == 0]
df_class_fake = df[df['is_fake'] == 1]

df_class_real_under = df_class_real.sample(count_class_fake)
df_new = pd.concat([df_class_real_under, df_class_fake], axis=0)

print('Random under-sampling:')
print(df_new.is_fake.value_counts())

df_new.is_fake.value_counts().plot(kind='bar', title='Count (target)');

df_new.head()
print(df_new.shape)
```

```
Random under-sampling:
1    10304
0    10304
Name: is_fake, dtype: int64
(20608, 7)
```



In [10]:

```
# Plot histogram with the length. Truncate max length to 5000 tokens.
plt.style.use("ggplot")

plt.figure(figsize=(10, 8))
df_new['length'] = df_new['all'].apply(lambda x: len(x.split()))
sns.distplot(df_new[df_new['length'] < 5000]['length'])
plt.title('Frequence of documents of a given length', fontsize=14)
plt.xlabel('length', fontsize=14)
None
```



In [11]:

```
# Set random seed and set device to GPU.
torch.manual_seed(17)

if torch.cuda.is_available():
    device = torch.device('cuda:0')
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
else:
    device = torch.device('cpu')

print(device)
```

```
cuda:0
```

In [12]:

```
# Initialize tokenizer.
tokenizer = RobertaTokenizer.from_pretrained("roberta-base")
```

In [13]:

```
df_new.to_csv("data_new.csv")
```

In [14]:

```
# Set tokenizer hyperparameters.
MAX_SEQ_LEN = 256
BATCH_SIZE = 16
PAD_INDEX = tokenizer.convert_tokens_to_ids(tokenizer.pad_token)
UNK_INDEX = tokenizer.convert_tokens_to_ids(tokenizer.unk_token)


# Define columns to read.
label_field = Field(sequential=False, use_vocab=False, batch_first=True)
text_field = Field(use_vocab=False,
                   tokenize=tokenizer.encode,
                   include_lengths=False,
                   batch_first=True,
                   fix_length=MAX_SEQ_LEN,
                   pad_token=PAD_INDEX,
                   unk_token=UNK_INDEX)

fields = {'all' : ('all', text_field), 'is_fake' : ('is_fake', label_field)}

# Read preprocessed CSV into TabularDataset and split it into train, test and vali
d.
train_data, valid_data, test_data = TabularDataset(path=f"./data_new.csv",
                                                   format='CSV',
                                                   fields=fields,
                                                   skip_header=False).split(spli
t_ratio=[0.70, 0.2, 0.1],
                                                                            stra
tified=True,
                                                                            stra
ta_field='is_fake')

# Create train and validation iterators.
train_iter, valid_iter = BucketIterator.splits((train_data, valid_data),
                                               batch_size=BATCH_SIZE,
                                               device=device,
                                               shuffle=True,
                                               sort_key=lambda x: len(x.all),
                                               sort=True,
                                               sort_within_batch=False)

# Test iterator, no shuffling or sorting required.
test_iter = Iterator(test_data, batch_size=BATCH_SIZE, device=device, train=Fals
e, shuffle=False, sort=False)
```

In [15]:

```
# Functions for saving and loading model parameters and metrics.
def save_checkpoint(path, model, valid_loss):
    torch.save({'model_state_dict': model.state_dict(),
                'valid_loss': valid_loss}, path)


def load_checkpoint(path, model):
    state_dict = torch.load(path, map_location=device)
    model.load_state_dict(state_dict['model_state_dict'])

    return state_dict['valid_loss']


def save_metrics(path, train_loss_list, valid_loss_list, global_steps_list):
    state_dict = {'train_loss_list': train_loss_list,
                'valid_loss_list': valid_loss_list,
                'global_steps_list': global_steps_list}

    torch.save(state_dict, path)


def load_metrics(path):
    state_dict = torch.load(path, map_location=device)
    return state_dict['train_loss_list'], state_dict['valid_loss_list'], state_d
ict['global_steps_list']
```

In [16]:

```
# Model with extra layers on top of RoBERTa
class ROBERTAClassifier(torch.nn.Module):
    def __init__(self, dropout_rate=0.3):
        super(ROBERTAClassifier, self).__init__()

        self.roberta = RobertaModel.from_pretrained('roberta-base')
        self.d1 = torch.nn.Dropout(dropout_rate)
        self.l1 = torch.nn.Linear(768, 64)
        self.bn1 = torch.nn.LayerNorm(64)
        self.d2 = torch.nn.Dropout(dropout_rate)
        self.l2 = torch.nn.Linear(64, 2)

    def forward(self, input_ids, attention_mask):
        _, x = self.roberta(input_ids=input_ids, attention_mask=attention_mask)
        x = self.d1(x)
        x = self.l1(x)
        x = self.bn1(x)
        x = torch.nn.Tanh()(x)
        x = self.d2(x)
        x = self.l2(x)

        return x
```

In [17]:

```
def pretrain(model,
             optimizer,
             train_iter,
             valid_iter,
             scheduler = None,
             valid_period = len(train_iter),
             num_epochs = 5):

    # Pretrain linear layers, do not train bert
    for param in model.roberta.parameters():
        param.requires_grad = False

    model.train()

    # Initialize losses and loss histories
    train_loss = 0.0
    valid_loss = 0.0
    global_step = 0

    # Train loop
    for epoch in range(num_epochs):
        for (source, target), _ in train_iter:
            mask = (source != PAD_INDEX).type(torch.uint8)

            y_pred = model(input_ids=source,
                           attention_mask=mask)

            loss = torch.nn.CrossEntropyLoss()(y_pred, target)

            loss.backward()

            # Optimizer and scheduler step
            optimizer.step()
            scheduler.step()

            optimizer.zero_grad()

            # Update train loss and global step
            train_loss += loss.item()
            global_step += 1

            # Validation loop. Save progress and evaluate model performance.
            if global_step % valid_period == 0:
```

```
                model.eval()

                with torch.no_grad():
                    for (source, target), _ in valid_iter:
                        mask = (source != PAD_INDEX).type(torch.uint8)
                        y_pred = model(input_ids=source,
                                       attention_mask=mask)

                        loss = torch.nn.CrossEntropyLoss()(y_pred, target)

                        valid_loss += loss.item()

                # Store train and validation loss history
                train_loss = train_loss / valid_period
                valid_loss = valid_loss / len(valid_iter)

                model.train()

                # print summary
                print('Epoch [{}/{}], global step [{}/{}], PT Loss: {:.4f}, Val
 Loss: {:.4f}'
                      .format(epoch+1, num_epochs, global_step, num_epochs*len(t
rain_iter),train_loss, valid_loss))

                train_loss = 0.0
                valid_loss = 0.0

    # Set bert parameters back to trainable
    for param in model.roberta.parameters():
        param.requires_grad = True

    print('Pre-training done!')
```

In [21]:

```
def train(model,
          optimizer,
          train_iter,
          valid_iter,
          scheduler = None,
          num_epochs = 5,
          valid_period = len(train_iter)):

    # Initialize losses and loss histories
    train_loss = 0.0
    valid_loss = 0.0
    train_loss_list = []
    valid_loss_list = []
    best_valid_loss = float('Inf')

    global_step = 0
    global_steps_list = []

    model.train()
    for epoch in range(num_epochs):
        for (source, target), _ in train_iter:
            mask = (source != PAD_INDEX).type(torch.uint8)

            y_pred = model(input_ids=source,
                           attention_mask=mask)
            #output = model(input_ids=source,
            #               labels=target,
            #               attention_mask=mask)

            loss = torch.nn.CrossEntropyLoss()(y_pred, target)
            #loss = output[0]

            loss.backward()

            #torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)

            # Optimizer and scheduler step
            optimizer.step()
            scheduler.step()

            optimizer.zero_grad()
            # Update train loss and global step
            train_loss += loss.item()
```

```
            global_step += 1

            # Validation loop. Save progress and evaluate model performance.
            if global_step % valid_period == 0:
                model.eval()

                with torch.no_grad():
                    for (source, target), _ in valid_iter:
                        mask = (source != PAD_INDEX).type(torch.uint8)

                        y_pred = model(input_ids=source,
                                       attention_mask=mask)
                        #output = model(input_ids=source,
                        #               labels=target,
                        #               attention_mask=mask)

                        loss = torch.nn.CrossEntropyLoss()(y_pred, target)
                        #loss = output[0]

                        valid_loss += loss.item()

                # Store train and validation loss history
                train_loss = train_loss / valid_period
                valid_loss = valid_loss / len(valid_iter)
                train_loss_list.append(train_loss)
                valid_loss_list.append(valid_loss)
                global_steps_list.append(global_step)

                # print summary
                print('Epoch [{}/{}], global step [{}/{}], Train Loss: {:.4f}, V
alid Loss: {:.4f}'
                      .format(epoch+1, num_epochs, global_step, num_epochs*len(t
rain_iter),
                              train_loss, valid_loss))
                # checkpoint
                if best_valid_loss > valid_loss:
                    best_valid_loss = valid_loss
                    save_checkpoint('/model.pkl', model, best_valid_loss)
                    save_metrics('/metric.pkl', train_loss_list, valid_loss_list
, global_steps_list)

                train_loss = 0.0
                valid_loss = 0.0
                model.train()
```

```
    save_metrics('/metric.pkl', train_loss_list, valid_loss_list, global_steps_l
ist)
    print('Training done!')
```

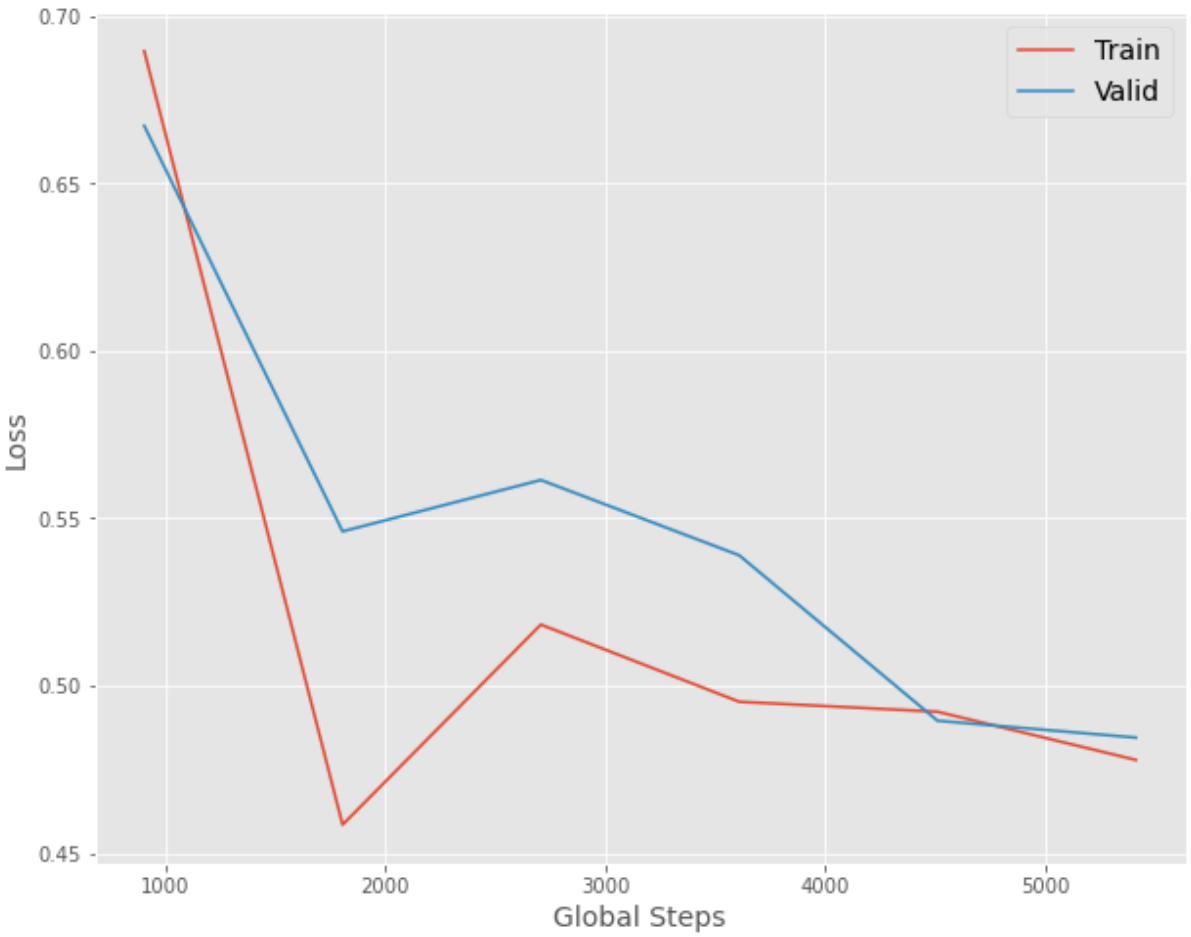
In [22]:

```
NUM_EPOCHS = 3
steps_per_epoch = len(train_iter)

model = ROBERTAClassifier(0.4)
model = model.to(device)


optimizer = AdamW(model.parameters(), lr=1e-4)
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=steps_per_epoch*1,
                                            num_training_steps=steps_per_epoch*N
UM_EPOCHS)

print("======================== Start pretraining ==============================="
)

pretrain(model=model,
         train_iter=train_iter,
         valid_iter=valid_iter,
         optimizer=optimizer,
         scheduler=scheduler,
         num_epochs=NUM_EPOCHS)

NUM_EPOCHS = 6
print("======================== Start training ==================================="
)
optimizer = AdamW(model.parameters(), lr=2e-6)
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=steps_per_epoch*2,
                                            num_training_steps=steps_per_epoch*N
UM_EPOCHS)

train(model=model,
      train_iter=train_iter,
      valid_iter=valid_iter,
      optimizer=optimizer,
      scheduler=scheduler,
      num_epochs=NUM_EPOCHS)
```

```
======================= Start pretraining ============================
====
Epoch [1/3], global step [902/2706], PT Loss: 0.6514, Val Loss: 0.73
00
Epoch [2/3], global step [1804/2706], PT Loss: 0.6164, Val Loss: 0.7
301
Epoch [3/3], global step [2706/2706], PT Loss: 0.6672, Val Loss: 0.6
943
Pre-training done!
======================= Start training ==============================
====
Epoch [1/6], global step [902/5412], Train Loss: 0.6895, Valid Loss:
0.6672
Epoch [2/6], global step [1804/5412], Train Loss: 0.4585, Valid Los
s: 0.5461
Epoch [3/6], global step [2706/5412], Train Loss: 0.5183, Valid Los
s: 0.5614
Epoch [4/6], global step [3608/5412], Train Loss: 0.4952, Valid Los
s: 0.5390
Epoch [5/6], global step [4510/5412], Train Loss: 0.4922, Valid Los
s: 0.4896
Epoch [6/6], global step [5412/5412], Train Loss: 0.4778, Valid Los
s: 0.4845
Training done!
```

In [23]:

```
plt.figure(figsize=(10, 8))
train_loss_list, valid_loss_list, global_steps_list = load_metrics('/metric.pkl'
)
plt.plot(global_steps_list, train_loss_list, label='Train')
plt.plot(global_steps_list, valid_loss_list, label='Valid')
plt.xlabel('Global Steps', fontsize=14)
plt.ylabel('Loss', fontsize=14)
plt.legend(fontsize=14)
plt.show()
```



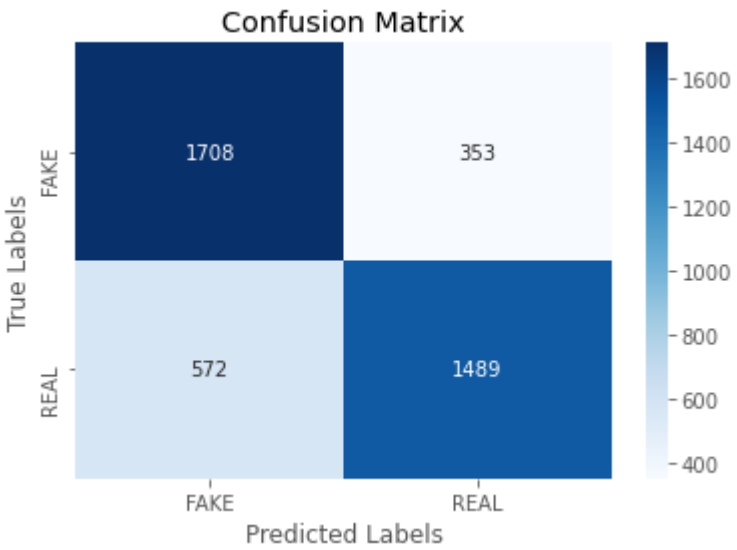
In [24]:

```
def evaluate(model, test_loader):
    y_pred = []
    y_true = []

    model.eval()
    with torch.no_grad():
        for (source, target), _ in test_loader:
                mask = (source != PAD_INDEX).type(torch.uint8)

                output = model(source, attention_mask=mask)

                y_pred.extend(torch.argmax(output, axis=-1).tolist())
                y_true.extend(target.tolist())

    print('Classification Report:')
    print(classification_report(y_true, y_pred, labels=[1,0], digits=4))

    cm = confusion_matrix(y_true, y_pred, labels=[1,0])
    ax = plt.subplot()

    sns.heatmap(cm, annot=True, ax = ax, cmap='Blues', fmt="d")

    ax.set_title('Confusion Matrix')

    ax.set_xlabel('Predicted Labels')
    ax.set_ylabel('True Labels')
    ax.xaxis.set_ticklabels(['FAKE', 'REAL'])
    ax.yaxis.set_ticklabels(['FAKE', 'REAL'])
```

In [25]:

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

load_checkpoint('/model.pkl', model)

evaluate(model, test_iter)
```

```
Classification Report:
              precision    recall  f1-score   support

           1     0.7491    0.8287    0.7869      2061
           0     0.8084    0.7225    0.7630      2061

    accuracy                         0.7756      4122
   macro avg     0.7787    0.7756    0.7750      4122
weighted avg     0.7787    0.7756    0.7750      4122
```



In [26]:

```
print(len(train_data))
print(len(valid_data))
print(len(test_data))
```

```
14426
2060
4122
```