Base_Model

November 1, 2021

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[2]:
```
import os
import pathlib
from pathlib import Path
os.chdir("/content/drive/My Drive/Classroom/projects/BERT")
!ls -l
```

```
total 11928
-rw------- 1 root root  362601 Nov  1 07:23 'Base Model.ipynb'
-rw------- 1 root root   98055 Oct 30 10:23 'Base Model.pdf'
drwx------ 2 root root    4096 Oct 28 08:56  clr
drwx------ 2 root root    4096 Sep 12 15:56  Data
-rw------- 1 root root  886614 Sep 21 17:52  EDA.ipynb
-rw------- 1 root root  634262 Sep 21 17:48  EDA.pdf
drwx------ 2 root root    4096 Oct 28 09:17  fit
-rw------- 1 root root   71427 Nov  1 03:28  model.png
drwx------ 2 root root    4096 Sep 12 15:53  papers
-rw------- 1 root root 5429800 Nov  1 03:50  tdl.hdf5
drwx------ 2 root root    4096 Oct 23 04:35  waste
-rw------- 1 root root 4708376 Oct 31 17:11  yolo.hdf5
```

[3]:
```
pip install -q tensorflow-text
```

```
     || 4.4 MB 5.3 MB/s
```

[4]:
```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import xml.etree.ElementTree as et
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pathlib
from pathlib import Path
import csv
from tqdm.notebook import tqdm
from wordcloud import WordCloud, STOPWORDS


from sklearn.preprocessing import StandardScaler, LabelBinarizer,OneHotEncoder
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
import scipy
import pickle

import tensorflow as tf
import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_text as tf_text
from tensorflow_text.tools.wordpiece_vocab import bert_vocab_from_dataset as␣
 ↪bert_vocab
import tensorflow_datasets as tfds
from tensorflow.keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

[5]:
```
csvfile = 'Data//data.csv'
sent_data_file = 'Data//sent_data.csv'
label_file = 'Data//label.csv'
vocab_file = 'Data//vocab_tr_w.txt'
```

[6]:
```
sent_df = pd.read_csv(sent_data_file)
```

[7]:
```
label = pd.read_csv(label_file)
```

[8]:
```
data = tf.data.TextLineDataset(sent_data_file)
tr_size = int(sent_df.shape[0]*0.7)
train = data.skip(1).take(tr_size)
test = data.skip(1+tr_size).take(sent_df.shape[0]-(tr_size))
```

[9]:
```
y_tr = label[:tr_size].values
y_ts = label[tr_size:].values
```

[ ]:
```
len(y_tr),len(y_ts)
```

[ ]:
```
(832588, 356824)
```

[ ]:
```
len([1 for _ in train])
```

[ ]:
```
832588
```

[ ]:
```
len([1 for _ in test])
```

[ ]:
```
356824
```

https://www.tensorflow.org/text/guide/subwords_tokenizer used to run the following cells

[ ]:
```
bert_tokenizer_params=dict(lower_case=True)
reserved_tokens=["[PAD]", "[UNK]", "[START]", "[END]"]

bert_vocab_args = dict(
    # The target vocabulary size
    vocab_size = 12000,
    # Reserved tokens that must be included in the vocabulary
    reserved_tokens=reserved_tokens,
    # Arguments for `text.BertTokenizer`
    bert_tokenizer_params=bert_tokenizer_params,
    # Arguments for `wordpiece_vocab.wordpiece_tokenizer_learner_lib.learn`
    learn_params={},
)
```

[ ]:
```
%%time
vocabulary = bert_vocab.bert_vocab_from_dataset(
    train.batch(10).prefetch(2),
    **bert_vocab_args
)
```

```
CPU times: user 9min 3s, sys: 7.54 s, total: 9min 11s
Wall time: 8min 38s
```

[ ]:
```
print(vocabulary[:10])
print(vocabulary[100:110])
print(vocabulary[1000:1010])
print(vocabulary[-10:])
```

```
['[PAD]', '[UNK]', '[START]', '[END]', '!', '"', '#', '$', '%', '&']
['as', 'be', 'this', 'cite', 'on', 'can', 'from', 'it', 'an', 'which']
['typical', 'implementation', 'makes', 'notation', 'reported', 'relatively',
'color', 'iii', 'sequences', 'sect']
['##ø', '##', '##', '##', '##', '##', '##', '##', '##', '##']
```

[ ]:
```
def write_vocab_file(filepath, vocab):
  with open(filepath, 'wb') as f:
    pickle.dump(vocab,f)

write_vocab_file('Data//vocab_tr_wb', vocabulary)
```

[10]:
```
vocab = []
with open('Data//vocab_tr_wb','rb') as f:
  vocab = pickle.load(f)
```

[ ]:
```
vocab[:10]
```

[ ]:
```
['[PAD]', '[UNK]', '[START]', '[END]', '!', '"', '#', '$', '%', '&']
```

[ ]:
```
type(vocab)
```

[ ]:
```
list
```

[ ]:
```
tokenizer = tf_text.BertTokenizer('Data//vocab_tr_w.txt',␣
 ↪**bert_tokenizer_params)
```

[ ]:
```
train_tok = train.map( lambda x: tokenizer.tokenize(x))
train_tok = train_tok.map( lambda x: x.merge_dims(-2,-1))
```

[ ]:
```
type(train_tok)
```

[ ]:
```
tensorflow.python.data.ops.dataset_ops.MapDataset
```

[ ]:
```
ListOfTensors = []
for ele in train_tok:
  #print(ele.numpy()[0])
  ListOfTensors.append(ele.numpy()[0].tolist())
#print(ListOfTensors)
```

[ ]:
```
file1 = open('Data//ListOfTensors', 'wb')
pickle.dump(ListOfTensors, file1)
file1.close()
```

[11]:
```
#file1 = open('Data//ListOfTensors', 'wb')
#pickle.dump(ListOfTensors, file1)
#file1.close()

ListOfTensors = []
with open('Data//ListOfTensors', 'rb') as f:
  ListOfTensors = pickle.load(f)
```

[ ]:
```
type(ListOfTensors),len(ListOfTensors)
```

[ ]:
```
(list, 832588)
```

[ ]:
```
test_tok = test.map( lambda x: tokenizer.tokenize(x))
test_tok = test_tok.map( lambda x: x.merge_dims(-2,-1))
```

[ ]:
```
ListOfTensors_ts = []
for ele in test_tok:
  #print(ele.numpy()[0])
  ListOfTensors_ts.append(ele.numpy()[0].tolist())
#print(ListOfTensors)
```

[ ]:
```
file1 = open('Data//ListOfTensors_ts', 'wb')
pickle.dump(ListOfTensors_ts, file1)
file1.close()
```

[12]:
```
#file1 = open('Data//ListOfTensors_ts', 'wb')
#pickle.dump(ListOfTensors_ts, file1)
#file1.close()
```

```
ListOfTensors_ts = []
with open('Data//ListOfTensors_ts', 'rb') as f:
  ListOfTensors_ts = pickle.load(f)
```

[ ]:
```
type(ListOfTensors),len(ListOfTensors)
```

[ ]:
```
(list, 832588)
```

[ ]:
```
type(ListOfTensors_ts),len(ListOfTensors_ts)
```

[ ]:
```
(list, 356824)
```

[13]:
```
# Credits: https://machinelearningmastery.com/
 ↪sequence-classification-lstm-recurrent-neural-networks-python-keras/
# LSTM for sequence classification in the IMDB dataset
import numpy
# from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense,Dropout
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
numpy.random.seed(7)
```

[14]:
```
from sklearn.model_selection import train_test_split

X_tr, X_ts = ListOfTensors, ListOfTensors_ts
```

[15]:
```
max_sbe_length = 128
X_tr = sequence.pad_sequences(X_tr, maxlen=max_sbe_length)
X_ts = sequence.pad_sequences(X_ts, maxlen=max_sbe_length)
```

[16]:
```
# truncate and/or pad input sequences
print(X_tr.shape)
print(X_tr[0])
```

```
(832588, 128)
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    5   91 4706  108 3078 1003 1577   85  227   94  313  818
  315 6534 6045   88 2218  210   92  372  118 1657   39   86   39   14
   39   96   39  112  100 1235   39   86   39   92   39   86   39   88
   18 1015  232   39   86   39   88   39   86   39   92   39   86   39
   16    5]
```

[17]:
```
tf.__version__
```

[17]:
```
'2.6.0'
```

0.1 CNN Based model

[18]:
```
# create the model
import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras.layers import␣
 ↪Dense,Dropout,LSTM,Conv1D,MaxPooling1D,LeakyReLU,Flatten,Activation
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import␣
 ↪Embedding,Flatten,Input,Conv2D,MaxPooling2D,Reshape,BatchNormalization
from keras.layers.merge import concatenate
from tensorflow.keras.callbacks import ModelCheckpoint
from clr import clr_callback
from tensorflow.python.keras.callbacks import TensorBoard
from time import time


batchSize = 128
sequence_length = max_sbe_length
channels = 1
embedding_vecor_length = 32
num_filters =64
```

[19]:
```
#def space_to_depth_x2(x):
#    return tf.nn.space_to_depth(x, block_size=8)
```

[20]:
```
input = Input(shape=(sequence_length,), dtype='int32')
embedding = Embedding(input_dim=len(vocab), output_dim=embedding_vecor_length,␣
 ↪input_length=sequence_length)(input)
reshape = Reshape((sequence_length,embedding_vecor_length,1))(embedding)
conv_0 = Conv2D(64, kernel_size=(3, embedding_vecor_length), padding='valid',␣
 ↪kernel_initializer='normal', activation='relu')(reshape)
conv_1 = Conv2D(32, kernel_size=(3, embedding_vecor_length), padding='valid',␣
 ↪kernel_initializer='normal', activation='relu')(reshape)
conv_2 = Conv2D(8, kernel_size=(3, embedding_vecor_length), padding='valid',␣
 ↪kernel_initializer='normal', activation='relu')(reshape)

maxpool_0 = MaxPooling2D(pool_size=(sequence_length - 3 + 1, 1), strides=(1,1),␣
 ↪padding='valid')(conv_0)
maxpool_1 = MaxPooling2D(pool_size=(sequence_length - 3 + 1, 1), strides=(1,1),␣
 ↪padding='valid')(conv_1)
maxpool_2 = MaxPooling2D(pool_size=(sequence_length - 3 + 1, 1), strides=(1,1),␣
 ↪padding='valid')(conv_2)

concatenated_tensor = concatenate([maxpool_0, maxpool_1, maxpool_2])
```

```python
flatten = Flatten()(concatenated_tensor)
dropout = Dropout(0.2)(flatten)
output = Dense(units=1, activation='sigmoid')(dropout)

# this creates a model that includes
model = Model(inputs=input, outputs=output)


print(model.summary())
```

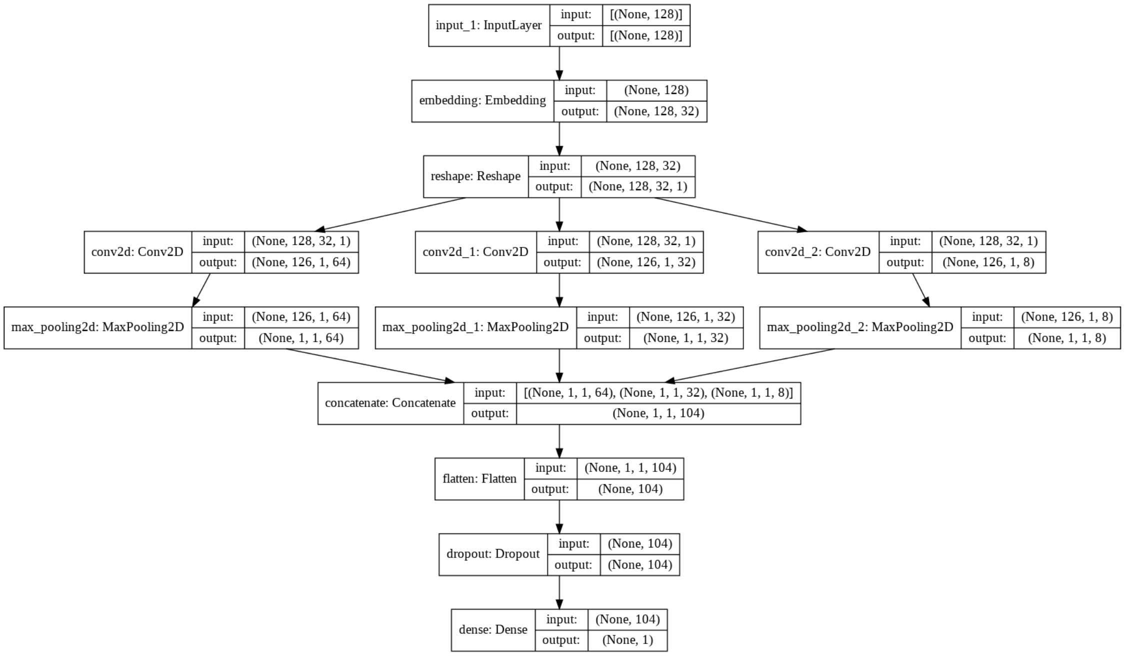
```
Model: "model"
--------------------------------------------------------------------------------
------------------
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_1 (InputLayer)            [(None, 128)]        0
--------------------------------------------------------------------------------
------------------
embedding (Embedding)           (None, 128, 32)      377600      input_1[0][0]
--------------------------------------------------------------------------------
------------------
reshape (Reshape)               (None, 128, 32, 1)   0           embedding[0][0]
--------------------------------------------------------------------------------
------------------
conv2d (Conv2D)                 (None, 126, 1, 64)   6208        reshape[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_1 (Conv2D)               (None, 126, 1, 32)   3104        reshape[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_2 (Conv2D)               (None, 126, 1, 8)    776         reshape[0][0]
--------------------------------------------------------------------------------
------------------
max_pooling2d (MaxPooling2D)    (None, 1, 1, 64)     0           conv2d[0][0]
--------------------------------------------------------------------------------
------------------
max_pooling2d_1 (MaxPooling2D)  (None, 1, 1, 32)     0           conv2d_1[0][0]
--------------------------------------------------------------------------------
------------------
max_pooling2d_2 (MaxPooling2D)  (None, 1, 1, 8)      0           conv2d_2[0][0]
--------------------------------------------------------------------------------
------------------
concatenate (Concatenate)       (None, 1, 1, 104)    0
max_pooling2d[0][0]
max_pooling2d_1[0][0]
max_pooling2d_2[0][0]
```

```
--------------------------------------------------------------------------------
------------------
flatten (Flatten)               (None, 104)          0
concatenate[0][0]

--------------------------------------------------------------------------------
------------------
dropout (Dropout)               (None, 104)          0           flatten[0][0]

--------------------------------------------------------------------------------
------------------
dense (Dense)                   (None, 1)            105         dropout[0][0]
================================================================================
==================
Total params: 387,793
Trainable params: 387,793
Non-trainable params: 0

--------------------------------------------------------------------------------
------------------
None
```

[21]:

```python
from keras.utils.vis_utils import plot_model
plot_model(model, show_shapes=True, show_layer_names=True)
```

[21]:



input_1: InputLayer | input: [(None, 128)] | output: [(None, 128)]
embedding: Embedding | input: (None, 128) | output: (None, 128, 32)
reshape: Reshape | input: (None, 128, 32) | output: (None, 128, 32, 1)
conv2d: Conv2D | input: (None, 128, 32, 1) | output: (None, 126, 1, 64)
conv2d_1: Conv2D | input: (None, 128, 32, 1) | output: (None, 126, 1, 32)
conv2d_2: Conv2D | input: (None, 128, 32, 1) | output: (None, 126, 1, 8)
max_pooling2d: MaxPooling2D | input: (None, 126, 1, 64) | output: (None, 1, 1, 64)
max_pooling2d_1: MaxPooling2D | input: (None, 126, 1, 32) | output: (None, 1, 1, 32)
max_pooling2d_2: MaxPooling2D | input: (None, 126, 1, 8) | output: (None, 1, 1, 8)
concatenate: Concatenate | input: [(None, 1, 1, 64), (None, 1, 1, 32), (None, 1, 1, 8)] | output: (None, 1, 1, 104)
flatten: Flatten | input: (None, 1, 1, 104) | output: (None, 104)
dropout: Dropout | input: (None, 104) | output: (None, 104)
dense: Dense | input: (None, 104) | output: (None, 1)

[22]:

```python
filepath="yolo.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_loss',␣
 ↪verbose=1,save_best_only=True, mode='min')
```

```
callbacks_list = [checkpoint]
model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
```

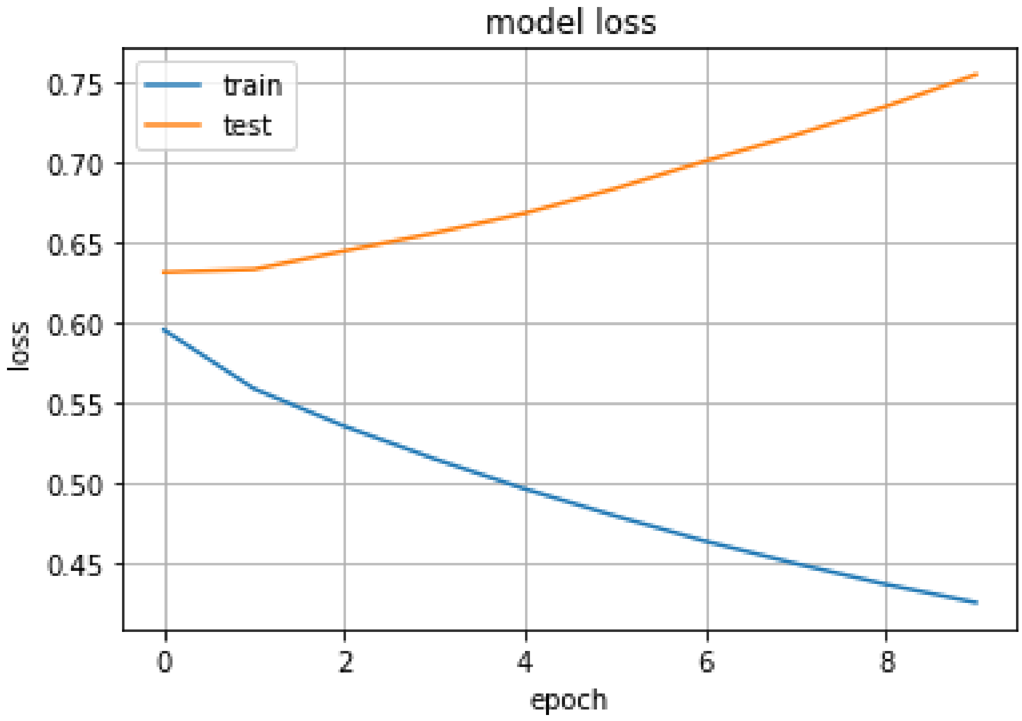
[23]:
```
history = model.fit(X_tr, y_tr, validation_split = 0.3, epochs=10,␣
 ↪verbose=1,batch_size=batchSize, callbacks = callbacks_list)
```

```
Epoch 1/10
4554/4554 [==============================] - 350s 77ms/step - loss: 0.5955 -
accuracy: 0.6769 - val_loss: 0.6315 - val_accuracy: 0.6560

Epoch 00001: val_loss improved from inf to 0.63146, saving model to yolo.hdf5
Epoch 2/10
4554/4554 [==============================] - 350s 77ms/step - loss: 0.5586 -
accuracy: 0.7103 - val_loss: 0.6333 - val_accuracy: 0.6569

Epoch 00002: val_loss did not improve from 0.63146
Epoch 3/10
4554/4554 [==============================] - 350s 77ms/step - loss: 0.5352 -
accuracy: 0.7294 - val_loss: 0.6447 - val_accuracy: 0.6553

Epoch 00003: val_loss did not improve from 0.63146
Epoch 4/10
4554/4554 [==============================] - 350s 77ms/step - loss: 0.5148 -
accuracy: 0.7453 - val_loss: 0.6559 - val_accuracy: 0.6566

Epoch 00004: val_loss did not improve from 0.63146
Epoch 5/10
4554/4554 [==============================] - 369s 81ms/step - loss: 0.4960 -
accuracy: 0.7592 - val_loss: 0.6683 - val_accuracy: 0.6561

Epoch 00005: val_loss did not improve from 0.63146
Epoch 6/10
4554/4554 [==============================] - 349s 77ms/step - loss: 0.4792 -
accuracy: 0.7706 - val_loss: 0.6837 - val_accuracy: 0.6535

Epoch 00006: val_loss did not improve from 0.63146
Epoch 7/10
4554/4554 [==============================] - 349s 77ms/step - loss: 0.4633 -
accuracy: 0.7806 - val_loss: 0.7010 - val_accuracy: 0.6510

Epoch 00007: val_loss did not improve from 0.63146
Epoch 8/10
4554/4554 [==============================] - 349s 77ms/step - loss: 0.4493 -
accuracy: 0.7896 - val_loss: 0.7172 - val_accuracy: 0.6484

Epoch 00008: val_loss did not improve from 0.63146
Epoch 9/10
```

```
4554/4554 [==============================] - 349s 77ms/step - loss: 0.4363 -
accuracy: 0.7970 - val_loss: 0.7350 - val_accuracy: 0.6465

Epoch 00009: val_loss did not improve from 0.63146
Epoch 10/10
4554/4554 [==============================] - 369s 81ms/step - loss: 0.4249 -
accuracy: 0.8043 - val_loss: 0.7550 - val_accuracy: 0.6467

Epoch 00010: val_loss did not improve from 0.63146
```

[24]:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.grid()
plt.show()
```



[25]:

```
from keras.models import load_model
model2 = load_model('yolo.hdf5')
```

[26]:

```
scores = model2.evaluate(X_ts, y_ts, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 56.40%
```

[27]:

```
model2.metrics_names
```

[27]:

```
['loss', 'accuracy']
```

[28]:

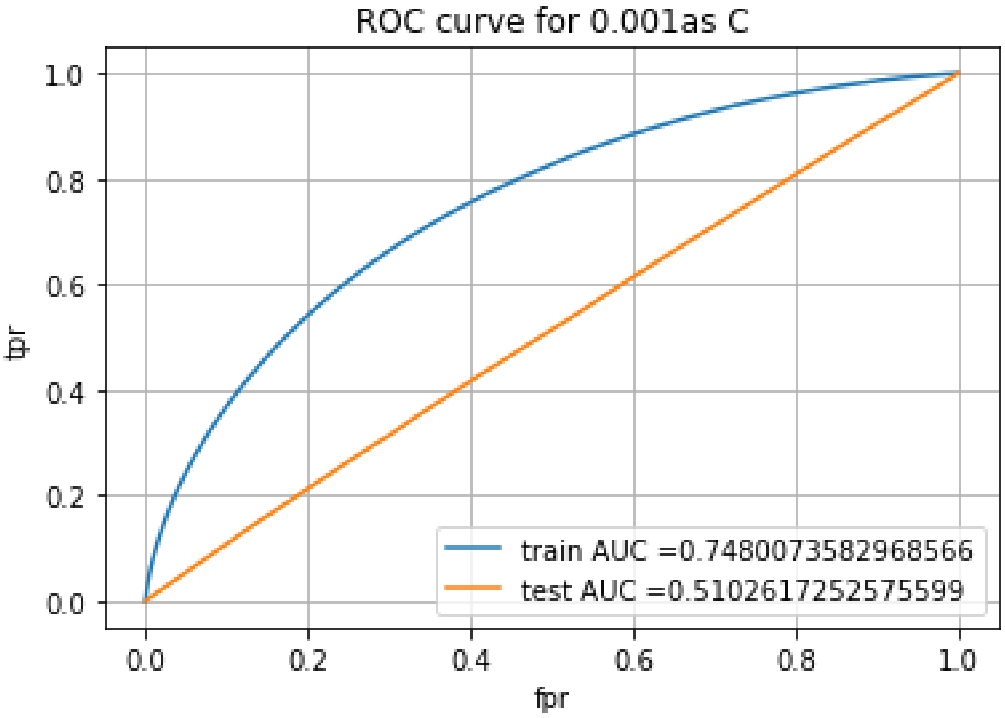
```
y_pr_ts = model2.predict(X_ts)
```

[29]:

```
y_pred_tr = model2.predict(X_tr)
```

[30]:

```
from sklearn.metrics import␣
 ↪roc_curve,auc,confusion_matrix,accuracy_score,precision_score,recall_score,f1_score



train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_pred_tr)
test_fpr, test_tpr, te_thresholds = roc_curve(y_ts, y_pr_ts)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr,␣
 ↪train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title('ROC curve for '+str (0.001)+'as C')
plt.legend()
plt.grid()
plt.show()
```



[31]:

```
# This section of code where ever implemented is taken from sample kNN python␣
 ↪notebook

def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very␣
 ↪high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for␣
 ↪threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


print('test')
best_ts_thres = find_best_threshold(te_thresholds, test_fpr, test_tpr)

print('train')
best_tr_thres = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
```

```
test
the maximum value of tpr*(1-fpr) 0.2573567550077585 for threshold 0.33
train
the maximum value of tpr*(1-fpr) 0.4654546834638477 for threshold 0.346
```

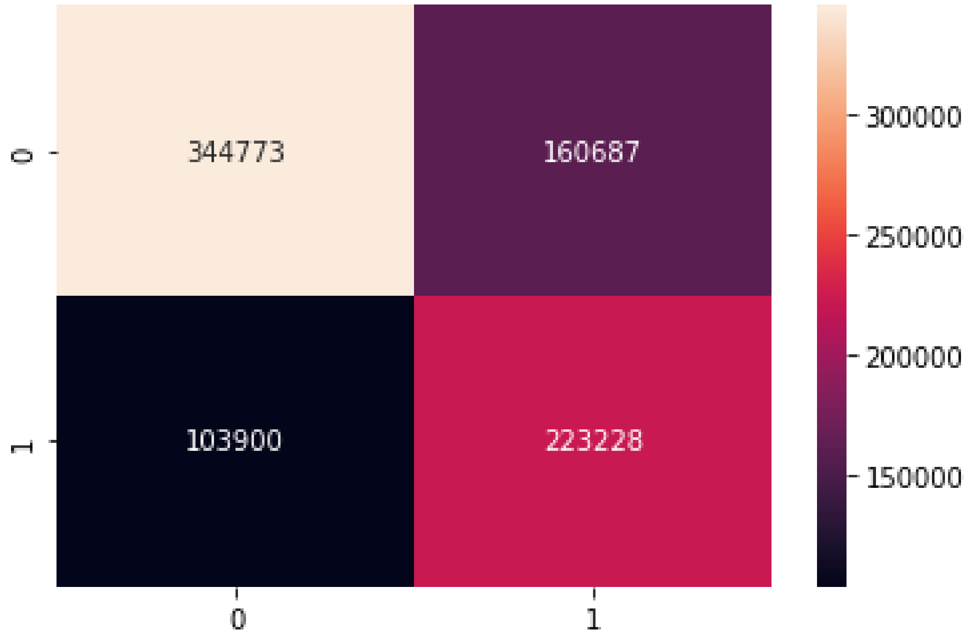
[32]:

```
print('train Confusion Matrix')
cm2 = pd.DataFrame(confusion_matrix(y_tr, predict_with_best_t(y_pred_tr,␣
 ↪best_tr_thres)), range(2),range(2))
sns.heatmap(cm2, annot=True,fmt='g')
```

```
train Confusion Matrix
```

[32]:

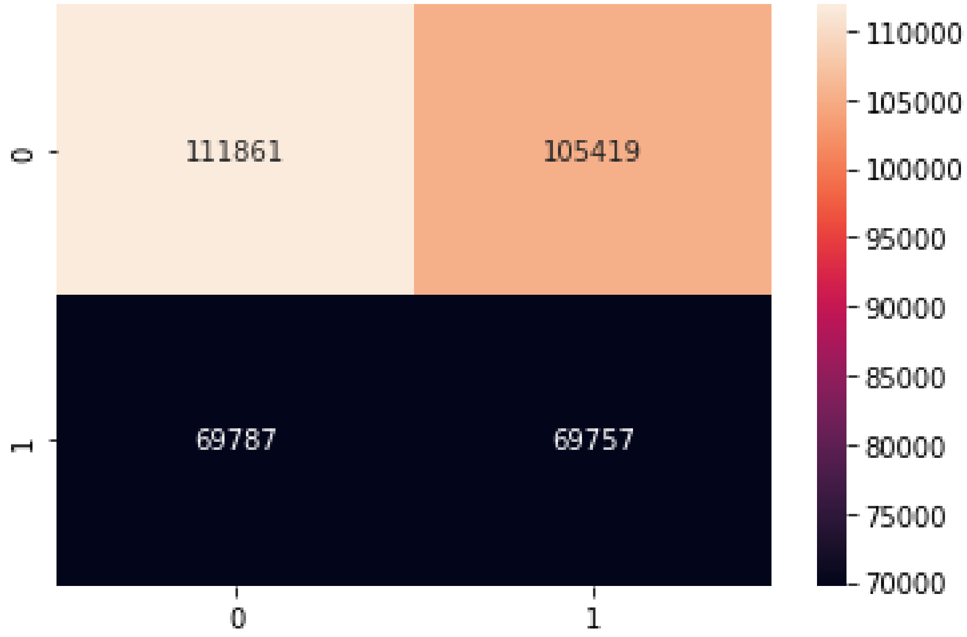
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6f07000990>
```



[33]:
```
print('Test Confusion Matrix')
cm2 = pd.DataFrame(confusion_matrix(y_ts, predict_with_best_t(y_pr_ts,␣
↪best_ts_thres)), range(2),range(2))
sns.heatmap(cm2, annot=True,fmt='g')
```

```
Test Confusion Matrix
```

[33]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6f0447f890>
```



[34]:

```
acc=accuracy_score(y_ts, predict_with_best_t(y_pr_ts, best_ts_thres))*100
ps=precision_score(y_ts, predict_with_best_t(y_pr_ts, best_ts_thres))*100
rc=recall_score(y_ts, predict_with_best_t(y_pr_ts, best_ts_thres))*100
f1=f1_score(y_ts, predict_with_best_t(y_pr_ts, best_ts_thres))*100

print("Accuracy on test set: %0.2f%%"%(acc))
print("Precision on test set: %0.2f%%"%(ps))
print("recall score on test set: %0.2f%%"%(rc))
print("f1 score on test set: %0.2f%%"%(f1))
```

```
Accuracy on test set: 50.90%
Precision on test set: 39.82%
recall score on test set: 49.99%
f1 score on test set: 44.33%
```

[ ]:

[ ]:

[ ]:

0.2 LSTM Based Model

[33]:

```
# create the model
import tensorflow as tf
from tensorflow.keras.layers import␣
 ↪Dense,Dropout,LSTM,Conv1D,MaxPooling1D,Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import TimeDistributed as Tdl
from tensorflow.keras.layers import Embedding
from tensorflow.keras.callbacks import ModelCheckpoint
from clr import clr_callback
from tensorflow.python.keras.callbacks import TensorBoard
from time import time


batchSize = 128
column = max_sbe_length
channels = 1
embedding_vecor_length = 32



model4 = Sequential()
model4.add(Embedding(len(vocab)+1, embedding_vecor_length,␣
 ↪input_length=max_sbe_length))
```

```
model4.add(LSTM(64,return_sequences=True))
model4.add(Dropout(0.3))
model4.add(LSTM(32,return_sequences=True))
model4.add(Flatten())
model4.add(Dropout(0.3))
model4.add(Dense(8,activation='relu'))
model4.add(Dense(1, activation='sigmoid'))
```

[34]:

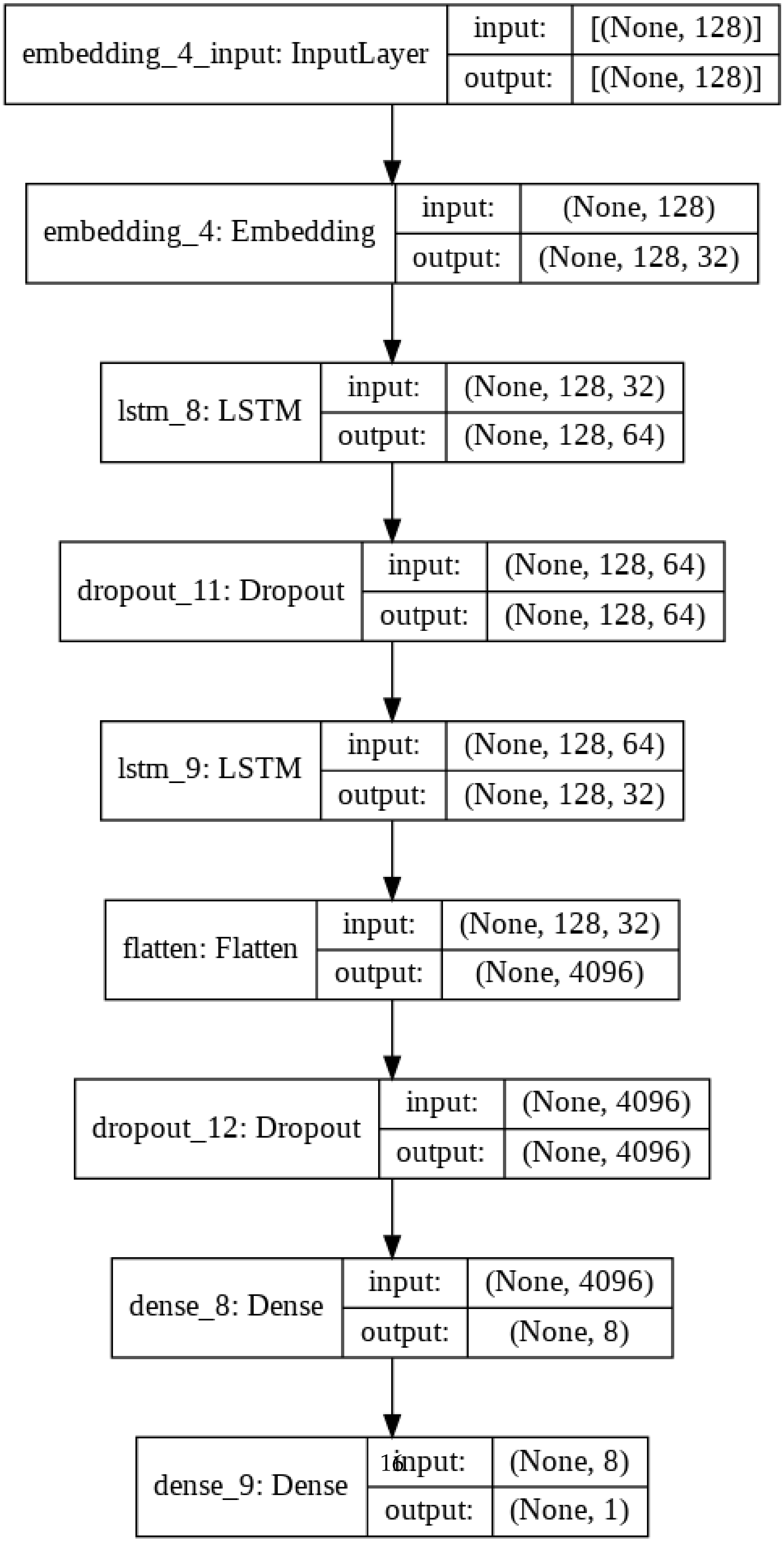
```
model4.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 128, 32)           377632
_________________________________________________________________
lstm_8 (LSTM)                (None, 128, 64)           24832
_________________________________________________________________
dropout_11 (Dropout)         (None, 128, 64)           0
_________________________________________________________________
lstm_9 (LSTM)                (None, 128, 32)           12416
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dropout_12 (Dropout)         (None, 4096)              0
_________________________________________________________________
dense_8 (Dense)              (None, 8)                 32776
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 9
=================================================================
Total params: 447,665
Trainable params: 447,665
Non-trainable params: 0
_________________________________________________________________
```

[35]:

```
from keras.utils.vis_utils import plot_model
plot_model(model4, show_shapes=True, show_layer_names=True)
```

[35]:


embedding_4_input: InputLayer
input:
[(None, 128)]
output:
[(None, 128)]
embedding_4: Embedding
input:
(None, 128)
output:
(None, 128, 32)
lstm_8: LSTM
input:
(None, 128, 32)
output:
(None, 128, 64)
dropout_11: Dropout
input:
(None, 128, 64)
output:
(None, 128, 64)
lstm_9: LSTM
input:
(None, 128, 64)
output:
(None, 128, 32)
flatten: Flatten
input:
(None, 128, 32)
output:
(None, 4096)
dropout_12: Dropout
input:
(None, 4096)
output:
(None, 4096)
dense_8: Dense
input:
(None, 4096)
output:
(None, 8)
dense_9: Dense
input:
(None, 8)
output:
(None, 1)

[36]:

```
filepath="tdl.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_loss',␣
 ↪verbose=1,save_best_only=True, mode='min')

callbacks_list = [checkpoint]
model4.compile(loss='binary_crossentropy',␣
 ↪optimizer='adam',metrics=['accuracy'])
```

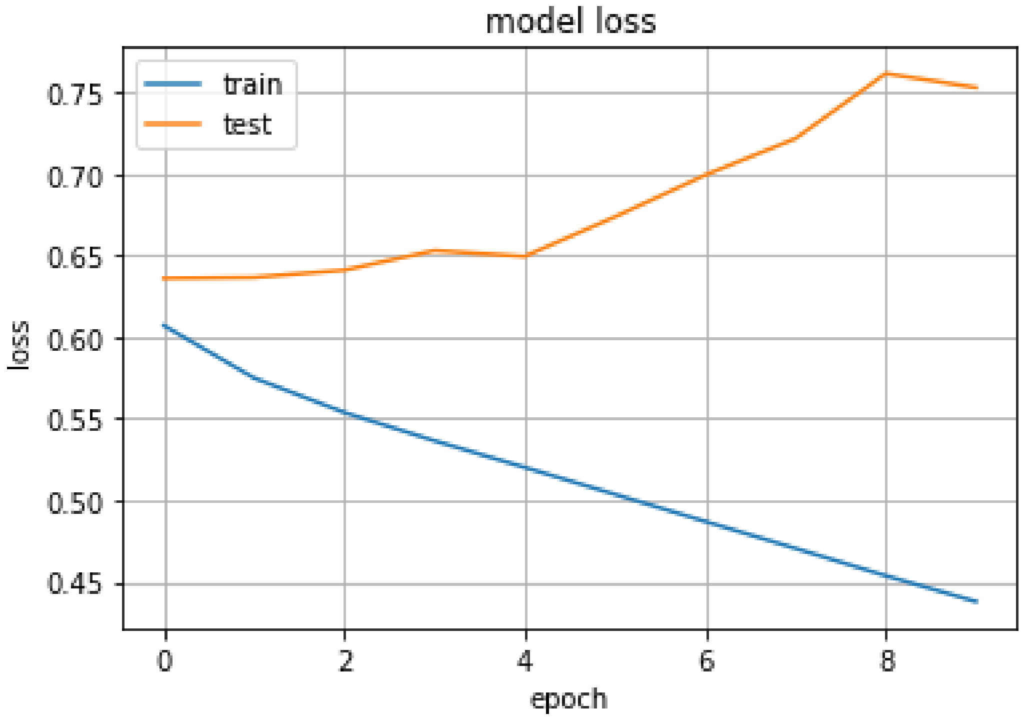
[37]:

```
history2 = model4.fit(X_tr, y_tr, validation_split = 0.3, epochs=10,␣
 ↪verbose=1,batch_size=batchSize, callbacks = callbacks_list)
```

```
Epoch 1/10
4554/4554 [==============================] - 1319s 289ms/step - loss: 0.6072 -
accuracy: 0.6697 - val_loss: 0.6362 - val_accuracy: 0.6471

Epoch 00001: val_loss improved from inf to 0.63616, saving model to tdl.hdf5
Epoch 2/10
4554/4554 [==============================] - 1313s 288ms/step - loss: 0.5750 -
accuracy: 0.6963 - val_loss: 0.6367 - val_accuracy: 0.6511

Epoch 00002: val_loss did not improve from 0.63616
Epoch 3/10
4554/4554 [==============================] - 1334s 293ms/step - loss: 0.5538 -
accuracy: 0.7130 - val_loss: 0.6411 - val_accuracy: 0.6505

Epoch 00003: val_loss did not improve from 0.63616
Epoch 4/10
4554/4554 [==============================] - 1329s 292ms/step - loss: 0.5364 -
accuracy: 0.7257 - val_loss: 0.6531 - val_accuracy: 0.6560

Epoch 00004: val_loss did not improve from 0.63616
Epoch 5/10
4554/4554 [==============================] - 1326s 291ms/step - loss: 0.5202 -
accuracy: 0.7375 - val_loss: 0.6495 - val_accuracy: 0.6562

Epoch 00005: val_loss did not improve from 0.63616
Epoch 6/10
4554/4554 [==============================] - 1316s 289ms/step - loss: 0.5037 -
accuracy: 0.7487 - val_loss: 0.6738 - val_accuracy: 0.6531

Epoch 00006: val_loss did not improve from 0.63616
Epoch 7/10
4554/4554 [==============================] - 1323s 291ms/step - loss: 0.4870 -
accuracy: 0.7598 - val_loss: 0.6994 - val_accuracy: 0.6471

Epoch 00007: val_loss did not improve from 0.63616
Epoch 8/10
4554/4554 [==============================] - 1317s 289ms/step - loss: 0.4705 -
accuracy: 0.7706 - val_loss: 0.7218 - val_accuracy: 0.6503

Epoch 00008: val_loss did not improve from 0.63616
Epoch 9/10
4554/4554 [==============================] - 1317s 289ms/step - loss: 0.4537 -
accuracy: 0.7807 - val_loss: 0.7617 - val_accuracy: 0.6486

Epoch 00009: val_loss did not improve from 0.63616
Epoch 10/10
4554/4554 [==============================] - 1327s 291ms/step - loss: 0.4379 -
accuracy: 0.7907 - val_loss: 0.7532 - val_accuracy: 0.6465

Epoch 00010: val_loss did not improve from 0.63616
```

[38]:

```
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.grid()
plt.show()
```



[17]:

```
from keras.models import load_model
model2 = load_model('tdl.hdf5')
```

[42]:

```
scores = model2.evaluate(X_ts, y_ts, verbose=0)
print(scores)
```

```
[0.751381516456604, 0.5553241968154907]
```

[43]:

```
model2.metrics_names
```

[43]:

```
['loss', 'accuracy']
```

[18]:

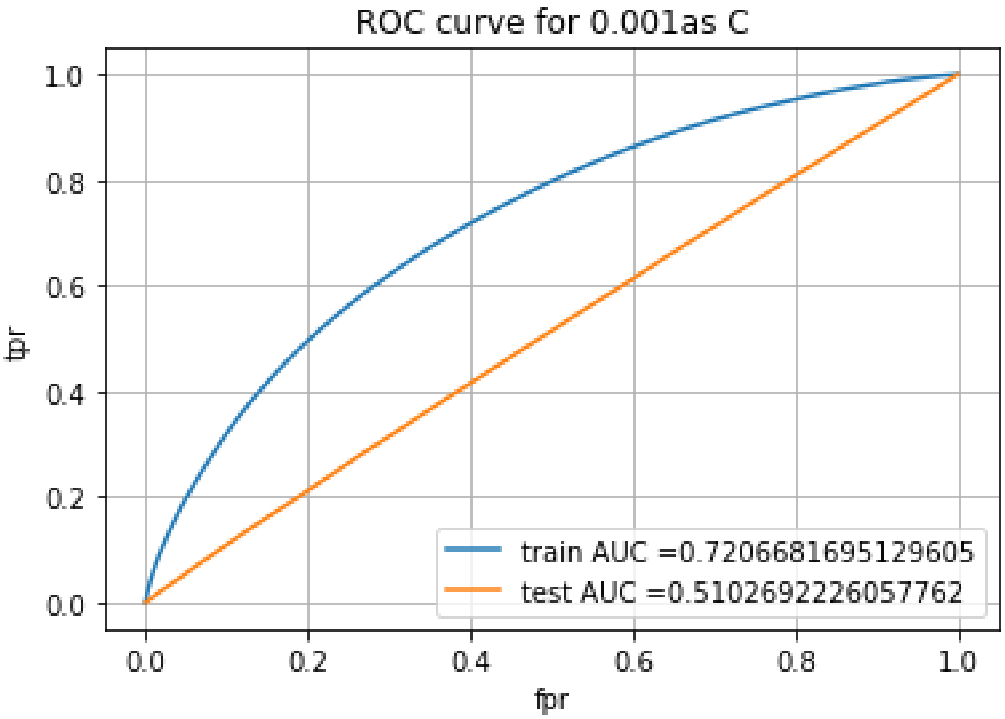
```
y_pr_ts = model2.predict(X_ts)
```

[19]:

```
y_pred_tr = model2.predict(X_tr)
```

[20]:

```
from sklearn.metrics import␣
 ↪roc_curve,auc,confusion_matrix,accuracy_score,precision_score,recall_score,f1_score



train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_pred_tr)
test_fpr, test_tpr, te_thresholds = roc_curve(y_ts, y_pr_ts)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr,␣
 ↪train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title('ROC curve for '+str (0.001)+'as C')
plt.legend()
plt.grid()
plt.show()
```



[21]:

```
# This section of code where ever implemented is taken from sample kNN python notebook

def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions


print('test')
best_ts_thres = find_best_threshold(te_thresholds, test_fpr, test_tpr)
```

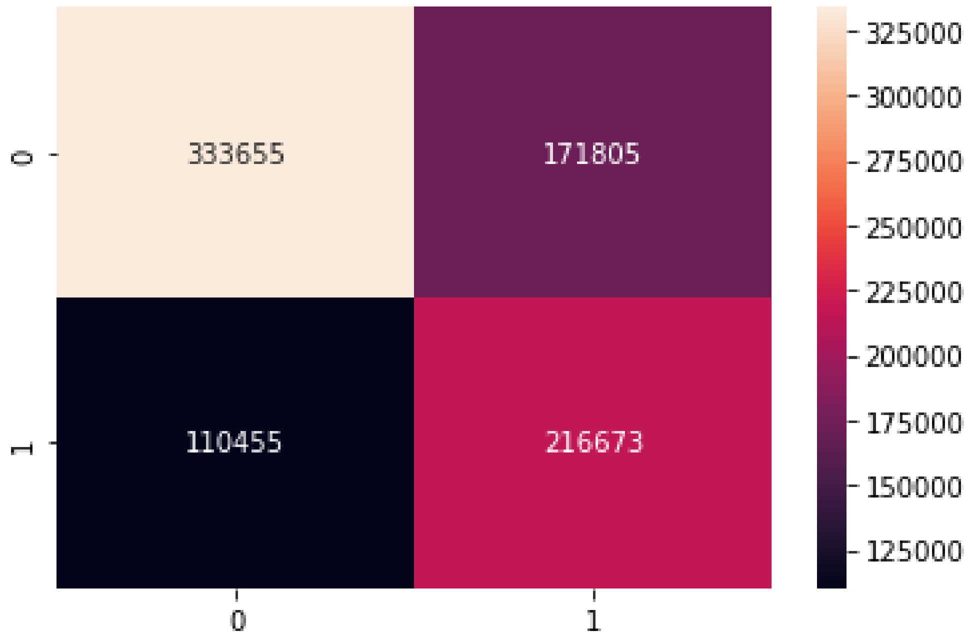
```
print('train')
best_tr_thres = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
```

```
test
the maximum value of tpr*(1-fpr) 0.2571559455473823 for threshold 0.362
train
the maximum value of tpr*(1-fpr) 0.4372178883491832 for threshold 0.378
```

[22]:
```
print('train Confusion Matrix')
cm2 = pd.DataFrame(confusion_matrix(y_tr, predict_with_best_t(y_pred_tr,␣
 ↪best_tr_thres)), range(2),range(2))
sns.heatmap(cm2, annot=True,fmt='g')
```

```
train Confusion Matrix
```

[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9167c4dd90>
```

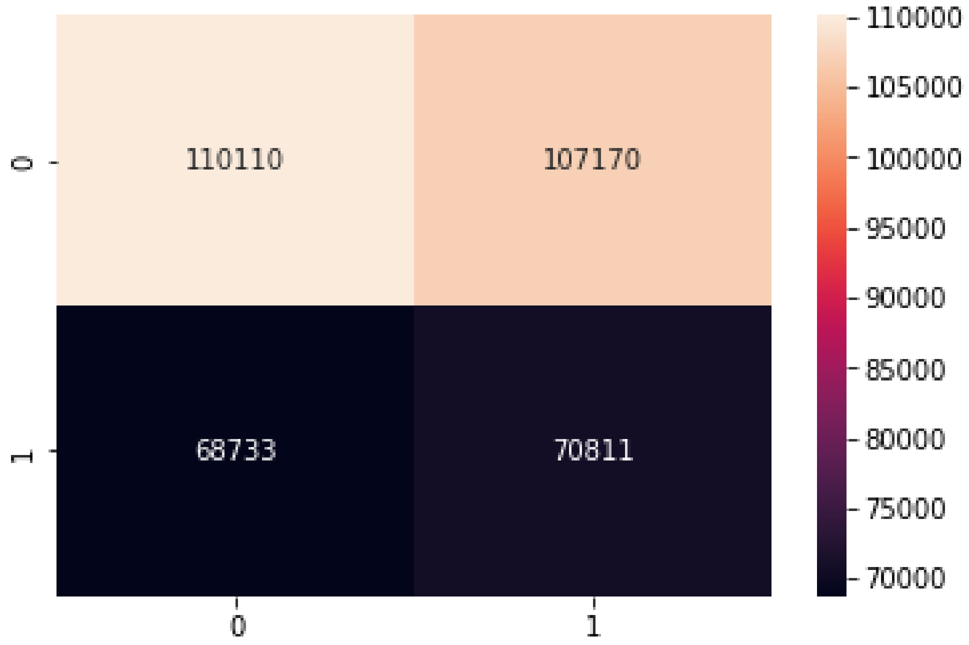


[23]:
```
print('Test Confusion Matrix')
cm2 = pd.DataFrame(confusion_matrix(y_ts, predict_with_best_t(y_pr_ts,␣
 ↪best_ts_thres)), range(2),range(2))
sns.heatmap(cm2, annot=True,fmt='g')
```

```
Test Confusion Matrix
```

[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90bfc8ce10>
```



[24]:

```
acc=accuracy_score(y_ts, predict_with_best_t(y_pr_ts, best_ts_thres))*100
ps=precision_score(y_ts, predict_with_best_t(y_pr_ts, best_ts_thres))*100
rc=recall_score(y_ts, predict_with_best_t(y_pr_ts, best_ts_thres))*100
f1=f1_score(y_ts, predict_with_best_t(y_pr_ts, best_ts_thres))*100

print("Accuracy on test set: %0.2f%%"%(acc))
print("Precision on test set: %0.2f%%"%(ps))
print("recall score on test set: %0.2f%%"%(rc))
print("f1 score on test set: %0.2f%%"%(f1))
```

```
Accuracy on test set: 50.70%
Precision on test set: 39.79%
recall score on test set: 50.74%
f1 score on test set: 44.60%
```

[ ]:

```

```