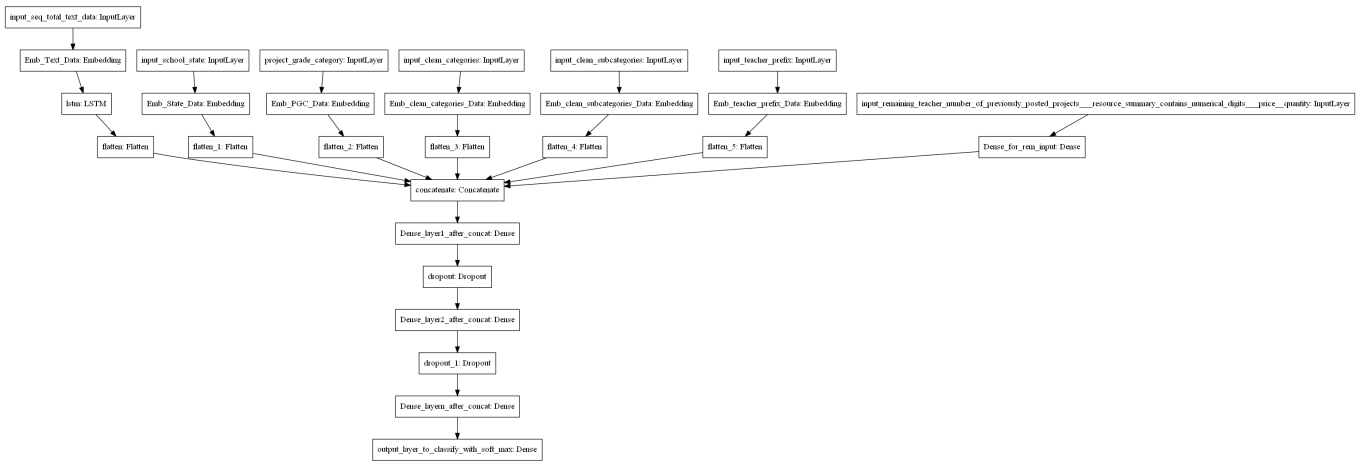
Assignment : 14

1. Preprocess all the Data we have in DonorsChoose Dataset (https://drive.google.com/drive/folders/1MIwK7BQMev8f5CbDDVNLPaFGB32pFN60) use train.csv
2. Combine 4 essay's into one column named - 'preprocessed_essays'.
3. After step 2 you have to train 3 types of models as discussed below.
4. For all the model use 'auc' (https://scikit-learn.org/stable/modules/model_evaluation.html#roc-metrics) as a metric. check this (https://datascience.stackexchange.com/a/20192) for using auc as a metric
5. You are free to choose any number of layers/hiddden units but you have to use same type of architectures shown below.
6. You can use any one of the optimizers and choice of Learning rate and momentum, resources: cs231n class notes (http://cs231n.github.io/neural-networks-3/), cs231n class video (https://www.youtube.com/watch?v=hd_KFJ5ktUc).
7. For all the model's use TensorBoard (https://www.youtube.com/watch?v=2U6Jl7oqRkM) and plot the Metric value and Loss with epoch. While submitting, take a screenshot of plots and include those images in .ipynb notebook and PDF.
8. Use Categorical Cross Entropy as Loss to minimize.

Model-1

Build and Train deep neural network as shown below



ref: https://i.imgur.com/w395Yk9.png (https://i.imgur.com/w395Yk9.png)

- **Input_seq_total_text_data** --- You have to give Total text data columns. After this use the Embedding layer to get word vectors. Use given predefined glove word vectors, don't train any word vectors. After this use LSTM and get the LSTM output and Flatten that output.
- **Input_school_state** --- Give 'school_state' column as input to embedding layer and Train the Keras Embedding layer.
- **Project_grade_category** --- Give 'project_grade_category' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_clean_categories** --- Give 'input_clean_categories' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_clean_subcategories** --- Give 'input_clean_subcategories' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_clean_subcategories** --- Give 'input_teacher_prefix' column as input to embedding layer and Train the Keras Embedding layer.
- **Input_remaining_teacher_number_of_previously_posted_projects._resource_summary_contains_** ---concatenate remaining columns and add a Dense layer after that.

- For LSTM, you can choose your sequence padding methods on your own or you can train your LSTM without padding, there is no restriction on that.

Below is an example of embedding layer for a categorical columns. In below code all are dummy values, we gave only for referance.

In [ ]:

```
# https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-layer-work
input_layer = Input(shape=(n,))
embedding = Embedding(no_1, no_2, input_length=n)(input_layer)
flatten = Flatten()(embedding)
```

1. Go through this blog, if you have any doubt on using predefined Embedding values in Embedding layer - https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/ (https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/)

2. Please go through this link https://keras.io/getting-started/functional-api-guide/ (https://keras.io/getting-started/functional-api-guide/) and check the 'Multi-input and multi-output models' then you will get to know how to give multiple inputs.

Model-2

Use the same model as above but for 'input_seq_total_text_data' give only some words in the sentance not all the words. Filter the words as below.

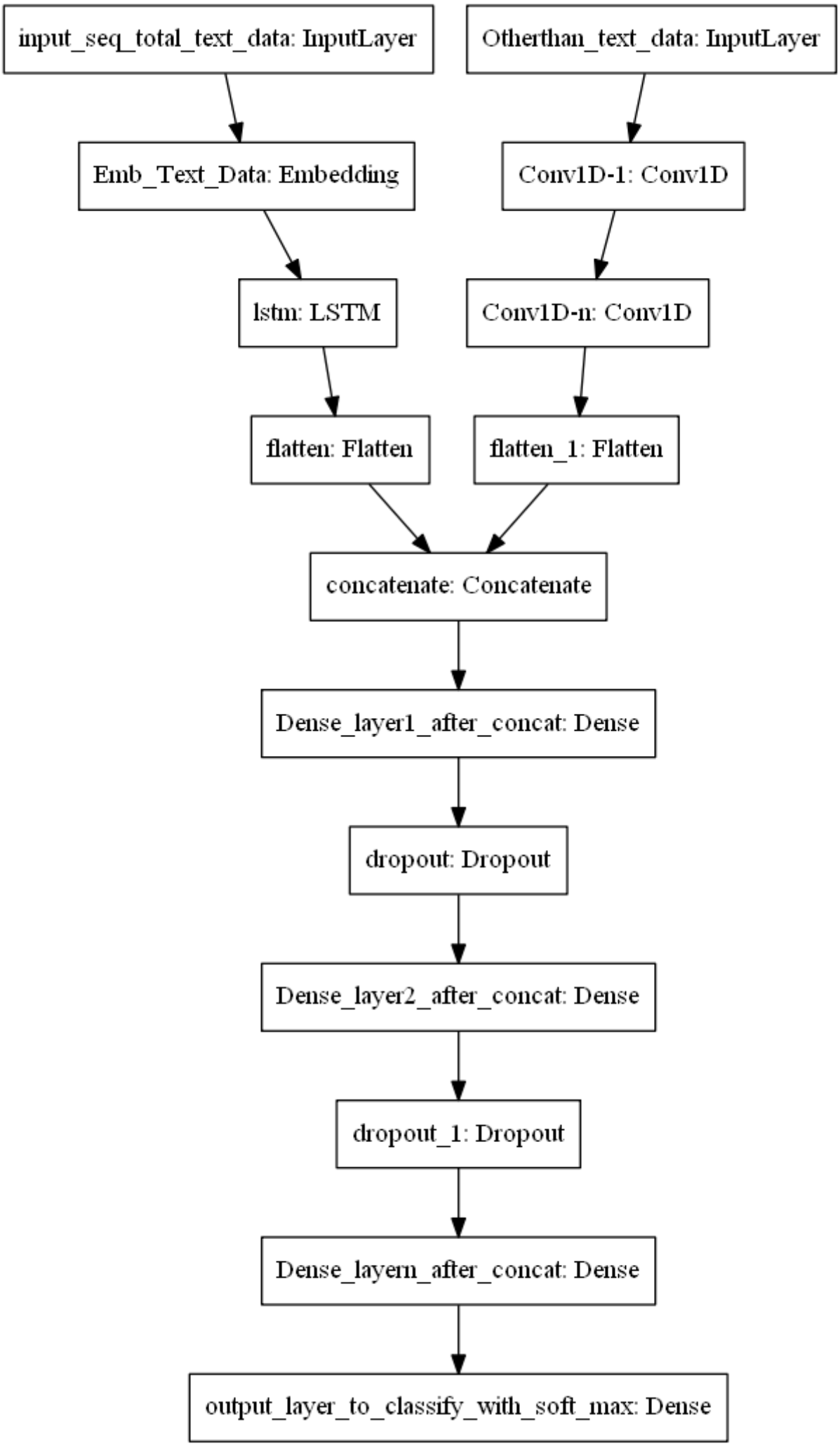
1. Train the TF-IDF on the Train data

2. Get the idf value for each word we have in the train data.

3. Remove the low idf value and high idf value words from our data. Do some analysis on the Idf values and based on those values choose the low and high threshold value. Because very frequent words and very very rare words don't give much information. (you can plot a box plots and take only the idf scores within IQR range and corresponding words)

4. Train the LSTM after removing the Low and High idf value words. (In model-1 Train on total data but in Model-2 train on data after removing some words based on IDF values)

Model-3


input_seq_total_text_data: InputLayer
Otherthan_text_data: InputLayer
Emb_Text_Data: Embedding
Conv1D-1: Conv1D
lstm: LSTM
Conv1D-n: Conv1D
flatten: Flatten
flatten_1: Flatten
concatenate: Concatenate
Dense_layer1_after_concat: Dense
dropout: Dropout
Dense_layer2_after_concat: Dense
dropout_1: Dropout
Dense_layern_after_concat: Dense
output_layer_to_classify_with_soft_max: Dense

ref: https://i.imgur.com/fkQ8nGo.png (https://i.imgur.com/fkQ8nGo.png)

- **input_seq_total_text_data**:

  . Use text column('essay'), and use the Embedding layer to get word vectors.

  . Use given predefined glove word vectors, don't train any word vectors.

  . Use LSTM that is given above, get the LSTM output and Flatten that output.

  . You are free to preprocess the input text as you needed.

- **Other_than_text_data**:

  . Convert all your Categorical values to onehot coded and then concatenate all these onehot vectors

  . Neumerical values and use CNN1D (https://keras.io/getting-started/sequential-model-guide/#sequence-classification-with-1d-convolutions) as shown in above figure.

  . You are free to choose all CNN parameters like kernel sizes, stride.

</pre>

In [1]:

```
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers,Input
from keras.layers import SpatialDropout1D, LSTM, BatchNormalization,concatenate,Flatten
,Dense,Dropout,MaxPooling2D,Reshape,Embedding

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
import pickle
import pandas as pd
import os
from keras.initializers import he_normal,glorot_normal
from keras.regularizers import l1,l2
from keras import Model,Input
from time import time
from keras.callbacks import ModelCheckpoint
from tensorflow.python.keras.callbacks import TensorBoard
#Train test split
from sklearn.model_selection import train_test_split
#to categorical
from keras.utils import to_categorical
from keras.preprocessing.text import one_hot
#Text Tokenizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from tqdm import tqdm
#hstack
from scipy.sparse import hstack
from sklearn.metrics import roc_auc_score
from keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from keras.layers.convolutional import Conv2D,Conv1D
```

In [ ]:

```
!wget 'http://nlp.stanford.edu/data/glove.6B.zip'
!unzip '/content/glove.6B.zip'
```

In [10]:

```
embeddings_index = {}
dbfile = open(os.path.join('glove.6B.100d.txt'))
for line in dbfile:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
dbfile.close()
```

In [75]:

```
#Reading the dataset
project_data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/LSTM/preprocessed_data.csv')
project_data.shape

#Formatting the dataframe removing columns and including the labels
class_label = project_data['project_is_approved']
project_data['remaining_input'] = project_data['teacher_number_of_previously_posted_projects']  +\
                                    project_data['price']

col = ['project_is_approved','teacher_number_of_previously_posted_projects','price']
project_data.drop(labels=col,axis =1, inplace=True)

#printing the columns in the dataframe
col = project_data.columns
print(col)

#performing train test split
#Stratify parameter
#For example, if variable y is a binary categorical variable with values 0 and 1 and there are 25% of zeros and 75% of ones, stratify=y will make sure that your random split has 25% of 0's and 75% of 1's.
train,test,y_train,y_test = train_test_split(project_data, class_label , train_size = 0.7,stratify=class_label)

print("Shape of the Train dataset: ", train.shape[0])
print("Shape of the Test dataset: ", test.shape[0])

#converting class labels to categorical variables
#We are using categorigal cross entrophy loss function and softmax classifier,therefor we are encoding the labels
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'clean_categories', 'clean_subcategories', 'essay', 'remaining_input'],
      dtype='object')
Shape of the Train dataset:  76473
Shape of the Test dataset:  32775
```

In [76]:

```
#USER DEFINED FUNCTION 2 CUSTOM MERTRIC AUC
def auc( y_true, y_pred ) :
    score = tf.py_function( lambda y_true, y_pred : roc_auc_score( y_true, y_pred, average='macro', sample_weight=None).astype('float32'),
                        [y_true, y_pred],
                        'float32',
                        name='sklearnAUC' )
    return score
```

In [77]:

```
#Tokenizing the Essay column
# prepare tokenizer
t = Tokenizer()
t.fit_on_texts(train['essay'])
vocab_size_essay = len(t.word_index) + 1
word_index=t.word_index
# integer encode the documents
encoded_essay = t.texts_to_sequences(train['essay'])

# pad documents to a max length of 250 words
max_length = 250
padded_essay = pad_sequences(encoded_essay, maxlen=max_length, padding='post')

# integer encode the documents
#One hot encoding and padding for input to embedding layer
vocab_size = 52
encoded_state = [one_hot(d, vocab_size) for d in train['school_state']]
max_length = 52
encoded_state = pad_sequences(encoded_state, maxlen=max_length, padding='post')

vocab_size = 5
encoded_proj_grade = [one_hot(d, vocab_size) for d in train['project_grade_category']]
max_length = 5
encoded_proj_grade = pad_sequences(encoded_proj_grade, maxlen=max_length, padding='pos
t')


vocab_size = 50
encoded_cat = [one_hot(d, vocab_size) for d in train['clean_categories']]
max_length = 50
encoded_cat = pad_sequences(encoded_cat, maxlen=max_length, padding='post')


vocab_size = 385
encoded_sub_cat = [one_hot(d, vocab_size) for d in train['clean_subcategories']]
max_length = 385
encoded_sub_cat = pad_sequences(encoded_sub_cat, maxlen=max_length, padding='post')

vocab_size = 6
encoded_prefix = [one_hot(d, vocab_size) for d in train['teacher_prefix']]
max_length = 6
encoded_prefix = pad_sequences(encoded_prefix, maxlen=max_length, padding='post')
```

In [78]:

```
#USER DEFINED FUNCTION 1 FOR GLOVE EMBEDDING OF TEXT FEATURES
#Embedding function to be used for embedding text features by using pre defined glove v
ectors
#Creating a matrix with rows as words and columns with 100 dim vectors for each word
def embedding_mat(word_index,embedding_dim = 100):
    embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
    #embedding_index=db
    for word,i in word_index.items():
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector
    return embedding_matrix

embedding_matrix=embedding_mat(word_index)
```

In [ ]:

```
```

In [79]:

```
#Encoding test data
#Tokenizing the Essay column
# prepare tokenizer
#t = Tokenizer()
#t.fit_on_texts(test['essay'])
#vocab_size_essay = len(t.word_index) + 1
#word_index_test=t.word_index

# integer encode the documents
encoded_essay_test = t.texts_to_sequences(test['essay'])

# pad documents to a max length of 250 words
max_length = 250
padded_essay_test = pad_sequences(encoded_essay_test, maxlen=max_length, padding='post'
)

# integer encode the documents
#One hot encoding and padding for input to embedding layer
vocab_size = 52
encoded_state_test = [one_hot(d, vocab_size) for d in test['school_state']]
max_length = 52
encoded_state_test = pad_sequences(encoded_state_test, maxlen=max_length, padding='pos
t')

vocab_size = 5
encoded_proj_grade_test = [one_hot(d, vocab_size) for d in test['project_grade_categor
y']]
max_length = 5
encoded_proj_grade_test = pad_sequences(encoded_proj_grade_test, maxlen=max_length, pad
ding='post')


vocab_size = 50
encoded_cat_test = [one_hot(d, vocab_size) for d in test['clean_categories']]
max_length = 50
encoded_cat_test = pad_sequences(encoded_cat_test, maxlen=max_length, padding='post')


vocab_size = 385
encoded_sub_cat_test = [one_hot(d, vocab_size) for d in test['clean_subcategories']]
max_length = 385
encoded_sub_cat_test = pad_sequences(encoded_sub_cat_test, maxlen=max_length, padding=
'post')

vocab_size = 6
encoded_prefix_test = [one_hot(d, vocab_size) for d in test['teacher_prefix']]
max_length = 6
encoded_prefix_test = pad_sequences(encoded_prefix_test, maxlen=max_length, padding='po
st')
```

Model 1

In [ ]:

```
#input 1
input1 = Input(shape=(250,))
x1 = Embedding(input_dim=len(word_index) + 1,output_dim= 100,weights=[embedding_matrix
],trainable=False)(input1)
x1 = LSTM(32,return_sequences=True)(x1)
x1 = layers.Dropout(0.5)(x1)
x1 = Flatten()(x1)

cat_vars=['school_state','project_grade_category','clean_categories','clean_subcategori
es','teacher_prefix']
cat_sizes={}
cat_embsizes={}
for cat in cat_vars:
  cat_sizes[cat]=train[cat].nunique()
  cat_embsizes[cat]=min(50,cat_sizes[cat]//2 +1)

#input 2
input2 = Input(shape=(52,))
x2 = Embedding(input_dim= cat_sizes['school_state']+1,output_dim= cat_embsizes['school_
state'])(input2)
x2 = Flatten()(x2)

#input 3
input3 = Input(shape=(5,))
x3 = Embedding(input_dim= cat_sizes['project_grade_category']+1,output_dim= cat_embsize
s['project_grade_category'])(input3)
x3 = Flatten()(x3)

#input 4
input4 = Input(shape=(50,))
x4 = Embedding(input_dim=cat_sizes['clean_categories']+1,output_dim= cat_embsizes['clea
n_categories'])(input4)
x4 = Flatten()(x4)

#input 5
input5 = Input(shape=(385,))
x5 = Embedding(input_dim= cat_sizes['clean_subcategories']+1,output_dim= cat_embsizes[
'clean_subcategories'])(input5)
x5 = Flatten()(x5)

#input 6
input6 = Input(shape=(6,))
x6 = Embedding(input_dim= cat_sizes['teacher_prefix']+1,output_dim= cat_embsizes['teach
er_prefix'])(input6)
x6 = Flatten()(x6)

#input 7
input7 = Input(shape=(1,))
x7 = Dense(4,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0
001))(input7)
x7 = layers.Dropout(0.5)(x7)
#merging all the inputs
concat = concatenate([x1,x2,x3,x4,x5,x6,x7])

x = Dense(64,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0
001))(concat)
x = layers.Dropout(0.2)(x)
x = Dense(32,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0
001))(x)
```

```
x = layers.Dropout(0.2)(x)
x = Dense(16,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0
001))(x)
x = layers.Dropout(0.2)(x)
x = Dense(8,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.00
01))(x)
x = layers.Dropout(0.2)(x)
output = Dense(2, activation = 'softmax')(x)

# create model with seven inputs
model = Model([input1,input2,input3,input4,input5,input6,input7], output)
tensorboard = keras.callbacks.TensorBoard(log_dir='lstm_logs/{}'.format(time()))
model.compile(optimizer=keras.optimizers.Adam(),loss='categorical_crossentropy',metrics
=['accuracy',auc])

print(model.summary())
```

```
Model: "functional_1"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
input_1 (InputLayer)            [(None, 250)]        0
__________________________________________________________________________________________
embedding (Embedding)           (None, 250, 100)     4910200     input_1[0][0]
__________________________________________________________________________________________
lstm (LSTM)                     (None, 250, 32)      17024       embedding[0][0]
__________________________________________________________________________________________
input_2 (InputLayer)            [(None, 52)]         0
__________________________________________________________________________________________
input_3 (InputLayer)            [(None, 5)]          0
__________________________________________________________________________________________
input_4 (InputLayer)            [(None, 50)]         0
__________________________________________________________________________________________
input_5 (InputLayer)            [(None, 385)]        0
__________________________________________________________________________________________
input_6 (InputLayer)            [(None, 6)]          0
__________________________________________________________________________________________
input_7 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________
dropout (Dropout)               (None, 250, 32)      0           lstm[0][0]
__________________________________________________________________________________________
embedding_1 (Embedding)         (None, 52, 26)       1352        input_2[0][0]
__________________________________________________________________________________________
embedding_2 (Embedding)         (None, 5, 3)         15          input_3[0][0]
__________________________________________________________________________________________
embedding_3 (Embedding)         (None, 50, 26)       1352        input_4[0][0]
__________________________________________________________________________________________
embedding_4 (Embedding)         (None, 385, 50)      19700       input_5[0][0]
__________________________________________________________________________________________
embedding_5 (Embedding)         (None, 6, 3)         18          input_6[0][0]
__________________________________________________________________________________________
dense (Dense)                   (None, 4)            8           input_7
```

```
[0][0]
__________________________________________________________________________________________________
__________________________
flatten (Flatten)               (None, 8000)         0           dropout
[0][0]
__________________________________________________________________________________________________
__________________________
flatten_1 (Flatten)             (None, 1352)         0           embeddi
ng_1[0][0]
__________________________________________________________________________________________________
__________________________
flatten_2 (Flatten)             (None, 15)           0           embeddi
ng_2[0][0]
__________________________________________________________________________________________________
__________________________
flatten_3 (Flatten)             (None, 1300)         0           embeddi
ng_3[0][0]
__________________________________________________________________________________________________
__________________________
flatten_4 (Flatten)             (None, 19250)        0           embeddi
ng_4[0][0]
__________________________________________________________________________________________________
__________________________
flatten_5 (Flatten)             (None, 18)           0           embeddi
ng_5[0][0]
__________________________________________________________________________________________________
__________________________
dropout_1 (Dropout)             (None, 4)            0           dense
[0][0]
__________________________________________________________________________________________________
__________________________
concatenate (Concatenate)       (None, 29939)        0           flatten
[0][0]
                                                                 flatten
_1[0][0]
                                                                 flatten
_2[0][0]
                                                                 flatten
_3[0][0]
                                                                 flatten
_4[0][0]
                                                                 flatten
_5[0][0]
                                                                 dropout
_1[0][0]
__________________________________________________________________________________________________
__________________________
dense_1 (Dense)                 (None, 64)           1916160     concate
nate[0][0]
__________________________________________________________________________________________________
__________________________
dropout_2 (Dropout)             (None, 64)           0           dense_1
[0][0]
__________________________________________________________________________________________________
__________________________
dense_2 (Dense)                 (None, 32)           2080        dropout
_2[0][0]
__________________________________________________________________________________________________
__________________________
dropout_3 (Dropout)             (None, 32)           0           dense_2
[0][0]
```

```
_________________________________________________________________
_____________________________
dense_3 (Dense)              (None, 16)            528         dropout
_3[0][0]
_________________________________________________________________
_____________________________
dropout_4 (Dropout)          (None, 16)            0           dense_3
[0][0]
_________________________________________________________________
_____________________________
dense_4 (Dense)              (None, 8)             136         dropout
_4[0][0]
_________________________________________________________________
_____________________________
dropout_5 (Dropout)          (None, 8)             0           dense_4
[0][0]
_________________________________________________________________
_____________________________
dense_5 (Dense)              (None, 2)             18          dropout
_5[0][0]
==================================================================
==========================
Total params: 6,868,591
Trainable params: 1,958,391
Non-trainable params: 4,910,200
_________________________________________________________________
_____________________________
None
```

In [ ]:

```
#Ref:#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
filepath="/content/drive/My Drive/Colab Notebooks/LSTM/weights.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_auc', verbose=1, save_best_only=Tru
e, mode='max')
earlystop = EarlyStopping(monitor='val_auc', patience=2, verbose=1)
callbacks_list = [checkpoint,tensorboard,earlystop]

model.fit([padded_essay,encoded_state,encoded_proj_grade,encoded_cat,encoded_sub_cat,
           encoded_prefix,train['remaining_input']], y_train, epochs=10,verbose=1,batch
_size=256,
           validation_split=0.33,callbacks = callbacks_list)
```

```
Epoch 1/10
  1/201 [..............................] - ETA: 0s - loss: 14.6815 - accur
acy: 0.8047 - auc: 0.3926WARNING:tensorflow:From /usr/local/lib/python3.6/
dist-packages/tensorflow/python/ops/summary_ops_v2.py:1277: stop (from ten
sorflow.python.eager.profiler) is deprecated and will be removed after 202
0-07-01.
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
  2/201 [..............................] - ETA: 18s - loss: 9.7503 - accur
acy: 0.8008 - auc: 0.4456WARNING:tensorflow:Callbacks method `on_train_bat
ch_end` is slow compared to the batch time (batch time: 0.0717s vs `on_tra
in_batch_end` time: 0.1112s). Check your callbacks.
200/201 [============================>.] - ETA: 0s - loss: 0.7702 - accura
cy: 0.8188 - auc: 0.4941
Epoch 00001: val_auc improved from -inf to 0.60067, saving model to /conte
nt/drive/My Drive/Colab Notebooks/LSTM/weights.hdf5
201/201 [==============================] - 11s 53ms/step - loss: 0.7699 -
accuracy: 0.8188 - auc: 0.4954 - val_loss: 0.4747 - val_accuracy: 0.8520 -
val_auc: 0.6007
Epoch 2/10
201/201 [==============================] - ETA: 0s - loss: 0.4894 - accura
cy: 0.8459 - auc: 0.5524
Epoch 00002: val_auc improved from 0.60067 to 0.68234, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights.hdf5
201/201 [==============================] - 8s 41ms/step - loss: 0.4894 - a
ccuracy: 0.8459 - auc: 0.5524 - val_loss: 0.4441 - val_accuracy: 0.8520 -
val_auc: 0.6823
Epoch 3/10
201/201 [==============================] - ETA: 0s - loss: 0.4517 - accura
cy: 0.8467 - auc: 0.6183
Epoch 00003: val_auc improved from 0.68234 to 0.70940, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights.hdf5
201/201 [==============================] - 8s 41ms/step - loss: 0.4517 - a
ccuracy: 0.8467 - auc: 0.6183 - val_loss: 0.4070 - val_accuracy: 0.8520 -
val_auc: 0.7094
Epoch 00003: early stopping
```

Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7f2e657914a8>
```

In [ ]:

```
#launch the tensor board
%load_ext tensorboard

%tensorboard --logdir /content/lstm_logs/1602569828.9323735
```

In [ ]:

```
#### Model1
y_train_pred = model.predict([padded_essay,encoded_state,encoded_proj_grade,encoded_cat
,encoded_sub_cat,
           encoded_prefix,train['remaining_input']])
print("Train AUC:",roc_auc_score(y_train,y_train_pred))

y_test_pred = model.predict([padded_essay_test,encoded_state_test,encoded_proj_grade_te
st,encoded_cat_test,encoded_sub_cat_test,
           encoded_prefix_test,test['remaining_input']])
print("Test AUC:",roc_auc_score(y_test,y_test_pred))
```

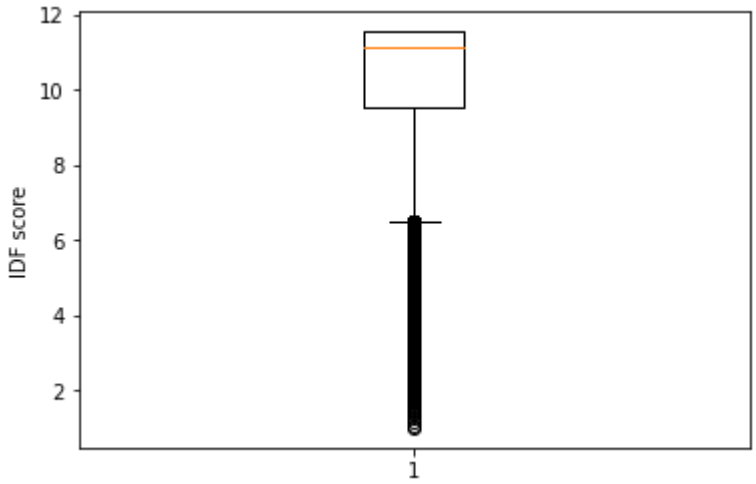
```
Train AUC: 0.7134284627289635
Test AUC: 0.7189304187678016
```

Model:2

In [6]:

```
#TFIDF Vectorization of essay feature
tfidf = TfidfVectorizer()
data_text = tfidf.fit_transform(train['essay'])
plt.boxplot(tfidf.idf_)
plt.ylabel("IDF score")
print("The 25 percentile of idf score is :", np.percentile(tfidf.idf_,[25]))
print("The 75 percentile of idf score is :",np.percentile(tfidf.idf_,[75]))
for i in range (0,101,10):
   p = np.percentile(tfidf.idf_, i)
   print(str(i)+" Percentile: "+ str(p))

#Select TFIDF values
feature_idf = zip(tfidf.get_feature_names(),tfidf.idf_)

feature_name = []
for x,y in feature_idf:

    if y >7 and y < 11.6 :
        feature_name.append(x)
    else:
        pass
```

```
The 25 percentile of idf score is : [9.53665589]
The 75 percentile of idf score is : [11.55155891]
0 Percentile: 1.0078108475060328
10 Percentile: 7.517318274052587
20 Percentile: 8.986609554743445
30 Percentile: 9.942120999770882
40 Percentile: 10.635268180330828
50 Percentile: 11.146093804096818
60 Percentile: 11.146093804096818
70 Percentile: 11.551558912204982
80 Percentile: 11.551558912204982
90 Percentile: 11.551558912204982
100 Percentile: 11.551558912204982
```



In [ ]:

```
#Selecting only those words which have idf values between 25th percentile to 75th perce
ntile

def few_text(df):
    processed_text = []
    for i,text in enumerate(tqdm(df)):
        sent = " "
        words = text.split()
        for word in words:
            if word in feature_name:
                sent = ' ' + word
            else:
                pass
        processed_text.append(sent)
    return processed_text

x_train_essay_text_filtered = few_text(train['essay'])
```

In [ ]:

```
with open('/content/drive/My Drive/Colab Notebooks/LSTM/train-essay.pkl', 'wb') as fp:
  pickle.dump(x_train_essay_text_filtered, fp)

x_test_essay_text_filtered1= few_text(test['essay'])

with open('/content/drive/My Drive/Colab Notebooks/LSTM/test1-essay.pkl', 'wb') as fp1:
  pickle.dump(x_test_essay_text_filtered1, fp1)
```

In [7]:

```
with open('/content/drive/My Drive/Colab Notebooks/LSTM/train-essay.pkl', 'rb') as fp:
  x_train_essay_text_filtered=pickle.load(fp)

with open('/content/drive/My Drive/Colab Notebooks/LSTM/test1-essay.pkl', 'rb') as fp1:
  x_test_essay_text_filtered1=pickle.load(fp1)
```

In [68]:

```
#Tokenizing the Essay column
# prepare tokenizer
t = Tokenizer()
t.fit_on_texts(x_train_essay_text_filtered)
vocab_size_essay = len(t.word_index) + 1
word_index=t.word_index
# integer encode the documents
encoded_essay = t.texts_to_sequences(x_train_essay_text_filtered)

# pad documents to a max length of 250 words
max_length = 250
max_length_essay = len(max(x_train_essay_text_filtered, key=len))
padded_essay = pad_sequences(encoded_essay, maxlen=max_length, padding='post')

# integer encode the documents
#One hot encoding and padding for input to embedding layer
vocab_size = 52
encoded_state = [one_hot(d, vocab_size) for d in train['school_state']]
max_length = 52
encoded_state = pad_sequences(encoded_state, maxlen=max_length, padding='post')

vocab_size = 5
encoded_proj_grade = [one_hot(d, vocab_size) for d in train['project_grade_category']]
max_length = 5
encoded_proj_grade = pad_sequences(encoded_proj_grade, maxlen=max_length, padding='pos
t')


vocab_size = 50
encoded_cat = [one_hot(d, vocab_size) for d in train['clean_categories']]
max_length = 50
encoded_cat = pad_sequences(encoded_cat, maxlen=max_length, padding='post')


vocab_size = 385
encoded_sub_cat = [one_hot(d, vocab_size) for d in train['clean_subcategories']]
max_length = 385
encoded_sub_cat = pad_sequences(encoded_sub_cat, maxlen=max_length, padding='post')

vocab_size = 6
encoded_prefix = [one_hot(d, vocab_size) for d in train['teacher_prefix']]
max_length = 6
encoded_prefix = pad_sequences(encoded_prefix, maxlen=max_length, padding='post')
```

In [69]:

```
#Embedding function to be used for embedding text features by using pre defined glove v
ectors
#Creating a matrix with rows as words and columns with 100 dim vectors for each word
def embedding_mat(word_index,embedding_dim = 100):
    embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
    #embedding_index=db
    for word,i in word_index.items():
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector
    return embedding_matrix

embedding_matrix=embedding_mat(word_index)
```

In [70]:

```
#Encoding test data
#Tokenizing the Essay column
# prepare tokenizer

# integer encode the documents
encoded_essay_test = t.texts_to_sequences(x_test_essay_text_filtered1)

# pad documents to a max length of 250 words
max_length = 250
padded_essay_test = pad_sequences(encoded_essay_test, maxlen=max_length, padding='post'
)
#print(padded_docs)

# integer encode the documents
#One hot encoding and padding for input to embedding layer
vocab_size = 52
encoded_state_test = [one_hot(d, vocab_size) for d in test['school_state']]
max_length = 52
encoded_state_test = pad_sequences(encoded_state_test, maxlen=max_length, padding='pos
t')

vocab_size = 5
encoded_proj_grade_test = [one_hot(d, vocab_size) for d in test['project_grade_categor
y']]
max_length = 5
encoded_proj_grade_test = pad_sequences(encoded_proj_grade_test, maxlen=max_length, pad
ding='post')


vocab_size = 50
encoded_cat_test = [one_hot(d, vocab_size) for d in test['clean_categories']]
max_length = 50
encoded_cat_test = pad_sequences(encoded_cat_test, maxlen=max_length, padding='post')


vocab_size = 385
encoded_sub_cat_test = [one_hot(d, vocab_size) for d in test['clean_subcategories']]
max_length = 385
encoded_sub_cat_test = pad_sequences(encoded_sub_cat_test, maxlen=max_length, padding=
'post')

vocab_size = 6
encoded_prefix_test = [one_hot(d, vocab_size) for d in test['teacher_prefix']]
max_length = 6
encoded_prefix_test = pad_sequences(encoded_prefix_test, maxlen=max_length, padding='po
st')
```

In [84]:

```
#input 1
input1 = Input(shape=(250,))
x1 = Embedding(input_length=padded_essay.shape[1],output_dim= 100,weights=[embedding_ma
trix],input_dim=vocab_size_essay,trainable=False)(input1)
x1 = LSTM(32,return_sequences=True)(x1)
x1 = layers.Dropout(0.5)(x1)
x1 = Flatten()(x1)

cat_vars=['school_state','project_grade_category','clean_categories','clean_subcategori
es','teacher_prefix']
cat_sizes={}
cat_embsizes={}
for cat in cat_vars:
  cat_sizes[cat]=train[cat].nunique()
  cat_embsizes[cat]=min(50,cat_sizes[cat]//2 +1)

#input 2
input2 = Input(shape=(52,))
x2 = Embedding(input_dim= cat_sizes['school_state']+1,output_dim= cat_embsizes['school_
state'])(input2)
x2 = Flatten()(x2)

#input 3
input3 = Input(shape=(5,))
x3 = Embedding(input_dim= cat_sizes['project_grade_category']+1,output_dim= cat_embsize
s['project_grade_category'])(input3)
x3 = Flatten()(x3)

#input 4
input4 = Input(shape=(50,))
x4 = Embedding(input_dim=cat_sizes['clean_categories']+1,output_dim= cat_embsizes['clea
n_categories'])(input4)
x4 = Flatten()(x4)

#input 5
input5 = Input(shape=(385,))
x5 = Embedding(input_dim= cat_sizes['clean_subcategories']+1,output_dim= cat_embsizes[
'clean_subcategories'])(input5)
x5 = Flatten()(x5)

#input 6
input6 = Input(shape=(6,))
x6 = Embedding(input_dim= cat_sizes['teacher_prefix']+1,output_dim= cat_embsizes['teach
er_prefix'])(input6)
x6 = Flatten()(x6)

#input 7
input7 = Input(shape=(1,))
x7 = Dense(4,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0
001))(input7)
x7 = layers.Dropout(0.5)(x7)
#merging all the inputs
concat = concatenate([x1,x2,x3,x4,x5,x6,x7])

x = Dense(32,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0
001))(concat)
x = layers.Dropout(0.2)(x)
x = Dense(32,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0
001))(x)
```

```
x = layers.Dropout(0.2)(x)
x = Dense(16,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0
001))(x)
x = layers.Dropout(0.2)(x)
x = Dense(8,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.00
01))(x)
x = layers.Dropout(0.2)(x)
output = Dense(2, activation = 'softmax')(x)

# create model with seven inputs
model = Model([input1,input2,input3,input4,input5,input6,input7], output)
tensorboard = keras.callbacks.TensorBoard(log_dir='lstm_logs/{}'.format(time()))
model.compile(optimizer=keras.optimizers.Adam(lr=0.0006,decay = 1e-4),loss='categorical
_crossentropy',metrics=[auc])

print(model.summary())
```

```
Model: "functional_51"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_176 (InputLayer)          [(None, 250)]        0
__________________________________________________________________________________________________
embedding_150 (Embedding)       (None, 250, 100)     4897900     input_176[0][0]
__________________________________________________________________________________________________
lstm_25 (LSTM)                  (None, 250, 32)      17024       embedding_150[0][0]
__________________________________________________________________________________________________
input_177 (InputLayer)          [(None, 52)]         0
__________________________________________________________________________________________________
input_178 (InputLayer)          [(None, 5)]          0
__________________________________________________________________________________________________
input_179 (InputLayer)          [(None, 50)]         0
__________________________________________________________________________________________________
input_180 (InputLayer)          [(None, 385)]        0
__________________________________________________________________________________________________
input_181 (InputLayer)          [(None, 6)]          0
__________________________________________________________________________________________________
input_182 (InputLayer)          [(None, 1)]          0
__________________________________________________________________________________________________
dropout_135 (Dropout)           (None, 250, 32)      0           lstm_25[0][0]
__________________________________________________________________________________________________
embedding_151 (Embedding)       (None, 52, 26)       1352        input_177[0][0]
__________________________________________________________________________________________________
embedding_152 (Embedding)       (None, 5, 3)         15          input_178[0][0]
__________________________________________________________________________________________________
embedding_153 (Embedding)       (None, 50, 26)       1352        input_179[0][0]
__________________________________________________________________________________________________
embedding_154 (Embedding)       (None, 385, 50)      19550       input_180[0][0]
__________________________________________________________________________________________________
embedding_155 (Embedding)       (None, 6, 3)         18          input_181[0][0]
__________________________________________________________________________________________________
dense_135 (Dense)               (None, 4)            8           input_182[0][0]
```

```
82[0][0]
__________________________________________________________________________________________________
__________________________
flatten_150 (Flatten)           (None, 8000)         0           dropout
_135[0][0]
__________________________________________________________________________________________________
__________________________
flatten_151 (Flatten)           (None, 1352)         0           embeddi
ng_151[0][0]
__________________________________________________________________________________________________
__________________________
flatten_152 (Flatten)           (None, 15)           0           embeddi
ng_152[0][0]
__________________________________________________________________________________________________
__________________________
flatten_153 (Flatten)           (None, 1300)         0           embeddi
ng_153[0][0]
__________________________________________________________________________________________________
__________________________
flatten_154 (Flatten)           (None, 19250)        0           embeddi
ng_154[0][0]
__________________________________________________________________________________________________
__________________________
flatten_155 (Flatten)           (None, 18)           0           embeddi
ng_155[0][0]
__________________________________________________________________________________________________
__________________________
dropout_136 (Dropout)           (None, 4)            0           dense_1
35[0][0]
__________________________________________________________________________________________________
__________________________
concatenate_25 (Concatenate)    (None, 29939)        0           flatten
_150[0][0]
                                                                 flatten
_151[0][0]
                                                                 flatten
_152[0][0]
                                                                 flatten
_153[0][0]
                                                                 flatten
_154[0][0]
                                                                 flatten
_155[0][0]
                                                                 dropout
_136[0][0]
__________________________________________________________________________________________________
__________________________
dense_136 (Dense)               (None, 32)           958080      concate
nate_25[0][0]
__________________________________________________________________________________________________
__________________________
dropout_137 (Dropout)           (None, 32)           0           dense_1
36[0][0]
__________________________________________________________________________________________________
__________________________
dense_137 (Dense)               (None, 32)           1056        dropout
_137[0][0]
__________________________________________________________________________________________________
__________________________
dropout_138 (Dropout)           (None, 32)           0           dense_1
37[0][0]
```

```
__________________________________________________________________________________________________
dense_138 (Dense)               (None, 16)           528         dropout_138[0][0]
__________________________________________________________________________________________________
dropout_139 (Dropout)           (None, 16)           0           dense_138[0][0]
__________________________________________________________________________________________________
dense_139 (Dense)               (None, 8)            136         dropout_139[0][0]
__________________________________________________________________________________________________
dropout_140 (Dropout)           (None, 8)            0           dense_139[0][0]
__________________________________________________________________________________________________
dense_140 (Dense)               (None, 2)            18          dropout_140[0][0]
==================================================================================================
Total params: 5,897,037
Trainable params: 999,137
Non-trainable params: 4,897,900
__________________________________________________________________________________________________
None
```

In [86]:

```
#Ref:#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
filepath="/content/drive/My Drive/Colab Notebooks/LSTM/weights1.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_auc', verbose=1, save_best_only=Tru
e, mode='max')
#earlystop = EarlyStopping(monitor='val_auc', patience=2, verbose=1)
callbacks_list = [checkpoint,tensorboard]

model.fit([padded_essay,encoded_state,encoded_proj_grade,encoded_cat,encoded_sub_cat,
           encoded_prefix,train['remaining_input']], y_train, epochs=10,verbose=1,batch
_size=256,
           validation_data=([padded_essay_test,encoded_state_test,encoded_proj_grade_te
st,encoded_cat_test,encoded_sub_cat_test,
           encoded_prefix_test,test['remaining_input']], y_test),callbacks = callbacks_
list)
```

```
Epoch 1/10
  2/299 [..............................] - ETA: 26s - loss: 0.4092 - auc:
0.6909WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow com
pared to the batch time (batch time: 0.0531s vs `on_train_batch_end` time:
0.1228s). Check your callbacks.
298/299 [============================>.] - ETA: 0s - loss: 0.4141 - auc:
0.6849
Epoch 00001: val_auc improved from -inf to 0.71905, saving model to /conte
nt/drive/My Drive/Colab Notebooks/LSTM/weights1.hdf5
299/299 [==============================] - 13s 43ms/step - loss: 0.4142 -
auc: 0.6850 - val_loss: 0.4026 - val_auc: 0.7191
Epoch 2/10
299/299 [==============================] - ETA: 0s - loss: 0.4077 - auc:
0.6979
Epoch 00002: val_auc improved from 0.71905 to 0.72642, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights1.hdf5
299/299 [==============================] - 13s 43ms/step - loss: 0.4077 -
auc: 0.6979 - val_loss: 0.3973 - val_auc: 0.7264
Epoch 3/10
299/299 [==============================] - ETA: 0s - loss: 0.4049 - auc:
0.7057
Epoch 00003: val_auc improved from 0.72642 to 0.73361, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights1.hdf5
299/299 [==============================] - 13s 44ms/step - loss: 0.4049 -
auc: 0.7057 - val_loss: 0.3975 - val_auc: 0.7336
Epoch 4/10
299/299 [==============================] - ETA: 0s - loss: 0.3999 - auc:
0.7158
Epoch 00004: val_auc did not improve from 0.73361
299/299 [==============================] - 12s 41ms/step - loss: 0.3999 -
auc: 0.7158 - val_loss: 0.4046 - val_auc: 0.7248
Epoch 5/10
299/299 [==============================] - ETA: 0s - loss: 0.3990 - auc:
0.7178
Epoch 00005: val_auc improved from 0.73361 to 0.73459, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights1.hdf5
299/299 [==============================] - 13s 42ms/step - loss: 0.3990 -
auc: 0.7178 - val_loss: 0.3904 - val_auc: 0.7346
Epoch 6/10
299/299 [==============================] - ETA: 0s - loss: 0.3949 - auc:
0.7274
Epoch 00006: val_auc did not improve from 0.73459
299/299 [==============================] - 12s 41ms/step - loss: 0.3949 -
auc: 0.7274 - val_loss: 0.4023 - val_auc: 0.7311
Epoch 7/10
299/299 [==============================] - ETA: 0s - loss: 0.3921 - auc:
0.7322
Epoch 00007: val_auc improved from 0.73459 to 0.73710, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights1.hdf5
299/299 [==============================] - 13s 43ms/step - loss: 0.3921 -
auc: 0.7322 - val_loss: 0.3890 - val_auc: 0.7371
Epoch 8/10
299/299 [==============================] - ETA: 0s - loss: 0.3902 - auc:
0.7372
Epoch 00008: val_auc did not improve from 0.73710
299/299 [==============================] - 12s 41ms/step - loss: 0.3902 -
auc: 0.7372 - val_loss: 0.3988 - val_auc: 0.7301
Epoch 9/10
298/299 [============================>.] - ETA: 0s - loss: 0.3865 - auc:
0.7431
Epoch 00009: val_auc did not improve from 0.73710
```

```
299/299 [==============================] - 12s 40ms/step - loss: 0.3866 -
auc: 0.7429 - val_loss: 0.3948 - val_auc: 0.7308
Epoch 10/10
299/299 [==============================] - ETA: 0s - loss: 0.3843 - auc:
0.7491
Epoch 00010: val_auc did not improve from 0.73710
299/299 [==============================] - 13s 42ms/step - loss: 0.3843 -
auc: 0.7491 - val_loss: 0.3906 - val_auc: 0.7296
```

Out[86]:

```
<tensorflow.python.keras.callbacks.History at 0x7f1ada500518>
```

In [87]:

```
#launch the tensor board
%load_ext tensorboard

%tensorboard --logdir /content/lstm_logs/1602928483.680405
```

In [88]:

```
#### Model2
y_train_pred = model.predict([padded_essay,encoded_state,encoded_proj_grade,encoded_cat
,encoded_sub_cat,
           encoded_prefix,train['remaining_input']])
print("Train AUC:",roc_auc_score(y_train,y_train_pred))

y_test_pred = model.predict([padded_essay_test,encoded_state_test,encoded_proj_grade_te
st,encoded_cat_test,encoded_sub_cat_test,
           encoded_prefix_test,test['remaining_input']])

print("Test AUC:",roc_auc_score(y_test,y_test_pred))
```

```
Train AUC: 0.7820996223160505
Test AUC: 0.7274645406353629
```

Model 3

In [ ]:

```
#Reading the dataset
project_data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/LSTM/preprocessed_data.csv')
project_data.shape

#Formatting the dataframe removing columns and including the labels
class_label = project_data['project_is_approved']
project_data['remaining_input'] = project_data['teacher_number_of_previously_posted_projects']  +\
                                    project_data['price']

col = ['project_is_approved','teacher_number_of_previously_posted_projects','price']
project_data.drop(labels=col,axis =1, inplace=True)

#printing the columns in the dataframe
col = project_data.columns
print(col)

#performing train test split
#Stratify parameter
#For example, if variable y is a binary categorical variable with values 0 and 1 and there are 25% of zeros and 75% of ones, stratify=y will make sure that your random split has 25% of 0's and 75% of 1's.
train,test,y_train,y_test = train_test_split(project_data, class_label , stratify = class_label, train_size = 0.7)

print("Shape of the Train dataset: ", train.shape[0])
print("Shape of the Test dataset: ", test.shape[0])

#converting class labels to categorical variables
#We are using categorigal cross entrophy loss function and softmax classifier,therefore we are encoding the labels
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'clean_categories', 'clean_subcategories', 'essay', 'remaining_input'],
      dtype='object')
Shape of the Train dataset:  76473
Shape of the Test dataset:  32775
```

In [ ]:

```
#Tokenizing the Essay column
# prepare tokenizer
t = Tokenizer()
t.fit_on_texts(train['essay'])
vocab_size_essay = len(t.word_index) + 1
word_index=t.word_index
# integer encode the documents
encoded_essay = t.texts_to_sequences(train['essay'])

# pad documents to a max length of 250 words
max_length = 250
padded_essay = pad_sequences(encoded_essay, maxlen=max_length, padding='post')

# integer encode the documents
#One hot encoding and padding for input to embedding layer
vocab_size = 52
encoded_state = [one_hot(d, vocab_size) for d in train['school_state']]
max_length = 52
encoded_state = pad_sequences(encoded_state, maxlen=max_length, padding='post')

vocab_size = 5
encoded_proj_grade = [one_hot(d, vocab_size) for d in train['project_grade_category']]
max_length = 5
encoded_proj_grade = pad_sequences(encoded_proj_grade, maxlen=max_length, padding='pos
t')


vocab_size = 50
encoded_cat = [one_hot(d, vocab_size) for d in train['clean_categories']]
max_length = 50
encoded_cat = pad_sequences(encoded_cat, maxlen=max_length, padding='post')


vocab_size = 385
encoded_sub_cat = [one_hot(d, vocab_size) for d in train['clean_subcategories']]
max_length = 385
encoded_sub_cat = pad_sequences(encoded_sub_cat, maxlen=max_length, padding='post')

vocab_size = 6
encoded_prefix = [one_hot(d, vocab_size) for d in train['teacher_prefix']]
max_length = 6
encoded_prefix = pad_sequences(encoded_prefix, maxlen=max_length, padding='post')
```

In [ ]:

```
#USER DEFINED FUNCTION 1 FOR GLOVE EMBEDDING OF TEXT FEATURES
#Embedding function to be used for embedding text features by using pre defined glove v
ectors
#Creating a matrix with rows as words and columns with 50 dim vectors for each word
def embedding_mat(word_index,embedding_dim = 100):
    embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
    #embedding_index=db
    for word,i in word_index.items():
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector
    return embedding_matrix

embedding_matrix=embedding_mat(word_index)
```

In [ ]:

```
#Encoding test data
#Tokenizing the Essay column
# prepare tokenizer
#t = Tokenizer()
#t.fit_on_texts(test['essay'])
#vocab_size_essay = len(t.word_index) + 1
#word_index_test=t.word_index

# integer encode the documents
encoded_essay_test = t.texts_to_sequences(test['essay'])

# pad documents to a max length of 250 words
max_length = 250
padded_essay_test = pad_sequences(encoded_essay_test, maxlen=max_length, padding='post'
)
#print(padded_docs)

# integer encode the documents
#One hot encoding and padding for input to embedding layer
vocab_size = 52
encoded_state_test = [one_hot(d, vocab_size) for d in test['school_state']]
max_length = 52
encoded_state_test = pad_sequences(encoded_state_test, maxlen=max_length, padding='pos
t')

vocab_size = 5
encoded_proj_grade_test = [one_hot(d, vocab_size) for d in test['project_grade_categor
y']]
max_length = 5
encoded_proj_grade_test = pad_sequences(encoded_proj_grade_test, maxlen=max_length, pad
ding='post')


vocab_size = 50
encoded_cat_test = [one_hot(d, vocab_size) for d in test['clean_categories']]
max_length = 50
encoded_cat_test = pad_sequences(encoded_cat_test, maxlen=max_length, padding='post')


vocab_size = 385
encoded_sub_cat_test = [one_hot(d, vocab_size) for d in test['clean_subcategories']]
max_length = 385
encoded_sub_cat_test = pad_sequences(encoded_sub_cat_test, maxlen=max_length, padding=
'post')

vocab_size = 6
encoded_prefix_test = [one_hot(d, vocab_size) for d in test['teacher_prefix']]
max_length = 6
encoded_prefix_test = pad_sequences(encoded_prefix_test, maxlen=max_length, padding='po
st')

#embedding_matrix_test=embedding_mat(word_index_test)
```

In [ ]:

```
from scipy.sparse import hstack

input2_train = np.hstack((encoded_state,encoded_proj_grade,encoded_cat,encoded_sub_cat,
encoded_prefix,train['remaining_input'][:,None]))

input2_test = np.hstack((encoded_state_test,encoded_proj_grade_test,encoded_cat_test,en
coded_sub_cat_test,encoded_prefix_test,test['remaining_input'][:,None]))
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: FutureWarn
ing: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is depre
cated and will be removed in a future version.  Convert to a numpy array b
efore indexing instead.
  This is separate from the ipykernel package so we can avoid doing import
s until
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: FutureWarn
ing: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is depre
cated and will be removed in a future version.  Convert to a numpy array b
efore indexing instead.
  """
```

In [ ]:

```
print(input2_train.shape)
print(input2_test.shape)
```

```
(76473, 499)
(32775, 499)
```

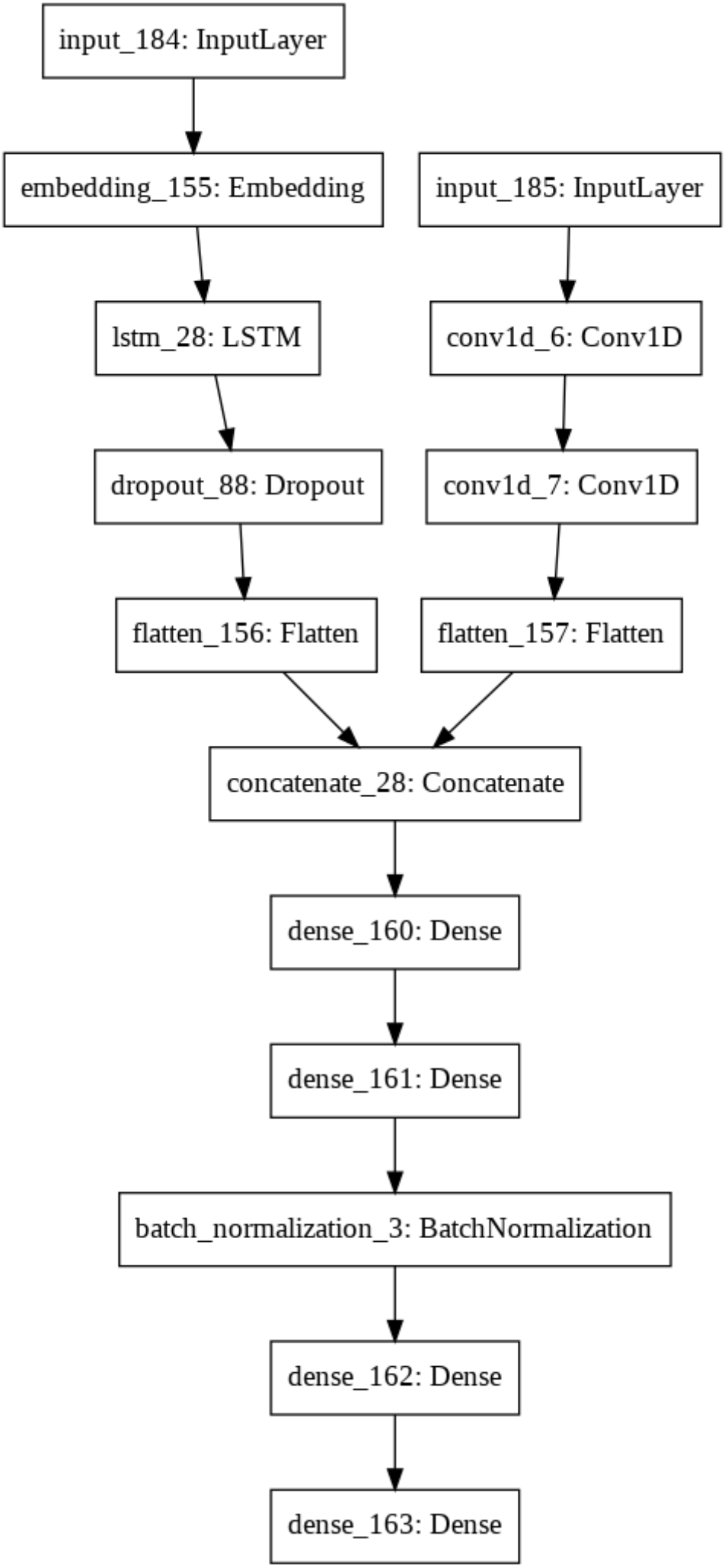
In [ ]:

```
# input 1

input1 = Input(shape=(250,))
x1 = Embedding(input_dim=len(word_index) + 1,output_dim= 100,weights=[embedding_matrix
],trainable=False)(input1)
x1 = LSTM(32,return_sequences=True)(x1)
x1 = layers.Dropout(0.5)(x1)
x1 = Flatten()(x1)


# input 2
input2 = Input(shape=(499,1))
x2 = Conv1D(filters=64,kernel_size=3,strides=1)(input2)
x2 = Conv1D(filters=64,kernel_size=3,strides=1)(x2)
x2 = Flatten()(x2)

# merging both the inputs
concat = concatenate([x1,x2])
x = Dense(256,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.
0001))(concat)
x = Dense(128,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.
0001))(x)
x = BatchNormalization()(x)
x = Dense(64,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.0
001))(x)
output = Dense(2, activation = 'softmax')(x)

# create model with two inputs
model = Model([input1,input2], output)
tensorboard = TensorBoard(log_dir='logs/fit/model3')
model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.Adam(), metri
cs=[auc])
print(model.summary())
# Plot model3 graph
tf.keras.utils.plot_model(model, to_file='Model3.png')
from IPython.display import Image
Image(filename='Model3.png')
```

```
Model: "functional_57"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_184 (InputLayer)          [(None, 250)]        0
__________________________________________________________________________________________________
embedding_155 (Embedding)       (None, 250, 100)     4920700     input_184[0][0]
__________________________________________________________________________________________________
input_185 (InputLayer)          [(None, 499, 1)]     0
__________________________________________________________________________________________________
lstm_28 (LSTM)                  (None, 250, 32)      17024       embedding_155[0][0]
__________________________________________________________________________________________________
conv1d_6 (Conv1D)               (None, 497, 64)      256         input_185[0][0]
__________________________________________________________________________________________________
dropout_88 (Dropout)            (None, 250, 32)      0           lstm_28[0][0]
__________________________________________________________________________________________________
conv1d_7 (Conv1D)               (None, 495, 64)      12352       conv1d_6[0][0]
__________________________________________________________________________________________________
flatten_156 (Flatten)           (None, 8000)         0           dropout_88[0][0]
__________________________________________________________________________________________________
flatten_157 (Flatten)           (None, 31680)        0           conv1d_7[0][0]
__________________________________________________________________________________________________
concatenate_28 (Concatenate)    (None, 39680)        0           flatten_156[0][0]
                                                                 flatten_157[0][0]
__________________________________________________________________________________________________
dense_160 (Dense)               (None, 256)          10158336    concatenate_28[0][0]
__________________________________________________________________________________________________
dense_161 (Dense)               (None, 128)          32896       dense_160[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 128)          512         dense_161[0][0]
__________________________________________________________________________________________________
dense_162 (Dense)               (None, 64)           8256        batch_normalization_3[0][0]
```

```
_________________________________________________________________
_________________________
dense_163 (Dense)            (None, 2)                 130       dense_1
62[0][0]
=================================================================
=========================
Total params: 15,150,462
Trainable params: 10,229,506
Non-trainable params: 4,920,956
_________________________________________________________________
_________________________
None
```

Out[ ]:


input_184: InputLayer
embedding_155: Embedding
input_185: InputLayer
lstm_28: LSTM
conv1d_6: Conv1D
dropout_88: Dropout
conv1d_7: Conv1D
flatten_156: Flatten
flatten_157: Flatten
concatenate_28: Concatenate
dense_160: Dense
dense_161: Dense
batch_normalization_3: BatchNormalization
dense_162: Dense
dense_163: Dense

In [ ]:

```
#Ref: #https://machinelearningmastery.com/check-point-deep-learning-models-keras/
filepath="/content/drive/My Drive/Colab Notebooks/LSTM/weights3.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_auc', verbose=1, save_best_only=Tru
e, mode='max')
#earlystop = EarlyStopping(monitor='val_auc', patience=2, verbose=1)
callbacks_list = [checkpoint,tensorboard]

model.fit([padded_essay,input2_train], y_train, epochs=15,verbose=1,batch_size=256,vali
dation_split=0.33,callbacks = callbacks_list)
```

```
Epoch 1/15
  2/201 [..............................] - ETA: 22s - loss: 1.5872 - auc:
0.5032WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow com
pared to the batch time (batch time: 0.0374s vs `on_train_batch_end` time:
0.1908s). Check your callbacks.
200/201 [============================>.] - ETA: 0s - loss: 0.4889 - auc:
0.6169
Epoch 00001: val_auc improved from -inf to 0.66924, saving model to /conte
nt/drive/My Drive/Colab Notebooks/LSTM/weights3.hdf5
201/201 [==============================] - 10s 49ms/step - loss: 0.4888 -
auc: 0.6177 - val_loss: 0.4490 - val_auc: 0.6692
Epoch 2/15
199/201 [============================>.] - ETA: 0s - loss: 0.4413 - auc:
0.6892
Epoch 00002: val_auc improved from 0.66924 to 0.69571, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights3.hdf5
201/201 [==============================] - 8s 41ms/step - loss: 0.4413 - a
uc: 0.6901 - val_loss: 0.4254 - val_auc: 0.6957
Epoch 3/15
199/201 [============================>.] - ETA: 0s - loss: 0.4262 - auc:
0.7126
Epoch 00003: val_auc improved from 0.69571 to 0.70145, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights3.hdf5
201/201 [==============================] - 8s 42ms/step - loss: 0.4267 - a
uc: 0.7124 - val_loss: 0.4226 - val_auc: 0.7014
Epoch 4/15
199/201 [============================>.] - ETA: 0s - loss: 0.4193 - auc:
0.7229
Epoch 00004: val_auc improved from 0.70145 to 0.71652, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights3.hdf5
201/201 [==============================] - 8s 41ms/step - loss: 0.4190 - a
uc: 0.7224 - val_loss: 0.4199 - val_auc: 0.7165
Epoch 5/15
199/201 [============================>.] - ETA: 0s - loss: 0.4125 - auc:
0.7336
Epoch 00005: val_auc did not improve from 0.71652
201/201 [==============================] - 8s 39ms/step - loss: 0.4126 - a
uc: 0.7339 - val_loss: 0.4217 - val_auc: 0.7052
Epoch 6/15
199/201 [============================>.] - ETA: 0s - loss: 0.4068 - auc:
0.7386
Epoch 00006: val_auc improved from 0.71652 to 0.72544, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights3.hdf5
201/201 [==============================] - 9s 47ms/step - loss: 0.4069 - a
uc: 0.7374 - val_loss: 0.4145 - val_auc: 0.7254
Epoch 7/15
201/201 [==============================] - ETA: 0s - loss: 0.4027 - auc:
0.7453
Epoch 00007: val_auc improved from 0.72544 to 0.73050, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights3.hdf5
201/201 [==============================] - 8s 42ms/step - loss: 0.4027 - a
uc: 0.7453 - val_loss: 0.4034 - val_auc: 0.7305
Epoch 8/15
201/201 [==============================] - ETA: 0s - loss: 0.3985 - auc:
0.7521
Epoch 00008: val_auc did not improve from 0.73050
201/201 [==============================] - 8s 38ms/step - loss: 0.3985 - a
uc: 0.7521 - val_loss: 0.4026 - val_auc: 0.7295
Epoch 9/15
199/201 [============================>.] - ETA: 0s - loss: 0.3982 - auc:
0.7561
```

```
Epoch 00009: val_auc improved from 0.73050 to 0.73639, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights3.hdf5
201/201 [==============================] - 9s 46ms/step - loss: 0.3981 - a
uc: 0.7555 - val_loss: 0.3972 - val_auc: 0.7364
Epoch 10/15
199/201 [============================>.] - ETA: 0s - loss: 0.3909 - auc:
0.7614
Epoch 00010: val_auc improved from 0.73639 to 0.73715, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights3.hdf5
201/201 [==============================] - 8s 41ms/step - loss: 0.3914 - a
uc: 0.7604 - val_loss: 0.3984 - val_auc: 0.7371
Epoch 11/15
199/201 [============================>.] - ETA: 0s - loss: 0.3862 - auc:
0.7689
Epoch 00011: val_auc improved from 0.73715 to 0.73977, saving model to /co
ntent/drive/My Drive/Colab Notebooks/LSTM/weights3.hdf5
201/201 [==============================] - 8s 42ms/step - loss: 0.3861 - a
uc: 0.7693 - val_loss: 0.3930 - val_auc: 0.7398
Epoch 12/15
199/201 [============================>.] - ETA: 0s - loss: 0.3834 - auc:
0.7755
Epoch 00012: val_auc did not improve from 0.73977
201/201 [==============================] - 8s 38ms/step - loss: 0.3834 - a
uc: 0.7754 - val_loss: 0.4788 - val_auc: 0.7186
Epoch 13/15
199/201 [============================>.] - ETA: 0s - loss: 0.3795 - auc:
0.7799
Epoch 00013: val_auc did not improve from 0.73977
201/201 [==============================] - 8s 39ms/step - loss: 0.3793 - a
uc: 0.7800 - val_loss: 0.4011 - val_auc: 0.7324
Epoch 14/15
199/201 [============================>.] - ETA: 0s - loss: 0.3767 - auc:
0.7861
Epoch 00014: val_auc did not improve from 0.73977
201/201 [==============================] - 8s 38ms/step - loss: 0.3765 - a
uc: 0.7862 - val_loss: 0.4040 - val_auc: 0.7260
Epoch 15/15
199/201 [============================>.] - ETA: 0s - loss: 0.3715 - auc:
0.7907
Epoch 00015: val_auc did not improve from 0.73977
201/201 [==============================] - 8s 37ms/step - loss: 0.3714 - a
uc: 0.7908 - val_loss: 0.3982 - val_auc: 0.7366
```

Out[ ]:

```
<tensorflow.python.keras.callbacks.History at 0x7f2cfc371080>
```

In [ ]:

```
#launch the tensor board
%reload_ext tensorboard

%tensorboard --logdir /content/logs/fit/model3
```

```
Reusing TensorBoard on port 6006 (pid 6022), started 0:01:56 ago. (Use '!k
ill 6022' to kill it.)
```

In [ ]:

```
#### Model3
y_train_pred = model.predict([padded_essay,input2_train])
print("Train AUC:",roc_auc_score(y_train,y_train_pred))

y_test_pred = model.predict([padded_essay_test,input2_test])
print("Test AUC:",roc_auc_score(y_test,y_test_pred))
```

```
Train AUC: 0.781493608703407
Test AUC: 0.7436623962815994
```

Summary

Model-1

Train AUC: 0.7274645406353629

Test AUC :0.7189304187678016

Model Overfitting with in few epochs

Model-2

Train AUC: 0.7820996223160505

Test AUC :0.7274645406353629

With IFIDF vectorization of text features,overfitting is less with average AUC score

Model-3

Train AUC: 0.781493608703407

Test AUC :0.7436623962815994

Model performance is better than model 1 and model 2 with less overfitting and good auc score.