Assignment : 14

**Difffernt approches used

1. Try up-sampling on overfitting
2. Try giving more weightage for minority classes
3. Try hyper-parameter tunning using simple loops.**

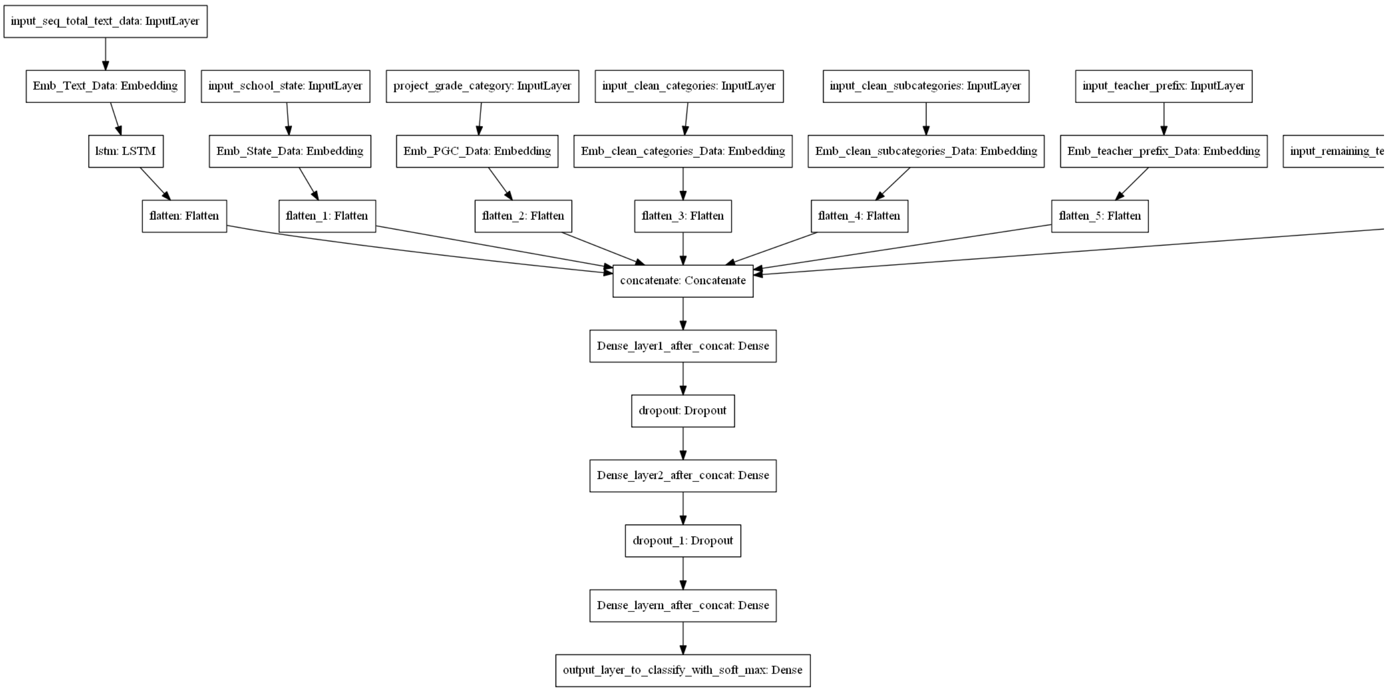
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

#from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
from keras.layers import Input, Embedding, LSTM, Dropout, BatchNormalization, Dense, concatenate,
from keras.preprocessing.text import Tokenizer, one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Model, load_model
from keras import regularizers
from keras.optimizers import *
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard, ReduceLROnPlateau
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics import roc_auc_score
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib inline
import re
from tqdm import tqdm
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import pickle
```

```
Using TensorFlow backend.
```

1. Download the preprocessed DonorsChoose data from here Dataset
2. Split the data into train, cv, and test
3. After step 2 you have to train 3 types of models as discussed below.
4. For all the model use 'auc' as a metric. check this for using auc as a metric. you epoch. Note: you should NOT use the tf.metric.auc
5. You are free to choose any number of layers/hiddden units but you have to use same
6. You can use any one of the optimizers and choice of Learning rate and momentum, res class video.
7. You should Save the best model weights.
8. For all the model's use TensorBoard and plot the Metric value and Loss with epoch. plots and include those images in .ipynb notebook and PDF.
9. Use Categorical Cross Entropy as Loss to minimize.
10. try to get AUC more than 0.8 for atleast one model

Model-1

Build and Train deep neural network as shown below


input_seq_total_text_data: InputLayer
Emb_Text_Data: Embedding
input_school_state: InputLayer
project_grade_category: InputLayer
input_clean_categories: InputLayer
input_clean_subcategories: InputLayer
input_teacher_prefix: InputLayer
lstm: LSTM
Emb_State_Data: Embedding
Emb_PGC_Data: Embedding
Emb_clean_categories_Data: Embedding
Emb_clean_subcategories_Data: Embedding
Emb_teacher_prefix_Data: Embedding
input_remaining_te
flatten: Flatten
flatten_1: Flatten
flatten_2: Flatten
flatten_3: Flatten
flatten_4: Flatten
flatten_5: 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/w395Yk9.png

- **Input_seq_total_text_data** --- You have to give Total text data columns. After this use the Embedding layer word vectors, don't train any word vectors. After this use LSTM and get the LSTM output and Flatten that
- **Input_school_state** --- Give 'school_state' column as input to embedding layer and Train the Keras Embedd
- **Project_grade_category** --- Give 'project_grade_category' column as input to embedding layer and Train th
- **Input_clean_categories** --- Give 'input_clean_categories' column as input to embedding layer and Train the
- **Input_clean_subcategories** --- Give 'input_clean_subcategories' column as input to embedding layer and T
- **Input_clean_subcategories** --- Give 'input_teacher_prefix' column as input to embedding layer and Train th

- **Input_remaining_teacher_number_of_previously_posted_projects._resource_summary_contains_numeri** columns and add a Dense layer after that.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473

Enter your authorization code:
··········
Mounted at /content/gdrive
```

```
data = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/preprocessed_data.csv')
data.head(5)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	
2	ca	mrs	grades_prek_2	
3	ga	mrs	grades_prek_2	
4	wa	mrs	grades_3_5	

```
X = data
y = data['project_is_approved'].values
X = X.drop(['project_is_approved'], axis=1)
```

```
X.columns
```

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_trai
```

```
print(X_train.shape,X_test.shape)
print(X_train.shape[0] + X_test.shape[0] )
```

```
(49041, 8) (36052, 8)
85093
```

```
unique = set(X_train['essay'].values)
print("NO OF UNIQUE WORDS IN TRAIN ESSAY",len(unique))
```

```
NO OF UNIQUE WORDS IN TRAIN ESSAY 48852
```

```
#Converts a text to a sequence of words (or tokens).
#A list of words (or tokens).
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train["essay"])
sequences_train = tokenizer.texts_to_sequences(X_train["essay"])
sequences_cv = tokenizer.texts_to_sequences(X_cv["essay"])
sequences_test = tokenizer.texts_to_sequences(X_test["essay"])
```

```
word2idx = tokenizer.word_index
print('Found %s unique tokens.' % len(word2idx))
```

```
Found 41409 unique tokens.
```

```
#WORD---->RANK
word2idx['zz']
```

```
40371
```

```
encoded_train = pad_sequences(sequences_train,maxlen=800,padding='post')
print('Shape of data tensor:', encoded_train.shape)
encoded_test = pad_sequences(sequences_test, maxlen=800,padding='post')
encoded_cv = pad_sequences(sequences_cv, maxlen=800, padding='post')
print('Shape of data tensor:', encoded_test.shape)
```

```
Shape of data tensor: (49041, 800)
Shape of data tensor: (36052, 800)
```

```
pickle_in = open("/content/gdrive/My Drive/Colab Notebooks/glove_vectors","rb")
```

```
glove_words = pickle.load(pickle_in)
#glove_words
```

```
num_words = len(word2idx) + 1
embedding_matrix = np.zeros((num_words, 300))
for word, i in word2idx.items():
    embedding_vector = glove_words.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector


print(num_words)
print(embedding_matrix.shape)
```

```
41410
(41410, 300)
```

```
input_text = Input(shape=(800,),name="input_text")
X = Embedding(num_words,300,weights=[embedding_matrix],input_length=800,trainable=False)(input_te
X = LSTM(128,recurrent_dropout=0.5,kernel_regularizer=regularizers.l2(0.001),return_sequences=Tru
flatten_1 = Flatten()(X)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob
```

```
# Teacher Prefix
#no_of_unique_prefix  = set(X_train['teacher_prefix'].values)
no_of_unique_prefix  = X_train["teacher_prefix"].nunique()
print('Unique Categories:', (no_of_unique_prefix))


# Defining Input and Embedding Layer for the same

input_prefix = Input(shape=(1,),name="teacher_prefix")
embedding_prefix = Embedding(no_of_unique_prefix,3,name="emb_pre",trainable=True)(input_prefix)
flatten_2 = Flatten()(embedding_prefix)

lb = LabelEncoder()
encoder_prefix_train = lb.fit_transform(X_train["teacher_prefix"])
encoder_prefix_cv = lb.fit_transform(X_cv["teacher_prefix"])
encoder_prefix_test = lb.transform(X_test["teacher_prefix"])
```

```
Unique Categories: 5
```

```
# School State
```

```
no_of_unique_state  = X_train["school_state"].nunique()
embedding_size_state= int(np.ceil((no_of_unique_state)/2))
print('Unique Categories:', no_of_unique_state,'Embedding Size:', embedding_size_state)


# Defining Input and Embedding Layer for the same

input_state = Input(shape=(1,),name="school_prefix")
embedding_state = Embedding(no_of_unique_state,embedding_size_state,name="emb_state",trainable=Tr
flatten_3 = Flatten()(embedding_state)


encoder_state_train = lb.fit_transform(X_train["school_state"])
encoder_state_cv = lb.fit_transform(X_cv["school_state"])
encoder_state_test = lb.transform(X_test["school_state"])
```

```
Unique Categories: 51 Embedding Size: 26
```

```
# For project_grade_category
no_of_unique_grade  = X_train["project_grade_category"].nunique()
embedding_size_grade = int(np.ceil((no_of_unique_grade)/2))
print('Unique Categories:', no_of_unique_grade,'Embedding Size:', embedding_size_grade)


# Defining Input and Embedding Layer for the same

input_project_grade_category = Input(shape=(1,),name="project_grade_category")
embedding_project_grade_category = Embedding(no_of_unique_grade,embedding_size_grade,name="emb_pr
flatten_4 = Flatten()(embedding_project_grade_category)


encoder_grade_train = lb.fit_transform(X_train["project_grade_category"])
encoder_grade_cv = lb.fit_transform(X_cv["project_grade_category"])
encoder_grade_test = lb.transform(X_test["project_grade_category"])
```

```
Unique Categories: 4 Embedding Size: 2
```

```
# For clean_categories
no_of_unique_grade  = X_train["clean_categories"].nunique()
embedding_size_grade = int(np.ceil((no_of_unique_grade)/2))
print('Unique Categories:', no_of_unique_grade,'Embedding Size:', embedding_size_grade)


# Defining Input and Embedding Layer for the same

input_clean_categories= Input(shape=(1,),name="clean_categories")
embedding_clean_categories = Embedding(500,embedding_size_grade,name="emb_clean_categories",trair
flatten_5 = Flatten()(embedding_clean_categories)

#https://stackoverflow.com/questions/21057621/sklearn-labelencoder-with-never-seen-before-values/
encoder_clean_categories_train = lb.fit_transform(X_train["clean_categories"])
X_test["clean_categories"] = X_test["clean_categories"].map(lambda s: '<unknown>' if s not in lb.
lb.classes_ = np.append(lb.classes_, '<unknown>')
encoder_clean_categories_test = lb.transform(X_test["clean_categories"])


encoder_clean_categories_cv = lb.fit_transform(X_train["clean_categories"])
X_cv["clean_categories"] = X_cv["clean_categories"].map(lambda s: '<unknown>' if s not in lb.clas
lb.classes_ = np.append(lb.classes_, '<unknown>')
encoder_clean_categories_cv = lb.transform(X_cv["clean_categories"])
```

```
Unique Categories: 50 Embedding Size: 25
```

```
# For clean_subcategories
no_of_unique_grade  = X_train["clean_subcategories"].nunique()
embedding_size_grade = int(np.ceil((no_of_unique_grade)/2))
print('Unique Categories:', no_of_unique_grade,'Embedding Size:', embedding_size_grade)
```

```
# Defining Input and Embedding Layer for the same

input_clean_subcategories= Input(shape=(1,),name="clean_subcategories")
embedding_clean_subcategories = Embedding(600,embedding_size_grade,name="emb_clean_subcategories"
flatten_6 = Flatten()(embedding_clean_subcategories)

#https://stackoverflow.com/questions/21057621/sklearn-labelencoder-with-never-seen-before-values/
encoder_clean_subcategories_train = lb.fit_transform(X_train["clean_subcategories"])
X_test["clean_subcategories"] = X_test["clean_subcategories"].map(lambda s: '<unknown>' if s not
lb.classes_ = np.append(lb.classes_, '<unknown>')
encoder_clean_subcategories_test = lb.transform(X_test["clean_subcategories"])

encoder_clean_subcategories_cv = lb.fit_transform(X_cv["clean_subcategories"])
X_cv["clean_subcategories"] = X_cv["clean_subcategories"].map(lambda s: '<unknown>' if s not in l
lb.classes_ = np.append(lb.classes_, '<unknown>')
encoder_clean_subcategories_cv = lb.transform(X_cv["clean_subcategories"])
```

```
Unique Categories: 379 Embedding Size: 190
```

```
# Now we will prepare numerical features for our model
#Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.r
#num_train_1=X_train['project_summary_numerical'].values
num_train_1=X_train['price'].values.reshape(-1, 1)
#num_train_3=X_train['quantity'].values
num_train_2=X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)

#num_test_1=X_test['project_summary_numerical'].values
num_test_1=X_test['price'].values.reshape(-1, 1)
#num_test_3=X_test['quantity'].values
num_test_2=X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)

#num_test_1=X_test['project_summary_numerical'].values
num_cv_1=X_cv['price'].values.reshape(-1, 1)
#num_test_3=X_test['quantity'].values
num_cv_2=X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)


num_train=np.concatenate((num_train_1,num_train_2),axis=1)
num_cv=np.concatenate((num_cv_1,num_cv_2),axis=1)
num_test=np.concatenate((num_test_1,num_test_2),axis=1)
```

```
from sklearn.preprocessing import StandardScaler
norm=StandardScaler()
norm_train=norm.fit_transform(num_train)
norm_test=norm.transform(num_test)
norm_cv=norm.transform(num_cv)
```

```
norm_train.shape
```

```
(76473, 2)
```

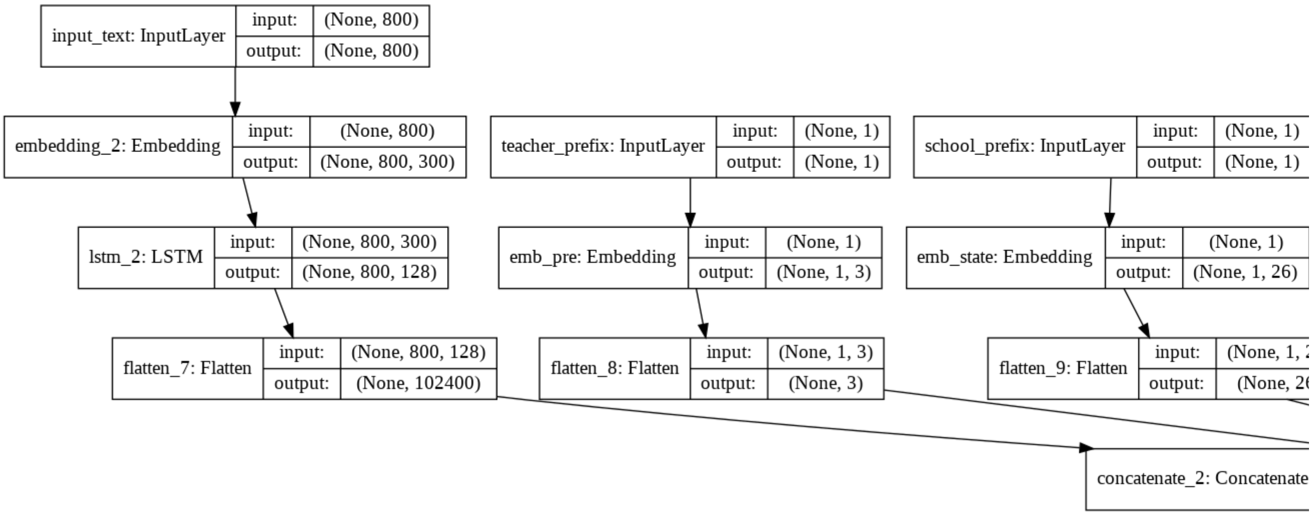
```
# Defining the Input and Embedding Layer for the same

num_feats = Input(shape=(2,),name="numerical_features")
num_feats_ = Dense(100,activation="relu",kernel_initializer='he_normal',kernel_regularizer=regula
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
```

```
x_concatenate = concatenate([flatten_1,flatten_2,flatten_3,flatten_4,flatten_5,flatten_6,num_feat
x = Dense(32,activation="relu", kernel_initializer='he_normal',kernel_regularizer=regularizers.l2

x=Dropout(0.5)(x)
x = Dense(128,activation="relu",kernel_initializer='he_normal',kernel_regularizer=regularizers.l2

x=Dropout(0.65)(x)
x = Dense(64,activation="relu", kernel_initializer='he_normal',kernel_regularizer=regularizers.l2
#x = BatchNormalization()(x)

output = Dense(2, activation='softmax', kernel_initializer="glorot_uniform",name='output')(x)
model_1 = Model(inputs=[input_text,
input_prefix,
input_state,
input_project_grade_category,
input_clean_categories,
input_clean_subcategories,
num_feats],outputs=[output])
```

```
WARNING:tensorflow:Large dropout rate: 0.65 (>0.5). In TensorFlow 2.x, dropout() uses
```

```
from keras.utils import plot_model
import pydot_ng as pydot
plot_model(model_1, show_shapes=True, show_layer_names=True, to_file='model_1.png')
from IPython.display import Image
Image(retina=True, filename='model_1.png')
```



```
train_data_1 = [encoded_train,
encoder_prefix_train,
encoder_state_train,
encoder_grade_train,
encoder_clean_categories_train,
encoder_clean_subcategories_train,
norm_train]
test_data_1 = [encoded_test,
encoder_prefix_test,
encoder_state_test,
encoder_grade_test,
encoder_clean_categories_test,
encoder_clean_subcategories_test ,
norm_test]
cv_data_1 = [encoded_cv,
encoder_prefix_cv,
encoder_state_cv,
encoder_grade_cv,
encoder_clean_categories_cv,
encoder_clean_subcategories_cv ,
norm_cv]
```

```
norm_train.shape
```

```
(76473, 2)
```

```
# Defining Custom ROC-AUC Metrics
from sklearn.metrics import roc_auc_score

def auc1(y_true, y_pred):
  return roc_auc_score(y_true, y_pred)


#https://www.tensorflow.org/api_docs/python/tf/py_func
def auc(y_true, y_pred):
    return tf.py_func(auc1, (y_true, y_pred), tf.double)
```

```
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[auc])
```

```
##convert Y to one hot coding vectors
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(categories='auto')
Y_train = encoder.fit_transform(y_train.reshape(-1,1))
Y_cv = encoder.fit_transform(y_cv.reshape(-1,1))
Y_test = encoder.transform(y_test.reshape(-1,1))

Y_train = Y_train.toarray()
Y_cv = Y_cv.toarray()
Y_test = Y_test.toarray()
```

Double-click (or enter) to edit

```
from tensorboardcolab import *
```

```
tbc=TensorBoardColab()
```

```
Wait for 8 seconds...
TensorBoard link:
https://178b9191.ngrok.io
```

```
history_1 = model_1.fit(train_data_1,Y_train,batch_size=512,
                         epochs=10,validation_data=(cv_data_1,Y_cv),callbacks=[TensorBoardColabCal
```

```
Train on 49041 samples, validate on 24155 samples
Epoch 1/10
49041/49041 [==============================] - 1064s 22ms/step - loss: 1.2731 - auc:
Epoch 2/10
49041/49041 [==============================] - 1065s 22ms/step - loss: 0.9089 - auc:
Epoch 3/10
49041/49041 [==============================] - 1071s 22ms/step - loss: 0.7787 - auc:
Epoch 4/10
49041/49041 [==============================] - 1069s 22ms/step - loss: 0.6918 - auc:
Epoch 5/10
49041/49041 [==============================] - 1066s 22ms/step - loss: 0.6297 - auc:
Epoch 6/10
49041/49041 [==============================] - 1133s 23ms/step - loss: 0.5816 - auc:
Epoch 7/10
49041/49041 [==============================] - 1105s 23ms/step - loss: 0.5471 - auc:
Epoch 8/10
49041/49041 [==============================] - 1105s 23ms/step - loss: 0.5154 - auc:
Epoch 9/10
49041/49041 [==============================] - 1099s 22ms/step - loss: 0.4964 - auc:
Epoch 10/10
49041/49041 [==============================] - 1107s 23ms/step - loss: 0.4777 - auc:
```

```
model_1 = load_model('/content/gdrive/My Drive/Colab Notebooks/model_1.h5', custom_objects={'auc'
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From <ipython-input-37-7e5f36a9008f>:2: py_func (from tensorflow.p
Instructions for updating:
tf.py_func is deprecated in TF V2. Instead, there are two
    options available in V2.
    - tf.py_function takes a python function which manipulates tf eager
    tensors instead of numpy arrays. It's easy to convert a tf eager tensor to
    an ndarray (just call tensor.numpy()) but having access to eager tensors
    means `tf.py_function`s can use accelerators such as GPUs as well as
    being differentiable using a gradient tape.
    - tf.numpy_function maintains the semantics of the deprecated tf.py_func
    (it is not differentiable, and manipulates numpy arrays). It drops the
    stateful argument making all functions stateful.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
```

```
scores = model_1.evaluate(test_data_1, Y_test, verbose=0,batch_size=512)
#print("%s: %.2f%%" % (model_1.metrics_names[0], scores[0]*100))
```

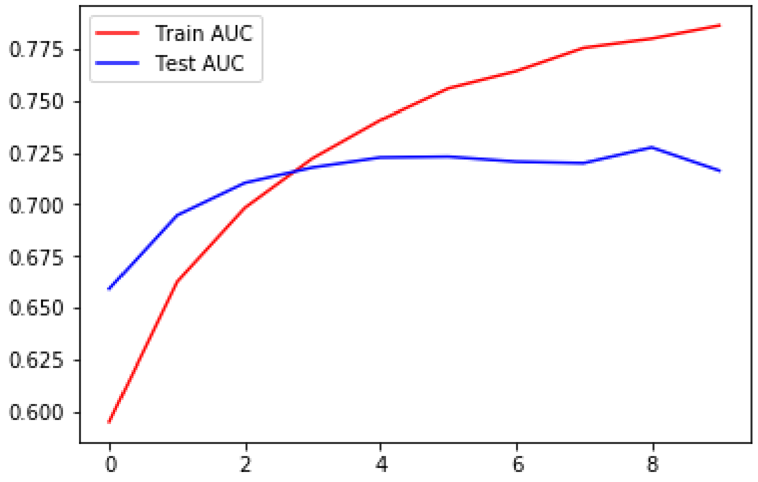
```
print("%s: %.2f%%" % (model_1.metrics_names[1], scores[1]*100))
```

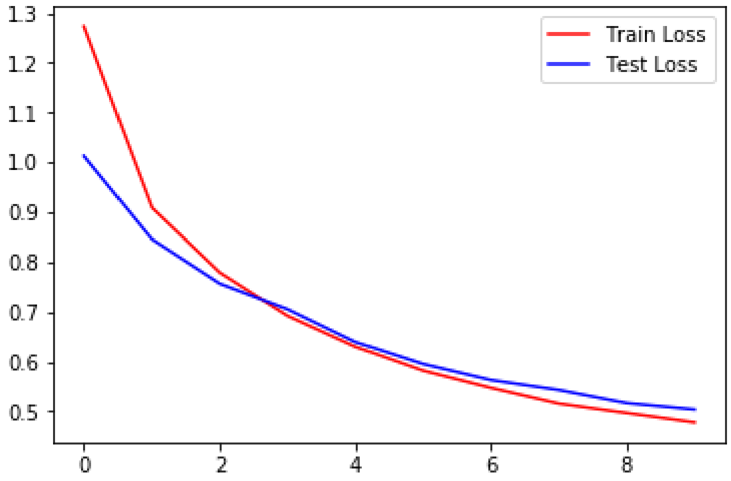
```
auc: 73.96%
```

```
model_1.save("/content/gdrive/My Drive/Colab Notebooks/model_12.h5")
```

```
plt.plot(history_1.history['auc'], 'r')
plt.plot(history_1.history['val_auc'], 'b')
plt.legend({'Train AUC': 'r', 'Test AUC':'b'})
plt.show()
```

```
plt.plot(history_1.history['loss'], 'r')
plt.plot(history_1.history['val_loss'], 'b')
plt.legend({'Train Loss': 'r', 'Test Loss':'b'})
plt.show()
```





- For LSTM, you can choose your sequence padding methods on your own or you can train your LSTM with

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

```
# 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 Embed
https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

2. Please go through this link https://keras.io/getting-started/functional-api-guide/ and check

Model-2

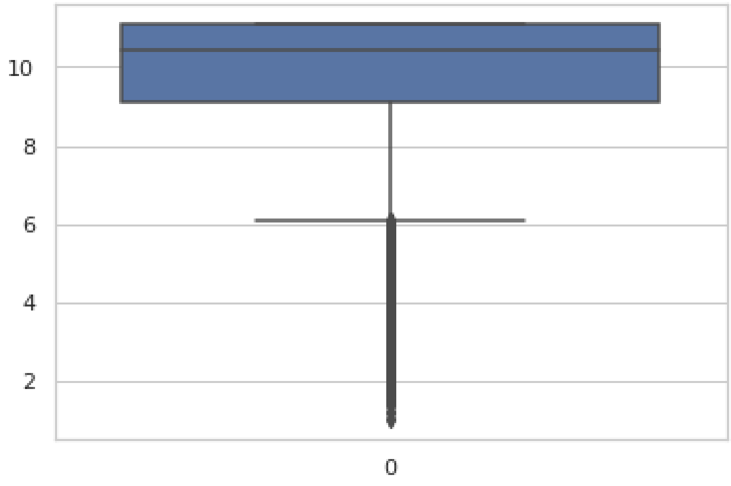
Use the same model as above but for 'input_seq_total_text_data' give only some words in the sentance not all th

```
1. Train the TF-IDF on the Train data feature 'essay'

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 o
values choose the low and high threshold value. Because very frequent words and very v
information. (you can plot a box plots and take only the idf scores within IQR range a

4. Train the LSTM after removing the Low and High idf value words. (In model-1 Train o
data after removing some words based on IDF values)
```

```
# Filtering Text Data based on idf values

tfidf = TfidfVectorizer(ngram_range=(1,1))
tfidf.fit(X_train["essay"])

# converting to dictionary
combine_dict = dict(zip(tfidf.get_feature_names(),list(tfidf.idf_)))
```

```
tfidf_df = pd.DataFrame(list(combine_dict.items()), columns=['Word', 'TFIDF'])
tfidf_df = tfidf_df.sort_values(by ='TFIDF' )
```

```
import seaborn as sns
sns.set(style="whitegrid")
ax = sns.boxplot( data= tfidf_df['TFIDF'] )
```



```
print("MIN VAL:",tfidf_df["TFIDF"].min(),"MAX VALUE",tfidf_df["TFIDF"].max())
```

```
MIN VAL: 1.0077713116568965 MAX VALUE 11.551558912204982
```

```
print("\nPercentiles:")
for i in range(1,101,5):
  print(i,"Percentile:")
  print(np.percentile(tfidf_df['TFIDF'],i))
```

```

Percentiles:
1 Percentile:
4.082632013181665
6 Percentile:
6.56453348374786
11 Percentile:
7.690829201164387
16 Percentile:
8.460516458846666
21 Percentile:
9.109211876835777
26 Percentile:
9.605648763149668
31 Percentile:
10.047481515428709
36 Percentile:
10.298795943709614
41 Percentile:
10.635268180330828
46 Percentile:
10.858411731645036
51 Percentile:
11.146093804096818
56 Percentile:
11.146093804096818
61 Percentile:
11.551558912204982
66 Percentile:
11.551558912204982
71 Percentile:
11.551558912204982
76 Percentile:
11.551558912204982
81 Percentile:
11.551558912204982
86 Percentile:
11.551558912204982
91 Percentile:
11.551558912204982
96 Percentile:
11.551558912204982
```

```
tfidf_df['TFIDF'][0]
```

```
7.182111059737961
```

```
comp_tfidf = []
for i in range(tfidf_df.shape[0]):
  if (tfidf_df['TFIDF'][i] > 4 and tfidf_df['TFIDF'][i] < 10 ):
    comp_tfidf.append(tfidf_df['Word'][i])
```

```
len(comp_tfidf)
```

```
14648
```

```
unique = set(comp_tfidf)
print("NO OF UNIQUE WORDS IN TRAIN ESSAY",len(unique))
```

```
NO OF UNIQUE WORDS IN TRAIN ESSAY 14734
```

```
#Converts a text to a sequence of words (or tokens).
#A list of words (or tokens).
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words = len(unique))
tokenizer.fit_on_texts(comp_tfidf)
sequences_train = tokenizer.texts_to_sequences(X_train["essay"])
sequences_cv = tokenizer.texts_to_sequences(X_cv["essay"])
sequences_test = tokenizer.texts_to_sequences(X_test["essay"])
```

```
word2idx = tokenizer.word_index
print('Found %s unique tokens.' % len(word2idx))
```

```
Found 14734 unique tokens.
```

```
encoded_train = pad_sequences(sequences_train,maxlen=800,padding='post', truncating='post')
print('Shape of data tensor:', encoded_train.shape)
encoded_test = pad_sequences(sequences_test, maxlen=800,padding='post', truncating='post')
print('Shape of data tensor:', encoded_test.shape)
```

```
Shape of data tensor: (49041, 800)
Shape of data tensor: (36052, 800)
```

```
encoded_cv = pad_sequences(sequences_cv, maxlen=800,padding='post', truncating='post')
```

```
num_words = len(word2idx) + 1
embedding_matrix = np.zeros((num_words, 300))
for word, i in word2idx.items():
    embedding_vector = glove_words.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

```
print(num_words)
print(embedding_matrix.shape)
```

```
14735
(14735, 300)
```

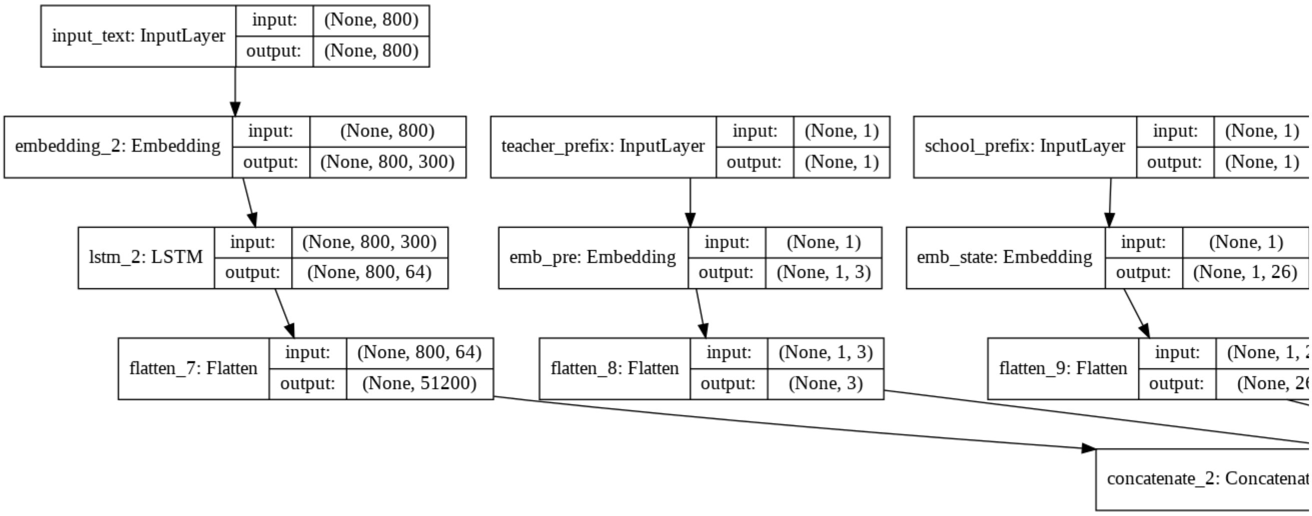
```
j = 'he_normal'
i = 0.001
input_text = Input(shape=(800,),name="input_text")
X = Embedding(num_words,300,weights=[embedding_matrix],input_length=800,trainable=False)(input_te
X = LSTM(64,recurrent_dropout=0.35,kernel_regularizer=regularizers.l2(i),return_sequences=True)(X
flatten_1 = Flatten()(X)

# Teacher Prefix
#no_of_unique_prefix  = set(X_train['teacher_prefix'].values)
no_of_unique_prefix  = X_train["teacher_prefix"].nunique()
print('Unique Categories:', (no_of_unique_prefix))


# Defining Input and Embedding Layer for the same
```

```
input_prefix = Input(shape=(1,),name="teacher_prefix")
embedding_prefix = Embedding(no_of_unique_prefix,3,name="emb_pre",trainable=True)(input_prefix)
flatten_2 = Flatten()(embedding_prefix)

lb = LabelEncoder()
encoder_prefix_train = lb.fit_transform(X_train["teacher_prefix"])
encoder_prefix_cv = lb.transform(X_cv["teacher_prefix"])
encoder_prefix_test = lb.transform(X_test["teacher_prefix"])

# School State
no_of_unique_state  = X_train["school_state"].nunique()
embedding_size_state= int(np.ceil((no_of_unique_state)/2))
print('Unique Categories:', no_of_unique_state,'Embedding Size:', embedding_size_state)


# Defining Input and Embedding Layer for the same

input_state = Input(shape=(1,),name="school_prefix")
embedding_state = Embedding(no_of_unique_state,embedding_size_state,name="emb_state",trainable=Tr
flatten_3 = Flatten()(embedding_state)


encoder_state_train = lb.fit_transform(X_train["school_state"])
encoder_state_cv = lb.transform(X_cv["school_state"])
encoder_state_test = lb.transform(X_test["school_state"])

# For project_grade_category
no_of_unique_grade  = X_train["project_grade_category"].nunique()
embedding_size_grade = int(np.ceil((no_of_unique_grade)/2))
print('Unique Categories:', no_of_unique_grade,'Embedding Size:', embedding_size_grade)


# Defining Input and Embedding Layer for the same

input_project_grade_category = Input(shape=(1,),name="project_grade_category")
embedding_project_grade_category = Embedding(no_of_unique_grade,embedding_size_grade,name="emb_pr
flatten_4 = Flatten()(embedding_project_grade_category)


encoder_grade_train = lb.fit_transform(X_train["project_grade_category"])
encoder_grade_cv = lb.transform(X_cv["project_grade_category"])
encoder_grade_test = lb.transform(X_test["project_grade_category"])


# For clean_categories
no_of_unique_grade  = X_train["clean_categories"].nunique()
embedding_size_grade = int(np.ceil((no_of_unique_grade)/2))
print('Unique Categories:', no_of_unique_grade,'Embedding Size:', embedding_size_grade)


# Defining Input and Embedding Layer for the same

input_clean_categories= Input(shape=(1,),name="clean_categories")
embedding_clean_categories = Embedding(500,embedding_size_grade,name="emb_clean_categories",train
flatten_5 = Flatten()(embedding_clean_categories)

#https://stackoverflow.com/questions/21057621/sklearn-labelencoder-with-never-seen-before-values/
encoder_clean_categories_train = lb.fit_transform(X_train["clean_categories"])
X_test["clean_categories"] = X_test["clean_categories"].map(lambda s: '<unknown>' if s not in lb.
lb.classes_ = np.append(lb.classes_, '<unknown>')
encoder_clean_categories_test = lb.transform(X_test["clean_categories"])

encoder_clean_categories_cv = lb.fit_transform(X_cv["clean_categories"])
X_cv["clean_categories"] = X_cv["clean_categories"].map(lambda s: '<unknown>' if s not in lb.clas
lb.classes_ = np.append(lb.classes_, '<unknown>')
encoder_clean_categories_cv = lb.transform(X_cv["clean_categories"])

# For clean_subcategories
no_of_unique_grade  = X_train["clean_subcategories"].nunique()
embedding_size_grade = int(np.ceil((no_of_unique_grade)/2))
print('Unique Categories:', no_of_unique_grade,'Embedding Size:', embedding_size_grade)


# Defining Input and Embedding Layer for the same
```

```
input_clean_subcategories= Input(shape=(1,),name="clean_subcategories")
embedding_clean_subcategories = Embedding(600,embedding_size_grade,name="emb_clean_subcategories"
flatten_6 = Flatten()(embedding_clean_subcategories)

#https://stackoverflow.com/questions/21057621/sklearn-labelencoder-with-never-seen-before-values/
encoder_clean_subcategories_train = lb.fit_transform(X_train["clean_subcategories"])
X_test["clean_subcategories"] = X_test["clean_subcategories"].map(lambda s: '<unknown>' if s not
lb.classes_ = np.append(lb.classes_, '<unknown>')
encoder_clean_subcategories_test = lb.transform(X_test["clean_subcategories"])

encoder_clean_subcategories_cv = lb.fit_transform(X_cv["clean_subcategories"])
X_cv["clean_subcategories"] = X_cv["clean_subcategories"].map(lambda s: '<unknown>' if s not in l
lb.classes_ = np.append(lb.classes_, '<unknown>')
encoder_clean_subcategories_cv = lb.transform(X_cv["clean_subcategories"])

# Now we will prepare numerical features for our model
#Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.r
#num_train_1=X_train['project_summary_numerical'].values
num_train_1=X_train['price'].values.reshape(-1, 1)
#num_train_3=X_train['quantity'].values
num_train_2=X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)

#num_test_1=X_test['project_summary_numerical'].values
num_test_1=X_test['price'].values.reshape(-1, 1)
#num_test_3=X_test['quantity'].values
num_test_2=X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)

#num_test_1=X_test['project_summary_numerical'].values
num_cv_1=X_cv['price'].values.reshape(-1, 1)
#num_test_3=X_test['quantity'].values
num_cv_2=X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)


num_train=np.concatenate((num_train_1,num_train_2),axis=1)
num_cv=np.concatenate((num_cv_1,num_cv_2),axis=1)
num_test=np.concatenate((num_test_1,num_test_2),axis=1)


from sklearn.preprocessing import StandardScaler
norm=StandardScaler()
norm_train=norm.fit_transform(num_train)
norm_cv=norm.transform(num_cv)
norm_test=norm.transform(num_test)

# Defining the Input and Embedding Layer for the same

num_feats = Input(shape=(2,),name="numerical_features")
num_feats_ = Dense(32,activation="relu",kernel_initializer=j,kernel_regularizer=regularizers.l2(i

x_concatenate = concatenate([flatten_1,flatten_2,flatten_3,flatten_4,flatten_5,flatten_6,num_feat

x = Dense(64,activation="relu", kernel_initializer=j,kernel_regularizer=regularizers.l2(i))(x_con
x=Dropout(0.5)(x)
x = Dense(256,activation="relu",kernel_initializer=j,kernel_regularizer=regularizers.l2(i))(x)

x=Dropout(0.5)(x)
x = Dense(16,activation="relu", kernel_initializer=j,kernel_regularizer=regularizers.l2(i))(x)
#x = BatchNormalization()(x)

output = Dense(2,activation='softmax',kernel_initializer="glorot_uniform", name='output')(x)
model_1 = Model(inputs=[input_text,
input_prefix,
input_state,
input_project_grade_category,
input_clean_categories,
input_clean_subcategories,
num_feats],outputs=[output])
```

```
Unique Categories: 5
Unique Categories: 51 Embedding Size: 26
Unique Categories: 4 Embedding Size: 2
Unique Categories: 50 Embedding Size: 25
Unique Categories: 378 Embedding Size: 189
```

```
from keras.utils import plot_model
import pydot_ng as pydot
plot_model(model_1, show_shapes=True, show_layer_names=True, to_file='model_2.png')
from IPython.display import Image
Image(retina=True, filename='model_2.png')
```



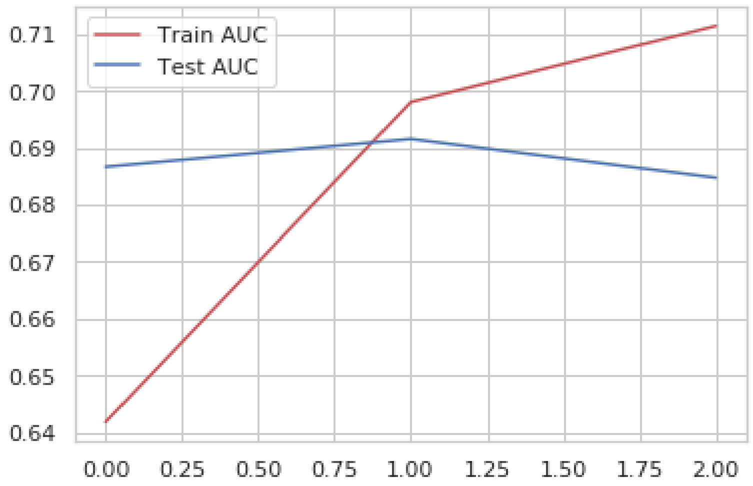
```
train_data_1 = [encoded_train,
encoder_prefix_train,
encoder_state_train,
encoder_grade_train,
encoder_clean_categories_train,
encoder_clean_subcategories_train,
norm_train]
test_data_1 = [encoded_test,
encoder_prefix_test,
encoder_state_test,
encoder_grade_test,
encoder_clean_categories_test,
encoder_clean_subcategories_test ,
norm_test]
cv_data_1 = [encoded_cv,
encoder_prefix_cv,
encoder_state_cv,
encoder_grade_cv,
encoder_clean_categories_cv,
encoder_clean_subcategories_cv ,
norm_cv]


def auc1(y_true, y_pred):
    return roc_auc_score(y_true, y_pred)



#https://www.tensorflow.org/api_docs/python/tf/py_func
def auc(y_true, y_pred):
  return tf.py_func(auc1, (y_true, y_pred), tf.double)


model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[auc])


##convert Y to one hot coding vectors
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(categories='auto')
Y_train = encoder.fit_transform(y_train.reshape(-1,1))
Y_cv = encoder.fit_transform(y_cv.reshape(-1,1))
Y_test = encoder.transform(y_test.reshape(-1,1))

Y_train = Y_train.toarray()
Y_cv = Y_cv.toarray()
Y_test = Y_test.toarray()


def auc(y_true, y_pred):
    return tf.py_func(roc_auc_score, (y_true, y_pred), tf.double)


history_1 = model_1.fit(train_data_1,Y_train,batch_size=512,
                        epochs=3,validation_data=(cv_data_1,Y_cv),callbacks=[TensorBoardColabCall
```

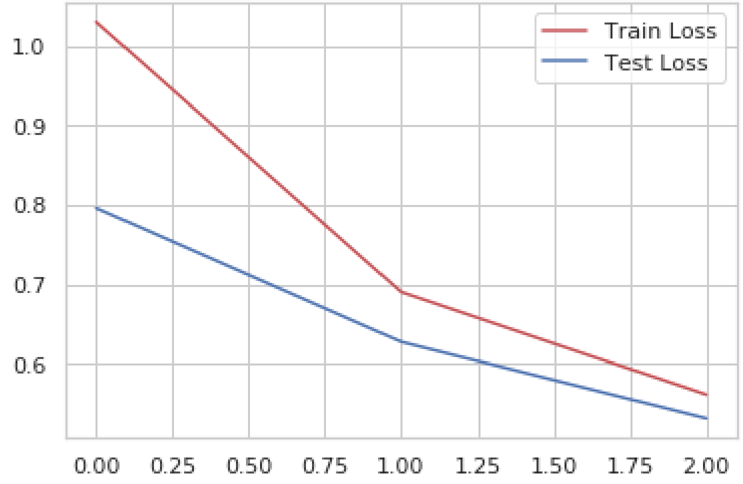
```
Train on 49041 samples, validate on 24155 samples
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorboardcolab/core.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/callbacks.py:112

Epoch 1/3
49041/49041 [==============================] - 1048s 21ms/step - loss: 1.0298 - auc:
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorboardcolab/callb

Epoch 2/3
49041/49041 [==============================] - 1036s 21ms/step - loss: 0.6899 - auc:
Epoch 3/3
49041/49041 [==============================] - 1001s 20ms/step - loss: 0.5611 - auc:
```

```
model_1.save("/content/gdrive/My Drive/Colab Notebooks/model_2.h5")


scores = model_1.evaluate(test_data_1, Y_test, verbose=0,batch_size=512)
print("%s: %.2f%%" % (model_1.metrics_names[1], scores[1]*100))
```

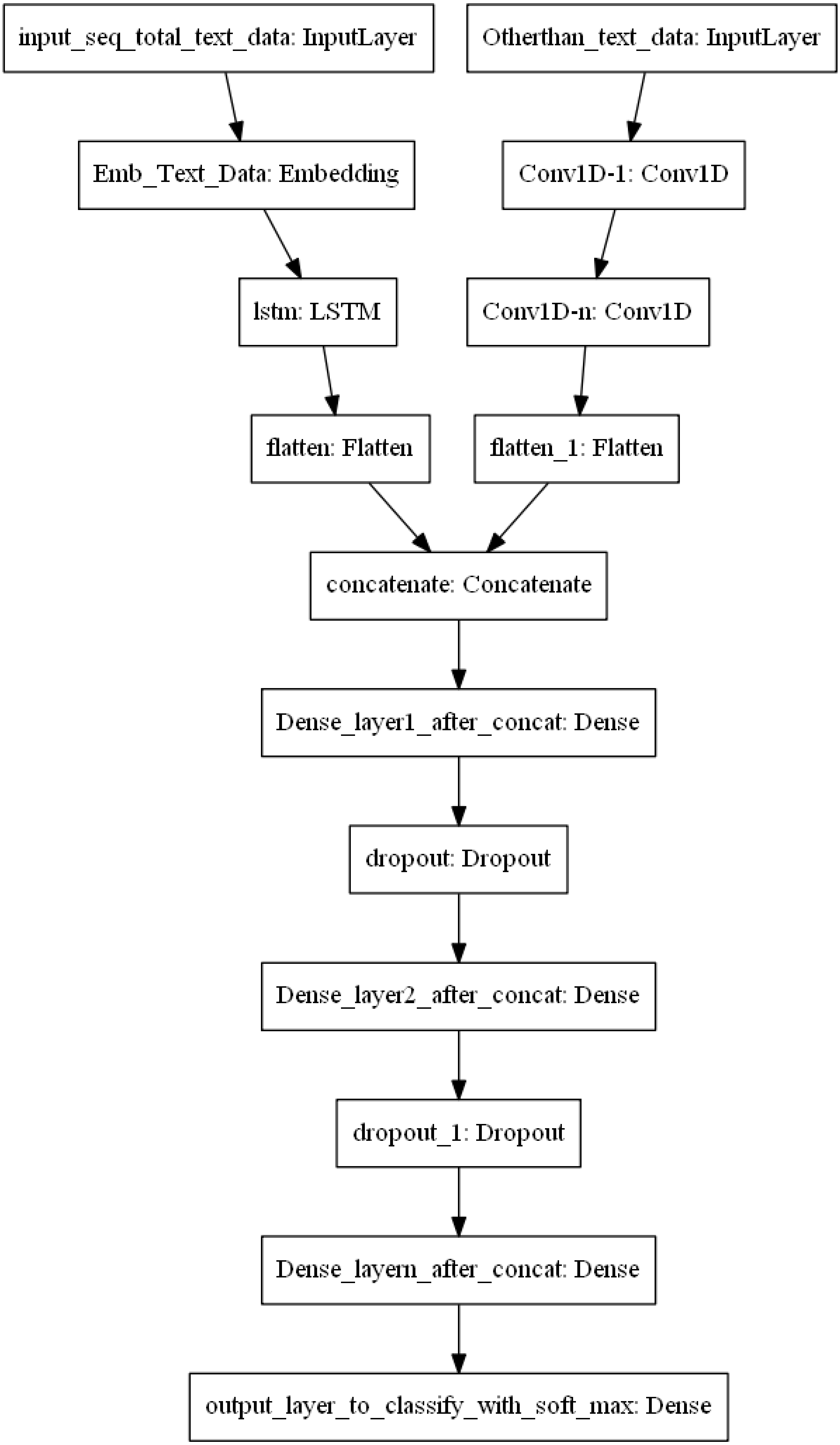
```
auc: 70.39%
```

```
plt.plot(history_1.history['auc'], 'r')
plt.plot(history_1.history['val_auc'], 'b')
plt.legend({'Train AUC': 'r', 'Test AUC':'b'})
plt.show()


plt.plot(history_1.history['loss'], 'r')
plt.plot(history_1.history['val_loss'], 'b')
plt.legend({'Train Loss': 'r', 'Test Loss':'b'})
plt.show()
```





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.ir

Please note that first part of input and model is taken from model one and the second half is the only thing whi

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values)


train_categories_ohe = vectorizer.transform(X_train['clean_categories'].values)
cv_categories_ohe = vectorizer.transform(X_cv['clean_categories'].values)
test_categories_ohe = vectorizer.transform(X_test['clean_categories'].values)
```

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values)
train_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
cv_subcategories_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
test_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].values)
```

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values)
train_state_ohe = vectorizer.transform(X_train['school_state'].values)
cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
test_state_ohe = vectorizer.transform(X_test['school_state'].values)
```

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values)
train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)
```

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values)
train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)
```

```
from scipy.sparse import hstack
stack_train = hstack((train_state_ohe, train_teacher_ohe, train_grade_ohe, train_categories_ohe,
print(stack_train.shape)
stack_cv = hstack((cv_state_ohe, cv_teacher_ohe, cv_grade_ohe, cv_categories_ohe, cv_subcategorie
print(stack_cv.shape)
stack_test = hstack((test_state_ohe, test_teacher_ohe, test_grade_ohe, test_categories_ohe, test_
print(stack_test.shape)
```

```
(49041, 101)
(24155, 101)
(36052, 101)
```

```
other_than_text_data_train = np.expand_dims(stack_train,2)
other_than_text_data_cv = np.expand_dims(stack_cv,2)
other_than_text_data_test = np.expand_dims(stack_test,2)
```

```
other_than_text_data_train.shape
```

```
(49041, 101, 1)
```

```
input_layer_other_than_text_data = Input(shape=(101,1),name="other_than_text_data")
conv1D_1 = Conv1D(filters=128, kernel_size=3, activation='relu',kernel_initializer="he_normal")(i
conv1D_2 = Conv1D(filters=128, kernel_size=3, activation='relu',kernel_initializer="he_normal")(c
flatten_other_than_text_data = Flatten()(conv1D_2)
```

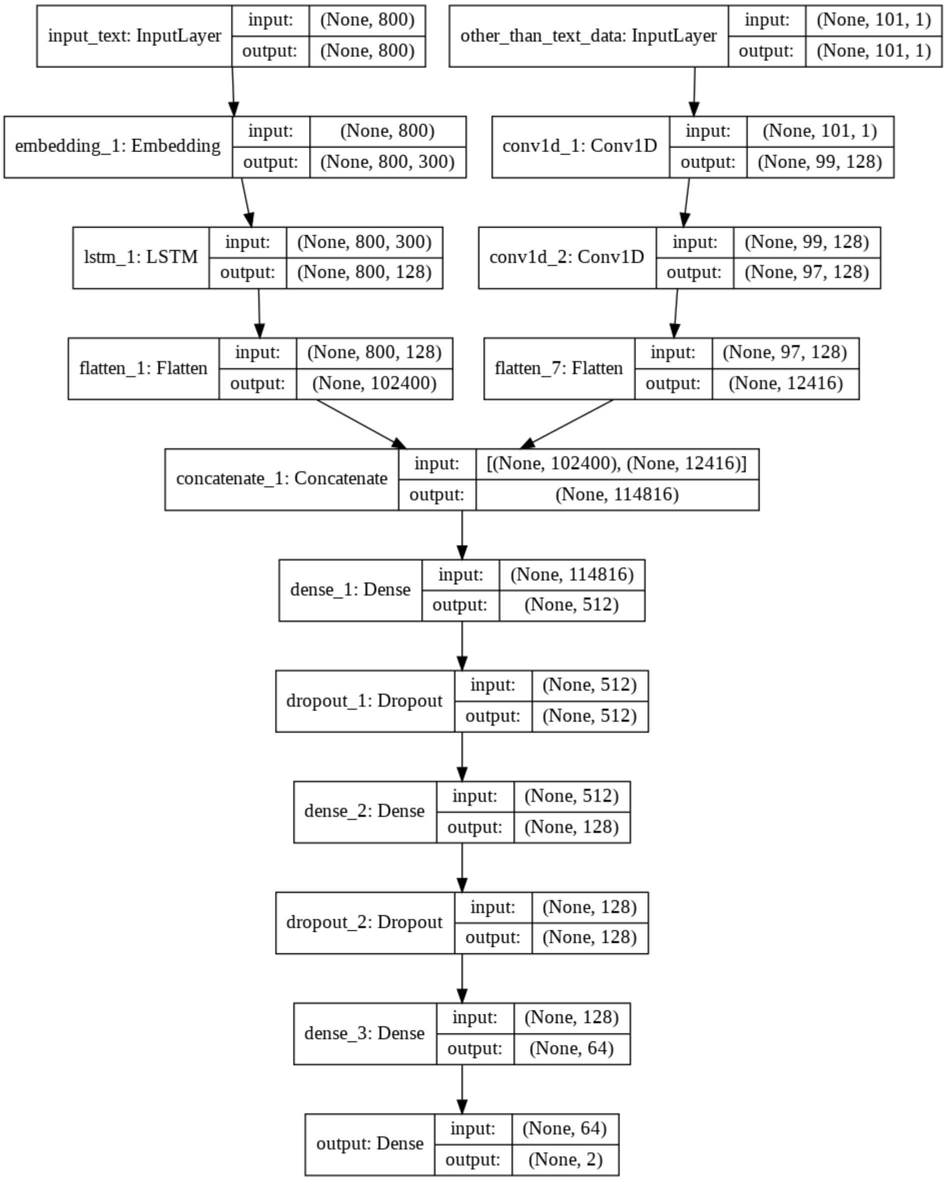
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
```

```
concatenate_layer_model_3 = concatenate([flatten_1,flatten_other_than_text_data])
model_3_dense_layer_1 = Dense(512,activation="relu",kernel_initializer="he_normal",kernel_regular
do3_1 = Dropout(0.5)(model_3_dense_layer_1)
model_3_dense_layer_2 = Dense(128,activation="relu",kernel_initializer="he_normal",kernel_regular
do3_2 = Dropout(0.5)(model_3_dense_layer_2)
model_3_dense_layer_3 = Dense(64,activation="relu",kernel_initializer="he_normal",kernel_regulari
output_3 = Dense(2, activation='softmax', name='output')(model_3_dense_layer_3)
```

```
model_1 = Model(inputs=[input_text,input_layer_other_than_text_data],outputs=[output_3])
```

```
X_train_3 = [encoded_train,other_than_text_data_train]
X_test_3 = [encoded_test,other_than_text_data_test]
X_cv_3 = [encoded_cv,other_than_text_data_cv]
```

```
from keras.utils import plot_model
plot_model(model_1, show_shapes=True, show_layer_names=True, to_file='model_3.png')
from IPython.display import Image
Image(retina=True, filename='model_3.png')
```


input_text: InputLayer | input: (None, 800) | output: (None, 800)
other_than_text_data: InputLayer | input: (None, 101, 1) | output: (None, 101, 1)
embedding_1: Embedding | input: (None, 800) | output: (None, 800, 300)
conv1d_1: Conv1D | input: (None, 101, 1) | output: (None, 99, 128)
lstm_1: LSTM | input: (None, 800, 300) | output: (None, 800, 128)
conv1d_2: Conv1D | input: (None, 99, 128) | output: (None, 97, 128)
flatten_1: Flatten | input: (None, 800, 128) | output: (None, 102400)
flatten_7: Flatten | input: (None, 97, 128) | output: (None, 12416)
concatenate_1: Concatenate | input: [(None, 102400), (None, 12416)] | output: (None, 114816)
dense_1: Dense | input: (None, 114816) | output: (None, 512)
dropout_1: Dropout | input: (None, 512) | output: (None, 512)
dense_2: Dense | input: (None, 512) | output: (None, 128)
dropout_2: Dropout | input: (None, 128) | output: (None, 128)
dense_3: Dense | input: (None, 128) | output: (None, 64)
output: Dense | input: (None, 64) | output: (None, 2)

```
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[auc])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From <ipython-input-24-7e5f36a9008f>:2: py_func (from tensorflow.p
Instructions for updating:
tf.py_func is deprecated in TF V2. Instead, there are two
    options available in V2.
    - tf.py_function takes a python function which manipulates tf eager
    tensors instead of numpy arrays. It's easy to convert a tf eager tensor to
    an ndarray (just call tensor.numpy()) but having access to eager tensors
    means `tf.py_function`s can use accelerators such as GPUs as well as
    being differentiable using a gradient tape.
    - tf.numpy_function maintains the semantics of the deprecated tf.py_func
    (it is not differentiable, and manipulates numpy arrays). It drops the
    stateful argument making all functions stateful.
```

```
print(encoded_cv.shape,norm_cv_1.shape)
```

```
(24155, 800) (24155, 7, 1)
```

```
history_1 = model_1.fit(X_train_3,Y_train,batch_size=512,
                        epochs=10,validation_data=(X_cv_3,Y_cv),callbacks=[TensorBoardColabCallba
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

Train on 49041 samples, validate on 24155 samples
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorboardcolab/core.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/callbacks.py:112

Epoch 1/10
49041/49041 [==============================] - 1194s 24ms/step - loss: 3.0560 - auc:
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorboardcolab/callb

Epoch 2/10
49041/49041 [==============================] - 1195s 24ms/step - loss: 2.6564 - auc:
Epoch 3/10
49041/49041 [==============================] - 1172s 24ms/step - loss: 2.6059 - auc:
Epoch 4/10
49041/49041 [==============================] - 1171s 24ms/step - loss: 0.8437 - auc:
Epoch 5/10
49041/49041 [==============================] - 1164s 24ms/step - loss: 0.5572 - auc:
Epoch 6/10
49041/49041 [==============================] - 1159s 24ms/step - loss: 0.5282 - auc:
Epoch 7/10
49041/49041 [==============================] - 1157s 24ms/step - loss: 0.5035 - auc:
Epoch 8/10
49041/49041 [==============================] - 1146s 23ms/step - loss: 0.4867 - auc:
Epoch 9/10
49041/49041 [==============================] - 1144s 23ms/step - loss: 0.4741 - auc:
Epoch 10/10
49041/49041 [==============================] - 1151s 23ms/step - loss: 0.4653 - auc:
```

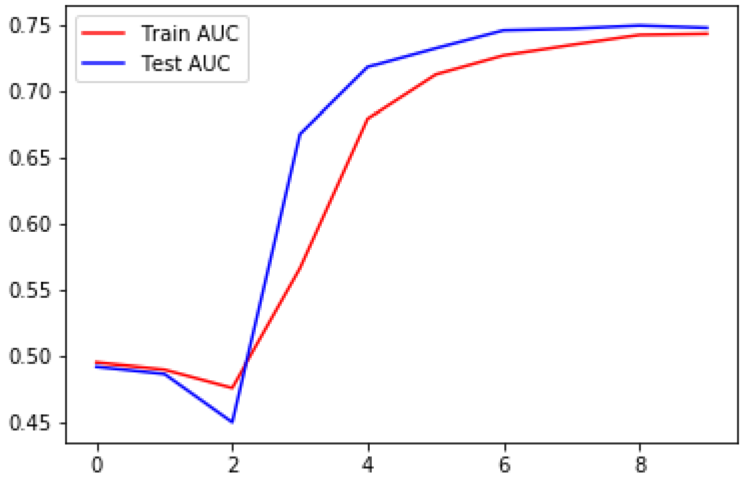
```
scores = model_1.evaluate(X_test_3, Y_test, verbose=0,batch_size=512)
print("%s: %.2f%%" % (model_1.metrics_names[1], scores[1]*100))
```

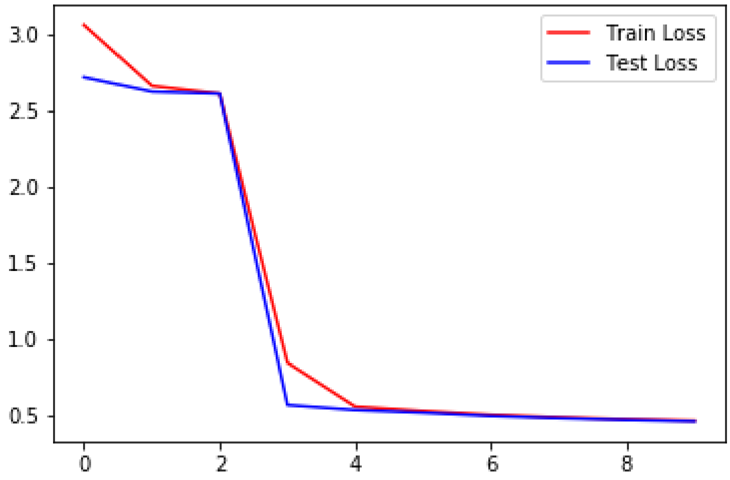
```
auc: 74.69%
```

```
model_1.save("/content/gdrive/My Drive/Colab Notebooks/model_3.h5")
```

```
plt.plot(history_1.history['auc'], 'r')
plt.plot(history_1.history['val_auc'], 'b')
plt.legend({'Train AUC': 'r', 'Test AUC':'b'})
plt.show()
```

```
plt.plot(history_1.history['loss'], 'r')
plt.plot(history_1.history['val_loss'], 'b')
plt.legend({'Train Loss': 'r', 'Test Loss':'b'})
plt.show()
```





- **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

. Neumerical values and use CNN1D as shown in above figure.

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

```
print("SUMMARY")
from prettytable import PrettyTable

x = PrettyTable()
```

```
x.field_names = ["MODEL","AUC"]
x.add_row(["MODEL 1",73.96])
x.add_row(["MODEL 2",70.39])
x.add_row(["MODEL 3",74.69])
print(x)
```

```
SUMMARY
+---------+-------+
|  MODEL  |  AUC  |
+---------+-------+
| MODEL 1 | 73.96 |
| MODEL 2 | 70.39 |
| MODEL 3 | 74.69 |
+---------+-------+
```