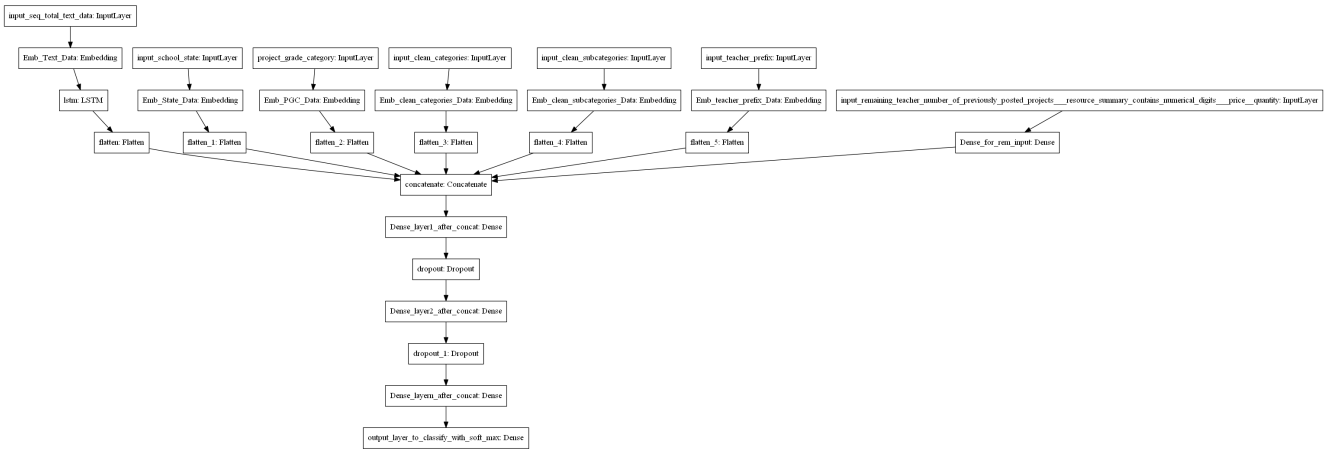
Assignment : 14

1. Preprocess all the Data we have in DonorsChoose Dataset 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' as a metric. check this 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, cs231n class video.
7. For all the model's use TensorBoard 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

- **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_numerical_digits._price** ---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.

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/

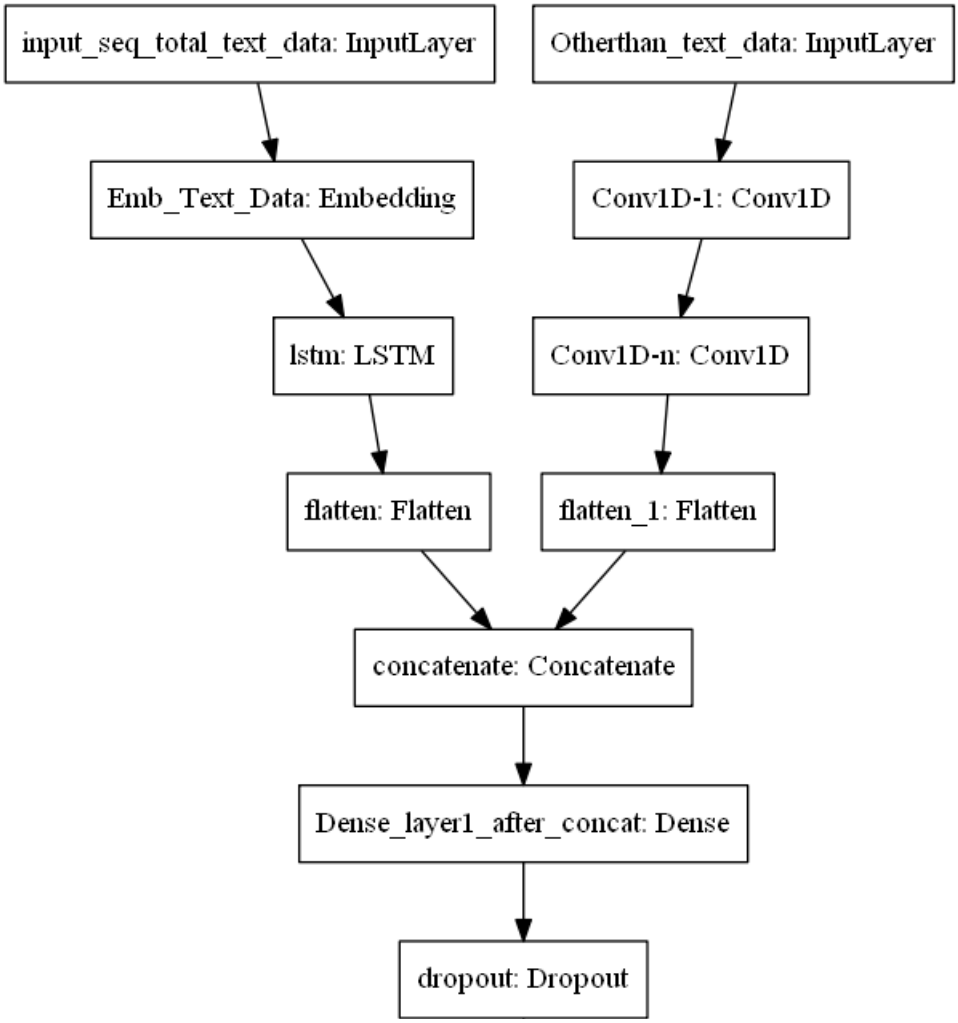
2. Please go through this link https://keras.io/getting-started/functional-api-guide/ and check the 'Multi-input and multi-output models' then you will get to know how to give multiple inputs.

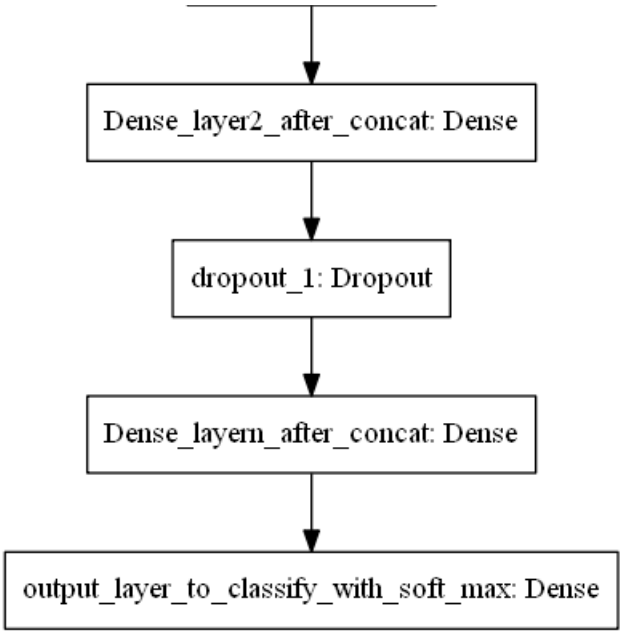
Model-2

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 pl
ots 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 tot
al data but in Model-2 train on data after removing some words based on IDF values)
```

Model-3





ref: 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 as shown in above figure.

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

</pre>

In [1]:

```
### Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from tensorflow.keras.utils import to_categorical
from sklearn.utils import compute_class_weight
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Input, Embedding, Flatten, LSTM, Dense, concatenate, Dropout,Ba
tchNormalization,SpatialDropout1D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.initializers import he_normal
from tensorflow.keras.regularizers import l2
from tensorflow.keras.models import Model, load_model
from sklearn.metrics import roc_auc_score
from tensorflow.keras import regularizers
from tensorflow.python.keras.callbacks import TensorBoard,ModelCheckpoint
from sklearn.preprocessing import Normalizer
import pickle
import warnings
```

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

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [2]:

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%
b&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly
ttps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

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

In [3]:

```
### read pre-processed data
project_data = pd.read_csv('/content/drive/My Drive/LSTM Assignment/preprocessed_data.csv')
project_data.shape
```

Out[3]:

```
(109248, 9)
```

In [4]:

```
y_data = project_data['project_is_approved']
x_data = project_data.drop(['project_is_approved'],axis=1)
```

In [5]:

```
### split ur data in train, test and Cross Validation data
x_train,x_test,y_train,y_test = train_test_split(x_data, y_data , stratify = y_data, train_size = 0
.8, random_state =99)

x_train,x_cv,y_train,y_cv = train_test_split(x_train, y_train, stratify = y_train, train_size = 0.8
, random_state = 99)
```

In [6]:

```
x_train.head()
```

Out[6]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_su
18589	mi	ms	grades_3_5	3	appliedlearning	charac
102121	ky	ms	grades_6_8	0	math_science	
40835	ca	ms	grades_3_5	0	math_science music_arts	environm

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_su
10349	ma	ms	grades_prek_2	17	literacy_language	
80988	ca	mrs	grades_prek_2	22	appliedlearning math_science	early healt

In [7]:

```
print("Shape of the Train dataset: ", x_train.shape[0])
print("Shape of the Test dataset: ", x_test.shape[0])
print("Shape of the cv dataset:", x_cv.shape[0])
```

```
Shape of the Train dataset:  69918
Shape of the Test dataset:  21850
Shape of the cv dataset: 17480
```

In [8]:

```
def tokenize_cat_data(x_train,x_cv,x_test,category):
    from tensorflow.keras.preprocessing.text import Tokenizer
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(x_train[category].tolist())
    seq_train = tokenizer.texts_to_sequences(x_train[category])
    seq_cv = tokenizer.texts_to_sequences(x_cv[category])
    seq_test = tokenizer.texts_to_sequences(x_test[category])
    vocab_size = len(tokenizer.word_index) + 1

    x_train[category] = seq_train

    x_train[category] = seq_train
    x_cv[category] = seq_cv
    x_test[category] = seq_test
    return x_train,x_cv,x_test, vocab_size
```

In [9]:

```
x_train,x_cv,x_test,school_state_size =
tokenize_cat_data(x_train,x_cv,x_test,category='school_state')
```

In [10]:

```
x_train,x_cv,x_test,tpr_size = tokenize_cat_data(x_train,x_cv,x_test,category='teacher_prefix')
```

In [11]:

```
x_train,x_cv,x_test,pgc_size =
tokenize_cat_data(x_train,x_cv,x_test,category='project_grade_category')
```

In [12]:

```
x_train,x_cv,x_test,cc_size = tokenize_cat_data(x_train,x_cv,x_test,category='clean_categories')
```

In [13]:

```
x_train,x_cv,x_test,csc_size =
tokenize_cat_data(x_train,x_cv,x_test,category='clean_subcategories')
```

In [14]:

```
x_train.head(10)
```

Out[14]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_su

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_su
18589	[9]	[2]	[1, 4, 5]	3	[8]	
102121	[29]	[2]	[1, 6, 7]	0	[3, 4]	
40835	[1]	[2]	[1, 4, 5]	0	[3, 4, 9, 10]	
10349	[14]	[2]	[1, 2, 3]	17	[1, 2]	
80988	[1]	[1]	[1, 2, 3]	22	[8, 3, 4]	
11622	[15]	[1]	[1, 8, 9]	7	[1, 2]	
78099	[18]	[1]	[1, 2, 3]	1	[1, 2, 3, 4]	
86441	[22]	[1]	[1, 2, 3]	0	[3, 4]	
88770	[21]	[2]	[1, 4, 5]	0	[1, 2, 3, 4]	
87673	[1]	[1]	[1, 2, 3]	1	[3, 4]	

In [15]:

```
#converting class labels to two class lable categorical variables
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_cv = to_categorical(y_cv)
```

In [16]:

```
## class weightage -- pass in our model fit parameter.
class_wght = compute_class_weight("balanced", classes= np.unique(y_data),y=y_data)
class_wght = {i:wght for i,wght in enumerate(class_wght)}
class_wght
```

Out[16]:

```
{0: 3.3021400072542617, 1: 0.5892175263736975}
```

In [17]:

```
##pad sequences to have equal number of features
### school state
max_length = x_train['school_state'].apply(lambda x : len(x)).max()
X_train_school_state = pad_sequences(x_train['school_state'], maxlen=max_length)
```

```
X_test_school_state = pad_sequences(x_test['school_state'], maxlen=max_length)
X_cv_school_state = pad_sequences(x_cv['school_state'], maxlen=max_length)
print(X_train_school_state[55])

### teacher prefix
max_length = x_train['teacher_prefix'].apply(lambda x : len(x)).max()
X_train_tpr = pad_sequences(x_train['teacher_prefix'], maxlen=max_length)
X_test_tpr = pad_sequences(x_test['teacher_prefix'], maxlen=max_length)
X_cv_tpr = pad_sequences(x_cv['teacher_prefix'], maxlen=max_length)
print(X_train_tpr[55])

### proejct grade category
max_length = x_train['project_grade_category'].apply(lambda x : len(x)).max()
X_train_pgc = pad_sequences(x_train['project_grade_category'], maxlen=max_length)
X_test_pgc = pad_sequences(x_test['project_grade_category'], maxlen=max_length)
X_cv_pgc = pad_sequences(x_cv['project_grade_category'], maxlen=max_length)
print(X_train_pgc[55])

### clean_categories
max_length = x_train['clean_categories'].apply(lambda x : len(x)).max()
X_train_cc = pad_sequences(x_train['clean_categories'], maxlen=max_length)
X_test_cc = pad_sequences(x_test['clean_categories'], maxlen=max_length)
X_cv_cc = pad_sequences(x_cv['clean_categories'], maxlen=max_length)
print(X_train_cc[55])

### clean_subcategories
max_length = x_train['clean_subcategories'].apply(lambda x : len(x)).max()
X_train_csc = pad_sequences(x_train['clean_subcategories'], maxlen=max_length)
X_test_csc = pad_sequences(x_test['clean_subcategories'], maxlen=max_length)
X_cv_csc = pad_sequences(x_cv['clean_subcategories'], maxlen=max_length)
print(X_train_csc[55])
```

```
[9]
[1]
[1 8 9]
[0 0 0 3 4]
[ 0  0  7  5 14]
```

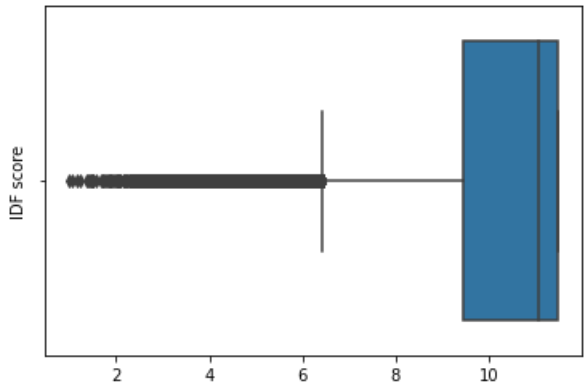
Select best features based on IDF values

In [18]:

```
tfidf = TfidfVectorizer()
tfidf.fit(x_train['essay'])
sns.boxplot(tfidf.idf_)
plt.ylabel("IDF score")
plt.plot()
```

Out[18]:

```
[]
```



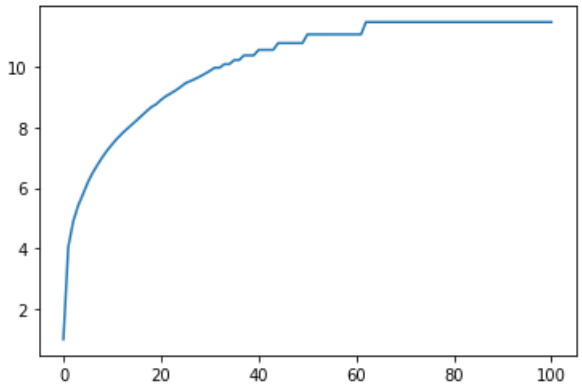
In [19]:

```
### CHECK THE PERCENTILE VALUE
idf_val = np.zeros(101)
for j,i in enumerate(range(0,101,1)):
```

```
    idf_val[j] = np.percentile(tfidf.idf_,i)
    #print("{}th percentile of idf {}".format(i,idf_val[j]))

plt.plot(idf_val)
```

Out[19]:

```
[<matplotlib.lines.Line2D at 0x7f919c9e6c50>]
```



In [20]:

```
### CHECK THE PERCENTILE VALUE
idf_val = np.zeros(101)
for j,i in enumerate(range(0,101,1)):
    idf_val[j] = np.percentile(tfidf.idf_,i)
    print("{}th percentile of idf {}".format(i,idf_val[j]))
```

```
0th percentile of idf 1.007839686363297
1th percentile of idf 4.0522460444480295
2th percentile of idf 4.884612865560149
3th percentile of idf 5.40548976775337
4th percentile of idf 5.78608289950784
5th percentile of idf 6.175954098631188
6th percentile of idf 6.495610492408024
7th percentile of idf 6.761465161815284
8th percentile of idf 7.013429151664985
9th percentile of idf 7.238188719178192
10th percentile of idf 7.43659383687255
11th percentile of idf 7.622493215014389
12th percentile of idf 7.785644855700624
13th percentile of idf 7.935585002991538
14th percentile of idf 8.077555264261925
15th percentile of idf 8.223267075443319
16th percentile of idf 8.370903074249384
17th percentile of idf 8.51750654844126
18th percentile of idf 8.658585146701164
19th percentile of idf 8.75389532650549
20th percentile of idf 8.896996170146164
21th percentile of idf 9.019598492238496
22th percentile of idf 9.110570270444223
23th percentile of idf 9.210653729001205
24th percentile of idf 9.321879364111428
25th percentile of idf 9.447042507065435
26th percentile of idf 9.516035378552386
27th percentile of idf 9.590143350706109
28th percentile of idf 9.670186058379645
29th percentile of idf 9.757197435369275
30th percentile of idf 9.8525076151736
31th percentile of idf 9.957868130831425
32th percentile of idf 9.957868130831425
33th percentile of idf 10.075651166487809
34th percentile of idf 10.075651166487809
35th percentile of idf 10.209182559112332
36th percentile of idf 10.209182559112332
37th percentile of idf 10.36333323893959
38th percentile of idf 10.36333323893959
39th percentile of idf 10.36333323893959
40th percentile of idf 10.545654795733544
41th percentile of idf 10.545654795733544
```

```
42th percentile of idf 10.545654795733544
43th percentile of idf 10.545654795733544
44th percentile of idf 10.768798347047754
45th percentile of idf 10.768798347047754
46th percentile of idf 10.768798347047754
47th percentile of idf 10.768798347047754
48th percentile of idf 10.768798347047754
49th percentile of idf 10.768798347047754
50th percentile of idf 11.056480419499536
51th percentile of idf 11.056480419499536
52th percentile of idf 11.056480419499536
53th percentile of idf 11.056480419499536
54th percentile of idf 11.056480419499536
55th percentile of idf 11.056480419499536
56th percentile of idf 11.056480419499536
57th percentile of idf 11.056480419499536
58th percentile of idf 11.056480419499536
59th percentile of idf 11.056480419499536
60th percentile of idf 11.056480419499536
61th percentile of idf 11.056480419499536
62th percentile of idf 11.4619455276077
63th percentile of idf 11.4619455276077
64th percentile of idf 11.4619455276077
65th percentile of idf 11.4619455276077
66th percentile of idf 11.4619455276077
67th percentile of idf 11.4619455276077
68th percentile of idf 11.4619455276077
69th percentile of idf 11.4619455276077
70th percentile of idf 11.4619455276077
71th percentile of idf 11.4619455276077
72th percentile of idf 11.4619455276077
73th percentile of idf 11.4619455276077
74th percentile of idf 11.4619455276077
75th percentile of idf 11.4619455276077
76th percentile of idf 11.4619455276077
77th percentile of idf 11.4619455276077
78th percentile of idf 11.4619455276077
79th percentile of idf 11.4619455276077
80th percentile of idf 11.4619455276077
81th percentile of idf 11.4619455276077
82th percentile of idf 11.4619455276077
83th percentile of idf 11.4619455276077
84th percentile of idf 11.4619455276077
85th percentile of idf 11.4619455276077
86th percentile of idf 11.4619455276077
87th percentile of idf 11.4619455276077
88th percentile of idf 11.4619455276077
89th percentile of idf 11.4619455276077
90th percentile of idf 11.4619455276077
91th percentile of idf 11.4619455276077
92th percentile of idf 11.4619455276077
93th percentile of idf 11.4619455276077
94th percentile of idf 11.4619455276077
95th percentile of idf 11.4619455276077
96th percentile of idf 11.4619455276077
97th percentile of idf 11.4619455276077
98th percentile of idf 11.4619455276077
99th percentile of idf 11.4619455276077
100th percentile of idf 11.4619455276077
```

In [21]:

```
x_train['essay'].head()
```

Out[21]:

```
18589     our students attend trix performance academy k...
102121    our school population usually receiving 50 5 f...
40835     our students learning academic english communi...
10349     my students sweet children love learn they enj...
80988     we title i school southern california most stu...
Name: essay, dtype: object
```

remove all those data having less than 5.25 as it's idf value

In [22]:

```
### find words corresponding to these idf value
### below words are more frequent words in our vocab
vocab2index = tfidf.vocabulary_
index2vocab = {value:key for key,value in vocab2index.items()}
indices = [np.where(tfidf.idf_ <= 2) or np.where(tfidf.idf_ >= 11)]
low_tfidf_vocab = []
for index in indices[0][0]:
    low_tfidf_vocab.append(index2vocab[index])
```

In [23]:

```
from tqdm import tqdm_notebook as tqdm
def clean_essay_before_tokenize(x_data,low_tfidf_vocab):

    for i,data in tqdm(enumerate(x_data['essay'])):
        x_data['essay'].iloc[i] = ' '.join([d for d in data.split(' ') if d not in low_tfidf_vocab]
)
    return x_data
```

In [24]:

```
x_train = clean_essay_before_tokenize(x_train, low_tfidf_vocab)
```

In [25]:

```
x_cv = clean_essay_before_tokenize(x_cv, low_tfidf_vocab)
```

In [26]:

```
x_test = clean_essay_before_tokenize(x_test, low_tfidf_vocab)
```

In [27]:

```
### https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/
from tensorflow.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train["essay"].tolist())
text_seq_train = tokenizer.texts_to_sequences(x_train["essay"])
text_seq_cv = tokenizer.texts_to_sequences(x_cv["essay"])
text_seq_test = tokenizer.texts_to_sequences(x_test["essay"])
```

In [28]:

```
padded_text_train = pad_sequences(text_seq_train,maxlen=500,padding='post', truncating='post')
padded_text_test = pad_sequences(text_seq_test, maxlen=500,padding='post', truncating='post')
padded_text_cv = pad_sequences(text_seq_cv, maxlen=500,padding='post', truncating='post')
```

In [29]:

```
vocab_size = len(tokenizer.word_index) + 1
vocab_size
```

Out[29]:

```
47351
```

In [30]:

```
glove_vector_saved = open("/content/drive/My Drive/LSTM Assignment/glove_vectors","rb")
glove_words = pickle.load(glove_vector_saved)
```

In [31]:

```
embedding_mat = np.zeros((vocab_size, 300))
for word, i in tokenizer.word_index.items():
    embedding_vec = glove_words.get(word)
    if embedding_vec is not None:
        embedding_mat[i] = embedding_vec

print(embedding_mat.shape)
```

```
(47351, 300)
```

In [32]:

```
### Normalize your data
def norm_data(X_tr, X_cv, X_test, col_name = 'price'):

    normalizer = Normalizer()
    normalizer.fit(X_tr[col_name].values.reshape(1,-1))

    X_tr_norm = normalizer.transform(X_tr[col_name].values.reshape(-1,1))
    X_cv_norm = normalizer.transform(X_cv[col_name].values.reshape(-1,1))
    X_test_norm = normalizer.transform(X_test[col_name].values.reshape(-1,1))


    print("After vectorizations")
    print("Shape of training data {}" .format(X_tr_norm.shape))
    print("Shape of cross validation data {}".format(X_cv_norm.shape))
    print("Shape of test data {}".format(X_test_norm.shape))
    print("="*100)
    return X_tr_norm,X_cv_norm,X_test_norm
```

In [33]:

```
X_train_price_norm,X_cv_price_norm,X_test_price_norm = norm_data(x_train, x_cv, x_test, "price")
```

```
After vectorizations
Shape of training data (69918, 1)
Shape of cross validation data (17480, 1)
Shape of test data (21850, 1)
====================================================================================================
```

In [34]:

```
X_train_nopp_norm,X_cv_nopp_norm,X_test_nopp_norm = norm_data(x_train, x_cv, x_test, "teacher_numbe
r_of_previously_posted_projects")
```

```
After vectorizations
Shape of training data (69918, 1)
Shape of cross validation data (17480, 1)
Shape of test data (21850, 1)
====================================================================================================
```

In [35]:

```
train_numeric_feature = np.array([X_train_price_norm,X_train_nopp_norm]).reshape(-1,2)
cv_numeric_feature = np.array([X_cv_price_norm,X_cv_nopp_norm]).reshape(-1,2)
test_numeric_feature = np.array([X_test_price_norm,X_test_nopp_norm]).reshape(-1,2)
```

In [36]:

```
def auc_roc(y_true, y_pred):
    if len(np.unique(y_true[:,1])) == 1:
        return 0.5
```

```
    else:
        return roc_auc_score(y_true, y_pred)
def auc_roc_score(y_true, y_pred):
    return tf.py_function(auc_roc, (y_true, y_pred), tf.double)
```

In [37]:

```
input_text = Input(shape=(500,),name="input_text") ## dim of text input
embed_text_data = Embedding(input_dim = vocab_size,output_dim =
300,weights=[embedding_mat],trainable=False)(input_text)
lstm_out = LSTM(128,kernel_regularizer=regularizers.l2(0.001),return_sequences=True)
(embed_text_data)
flatted_text_out = Flatten()(lstm_out)
```

In [38]:

```
input_state = Input(shape=(1),name="input_state") ## dim of state input
embed_state_data = Embedding(school_state_size,2)(input_state)
flatted_state_out = Flatten()(embed_state_data)

input_pgc = Input(shape=(3),name="input_pgc") ## dim of project grade category input
embed_pgc_data = Embedding(pgc_size,2)(input_pgc)
flatted_pgc_out = Flatten()(embed_pgc_data)

input_tchr_pre = Input(shape=(1),name="input_tchr_pre")
embed_tpr_data = Embedding(tpr_size,2)(input_tchr_pre)
flatted_tpr_out = Flatten()(embed_tpr_data)

sizeof_clean_cat = len(X_train_cc[0])
input_clean_cat = Input(shape=(sizeof_clean_cat),name="input_clean_cat")
embed_cc_data = Embedding(cc_size,2)(input_clean_cat)
flatted_cc_out = Flatten()(embed_cc_data)

sizeof_sub_clean_cat = len(X_train_csc[0])
input_clean_sub_cat = Input(shape=(sizeof_sub_clean_cat),name="input_clean_sub_cat")
embed_csc_data = Embedding(csc_size,2)(input_clean_sub_cat)
flatted_csc_out = Flatten()(embed_csc_data)
```

In [39]:

```
numerical_in = Input(shape=(2,),name="numerical_features")
numerical_dense_out = Dense(100,activation="relu",kernel_initializer="he_normal",kernel_regularizer
=regularizers.l2(0.001))(numerical_in)
```

In [56]:

```
concat_out =
concatenate([flatted_text_out,flatted_state_out,flatted_pgc_out,flatted_tpr_out,flatted_cc_out,fla
tted_csc_out,numerical_dense_out])
x = Dense(256,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.001))(concat
_out)
#x = Dropout(0.5)(x)
x = Dense(128,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.001))(x)
#x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Dense(128,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.001))(x)
x = Dropout(0.3)(x)
x = Dense(64,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.001))(x)
x = Dropout(0.3)(x)
x = Dense(64,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.001))(x)
x = Dropout(0.3)(x)
x = BatchNormalization()(x)
x = Dense(32,activation='relu',kernel_initializer=he_normal(),kernel_regularizer=l2(0.001))(x)
x = Dropout(0.3)(x)
output = Dense(2, activation = 'softmax')(x)
```

In [57]:

```
model = Model([input_text,input_state,input_pgc,input_tchr_pre,input_clean_cat,input_clean_sub_cat
,numerical_in], output)
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.0006,decay = 1e-
4),metrics=[auc_roc_score])
```

```
print(model.summary())
```

```
Model: "model_1"
```

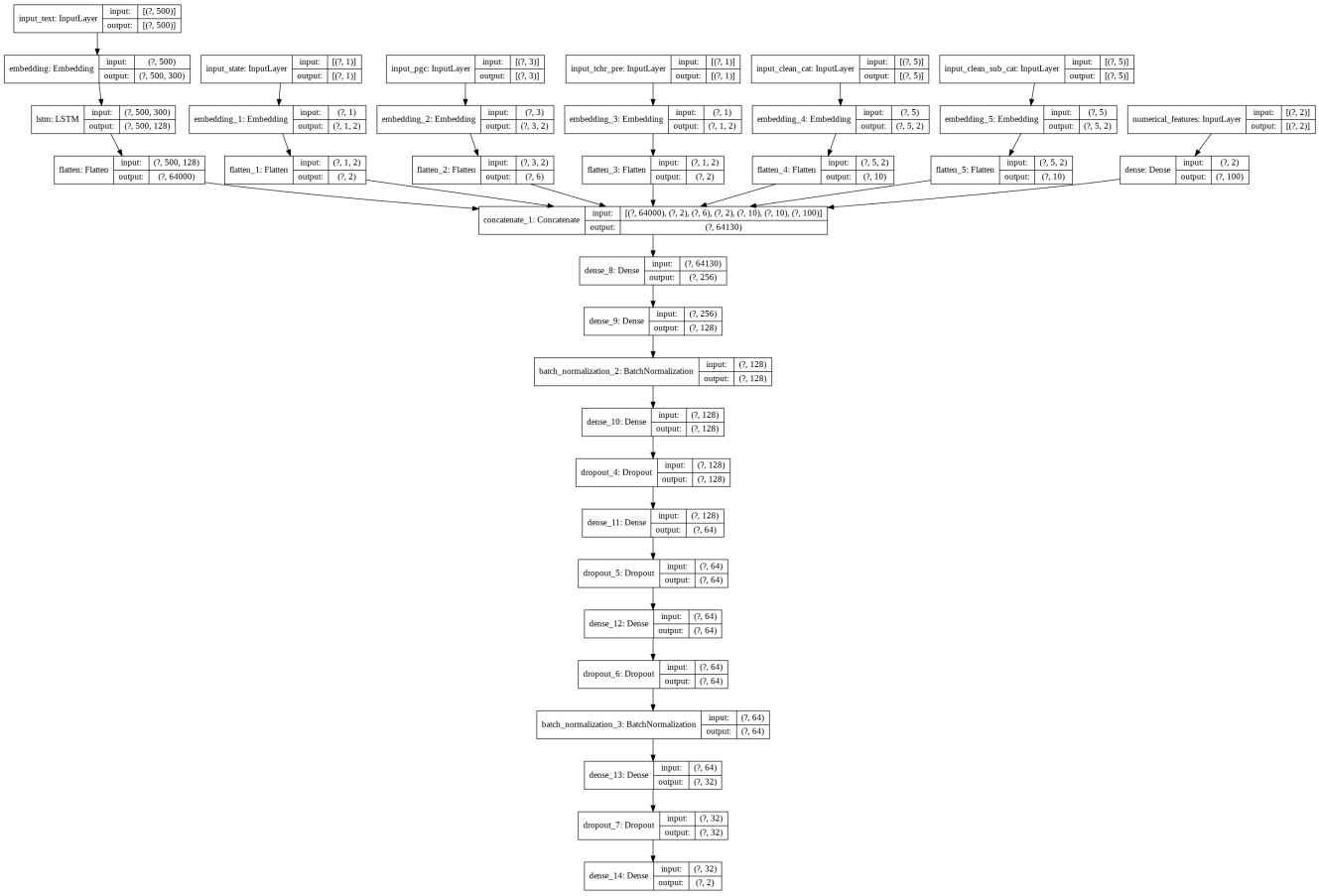
Layer (type)	Output Shape	Param #	Connected to
input_text (InputLayer)	[(None, 500)]	0	
embedding (Embedding)	(None, 500, 300)	14205300	input_text[0][0]
input_state (InputLayer)	[(None, 1)]	0	
input_pgc (InputLayer)	[(None, 3)]	0	
input_tchr_pre (InputLayer)	[(None, 1)]	0	
input_clean_cat (InputLayer)	[(None, 5)]	0	
input_clean_sub_cat (InputLayer	[(None, 5)]	0	
lstm (LSTM)	(None, 500, 128)	219648	embedding[0][0]
embedding_1 (Embedding)	(None, 1, 2)	104	input_state[0][0]
embedding_2 (Embedding)	(None, 3, 2)	20	input_pgc[0][0]
embedding_3 (Embedding)	(None, 1, 2)	12	input_tchr_pre[0][0]
embedding_4 (Embedding)	(None, 5, 2)	32	input_clean_cat[0][0]
embedding_5 (Embedding)	(None, 5, 2)	76	input_clean_sub_cat[0][0]
numerical_features (InputLayer)	[(None, 2)]	0	
flatten (Flatten)	(None, 64000)	0	lstm[0][0]
flatten_1 (Flatten)	(None, 2)	0	embedding_1[0][0]
flatten_2 (Flatten)	(None, 6)	0	embedding_2[0][0]
flatten_3 (Flatten)	(None, 2)	0	embedding_3[0][0]
flatten_4 (Flatten)	(None, 10)	0	embedding_4[0][0]
flatten_5 (Flatten)	(None, 10)	0	embedding_5[0][0]
dense (Dense)	(None, 100)	300	numerical_features[0][0]
concatenate_1 (Concatenate)	(None, 64130)	0	flatten[0][0] flatten_1[0][0] flatten_2[0][0] flatten_3[0][0] flatten_4[0][0] flatten_5[0][0] dense[0][0]
dense_8 (Dense)	(None, 256)	16417536	concatenate_1[0][0]
dense_9 (Dense)	(None, 128)	32896	dense_8[0][0]
batch_normalization_2 (BatchNor	(None, 128)	512	dense_9[0][0]
dense_10 (Dense)	(None, 128)	16512	batch_normalization_2[0][0]
dropout_4 (Dropout)	(None, 128)	0	dense_10[0][0]
dense_11 (Dense)	(None, 64)	8256	dropout_4[0][0]
dropout_5 (Dropout)	(None, 64)	0	dense_11[0][0]
dense_12 (Dense)	(None, 64)	4160	dropout_5[0][0]
dropout_6 (Dropout)	(None, 64)	0	dense_12[0][0]
batch_normalization_3 (BatchNor	(None, 64)	256	dropout_6[0][0]
dense_13 (Dense)	(None, 32)	2080	batch_normalization_3[0][0]

```
dense_13 (Dense)                (None, 32)           2080        batch_normalization_3[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 32)           0           dense_13[0][0]
__________________________________________________________________________________________________
dense_14 (Dense)                (None, 2)            66          dropout_7[0][0]
==================================================================================================
Total params: 30,907,766
Trainable params: 16,702,082
Non-trainable params: 14,205,684
__________________________________________________________________________________________________
None
```

In [58]:

```
# summarize the model
from tensorflow.keras.utils import plot_model

plot_model(model, 'model.png', show_shapes=True)
```

Out[58]:



In [59]:

```
train_data =
[padded_text_train,X_train_school_state,X_train_pgc,X_train_tpr,X_train_cc,X_train_csc,train_numeri
c_feature]
cv_data = [padded_text_cv,X_cv_school_state,X_cv_pgc,X_cv_tpr,X_cv_cc,X_cv_csc,cv_numeric_feature]
test_data =
[padded_text_test,X_test_school_state,X_test_pgc,X_test_tpr,X_test_cc,X_test_csc,test_numeric_featu
re]
```

In [60]:

```
# tensor-board in colab
# Refer: https://www.tensorflow.org/tensorboard/get_started
import os
import datetime

! rm -rf ./logs/
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
```

```
print(logdir)
```

```
logs/20200715-064011
```

In [61]:

```
%load_ext tensorboard
%tensorboard --logdir $logdir
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [62]:

```
class LearningRateScheduler(tf.keras.callbacks.Callback):

    def __init__(self):

        self.lr = None
        self.val_auc = None
        self.prev_val_auc = 1e10
        self.epoch_cnt = 0

    def on_epoch_end(self, epoch, logs={}):

        # Get the current learning rate from model's optimizer.
        self.lr = float(tf.keras.backend.get_value(self.model.optimizer.lr))
        self.val_auc = float(logs.get('val_auc_roc_score'))
        self.epoch_cnt += 1

        print('Validation Accuracy is {}'.format(self.val_auc))
        scheduled_lr = self.lr
        if self.val_auc < self.prev_val_auc:
            # Set the value back to the optimizer before this epoch starts
            scheduled_lr = 0.1 * scheduled_lr
            tf.keras.backend.set_value(self.model.optimizer.lr, scheduled_lr)

        print('Optimized Learning Rate is {}'.format(scheduled_lr))
        self.prev_val_auc = self.val_auc
```

In [63]:

```
learning_rate = LearningRateScheduler()
```

In [64]:

```
#model fitting
#https://machinelearningmastery.com/check-point-deep-learning-models-keras/
tensorboard_callback = TensorBoard(logdir, histogram_freq=1)
filepath="weights_copy.best.hdf5"
checkpoint_callback = ModelCheckpoint(filepath, monitor='val_auc_roc_score', verbose=1, save_best_o
nly=True, mode='max')
callbacks_list = [checkpoint_callback,tensorboard_callback,learning_rate]
```

In [65]:

```
model.fit(train_data, y_train,epochs=30,verbose=1,batch_size=256,
          validation_data=(cv_data,y_cv),callbacks =callbacks_list,class_weight = class_wght )
```

```
Epoch 1/30
274/274 [==============================] - ETA: 0s - loss: 2.0092 - auc_roc_score: 0.5681
Epoch 00001: val_auc_roc_score improved from -inf to 0.66707, saving model to
weights_copy.best.hdf5
Validation Accuracy is 0.6670708060264587
Optimized Learning Rate is 6.000000284984708e-05
274/274 [==============================] - 44s 160ms/step - loss: 2.0092 - auc_roc_score: 0.5681 -
val_loss: 1.7549 - val_auc_roc_score: 0.6671
Epoch 2/30
273/274 [============================>.] - ETA: 0s - loss: 1.6968 - auc_roc_score: 0.6222
Epoch 00002: val_auc_roc_score improved from 0.66707 to 0.69621, saving model to
```

```
Epoch 00002: val_auc_roc_score improved from 0.69307 to 0.69621, saving model to
weights_copy.best.hdf5
Validation Accuracy is 0.6962051391601562
Optimized Learning Rate is 6.000000212225132e-05
274/274 [==============================] - 42s 154ms/step - loss: 1.6968 - auc_roc_score: 0.6221 -
val_loss: 1.6980 - val_auc_roc_score: 0.6962
Epoch 3/30
274/274 [==============================] - ETA: 0s - loss: 1.6410 - auc_roc_score: 0.6452
Epoch 00003: val_auc_roc_score improved from 0.69621 to 0.70267, saving model to
weights_copy.best.hdf5
Validation Accuracy is 0.702670693397522
Optimized Learning Rate is 6.000000212225132e-05
274/274 [==============================] - 42s 155ms/step - loss: 1.6410 - auc_roc_score: 0.6452 -
val_loss: 1.5858 - val_auc_roc_score: 0.7027
Epoch 4/30
273/274 [============================>.] - ETA: 0s - loss: 1.5858 - auc_roc_score: 0.6607
Epoch 00004: val_auc_roc_score improved from 0.70267 to 0.70618, saving model to
weights_copy.best.hdf5
Validation Accuracy is 0.7061773538589478
Optimized Learning Rate is 6.000000212225132e-05
274/274 [==============================] - 42s 155ms/step - loss: 1.5858 - auc_roc_score: 0.6612 -
val_loss: 1.5469 - val_auc_roc_score: 0.7062
Epoch 5/30
273/274 [============================>.] - ETA: 0s - loss: 1.5385 - auc_roc_score: 0.6692
Epoch 00005: val_auc_roc_score did not improve from 0.70618
Validation Accuracy is 0.7059831023216248
Optimized Learning Rate is 6.000000212225132e-06
274/274 [==============================] - 42s 152ms/step - loss: 1.5385 - auc_roc_score: 0.6690 -
val_loss: 1.5013 - val_auc_roc_score: 0.7060
Epoch 6/30
273/274 [============================>.] - ETA: 0s - loss: 1.5113 - auc_roc_score: 0.6769
Epoch 00006: val_auc_roc_score improved from 0.70618 to 0.70963, saving model to
weights_copy.best.hdf5
Validation Accuracy is 0.709634006023407
Optimized Learning Rate is 6.000000212225132e-06
274/274 [==============================] - 42s 155ms/step - loss: 1.5113 - auc_roc_score: 0.6768 -
val_loss: 1.5120 - val_auc_roc_score: 0.7096
Epoch 7/30
273/274 [============================>.] - ETA: 0s - loss: 1.5085 - auc_roc_score: 0.6760
Epoch 00007: val_auc_roc_score improved from 0.70963 to 0.71060, saving model to
weights_copy.best.hdf5
Validation Accuracy is 0.7106037139892578
Optimized Learning Rate is 6.000000212225132e-06
274/274 [==============================] - 42s 155ms/step - loss: 1.5086 - auc_roc_score: 0.6761 -
val_loss: 1.5021 - val_auc_roc_score: 0.7106
Epoch 8/30
273/274 [============================>.] - ETA: 0s - loss: 1.5010 - auc_roc_score: 0.6794
Epoch 00008: val_auc_roc_score improved from 0.71060 to 0.71104, saving model to
weights_copy.best.hdf5
Validation Accuracy is 0.711039125919342
Optimized Learning Rate is 6.000000212225132e-06
274/274 [==============================] - 42s 154ms/step - loss: 1.5010 - auc_roc_score: 0.6797 -
val_loss: 1.4965 - val_auc_roc_score: 0.7110
Epoch 9/30
273/274 [============================>.] - ETA: 0s - loss: 1.4947 - auc_roc_score: 0.6794
Epoch 00009: val_auc_roc_score improved from 0.71104 to 0.71192, saving model to
weights_copy.best.hdf5
Validation Accuracy is 0.7119191288948059
Optimized Learning Rate is 6.000000212225132e-06
274/274 [==============================] - 42s 154ms/step - loss: 1.4946 - auc_roc_score: 0.6794 -
val_loss: 1.4945 - val_auc_roc_score: 0.7119
Epoch 10/30
273/274 [============================>.] - ETA: 0s - loss: 1.4877 - auc_roc_score: 0.6808
Epoch 00010: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7111571431159973
Optimized Learning Rate is 6.000000212225132e-07
274/274 [==============================] - 42s 152ms/step - loss: 1.4878 - auc_roc_score: 0.6806 -
val_loss: 1.4789 - val_auc_roc_score: 0.7112
Epoch 11/30
273/274 [============================>.] - ETA: 0s - loss: 1.4840 - auc_roc_score: 0.6809
Epoch 00011: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7107850313186646
Optimized Learning Rate is 6.000000212225131e-08
274/274 [==============================] - 42s 152ms/step - loss: 1.4840 - auc_roc_score: 0.6800 -
val_loss: 1.4786 - val_auc_roc_score: 0.7108
Epoch 12/30
273/274 [============================>.] - ETA: 0s - loss: 1.4835 - auc_roc_score: 0.6826
Epoch 00012: val_auc_roc_score did not improve from 0.71192
```

```
Epoch 00012: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7106549143791199
Optimized Learning Rate is 6.000000496442226e-09
274/274 [==============================] - 42s 152ms/step - loss: 1.4836 - auc_roc_score: 0.6825 -
val_loss: 1.4788 - val_auc_roc_score: 0.7107
Epoch 13/30
273/274 [============================>.] - ETA: 0s - loss: 1.4836 - auc_roc_score: 0.6834
Epoch 00013: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7106902003288269
Optimized Learning Rate is 6.000000496442226e-09
274/274 [==============================] - 42s 152ms/step - loss: 1.4836 - auc_roc_score: 0.6834 -
val_loss: 1.4780 - val_auc_roc_score: 0.7107
Epoch 14/30
273/274 [============================>.] - ETA: 0s - loss: 1.4842 - auc_roc_score: 0.6814
Epoch 00014: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7109714150428772
Optimized Learning Rate is 6.000000496442226e-09
274/274 [==============================] - 42s 152ms/step - loss: 1.4843 - auc_roc_score: 0.6815 -
val_loss: 1.4798 - val_auc_roc_score: 0.7110
Epoch 15/30
273/274 [============================>.] - ETA: 0s - loss: 1.4839 - auc_roc_score: 0.6841
Epoch 00015: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7109315991401672
Optimized Learning Rate is 6.000000496442226e-10
274/274 [==============================] - 42s 152ms/step - loss: 1.4838 - auc_roc_score: 0.6851 -
val_loss: 1.4779 - val_auc_roc_score: 0.7109
Epoch 16/30
273/274 [============================>.] - ETA: 0s - loss: 1.4853 - auc_roc_score: 0.6833
Epoch 00016: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7105202674865723
Optimized Learning Rate is 6.000000496442226e-11
274/274 [==============================] - 42s 152ms/step - loss: 1.4853 - auc_roc_score: 0.6832 -
val_loss: 1.4776 - val_auc_roc_score: 0.7105
Epoch 17/30
273/274 [============================>.] - ETA: 0s - loss: 1.4854 - auc_roc_score: 0.6814
Epoch 00017: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7109456062316895
Optimized Learning Rate is 6.000000496442226e-11
274/274 [==============================] - 42s 152ms/step - loss: 1.4853 - auc_roc_score: 0.6816 -
val_loss: 1.4787 - val_auc_roc_score: 0.7109
Epoch 18/30
273/274 [============================>.] - ETA: 0s - loss: 1.4854 - auc_roc_score: 0.6832
Epoch 00018: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7106720209121704
Optimized Learning Rate is 6.0000004964422265e-12
274/274 [==============================] - 42s 152ms/step - loss: 1.4854 - auc_roc_score: 0.6839 -
val_loss: 1.4770 - val_auc_roc_score: 0.7107
Epoch 19/30
273/274 [============================>.] - ETA: 0s - loss: 1.4857 - auc_roc_score: 0.6813
Epoch 00019: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7104616761207581
Optimized Learning Rate is 6.000000409706053e-13
274/274 [==============================] - 42s 152ms/step - loss: 1.4858 - auc_roc_score: 0.6812 -
val_loss: 1.4776 - val_auc_roc_score: 0.7105
Epoch 20/30
273/274 [============================>.] - ETA: 0s - loss: 1.4833 - auc_roc_score: 0.6820
Epoch 00020: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7106585502624512
Optimized Learning Rate is 6.000000301285835e-13
274/274 [==============================] - 42s 152ms/step - loss: 1.4833 - auc_roc_score: 0.6818 -
val_loss: 1.4767 - val_auc_roc_score: 0.7107
Epoch 21/30
273/274 [============================>.] - ETA: 0s - loss: 1.4817 - auc_roc_score: 0.6844
Epoch 00021: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7107952237129211
Optimized Learning Rate is 6.000000301285835e-13
274/274 [==============================] - 42s 152ms/step - loss: 1.4817 - auc_roc_score: 0.6842 -
val_loss: 1.4766 - val_auc_roc_score: 0.7108
Epoch 22/30
273/274 [============================>.] - ETA: 0s - loss: 1.4832 - auc_roc_score: 0.6836
Epoch 00022: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7106277346611023
Optimized Learning Rate is 6.000000301285835e-14
274/274 [==============================] - 42s 152ms/step - loss: 1.4831 - auc_roc_score: 0.6846 -
val_loss: 1.4803 - val_auc_roc_score: 0.7106
Epoch 23/30
273/274 [============================>.] - ETA: 0s - loss: 1.4837 - auc_roc_score: 0.6826
Epoch 00023: val_auc_roc_score did not improve from 0.71192
```

```
Epoch 00025: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7109218835830688
Optimized Learning Rate is 6.000000572336378e-14
274/274 [==============================] - 42s 152ms/step - loss: 1.4836 - auc_roc_score: 0.6830 -
val_loss: 1.4792 - val_auc_roc_score: 0.7109
Epoch 24/30
273/274 [============================>.] - ETA: 0s - loss: 1.4855 - auc_roc_score: 0.6810
Epoch 00024: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7107262611389l6
Optimized Learning Rate is 6.000000572336378e-15
274/274 [==============================] - 42s 151ms/step - loss: 1.4854 - auc_roc_score: 0.6817 -
val_loss: 1.4773 - val_auc_roc_score: 0.7107
Epoch 25/30
273/274 [============================>.] - ETA: 0s - loss: 1.4830 - auc_roc_score: 0.6849
Epoch 00025: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7107534408569336
Optimized Learning Rate is 6.000000657039673e-15
274/274 [==============================] - 42s 152ms/step - loss: 1.4830 - auc_roc_score: 0.6843 -
val_loss: 1.4788 - val_auc_roc_score: 0.7108
Epoch 26/30
273/274 [============================>.] - ETA: 0s - loss: 1.4864 - auc_roc_score: 0.6786
Epoch 00026: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7106808423995972
Optimized Learning Rate is 6.000000657039673e-16
274/274 [==============================] - 42s 152ms/step - loss: 1.4863 - auc_roc_score: 0.6790 -
val_loss: 1.4784 - val_auc_roc_score: 0.7107
Epoch 27/30
273/274 [============================>.] - ETA: 0s - loss: 1.4842 - auc_roc_score: 0.6822
Epoch 00027: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7110958695411682
Optimized Learning Rate is 6.000000762918791e-16
274/274 [==============================] - 42s 152ms/step - loss: 1.4842 - auc_roc_score: 0.6810 -
val_loss: 1.4782 - val_auc_roc_score: 0.7111
Epoch 28/30
273/274 [============================>.] - ETA: 0s - loss: 1.4837 - auc_roc_score: 0.6843
Epoch 00028: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7108835577964783
Optimized Learning Rate is 6.000000762918792e-17
274/274 [==============================] - 42s 152ms/step - loss: 1.4837 - auc_roc_score: 0.6839 -
val_loss: 1.4783 - val_auc_roc_score: 0.7109
Epoch 29/30
273/274 [============================>.] - ETA: 0s - loss: 1.4861 - auc_roc_score: 0.6830
Epoch 00029: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.71082603931427
Optimized Learning Rate is 6.000000895267689e-18
274/274 [==============================] - 42s 152ms/step - loss: 1.4862 - auc_roc_score: 0.6825 -
val_loss: 1.4789 - val_auc_roc_score: 0.7108
Epoch 30/30
273/274 [============================>.] - ETA: 0s - loss: 1.4835 - auc_roc_score: 0.6837
Epoch 00030: val_auc_roc_score did not improve from 0.71192
Validation Accuracy is 0.7108224034309387
Optimized Learning Rate is 6.000000895267689e-19
274/274 [==============================] - 42s 152ms/step - loss: 1.4836 - auc_roc_score: 0.6836 -
val_loss: 1.4780 - val_auc_roc_score: 0.7108
```

Out[65]:

```
<tensorflow.python.keras.callbacks.History at 0x7f919a3d2fd0>
```

In [66]:

```
### load the weight from the saved file
model.load_weights("weights_copy.best.hdf5")
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001,decay = 1e-
4),metrics=[auc_roc_score])
```

In [67]:

```
### plot AUC for test train-data
y_predict_test = model.predict(test_data)
y_predict_cv = model.predict(cv_data)
y_predict_train = model.predict(train_data)
```

In [68]:

```
print("ROC-AUC for test data: %0.3f"%roc_auc_score(y_test,y_predict_test))
print("ROC-AUC for CV data: %0.3f"%roc_auc_score(y_cv,y_predict_cv))
print("ROC-AUC for train data: %0.3f"%roc_auc_score(y_train,y_predict_train))
```

```
ROC-AUC for test data: 0.714
ROC-AUC for CV data: 0.713
ROC-AUC for train data: 0.721
```

In [69]:

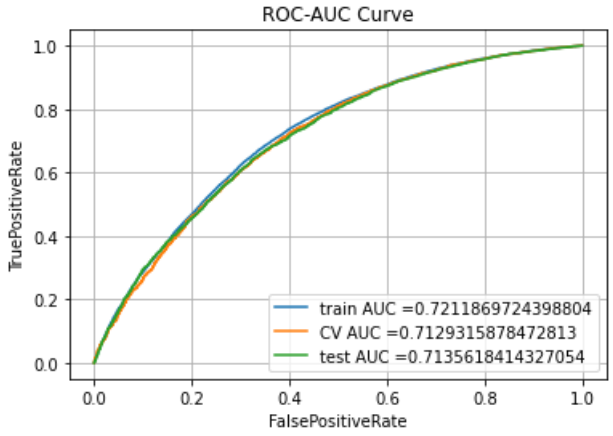
```
y_pred_tr = y_predict_train[:,1]
y_pred_cv = y_predict_cv[:,1]
y_pred_test = y_predict_test[:,1]
```

In [70]:

```
y_tr = np.where(y_train == 1)[1]
y_tst = np.where(y_test == 1)[1]
y_crs_val = np.where(y_cv == 1)[1]
```

In [71]:

```
from sklearn.metrics import roc_curve, auc
train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_pred_tr)
cv_fpr, cv_tpr, cv_thresholds = roc_curve(y_crs_val, y_pred_cv)
test_fpr, test_tpr, test_thresholds = roc_curve(y_tst, y_pred_test)
auc_tfidf_train = auc(train_fpr, train_tpr)
auc_tfidf_cv = auc(cv_fpr, cv_tpr)
auc_tfidf_test = auc(test_fpr, test_tpr)
### feature importance


plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc_tfidf_train))
plt.plot(cv_fpr, cv_tpr, label="CV AUC ="+str(auc_tfidf_cv))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc_tfidf_test))
plt.legend()
plt.xlabel("FalsePositiveRate")
plt.ylabel("TruePositiveRate")
plt.title("ROC-AUC Curve")
plt.grid()
plt.show()
```



In [55]: