In [1]:

```
# Preprocessed Data
!gdown --id "1Lz1IHzyhCoWTsDuaD9Kf7OyoLK4XdFTr"
# Glove Vectors
!gdown --id "1Z6bjXmyCaoEzXYo_tRDwLTsfeA2F3K3j"
# train_data.csv
!gdown --id "1T48h84GLW3dpy9F6ble5nF_1gQxBO8rx"
# test_data.csv
!gdown --id "1sh4p_gNyiD_tMVdMTd6F8fkJS7ysJFXK"
```

```
Downloading...
From: https://drive.google.com/uc?id=1Lz1IHzyhCoWTsDuaD9Kf7OyoLK4XdFTr
To: /content/preprocessed_data.csv
124MB [00:01, 91.1MB/s]
Downloading...
From: https://drive.google.com/uc?id=1Z6bjXmyCaoEzXYo_tRDwLTsfeA2F3K3j
To: /content/glove_vectors
128MB [00:01, 120MB/s]
Downloading...
From: https://drive.google.com/uc?id=1T48h84GLW3dpy9F6ble5nF_1gQxBO8rx
To: /content/train_data.csv
201MB [00:01, 111MB/s]
Downloading...
From: https://drive.google.com/uc?id=1sh4p_gNyiD_tMVdMTd6F8fkJS7ysJFXK
To: /content/test_data.csv
133MB [00:01, 85.0MB/s]
```

In [326]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
import re
import os
import pickle
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import re


from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)

import tensorflow as tf
import tensorflow.keras as keras
from sklearn.preprocessing import OneHotEncoder,LabelEncoder
from sklearn.model_selection import train_test_split

# from tensorflow.keras import utils
from keras.utils import np_utils
from keras import metrics

from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Input, Dense, Flatten, Embedding, Concatenate, Conv1D, MaxPooling1D, Dropout, LSTM
from tensorflow.keras.models import Sequential

from keras.utils.vis_utils import plot_model
from keras.utils import to_categorical
from sklearn.metrics import roc_auc_score, roc_curve
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.preprocessing import Normalizer

import datetime


import nltk
from nltk import ne_chunk, pos_tag, word_tokenize
from nltk.tree import Tree
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')

warnings.filterwarnings("ignore")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!
```

In [305]:

```
tf.keras.backend.clear_session()
```

In [306]:

```
%load_ext tensorboard
```

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

Load Dataset

In [307]:

```
data = pd.read_csv("/content/preprocessed_data.csv")
data.columns
```

Out[307]:

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

Model 1

Create Embedding Matrix for LSTM Input

In [308]:

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


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
2, stratify=y)
y_train = to_categorical(y_train,2)
y_test = to_categorical(y_test,2)
```

In [309]:

```
#use tf.tokenizer : remove '_' from filters as we need words joined by _ (new_york)
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}_~\t\n')
tokenizer.fit_on_texts(X_train["essay"])

# Encode training data sentences into sequences
train_sequences = tokenizer.texts_to_sequences(X_train["essay"])
test_sequences = tokenizer.texts_to_sequences(X_test["essay"])
```

In [310]:

```
vocab_size = len(tokenizer.word_index) + 1
print("Learned Vocab has size : ",vocab_size)
maxlen = max([len(x) for x in train_sequences])
print("Maximum len of words in train_data is: ", maxlen)
```

```
Learned Vocab has size :  51625
Maximum len of words in train_data is:  339
```

In [311]:

```
# Pad the sequences based on maxLen
train_padded = pad_sequences(train_sequences, padding='post', truncating='post', maxlen=maxlen)
test_padded = pad_sequences(test_sequences, padding='post', truncating='post', maxlen=maxlen)
```

In [312]:

```
with open('glove_vectors', 'rb') as f:
  glove_vector = pickle.load(f)
  glove_words = set(glove_vector.keys())
```

In [313]:

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

In [314]:

```
print("Shape of Embedding Matrix: %d x %d"%(len(embedding_matrix),len(embedding_matrix[0])))
```

```
Shape of Embedding Matrix: 51625 x 300
```

Tokenizing other inputs

Input_school_state

In [315]:

```
#use tf.tokenizer : remove '_' from filters as we need words joined by _ (new_york)
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}_~\t\n')
tokenizer.fit_on_texts(X_train["school_state"])

# Encode training data sentences into sequences
train_sequences_school_state = tokenizer.texts_to_sequences(X_train["school_state"])
test_sequences_school_state = tokenizer.texts_to_sequences(X_test["school_state"])

# all other data is of np.ndarray form, this is a list
train_sequences_school_state = np.array(train_sequences_school_state)
test_sequences_school_state = np.array(test_sequences_school_state)

vocab_size_school_state = len(tokenizer.word_index) + 1
maxlen_school_state = max([len(x) for x in train_sequences_school_state])
```

project_grade_category

In [316]:

```
#use tf.tokenizer : remove '-' from filters as we need words 5-6
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,./:;<=>?@[\\]^`{|}_~\t\n')
tokenizer.fit_on_texts(X_train["project_grade_category"])

# Encode training data sentences into sequences
train_sequences_grade = tokenizer.texts_to_sequences(X_train["project_grade_category"])
test_sequences_grade = tokenizer.texts_to_sequences(X_test["project_grade_category"])

# all other data is of np.ndarray form, this is a list
train_sequences_grade = np.array(train_sequences_grade)
test_sequences_grade = np.array(test_sequences_grade)

vocab_size_grade = len(tokenizer.word_index) + 1
maxlen_grade = max([len(x) for x in train_sequences_grade])

len(train_sequences_grade)
```

Out[316]:

```
87398
```

clean_categories

In [317]:

```
#use tf.tokenizer : remove '-' from filters as we need words 5-6
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,./:;<=>?@[\\]^`{|}_~\t\n')
tokenizer.fit_on_texts(X_train["clean_categories"])

# Encode training data sentences into sequences
train_sequences_clean_cats = tokenizer.texts_to_sequences(X_train["clean_categories"])
test_sequences_clean_cats = tokenizer.texts_to_sequences(X_test["clean_categories"])

vocab_size_clean_cats = len(tokenizer.word_index) + 1
maxlen_clean_cats = max([len(x) for x in train_sequences_clean_cats])
print(len(train_sequences_clean_cats))

# Pad the sequences based on maxLen
train_sequences_clean_cats = pad_sequences(train_sequences_clean_cats, padding='post', truncating='post', maxlen=maxlen_clean_cats)
test_sequences_clean_cats = pad_sequences(test_sequences_clean_cats, padding='post', truncating='post', maxlen=maxlen_clean_cats)
```

```
87398
```

clean_subcategories

In [318]:

```
#use tf.tokenizer : remove '-' from filters as we need words 5-6
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,./:;<=>?@[\\]^`{|}_~\t\n')
tokenizer.fit_on_texts(X_train["clean_subcategories"])

# Encode training data sentences into sequences
train_sequences_clean_subcats = tokenizer.texts_to_sequences(X_train["clean_subcategories"])
test_sequences_clean_subcats = tokenizer.texts_to_sequences(X_test["clean_subcategories"])

vocab_size_clean_subcats = len(tokenizer.word_index) + 1
maxlen_clean_subcats = max([len(x) for x in train_sequences_clean_subcats])

# Pad the sequences based on maxLen
train_sequences_clean_subcats = pad_sequences(train_sequences_clean_subcats, padding='post', truncating='post', maxlen=maxlen_clean_subcats)
test_sequences_clean_subcats = pad_sequences(test_sequences_clean_subcats, padding='post', truncating='post', maxlen=maxlen_clean_subcats)
```

teacher_prefix

In [319]:

```
#use tf.tokenizer : remove '-' from filters as we need words 5-6
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,./:;<=
>?@[\\]^`{|}_~\t\n')
tokenizer.fit_on_texts(X_train["teacher_prefix"])

# Encode training data sentences into sequences
train_sequences_teacher_prefix = tokenizer.texts_to_sequences(X_train[
"teacher_prefix"])
test_sequences_teacher_prefix = tokenizer.texts_to_sequences(X_test["t
eacher_prefix"])

vocab_size_teacher_prefix = len(tokenizer.word_index) + 1
maxlen_teacher_prefix = max([len(x) for x in train_sequences_teacher_p
refix])

# Pad the sequences based on maxLen
train_sequences_teacher_prefix = pad_sequences(train_sequences_teacher
_prefix, padding='post', truncating='post', maxlen=maxlen_teacher_pref
ix)
test_sequences_teacher_prefix = pad_sequences(test_sequences_teacher_p
refix, padding='post', truncating='post', maxlen=maxlen_teacher_prefix
)
```

Numerical Features

In [320]:

```
# price teacher_number_of_previously_posted_projects
train_numerical = X_train[["teacher_number_of_previously_posted_projec
ts","price"]].values
test_numerical = X_test[["teacher_number_of_previously_posted_project
s","price"]].values
print(train_numerical[0:2])
```

```
[[  0.    11.29]
 [  2.   803.63]]
```

In [321]:

```
maxlen_clean_subcats
```

Out[321]:

```
5
```

Model Building

In [322]:

```
Input_text = Input(shape=(maxlen,))
embed_1 = Embedding(vocab_size,300,input_length=maxlen, weights=[embed
ding_matrix], trainable=False)(Input_text)
lstm_1 = LSTM(units=128)(embed_1)
flatten_1 = Flatten()(lstm_1)

Input_school_state = Input(shape=(maxlen_school_state,))
embed_2 = Embedding(vocab_size_school_state,300,input_length=maxlen_sc
hool_state)(Input_school_state)
flatten_2 = Flatten()(embed_2)

Input_grade = Input(shape=(maxlen_grade,))
embed_3 = Embedding(vocab_size_grade,300,input_length=maxlen_grade)(In
put_grade)
flatten_3 = Flatten()(embed_3)

Input_clean_cats = Input(shape=(maxlen_clean_cats,))
embed_4 = Embedding(vocab_size_clean_cats,300,input_length=maxlen_clea
n_cats)(Input_clean_cats)
flatten_4 = Flatten()(embed_4)

Input_clean_subcats = Input(shape=(maxlen_clean_subcats,))
embed_5 = Embedding(vocab_size_clean_subcats,300,input_length=maxlen_c
lean_subcats)(Input_clean_subcats)
flatten_5 = Flatten()(embed_5)

Input_teacher_prefix = Input(shape=(maxlen_teacher_prefix,))
embed_6 = Embedding(vocab_size_teacher_prefix,300,input_length=maxlen_
teacher_prefix)(Input_teacher_prefix)
flatten_6 = Flatten()(embed_6)

Input_numerical = Input(shape=(2,))
dense_numerical = dense_2 = Dense(300,activation='relu')(Input_numeric
al)

concat = Concatenate()([flatten_1, flatten_2, flatten_3, flatten_4, fl
atten_5, flatten_6, dense_numerical])

dense_1 = Dense(512,activation='relu')(concat)
drop_1 = Dropout(0.1)(dense_1)
dense_2 = Dense(256,activation='relu')(drop_1)
drop_2 = Dropout(0.1)(dense_2)
dense_3 = Dense(64,activation='relu')(drop_2)
output = Dense(2, activation='softmax')(dense_3)
```

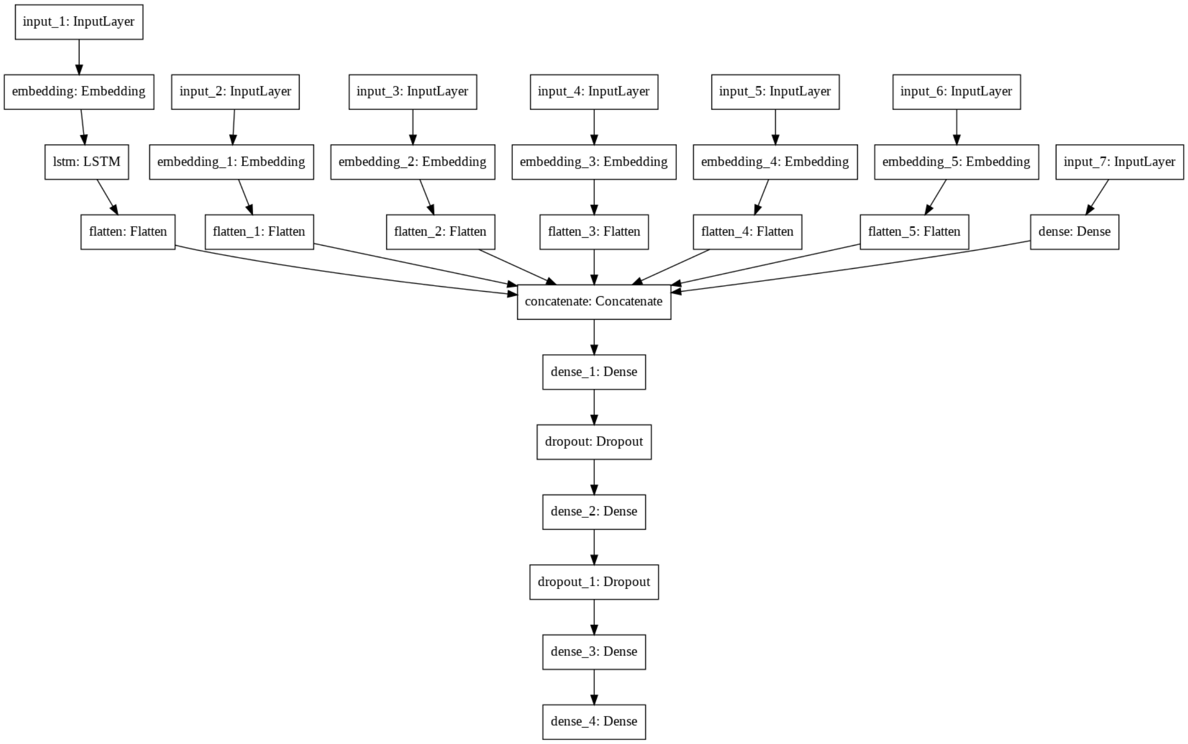
In [323]:

```
model = keras.Model([Input_text, Input_school_state, Input_grade, Inpu
t_clean_cats,Input_clean_subcats,Input_teacher_prefix, Input_numerical
],output)
```

In [324]:

```
plot_model(model)
```

Out[324]:



In [327]:

```
!rm -rf ./logs
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S"
)

# tensorboard Callback
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir,
    histogram_freq=1,
    write_graph=True)

#Model checkpoint callback
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath="models/LSTM_Model_1_{epoch:04d}.hdf5",
    verbose=1,
    save_weights_only=True,
    period=10)
```

```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_f
req` to specify the frequency in number of batches seen.
```

In [328]:

```
def auroc(y_true, y_pred):
  return tf.py_function(roc_auc_score, (y_true, y_pred), tf.double)


model.compile(optimizer= tf.keras.optimizers.Adam(learning_rate=0.0001
),
              loss="categorical_crossentropy",
              metrics=["accuracy", auroc])


train_data_inputs = [train_padded, train_sequences_school_state, train
_sequences_grade, train_sequences_clean_cats, train_sequences_clean_su
bcats, train_sequences_teacher_prefix, train_numerical]
test_data_inputs = [test_padded, test_sequences_school_state, test_seq
uences_grade, test_sequences_clean_cats, test_sequences_clean_subcats,
test_sequences_teacher_prefix, test_numerical]
```

In [329]:

```
model.fit(train_data_inputs,y_train,
          batch_size = 128,
          epochs=20,
          validation_data=(test_data_inputs,y_test),
          callbacks = [cp_callback, tensorboard_callback]
)
```

```
Epoch 1/20
683/683 [==============================] - 35s 48ms/step - loss: 0.538
5 - accuracy: 0.8253 - auroc: 0.5370 - val_loss: 0.5311 - val_accurac
y: 0.7912 - val_auroc: 0.6127
Epoch 2/20
683/683 [==============================] - 31s 46ms/step - loss: 0.469
5 - accuracy: 0.8412 - auroc: 0.5752 - val_loss: 0.4141 - val_accurac
y: 0.8486 - val_auroc: 0.6124
Epoch 3/20
683/683 [==============================] - 31s 46ms/step - loss: 0.444
1 - accuracy: 0.8438 - auroc: 0.5884 - val_loss: 0.4322 - val_accurac
y: 0.8486 - val_auroc: 0.5696
Epoch 4/20
683/683 [==============================] - 31s 46ms/step - loss: 0.426
6 - accuracy: 0.8460 - auroc: 0.6062 - val_loss: 0.4186 - val_accurac
y: 0.8468 - val_auroc: 0.6248
Epoch 5/20
683/683 [==============================] - 31s 46ms/step - loss: 0.420
5 - accuracy: 0.8469 - auroc: 0.6163 - val_loss: 0.4170 - val_accurac
y: 0.8469 - val_auroc: 0.6229
Epoch 6/20
683/683 [==============================] - 31s 46ms/step - loss: 0.418
6 - accuracy: 0.8480 - auroc: 0.6165 - val_loss: 0.4128 - val_accurac
y: 0.8486 - val_auroc: 0.6250
Epoch 7/20
683/683 [==============================] - 31s 45ms/step - loss: 0.420
8 - accuracy: 0.8460 - auroc: 0.6219 - val_loss: 0.4128 - val_accurac
y: 0.8486 - val_auroc: 0.6281
Epoch 8/20
683/683 [==============================] - 31s 46ms/step - loss: 0.417
5 - accuracy: 0.8487 - auroc: 0.6129 - val_loss: 0.4453 - val_accurac
y: 0.8298 - val_auroc: 0.6142
Epoch 9/20
683/683 [==============================] - 31s 46ms/step - loss: 0.415
7 - accuracy: 0.8486 - auroc: 0.6226 - val_loss: 0.4120 - val_accurac
y: 0.8485 - val_auroc: 0.6277
Epoch 10/20
683/683 [==============================] - 31s 45ms/step - loss: 0.413
6 - accuracy: 0.8472 - auroc: 0.6380 - val_loss: 0.3991 - val_accurac
y: 0.8486 - val_auroc: 0.6917

Epoch 00010: saving model to models/LSTM_Model_1_0010.hdf5
Epoch 11/20
683/683 [==============================] - 31s 45ms/step - loss: 0.394
4 - accuracy: 0.8471 - auroc: 0.7066 - val_loss: 0.3829 - val_accurac
y: 0.8486 - val_auroc: 0.7240
Epoch 12/20
683/683 [==============================] - 30s 45ms/step - loss: 0.380
8 - accuracy: 0.8497 - auroc: 0.7290 - val_loss: 0.3774 - val_accurac
y: 0.8486 - val_auroc: 0.7336
Epoch 13/20
683/683 [==============================] - 30s 44ms/step - loss: 0.375
7 - accuracy: 0.8497 - auroc: 0.7476 - val_loss: 0.3949 - val_accurac
y: 0.8374 - val_auroc: 0.7383
Epoch 14/20
683/683 [==============================] - 30s 45ms/step - loss: 0.370
5 - accuracy: 0.8516 - auroc: 0.7507 - val_loss: 0.3739 - val_accurac
```

```
y: 0.8518 - val_auroc: 0.7412
Epoch 15/20
683/683 [==============================] - 31s 45ms/step - loss: 0.368
6 - accuracy: 0.8522 - auroc: 0.7500 - val_loss: 0.3742 - val_accurac
y: 0.8515 - val_auroc: 0.7405
Epoch 16/20
683/683 [==============================] - 31s 45ms/step - loss: 0.365
7 - accuracy: 0.8546 - auroc: 0.7560 - val_loss: 0.3685 - val_accurac
y: 0.8541 - val_auroc: 0.7474
Epoch 17/20
683/683 [==============================] - 31s 45ms/step - loss: 0.360
0 - accuracy: 0.8546 - auroc: 0.7695 - val_loss: 0.3675 - val_accurac
y: 0.8536 - val_auroc: 0.7500
Epoch 18/20
683/683 [==============================] - 31s 45ms/step - loss: 0.362
5 - accuracy: 0.8540 - auroc: 0.7675 - val_loss: 0.3682 - val_accurac
y: 0.8520 - val_auroc: 0.7515
Epoch 19/20
683/683 [==============================] - 31s 45ms/step - loss: 0.356
6 - accuracy: 0.8553 - auroc: 0.7775 - val_loss: 0.3665 - val_accurac
y: 0.8542 - val_auroc: 0.7517
Epoch 20/20
683/683 [==============================] - 31s 45ms/step - loss: 0.356
0 - accuracy: 0.8561 - auroc: 0.7782 - val_loss: 0.3649 - val_accurac
y: 0.8534 - val_auroc: 0.7548

Epoch 00020: saving model to models/LSTM_Model_1_0020.hdf5
```

```
Out[329]: <tensorflow.python.keras.callbacks.History at 0x7fb70ea0aed0>
```

```
In [332]: %tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6006 (pid 5520), started 0:06:29 ago. (Use
'!kill 5520' to kill it.)
```

Model 2

Remove words based in IDF Score in essay

```
In [333]: y = data["project_is_approved"]
          X = data.drop("project_is_approved", axis=1)

          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
          2, stratify=y)
          y_train = to_categorical(y_train,2)
          y_test = to_categorical(y_test,2)
```

In [334]:

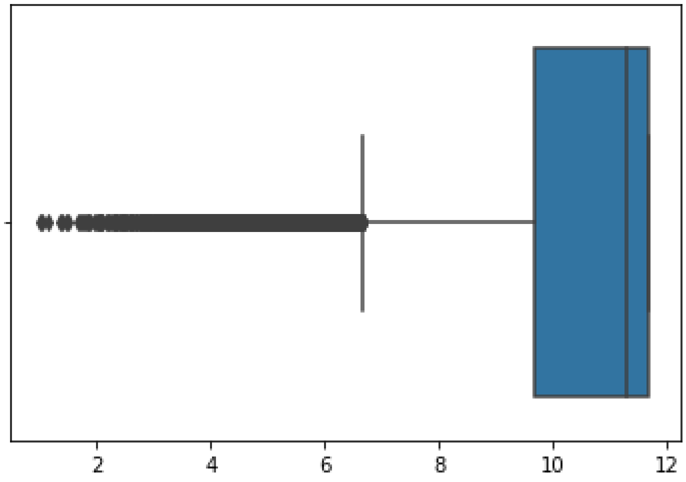
```
tfidf = TfidfVectorizer(analyzer='word',stop_words= 'english')

tfidf.fit(X_train["essay"].values)
x_train_fit = tfidf.transform(X_train["essay"])

tfidf_scores = zip( tfidf.idf_, tfidf.get_feature_names())

# sort words in increasing order of tfidf values
tfidf_scores = sorted(tfidf_scores, key = lambda x:x[0])
word_list = [x[1] for x in tfidf_scores]
idf_scores_list =  [x[0] for x in tfidf_scores]
```

In [335]:

```
sns.boxplot([x[0] for x in tfidf_scores])
```

Out[335]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb86f2666d0>
```

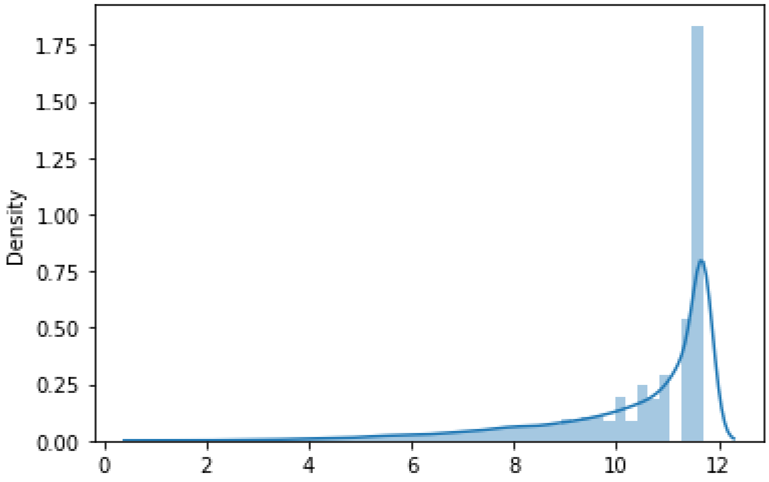


In [336]:

```
sns.distplot(idf_scores_list)
```

Out[336]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb869f33510>
```



In [337]:

```
percent_25=  np.percentile([x[0] for x in tfidf_scores],25)
percent_50=  np.percentile([x[0] for x in tfidf_scores],50)
percent_75=  np.percentile([x[0] for x in tfidf_scores],75)
print("25th Percentile : ",percent_25)
print("50th Percentile : ",percent_50)
print("75th Percentile : ",percent_75)
print("IQR : ", percent_75-percent_25)

print("So, we can ignore words with IDF < %.3f & IDF > %.3f which would eliminate frequent and rare words which wont add much knowledge"%(percent_25,percent_75))
```

```
25th Percentile :  9.670188918828364
50th Percentile :  11.279626831262464
75th Percentile :  11.685091939370627
IQR :  2.0149030205422633
So, we can ignore words with IDF < 9.670 & IDF > 11.685 which would eliminate frequent and rare words which wont add much knowledge
```

In [338]:

```
#removing low and high IDF values
tfidf_removed_words = set([word for idf_vals, word in tfidf_scores if idf_vals<percent_25 and idf_vals>percent_75])

def remove_words(data_values, word_to_be_removed):
  re_remove_words = re.compile("("+ "|".join(word_to_be_removed) + ")")
  clean_data = data_values
  for idx, row in tqdm(enumerate(clean_data)):
    row = re.sub(re_remove_words, "",row) #remove words
    row = re.sub(" +"," ",row) # replace multiple spaces by one space
    clean_data[idx] = row

  return clean_data
```

In [339]:

```
X_train_cleaned_text = remove_words(X_train["essay"].values, tfidf_removed_words)
X_test_cleaned_text = remove_words(X_test["essay"].values, tfidf_removed_words)
```

```
87398it [00:14, 6026.57it/s]
21850it [00:03, 6035.01it/s]
```

In [340]:

```
#use tf.tokenizer : remove '_' from filters as we need words joined by _ (new_york)
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}_~\t\n')
tokenizer.fit_on_texts(X_train_cleaned_text)

# Encode training data sentences into sequences
train_sequences_seq_text = tokenizer.texts_to_sequences(X_train_cleaned_text)
test_sequences_seq_text  = tokenizer.texts_to_sequences(X_test_cleaned_text)

vocab_size_seq_text = len(tokenizer.word_index) + 1
maxlen_seq_text  = max([len(x) for x in train_sequences_seq_text])

# Pad the sequences based on maxLen
train_sequences_seq_text = pad_sequences(train_sequences_seq_text, padding='post', truncating='post', maxlen=maxlen_seq_text)
test_sequences_seq_text = pad_sequences(test_sequences_seq_text, padding='post', truncating='post', maxlen=maxlen_seq_text)
```

Create Embedding Matrix for LSTM Input

In [341]:

```
len(tokenizer.word_index.items())
```

Out[341]:

```
51564
```

In [342]:

```
with open('glove_vectors', 'rb') as f:
  glove_vector = pickle.load(f)
  glove_words = set(glove_vector.keys())
```

In [343]:

```
embedding_matrix = np.zeros((vocab_size_seq_text, 300))
for word, i in tokenizer.word_index.items():
        embedding_vector = glove_vector.get(word)
        if embedding_vector is not None:
                embedding_matrix[i] = embedding_vector
```

In [344]:

```
print("Shape of Embedding Matrix: %d x %d"%(len(embedding_matrix),len(embedding_matrix[0])))
```

```
Shape of Embedding Matrix: 51565 x 300
```

Tokenizing other inputs

Input_school_state

In [345]:
```
#use tf.tokenizer : remove '_' from filters as we need words joined by _ (new_york)
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,-./:;<=>?@[\\]^`{|}_~\t\n')
tokenizer.fit_on_texts(X_train["school_state"])

# Encode training data sentences into sequences
train_sequences_school_state = tokenizer.texts_to_sequences(X_train["school_state"])
test_sequences_school_state = tokenizer.texts_to_sequences(X_test["school_state"])

# all other data is of np.ndarray form, this is a list
train_sequences_school_state = np.array(train_sequences_school_state)
test_sequences_school_state = np.array(test_sequences_school_state)

# Pad the sequences based on maxLen
# train_sequences_clean_cats = pad_sequences(train_sequences_clean_cats, padding='post', truncating='post', maxlen=maxlen_clean_cats)
# test_sequences_clean_cats = pad_sequences(test_sequences_clean_cats, padding='post', truncating='post', maxlen=maxlen_clean_cats)

vocab_size_school_state = len(tokenizer.word_index) + 1
maxlen_school_state = max([len(x) for x in train_sequences_school_state])
```

project_grade_category

In [346]:
```
#use tf.tokenizer : remove '-' from filters as we need words 5-6
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,./:;<=>?@[\\]^`{|}_~\t\n')
tokenizer.fit_on_texts(X_train["project_grade_category"])

# Encode training data sentences into sequences
train_sequences_grade = tokenizer.texts_to_sequences(X_train["project_grade_category"])
test_sequences_grade = tokenizer.texts_to_sequences(X_test["project_grade_category"])

# all other data is of np.ndarray form, this is a list
train_sequences_grade = np.array(train_sequences_grade)
test_sequences_grade = np.array(test_sequences_grade)

vocab_size_grade = len(tokenizer.word_index) + 1
maxlen_grade = max([len(x) for x in train_sequences_grade])

len(train_sequences_grade)
```

Out[346]:
```
87398
```

clean_categories

In [347]:

```
#use tf.tokenizer : remove '-' from filters as we need words 5-6
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,./:;<=>?@[\\]^`{|}_~\t\n')
tokenizer.fit_on_texts(X_train["clean_categories"])

# Encode training data sentences into sequences
train_sequences_clean_cats = tokenizer.texts_to_sequences(X_train["clean_categories"])
test_sequences_clean_cats = tokenizer.texts_to_sequences(X_test["clean_categories"])

vocab_size_clean_cats = len(tokenizer.word_index) + 1
maxlen_clean_cats = max([len(x) for x in train_sequences_clean_cats])
print(len(train_sequences_clean_cats))

# Pad the sequences based on maxLen
train_sequences_clean_cats = pad_sequences(train_sequences_clean_cats, padding='post', truncating='post', maxlen=maxlen_clean_cats)
test_sequences_clean_cats = pad_sequences(test_sequences_clean_cats, padding='post', truncating='post', maxlen=maxlen_clean_cats)
```

```
87398
```

clean_subcategories

In [348]:

```
#use tf.tokenizer : remove '-' from filters as we need words 5-6
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,./:;<=>?@[\\]^`{|}_~\t\n')
tokenizer.fit_on_texts(X_train["clean_subcategories"])

# Encode training data sentences into sequences
train_sequences_clean_subcats = tokenizer.texts_to_sequences(X_train["clean_subcategories"])
test_sequences_clean_subcats = tokenizer.texts_to_sequences(X_test["clean_subcategories"])

vocab_size_clean_subcats = len(tokenizer.word_index) + 1
maxlen_clean_subcats = max([len(x) for x in train_sequences_clean_subcats])

# Pad the sequences based on maxLen
train_sequences_clean_subcats = pad_sequences(train_sequences_clean_subcats, padding='post', truncating='post', maxlen=maxlen_clean_subcats)
test_sequences_clean_subcats = pad_sequences(test_sequences_clean_subcats, padding='post', truncating='post', maxlen=maxlen_clean_subcats)
```

teacher_prefix

In [349]:

```
#use tf.tokenizer : remove '-' from filters as we need words 5-6
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,./:;<=>?@[\\]^`{|}_~\t\n')
tokenizer.fit_on_texts(X_train["teacher_prefix"])

# Encode training data sentences into sequences
train_sequences_teacher_prefix = tokenizer.texts_to_sequences(X_train["teacher_prefix"])
test_sequences_teacher_prefix = tokenizer.texts_to_sequences(X_test["teacher_prefix"])

vocab_size_teacher_prefix = len(tokenizer.word_index) + 1
maxlen_teacher_prefix = max([len(x) for x in train_sequences_teacher_prefix])

# Pad the sequences based on maxLen
train_sequences_teacher_prefix = pad_sequences(train_sequences_teacher_prefix, padding='post', truncating='post', maxlen=maxlen_teacher_prefix)
test_sequences_teacher_prefix = pad_sequences(test_sequences_teacher_prefix, padding='post', truncating='post', maxlen=maxlen_teacher_prefix)
```

Numerical Features

In [350]:

```
# price teacher_number_of_previously_posted_projects

train_numerical = X_train[["teacher_number_of_previously_posted_projects","price"]].values
test_numerical = X_test[["teacher_number_of_previously_posted_projects","price"]].values

# train_numerical = [[x,y] for x,y in zip(X_train_price_norm, X_train_prev_proj_norm)]
# test_numerical =[[x,y] for x,y in zip(X_test_price_norm, X_test_prev_proj_norm)]
print(train_numerical[0])
```

```
[   0.   1051.92]
```

Model Building

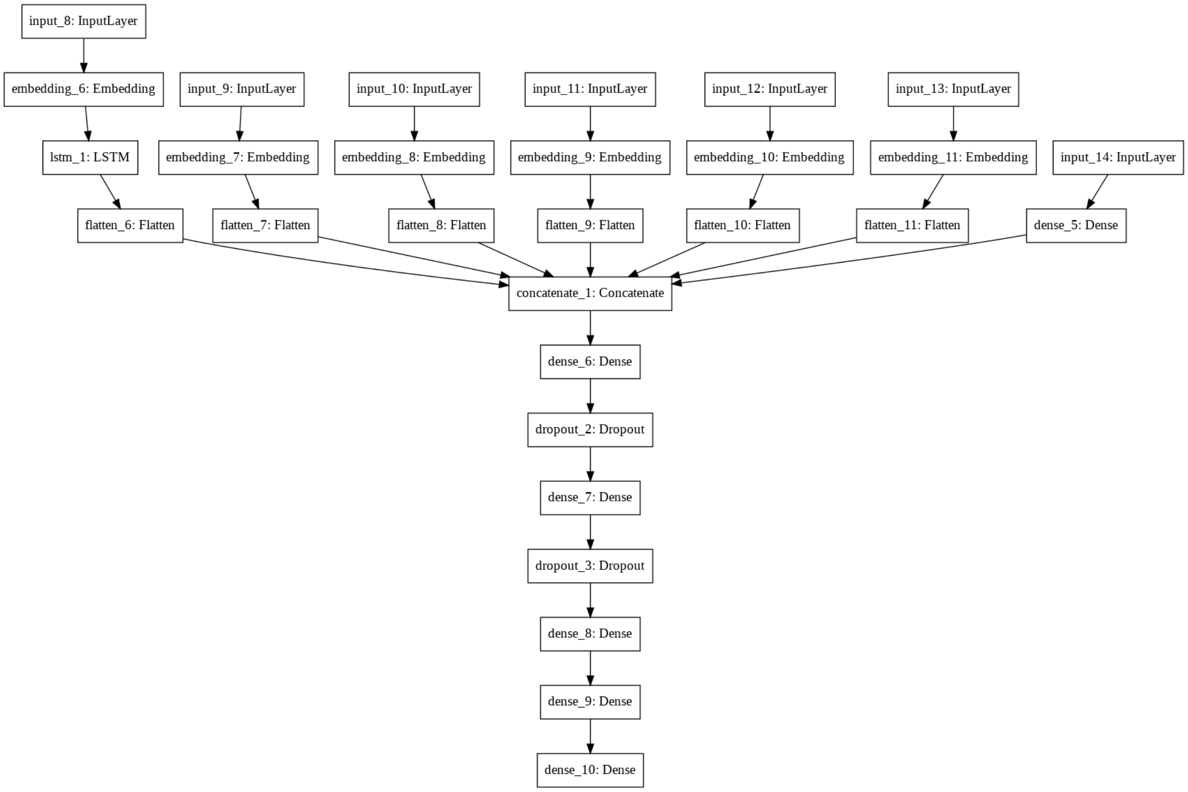
In [351]:

```
# Input_text = Input(shape=(maxlen_seq_text,))
# embed_1 = Embedding(vocab_size_seq_text,300,input_length=maxlen_seq_text)(Input_text)
# lstm_1 = LSTM(units=256)(embed_1)
# flatten_1 = Flatten()(lstm_1)

Input_text = Input(shape=(maxlen_seq_text,))
embed_1 = Embedding(vocab_size_seq_text,300,input_length=maxlen, weights=[embedding_matrix], trainable=False)(Input_text)
lstm_1 = LSTM(units=128)(embed_1)
flatten_1 = Flatten()(lstm_1)

Input_school_state = Input(shape=(maxlen_school_state,))
embed_2 = Embedding(vocab_size_school_state,300,input_length=maxlen_school_state)(Input_school_state)
flatten_2 = Flatten()(embed_2)

Input_grade = Input(shape=(maxlen_grade,))
embed_3 = Embedding(vocab_size_grade,300,input_length=maxlen_grade)(Input_grade)
flatten_3 = Flatten()(embed_3)

Input_clean_cats = Input(shape=(maxlen_clean_cats,))
embed_4 = Embedding(vocab_size_clean_cats,300,input_length=maxlen_clean_cats)(Input_clean_cats)
flatten_4 = Flatten()(embed_4)

Input_clean_subcats = Input(shape=(maxlen_clean_subcats,))
embed_5 = Embedding(vocab_size_clean_subcats,300,input_length=maxlen_clean_subcats)(Input_clean_subcats)
flatten_5 = Flatten()(embed_5)

Input_teacher_prefix = Input(shape=(maxlen_teacher_prefix,))
embed_6 = Embedding(vocab_size_teacher_prefix,300,input_length=maxlen_teacher_prefix)(Input_teacher_prefix)
flatten_6 = Flatten()(embed_6)

Input_numerical = Input(shape=(2,))
dense_numerical = dense_2 = Dense(300,activation='relu')(Input_numerical)

concat = Concatenate()([flatten_1, flatten_2, flatten_3, flatten_4, flatten_5, flatten_6, dense_numerical])

dense_1 = Dense(512,activation='relu')(concat)
drop_1 = Dropout(0.2)(dense_1)
dense_2 = Dense(128,activation='relu')(drop_1)
drop_2 = Dropout(0.2)(dense_2)
dense_3 = Dense(64,activation='relu')(drop_2)
dense_4 = Dense(32,activation='relu')(dense_3)
output = Dense(2, activation='softmax')(dense_4)
```

In [352]:
```
model2 = keras.Model([Input_text, Input_school_state, Input_grade, Inp
ut_clean_cats,Input_clean_subcats,Input_teacher_prefix, Input_numerica
l],output)
```

In [353]:
```
plot_model(model2)
```

Out[353]:



In [355]:
```
!rm -rf ./logs
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S"
)

# tensorboard Callback
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir,
    histogram_freq=1,
    write_graph=True)

#Model checkpoint callback
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath="models/LSTM_Model_2_{epoch:04d}.hdf5",
    verbose=1,
    save_weights_only=True,
    period=10)
```

```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_f
req` to specify the frequency in number of batches seen.
```

In [356]:

```
def auroc(y_true, y_pred):
  return tf.py_function(roc_auc_score, (y_true, y_pred), tf.double)


model2.compile(optimizer= tf.keras.optimizers.Adam(learning_rate=0.00001),
              loss="categorical_crossentropy",
              metrics=["accuracy", auroc])

train_data_inputs = [train_sequences_seq_text, train_sequences_school_state, train_sequences_grade, train_sequences_clean_cats, train_sequences_clean_subcats, train_sequences_teacher_prefix, train_numerical]
test_data_inputs = [test_sequences_seq_text, test_sequences_school_state, test_sequences_grade, test_sequences_clean_cats, test_sequences_clean_subcats, test_sequences_teacher_prefix, test_numerical]
```

In [357]:

```
model2.fit(train_data_inputs,y_train,
           batch_size=128,
          epochs=20,
          validation_data=(test_data_inputs,y_test),
           callbacks= [tensorboard_callback, cp_callback]
)
```

```
Epoch 1/20
683/683 [==============================] - 34s 47ms/step - loss: 0.556
4 - accuracy: 0.7956 - auroc: 0.4944 - val_loss: 0.4221 - val_accurac
y: 0.8486 - val_auroc: 0.6082
Epoch 2/20
683/683 [==============================] - 31s 46ms/step - loss: 0.419
3 - accuracy: 0.8481 - auroc: 0.6094 - val_loss: 0.4157 - val_accurac
y: 0.8486 - val_auroc: 0.6137
Epoch 3/20
683/683 [==============================] - 31s 46ms/step - loss: 0.418
0 - accuracy: 0.8465 - auroc: 0.6219 - val_loss: 0.4112 - val_accurac
y: 0.8486 - val_auroc: 0.6313
Epoch 4/20
683/683 [==============================] - 31s 46ms/step - loss: 0.411
1 - accuracy: 0.8495 - auroc: 0.6331 - val_loss: 0.4109 - val_accurac
y: 0.8486 - val_auroc: 0.6329
Epoch 5/20
683/683 [==============================] - 31s 46ms/step - loss: 0.409
4 - accuracy: 0.8478 - auroc: 0.6503 - val_loss: 0.3935 - val_accurac
y: 0.8486 - val_auroc: 0.6981
Epoch 6/20
683/683 [==============================] - 31s 45ms/step - loss: 0.391
9 - accuracy: 0.8483 - auroc: 0.7081 - val_loss: 0.3916 - val_accurac
y: 0.8483 - val_auroc: 0.7057
Epoch 7/20
683/683 [==============================] - 31s 46ms/step - loss: 0.387
6 - accuracy: 0.8494 - auroc: 0.7145 - val_loss: 0.3874 - val_accurac
y: 0.8481 - val_auroc: 0.7110
Epoch 8/20
683/683 [==============================] - 31s 46ms/step - loss: 0.389
2 - accuracy: 0.8472 - auroc: 0.7191 - val_loss: 0.3936 - val_accurac
y: 0.8486 - val_auroc: 0.7050
Epoch 9/20
683/683 [==============================] - 31s 46ms/step - loss: 0.390
4 - accuracy: 0.8469 - auroc: 0.7160 - val_loss: 0.3888 - val_accurac
y: 0.8486 - val_auroc: 0.7126
Epoch 10/20
683/683 [==============================] - 31s 46ms/step - loss: 0.381
3 - accuracy: 0.8497 - auroc: 0.7258 - val_loss: 0.3838 - val_accurac
y: 0.8486 - val_auroc: 0.7206

Epoch 00010: saving model to models/LSTM_Model_2_0010.hdf5
Epoch 11/20
683/683 [==============================] - 31s 46ms/step - loss: 0.386
0 - accuracy: 0.8482 - auroc: 0.7239 - val_loss: 0.3877 - val_accurac
y: 0.8486 - val_auroc: 0.7241
Epoch 12/20
683/683 [==============================] - 31s 45ms/step - loss: 0.384
8 - accuracy: 0.8465 - auroc: 0.7295 - val_loss: 0.3951 - val_accurac
y: 0.8481 - val_auroc: 0.7246
Epoch 13/20
683/683 [==============================] - 31s 46ms/step - loss: 0.383
9 - accuracy: 0.8479 - auroc: 0.7271 - val_loss: 0.3809 - val_accurac
y: 0.8486 - val_auroc: 0.7280
Epoch 14/20
683/683 [==============================] - 31s 45ms/step - loss: 0.376
8 - accuracy: 0.8498 - auroc: 0.7351 - val_loss: 0.3868 - val_accurac
```

```
y: 0.8486 - val_auroc: 0.7271
Epoch 15/20
683/683 [==============================] - 31s 46ms/step - loss: 0.375
8 - accuracy: 0.8495 - auroc: 0.7364 - val_loss: 0.3804 - val_accurac
y: 0.8494 - val_auroc: 0.7322
Epoch 16/20
683/683 [==============================] - 31s 46ms/step - loss: 0.377
5 - accuracy: 0.8483 - auroc: 0.7426 - val_loss: 0.3801 - val_accurac
y: 0.8497 - val_auroc: 0.7288
Epoch 17/20
683/683 [==============================] - 31s 46ms/step - loss: 0.378
6 - accuracy: 0.8479 - auroc: 0.7385 - val_loss: 0.3821 - val_accurac
y: 0.8487 - val_auroc: 0.7309
Epoch 18/20
683/683 [==============================] - 31s 46ms/step - loss: 0.376
5 - accuracy: 0.8494 - auroc: 0.7415 - val_loss: 0.3913 - val_accurac
y: 0.8418 - val_auroc: 0.7293
Epoch 19/20
683/683 [==============================] - 31s 46ms/step - loss: 0.375
2 - accuracy: 0.8492 - auroc: 0.7442 - val_loss: 0.3774 - val_accurac
y: 0.8489 - val_auroc: 0.7358
Epoch 20/20
683/683 [==============================] - 31s 46ms/step - loss: 0.370
0 - accuracy: 0.8505 - auroc: 0.7527 - val_loss: 0.3758 - val_accurac
y: 0.8496 - val_auroc: 0.7395

Epoch 00020: saving model to models/LSTM_Model_2_0020.hdf5
```

```
Out[357]: <tensorflow.python.keras.callbacks.History at 0x7fb70f1a4a90>
```

```
In [358]: %tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6006 (pid 5520), started 0:25:32 ago. (Use
'!kill 5520' to kill it.)
```

```
In [ ]:
```

Model 3

```
In [371]: y = data["project_is_approved"]
          X = data.drop("project_is_approved", axis=1)

          X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
          15, stratify=y)
          y_train = to_categorical(y_train,2)
          y_test = to_categorical(y_test,2)
```

Create Embedding Matrix for LSTM Input

In [372]:

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
25, stratify=y)
y_train = to_categorical(y_train,2)
y_test = to_categorical(y_test,2)
```

In [373]:

```
#use tf.tokenizer : remove '_' from filters as we need words joined by
_ (new_york)
tokenizer = Tokenizer(lower=True, split=' ',filters='!"#$%&()*+,-./:;<
=>?@[\\]^`{|}_~\t\n')
tokenizer.fit_on_texts(X_train["essay"])

# Encode training data sentences into sequences
train_sequences = tokenizer.texts_to_sequences(X_train["essay"])
test_sequences = tokenizer.texts_to_sequences(X_test["essay"])
```

In [374]:

```
vocab_size = len(tokenizer.word_index) + 1
print("Learned Vocab has size : ",vocab_size)
maxlen = max([len(x) for x in train_sequences])
print("Maximum len of words in train_data is: ", maxlen)
```

```
Learned Vocab has size :  50497
Maximum len of words in train_data is:  339
```

In [375]:

```
# Pad the sequences based on maxLen
train_padded = pad_sequences(train_sequences, padding='post', truncati
ng='post', maxlen=maxlen)
test_padded = pad_sequences(test_sequences, padding='post', truncating
='post', maxlen=maxlen)
```

In [376]:

```
with open('glove_vectors', 'rb') as f:
  glove_vector = pickle.load(f)
  glove_words = set(glove_vector.keys())
```

In [377]:

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

In [378]:

```
print("Shape of Embedding Matrix: %d x %d"%(len(embedding_matrix),len(
embedding_matrix[0])))
```

```
Shape of Embedding Matrix: 50497 x 300
```

One hot encoding categorical features and concatenate

In [379]:

```
# teacher_prefix
# clean_subcategories
# clean_categories
# project_grade_category
```

In [380]:

```
ohe = OneHotEncoder().fit(np.reshape(X_train['school_state'].values,(-1,1)))

train_ohe_school_state = ohe.transform(X_train["school_state"].values.reshape(-1,1)).toarray()
test_ohe_school_state = ohe.transform(X_test["school_state"].values.reshape(-1,1)).toarray()

print(len(train_ohe_school_state),len(train_ohe_school_state[0]))
print(len(test_ohe_school_state),len(test_ohe_school_state[0]))
```

```
81936 51
27312 51
```

In [381]:

```
ohe = OneHotEncoder().fit(np.reshape(X_train['teacher_prefix'].values,(-1,1)))
train_ohe_teacher_prefix = ohe.transform(X_train["teacher_prefix"].values.reshape(-1,1)).toarray()
test_ohe_teacher_prefix = ohe.transform(X_test["teacher_prefix"].values.reshape(-1,1)).toarray()

ohe = OneHotEncoder(handle_unknown = 'ignore').fit(np.reshape(X_train['clean_subcategories'].values,(-1,1)))
train_ohe_clean_subcategories = ohe.transform(X_train["clean_subcategories"].values.reshape(-1,1)).toarray()
test_ohe_clean_subcategories = ohe.transform(X_test["clean_subcategories"].values.reshape(-1,1)).toarray()

ohe = OneHotEncoder().fit(np.reshape(X_train['clean_categories'].values,(-1,1)))
train_ohe_clean_categories = ohe.transform(X_train["clean_categories"].values.reshape(-1,1)).toarray()
test_ohe_clean_categories = ohe.transform(X_test["clean_categories"].values.reshape(-1,1)).toarray()

ohe = OneHotEncoder().fit(np.reshape(X_train['project_grade_category'].values,(-1,1)))
train_ohe_project_grade_category = ohe.transform(X_train["project_grade_category"].values.reshape(-1,1)).toarray()
test_ohe_project_grade_category = ohe.transform(X_test["project_grade_category"].values.reshape(-1,1)).toarray()
```

In [382]:

```
normalizer = Normalizer()
normalizer.fit(X_train['price'].values.reshape(1,-1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))

X_train_price_norm = X_train_price_norm.reshape(1,-1)[0]
X_test_price_norm = X_test_price_norm.reshape(1,-1)[0]
```

In [383]:

```
# teacher_number_of_previously_posted_projects
normalizer1 = Normalizer()
normalizer1.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

X_train_prev_proj_norm = normalizer1.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_test_prev_proj_norm = normalizer1.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

X_train_prev_proj_norm = X_train_price_norm.reshape(1,-1)[0]
X_test_prev_proj_norm = X_test_price_norm.reshape(1,-1)[0]
```

In [384]:

```
# price teacher_number_of_previously_posted_projects

# train_numerical = X_train[["teacher_number_of_previously_posted_projects","price"]].values
# test_numerical = X_test[["teacher_number_of_previously_posted_projects","price"]].values

train_numerical = np.array([[x,y] for x,y in zip(X_train_price_norm, X_train_prev_proj_norm)])
test_numerical = np.array([[x,y] for x,y in zip(X_test_price_norm, X_test_prev_proj_norm)])

print(train_numerical.shape)
print(test_numerical.shape)
```

```
(81936, 2)
(27312, 2)
```

In [385]:

```
# train_ohe_teacher_prefix
# train_ohe_clean_subcategories
# train_ohe_clean_categories
# train_ohe_project_grade_category
```

In [386]:

```
X_train_rem_features = np.hstack((train_ohe_teacher_prefix,train_ohe_clean_subcategories,train_ohe_clean_categories,train_ohe_project_grade_category,train_numerical))
X_test_rem_features = np.hstack((test_ohe_teacher_prefix,test_ohe_clean_subcategories,test_ohe_clean_categories,test_ohe_project_grade_category,test_numerical))

X_train_rem_features = np.expand_dims(X_train_rem_features, axis=2)
X_test_rem_features = np.expand_dims(X_test_rem_features, axis=2)

# print(X_train_numerical.shape)
# print(X_test_numerical.shape)
```

Model Building

In [392]:

```
#input for LSTM input layer
Input_text = Input(shape=(maxlen,))
embed_1 = Embedding(vocab_size,300,input_length=maxlen, weights=[embedding_matrix], trainable=False)(Input_text)
lstm_1 = LSTM(units=128)(embed_1)
flatten_1 = Flatten()(lstm_1)


# input for other features
Input_rem_features = Input(shape=X_train_rem_features[0].shape)
conv1 = Conv1D(filters=16, kernel_size=12, activation= 'relu')(Input_rem_features)
conv2 = Conv1D(filters=8, kernel_size=12, activation= 'relu')(conv1)
flatten_2 = Flatten()(conv2)

concat_layer = Concatenate()([flatten_1,flatten_2])

dense1 = Dense(128,activation='relu')(concat_layer)
drop1 = Dropout(0.1)(dense1)
dense2 = Dense(64,activation='relu')(drop1)
drop2 = Dropout(0.1)(dense2)
dense3 = Dense(48,activation='relu')(drop2)

output = Dense(2, activation='softmax')(dense3)
```

In [393]:

```
model3 = keras.Model([Input_text,Input_rem_features], output)
```

In [394]:

```
!rm -rf ./logs
log_dir="logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S"
)

# tensorboard Callback
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir,
    histogram_freq=1,
    write_graph=True)

#Model checkpoint callback
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath="models/LSTM_Model_3_{epoch:04d}.hdf5",
    verbose=1,
    save_weights_only=True,
    period=10)
```

```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_f
req` to specify the frequency in number of batches seen.
```

In [395]:

```
def auroc(y_true, y_pred):
    return tf.py_function(roc_auc_score, (y_true, y_pred), tf.double)


model3.compile(optimizer=tf.keras.optimizers.Adam(lr = 0.00001),
              loss="categorical_crossentropy",
              metrics=["accuracy", auroc])

train_data_inputs = [train_padded, X_train_rem_features]
test_data_inputs = [test_padded, X_test_rem_features]
```

In [396]:

```
model3.fit(train_data_inputs,y_train,
  epochs=20,
  batch_size = 128,
  validation_data=(test_data_inputs,y_test),
  callbacks = [tensorboard_callback, cp_callback]
)
```

```
Epoch 1/20
641/641 [==============================] - 31s 46ms/step - loss: 0.600
5 - accuracy: 0.7865 - auroc: 0.4941 - val_loss: 0.4237 - val_accurac
y: 0.8486 - val_auroc: 0.5661
Epoch 2/20
641/641 [==============================] - 28s 44ms/step - loss: 0.429
7 - accuracy: 0.8466 - auroc: 0.5257 - val_loss: 0.4224 - val_accurac
y: 0.8486 - val_auroc: 0.5857
Epoch 3/20
641/641 [==============================] - 28s 44ms/step - loss: 0.418
8 - accuracy: 0.8491 - auroc: 0.5824 - val_loss: 0.4007 - val_accurac
y: 0.8486 - val_auroc: 0.6793
Epoch 4/20
641/641 [==============================] - 28s 44ms/step - loss: 0.406
6 - accuracy: 0.8468 - auroc: 0.6594 - val_loss: 0.3968 - val_accurac
y: 0.8486 - val_auroc: 0.6935
Epoch 5/20
641/641 [==============================] - 28s 44ms/step - loss: 0.403
0 - accuracy: 0.8461 - auroc: 0.6749 - val_loss: 0.3922 - val_accurac
y: 0.8486 - val_auroc: 0.7032
Epoch 6/20
641/641 [==============================] - 28s 44ms/step - loss: 0.396
4 - accuracy: 0.8470 - auroc: 0.6876 - val_loss: 0.3975 - val_accurac
y: 0.8486 - val_auroc: 0.6995
Epoch 7/20
641/641 [==============================] - 28s 44ms/step - loss: 0.395
4 - accuracy: 0.8466 - auroc: 0.6933 - val_loss: 0.3886 - val_accurac
y: 0.8486 - val_auroc: 0.7128
Epoch 8/20
641/641 [==============================] - 28s 44ms/step - loss: 0.391
6 - accuracy: 0.8481 - auroc: 0.6974 - val_loss: 0.3900 - val_accurac
y: 0.8486 - val_auroc: 0.7150
Epoch 9/20
641/641 [==============================] - 28s 44ms/step - loss: 0.390
2 - accuracy: 0.8475 - auroc: 0.7033 - val_loss: 0.3872 - val_accurac
y: 0.8486 - val_auroc: 0.7165
Epoch 10/20
641/641 [==============================] - 28s 44ms/step - loss: 0.387
6 - accuracy: 0.8486 - auroc: 0.7044 - val_loss: 0.3851 - val_accurac
y: 0.8486 - val_auroc: 0.7205

Epoch 00010: saving model to models/LSTM_Model_3_0010.hdf5
Epoch 11/20
641/641 [==============================] - 28s 44ms/step - loss: 0.386
3 - accuracy: 0.8477 - auroc: 0.7123 - val_loss: 0.3847 - val_accurac
y: 0.8486 - val_auroc: 0.7223
Epoch 12/20
641/641 [==============================] - 28s 44ms/step - loss: 0.385
8 - accuracy: 0.8491 - auroc: 0.7065 - val_loss: 0.3835 - val_accurac
y: 0.8486 - val_auroc: 0.7229
Epoch 13/20
641/641 [==============================] - 28s 44ms/step - loss: 0.384
7 - accuracy: 0.8481 - auroc: 0.7136 - val_loss: 0.3880 - val_accurac
y: 0.8486 - val_auroc: 0.7252
Epoch 14/20
641/641 [==============================] - 28s 44ms/step - loss: 0.382
5 - accuracy: 0.8493 - auroc: 0.7155 - val_loss: 0.3835 - val_accurac
```

```
y: 0.8486 - val_auroc: 0.7227
Epoch 15/20
641/641 [==============================] - 28s 44ms/step - loss: 0.386
5 - accuracy: 0.8454 - auroc: 0.7213 - val_loss: 0.3812 - val_accurac
y: 0.8486 - val_auroc: 0.7268
Epoch 16/20
641/641 [==============================] - 28s 44ms/step - loss: 0.380
6 - accuracy: 0.8487 - auroc: 0.7213 - val_loss: 0.3813 - val_accurac
y: 0.8487 - val_auroc: 0.7288
Epoch 17/20
641/641 [==============================] - 28s 44ms/step - loss: 0.379
5 - accuracy: 0.8499 - auroc: 0.7198 - val_loss: 0.3802 - val_accurac
y: 0.8487 - val_auroc: 0.7291
Epoch 18/20
641/641 [==============================] - 28s 44ms/step - loss: 0.381
5 - accuracy: 0.8469 - auroc: 0.7250 - val_loss: 0.3800 - val_accurac
y: 0.8488 - val_auroc: 0.7293
Epoch 19/20
641/641 [==============================] - 29s 45ms/step - loss: 0.379
3 - accuracy: 0.8482 - auroc: 0.7290 - val_loss: 0.3843 - val_accurac
y: 0.8486 - val_auroc: 0.7312
Epoch 20/20
641/641 [==============================] - 29s 45ms/step - loss: 0.381
7 - accuracy: 0.8470 - auroc: 0.7275 - val_loss: 0.3800 - val_accurac
y: 0.8483 - val_auroc: 0.7318

Epoch 00020: saving model to models/LSTM_Model_3_0020.hdf5
```

Out[396]:
```
<tensorflow.python.keras.callbacks.History at 0x7fb70ebd9290>
```

In [397]:
```
%tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6006 (pid 5520), started 0:58:40 ago. (Use
'!kill 5520' to kill it.)
```

In [ ]:
```
!jupyter nbconver
```