In [ ]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics
from sklearn.metrics import confusion_matrix, roc_curve, auc
import re
from keras.initializers import RandomNormal
from keras import optimizers
from keras.callbacks import TensorBoard, ModelCheckpoint, EarlyStopping
import pickle
from tqdm import tqdm
import os
import keras
```

```
Using TensorFlow backend.
```

In [ ]:
```
%load_ext tensorboard
```

In [ ]:
```
# Load the Drive helper and mount
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

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

In [ ]:
```
import io
resource_data=pd.read_csv('drive/My Drive/AAIC/LSTM assignment/LSTM Assignment/resources.csv')
project_data=pd.read_csv('drive/My Drive//AAIC/LSTM assignment/LSTM Assignment/train_data.csv')
pre_data = pd.read_csv('drive/My Drive//AAIC/LSTM assignment/LSTM Assignment/preprocessed_data.csv')
#project_data=project_data.sample(n=1000)
```

In [ ]:
```
pre_data.head()
```

Out[ ]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	c
0	ca	mrs	grades_prek_2	53	1	math_science	
1	ut	ms	grades_3_5	4	1	specialneeds	
2	ca	mrs	grades_prek_2	10	1	literacy_language	
3	ga	mrs	grades_prek_2	2	1	appliedlearning	
4	wa	mrs	grades_3_5	2	1	literacy_language	

In [ ]:
```
pre_data.columns
```

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

In [ ]:
```

```

Train-Test split

In [ ]:
```
y = pre_data['project_is_approved'].values
X = pre_data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[ ]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	f
0	ca	mrs	grades_prek_2	53	math_science	appliedsciences health_lifescience	

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

X_train,X_test,y_train, y_test=train_test_split(X, y, test_size=0.2)
X_train,X_cv,y_train,y_cv=train_test_split(X_train, y_train, test_size=0.2)
print(X_train.shape)
print(X_test.shape)

print(y_train.shape)
print(y_test.shape)
```

```
(69918, 8)
(21850, 8)
(69918,)
(21850,)
```

In [ ]:
```

```

In [ ]:
```
### https://keras.io/api/preprocessing/text/
### https://stackoverflow.com/questions/51699001/tokenizer-texts-to-sequences-keras-tokenizer-gives-almost-all-zeros

from keras.preprocessing.text import Tokenizer

#max_words = 10000
tokenizer = keras.preprocessing.text.Tokenizer( lower=True, split=' ', char_level=False, oov_token=None, document_count=0)
tokenizer.fit_on_texts(X_train["essay"])

essay_sequences_train = tokenizer.texts_to_sequences(X_train['essay'])
essay_sequences_cv = tokenizer.texts_to_sequences(X_cv['essay'])
essay_sequences_test = tokenizer.texts_to_sequences(X_test['essay'])
```

In [ ]:
```
essay_sequences_train[0]
```

Out[ ]: [2,
49,
1,
4807,
472,
8,
290,
92,
1328,
565,
102,
61,
1,
575,
509,
717,
283,
716,
628,
61,
2182,
2313,
433,
1806,
1013,
23,
575,
340,
149,
716,
3583,
1219,
94,
279,
2,
2,
34,
55,
259,
61,
98,
279,
2,
34,
1,
10,
1593,
27,
67,
69,
256,
716,
27,
2,
34,
63,
848,
266,
3,
712,
15,
124,
716,
27,
282,
99,
216,
1,
290,
83,
644,
33,
222,
668,
3,
30,
2,
34,
1,
87,
825,
424,
716,
290,
33,
269,
2,

```
28,
34,
1,
80,
8360,
53,
1050,
1028,
3904,
41,
3415,
5,
80,
7799,
344,
2,
16,
12,
638,
1,
2,
34,
55,
716,
1593,
61,
261,
2,
34,
259,
61,
80,
23,
2,
28,
34,
10,
716,
27,
33,
424,
269,
13]
```

In [ ]:
```
len(essay_sequences_train[0])
```

Out[ ]:
```
130
```

Observation : With the help of tokenizer in KERAS API we have converted text data into vector format

In [ ]:
```

```

In [ ]:
```
### https://keras.io/api/preprocessing/timeseries/

from keras.preprocessing.sequence import pad_sequences
max_length=0;
for essay in X_train["essay"]:
    temp =len(essay)
    max_length= temp if temp > max_length else max_length

padded_essay_train = pad_sequences(essay_sequences_train, maxlen=max_length, dtype='int32', padding='post', truncat
ing='post', value=0.0)
padded_essay_cv = pad_sequences(essay_sequences_cv, maxlen=max_length, dtype='int32', padding='post', truncating='p
ost', value=0.0)
padded_essay_test = pad_sequences(essay_sequences_test, maxlen=max_length, dtype='int32', padding='post', truncatin
g='post', value=0.0)
```

In [ ]:
```
padded_essay_train[0]
```

Out[ ]:
```
array([ 2, 49,  1, ...,  0,  0,  0], dtype=int32)
```

In [ ]:
```
padded_essay_train[0].shape
```

Out[ ]:
```
(2657,)
```

In [ ]:
```
padded_essay_train[5].shape
```

Out[ ]:
```
(2657,)
```

Observation : After vectorization of text data, we have padded each vector using Keras pad_sequences to bring them into one same shape

In [ ]:

In [ ]:
```
### Loading Glove Vectors

import pickle
with open("drive/My Drive/AAIC/DONORS_CHOOSE/Assignments_DonorsChoose_2018/glove_vectors","rb") as file:
    embeddings = pickle.load(file)

# vocab_size + 1 since, 0 is not considered by tokenizer.
vocab_size = len(tokenizer.word_index)+1

# create a weight matrix for words in training docs
embedding_matrix = np.zeros((vocab_size, 300))

for word, i in tqdm(tokenizer.word_index.items()):
                embedding_vector = embeddings.get(word)
                if embedding_vector is not None:
                                embedding_matrix[i] = embedding_vector

embedding_matrix.shape
```

```
100%|██████████| 47280/47280 [00:00<00:00, 253868.64it/s]
```

Out[ ]:
```
(47281, 300)
```

In [ ]:
```
### https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-layer-work
### https://stackoverflow.com/questions/53503389/how-to-set-parameters-in-keras-to-be-non-trainable

from keras.engine.input_layer import Input
from keras.layers import Embedding, Flatten, LSTM, Dense
from keras.initializers import Constant

essay_input_layer = Input(shape=(max_length,), name="essay_i/p")#, batch_shape=(1000,max_length))
essay_embedding_layer = Embedding(vocab_size, 300, \
                                  embeddings_initializer=Constant(embedding_matrix),
                                  input_length=max_length)(essay_input_layer)
essay_embedding_layer.trainable = False
essay_lstm_layer = LSTM(10, return_sequences=True)(essay_embedding_layer)
# https://github.com/keras-team/keras/issues/7403
essay_flatten = Flatten()(essay_lstm_layer)
```

In [ ]:
```
max_length, essay_input_layer.shape , essay_embedding_layer.shape , essay_lstm_layer.shape , essay_flatten.shape
```

Out[ ]:
```
(2657,
 TensorShape([None, 2657]),
 TensorShape([None, 2657, 300]),
 TensorShape([None, 2657, 10]),
 TensorShape([None, None]))
```

School State

In [ ]:
```
"""
Using LabelEncoder and create embeddings on categorical data.
Also dealing with unseen data, during le.transform()
https://medium.com/@satnalikamayank12/on-learning-embeddings-for-categorical-data-using-keras-165ff2773fc9
https://stackoverflow.com/questions/21057621/sklearn-labelencoder-with-never-seen-before-values
"""

# school_state - LabelEncoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
school_state_x_train = le.fit_transform(X_train["school_state"])
school_state_x_val = le.transform(X_cv["school_state"])
school_state_x_test = le.transform(X_test["school_state"])
print(le.classes_)
# we don't have any unseen data in val, test data.

# school_state
school_state_vocab_size = X_train["school_state"].nunique() # gives unique values count
school_state_output_dim = int(min(np.ceil((school_state_vocab_size)/2), 50 ))  # Half of the input_dim i.e vocalbul
ary_size
school_state_input_length = 1 # Max num of words for one row, here every row has only single value.

#layers
school_state_input_layer = Input(shape=(school_state_input_length,), name="school_state")
school_state_embedding_layer = Embedding(school_state_vocab_size+1, school_state_output_dim,\
                                         embeddings_initializer='he_normal',
                                         input_length=school_state_input_length)(school_state_input_layer)

school_state_flatten = Flatten()(school_state_embedding_layer)

print("school_state Layers created")
print(school_state_x_train.shape, "\n", school_state_x_val.shape, "\n", school_state_x_test.shape)
```

```
['ak' 'al' 'ar' 'az' 'ca' 'co' 'ct' 'dc' 'de' 'fl' 'ga' 'hi' 'ia' 'id'
 'il' 'in' 'ks' 'ky' 'la' 'ma' 'md' 'me' 'mi' 'mn' 'mo' 'ms' 'mt' 'nc'
 'nd' 'ne' 'nh' 'nj' 'nm' 'nv' 'ny' 'oh' 'ok' 'or' 'pa' 'ri' 'sc' 'sd'
 'tn' 'tx' 'ut' 'va' 'vt' 'wa' 'wi' 'wv' 'wy']
school_state Layers created
(69918,)
 (17480,)
 (21850,)
```

In [ ]:
```
school_state_input_layer.shape , school_state_embedding_layer.shape , school_state_flatten.shape
```

Out[ ]:
```
(TensorShape([None, 1]), TensorShape([None, 1, 26]), TensorShape([None, None]))
```

teacher_prefix

In [ ]:
```
"""
Using LabelEncoder and create embeddings on categorical data.
Also dealing with unseen data, during le.transform()
https://medium.com/@satnalikamayank12/on-learning-embeddings-for-categorical-data-using-keras-165ff2773fc9
https://stackoverflow.com/questions/21057621/sklearn-labelencoder-with-never-seen-before-values
"""

# teacher_prefix - LabelEncoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
teacher_prefix_x_train = le.fit_transform(X_train["teacher_prefix"])
teacher_prefix_x_val = le.transform(X_cv["teacher_prefix"])
teacher_prefix_x_test = le.transform(X_test["teacher_prefix"])
print(le.classes_)
# we don't have any unseen data in val, test data

# teacher_prefix
teacher_prefix_vocab_size = X_train["teacher_prefix"].nunique() # gives unique values count
teacher_prefix_output_dim = int(min(np.ceil((teacher_prefix_vocab_size)/2), 50 ))  # Half of the input_dim i.e voca
lbulary_size
teacher_prefix_input_length = 1 # Max num of words for one row, here every row has only single value.

# layers
teacher_prefix_input_layer = Input(shape=(teacher_prefix_input_length,), name="teacher_prefix_i/p")
teacher_prefix_embedding_layer = Embedding(teacher_prefix_vocab_size+1, teacher_prefix_output_dim,\
                                           embeddings_initializer="he_normal",\
                                          input_length=teacher_prefix_input_length)(teacher_prefix_input_layer)

teacher_prefix_flatten = Flatten()(teacher_prefix_embedding_layer)

print("teacher_prefix Layers created")
print(teacher_prefix_x_train.shape, "\n", teacher_prefix_x_val.shape, "\n", teacher_prefix_x_test.shape)
```

```
['dr' 'mr' 'mrs' 'ms' 'teacher']
teacher_prefix Layers created
(69918,)
 (17480,)
 (21850,)
```

project_grade_category

In [ ]:
```
"""
Use LabelEncoder & then create embeddings on categorical data.
Also dealing with unseen data, during le.transform()
https://medium.com/@satnalikamayank12/on-learning-embeddings-for-categorical-data-using-keras-165ff2773fc9
https://stackoverflow.com/questions/21057621/sklearn-labelencoder-with-never-seen-before-values
"""

# project_grade_category - LabelEncoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
grade_x_train = le.fit_transform(X_train["project_grade_category"])
grade_x_val = le.transform(X_cv["project_grade_category"])
grade_x_test = le.transform(X_test["project_grade_category"])
print(le.classes_)
# we don't have any unseen data in val, test data

# project_grade_category
grade_vocab_size = X_train["project_grade_category"].nunique()  # gives unique values count
grade_output_dim = int(min(np.ceil((grade_vocab_size)/2), 50 ))  # Half of the input_dim i.e vocalbulary_size
grade_input_length = 1 # Max num of words for one row, here every row has only single value.

# layers
grade_input_layer = Input(shape=(grade_input_length,), name="grade_i/p")
grade_embedding_layer = Embedding(grade_vocab_size+1, grade_output_dim,\
                                  embeddings_initializer="he_normal",\
                                  input_length=grade_input_length)(grade_input_layer)

grade_flatten = Flatten()(grade_embedding_layer)

print("project_grade_category Layers created")
print(grade_x_train.shape, "\n", grade_x_val.shape, "\n", grade_x_test.shape)
```

```
['grades_3_5' 'grades_6_8' 'grades_9_12' 'grades_prek_2']
project_grade_category Layers created
(69918,)
 (17480,)
 (21850,)
```

In [ ]:
```
```

clean_categories

clean_categories have more than one value in each row, which is seperated by space.LabelEncoding may consider multiple values in a row as a unique token. Therefore, use OneHotEncoding (or) Do tokenizer-sequences, padding and then pass it to Embedding Layer

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer


def vectorize_cat_data(x_train, x_val, x_test, col_name):
    #filling nan with NAN
    x_train[col_name] = x_train[col_name].fillna('NAN')
    x_val[col_name] = x_val[col_name].fillna('NAN')
    x_test[col_name] = x_test[col_name].fillna('NAN')

    vectorizer= CountVectorizer(lowercase=False, binary=True)
    vectorizer.fit(x_train[col_name].values)

    col_name_one_hot = vectorizer.transform(x_train[col_name].values)
    # store the feature names of all categorical columns
    print(vectorizer.get_feature_names())
    print("Shape of matrix after one hot encoding ",col_name_one_hot.shape,"\n")

    return vectorizer , col_name_one_hot, vectorizer.transform(x_val[col_name].values), vectorizer.transform(x_test
[col_name].values)
```

In [ ]:
```
# OHE clean_categories
vectorizer, categories_x_train, categories_x_val, categories_x_test = vectorize_cat_data(X_train, X_cv, X_test, 'cl
ean_categories')

# clean_categories - Layer variables/parameters
categories_vocab_size = len(vectorizer.get_feature_names())  # gives unique values count
categories_output_dim = int(min(np.ceil((categories_vocab_size)/2), 50 ))  # Half of the input_dim i.e vocalbulary_
size
categories_input_length = categories_vocab_size # Max num of words for one row, since ts ohe, it has 9 shaped <...>
vector

# Layers
categories_input_layer = Input(shape=(categories_input_length,), name="categories_i/p")
categories_embedding_layer = Embedding(categories_vocab_size, categories_output_dim, \
                                       embeddings_initializer="he_normal",\
                                         input_length=categories_input_length)(categories_input_layer)

categories_flatten = Flatten()(categories_embedding_layer)

print("clean_categories Layers created")
print(categories_x_train.shape, "\n", categories_x_val.shape, "\n", categories_x_test.shape)
```

```
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_
arts', 'specialneeds', 'warmth']
Shape of matrix after one hot encoding  (69918, 9)

clean_categories Layers created
(69918, 9)
 (17480, 9)
 (21850, 9)
```

In [ ]:
```
categories_input_layer.shape , categories_embedding_layer.shape , categories_flatten.shape
```

Out[ ]:
```
(TensorShape([None, 9]), TensorShape([None, 9, 5]), TensorShape([None, None]))
```

clean_subcategories

In [ ]:
```
# OHE clean_subcategories
vectorizer, subcategories_x_train, subcategories_x_val, subcategories_x_test = vectorize_cat_data(X_train, X_cv, X_
test, 'clean_subcategories')

# clean_subcategories - Layer variables/parameters
subcategories_vocab_size = len(vectorizer.get_feature_names())  # gives unique values count
subcategories_output_dim = int(min(np.ceil((subcategories_vocab_size)/2), 50 ))  # Half of the input_dim i.e vocalb
ulary_size
subcategories_input_length = subcategories_vocab_size # Max num of words for one row, here every row has only singl
e value.

# Layers
subcategories_input_layer = Input(shape=(subcategories_input_length,), name="subcategories_i/p")
subcategories_embedding_layer = Embedding(subcategories_vocab_size, subcategories_output_dim, \
                                          embeddings_initializer="glorot_normal",\
                                         input_length=subcategories_input_length)(subcategories_input_layer)

subcategories_flatten = Flatten()(subcategories_embedding_layer)

print("clean_subcategories Layers created")
print(subcategories_x_train.shape, "\n", subcategories_x_val.shape, "\n", subcategories_x_test.shape)
```

```
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityserv
ice', 'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'for
eignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literatu
re_writing', 'mathematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'social
sciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
Shape of matrix after one hot encoding  (69918, 30)

clean_subcategories Layers created
(69918, 30)
 (17480, 30)
 (21850, 30)
```

Numerical_features - combined

In [ ]:
```
from sklearn.preprocessing import StandardScaler, Normalizer
#This function return the normalized data.
def standardize_data(x_train, x_val, x_test, column):
    scalar = StandardScaler()
    scalar.fit(x_train[column].values.reshape(-1,1))
    std_scalar_x_train = scalar.transform(x_train[column].values.reshape(-1,1))
    std_scalar_x_val = scalar.transform(x_val[column].values.reshape(-1,1))
    std_scalar_x_test = scalar.transform(x_test[column].values.reshape(-1,1))
    return std_scalar_x_train, std_scalar_x_val, std_scalar_x_test
```

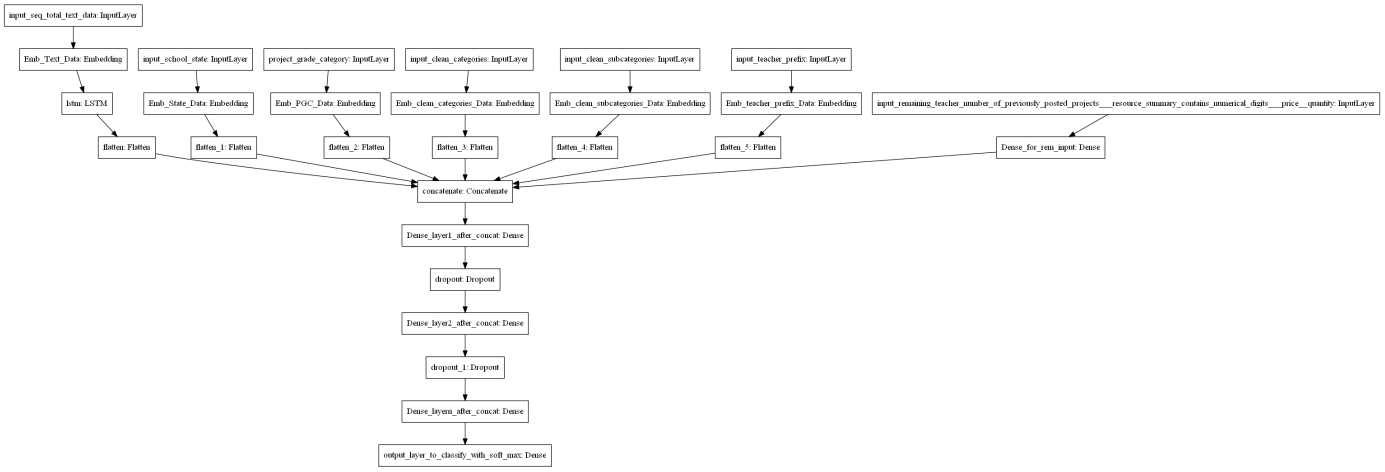
In [ ]:
```
price_x_train, price_x_val, price_x_test = standardize_data(X_train, X_cv, X_test, "price")

teacher_prev_projects_x_train, teacher_prev_projects_x_val, teacher_prev_projects_x_test = standardize_data(X_train
, X_cv, X_test, "teacher_number_of_previously_posted_projects")

numerical_features_x_train = np.hstack((price_x_train, teacher_prev_projects_x_train))
numerical_features_x_val = np.hstack((price_x_val, teacher_prev_projects_x_val))
numerical_features_x_test = np.hstack((price_x_test, teacher_prev_projects_x_test))
```

In [ ]:
```
# Defining Layers
numerical_input_layer = Input(shape=(2,), name="numerical_layer")
numerical_dense = Dense(10, activation='relu')(numerical_input_layer)
```

In [ ]:
```

```

Model-1 Build and Train deep neural network as shown below

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



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

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

In [ ]:
```
from keras.layers import concatenate, Dropout, BatchNormalization, Activation

concatenate_layer = concatenate([essay_flatten, school_state_flatten, grade_flatten, categories_flatten,
                     subcategories_flatten, teacher_prefix_flatten, numerical_dense])
```

In [ ]:
```
dense_layer0 = Dense(64, activation="relu")(concatenate_layer)

BN_layer0 = BatchNormalization()(dense_layer0)

dropout_layer0 = Dropout(0.5)(BN_layer0)

dense_layer1 = Dense(32, activation="relu")(dropout_layer0)

BN_layer1 = BatchNormalization()(dense_layer1)

dropout_layer1 = Dropout(0.40)(BN_layer1)

dense_layer2 = Dense(16, activation="relu")(dropout_layer1)

final_layer = Dense(1, activation='sigmoid', name="o/p_layer")(dense_layer2)
```

In [ ]:
```
from keras.models import Model
model1= Model(inputs=[essay_input_layer, school_state_input_layer, teacher_prefix_input_layer, grade_input_layer,
                      categories_input_layer, subcategories_input_layer, numerical_input_layer],
              outputs=[final_layer])
```

In [ ]:

```
# https://stackoverflow.com/questions/41032551/how-to-compute-receiving-operating-characteristic-roc-and-auc-in-keras
import tensorflow as tf
from sklearn.metrics import roc_auc_score

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

In [ ]:
```
#model1.layers

from keras.optimizers import Adam
adam_optimizer = Adam(learning_rate=0.0005, beta_1=0.9, beta_2=0.999)
model1.compile(optimizer=adam_optimizer, loss='binary_crossentropy', metrics = [aucroc])
model1.summary()
```

```
Model: "model_1"
```

Layer (type)	Output Shape	Param #	Connected to
essay_i/p (InputLayer)	(None, 2527)	0	
embedding_1 (Embedding)	(None, 2527, 300)	14181000	essay_i/p[0][0]
school_state (InputLayer)	(None, 1)	0	
grade_i/p (InputLayer)	(None, 1)	0	
categories_i/p (InputLayer)	(None, 9)	0	
subcategories_i/p (InputLayer)	(None, 30)	0	
teacher_prefix_i/p (InputLayer)	(None, 1)	0	
lstm_1 (LSTM)	(None, 2527, 10)	12440	embedding_1[0][0]
embedding_2 (Embedding)	(None, 1, 26)	1352	school_state[0][0]
embedding_4 (Embedding)	(None, 1, 2)	10	grade_i/p[0][0]
embedding_5 (Embedding)	(None, 9, 5)	45	categories_i/p[0][0]
embedding_6 (Embedding)	(None, 30, 15)	450	subcategories_i/p[0][0]
embedding_3 (Embedding)	(None, 1, 3)	18	teacher_prefix_i/p[0][0]
numerical_layer (InputLayer)	(None, 2)	0	
flatten_1 (Flatten)	(None, 25270)	0	lstm_1[0][0]
flatten_2 (Flatten)	(None, 26)	0	embedding_2[0][0]
flatten_4 (Flatten)	(None, 2)	0	embedding_4[0][0]
flatten_5 (Flatten)	(None, 45)	0	embedding_5[0][0]
flatten_6 (Flatten)	(None, 450)	0	embedding_6[0][0]
flatten_3 (Flatten)	(None, 3)	0	embedding_3[0][0]
dense_1 (Dense)	(None, 10)	30	numerical_layer[0][0]
concatenate_1 (Concatenate)	(None, 25806)	0	flatten_1[0][0] flatten_2[0][0] flatten_4[0][0] flatten_5[0][0] flatten_6[0][0] flatten_3[0][0] dense_1[0][0]
dense_2 (Dense)	(None, 64)	1651648	concatenate_1[0][0]
batch_normalization_1 (BatchNor	(None, 64)	256	dense_2[0][0]
dropout_1 (Dropout)	(None, 64)	0	batch_normalization_1[0][0]
dense_3 (Dense)	(None, 32)	2080	dropout_1[0][0]
batch_normalization_2 (BatchNor	(None, 32)	128	dense_3[0][0]
dropout_2 (Dropout)	(None, 32)	0	batch_normalization_2[0][0]
dense_4 (Dense)	(None, 16)	528	dropout_2[0][0]
o/p_layer (Dense)	(None, 1)	17	dense_4[0][0]

```
Total params: 15,850,002
Trainable params: 15,849,810
Non-trainable params: 192
```

In [ ]:
```
input_train = [  padded_essay_train,
                 school_state_x_train,
                 grade_x_train,
                 teacher_prefix_x_train,
                 categories_x_train,
                 subcategories_x_train,
                 numerical_features_x_train,
                ]

input_val = [  padded_essay_cv,
               school_state_x_val,
               grade_x_val,
               teacher_prefix_x_val,
               categories_x_val,
               subcategories_x_val,
               numerical_features_x_val,
             ]

input_test = [  padded_essay_test,
                school_state_x_test,
                grade_x_test,
                teacher_prefix_x_test,
                categories_x_test,
                subcategories_x_test,
                numerical_features_x_test,
                ]
```

In [ ]:
```
for i in input_train:
  print(i.shape)

y_train.shape, y_cv.shape, y_test.shape
```

```
(69918, 2527)
(69918,)
(69918,)
(69918,)
(69918, 9)
(69918, 30)
(69918, 2)
```

Out[ ]:
```
((69918,), (17480,), (21850,))
```

In [ ]:
```
from keras.utils import plot_model
plot_model(model1, to_file='model1.png')
```

Out[ ]:

In [ ]:
```
batch_size=400
epochs = 10
```

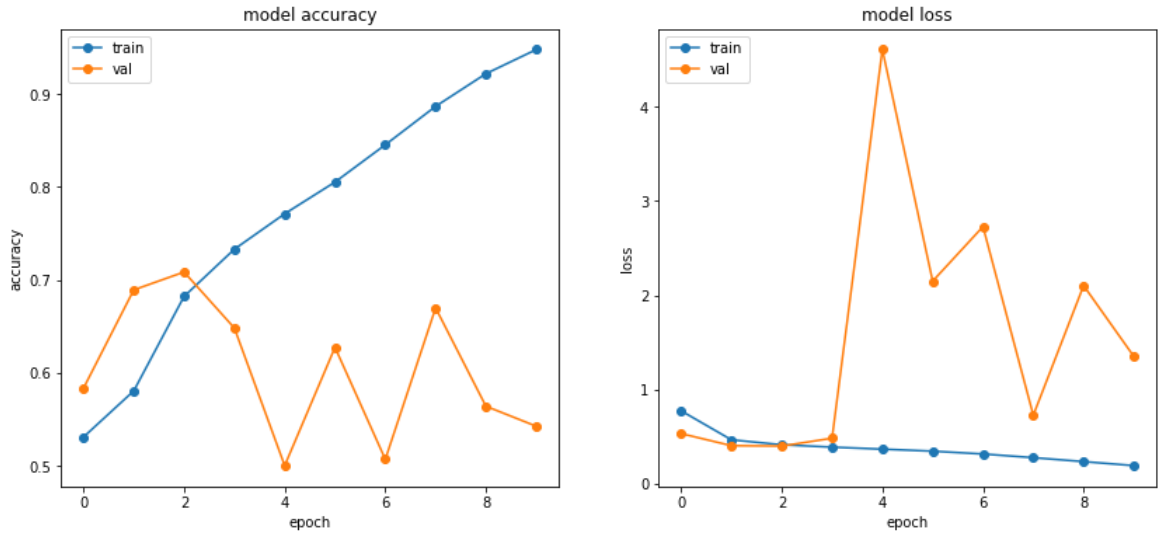
In [ ]:
```
# https://keras.io/callbacks/

tensorboard_callback = TensorBoard(log_dir='./logs', histogram_freq=1, batch_size=batch_size, write_graph=True,
                          write_grads=False, write_images=True, embeddings_freq=0, update_freq='epoch')

filepath="Model1-weights-improvement-{epoch:02d}-{val_loss:.2f}.hdf5"
# I'm saving only best weights among the all epochs
model_checkpoint_callback = ModelCheckpoint(filepath, monitor='val_loss', verbose=0,
                                            save_best_only=True, save_weights_only=False, mode='auto', period=1)
early_stopping_callback =  EarlyStopping(monitor="val_loss", min_delta=0, patience=3, verbose=0, mode="auto", basel
ine=None)
```

In [ ]:
```
%tensorboard --logdir logs
```

In [ ]:
```
history1 = model1.fit(input_train, y_train ,batch_size=batch_size,
                      epochs=epochs, validation_data=(input_val, y_cv),
                     callbacks=[tensorboard_callback, model_checkpoint_callback])
```

```
Train on 69918 samples, validate on 17480 samples
Epoch 1/10
69918/69918 [==============================] - 1376s 20ms/step - loss: 0.7708 - aucroc: 0.5307 - val_loss: 0.5289
- val_aucroc: 0.5833
Epoch 2/10
69918/69918 [==============================] - 1367s 20ms/step - loss: 0.4625 - aucroc: 0.5805 - val_loss: 0.4004
- val_aucroc: 0.6893
Epoch 3/10
69918/69918 [==============================] - 1369s 20ms/step - loss: 0.4103 - aucroc: 0.6822 - val_loss: 0.3964
- val_aucroc: 0.7085
Epoch 4/10
69918/69918 [==============================] - 1373s 20ms/step - loss: 0.3848 - aucroc: 0.7331 - val_loss: 0.4796
- val_aucroc: 0.6482
Epoch 5/10
69918/69918 [==============================] - 1371s 20ms/step - loss: 0.3636 - aucroc: 0.7713 - val_loss: 4.6103
- val_aucroc: 0.5000
Epoch 6/10
69918/69918 [==============================] - 1373s 20ms/step - loss: 0.3421 - aucroc: 0.8054 - val_loss: 2.1490
- val_aucroc: 0.6273
Epoch 7/10
69918/69918 [==============================] - 1375s 20ms/step - loss: 0.3120 - aucroc: 0.8456 - val_loss: 2.7284
- val_aucroc: 0.5068
Epoch 8/10
69918/69918 [==============================] - 1371s 20ms/step - loss: 0.2730 - aucroc: 0.8870 - val_loss: 0.7239
- val_aucroc: 0.6698
Epoch 9/10
69918/69918 [==============================] - 1375s 20ms/step - loss: 0.2310 - aucroc: 0.9222 - val_loss: 2.1014
- val_aucroc: 0.5639
Epoch 10/10
69918/69918 [==============================] - 1371s 20ms/step - loss: 0.1886 - aucroc: 0.9482 - val_loss: 1.3478
- val_aucroc: 0.5424
```

In [ ]:
```
#https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/
fig = plt.figure(figsize=(14,6))

# summarize history for accuracy
plt.subplot(1,2,1)
plt.title('model accuracy')
plt.plot(history1.history['aucroc'], marker='o')
plt.plot(history1.history['val_aucroc'], marker='o')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')

# summarize history for loss
plt.subplot(1,2,2)
plt.title('model loss')
plt.plot(history1.history['loss'], marker='o')
plt.plot(history1.history['val_loss'], marker='o')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



In [ ]:

In [ ]:

Model-2

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

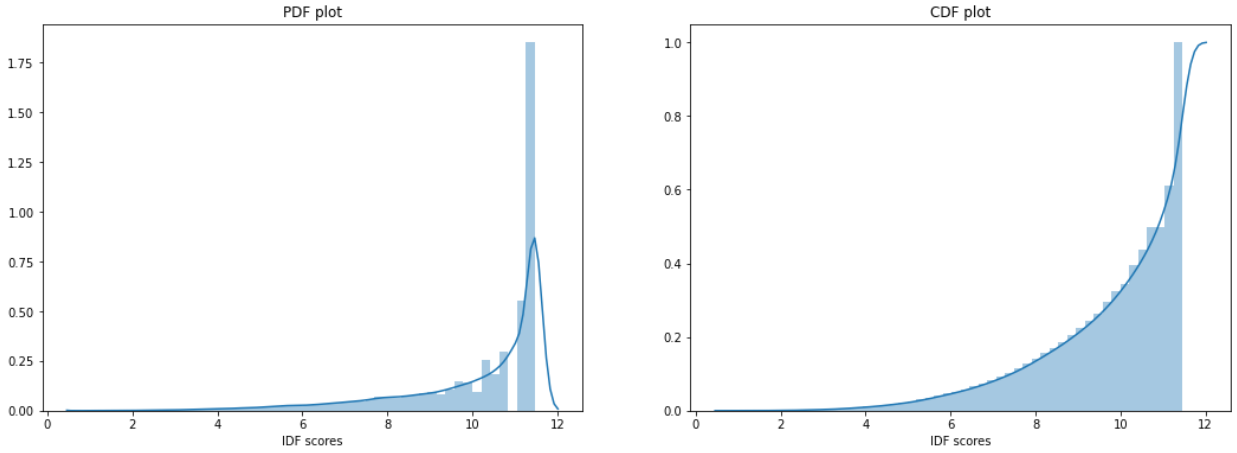
1. Train the TF-IDF on the Train data
2. Get the idf value for each word we have in the train data.
3. Remove the low idf value and high idf value words from our data. Do some analysis on the Idf values and based on those values choose the low and high threshold value. Because very frequent words and very very rare words don't give much information. (you can plot a box plots and take only the idf scores within IQR range and corresponding words)
4. Train the LSTM after removing the Low and High idf value words. (In model-1 Train on total data but in Model-2 train on data after removing some words based on IDF values)

In [ ]:
```
tfidf_vectorizer = TfidfVectorizer()
tfidf_vectorizer.fit(X_train['essay'])

tfidf_vectorizer.idf_.shape,  max(tfidf_vectorizer.idf_), min(tfidf_vectorizer.idf_)
```

Out[ ]:
```
((47244,), 11.4619455276077, 1.0073785181584354)
```

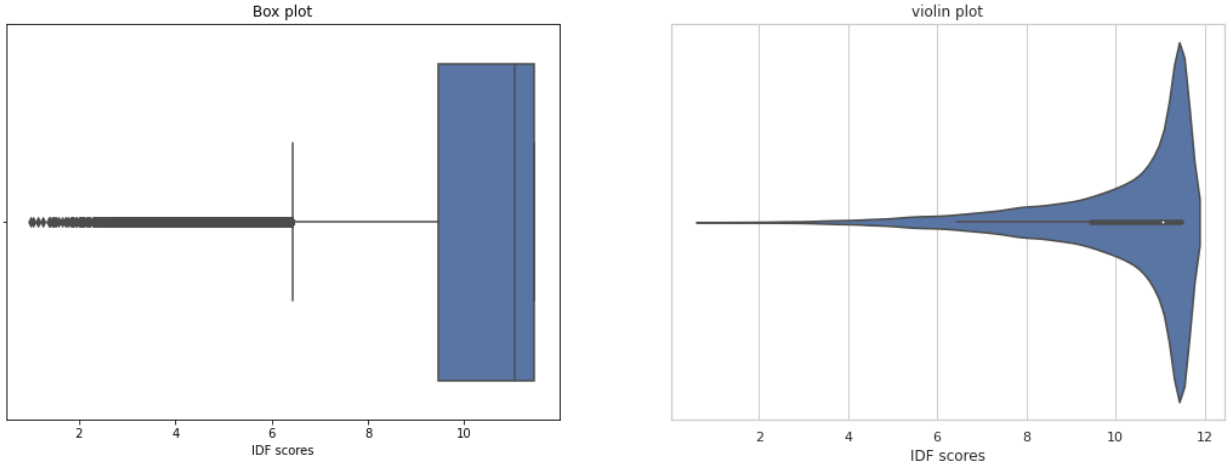
In [ ]:
```
fig = plt.figure(figsize=(18,6))

# PDF plots
plt.subplot(1,2,1)
plt.title('PDF plot')
ax= sns.distplot(tfidf_vectorizer.idf_)
plt.xlabel('IDF scores')

# CDF plots
plt.subplot(1,2,2)
plt.title('CDF plot')
kwargs = {'cumulative': True}
ax= sns.distplot(tfidf_vectorizer.idf_, hist_kws=kwargs, kde_kws=kwargs)
plt.xlabel('IDF scores')
```

Out[ ]:
```
Text(0.5, 0, 'IDF scores')
```



In [ ]:
```
fig = plt.figure(figsize=(18,6))
# Box plot
plt.subplot(1,2,1)
plt.title('Box plot')
sns.set(style="whitegrid")
ax = sns.boxplot(tfidf_vectorizer.idf_)
plt.xlabel('IDF scores')

# Violin plot
plt.subplot(1,2,2)
plt.title('violin plot')
ax= sns.violinplot(tfidf_vectorizer.idf_)
plt.xlabel('IDF scores')
```

Out[ ]:
```
Text(0.5, 0, 'IDF scores')
```



In [ ]:
```
print("25th percentile : {0} ".format(np.percentile(tfidf_vectorizer.idf_, 25)))
print("75th percentile : {0} ".format(np.percentile(tfidf_vectorizer.idf_, 75)))
```

```
25th percentile : 9.447042507065435
75th percentile : 11.4619455276077
```

In [ ]:
```
# Setting threshold as 4 - 11
min_threshold = 4
max_threshold = 11
print("{0:.2f}% of words present with in {1} - {2} thresholds".format(
(100*len(list(filter(lambda x: x>min_threshold and x<max_threshold+0.01 , tfidf_vectorizer.idf_)))) / tfidf_vectori
zer.idf_.shape[0],
 min_threshold, max_threshold))
```

```
48.78% of words present with in 4 - 11 thresholds
```

Filter essays with in threshld IDF range

In [ ]:
```
idf_df = pd.DataFrame({"words": tfidf_vectorizer.get_feature_names() , "IDF":tfidf_vectorizer.idf_})
idf_df.sort_values(by=['IDF'], inplace=True)

# Filter dataframe with in threshold range 4 - 11
idf_df = idf_df[(idf_df['IDF']>min_threshold-0.01) & (idf_df['IDF']<max_threshold+0.01) ]
idf_df
```

Out[ ]:

	words	IDF
14871	equipment	3.993432
29010	number	3.993432
7114	care	3.993718
2479	allowing	3.994289
8056	choice	3.995146
...	...	...
9536	congregate	10.768798
8900	collographs	10.768798
15721	extravaganza	10.768798
8235	cinnamon	10.768798
17731	gameboard	10.768798

23053 rows × 2 columns

In [ ]:
```
from datetime import datetime as dt

def remove_low_high_idf_words(essay):
    essay=essay.split()
    return " ".join(list(filter(lambda x:x in idf_df["words"].values, essay)))

start_time = dt.now()
X_train['filtered_essay'] = X_train["essay"].apply(remove_low_high_idf_words)
print(".", end="")
X_cv['filtered_essay'] = X_cv["essay"].apply(remove_low_high_idf_words)
print(".", end="")
X_test["filtered_essay"] = X_test["essay"].apply(remove_low_high_idf_words)
print(".", end="")
print(dt.now() - start_time)
```

```
...2:35:28.083786
```

In [ ]:
```
import pickle

pickle.dump(X_train['filtered_essay'],open(b"filtered_essay_train.pkl","wb"))
pickle.dump(X_cv['filtered_essay'],open(b"filtered_essay_val.pkl","wb"))
pickle.dump(X_test['filtered_essay'],open(b"filtered_essay_test.pkl","wb"))
```

In [ ]:
```
#load those pickle files
import pickle
X_train['filtered_essay'] = pickle.load(open(b"/content/filtered_essay_train.pkl","rb"))
X_cv['filtered_essay'] = pickle.load(open(b"/content/filtered_essay_val.pkl","rb"))
X_test['filtered_essay'] = pickle.load(open(b"/content/filtered_essay_test.pkl","rb"))
```

In [ ]:
```
X_train.columns
```

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

In [ ]:
```

```

Filtered essay Layer

In [ ]:
```
"""
https://towardsdatascience.com/text-classification-in-keras-part-2-how-to-use-the-keras-tokenizer-word-representations-fd571674df23
https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
"""
from keras.preprocessing.text import Tokenizer
#max_words = 10000
tokenizer = keras.preprocessing.text.Tokenizer( lower=True, split=' ', char_level=False, oov_token=None, document_count=0)
#tokenizer.fit_on_texts(idf_df["words"].values)
# (or)
tokenizer.fit_on_texts(X_train["filtered_essay"].tolist()) #coz filtered_essay has the words that are id idf_df['words']

filtered_essay_sequences_train = tokenizer.texts_to_sequences(X_train['filtered_essay'])
filtered_essay_sequences_val = tokenizer.texts_to_sequences(X_cv['filtered_essay'])
filtered_essay_sequences_test = tokenizer.texts_to_sequences(X_test['filtered_essay'])

########################################################################

"""
https://keras.io/preprocessing/sequence/#pad_sequences
"""
from keras.preprocessing.sequence import pad_sequences
max_length=0;
for filtered_essay in X_train["filtered_essay"]:
    temp =len(filtered_essay)
    max_length= temp if temp > max_length else max_length

padded_filtered_essay_train = pad_sequences(filtered_essay_sequences_train, maxlen=max_length, dtype='int32', padding='post', truncating='post', value=0.0)
padded_filtered_essay_val = pad_sequences(filtered_essay_sequences_val, maxlen=max_length, dtype='int32', padding='post', truncating='post', value=0.0)
padded_filtered_essay_test = pad_sequences(filtered_essay_sequences_test, maxlen=max_length, dtype='int32', padding='post', truncating='post', value=0.0)
```

Loading Glove vectors

In [ ]:
```
import pickle
with open("drive/My Drive/AAIC/DONORS_CHOOSE/Assignments_DonorsChoose_2018/glove_vectors","rb") as file:
    embeddings = pickle.load(file)

# vocab_size + 1 since, 0 is not considered by tokenizer.
vocab_size = len(tokenizer.word_index)+1

# create a weight matrix for words in training docs
embedding_matrix = np.zeros((vocab_size, 300))

for word, i in tqdm(tokenizer.word_index.items()):
                embedding_vector = embeddings.get(word)
                if embedding_vector is not None:
                            embedding_matrix[i] = embedding_vector

embedding_matrix.shape
```

```
100%|██████████| 23053/23053 [00:00<00:00, 360109.83it/s]
```

Out[ ]:
```
(23054, 300)
```

In [ ]:
```
"""
https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-layer-work
"""
from keras.engine.input_layer import Input
from keras.layers import Embedding, Flatten, LSTM, Dense
from keras.initializers import Constant

filtered_essay_input_layer = Input(shape=(max_length,), name="filtered_essay_i/p")#, batch_shape=(1000,max_length))
filtered_essay_embedding_layer = Embedding(vocab_size, 300, \
                                embeddings_initializer=Constant(embedding_matrix),\
                                input_length=max_length)(filtered_essay_input_layer)

filtered_essay_embedding_layer.trainable = False

filtered_essay_lstm_layer = LSTM(10, return_sequences=True)(filtered_essay_embedding_layer)
# https://github.com/keras-team/keras/issues/7403
filtered_essay_flatten = Flatten()(filtered_essay_lstm_layer)
```

In [ ]:
```
max_length, filtered_essay_input_layer.shape , filtered_essay_embedding_layer.shape , filtered_essay_lstm_layer.sha
pe , filtered_essay_flatten.shape
```

Out[ ]:
```
(1496,
 TensorShape([None, 1496]),
 TensorShape([None, 1496, 300]),
 TensorShape([None, 1496, 10]),
 TensorShape([None, None]))
```

In [ ]:
```
from keras.layers import concatenate, Dropout, BatchNormalization
from keras.models import Model

concatenate_layer = concatenate([filtered_essay_flatten, school_state_flatten, grade_flatten, categories_flatten,
                     subcategories_flatten, teacher_prefix_flatten, numerical_dense])
```

In [ ]:
```
dense_layer0 = Dense(64, activation="relu")(concatenate_layer)

BN_layer0 = BatchNormalization()(dense_layer0)

dropout_layer0 = Dropout(0.3)(BN_layer0)

dense_layer1 = Dense(32, activation="relu")(dropout_layer0)

BN_layer1 = BatchNormalization()(dense_layer1)

dropout_layer1 = Dropout(0.2)(dense_layer1)

dense_layer2 = Dense(16, activation="relu")(dropout_layer1)

final_layer = Dense(1, activation='sigmoid', name="o/p_layer")(dense_layer2)
```

In [ ]:
```
model2= Model(inputs=[filtered_essay_input_layer, school_state_input_layer, teacher_prefix_input_layer, grade_input
_layer,
                      categories_input_layer, subcategories_input_layer, numerical_input_layer],
              outputs=[final_layer])
```

In [ ]:
```
# https://stackoverflow.com/questions/41032551/how-to-compute-receiving-operating-characteristic-roc-and-auc-in-ker
as
import tensorflow as tf
from sklearn.metrics import roc_auc_score

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

In [ ]:
```
model2.compile(optimizer='adam', loss='binary_crossentropy', metrics = [aucroc])
model2.summary()
```

```
Model: "model_1"
```

Layer (type)	Output Shape	Param #	Connected to
filtered_essay_i/p (InputLayer)	(None, 1496)	0	
embedding_7 (Embedding)	(None, 1496, 300)	6916200	filtered_essay_i/p[0][0]
school_state (InputLayer)	(None, 1)	0	
grade_i/p (InputLayer)	(None, 1)	0	
categories_i/p (InputLayer)	(None, 9)	0	
subcategories_i/p (InputLayer)	(None, 30)	0	
teacher_prefix_i/p (InputLayer)	(None, 1)	0	
lstm_2 (LSTM)	(None, 1496, 10)	12440	embedding_7[0][0]
embedding_2 (Embedding)	(None, 1, 26)	1352	school_state[0][0]
embedding_4 (Embedding)	(None, 1, 2)	10	grade_i/p[0][0]
embedding_5 (Embedding)	(None, 9, 5)	45	categories_i/p[0][0]
embedding_6 (Embedding)	(None, 30, 15)	450	subcategories_i/p[0][0]
embedding_3 (Embedding)	(None, 1, 3)	18	teacher_prefix_i/p[0][0]
numerical_layer (InputLayer)	(None, 2)	0	
flatten_7 (Flatten)	(None, 14960)	0	lstm_2[0][0]
flatten_2 (Flatten)	(None, 26)	0	embedding_2[0][0]
flatten_4 (Flatten)	(None, 2)	0	embedding_4[0][0]
flatten_5 (Flatten)	(None, 45)	0	embedding_5[0][0]
flatten_6 (Flatten)	(None, 450)	0	embedding_6[0][0]
flatten_3 (Flatten)	(None, 3)	0	embedding_3[0][0]
dense_1 (Dense)	(None, 10)	30	numerical_layer[0][0]
concatenate_1 (Concatenate)	(None, 15496)	0	flatten_7[0][0] flatten_2[0][0] flatten_4[0][0] flatten_5[0][0] flatten_6[0][0] flatten_3[0][0] dense_1[0][0]
dense_10 (Dense)	(None, 64)	991808	concatenate_1[0][0]
batch_normalization_7 (BatchNor	(None, 64)	256	dense_10[0][0]
dropout_7 (Dropout)	(None, 64)	0	batch_normalization_7[0][0]
dense_11 (Dense)	(None, 32)	2080	dropout_7[0][0]
dropout_8 (Dropout)	(None, 32)	0	dense_11[0][0]
dense_12 (Dense)	(None, 16)	528	dropout_8[0][0]
o/p_layer (Dense)	(None, 1)	17	dense_12[0][0]

```
Total params: 7,925,234
Trainable params: 7,925,106
Non-trainable params: 128
```

In [ ]:
```
input_train = [  padded_filtered_essay_train,
                 school_state_x_train,
                 grade_x_train,
                 teacher_prefix_x_train,
                 categories_x_train,
                 subcategories_x_train,
                 numerical_features_x_train,
                ]

input_val = [  padded_filtered_essay_val,
               school_state_x_val,
               grade_x_val,
               teacher_prefix_x_val,
               categories_x_val,
               subcategories_x_val,
               numerical_features_x_val,
             ]

input_test = [  padded_filtered_essay_test,
                school_state_x_test,
                grade_x_test,
                teacher_prefix_x_test,
                categories_x_test,
                subcategories_x_test,
                numerical_features_x_test,
                ]
```

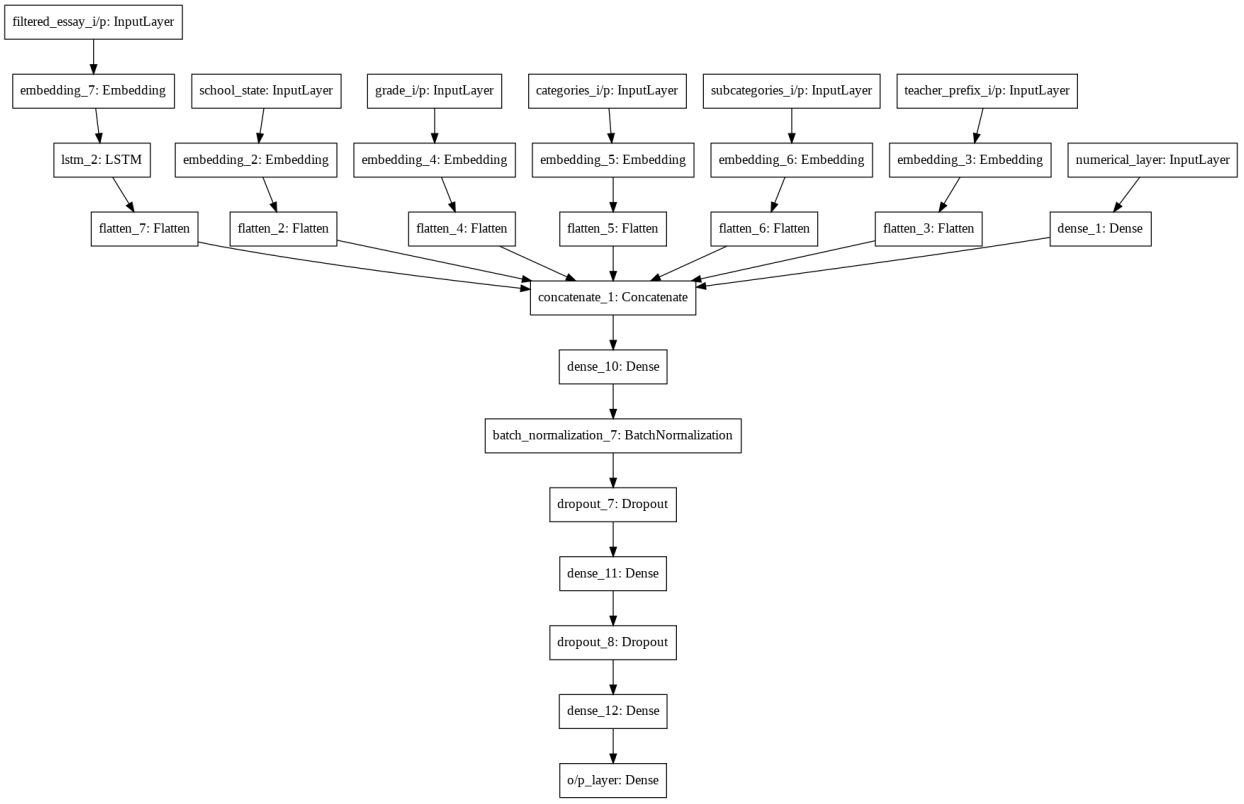
In [ ]:
```
for i in input_train:
    print(i.shape)

y_train.shape, y_cv.shape, y_test.shape
```

```
(69918, 1496)
(69918,)
(69918,)
(69918,)
(69918, 9)
(69918, 30)
(69918, 2)
```

Out[ ]:
```
((69918,), (17480,), (21850,))
```

In [ ]:
```
from keras.utils import plot_model
plot_model(model2, to_file='model2.png')
```

Out[ ]:



In [ ]:
```
batch_size=400
epochs = 20
```

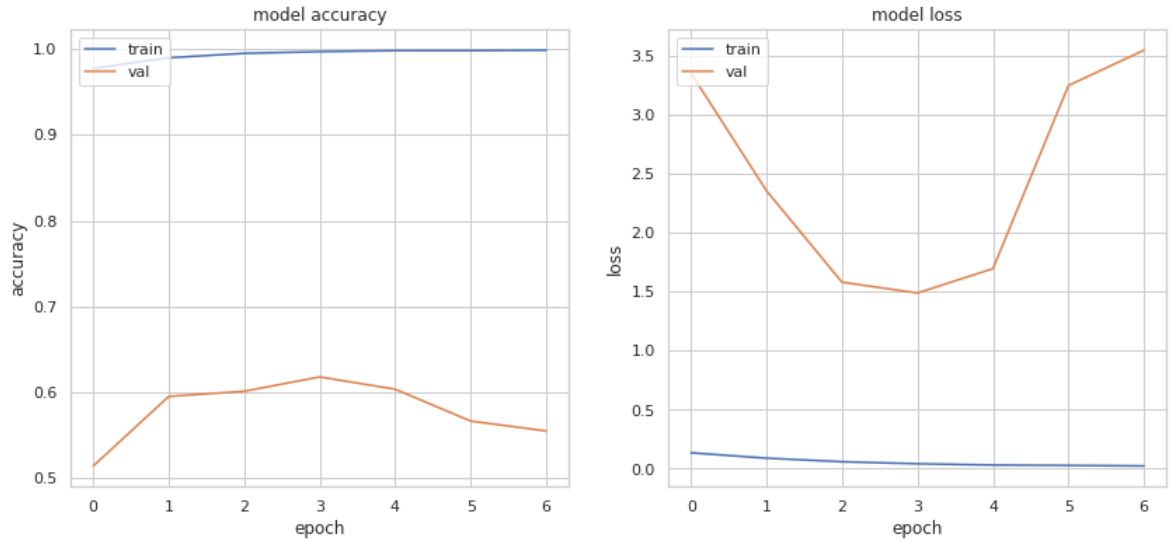
In [ ]:
```
# https://keras.io/callbacks/

tensorboard_callback = TensorBoard(log_dir='./logs', histogram_freq=1, batch_size=batch_size, write_graph=True,
                          write_grads=False, write_images=True, embeddings_freq=0, update_freq='epoch')

filepath="Model2-weights-improvement-{epoch:02d}-{val_loss:.2f}.hdf5"
# I'm saving only best weights among the all epochs
model_checkpoint_callback = ModelCheckpoint(filepath, monitor='val_loss', verbose=0,
                                            save_best_only=True, save_weights_only=False, mode='auto', period=1)
```

In [ ]:
```
%tensorboard --logdir logs
```

In [ ]:
```
history2 = model2.fit(input_train, y_train ,batch_size=batch_size,
                        epochs=epochs, validation_data=(input_val, y_cv),
                      callbacks=[tensorboard_callback, model_checkpoint_callback, early_stopping_callback])
```

```
Train on 69918 samples, validate on 17480 samples
Epoch 1/20
69918/69918 [==============================] - 808s 12ms/step - loss: 0.1353 - aucroc: 0.9777 - val_loss: 3.3548 -
val_aucroc: 0.5137
Epoch 2/20
69918/69918 [==============================] - 810s 12ms/step - loss: 0.0899 - aucroc: 0.9901 - val_loss: 2.3584 -
val_aucroc: 0.5949
Epoch 3/20
69918/69918 [==============================] - 806s 12ms/step - loss: 0.0597 - aucroc: 0.9953 - val_loss: 1.5830 -
val_aucroc: 0.6009
Epoch 4/20
69918/69918 [==============================] - 810s 12ms/step - loss: 0.0426 - aucroc: 0.9973 - val_loss: 1.4914 -
val_aucroc: 0.6176
Epoch 5/20
69918/69918 [==============================] - 802s 11ms/step - loss: 0.0316 - aucroc: 0.9985 - val_loss: 1.6983 -
val_aucroc: 0.6033
Epoch 6/20
69918/69918 [==============================] - 805s 12ms/step - loss: 0.0283 - aucroc: 0.9985 - val_loss: 3.2518 -
val_aucroc: 0.5661
Epoch 7/20
69918/69918 [==============================] - 820s 12ms/step - loss: 0.0239 - aucroc: 0.9989 - val_loss: 3.5514 -
val_aucroc: 0.5546
```

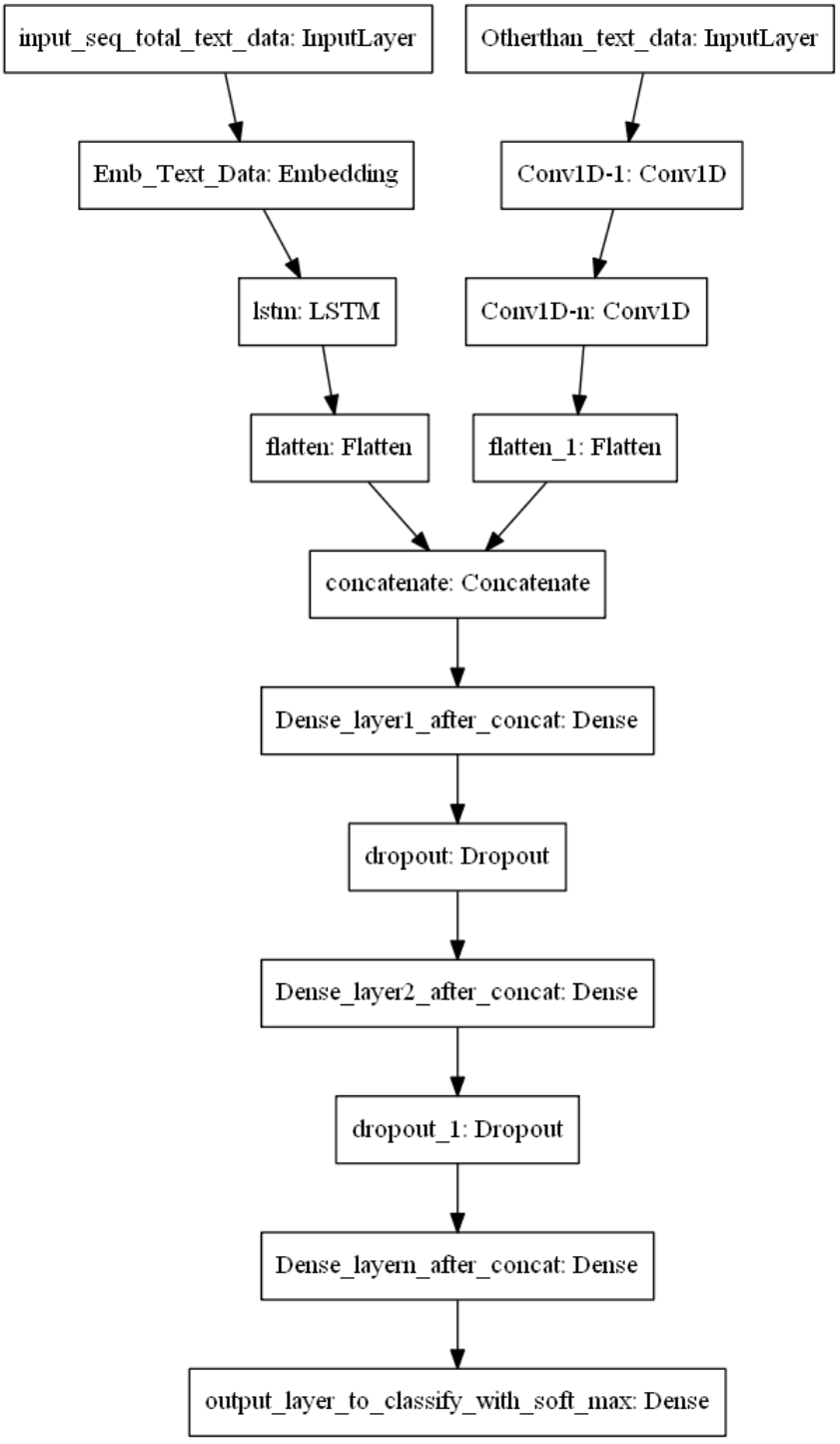
In [ ]:
```
#https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/
fig = plt.figure(figsize=(14,6))

# summarize history for accuracy
plt.subplot(1,2,1)
plt.title('model accuracy')
plt.plot(history2.history['aucroc'])
plt.plot(history2.history['val_aucroc'])
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')

# summarize history for loss
plt.subplot(1,2,2)
plt.title('model loss')
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



In [ ]:
```
from keras.models import load_model
model2.save("model2.hd5")
```

In [ ]:
```

```

Model-3



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

- **input_seq_total_text_data**:

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

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

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

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

- **Other_than_text_data**:

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

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

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

</pre>

In [ ]:

input_seq_total_text_data:

essay

In [ ]:
```
"""
https://towardsdatascience.com/text-classification-in-keras-part-2-how-to-use-the-keras-tokenizer-word-representations-fd571674df23
https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
"""
from keras.preprocessing.text import Tokenizer
#max_words = 10000
tokenizer = keras.preprocessing.text.Tokenizer( lower=True, split=' ', char_level=False, oov_token=None, document_count=0)
tokenizer.fit_on_texts(X_train["essay"])

essay_sequences_train = tokenizer.texts_to_sequences(X_train['essay'])
essay_sequences_val = tokenizer.texts_to_sequences(X_cv['essay'])
essay_sequences_test = tokenizer.texts_to_sequences(X_test['essay'])

########################################################################

"""
https://keras.io/preprocessing/sequence/#pad_sequences
"""
from keras.preprocessing.sequence import pad_sequences
max_length=0;
for essay in X_train["essay"]:
    temp =len(essay)
    max_length= temp if temp > max_length else max_length

padded_essay_train = pad_sequences(essay_sequences_train, maxlen=max_length, dtype='int32', padding='post', truncating='post', value=0.0)
padded_essay_val = pad_sequences(essay_sequences_val, maxlen=max_length, dtype='int32', padding='post', truncating='post', value=0.0)
padded_essay_test = pad_sequences(essay_sequences_test, maxlen=max_length, dtype='int32', padding='post', truncating='post', value=0.0)
```

In [ ]:
```
import pickle
with open("drive/My Drive/AAIC/DONORS_CHOOSE/Assignments_DonorsChoose_2018/glove_vectors","rb") as file:
    embeddings = pickle.load(file)

# vocab_size + 1 since, 0 is not considered by tokenizer.
vocab_size = len(tokenizer.word_index)+1

# create a weight matrix for words in training docs
embedding_matrix = np.zeros((vocab_size, 300))

for word, i in tqdm(tokenizer.word_index.items()):
                embedding_vector = embeddings.get(word)
                if embedding_vector is not None:
                                embedding_matrix[i] = embedding_vector

embedding_matrix.shape
```

```
100%|██████████| 47280/47280 [00:00<00:00, 364032.48it/s]
```

Out[ ]:
```
(47281, 300)
```

In [ ]:
```
"""
https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-layer-work
"""
from keras.engine.input_layer import Input
from keras.layers import Embedding, Flatten, LSTM, Dense
from keras.initializers import Constant

essay_input_layer = Input(shape=(max_length,), name="essay_i/p")#, batch_shape=(1000,max_length))
essay_embedding_layer = Embedding(vocab_size, 300, \
                                  embeddings_initializer=Constant(embedding_matrix),\
                                  input_length=max_length)(essay_input_layer)
essay_embedding_layer.trainable=False
essay_lstm_layer = LSTM(10, return_sequences=True)(essay_embedding_layer)
# https://github.com/keras-team/keras/issues/7403
essay_flatten = Flatten()(essay_lstm_layer)
```

In [ ]:
```
max_length, essay_input_layer.shape , essay_embedding_layer.shape , essay_lstm_layer.shape , essay_flatten.shape
```

Out[ ]:
```
(2657,
 TensorShape([None, 2657]),
 TensorShape([None, 2657, 300]),
 TensorShape([None, 2657, 10]),
 TensorShape([None, None]))
```

Other_than_text_data

OHE categorical_features

In [ ]:
```
from sklearn.feature_extraction.text import CountVectorizer

def vectorize_cat_data(x_train, x_val, x_test, col_name):
    #filling nan with NAN
    x_train[col_name] = x_train[col_name].fillna('NAN')
    x_val[col_name] = x_val[col_name].fillna('NAN')
    x_test[col_name] = x_test[col_name].fillna('NAN')

    vectorizer= CountVectorizer(binary=True)
    vectorizer.fit(x_train[col_name].values)

    col_name_one_hot = vectorizer.transform(x_train[col_name].values)
    # store the feature names of all categorical columns
    print(vectorizer.get_feature_names())
    print("Shape of matrix after one hot encoding ",col_name_one_hot.shape,"\n")

    return vectorizer , col_name_one_hot, vectorizer.transform(x_val[col_name].values), vectorizer.transform(x_test
[col_name].values)
```

In [ ]:
```
#OHE clean_categories
categories_vectorizer, categories_x_train, categories_x_val, categories_x_test = \
                                                    vectorize_cat_data(X_train, X_cv, X_test, 'clean_categories')
#OHE clean_subcategories
subcategories_vectorizer, subcategories_x_train, subcategories_x_val, subcategories_x_test = \
                                                    vectorize_cat_data(X_train, X_cv, X_test, 'clean_subcategories'
)
#OHE school_state
school_state_vectorizer, school_state_x_train, school_state_x_val, school_state_x_test = \
                                                    vectorize_cat_data(X_train, X_cv, X_test, 'school_state')
#OHE teacher_prefix
teacher_prefix_vectorizer, teacher_prefix_x_train, teacher_prefix_x_val, teacher_prefix_x_test = \
                                                    vectorize_cat_data(X_train, X_cv, X_test, 'school_state')
#OHE project_grade_category
grade_vectorizer, grade_x_train, grade_prefix_x_val, grade_prefix_x_test = \
                                                    vectorize_cat_data(X_train, X_cv, X_test, 'project_grade_catego
ry')
```

```
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_
arts', 'specialneeds', 'warmth']
Shape of matrix after one hot encoding  (69918, 9)

['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityserv
ice', 'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'for
eignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literatu
re_writing', 'mathematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'social
sciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
Shape of matrix after one hot encoding  (69918, 30)

['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la',
'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or',
'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
Shape of matrix after one hot encoding  (69918, 51)

['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la',
'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or',
'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
Shape of matrix after one hot encoding  (69918, 51)

['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
Shape of matrix after one hot encoding  (69918, 4)
```

In [ ]:
```
type(grade_x_train)
```

Out[ ]:
```
scipy.sparse.csr.csr_matrix
```

Numerical_features

In [ ]:
```
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.h
tml

from sklearn.preprocessing import StandardScaler, Normalizer
from scipy import sparse
#This function return the normalized data.
def standardize_data(x_train, x_val, x_test, column):
    scalar = StandardScaler()
    scalar.fit(x_train[column].values.reshape(-1,1))
    std_scalar_x_train = scalar.transform(x_train[column].values.reshape(-1,1))
    std_scalar_x_val = scalar.transform(x_val[column].values.reshape(-1,1))
    std_scalar_x_test = scalar.transform(x_test[column].values.reshape(-1,1))
    print(std_scalar_x_train.shape)
    return std_scalar_x_train, std_scalar_x_val, std_scalar_x_test
    #return sparse.csr_matrix(std_scalar_x_train), sparse.csr_matrix(std_scalar_x_val), sparse.csr_matrix(std_scala
r_x_test)
```

In [ ]:
```
price_x_train, price_x_val, price_x_test = \
                                        standardize_data(X_train, X_cv, X_test, "price")

teacher_prev_projects_x_train, teacher_prev_projects_x_val, teacher_prev_projects_x_test = \
                                        standardize_data(X_train, X_cv, X_test, "teacher_number_of_previously_poste
d_projects")
```

```
(69918, 1)
(69918, 1)
```

In [ ]:
```
from scipy.sparse import hstack
other_than_text_x_train = hstack((categories_x_train, subcategories_x_train, school_state_x_train, teacher_prefix_x
_train, grade_x_train, price_x_train, teacher_prev_projects_x_train))

other_than_text_x_val = hstack((categories_x_val, subcategories_x_val, school_state_x_val, teacher_prefix_x_val, gr
ade_prefix_x_val, price_x_val, teacher_prev_projects_x_val))

other_than_text_x_test = hstack((categories_x_test, subcategories_x_test, school_state_x_test, teacher_prefix_x_tes
t, grade_prefix_x_test, price_x_test, teacher_prev_projects_x_test))
```

In [ ]:
```
# these should be having 3 dimensions as we are using conv1D filters,
# so change the dimension to 3D (Ex: (x,x,1) like this)
# https://stackoverflow.com/questions/17394882/how-can-i-add-new-dimensions-to-a-numpy-array

other_than_text_x_train = other_than_text_x_train.todense()[..., None]
other_than_text_x_val = other_than_text_x_val.todense()[..., None]
other_than_text_x_test = other_than_text_x_test.todense()[..., None]

print(other_than_text_x_train.shape , other_than_text_x_val.shape, other_than_text_x_test.shape)
```

```
(69918, 147, 1) (17480, 147, 1) (21850, 147, 1)
```

In [ ]:
```
from keras.layers import Conv1D, Dropout

# https://stackoverflow.com/questions/49840968/valueerror-input-0-is-incompatible-with-layer-conv1d-1-expected-ndim
-3-found
other_than_text_input_layer = Input(shape=(other_than_text_x_train.shape[1],1) , name="other_than_text_i/p")

conv1d_layer0 = Conv1D(filters=128, kernel_size=3, padding="valid",
                       activation="relu", strides=1, kernel_initializer="he_normal")(other_than_text_input_layer)

conv1d_layer1 = Conv1D(filters=64, kernel_size=3, padding="valid",
                       activation="relu", strides=1, kernel_initializer="he_normal")(conv1d_layer0)

flatten1 = Flatten()(conv1d_layer1)

concatenate_layer = concatenate([essay_flatten, flatten1])

dense_layer0 = Dense(128,activation="relu",kernel_initializer="he_normal")(concatenate_layer)

dropout_layer0 = Dropout(0.5)(dense_layer0)

dense_layer1 = Dense(64,activation="relu",kernel_initializer="he_normal")(dropout_layer0)

dropout_layer1 = Dropout(0.5)(dense_layer1)

dense_layer2 = Dense(32,activation="relu",kernel_initializer="he_normal")(dropout_layer1)

output_layer = Dense(1, activation='sigmoid', name='output')(dense_layer2)
```

In [ ]:
```
from keras.models import Model
from keras.optimizers import Adam
model3= Model(inputs=[essay_input_layer, other_than_text_input_layer],
              outputs=[output_layer])
adam_optimizer = Adam(learning_rate=0.005, beta_1=0.9, beta_2=0.999)
model3.compile(optimizer=adam_optimizer, loss='binary_crossentropy', metrics = [aucroc])

model3.summary()
```

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
essay_i/p (InputLayer)          (None, 2657)         0
__________________________________________________________________________________________________
other_than_text_i/p (InputLayer (None, 147, 1)       0
__________________________________________________________________________________________________
embedding_8 (Embedding)         (None, 2657, 300)    14184300    essay_i/p[0][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 145, 128)     512         other_than_text_i/p[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                   (None, 2657, 10)     12440       embedding_8[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 143, 64)      24640       conv1d_1[0][0]
__________________________________________________________________________________________________
flatten_8 (Flatten)             (None, 26570)        0           lstm_3[0][0]
__________________________________________________________________________________________________
flatten_9 (Flatten)             (None, 9152)         0           conv1d_2[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 35722)        0           flatten_8[0][0]
                                                                 flatten_9[0][0]
__________________________________________________________________________________________________
dense_13 (Dense)                (None, 128)          4572544     concatenate_2[0][0]
__________________________________________________________________________________________________
dropout_9 (Dropout)             (None, 128)          0           dense_13[0][0]
__________________________________________________________________________________________________
dense_14 (Dense)                (None, 64)           8256        dropout_9[0][0]
__________________________________________________________________________________________________
dropout_10 (Dropout)            (None, 64)           0           dense_14[0][0]
__________________________________________________________________________________________________
dense_15 (Dense)                (None, 32)           2080        dropout_10[0][0]
__________________________________________________________________________________________________
output (Dense)                  (None, 1)            33          dense_15[0][0]
==================================================================================================
Total params: 18,804,805
Trainable params: 18,804,805
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [ ]:
```
# https://stackoverflow.com/questions/41032551/how-to-compute-receiving-operating-characteristic-roc-and-auc-in-keras
import tensorflow as tf
from sklearn.metrics import roc_auc_score

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

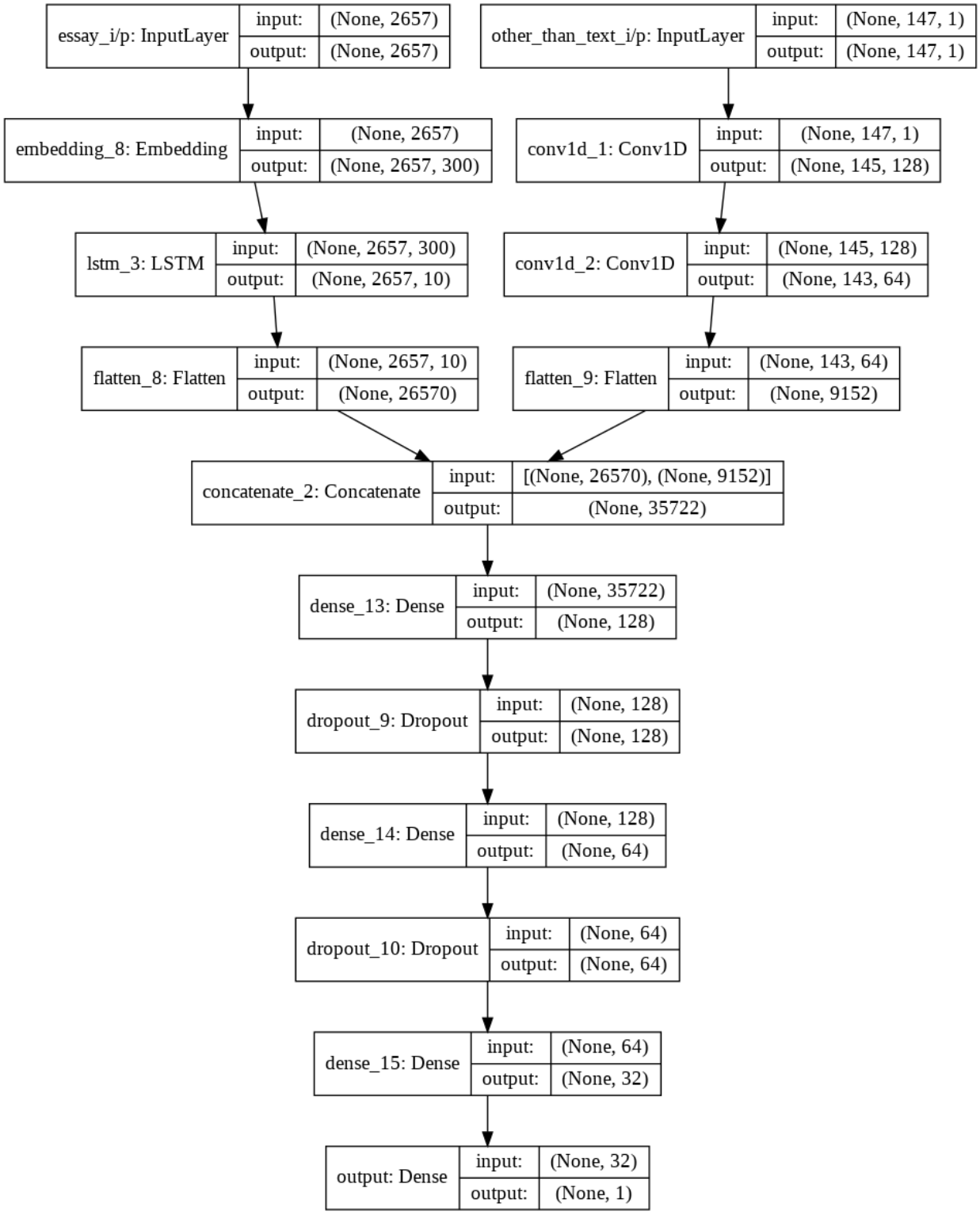
In [ ]:
```
input_train = [ padded_essay_train,
                other_than_text_x_train
              ]

input_val = [ padded_essay_val,
              other_than_text_x_val
            ]

input_test = [ padded_essay_test,
               other_than_text_x_test
             ]
```

In [ ]:
```
y_train.shape, y_cv.shape, y_test.shape
```

Out[ ]:
```
((69918,), (17480,), (21850,))
```

In [ ]:
```
from keras.utils import plot_model
plot_model(model3, to_file='model3.png', show_shapes=True)
```

Out[ ]:



In [ ]:
```
batch_size=400
epochs = 5
```

In [ ]:
```
# https://keras.io/callbacks/
tensorboard_callback = TensorBoard(log_dir='./logs3', histogram_freq=1, batch_size=batch_size, write_graph=True,
                          write_grads=False, write_images=True, embeddings_freq=0, update_freq='epoch')

filepath="Model3-weights-improvement-{epoch:02d}-{val_loss:.2f}.hdf5"
# I'm saving only best weights among the all epochs
model_checkpoint_callback = ModelCheckpoint(filepath, monitor='val_loss', verbose=0,
                                            save_best_only=True, save_weights_only=False, mode='auto', period=1)
```

In [ ]:
```
%tensorboard --logdir logs3
```

In [ ]:
```
history3 = model3.fit(input_train, y_train ,batch_size=batch_size,
                      epochs=epochs, validation_data=(input_val, y_cv),
                     callbacks=[tensorboard_callback, model_checkpoint_callback])
```

```
Train on 69918 samples, validate on 17480 samples
Epoch 1/5
69918/69918 [==============================] - 2121s 30ms/step - loss: 0.4613 - aucroc: 0.6102 - val_loss: 0.4151
- val_aucroc: 0.7430
Epoch 2/5
69918/69918 [==============================] - 2026s 29ms/step - loss: 0.3706 - aucroc: 0.7674 - val_loss: 0.4087
- val_aucroc: 0.7511
Epoch 3/5
69918/69918 [==============================] - 1987s 28ms/step - loss: 0.3227 - aucroc: 0.8397 - val_loss: 0.4180
- val_aucroc: 0.7378
Epoch 4/5
69918/69918 [==============================] - 1960s 28ms/step - loss: 0.2632 - aucroc: 0.9015 - val_loss: 0.4069
- val_aucroc: 0.7138
Epoch 5/5
69918/69918 [==============================] - 1948s 28ms/step - loss: 0.1957 - aucroc: 0.9484 - val_loss: 0.4323
- val_aucroc: 0.6822
```

In [1]:
```
from prettytable import PrettyTable

pt = PrettyTable(["model","train-auc", "test-auc", "train-loss", "test-loss"])
pt.add_row(["1", 0.6822, 0.7085, 0.4103, 0.3964])
pt.add_row(["2", 0.9973, 0.6176, 0.0426, 1.4914])
pt.add_row(["3", 0.7674, 0.7511, 0.3706 , 0.4087])
print(pt)
```

```
+-------+-----------+----------+------------+-----------+
| model | train-auc | test-auc | train-loss | test-loss |
+-------+-----------+----------+------------+-----------+
|   1   |   0.6822  |  0.7085  |   0.4103   |   0.3964  |
|   2   |   0.9973  |  0.6176  |   0.0426   |   1.4914  |
|   3   |   0.7674  |  0.7511  |   0.3706   |   0.4087  |
+-------+-----------+----------+------------+-----------+
```

In [ ]: