In [5]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_rem
ount=True).
```

In [6]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
np.random.seed(7)
from keras.utils import np_utils
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import Normalizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Conv1D
from keras.layers import MaxPooling1D
from tensorflow.keras.models import Model
from keras.layers import Activation, Flatten, Dropout, Input, concatenate, BatchNormalization
from keras import regularizers, optimizers
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
import datetime
import os
from tensorflow.keras import regularizers
```

In [7]:
```
import pickle
```

In [8]:
```
glove_pickle = open("/content/drive/My Drive/glove_vectors","rb")
glove_vec = pickle.load(glove_pickle)
glove_words =  set(glove_vec.keys())
```

In [9]:
```
preprocessed_data = pd.read_csv('/content/drive/My Drive/preprocessed_data.csv')
```

In [10]:
```
preprocessed_data.head()
```

Out[10]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_
0	ca	mrs	grades_prek_2	53	1	math_science	a hea
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	ea
4	wa	mrs	grades_3_5	2	1	literacy_language	

In [12]:
```
# Drop the class label column before splitting and apply featurizing techniques.
y = preprocessed_data['project_is_approved'].values
preprocessed_data.drop(['project_is_approved'], axis=1, inplace=True)
x = preprocessed_data
```

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

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.10, stratify = y)
x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, test_size = 0.20, stratify = y_train)
```

In [ ]:

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

```
(78658, 8)
(10925, 8)
(10925,)
(78658,)
```

In [78]:

```
## https://numpy.org/doc/stable/reference/generated/numpy.asarray.html
def fit(train_data):
  count_vec = CountVectorizer(lowercase=False)
  fit_ = count_vec.fit_transform(train_data)

  replace = fit_.sum(axis=0).A1
  num = replace.argsort()
  fea = count_vec.get_feature_names()

  str_ = {}
  score = 1
  for i in num[::-1]:
    str_[fea[i]] = score
    score += 1

  return str_

def transform(data, str_):

  abc_ = []
  for k in data:
    sent = k.split()
    xyz_ = []
    for l in sent:
      if l in str_:
        xyz_.append(str_[l])

    abc_.append(xyz_)
```

```
  return abc_
```

In [ ]:

```
# Encoding of the school_state feature
fit_val = fit(x_train['school_state'].values)

x_train_state_onehot = transform(x_train['school_state'].values, fit_val)
x_cv_state_onehot = transform(x_cv['school_state'].values, fit_val)
x_test_state_onehot = transform(x_test['school_state'].values, fit_val)

# Encoding of the teacher_prefix feature
fit_val = fit(x_train['teacher_prefix'].values)

x_train_teacher_onehot = transform(x_train['teacher_prefix'].values, fit_val)
x_cv_teacher_onehot = transform(x_cv['teacher_prefix'].values, fit_val)
x_test_teacher_onehot = transform(x_test['teacher_prefix'].values, fit_val)


#Encoding of the project_grade_category feature
fit_val = fit(x_train['project_grade_category'].values)

x_train_grade_onehot = transform(x_train['project_grade_category'].values, fit_val)
x_cv_grade_onehot = transform(x_cv['project_grade_category'].values, fit_val)
x_test_grade_onehot = transform(x_test['project_grade_category'].values, fit_val)


#Encoding of the clean_categories feature
fit_val = fit(x_train['clean_categories'].values)

x_train_clean_onehot = transform(x_train['clean_categories'].values, fit_val)
x_cv_clean_onehot = transform(x_cv['clean_categories'].values, fit_val)
x_test_clean_onehot = transform(x_test['clean_categories'].values, fit_val)


#Encoding of the clean_subcategories feature
fit_val = fit(x_train['clean_subcategories'].values)

x_train_clean_sub_onehot = transform(x_train['clean_subcategories'].values, fit_val)
x_cv_clean_sub_onehot = transform(x_cv['clean_subcategories'].values, fit_val)
x_test_clean_sub_onehot = transform(x_test['clean_subcategories'].values, fit_val)
```

In [ ]:
```
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_cv_price_norm = normalizer.transform(x_cv['price'].values.reshape(-1,1))
x_test_price_norm = normalizer.transform(x_test['price'].values.reshape(-1,1))

print("Shape of Normalised Values of remaining input(addition of price and previously posted projects:")
print(x_train_price_norm.shape, y_train.shape)
print(x_cv_price_norm.shape, y_cv.shape)
print(x_test_price_norm.shape, y_test.shape)
print("="*100)


normalizer.fit(x_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

x_train_projects_norm = normalizer.transform(x_train['teacher_number_of_previously_posted_projects'].values.reshape(1
x_cv_projects_norm = normalizer.transform(x_cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1).T
x_test_projects_norm = normalizer.transform(x_test['teacher_number_of_previously_posted_projects'].values.reshape(1,

print('Shape of Normalised Values of Previously posted projects by teachers:')
print(x_train_projects_norm.shape,y_train.shape)
print(x_cv_projects_norm.shape, y_cv.shape)
print(x_test_projects_norm.shape, y_test.shape)
print("="*100)
```

```
Shape of Normalised Values of remaining input(addition of price and previously posted projects:
(78658, 1) (78658,)
(19665, 1) (19665,)
(10925, 1) (10925,)
====================================================================================================
Shape of Normalised Values of Previously posted projects by teachers:
(78658, 1) (78658,)
(19665, 1) (19665,)
(10925, 1) (10925,)
====================================================================================================
```

In [ ]:
```
x_train_numerical = np.hstack([x_train_price_norm,x_train_projects_norm])
x_cv_numerical = np.hstack([x_cv_price_norm,x_cv_projects_norm])
x_test_numerical = np.hstack([x_test_price_norm,x_test_projects_norm])

print('shape of numerical data train', x_train_numerical.shape)
```

```
print('shape of numerical data cv', x_cv_numerical.shape)
print('shape of numerical data test', x_test_numerical.shape)
```

```
shape of numerical data train (78658, 2)
shape of numerical data cv (19665, 2)
shape of numerical data test (10925, 2)
```

In [ ]:
```
# Credits: https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

# prepare tokenizer
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train['essay'])
word_index_size = len(tokenizer.word_index) + 1
print('vocabulary size is :', word_index_size)

# integer encode the documents

text_to_seq = tokenizer.texts_to_sequences(x_train['essay'])
x_train_essay = pad_sequences(text_to_seq, maxlen=250, padding='post')

text_to_seq = tokenizer.texts_to_sequences(x_cv['essay'])
x_cv_essay = pad_sequences(text_to_seq, maxlen=250, padding='post')

text_to_seq = tokenizer.texts_to_sequences(x_test['essay'])
x_test_essay = pad_sequences(text_to_seq, maxlen=250, padding='post')
```

```
vocabulary size is : 49576
```

In [25]:
```
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train['essay'])
word_index_size = len(tokenizer.word_index) + 1
print('vocabulary size is :', word_index_size)
```

```
vocabulary size is : 47315
```

In [26]:
```
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
from numpy import zeros
mat = np.zeros((word_index_size, 300))
for word, i in tokenizer.word_index.items():
    vec = glove_vec.get(word) # vec=embedding vector
```

```
    if vec is not None:
        mat[i] = vec #mat=embedding matrix
```

In [27]:
```
print('The Embedding matrix shape is: ',mat.shape)
```

```
The Embedding matrix shape is:  (47315, 300)
```

In [ ]:
```
# Convert the class labels to categorical variable.

y_train = np_utils.to_categorical(y_train)
y_cv= np_utils.to_categorical(y_cv)
y_test= np_utils.to_categorical(y_test)
```

In [ ]:
```
# Consider maximum length for all the categorical feature as their own maximum length.

# Padding of the school_state feature
x_train_state_onehot = pad_sequences(x_train_state_onehot, maxlen=1)
x_cv_state_onehot = pad_sequences(x_cv_state_onehot, maxlen=1)
x_test_state_onehot = pad_sequences(x_test_state_onehot, maxlen=1)
print('Shape of train data for school state feature: ', x_train_state_onehot.shape)
print('Shape of cv data for school state feature: ', x_cv_state_onehot.shape)
print('Shape of test data for school state feature: ', x_test_state_onehot.shape)


# Padding of the teacher_prefix feature
x_train_teacher_onehot = pad_sequences(x_train_teacher_onehot, maxlen=1)
x_cv_teacher_onehot = pad_sequences(x_cv_teacher_onehot, maxlen=1)
x_test_teacher_onehot = pad_sequences(x_test_teacher_onehot, maxlen=1)
print('Shape of train data for teacher prefix feature: ', x_train_teacher_onehot.shape)
print('Shape of cv data for teacher prefix feature: ', x_cv_teacher_onehot.shape)
print('Shape of test data for teacher prefix feature: ', x_test_teacher_onehot.shape)

# Padding of the project_grade_category feature
x_train_grade_onehot = pad_sequences(x_train_grade_onehot, maxlen=1)
x_cv_grade_onehot = pad_sequences(x_cv_grade_onehot, maxlen=1)
x_test_grade_onehot = pad_sequences(x_test_grade_onehot, maxlen=1)
print('Shape of train data for project grade feature: ', x_train_grade_onehot.shape)
print('Shape of test data for project grade feature: ', x_test_grade_onehot.shape)
print('Shape of cv data for project grade feature: ', x_cv_grade_onehot.shape)

# Padding of the clean_categories feature
x_train_clean_onehot = pad_sequences(x_train_clean_onehot, maxlen=1)
```

```
x_cv_clean_onehot = pad_sequences(x_cv_clean_onehot, maxlen=1)
x_test_clean_onehot = pad_sequences(x_test_clean_onehot, maxlen=1)
print('Shape of train data for clean categories feature: ', x_train_clean_onehot.shape)
print('Shape of cv data for clean categories feature: ', x_cv_clean_onehot.shape)
print('Shape of test data for clean categories feature: ', x_test_clean_onehot.shape)

# Padding of the clean_subcategories feature
x_train_clean_sub_onehot = pad_sequences(x_train_clean_sub_onehot, maxlen=1)
x_cv_clean_sub_onehot = pad_sequences(x_cv_clean_sub_onehot, maxlen=1)
x_test_clean_sub_onehot = pad_sequences(x_test_clean_sub_onehot, maxlen=1)
print('Shape of train data for clean sub categories feature: ', x_train_clean_sub_onehot.shape)
print('Shape of cv data for clean sub categories feature: ', x_cv_clean_sub_onehot.shape)
print('Shape of test data for clean sub categories feature: ', x_test_clean_sub_onehot.shape)
```

```
Shape of train data for school state feature:  (78658, 1)
Shape of cv data for school state feature:  (19665, 1)
Shape of test data for school state feature:  (10925, 1)
Shape of train data for teacher prefix feature:  (78658, 1)
Shape of cv data for teacher prefix feature:  (19665, 1)
Shape of test data for teacher prefix feature:  (10925, 1)
Shape of train data for project grade feature:  (78658, 1)
Shape of test data for project grade feature:  (10925, 1)
Shape of cv data for project grade feature:  (19665, 1)
Shape of train data for clean categories feature:  (78658, 1)
Shape of cv data for clean categories feature:  (19665, 1)
Shape of test data for clean categories feature:  (10925, 1)
Shape of train data for clean sub categories feature:  (78658, 1)
Shape of cv data for clean sub categories feature:  (19665, 1)
Shape of test data for clean sub categories feature:  (10925, 1)
```

In [76]:

```
# 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 auc_(y_true, y_pred):
    if len(np.unique(y_true[:,1])) == 1:
        return 0.5
    else:
        return roc_auc_score(y_true, y_pred)

def auc(y_true, y_pred):
    return tf.py_function(auc_, (y_true, y_pred), tf.double)
```

In [43]:
```
x_train_essay_embbeding = mat
```

In [ ]:
```
# https://stackoverflow.com/questions/37213388/keras-accuracy-does-not-change#:~:text=If%20the%20accuracy%20is%20not,

from sklearn.utils import compute_class_weight
classWeight = compute_class_weight('balanced', classes = np.unique(y), y=y)
classWeight = dict(enumerate(classWeight))
classWeight
```

Out[ ]:
```
{0: 3.3021400072542617, 1: 0.5892175263736975}
```

In [ ]:
```
# Load the TensorBoard notebook extension
%load_ext tensorboard
```

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

In [ ]:
```
# Credits--> https://medium.com/@davidheffernan_99410/an-introduction-to-using-categorical-embeddings-ee686ed7e7f9
tf.keras.backend.clear_session()
# Clear any logs from previous runs
!rm -rf ./logs/fit
```

In [ ]:
```
input1_ = Input(shape=(x_train_essay.shape[1],))
embedding1_ = Embedding(input_dim=x_train_essay_embbeding.shape[0]+1, output_dim=300, trainable=False, input_length=x
lstm = LSTM(4, return_sequences=True)(embedding1_)
flatt_ = Flatten()(lstm)
```

In [ ]:
```
input2_ = Input(shape=(1,))
dimension_out = min(50,(x_train['school_state'].nunique()//2)+1)
embedding2_ = Embedding(input_dim=x_train['school_state'].nunique()+1, output_dim=dimension_out)(input2_)
flatt_1 = Flatten()(embedding2_)
```

In [ ]:
```
input3_ = Input(shape=(1,))
dimension_out = min(50,(x_train['teacher_prefix'].nunique()//2)+1)
embedding3_ = Embedding(input_dim=x_train['teacher_prefix'].nunique()+1, output_dim=dimension_out)(input3_)
flatt_2 = Flatten()(embedding3_)
```

In [ ]:
```
input4_ = Input(shape=(1,))
dimension_out = min(50,(x_train['project_grade_category'].nunique()//2)+1)
```

```
embedding4_ = Embedding(input_dim=x_train['project_grade_category'].nunique()+1, output_dim=dimension_out)(input4_)
flatt_3 = Flatten()(embedding4_)
```

In [ ]:
```
input5_ = Input(shape=(1,))
dimension_out = min(50,(x_train['clean_categories'].nunique()//2)+1)
embedding5_ = Embedding(input_dim=x_train['clean_categories'].nunique()+1, output_dim=dimension_out)(input5_)
flatt_4 = Flatten()(embedding5_)
```

In [ ]:
```
input6_ = Input(shape=(1,))
dimension_out = min(50,(x_train['clean_subcategories'].nunique()//2)+1)
embedding6_ = Embedding(input_dim=x_train['clean_subcategories'].nunique()+1, output_dim=dimension_out)(input6_)
flatt_5 = Flatten()(embedding6_)
```

In [ ]:
```
input7_ = Input(shape=(2,))
out7 = Dense(units=48,
             activation="relu",
             kernel_initializer=tf.keras.initializers.he_normal(seed=32),
             name='remain_dense')(input7_)
```

In [ ]:
```
# For remaining_data column (price + previously_posted_projects)

concat = concatenate([flatt_,flatt_1,flatt_2,flatt_3,flatt_4,flatt_5,out7])
```

In [ ]:
```
x = Dense(units=128,
                activation='relu',
                kernel_initializer=tf.keras.initializers.he_normal(seed=24),
                name = 'concat_dense_1')(concat)

x = BatchNormalization()(x)
x = Dropout(0.5, seed=22)(x)
```

In [ ]:
```
x = Dense(units=64,
              activation='relu',
              kernel_initializer=tf.keras.initializers.he_normal(seed=22),
              name='concat_dense_2')(x)
x = Dropout(0.6, seed=18)(x)
```

In [ ]:
```
x = Dense(units=32,
                activation='relu',
```

```
                kernel_initializer=tf.keras.initializers.he_normal(seed=20),
                name='concat_dense_3')(x)
x = BatchNormalization()(x)
x = Dropout(0.6, seed=26)(x)
```

In [ ]:
```
x = Dense(units=16,
                activation='relu',
                kernel_initializer=tf.keras.initializers.he_normal(seed=20),
                name='concat_dense_4')(x)
x = BatchNormalization()(x)
x = Dropout(0.5, seed=18)(x)
```

In [ ]:
```
x = Dense(units=8,
                activation='relu',
                kernel_initializer=tf.keras.initializers.he_normal(seed=20),
                name='concat_dense_5')(x)
x = BatchNormalization()(x)
x = Dropout(0.5, seed=18)(x)
```

In [ ]:
```
x = Dense(units=2,
               activation='softmax',
               kernel_initializer=tf.keras.initializers.he_normal(seed=42),
               name='Output')(x)
```

In [ ]:
```
# https://colab.research.google.com/github/tensorflow/tensorboard/blob/master/docs/get_started.ipynb#scrollTo=Ao7fJW1
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
                                                     histogram_freq=1)

#Creating a model
model = Model(inputs=[input1_,input2_,input3_,input4_,input5_,input6_,input7_], outputs = x)
model.run_eagerly = True

#file path, it saves the model in the 'model_save' folder and we are naming model with epoch number
filepath="content/drive/MyDrive/model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint_dir = os.path.dirname(filepath)
checkpoint = ModelCheckpoint(filepath=filepath,
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
```

```
                                mode='auto')

reduce_lr = ReduceLROnPlateau(monitor='val_accuracy',
                              factor=0.92,
                              patience=1)
earlystopping = EarlyStopping(monitor='val_accuracy',
                              min_delta=0.00008,
                              patience=8,
                              verbose=1,
                              mode='auto')
```

In [ ]:
```
# here we are creating a list with all the callbacks we want
callback_list = [checkpoint, earlystopping, tensorboard, reduce_lr]
```

In [ ]:
```
#compiling
opt = tf.keras.optimizers.Adam(learning_rate=0.0091)
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['accuracy', auc])

model.summary()
```

```
Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 250)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 250, 300)     14873100    input_1[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_5 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_6 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 250, 4)       4880        embedding[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 26)        1352        input_2[0][0]
```

embedding_2 (Embedding)	(None, 1, 3)	18	input_3[0][0]
embedding_3 (Embedding)	(None, 1, 3)	15	input_4[0][0]
embedding_4 (Embedding)	(None, 1, 26)	1352	input_5[0][0]
embedding_5 (Embedding)	(None, 1, 50)	19950	input_6[0][0]
input_7 (InputLayer)	[(None, 2)]	0	
flatten (Flatten)	(None, 1000)	0	lstm[0][0]
flatten_1 (Flatten)	(None, 26)	0	embedding_1[0][0]
flatten_2 (Flatten)	(None, 3)	0	embedding_2[0][0]
flatten_3 (Flatten)	(None, 3)	0	embedding_3[0][0]
flatten_4 (Flatten)	(None, 26)	0	embedding_4[0][0]
flatten_5 (Flatten)	(None, 50)	0	embedding_5[0][0]
remain_dense (Dense)	(None, 48)	144	input_7[0][0]
concatenate_2 (Concatenate)	(None, 1156)	0	flatten[0][0] flatten_1[0][0] flatten_2[0][0] flatten_3[0][0] flatten_4[0][0] flatten_5[0][0] remain_dense[0][0]
concat_dense_1 (Dense)	(None, 128)	148096	concatenate_2[0][0]
batch_normalization_4 (BatchNor	(None, 128)	512	concat_dense_1[0][0]
dropout_5 (Dropout)	(None, 128)	0	batch_normalization_4[0][0]
concat_dense_2 (Dense)	(None, 64)	8256	dropout_5[0][0]
dropout_6 (Dropout)	(None, 64)	0	concat_dense_2[0][0]
concat_dense_3 (Dense)	(None, 32)	2080	dropout_6[0][0]

```
batch_normalization_5 (BatchNor (None, 32)           128         concat_dense_3[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 32)           0           batch_normalization_5[0][0]
__________________________________________________________________________________________________
concat_dense_4 (Dense)          (None, 16)           528         dropout_7[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 16)           64          concat_dense_4[0][0]
__________________________________________________________________________________________________
dropout_8 (Dropout)             (None, 16)           0           batch_normalization_6[0][0]
__________________________________________________________________________________________________
concat_dense_5 (Dense)          (None, 8)            136         dropout_8[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 8)            32          concat_dense_5[0][0]
__________________________________________________________________________________________________
dropout_9 (Dropout)             (None, 8)            0           batch_normalization_7[0][0]
__________________________________________________________________________________________________
Output (Dense)                  (None, 2)            18          dropout_9[0][0]
==================================================================================================
Total params: 15,060,661
Trainable params: 187,193
Non-trainable params: 14,873,468
__________________________________________________________________________________________________
```

In [ ]:

```
model.fit(x = [x_train_essay, x_train_state_onehot, x_train_teacher_onehot, x_train_grade_onehot, x_train_clean_oneho
          y = y_train,
          epochs = 80,
          batch_size = 24,
          validation_data = ([x_cv_essay, x_cv_state_onehot, x_cv_teacher_onehot, x_cv_grade_onehot, x_cv_clean_oneho
          callbacks = callback_list,
          class_weight=classWeight)
```

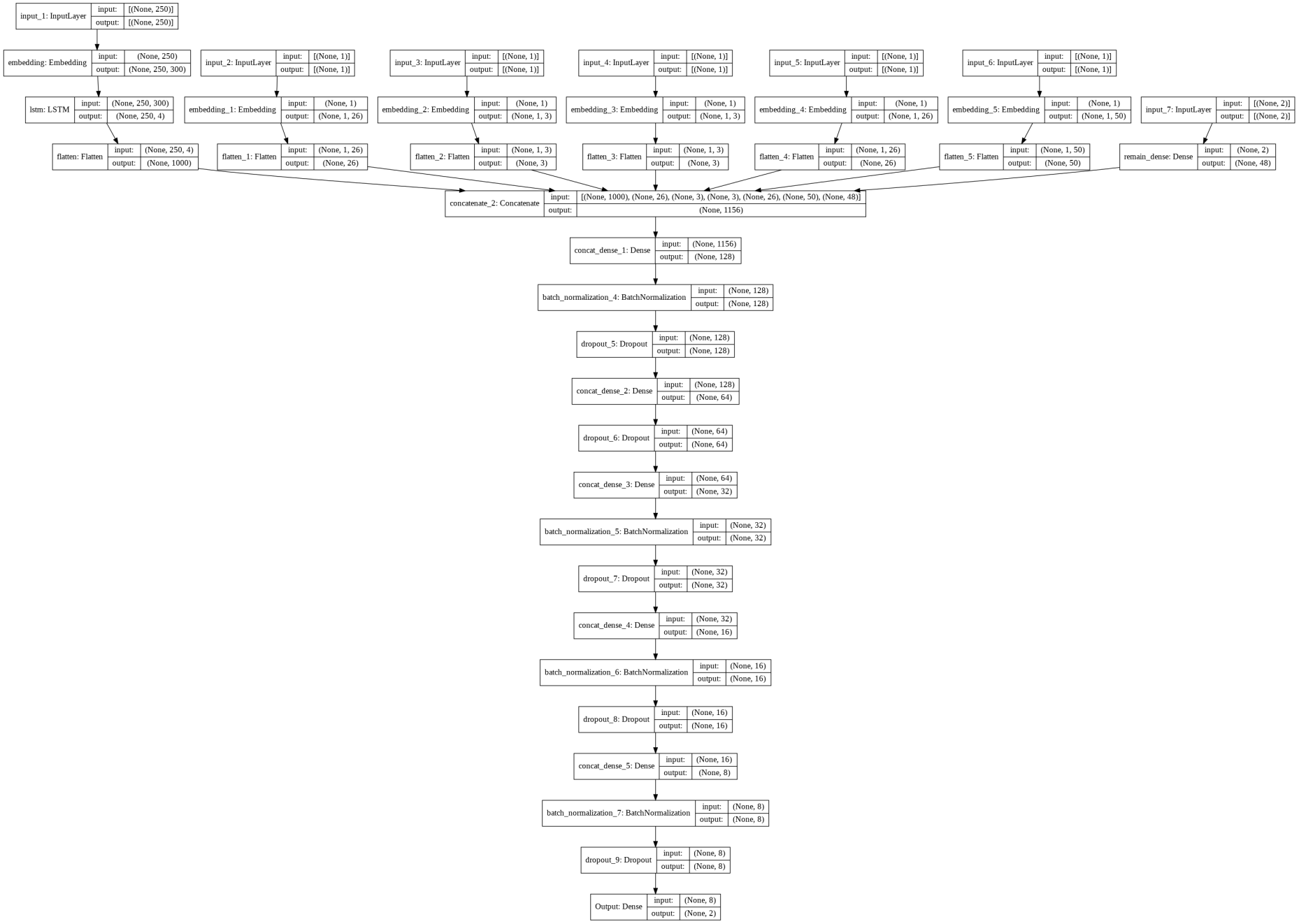
```
Epoch 1/80
3278/3278 [==============================] - 67s 20ms/step - loss: 0.6937 - accuracy: 0.4894 - auc: 0.5200 - val_los
s: 0.6547 - val_accuracy: 0.8486 - val_auc: 0.5753

Epoch 00001: val_accuracy improved from -inf to 0.84856, saving model to content/drive/MyDrive/model_save/weights-01-
0.8486.hdf5
Epoch 2/80
3278/3278 [==============================] - 65s 20ms/step - loss: 0.6911 - accuracy: 0.5088 - auc: 0.5448 - val_los
s: 0.5758 - val_accuracy: 0.7909 - val_auc: 0.6365

Epoch 00002: val_accuracy did not improve from 0.84856
Epoch 3/80
3278/3278 [==============================] - 64s 20ms/step - loss: 0.6700 - accuracy: 0.5986 - auc: 0.6351 - val_los
s: 0.6251 - val_accuracy: 0.7179 - val_auc: 0.6809
```

```
Epoch 00003: val_accuracy did not improve from 0.84856
Epoch 4/80
3278/3278 [==============================] - 64s 20ms/step - loss: 0.6528 - accuracy: 0.6596 - auc: 0.6742 - val_los
s: 0.5748 - val_accuracy: 0.7257 - val_auc: 0.7048

Epoch 00004: val_accuracy did not improve from 0.84856
Epoch 5/80
3278/3278 [==============================] - 64s 19ms/step - loss: 0.6421 - accuracy: 0.6881 - auc: 0.6909 - val_los
s: 0.7034 - val_accuracy: 0.5557 - val_auc: 0.7071

Epoch 00005: val_accuracy did not improve from 0.84856
Epoch 6/80
3278/3278 [==============================] - 64s 20ms/step - loss: 0.6346 - accuracy: 0.6866 - auc: 0.7049 - val_los
s: 0.6320 - val_accuracy: 0.6888 - val_auc: 0.7146

Epoch 00006: val_accuracy did not improve from 0.84856
Epoch 7/80
3278/3278 [==============================] - 63s 19ms/step - loss: 0.6309 - accuracy: 0.6871 - auc: 0.7114 - val_los
s: 0.6283 - val_accuracy: 0.6770 - val_auc: 0.7138

Epoch 00007: val_accuracy did not improve from 0.84856
Epoch 8/80
3278/3278 [==============================] - 63s 19ms/step - loss: 0.6222 - accuracy: 0.7067 - auc: 0.7218 - val_los
s: 0.6260 - val_accuracy: 0.6810 - val_auc: 0.7163

Epoch 00008: val_accuracy did not improve from 0.84856
Epoch 9/80
3278/3278 [==============================] - 63s 19ms/step - loss: 0.6208 - accuracy: 0.7000 - auc: 0.7253 - val_los
s: 0.6034 - val_accuracy: 0.7531 - val_auc: 0.7180

Epoch 00009: val_accuracy did not improve from 0.84856
Epoch 00009: early stopping
```

Out[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x7ff19be34f50>
```

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

Out[ ]:



In [ ]:
```
test_outcomes = model.evaluate([x_test_essay, x_test_state_onehot, x_test_teacher_onehot, x_test_grade_onehot, x_test
                         y_test)

print('Test loss score:', test_outcomes[0])
print('Test accuracy score:', test_outcomes[1])
print('Test AUC score:', test_outcomes[2])
```

```
342/342 [==============================] - 4s 12ms/step - loss: 0.5981 - accuracy: 0.7536 - auc: 0.7448
Test loss score: 0.5981236100196838
Test accuracy score: 0.7535926699638367
Test AUC score: 0.7448391318321228
```

In [ ]:
```
train_outcome = model.evaluate([x_train_essay, x_train_state_onehot, x_train_teacher_onehot, x_train_grade_onehot, x_
                         y_train)

print('Train loss score:', train_outcome[0])
print('Train accuracy score:', train_outcome[1])
print('Train AUC score:', train_outcome[2])
```

```
2459/2459 [==============================] - 27s 11ms/step - loss: 0.5915 - accuracy: 0.7661 - auc: 0.7575
Train loss score: 0.5914525985717773
Train accuracy score: 0.7661140561103821
Train AUC score: 0.7574912905693054
```

In [ ]:
```
cv_outcome = model.evaluate([x_cv_essay, x_cv_state_onehot, x_cv_teacher_onehot, x_cv_grade_onehot, x_cv_clean_onehot
                         y_cv)

print('Train loss score:', cv_outcome[0])
print('Train accuracy score:', cv_outcome[1])
print('Train AUC score:', cv_outcome[2])
```

```
615/615 [==============================] - 7s 11ms/step - loss: 0.6034 - accuracy: 0.7531 - auc: 0.7199
Train loss score: 0.6033660173416138
Train accuracy score: 0.7531147003173828
Train AUC score: 0.7198941111564636
```

In [ ]:
```
from IPython.display import Image
%load_ext tensorboard
%tensorboard --logdir logs/fit
```

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

Model 2

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 [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [16]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
np.random.seed(7)
from keras.utils import np_utils
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import Normalizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Conv1D
from keras.layers import MaxPooling1D
from tensorflow.keras.models import Model
from keras.layers import Activation, Flatten, Dropout, Input, concatenate, BatchNormalization
from keras import regularizers, optimizers
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
import datetime
import os
```

```
from tensorflow.keras import regularizers
import pandas as pd
import tensorflow as tf
from tensorflow.keras.layers import *
from time import time
from tensorflow.keras.callbacks import TensorBoard
from sklearn.model_selection import train_test_split
from tqdm import tqdm
import pickle
import numpy as np
from collections import Counter
import re
from sklearn.feature_extraction.text import CountVectorizer
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
import pickle
```

In [4]:
```
glove_pickle = open("/content/drive/My Drive/glove_vectors","rb")
glove_vec = pickle.load(glove_pickle)
glove_words =  set(glove_vec.keys())
```

In [5]:
```
data = pd.read_csv('/content/drive/My Drive/preprocessed_data.csv')
```

In [6]:
```
data.shape
```

Out[6]:
```
(109248, 9)
```

In [7]:
```
project_is_approved = data.project_is_approved.values
preprocessed_data = data.drop(columns=['project_is_approved'])
```

In [8]:
```
x_train, x_test, y_train, y_test = train_test_split(preprocessed_data, project_is_approved, test_size=0.2, random_sta
```

In [9]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(87398, 8)
```

```
(21850, 8)
(87398,)
(21850,)
```

In [10]:
```
x_train,x_cv,y_train,y_cv=train_test_split(x_train, y_train, test_size=0.2)
```

In [11]:
```
x_train['essay']=x_train['essay'].str.lower()
x_test['essay']=x_test['essay'].str.lower()
x_cv['essay']=x_cv['essay'].str.lower()
```

In [12]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vec = TfidfVectorizer(min_df=3, use_idf=True)
vec.fit(x_train['essay'])

# we use the fitted CountVectorizer to convert the text to vector
tr_essay=vec.transform(x_train['essay'].values)
te_essay=vec.transform(x_test['essay'].values)
cv_essay=vec.transform(x_cv['essay'].values)
```

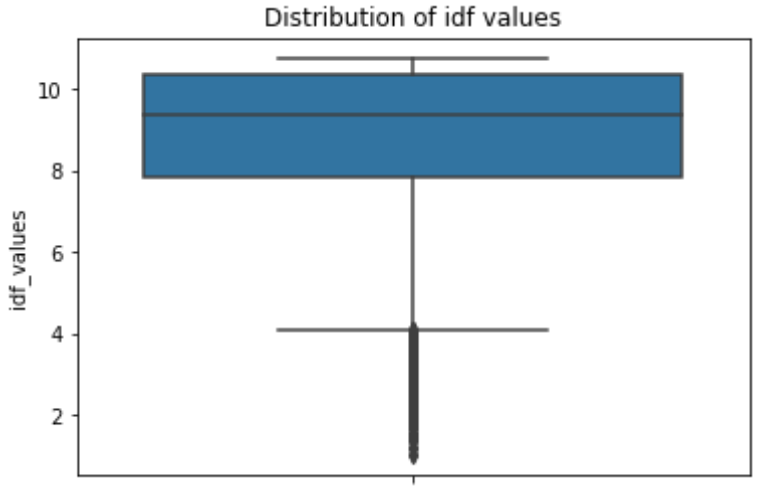
In [13]:
```
ans = vec.idf_
dic = dict(zip(vec.get_feature_names(), ans))

data_1 = pd.DataFrame(dic.items(), columns=['words', 'idf_values'])
data_1 = data_1.sort_values(by='idf_values')
data_1.head()
```

Out[13]:

	words	idf_values
20181	students	1.007595
13819	nannan	1.044497
18283	school	1.163296
13774	my	1.244871
12086	learning	1.362192

In [ ]:
```
# vocabulary = list(feature_idf_dict_sorted.keys())
# idf_val = list(feature_idf_dict_sorted.values())
```

In [14]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(y='idf_values',data=data_1)
plt.title('Distribution of idf values')
plt.show()
```



In [17]:
```
# As mention consider the IQR values.

down = data_1['idf_values'].quantile(0.0005)
upp = data_1['idf_values'].quantile(0.98)

print(down)
print(upp)
```

```
1.6438584912324135
10.768798347047754
```

In [18]:
```
data_range = data_1[(data_1['idf_values'] > down) & (data_1['idf_values'] < upp)]
data_range.shape
```

Out[18]:
```
(20560, 2)
```

In [19]:
```
data_range = data_1[(data_1['idf_values'] <= down) | (data_1['idf_values'] >= upp)]
delt=list(data_range['words'])
```

```
len(delt)
```

Out[19]: 2923

In [20]:
```
from tqdm import tqdm
def del_let(data):
    txt_essay = []
    for sent in tqdm(data.values):
        sent = ' '.join(e for e in sent.split() if e.lower() not in delt)
        txt_essay.append(sent.lower().strip())
    return txt_essay
```

In [21]:
```
x_train['essay'] = del_let(x_train['essay'])
x_cv['essay'] = del_let(x_cv['essay'])
x_test['essay'] = del_let(x_test['essay'])
```

```
100%|██████████| 69918/69918 [11:34<00:00, 100.69it/s]
100%|██████████| 17480/17480 [02:53<00:00, 100.68it/s]
100%|██████████| 21850/21850 [03:35<00:00, 101.36it/s]
```

In [72]:
```
# prepare tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train['essay'])
vocab_size = len(tokenizer.word_index) + 1
print('vocab size:', vocab_size)
```

```
vocab size: 47315
```

In [73]:
```
tok_ = t.texts_to_sequences(x_train['essay'])
x_tr_essay = pad_sequences(tok_, maxlen=250, padding='post')
print(x_tr_essay.shape)

tok_ = t.texts_to_sequences(x_cv['essay'])
x_cv_essay = pad_sequences(tok_, maxlen=250, padding='post')
print(x_cv_essay.shape)

tok_ = t.texts_to_sequences(x_test['essay'])
x_test_essay = pad_sequences(tok_, maxlen=250, padding='post')
print(x_test_essay.shape)
```

```
(69918, 250)
(17480, 250)
```

```
(21850, 250)
```

In [74]:
```
mat = np.zeros((vocab_size, 300))
for word, i in t.word_index.items():
    vect = glove_vec.get(word)
    if vect is not None:
        mat[i] = vect

print('Embedding matrix shape: ', mat.shape)
```

```
Embedding matrix shape:  (47315, 300)
```

In [81]:
```
# Encoding of the school_state feature
fit_val = fit(x_train['school_state'].values)

x_train_state_onehot = transform(x_train['school_state'].values, fit_val)
x_cv_state_onehot = transform(x_cv['school_state'].values, fit_val)
x_test_state_onehot = transform(x_test['school_state'].values, fit_val)

# print(x_train_state_onehot)
# print(x_cv_state_onehot)
# print(x_test_state_onehot)
```

In [ ]:
```
fit_val = fit(x_train['teacher_prefix'].values)

x_train_teacher_onehot = transform(x_train['teacher_prefix'].values, fit_val)
x_cv_teacher_onehot = transform(x_cv['teacher_prefix'].values, fit_val)
x_test_teacher_onehot = transform(x_test['teacher_prefix'].values, fit_val)

# print(x_train_teacher_onehot)
# print(x_cv_teacher_onehot)
# print(x_test_teacher_onehot)
```

In [82]:
```
fit_val = fit(x_train['project_grade_category'].values)

x_train_grade_onehot = transform(x_train['project_grade_category'].values, fit_val)
x_cv_grade_onehot = transform(x_cv['project_grade_category'].values, fit_val)
x_test_grade_onehot = transform(x_test['project_grade_category'].values, fit_val)

# print(x_train_grade_onehot)
# print(x_cv_grade_onehot)
# print(x_test_grade_onehot)
```

In [83]:
```
fit_val = fit(x_train['clean_categories'].values)

x_train_clean_onehot = transform(x_train['clean_categories'].values, fit_val)
x_cv_clean_onehot = transform(x_cv['clean_categories'].values, fit_val)
x_test_clean_onehot = transform(x_test['clean_categories'].values, fit_val)

# print(x_train_clean_onehot)
# print(x_cv_clean_onehot)
# print(x_test_clean_onehot)
```

In [84]:
```
fit_val = fit(x_train['clean_subcategories'].values)

x_train_clean_sub_onehot = transform(x_train['clean_subcategories'].values, fit_val)
x_cv_clean_sub_onehot = transform(x_cv['clean_subcategories'].values, fit_val)
x_test_clean_sub_onehot = transform(x_test['clean_subcategories'].values, fit_val)

# print(x_train_clean_sub_onehot)
# print(x_cv_clean_sub_onehot)
# print(x_test_clean_sub_onehot)
```

In [85]:
```
x_train_state_onehot = pad_sequences(x_train_state_onehot, maxlen=1)
x_cv_state_onehot = pad_sequences(x_cv_state_onehot, maxlen=1)
x_test_state_onehot = pad_sequences(x_test_state_onehot, maxlen=1)
print('shape of tr of school: ', x_train_state_onehot.shape)
print('shape of te of school: ', x_cv_state_onehot.shape)
print('shape of cv of school: ', x_test_state_onehot.shape)
```

```
shape of tr of school:  (69918, 1)
shape of te of school:  (17480, 1)
shape of cv of school:  (21850, 1)
```

In [26]:
```
Consider maximum length for all the categorical feature as their own maximum length.

# school_state feature
x_train_state_onehot = pad_sequences(x_train_state_onehot, maxlen=1)
x_cv_state_onehot = pad_sequences(x_cv_state_onehot, maxlen=1)
x_test_state_onehot = pad_sequences(x_test_state_onehot, maxlen=1)
print('Shape of train data for school state feature: ', x_train_state_onehot.shape)

#  teacher_prefix feature
```

```
x_train_teacher_onehot = pad_sequences(x_train_teacher_onehot, maxlen=1)
x_cv_teacher_onehot = pad_sequences(x_cv_teacher_onehot, maxlen=1)
x_test_teacher_onehot = pad_sequences(x_test_teacher_onehot, maxlen=1)
print('Shape of train data for teacher prefix feature: ', x_train_teacher_onehot.shape)

# project_grade_category feature
x_train_grade_onehot = pad_sequences(x_train_grade_onehot, maxlen=1)
x_cv_grade_onehot = pad_sequences(x_cv_grade_onehot, maxlen=1)
x_test_grade_onehot = pad_sequences(x_test_grade_onehot, maxlen=1)
print('Shape of train data for project grade feature: ', x_train_grade_onehot.shape)

# clean_categories feature
x_train_clean_onehot = pad_sequences(x_train_clean_onehot, maxlen=1)
x_cv_clean_onehot = pad_sequences(x_cv_clean_onehot, maxlen=1)
x_test_clean_onehot = pad_sequences(x_test_clean_onehot, maxlen=1)
print('Shape of train data for clean categories feature: ', x_train_clean_onehot.shape)

# clean_subcategories feature
x_train_clean_sub_onehot = pad_sequences(x_train_clean_sub_onehot, maxlen=1)
x_cv_clean_sub_onehot = pad_sequences(x_cv_clean_sub_onehot, maxlen=1)
x_test_clean_sub_onehot = pad_sequences(x_test_clean_sub_onehot, maxlen=1)
print('Shape of train data for clean sub categories feature: ', x_train_clean_sub_onehot.shape)
```

```
Shape of train data for school state feature:  (69918, 1)
Shape of train data for teacher prefix feature:  (69918, 1)
Shape of train data for project grade feature:  (69918, 1)
Shape of train data for clean categories feature:  (69918, 1)
Shape of train data for clean sub categories feature:  (69918, 1)
```

In [27]:

```
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_cv_price_norm = normalizer.transform(x_cv['price'].values.reshape(-1,1))
x_test_price_norm = normalizer.transform(x_test['price'].values.reshape(-1,1))

print("Shape of Normalised Values of remaining input(addition of price and previously posted projects:")
print(x_train_price_norm.shape, y_train.shape)
print(x_cv_price_norm.shape, y_cv.shape)
print(x_test_price_norm.shape, y_test.shape)


normalizer.fit(x_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
```

```
x_train_projects_norm = normalizer.transform(x_train['teacher_number_of_previously_posted_projects'].values.reshape(1
x_cv_projects_norm = normalizer.transform(x_cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1).T
x_test_projects_norm = normalizer.transform(x_test['teacher_number_of_previously_posted_projects'].values.reshape(1,

print('Shape of Normalised Values of Previously posted projects by teachers:')
print(x_train_projects_norm.shape,y_train.shape)
print(x_cv_projects_norm.shape, y_cv.shape)
print(x_test_projects_norm.shape, y_test.shape)
print("="*100)
```

```
Shape of Normalised Values of remaining input(addition of price and previously posted projects:
(69918, 1) (69918,)
(17480, 1) (17480,)
(21850, 1) (21850,)
====================================================================================================
Shape of Normalised Values of Previously posted projects by teachers:
(69918, 1) (69918,)
(17480, 1) (17480,)
(21850, 1) (21850,)
====================================================================================================
```

In [28]:
```
x_train_numerical = np.hstack([x_train_price_norm,x_train_projects_norm])
x_cv_numerical = np.hstack([x_cv_price_norm,x_cv_projects_norm])
x_test_numerical = np.hstack([x_test_price_norm,x_test_projects_norm])

print('shape of numerical data train', x_train_numerical.shape)
print('shape of numerical data cv', x_cv_numerical.shape)
print('shape of numerical data test', x_test_numerical.shape)
```

```
shape of numerical data train (69918, 2)
shape of numerical data cv (17480, 2)
shape of numerical data test (21850, 2)
```

In [29]:
```
# Convert the class labels to categorical variable.

y_train = np_utils.to_categorical(y_train)
y_cv= np_utils.to_categorical(y_cv)
y_test= np_utils.to_categorical(y_test)
```

In [35]:
```
# https://stackoverflow.com/questions/37213388/keras-accuracy-does-not-change#:~:text=If%20the%20accuracy%20is%20not,

from sklearn.utils import compute_class_weight
```

```
classWeight = compute_class_weight('balanced', classes = np.unique(y), y=y)
classWeight = dict(enumerate(classWeight))
classWeight
```

Out[35]:
```
{0: 3.3021400072542617, 1: 0.5892175263736975}
```

In [36]:
```
# Load the TensorBoard notebook extension
%load_ext tensorboard
```

In [37]:
```
# Credits--> https://medium.com/@davidheffernan_99410/an-introduction-to-using-categorical-embeddings-ee686ed7e7f9
tf.keras.backend.clear_session()
# Clear any logs from previous runs
!rm -rf ./logs/fit
```

In [44]:
```
input1_ = Input(shape=(x_tr_essay.shape[1],))
embedding1_ = Embedding(input_dim=x_train_essay_embbeding.shape[0]+1, output_dim=300, trainable=False, input_length=x
lstm = LSTM(4, return_sequences=True)(embedding1_)
flatt_ = Flatten()(lstm)
```

In [45]:
```
input2_ = Input(shape=(1,))
dimension_out = min(50,(x_train['school_state'].nunique()//2)+1)
embedding2_ = Embedding(input_dim=x_train['school_state'].nunique()+1, output_dim=dimension_out)(input2_)
flatt_1 = Flatten()(embedding2_)
```

In [46]:
```
input3_ = Input(shape=(1,))
dimension_out = min(50,(x_train['teacher_prefix'].nunique()//2)+1)
embedding3_ = Embedding(input_dim=x_train['teacher_prefix'].nunique()+1, output_dim=dimension_out)(input3_)
flatt_2 = Flatten()(embedding3_)
```

In [47]:
```
input4_ = Input(shape=(1,))
dimension_out = min(50,(x_train['project_grade_category'].nunique()//2)+1)
embedding4_ = Embedding(input_dim=x_train['project_grade_category'].nunique()+1, output_dim=dimension_out)(input4_)
flatt_3 = Flatten()(embedding4_)
```

In [48]:
```
input5_ = Input(shape=(1,))
dimension_out = min(50,(x_train['clean_categories'].nunique()//2)+1)
embedding5_ = Embedding(input_dim=x_train['clean_categories'].nunique()+1, output_dim=dimension_out)(input5_)
flatt_4 = Flatten()(embedding5_)
```

In [49]:
```
input6_ = Input(shape=(1,))
dimension_out = min(50,(x_train['clean_subcategories'].nunique()//2)+1)
embedding6_ = Embedding(input_dim=x_train['clean_subcategories'].nunique()+1, output_dim=dimension_out)(input6_)
flatt_5 = Flatten()(embedding6_)
```

In [50]:
```
input7_ = Input(shape=(2,))
out7 = Dense(units=48,
             activation="relu",
             kernel_initializer=tf.keras.initializers.he_normal(seed=32),
             name='remain_dense')(input7_)
```

In [51]:
```
# For remaining_data column (price + previously_posted_projects)

concat = concatenate([flatt_,flatt_1,flatt_2,flatt_3,flatt_4,flatt_5,out7])
```

In [52]:
```
x = Dense(units=128,
                 activation='relu',
                 kernel_initializer=tf.keras.initializers.he_normal(seed=24),
                 name = 'concat_dense_1')(concat)

x = BatchNormalization()(x)
x = Dropout(0.5, seed=22)(x)
```

In [53]:
```
x = Dense(units=64,
              activation='relu',
              kernel_initializer=tf.keras.initializers.he_normal(seed=22),
              name='concat_dense_2')(x)
x = Dropout(0.6, seed=18)(x)
```

In [54]:
```
x = Dense(units=32,
                 activation='relu',
                 kernel_initializer=tf.keras.initializers.he_normal(seed=20),
                 name='concat_dense_3')(x)
x = BatchNormalization()(x)
x = Dropout(0.6, seed=26)(x)
```

In [55]:
```
x = Dense(units=16,
                 activation='relu',
                 kernel_initializer=tf.keras.initializers.he_normal(seed=20),
                 name='concat_dense_4')(x)
```

```
x = BatchNormalization()(x)
x = Dropout(0.5, seed=18)(x)
```

In [56]:
```
x = Dense(units=8,
                activation='relu',
                kernel_initializer=tf.keras.initializers.he_normal(seed=20),
                name='concat_dense_5')(x)
x = BatchNormalization()(x)
x = Dropout(0.5, seed=18)(x)
```

In [57]:
```
x = Dense(units=2,
               activation='softmax',
               kernel_initializer=tf.keras.initializers.he_normal(seed=42),
               name='Output')(x)
```

In [58]:
```
# https://colab.research.google.com/github/tensorflow/tensorboard/blob/master/docs/get_started.ipynb#scrollTo=Ao7fJW1
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=log_dir,
                                                        histogram_freq=1)

#Creating a model
model = Model(inputs=[input1_,input2_,input3_,input4_,input5_,input6_,input7_], outputs = x)
model.run_eagerly = True

#file path, it saves the model in the 'model_save' folder and we are naming model with epoch number
filepath="content/drive/MyDrive/model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpoint_dir = os.path.dirname(filepath)
checkpoint = ModelCheckpoint(filepath=filepath,
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='auto')

reduce_lr = ReduceLROnPlateau(monitor='val_accuracy',
                              factor=0.92,
                              patience=1)
earlystopping = EarlyStopping(monitor='val_accuracy',
                            min_delta=0.00008,
                            patience=8,
                            verbose=1,
                            mode='auto')
```

In [59]:
```
# here we are creating a list with all the callbacks we want
callback_list = [checkpoint, earlystopping, tensorboard, reduce_lr]
```

In [62]:
```
#compiling
opt = tf.keras.optimizers.Adam(learning_rate=0.0091)
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['accuracy', auc])

model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 250)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 250, 300)     13317900    input_2[0][0]
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_5 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_6 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_7 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 250, 4)       4880        embedding[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 26)        1352        input_3[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 1, 3)         18          input_4[0][0]
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 1, 3)         15          input_5[0][0]
__________________________________________________________________________________________________
embedding_4 (Embedding)         (None, 1, 26)        1326        input_6[0][0]
__________________________________________________________________________________________________
embedding_5 (Embedding)         (None, 1, 50)        19350       input_7[0][0]
__________________________________________________________________________________________________
input_8 (InputLayer)            [(None, 2)]          0
__________________________________________________________________________________________________
```

flatten (Flatten)	(None, 1000)	0	lstm[0][0]
flatten_1 (Flatten)	(None, 26)	0	embedding_1[0][0]
flatten_2 (Flatten)	(None, 3)	0	embedding_2[0][0]
flatten_3 (Flatten)	(None, 3)	0	embedding_3[0][0]
flatten_4 (Flatten)	(None, 26)	0	embedding_4[0][0]
flatten_5 (Flatten)	(None, 50)	0	embedding_5[0][0]
remain_dense (Dense)	(None, 48)	144	input_8[0][0]
concatenate (Concatenate)	(None, 1156)	0	flatten[0][0] flatten_1[0][0] flatten_2[0][0] flatten_3[0][0] flatten_4[0][0] flatten_5[0][0] remain_dense[0][0]
concat_dense_1 (Dense)	(None, 128)	148096	concatenate[0][0]
batch_normalization (BatchNorma	(None, 128)	512	concat_dense_1[0][0]
dropout (Dropout)	(None, 128)	0	batch_normalization[0][0]
concat_dense_2 (Dense)	(None, 64)	8256	dropout[0][0]
dropout_1 (Dropout)	(None, 64)	0	concat_dense_2[0][0]
concat_dense_3 (Dense)	(None, 32)	2080	dropout_1[0][0]
batch_normalization_1 (BatchNor	(None, 32)	128	concat_dense_3[0][0]
dropout_2 (Dropout)	(None, 32)	0	batch_normalization_1[0][0]
concat_dense_4 (Dense)	(None, 16)	528	dropout_2[0][0]
batch_normalization_2 (BatchNor	(None, 16)	64	concat_dense_4[0][0]
dropout_3 (Dropout)	(None, 16)	0	batch_normalization_2[0][0]
concat_dense_5 (Dense)	(None, 8)	136	dropout_3[0][0]

```
batch_normalization_3 (BatchNor (None, 8)            32          concat_dense_5[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 8)            0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
Output (Dense)                  (None, 2)            18          dropout_4[0][0]
==================================================================================================
Total params: 13,504,835
Trainable params: 186,567
Non-trainable params: 13,318,268
__________________________________________________________________________________________________
```

In [63]:

```
model.fit(x = [x_tr_essay, x_train_state_onehot, x_train_teacher_onehot, x_train_grade_onehot, x_train_clean_onehot,
          y = y_train,
          epochs = 80,
          batch_size = 24,
          validation_data = ([x_cv_essay, x_cv_state_onehot, x_cv_teacher_onehot, x_cv_grade_onehot, x_cv_clean_oneho
          callbacks = callback_list,
          class_weight=classWeight)
```

```
Epoch 1/80
2914/2914 [==============================] - 100s 23ms/step - loss: 0.7151 - accuracy: 0.5359 - auc: 0.5122 - val_los
s: 0.6834 - val_accuracy: 0.6218 - val_auc: 0.5761

Epoch 00001: val_accuracy improved from -inf to 0.62180, saving model to content/drive/MyDrive/model_save/weights-01-
0.6218.hdf5
Epoch 2/80
2914/2914 [==============================] - 66s 23ms/step - loss: 0.6941 - accuracy: 0.5131 - auc: 0.5437 - val_los
s: 0.6402 - val_accuracy: 0.8463 - val_auc: 0.5707

Epoch 00002: val_accuracy improved from 0.62180 to 0.84628, saving model to content/drive/MyDrive/model_save/weights-
02-0.8463.hdf5
Epoch 3/80
2914/2914 [==============================] - 66s 23ms/step - loss: 0.6925 - accuracy: 0.5246 - auc: 0.5438 - val_los
s: 0.6710 - val_accuracy: 0.7560 - val_auc: 0.6295

Epoch 00003: val_accuracy did not improve from 0.84628
Epoch 4/80
2914/2914 [==============================] - 66s 23ms/step - loss: 0.6768 - accuracy: 0.6211 - auc: 0.6289 - val_los
s: 0.4943 - val_accuracy: 0.8447 - val_auc: 0.6307

Epoch 00004: val_accuracy did not improve from 0.84628
Epoch 5/80
2914/2914 [==============================] - 67s 23ms/step - loss: 0.6592 - accuracy: 0.6458 - auc: 0.6589 - val_los
s: 0.6877 - val_accuracy: 0.6133 - val_auc: 0.6775
```

```
Epoch 00005: val_accuracy did not improve from 0.84628
Epoch 6/80
2914/2914 [==============================] - 66s 23ms/step - loss: 0.6469 - accuracy: 0.6622 - auc: 0.6852 - val_los
s: 0.7123 - val_accuracy: 0.5490 - val_auc: 0.6840

Epoch 00006: val_accuracy did not improve from 0.84628
Epoch 7/80
2914/2914 [==============================] - 66s 23ms/step - loss: 0.6395 - accuracy: 0.6842 - auc: 0.6880 - val_los
s: 0.6044 - val_accuracy: 0.7185 - val_auc: 0.6887

Epoch 00007: val_accuracy did not improve from 0.84628
Epoch 8/80
2914/2914 [==============================] - 66s 23ms/step - loss: 0.6308 - accuracy: 0.6787 - auc: 0.7045 - val_los
s: 0.6071 - val_accuracy: 0.6822 - val_auc: 0.6973

Epoch 00008: val_accuracy did not improve from 0.84628
Epoch 9/80
2914/2914 [==============================] - 67s 23ms/step - loss: 0.6261 - accuracy: 0.6765 - auc: 0.7197 - val_los
s: 0.6346 - val_accuracy: 0.6343 - val_auc: 0.7021

Epoch 00009: val_accuracy did not improve from 0.84628
Epoch 10/80
2914/2914 [==============================] - 66s 23ms/step - loss: 0.6237 - accuracy: 0.6939 - auc: 0.7259 - val_los
s: 0.6228 - val_accuracy: 0.6557 - val_auc: 0.6962

Epoch 00010: val_accuracy did not improve from 0.84628
Epoch 00010: early stopping
```

Out[63]:
```
<tensorflow.python.keras.callbacks.History at 0x7fcba261fb50>
```

In [67]:
```
test_outcomes_tfidf = model.evaluate([x_test_essay, x_test_state_onehot, x_test_teacher_onehot, x_test_grade_onehot,
                        y_test)

print('Test loss score:', test_outcomes_tfidf[0])
print('Test accuracy score:', test_outcomes_tfidf[1])
print('Test AUC score:', test_outcomes_tfidf[2])
```

```
683/683 [==============================] - 10s 14ms/step - loss: 0.6230 - accuracy: 0.6560 - auc: 0.6968
Test loss score: 0.6229678392410278
Test accuracy score: 0.6560183167457581
Test AUC score: 0.696783185005188
```

In [66]:
```
train_outcome = model.evaluate([x_tr_essay, x_train_state_onehot, x_train_teacher_onehot, x_train_grade_onehot, x_tra
```

```
                        y_train)

print('Train loss score:', train_outcome[0])
print('Train accuracy score:', train_outcome[1])
print('Train AUC score:', train_outcome[2])
```

```
2185/2185 [==============================] - 31s 14ms/step - loss: 0.6055 - accuracy: 0.6833 - auc: 0.7539
Train loss score: 0.6054765582084656
Train accuracy score: 0.6833290457725525
Train AUC score: 0.7538618445396423
```

In [68]:
```
cv_outcome = model.evaluate([x_cv_essay, x_cv_state_onehot, x_cv_teacher_onehot, x_cv_grade_onehot, x_cv_clean_onehot
                        y_cv)

print('Train loss score:', cv_outcome[0])
print('Train accuracy score:', cv_outcome[1])
print('Train AUC score:', cv_outcome[2])
```

```
547/547 [==============================] - 8s 14ms/step - loss: 0.6228 - accuracy: 0.6557 - auc: 0.7029
Train loss score: 0.6228119134902954
Train accuracy score: 0.6557208299636841
Train AUC score: 0.7028825283050537
```

In [71]:
```
%tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6006 (pid 1095), started 0:02:26 ago. (Use '!kill 1095' to kill it.)
```

MODEL 3

In [17]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

In [18]:
```
x_train,x_cv,y_train,y_cv=train_test_split(x_train, y_train, test_size=0.2)
```

In [19]:
```
print(x_train.shape)
print(x_test.shape)
print(x_cv.shape)
print(y_train.shape)
print(y_test.shape)
print(y_cv.shape)
```

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

```
(17480, 8)
(69918,)
(21850,)
(17480,)
```

In [61]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x_train['essay'])
vocab_size = len(t.word_index) + 1
print('vocab size:', vocab_size)
```

```
vocab size: 47315
```

In [63]:
```
tok_ = tokenizer.texts_to_sequences(x_train['essay'])
x_tr_essay = pad_sequences(tok_, maxlen=250, padding='post')
print(x_tr_essay.shape)

tok_ = tokenizer.texts_to_sequences(x_cv['essay'])
x_cv_essay = pad_sequences(tok_, maxlen=250, padding='post')
print(x_cv_essay.shape)

tok_ = tokenizer.texts_to_sequences(x_test['essay'])
x_test_essay = pad_sequences(tok_, maxlen=250, padding='post')
print(x_test_essay.shape)
```

```
(69918, 250)
(17480, 250)
(21850, 250)
```

In [64]:
```
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_cv_price_norm = normalizer.transform(x_cv['price'].values.reshape(-1,1))
x_test_price_norm = normalizer.transform(x_test['price'].values.reshape(-1,1))

print("Shape of Normalised Values of remaining input(addition of price and previously posted projects:")
print(x_train_price_norm.shape, y_train.shape)
print(x_cv_price_norm.shape, y_cv.shape)
print(x_test_price_norm.shape, y_test.shape)
```

```
Shape of Normalised Values of remaining input(addition of price and previously posted projects:
(69918, 1) (69918, 2)
(17480, 1) (17480, 2)
(21850, 1) (21850, 2)
```

In [65]:
```
normalizer.fit(x_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

x_train_projects_norm = normalizer.transform(x_train['teacher_number_of_previously_posted_projects'].values.reshape(1
x_cv_projects_norm = normalizer.transform(x_cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1).T
x_test_projects_norm = normalizer.transform(x_test['teacher_number_of_previously_posted_projects'].values.reshape(1,

print('Shape of Normalised Values of Previously posted projects by teachers:')
print(x_train_projects_norm.shape,y_train.shape)
print(x_cv_projects_norm.shape, y_cv.shape)
print(x_test_projects_norm.shape, y_test.shape)
print("="*100)
```

```
Shape of Normalised Values of Previously posted projects by teachers:
(69918, 1) (69918, 2)
(17480, 1) (17480, 2)
(21850, 1) (21850, 2)
====================================================================================================
```

In [67]:
```
vec = CountVectorizer(lowercase=False, binary=True)
vec.fit(preprocessed_data['school_state'].values)

state_tr_one = vec.transform(x_train['school_state'].values)
print("text to vector of state tr",state_tr_one.shape)

state_te_one = vec.transform(x_test['school_state'].values)
print("text to vector of state tre",state_te_one.shape)

state_cv_one = vec.transform(x_cv['school_state'].values)
print("text to vector of state cv",state_cv_one.shape)
```

```
text to vector of state tr (69918, 51)
text to vector of state tre (21850, 51)
text to vector of state cv (17480, 51)
```

In [68]:
```
vec = CountVectorizer(lowercase=False, binary=True)
vec.fit(preprocessed_data['teacher_prefix'].values)

teac_tr_one = vec.transform(x_train['teacher_prefix'].values)
print("text to vector of teacher tr",teac_tr_one.shape)

teac_te_one = vec.transform(x_test['teacher_prefix'].values)
print("text to vector of teacher te",teac_te_one.shape)
```

```
teac_cv_one = vec.transform(x_cv['teacher_prefix'].values)
print("text to vector of teacher cv",teac_cv_one.shape)
```

```
text to vector of teacher tr (69918, 5)
text to vector of teacher tr (21850, 5)
text to vector of teacher tr (17480, 5)
```

In [69]:
```
vec = CountVectorizer(lowercase=False, binary=True)
vec.fit(preprocessed_data['project_grade_category'].values)

# we use the fitted CountVectorizer to convert the text to vector
grade_tr_one = vec.transform(x_train['project_grade_category'].values)
print("text to vector of grade tr",grade_tr_one.shape)

grade_te_one = vec.transform(x_test['project_grade_category'].values)
print("text to vector of grade tr",grade_te_one.shape)

grade_cv_one = vec.transform(x_cv['project_grade_category'].values)
print("text to vector of grade tr",grade_cv_one.shape)
```

```
text to vector of grade tr (69918, 4)
text to vector of grade tr (21850, 4)
text to vector of grade tr (17480, 4)
```

In [70]:
```
vec = CountVectorizer(lowercase=False, binary=True)
vec.fit(preprocessed_data['clean_categories'].values)

clean_tr_one = vec.transform(x_train['clean_categories'].values)
print("text to vector of clean tr",clean_tr_one.shape)

clean_te_one = vec.transform(x_test['clean_categories'].values)
print("text to vector of clean te",clean_te_one.shape)

clean_cv_one = vec.transform(x_cv['clean_categories'].values)
print("text to vector of clean cv",clean_cv_one.shape)
```

```
text to vector of clean tr (69918, 9)
text to vector of clean te (21850, 9)
text to vector of clean cv (17480, 9)
```

In [71]:
```
vec = CountVectorizer(lowercase=False, binary=True)
vec.fit(preprocessed_data['clean_subcategories'].values)
```

```
subclean_tr_one = vec.transform(x_train['clean_subcategories'].values)
print("text to vector of subclean tr",subclean_tr_one.shape)

subclean_te_one = vec.transform(x_test['clean_subcategories'].values)
print("text to vector of subclean tr",subclean_te_one.shape)

subclean_cv_one = vec.transform(x_cv['clean_subcategories'].values)
print("text to vector of subclean cv",subclean_cv_one.shape)
```

```
text to vector of subclean tr (69918, 30)
text to vector of subclean tr (21850, 30)
text to vector of subclean cv (17480, 30)
```

In [23]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack


tr_all = hstack((x_train_price_norm,x_train_projects_norm,state_tr_one,teac_tr_one,grade_tr_one,clean_tr_one,subclear
te_all = hstack((x_test_price_norm,x_test_projects_norm,state_te_one,teac_te_one,grade_te_one,clean_te_one,subclean_t
cv_all = hstack((x_cv_price_norm,x_cv_projects_norm,state_cv_one,teac_cv_one,grade_cv_one,clean_cv_one,subclean_cv_or

print(tr_all.shape)
print(te_all.shape)
print(cv_all.shape)
```

```
(69918, 101)
(21850, 101)
(17480, 101)
```

In [24]:
```
y_train = np_utils.to_categorical(y_train)
y_cv= np_utils.to_categorical(y_cv)
y_test= np_utils.to_categorical(y_test)
```

In [28]:
```
x_tr_essay_emb = mat
```

In [45]:
```
tf.keras.backend.clear_session()
# Clear any logs from previous runs
!rm -rf ./logs/fit
```

In [47]:
```
input1_ = Input(shape=(x_tr_essay.shape[1],))
embedding1_ = Embedding(input_dim=x_tr_essay_emb.shape[0]+1, output_dim=300, trainable=False, input_length=x_tr_essay
```

```
lstm = LSTM(3, return_sequences=True)(embedding1_)
flatt_ = Flatten()(lstm)
```

In [48]:
```
x_out = Input(shape=(tr_all.shape[1],))
x_embbeding = Embedding(input_dim=tr_all.shape[1]+1, output_dim=64, input_length=tr_all.shape[0])(x_out)
x_convolution = Conv1D(32,3, padding='same', activation='relu', kernel_initializer='glorot_normal')(x_embbeding)
max_p = MaxPooling1D(3)(x_convolution)
flatt_2 = Flatten()(max_p)
```

In [49]:
```
x = concatenate([flatt_, flatt_2])
```

In [50]:
```
x = Dense(units=128,activation='relu',kernel_initializer=tf.keras.initializers.glorot_normal(12),name='dense1')(x)
x = Dropout(0.5)(x)
```

In [51]:
```
x = Dense(units=64,activation='relu',kernel_initializer=tf.keras.initializers.glorot_normal(16),name='dense2')(x)
x = Dropout(0.25)(x)
```

In [52]:
```
x = Dense(units=32,activation='relu',kernel_initializer=tf.keras.initializers.glorot_normal(seed=22),name='dense3')(x
```

In [53]:
```
x = Dense(units=2,activation='softmax',kernel_initializer=tf.keras.initializers.he_normal(seed=42),name='Output')(x)
```

In [54]:
```
#Creating a model
model = Model(inputs=[input1_, x_out], outputs = x)
model.run_eagerly = True
```

In [55]:
```
# https://colab.research.google.com/github/tensorflow/tensorboard/blob/master/docs/get_started.ipynb#scrollTo=Ao7fJW1
earlystop = EarlyStopping(monitor='val_accuracy',
                          min_delta=0.0001,
                          patience=2,
                          verbose=1,
                          mode='auto')
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy',
                              factor=0.9,
                              patience=1)

filepath="content/drive/MyDrive/model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5" #to save the model in respect
checkpoint_dir = os.path.dirname(filepath)
checkpoint = ModelCheckpoint(filepath=filepath,
                             monitor='val_accuracy',
```

```
                             verbose=1,
                             save_best_only=True,
                             mode='auto')



log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_freq=1)

callback_list = [checkpoint, earlystop, tensorboard_callback, reduce_lr]
opt = tf.keras.optimizers.Adam(learning_rate=0.015)
```

In [57]:

```
#compiling
opt = tf.keras.optimizers.Adam(learning_rate=0.015)
model.compile(optimizer=opt, loss='categorical_crossentropy',metrics=['accuracy', auc])

model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 101)]        0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 250)]        0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 101, 64)      6528        input_3[0][0]
__________________________________________________________________________________________________
embedding (Embedding)           (None, 250, 300)     14194800    input_2[0][0]
__________________________________________________________________________________________________
conv1d (Conv1D)                 (None, 101, 32)      6176        embedding_1[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 250, 3)       3648        embedding[0][0]
__________________________________________________________________________________________________
max_pooling1d (MaxPooling1D)    (None, 33, 32)       0           conv1d[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 750)          0           lstm[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 1056)         0           max_pooling1d[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 1806)         0           flatten[0][0]
                                                                 flatten_1[0][0]
__________________________________________________________________________________________________
dense1 (Dense)                  (None, 128)          231296      concatenate[0][0]
```

```
dropout (Dropout)               (None, 128)          0           dense1[0][0]
__________________________________________________________________________________________________
dense2 (Dense)                  (None, 64)           8256        dropout[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 64)           0           dense2[0][0]
__________________________________________________________________________________________________
dense3 (Dense)                  (None, 32)           2080        dropout_1[0][0]
__________________________________________________________________________________________________
Output (Dense)                  (None, 2)            66          dense3[0][0]
==================================================================================================
Total params: 14,452,850
Trainable params: 258,050
Non-trainable params: 14,194,800
__________________________________________________________________________________________________
```

In [41]:
```
model.fit(x = [x_tr_essay, tr_all.todense()], y = y_train,epochs = 50,batch_size = 128,validation_data = ([x_cv_essay
```

```
Epoch 1/50
547/547 [==============================] - 94s 172ms/step - loss: 0.3885 - accuracy: 0.8489 - auc: 0.7165 - val_loss:
0.3961 - val_accuracy: 0.8458 - val_auc: 0.7015

Epoch 00001: val_accuracy did not improve from 0.84628
Epoch 2/50
547/547 [==============================] - 93s 169ms/step - loss: 0.3826 - accuracy: 0.8497 - auc: 0.7318 - val_loss:
0.4036 - val_accuracy: 0.8449 - val_auc: 0.6974

Epoch 00002: val_accuracy did not improve from 0.84628
Epoch 3/50
547/547 [==============================] - 93s 171ms/step - loss: 0.3757 - accuracy: 0.8512 - auc: 0.7431 - val_loss:
0.3960 - val_accuracy: 0.8461 - val_auc: 0.6996

Epoch 00003: val_accuracy did not improve from 0.84628
Epoch 4/50
547/547 [==============================] - 93s 170ms/step - loss: 0.3713 - accuracy: 0.8526 - auc: 0.7514 - val_loss:
0.3990 - val_accuracy: 0.8456 - val_auc: 0.7050

Epoch 00004: val_accuracy did not improve from 0.84628
Epoch 5/50
547/547 [==============================] - 93s 170ms/step - loss: 0.3626 - accuracy: 0.8559 - auc: 0.7658 - val_loss:
0.4028 - val_accuracy: 0.8456 - val_auc: 0.7010

Epoch 00005: val_accuracy did not improve from 0.84628
Epoch 00005: early stopping
<tensorflow.python.keras.callbacks.History at 0x7ff3bb1318d0>
```

Out[41]:

In [42]:
```
tr_result_3 = model.evaluate([x_test_essay, te_all.todense()],y_test)

print('The result of train loss is:', tr_result_3[0])
print('The result of train accuracy is:', tr_result_3[1])
print('The result of train auc is:', tr_result_3[2])
```

```
683/683 [==============================] - 19s 28ms/step - loss: 0.3926 - accuracy: 0.8506 - auc: 0.7027
The result of train loss is: 0.39256587624549866
The result of train accuracy is: 0.8506178259849548
The result of train auc is: 0.7027316093444824
```

In [43]:
```
te_result_3 = model.evaluate([x_tr_essay, tr_all.todense()],y_train)

print('The result of test loss is:', te_result_3[0])
print('The result of test accuracy is:', te_result_3[1])
print('The result of test auc is:', te_result_3[2])
```

```
2185/2185 [==============================] - 61s 28ms/step - loss: 0.3451 - accuracy: 0.8542 - auc: 0.8022
The result of test loss is: 0.3450755476951599
The result of test accuracy is: 0.8541577458381653
The result of test auc is: 0.8022053241729736
```

In [44]:
```
cv_result_3 = model.evaluate([x_cv_essay, cv_all.todense()],y_cv)

print('The result of cv loss is:', cv_result_3[0])
print('The result of cv loss is:', cv_result_3[1])
print('The result of cv loss is:', cv_result_3[2])
```

```
547/547 [==============================] - 15s 28ms/step - loss: 0.4028 - accuracy: 0.8456 - auc: 0.6953
The result of cv loss is: 0.40276384353637695
The result of cv loss is: 0.8455949425697327
The result of cv loss is: 0.6953148245811462
```

In [40]:
```
%tensorboard --logdir logs/fit
```

```
UsageError: Line magic function `%tensorboard` not found.
```

In [ ]:
```
```