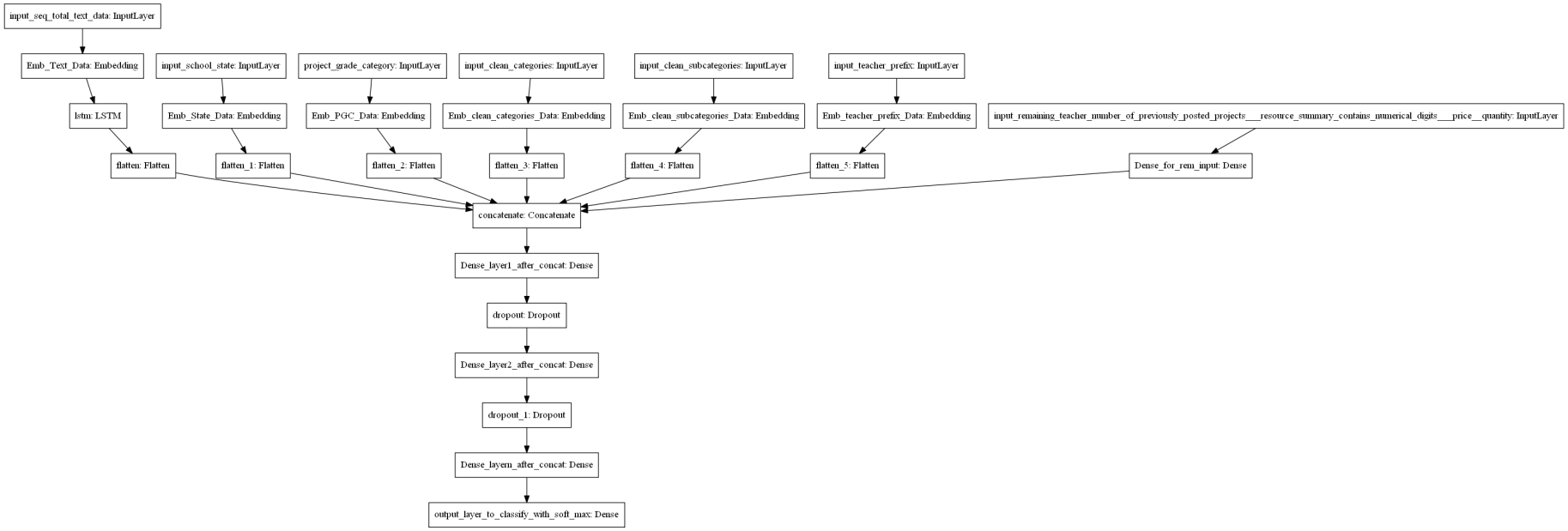
Assignment : 14

1. Preprocess all the Data we have in DonorsChoose Dataset use train.csv
2. Combine 4 essay's into one column named - 'preprocessed_essays'.
3. After step 2 you have to train 3 types of models as discussed below.
4. For all the model use 'auc' as a metric. check this for using auc as a metric
5. You are free to choose any number of layers/hiddden units but you have to use same type of architectures shown below.
6. You can use any one of the optimizers and choice of Learning rate and momentum, resources: cs231n class notes, cs231n class video.
7. For all the model's use TensorBoard and plot the Metric value and Loss with epoch. While submitting, take a screenshot of plots and include those images in .ipynb notebook and PDF.
8. Use Categorical Cross Entropy as Loss to minimize.

Model-1

Build and Train deep neural network as shown below



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

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

- For LSTM, you can choose your sequence padding methods on your own or you can train your LSTM without padding, there is no restriction on that.

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

In [ ]:
```
# https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-layer-work
input_layer = Input(shape=(n,))
embedding = Embedding(no_1, no_2, input_length=n)(input_layer)
flatten = Flatten()(embedding)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-ed7dba31d057> in <module>()
      1 # https://stats.stackexchange.com/questions/270546/how-does-keras-embedding-layer-work
----> 2 input_layer = Input(shape=(n,))
      3 embedding = Embedding(no_1, no_2, input_length=n)(input_layer)
      4 flatten = Flatten()(embedding)

NameError: name 'Input' is not defined
```

1. Go through this blog, if you have any doubt on using predefined Embedding values in Embedding layer - https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

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

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

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

In [ ]:
```
import numpy as np
import pandas as pd
```

In [ ]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

In [ ]:
```
processed_data=pd.read_csv('/content/drive/MyDrive/Applied ai/for_colab/preprocessed_data.csv')
```

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

Out[ ]:

	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 [ ]:
```
processed_data.shape
```

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

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

In [ ]:
```
print(X.shape, y.shape)
```

```
(109248, 8) (109248,)
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.22, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.22, stratify=y_train)
```

In [ ]:
```
print(X_train.shape, X_test.shape, X_cv.shape)
```

```
(66466, 8) (24035, 8) (18747, 8)
```

In [ ]:
```
X_train.head(3)
```

Out[ ]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	
42206	me	mrs	grades_prek_2	1	music_arts	music	in
25129	nv	teacher	grades_prek_2	0	health_sports	health_wellness	
78668	oh	mrs	grades_prek_2	1	literacy_language	literacy literature_writing	co

Essay encoding:

In [ ]:
```
essay_encode_train=X_train['essay'].tolist()
essay_encode_test=X_test['essay'].tolist()
essay_encode_cv=X_cv['essay'].tolist()
```

In [ ]:
```
len(essay_encode_cv)
```

Out[ ]:
```
18747
```

In [ ]:
```
token=tf.keras.preprocessing.text.Tokenizer()
```

In [ ]:
```
token.fit_on_texts(essay_encode_train)
```

In [ ]:
```
type(token.word_index)
```

Out[ ]:
```
dict
```

In [ ]:
```
encoded_essay_train=token.texts_to_sequences(essay_encode_train)
encoded_essay_test=token.texts_to_sequences(essay_encode_test)
encoded_essay_cv=token.texts_to_sequences(essay_encode_cv)
```

In [ ]:
```
vocab_size=len(token.word_index)+1
```

In [ ]:
```
vocab_size
```

Out[ ]:
```
46325
```

In [ ]:
```
len(token.index_word), len(token.word_index)
```

Out[ ]:
```
(46324, 46324)
```

In [ ]:
```
glove_mat=np.zeros((vocab_size, 300))
```

In [ ]:
```
glove_mat
```

Out[ ]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [ ]:
```
glove_mat.shape
```

Out[ ]:
```
(46325, 300)
```

In [ ]:
```
len(encoded_essay_train)
```

Out[ ]:
```
66466
```

In [ ]:
```
len(X['essay']), len(X['essay'].tolist())
```

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

In [ ]:
```
len(encoded_essay_train[5]), len(encoded_essay_train[100])
```

Out[ ]:
```
(126, 158)
```

In [ ]:
```
mx_len=0
for ele in encoded_essay_train:
```

```
    if len(ele)>mx_len:
        mx_len=len(ele)
print(mx_len)
```

```
333
```

In [ ]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences
essay_padded_train=pad_sequences(encoded_essay_train, maxlen=mx_len, padding='pre')
essay_padded_test=pad_sequences(encoded_essay_test, maxlen=mx_len, padding='pre')
essay_padded_cv=pad_sequences(encoded_essay_cv, maxlen=mx_len, padding='pre')
```

In [ ]:
```
len(essay_padded_train), len(essay_padded_test), len(essay_padded_cv)
```

Out[ ]:
```
(66466, 24035, 18747)
```

In [ ]:
```
essay_padded_train[:1]
```

Out[ ]:
```
array([[    0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     4,     1,    19,  1483,   115,   116,   618,   694,
          119,    72,   153,   170,    20,   708,    20,    31,    75,
           56,   154,   862,  3912,    12,     1,   184,   649,   153,
         1431,   618,  4266,    10,    14,   105,   231,     1,     3,
```

```
       12,    31,     1,  4926,    96,     3,   432,   309,    12,
        1,     8,     5,    92,    74,    35,    28,   333,     6,
      168,    74,    35,     2,   360,    33,     6,   168,    98,
      219,  3433,   144,   268,   414,   618,     8, 13262,  1160,
        2,   137,    87,   713,   168,   257,   427,   618,   222,
        5,   168,     6,     7,   465,  2185,  2516,   185,   412,
      351,   312,  2750,    57,    18,   618,   317,  1261,    85,
        1,    88,   990,    22,   168,    23,   432,  1013,   268,
      414,   618,  1577,  2185,    11,    96,   184,  2544,  2409,
     5600,   168,    11,    96,    56,   890,  2226,   890,  3850,
    10358,   456,    17,    14,  1228,  1913,     1,   816,   167,
      901,   160,  4785,   492,  1428,  2443,     1,   512,   882,
       10,   193,   224,   859,   618,    29,   419,    31,   465,
     2185,  1331,   182,  1146,   618,   372,  1717,   214,   777,
      495,   961,   738,   618,   209,     1,    34,    10,   116,
     2185,   756,   164,     2,   296,    33,   991,     6,    13]],
    dtype=int32)
```

In [ ]:
```
len(essay_padded_train[5])
```

Out[ ]:
```
333
```

In [ ]:
```
len(essay_padded_test[5])
```

Out[ ]:
```
333
```

In [ ]:
```
try:
    import dill as pickle
except ImportError:
    import pickle
```

In [ ]:
```
with open('/content/drive/MyDrive/Applied ai/for_colab/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [ ]:
```
#glove_words
```

In [ ]:
```
model['catalyzing']
```

Out[ ]:
```
array([ 3.7284e-01, -1.6815e-01,  6.7185e-02,  1.5902e-02, -4.0426e-01,
       -7.0171e-01, -1.6178e-01,  6.3225e-01,  5.7326e-01, -1.4164e-01,
       -3.4372e-01,  3.8187e-01,  3.8026e-01,  6.9173e-02, -8.1617e-02,
       -1.1243e-01, -4.3106e-01, -1.4534e-01,  3.2936e-02, -4.6387e-02,
        3.4674e-01,  2.6480e-01, -1.0388e-01,  4.3699e-02,  1.6757e-01,
        1.7132e-01,  8.7222e-02, -4.3251e-01,  1.4380e-01,  1.6029e-01,
        2.4105e-02,  3.0156e-01,  2.6712e-01, -9.2920e-02,  1.0270e-01,
       -2.3622e-03,  2.7963e-02, -3.5531e-01,  6.1821e-01, -4.8037e-01,
       -2.8285e-01, -2.8797e-02,  6.2059e-01, -4.7502e-01, -3.3531e-01,
       -5.4226e-01,  6.3721e-01, -4.5038e-01, -1.9627e-01, -3.1119e-01,
       -2.4223e-01,  2.5410e-01, -2.6833e-01, -4.5190e-01, -5.7673e-02,
        3.1329e-01, -3.2769e-01,  2.1680e-01,  4.5747e-02,  1.8459e-01,
       -4.6649e-01,  4.3855e-01, -3.6516e-01,  1.1623e-01,  3.9441e-01,
       -2.8156e-01, -2.1946e-01,  7.4481e-02, -4.5509e-01,  4.6715e-02,
       -1.3627e-01,  3.5227e-01, -4.6264e-01, -8.7228e-01, -3.5898e-01,
       -3.2485e-01, -1.0506e-01,  1.3512e-01,  4.7139e-02, -5.0308e-01,
       -6.7541e-01, -3.1775e-01, -4.3901e-01,  2.7041e-01, -1.0283e-01,
       -5.9191e-01,  8.2085e-01, -5.6543e-02, -2.5215e-01, -2.2431e-01,
        3.1225e-02, -3.0353e-01, -2.2095e-01, -3.3989e-01,  2.6049e-02,
        5.0759e-01, -3.8674e-01, -2.7844e-01, -2.2618e-01,  9.0464e-02,
        1.5203e-01,  2.0260e-01,  2.2809e-01,  2.7093e-01, -4.1994e-01,
       -7.4791e-01, -1.6576e-01,  4.6991e-02,  8.6495e-01, -4.0607e-01,
       -1.6512e-01,  2.5455e-02, -3.5758e-01, -1.5259e-01,  6.2355e-01,
        2.2686e-01,  2.9290e-01,  2.5683e-01,  6.5252e-01,  2.3173e-01,
       -7.6061e-02,  3.9020e-01,  2.4567e-01, -2.1195e-01, -5.6482e-01,
        8.1335e-01, -2.0213e-01,  4.2573e-01,  1.0810e-01, -3.1317e-01,
        1.2821e-01, -9.0856e-02, -1.3879e-01, -4.3852e-01,  7.1853e-01,
       -9.4413e-01, -2.6379e-01,  4.7286e-02, -3.2667e-01, -2.9269e-02,
        5.4736e-02,  6.8003e-02, -3.9693e-01,  1.5682e-02, -4.2927e-01,
        4.3325e-01, -2.4153e-01,  2.4661e-01,  6.9286e-02, -8.0466e-02,
        4.1837e-01, -1.7858e-01,  9.7569e-01,  7.8728e-01, -4.2635e-01,
        3.3002e-01,  2.1009e-01,  1.4372e-01, -7.3485e-01,  6.1252e-02,
        3.3738e-01, -5.4760e-01, -2.5525e-01, -6.8680e-03,  1.0259e-01,
       -2.6133e-01,  1.8873e-02, -1.2772e-01, -4.4678e-01, -8.3848e-01,
        1.0628e-01, -2.5023e-01,  1.6144e-02, -1.9646e-01,  5.1523e-03,
        3.3254e-01,  2.7758e-01,  5.8486e-01, -1.5373e-01, -4.2149e-02,
        4.0612e-02, -7.1638e-02, -1.2744e-01,  5.3809e-01, -1.5832e-01,
        1.9614e-01, -2.1025e-01,  7.6517e-02,  4.7336e-01, -2.2802e-01,
```

```
-1.3411e-01,  3.4369e-01, -4.4268e-01,  6.2527e-01,  1.0734e-01,
 9.4143e-02, -3.2357e-01, -1.3582e-01,  2.3147e-01,  2.0890e-01,
 1.1875e-01,  4.2238e-01,  1.8666e-01, -1.7836e-01, -4.1608e-01,
 2.0040e-01,  4.9170e-01, -2.9802e-01,  3.1076e-01, -4.1989e-01,
-1.0546e+00,  4.4626e-01, -4.3224e-01, -8.6584e-02, -7.4392e-01,
 4.5178e-01,  1.9603e-01, -3.8665e-01, -2.7485e-01,  3.9382e-01,
 1.2915e-01,  1.4542e-01,  2.4649e-01,  6.3603e-01,  1.0424e+00,
 2.0561e-01,  7.9861e-01,  2.9076e-01,  4.7562e-01, -2.7262e-01,
 2.4787e-01,  2.0756e-01,  7.3792e-02, -6.1109e-02, -5.5458e-01,
 1.8075e-01, -2.3883e-01, -2.0367e-01,  1.5619e-01,  1.0563e-01,
-1.3270e-01, -6.3675e-02, -1.5188e-01, -1.6803e-01, -3.0585e-01,
-1.3330e-02,  4.0105e-01, -7.2343e-01,  5.9227e-02, -4.5321e-01,
-3.1926e-01,  9.8133e-02, -1.3011e-01,  2.2908e-01,  6.5397e-01,
 3.8928e-01,  2.7809e-01, -7.0554e-01,  4.1436e-02, -1.4268e-01,
 1.8744e-01,  1.1471e-01,  1.6238e-01,  3.5406e-01, -8.8451e-02,
 6.5220e-02,  4.7835e-01, -3.7479e-01,  3.5724e-01, -1.7613e-01,
-6.3171e-01, -2.1878e-02, -5.4250e-02,  8.9051e-02,  5.4039e-01,
-2.4508e-01, -5.2191e-02,  1.8201e-01, -6.7788e-01, -1.1583e-01,
 2.6440e-02,  4.4800e-01, -1.9153e-01, -2.3555e-01, -2.2194e-01,
-3.8872e-01,  4.5033e-02,  4.9640e-01,  2.8074e-01,  1.0482e-04,
-3.0174e-01, -6.5126e-01, -1.7654e-01,  5.4076e-01,  1.4305e-01,
-6.4243e-01,  3.9328e-01, -1.3775e-01, -6.0606e-01, -4.7656e-01])
```

In [ ]:
```
model['catalyzing'].shape
```

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

In [ ]:
```
len(model)
```

Out[ ]:
```
51510
```

In [ ]:
```
non_glove_words=[]
for index, word in token.index_word.items():
    if word in glove_words:
        glove_mat[index]=model[word]

    else:
        non_glove_words.append(word)
print('Totally ',len(non_glove_words),' words are not there in glove')
```

```
Totally  4303  words are not there in glove
```

In [ ]:
```
#non_glove_words
```

In [ ]:
```
glove_mat
```

Out[ ]:
```
array([[ 0.       ,  0.       ,  0.       , ...,  0.       ,  0.       ,
         0.       ],
       [ 0.15243  , -0.16945  , -0.022748 , ...,  0.61801  ,  0.41281  ,
         0.0010077],
       [-0.043504 , -0.18484  , -0.14613  , ...,  0.1008   ,  0.1068   ,
         0.089065 ],
       ...,
       [ 0.23381  ,  0.36567  ,  0.1353   , ..., -0.0093819, -0.35703  ,
        -0.57934  ],
       [-0.50861  ,  0.28529  ,  0.02926  , ..., -0.12062  ,  0.1823   ,
         0.24667  ],
       [ 0.47401  , -0.52709  , -0.37333  , ...,  0.036867 , -0.29506  ,
         0.21174  ]])
```

In [ ]:
```
glove_mat.shape
```

Out[ ]:
```
(46325, 300)
```

School_state encoding:

In [ ]:
```
schoolstate_train=X_train['school_state'].tolist()
```

In [ ]:
```
len(schoolstate_train)
```

Out[ ]:
```
66466
```

In [ ]:
```
len(set(schoolstate_train))
```

Out[ ]:
```
51
```

In [ ]:
```
schoolstate_len=len(set(schoolstate_train))
```

In [ ]:
```
from tensorflow.keras.preprocessing.text import one_hot
ohe_school_state_train=[one_hot(state, schoolstate_len) for state in X_train['school_state'].tolist()]
ohe_school_state_test=[one_hot(state, schoolstate_len) for state in X_test['school_state'].tolist()]
ohe_school_state_cv=[one_hot(state, schoolstate_len) for state in X_cv['school_state'].tolist()]
```

In [ ]:
```
ohe_school_state_train[5]
```

Out[ ]:
```
[12]
```

In [ ]:
```
len(ohe_school_state_train[5])
```

Out[ ]:
```
1
```

Project_grade:

In [ ]:
```
prjectgrade_len=len(set(X_train['project_grade_category'].tolist()))
ohe_project_grade_train=[one_hot(project_grade, prjectgrade_len, filters = '') for project_grade in X_train['project_
ohe_project_grade_test=[one_hot(project_grade, prjectgrade_len, filters = '') for project_grade in X_test['project_gr
ohe_project_grade_cv=[one_hot(project_grade, prjectgrade_len, filters = '') for project_grade in X_cv['project_grade_
```

In [ ]:
```
ohe_project_grade_train[:5]
```

Out[ ]:
```
[[1], [1], [1], [1], [1]]
```

Clean_categories:

In [ ]:
```
cleancate_lst=X_train['clean_categories'].tolist()
cleancate_join=' '.join(cleancate_lst)
cleancate_len=len(set(cleancate_join.split()))
clean_cate_train=[one_hot(clean_cate, cleancate_len, filters = ' ') for clean_cate in X_train['clean_categories'].tol
clean_cate_test=[one_hot(clean_cate, cleancate_len, filters = ' ') for clean_cate in X_test['clean_categories'].tolis
clean_cate_cv=[one_hot(clean_cate, cleancate_len, filters = ' ') for clean_cate in X_cv['clean_categories'].tolist()]
```

In [ ]:
```
cleancate_len
```

Out[ ]:
```
9
```

In [ ]:
```
len(clean_cate_train)
```

Out[ ]:
```
66466
```

In [ ]:
```
len(clean_cate_train[200])
```

Out[ ]:
```
2
```

In [ ]:
```
cleancate_mxln=0
val=''
for ele in clean_cate_train:
    if len(ele)>cleancate_mxln:
        val=ele
        cleancate_mxln=len(ele)
```

In [ ]:
```
cleancate_mxln
```

Out[ ]:
```
3
```

In [ ]:
```
val
```

Out[ ]:
```
[2, 1, 5]
```

In [ ]:
```
padded_cleancate_train = pad_sequences(clean_cate_train, maxlen=cleancate_mxln)
padded_cleancate_test = pad_sequences(clean_cate_test, maxlen=cleancate_mxln)
padded_cleancate_cv = pad_sequences(clean_cate_cv, maxlen=cleancate_mxln)
```

In [ ]:
```
padded_cleancate_train[:5]
```

Out[ ]:
```
array([[0, 0, 1],
       [0, 0, 4],
       [0, 0, 5],
       [0, 5, 2],
       [0, 0, 5]], dtype=int32)
```

Clean_subcategory:

In [ ]:
```
# cleansubcate_len=len(set(X_train['clean_subcategories'].tolist()))
cleansubcate_lst=X_train['clean_subcategories'].tolist()
cleansubcate_join=' '.join(cleansubcate_lst)
cleansubcate_len=len(set(cleansubcate_join.split()))
clean_subcate_train=[one_hot(clean_subcate, cleansubcate_len) for clean_subcate in X_train['clean_subcategories'].tol
clean_subcate_test=[one_hot(clean_subcate, cleansubcate_len) for clean_subcate in X_test['clean_subcategories'].tolis
clean_subcate_cv=[one_hot(clean_subcate, cleansubcate_len) for clean_subcate in X_cv['clean_subcategories'].tolist()]
```

In [ ]:
```
clean_subcate_train[:5]
```

Out[ ]:
```
[[25], [11, 19], [22, 25, 12], [25, 12, 19], [3, 25, 12]]
```

In [ ]:
```
cleansubcate_len
```

Out[ ]:
```
30
```

In [ ]:
```
cleansubcate_mxln=0
val2=''
for ele in clean_subcate_train:
```

```
    if len(ele)>cleansubcate_mxln:
        val2=ele
        cleansubcate_mxln=len(ele)
```

In [ ]:
```
cleansubcate_mxln
```

Out[ ]:
```
5
```

In [ ]:
```
val2
```

Out[ ]:
```
[25, 12, 11, 21, 15]
```

In [ ]:
```
padded_cleansubcate_train = pad_sequences(clean_subcate_train, maxlen=cleansubcate_mxln)
padded_cleansubcate_test = pad_sequences(clean_subcate_test, maxlen=cleansubcate_mxln)
padded_cleansubcate_cv = pad_sequences(clean_subcate_cv, maxlen=cleansubcate_mxln)
```

In [ ]:
```
padded_cleansubcate_train[:5]
```

Out[ ]:
```
array([[ 0,  0,  0,  0, 25],
       [ 0,  0,  0, 11, 19],
       [ 0,  0, 22, 25, 12],
       [ 0,  0, 25, 12, 19],
       [ 0,  0,  3, 25, 12]], dtype=int32)
```

Teacher_prefix:

In [ ]:
```
X_train['teacher_prefix'].value_counts()
```

Out[ ]:
```
mrs        34933
ms         23637
mr          6476
teacher     1409
dr            11
Name: teacher_prefix, dtype: int64
```

In [ ]:
```
#X_train['teacher_prefix'].tolist()
```

In [ ]:
```
X_train['teacher_prefix'].values
```

Out[ ]:
```
array(['mrs', 'teacher', 'mrs', ..., 'mr', 'mrs', 'mrs'], dtype=object)
```

In [ ]:
```
teacher_prefix_len=len(set(X_train['teacher_prefix'].tolist()))
teacher_prefix_train=[one_hot(prefix, teacher_prefix_len) for prefix in X_train['teacher_prefix'].tolist()]
teacher_prefix_test=[one_hot(prefix, teacher_prefix_len) for prefix in X_test['teacher_prefix'].tolist()]
teacher_prefix_cv=[one_hot(prefix, teacher_prefix_len) for prefix in X_cv['teacher_prefix'].tolist()]
```

In [ ]:
```
#teacher_prefix_len
```

Numerical_features:

In [ ]:
```
remain_numeric_train = X_train[['teacher_number_of_previously_posted_projects', 'price']].values
remain_numeric_test = X_test[['teacher_number_of_previously_posted_projects', 'price']].values
remain_numeric_cv = X_cv[['teacher_number_of_previously_posted_projects', 'price']].values
```

In [ ]:
```

```

In [ ]:
```
from tensorflow.keras.layers import Input, Embedding, LSTM, Concatenate, Dense,Dropout, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

In [ ]:
```
essay_padded_train.shape
```

Out[ ]:
```
(66466, 333)
```

In [ ]:
```
vocab_size
```

Out[ ]:
```
46325
```

In [ ]:
```
mx_len
```

Out[ ]:
```
333
```

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

Out[ ]:
```
numpy.ndarray
```

In [ ]:
```
essay_padded_train.shape
```

Out[ ]:
```
(66466, 333)
```

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

Out[ ]:
```
numpy.ndarray
```

In [ ]:
```
type(essay_padded_train), type(ohe_school_state_train), type(ohe_project_grade_train), type(padded_cleancate_train),
```

Out[ ]:
```
(numpy.ndarray, list, list, numpy.ndarray, numpy.ndarray, list, numpy.ndarray)
```

Model-1

In [ ]:
```
def lr_update(epoch,lr):
    if epoch%5 == 0 and lr>1e-4:
        return lr - (0.1*lr)
    else:
        return lr

from tensorflow.keras.callbacks import ModelCheckpoint
```

```
from tensorflow.keras.callbacks import TerminateOnNaN
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.callbacks import TensorBoard
import datetime
path="model1.hdf5"
log_dir="logs1/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = TensorBoard(log_dir=log_dir,histogram_freq=1, write_graph=True,write_grads=True)
learning_rate_callback = LearningRateScheduler(lr_update,verbose=1)
model_callback = ModelCheckpoint(filepath=path, monitor='val_auc',  verbose=1, save_best_only=True, mode='max')
early_callback = EarlyStopping(monitor="val_auc",patience=10,mode='auto')
```

```
WARNING:tensorflow:`write_grads` will be ignored in TensorFlow 2.0 for the `TensorBoard` Callback.
```

In [ ]:

```
from sklearn.metrics import roc_auc_score
def auc(y_true,y_pred):
    return tf.py_function(roc_auc_score,(y_true,y_pred),tf.double)
```

In [ ]:

```
essay_input=Input(shape=(mx_len,))
essay_embed = Embedding(vocab_size, 300, input_length=mx_len,weights=[glove_mat],trainable = False)(essay_input)
essay_lstm=LSTM(15)(essay_embed)
flatten_lstm = Flatten(data_format='channels_last',name='flatten')(essay_lstm)

input_school_state = Input(shape=(1,))
emb_school_state = Embedding(schoolstate_len, 3, input_length=1)(input_school_state)
flatten_school_state = Flatten(data_format='channels_last',name='flatten_school_state')(emb_school_state)

input_proj_grade = Input(shape=(1,))
emb_proj_grade = Embedding(prjectgrade_len, 3, input_length=1)(input_proj_grade)
flatten_proj_grade = Flatten(data_format='channels_last',name='flatten_proj_grade')(emb_proj_grade)

input_clean_cate = Input(shape=(cleancate_mxln,))
emb_clean_cate = Embedding(cleancate_len, 5, input_length=cleancate_mxln)(input_clean_cate)
flatten_clean_cate = Flatten(data_format='channels_last',name='flatten_clean_cate')(emb_clean_cate)

input_clean_subcate = Input(shape=(cleansubcate_mxln,))
emb_clean_subcate = Embedding(cleansubcate_len, 5, input_length=cleansubcate_mxln)(input_clean_subcate)
flatten_clean_subcate = Flatten(data_format='channels_last',name='flatten_clean_subcate')(emb_clean_subcate)
```

```
input_teacher_prefix = Input(shape=(1,))
emb_teacher_prefix = Embedding(teacher_prefix_len, 3, input_length=1)(input_teacher_prefix)
flatten_teacher_prefix = Flatten(data_format='channels_last',name='flatten_teacher_prefix')(emb_teacher_prefix)

input_numeric = Input(shape=(2,))
numeric_layer = Dense(16, activation='relu')(input_numeric)


conc = Concatenate(axis=1)([flatten_lstm,flatten_school_state,flatten_proj_grade,flatten_clean_cate,flatten_clean_sub

#dense
FC1 = Dense(512,activation='relu',kernel_initializer='he_normal')(conc)

#dropout
drop1 = Dropout(0.4)(FC1)

#dense
FC2 = Dense(256,activation='relu',kernel_initializer='he_normal')(drop1)

#dropout
drop2 = Dropout(0.35)(FC2)

#dense
FC3 =Dense(64,activation='relu',kernel_initializer='he_normal')(drop2)

#output
output = Dense(1,activation='sigmoid',kernel_initializer='glorot_normal')(FC3)
```

In [ ]:

```
model1= Model(inputs=[essay_input, input_school_state,input_proj_grade,input_clean_cate,input_clean_subcate,input_tea
```

In [ ]:

```
model1.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 333)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 333, 300)     13897500    input_1[0][0]
```

input_2 (InputLayer)	[(None, 1)]	0	
input_3 (InputLayer)	[(None, 1)]	0	
input_4 (InputLayer)	[(None, 3)]	0	
input_5 (InputLayer)	[(None, 5)]	0	
input_6 (InputLayer)	[(None, 1)]	0	
lstm (LSTM)	(None, 15)	18960	embedding[0][0]
embedding_1 (Embedding)	(None, 1, 3)	153	input_2[0][0]
embedding_2 (Embedding)	(None, 1, 3)	12	input_3[0][0]
embedding_3 (Embedding)	(None, 3, 5)	45	input_4[0][0]
embedding_4 (Embedding)	(None, 5, 5)	150	input_5[0][0]
embedding_5 (Embedding)	(None, 1, 3)	15	input_6[0][0]
input_7 (InputLayer)	[(None, 2)]	0	
flatten (Flatten)	(None, 15)	0	lstm[0][0]
flatten_school_state (Flatten)	(None, 3)	0	embedding_1[0][0]
flatten_proj_grade (Flatten)	(None, 3)	0	embedding_2[0][0]
flatten_clean_cate (Flatten)	(None, 15)	0	embedding_3[0][0]
flatten_clean_subcate (Flatten)	(None, 25)	0	embedding_4[0][0]
flatten_teacher_prefix (Flatten	(None, 3)	0	embedding_5[0][0]
dense (Dense)	(None, 16)	48	input_7[0][0]
concatenate (Concatenate)	(None, 80)	0	flatten[0][0] flatten_school_state[0][0] flatten_proj_grade[0][0]

```
                                                                 flatten_clean_cate[0][0]
                                                                 flatten_clean_subcate[0][0]
                                                                 flatten_teacher_prefix[0][0]
                                                                 dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 512)           41472       concatenate[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 512)           0           dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 256)           131328      dropout[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 256)           0           dense_2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 64)            16448       dropout_1[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 1)             65          dense_3[0][0]
==================================================================================================
Total params: 14,106,196
Trainable params: 208,696
Non-trainable params: 13,897,500
__________________________________________________________________________________________________
```

In [ ]:

```
# !pip install pydot
# !pip install graphviz
```

In [ ]:

```
from tensorflow.keras.utils import plot_model
```

In [ ]:

```
#https://machinelearningmastery.com/visualize-deep-learning-neural-network-model-keras/
# from tensorflow.keras.utils.vis_utils import plot_model
plot_model(model1, to_file='model1_plot.png', show_shapes=True, show_layer_names=True)
```

Out[ ]:

In [ ]:

```
from tensorflow.keras.optimizers import Adam
model1.compile(optimizer=Adam(learning_rate=0.001),loss='binary_crossentropy',metrics=['accuracy',auc])
```

In [ ]:

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

In [ ]:

```
model1.fit([essay_padded_train,np.array(ohe_school_state_train),np.array(ohe_project_grade_train),padded_cleancate_tr
```

```
Epoch 1/20

Epoch 00001: LearningRateScheduler setting learning rate to 0.0009000000427477062.
1039/1039 [==============================] - 78s 68ms/step - loss: 0.7157 - accuracy: 0.8283 - auc: 0.5280 - val_los
s: 0.4191 - val_accuracy: 0.8486 - val_auc: 0.5865

Epoch 00001: val_auc improved from -inf to 0.58653, saving model to model1.hdf5
```

```
Epoch 2/20

Epoch 00002: LearningRateScheduler setting learning rate to 0.0009000000427477062.
1039/1039 [==============================] - 69s 66ms/step - loss: 0.4281 - accuracy: 0.8483 - auc: 0.5759 - val_los
s: 0.4154 - val_accuracy: 0.8486 - val_auc: 0.6212

Epoch 00002: val_auc improved from 0.58653 to 0.62123, saving model to model1.hdf5
Epoch 3/20

Epoch 00003: LearningRateScheduler setting learning rate to 0.0009000000427477062.
1039/1039 [==============================] - 64s 62ms/step - loss: 0.4060 - accuracy: 0.8485 - auc: 0.6666 - val_los
s: 0.3865 - val_accuracy: 0.8486 - val_auc: 0.7145

Epoch 00003: val_auc improved from 0.62123 to 0.71449, saving model to model1.hdf5
Epoch 4/20

Epoch 00004: LearningRateScheduler setting learning rate to 0.0009000000427477062.
1039/1039 [==============================] - 63s 61ms/step - loss: 0.3860 - accuracy: 0.8485 - auc: 0.7224 - val_los
s: 0.3842 - val_accuracy: 0.8486 - val_auc: 0.7322

Epoch 00004: val_auc improved from 0.71449 to 0.73223, saving model to model1.hdf5
Epoch 5/20

Epoch 00005: LearningRateScheduler setting learning rate to 0.0009000000427477062.
1039/1039 [==============================] - 63s 60ms/step - loss: 0.3774 - accuracy: 0.8484 - auc: 0.7406 - val_los
s: 0.3840 - val_accuracy: 0.8486 - val_auc: 0.7389

Epoch 00005: val_auc improved from 0.73223 to 0.73894, saving model to model1.hdf5
Epoch 6/20

Epoch 00006: LearningRateScheduler setting learning rate to 0.0008100000384729355.
1039/1039 [==============================] - 62s 59ms/step - loss: 0.3701 - accuracy: 0.8489 - auc: 0.7568 - val_los
s: 0.3708 - val_accuracy: 0.8486 - val_auc: 0.7480

Epoch 00006: val_auc improved from 0.73894 to 0.74800, saving model to model1.hdf5
Epoch 7/20

Epoch 00007: LearningRateScheduler setting learning rate to 0.0008100000559352338.
1039/1039 [==============================] - 64s 62ms/step - loss: 0.3611 - accuracy: 0.8513 - auc: 0.7696 - val_los
s: 0.3724 - val_accuracy: 0.8487 - val_auc: 0.7438

Epoch 00007: val_auc did not improve from 0.74800
```

```
Epoch 8/20

Epoch 00008: LearningRateScheduler setting learning rate to 0.0008100000559352338.
1039/1039 [==============================] - 62s 60ms/step - loss: 0.3550 - accuracy: 0.8555 - auc: 0.7766 - val_los
s: 0.3677 - val_accuracy: 0.8517 - val_auc: 0.7512

Epoch 00008: val_auc improved from 0.74800 to 0.75121, saving model to model1.hdf5
Epoch 9/20

Epoch 00009: LearningRateScheduler setting learning rate to 0.0008100000559352338.
1039/1039 [==============================] - 54s 52ms/step - loss: 0.3491 - accuracy: 0.8586 - auc: 0.7872 - val_los
s: 0.3762 - val_accuracy: 0.8547 - val_auc: 0.7528

Epoch 00009: val_auc improved from 0.75121 to 0.75279, saving model to model1.hdf5
Epoch 10/20

Epoch 00010: LearningRateScheduler setting learning rate to 0.0008100000559352338.
1039/1039 [==============================] - 58s 56ms/step - loss: 0.3442 - accuracy: 0.8612 - auc: 0.7961 - val_los
s: 0.3671 - val_accuracy: 0.8534 - val_auc: 0.7501

Epoch 00010: val_auc did not improve from 0.75279
Epoch 11/20

Epoch 00011: LearningRateScheduler setting learning rate to 0.0007290000503417104.
1039/1039 [==============================] - 59s 57ms/step - loss: 0.3381 - accuracy: 0.8642 - auc: 0.8048 - val_los
s: 0.3745 - val_accuracy: 0.8445 - val_auc: 0.7491

Epoch 00011: val_auc did not improve from 0.75279
```

Out[ ]:
```
<keras.callbacks.History at 0x7f7f7a355950>
```

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

In [ ]:
```
%tensorboard --logdir logs1/fit/
```

In [ ]:
```
from sklearn.metrics import auc, roc_curve
y_test_pred = model1.predict([essay_padded_test,np.array(ohe_school_state_test),np.array(ohe_project_grade_test),padd
```

```
test_fpr, test_tpr, thres = roc_curve(y_test, y_test_pred)
auc(test_fpr, test_tpr)
```

Out[ ]:
```
0.7565094813271557
```

In the initial epochs, the model performed well and gave higher AUC scores for validation data compared to train data. But on later epochs, model slowly started overfitting. Compared to model2, model1 converged faster for higher AUC

In [ ]:
```

```