In [0]:

```
%tensorflow_version 2.x
%load_ext tensorboard
import tensorflow as tf
```

```
TensorFlow 2.x selected.
```

In [0]:

```
import numpy as np
import pandas as pd
```

In [0]:

```
data = pd.read_csv('/content/drive/My Drive/preprocessed_data.csv')  # loading the
```

In [0]:

```
data.head(2) # displaying only the top 5 values of the dataset
```

Out[4]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_pr
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	

In [0]:

```
data['project_is_approved'].value_counts()
```

Out[5]:

```
1    92706
0    16542
Name: project_is_approved, dtype: int64
```

In [0]:

```
x = data.drop(['project_is_approved'],axis=1) # storing all the featuers except the
y = data['project_is_approved'] # storing the class label in the variable y
```

In [0]:

```
from sklearn.model_selection import train_test_split # importing the train test spl
```

In [0]:

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.40,stratify=y) # s
```

In [0]:

```
x_test,x_cv,y_test,y_cv = train_test_split(x_test,y_test,test_size=0.50,stratify=y_
```

In [0]:

```
y_train = tf.keras.utils.to_categorical(y_train,2)  # converting the train label in
y_test = tf.keras.utils.to_categorical(y_test,2) # converting the test label into c
y_cv = tf.keras.utils.to_categorical(y_cv,2) # converting the cv label into categor
```

In [0]:

```
from tensorflow.keras.preprocessing.text import Tokenizer # importing the tokenizer
```

In [0]:

```
embedded_index = dict() # defining the dictionary which is used to store the unique
f = open('/content/drive/My Drive/glove.6B.300d.txt') # opening the file in the rea
for line in f: # for each line in the file
    line1 = line.split()
    word = line1[0] # storing the word in the place of key
    word_value = line1[1:] # storing the corresponding word vectors in the plave of
    embedded_index[word] = word_value
    # embedded_index[line1[0]] = line1[1:]
f.close() # closing the file which was opened previously
```

In [0]:

```
# function which returns the word vectors of the words present in the sentence or t
def embedded_matrix_values(t,index_value,embedded_index):
    embedded_matrix = np.zeros((index_value,300)) # initilizing the matrix with zer
    for word,index in t.word_index.items(): # for each word in the sentence
        value = embedded_index.get(word) # get the word vector of the perticular wo
        if value is not None: # if the word is present in the word vector matrix
            embedded_matrix[index] = value # get that word vector and store it in t
    return embedded_matrix
```

In [0]:

```
t_essay = Tokenizer() # initilizing the tokenizer
t_essay.fit_on_texts(x_train['essay']) # spliting the sentence in to individual wor
essay_size = len(t_essay.word_index)+1 # calculating the length of the unique words
```

In [0]:

```
essay_embedded_matrix = embedded_matrix_values(t_essay,essay_size,embedded_index) #
```

In [0]:

```
t_school_state = Tokenizer()
t_school_state.fit_on_texts(x_train['school_state'])
school_state_size = len(t_school_state.word_index)+1
school_state_embedded_matrix = embedded_matrix_values(t_school_state,school_state_s
```

In [0]:

```
t_project_grade_category = Tokenizer()
t_project_grade_category.fit_on_texts(x_train['project_grade_category'])
project_grade_category_size = len(t_project_grade_category.word_index)+1
project_grade_category_embedded_matrix = embedded_matrix_values(t_project_grade_cat
```

In [0]:

```
t_clean_categories = Tokenizer()
t_clean_categories.fit_on_texts(x_train['clean_categories'])
clean_categories_size = len(t_clean_categories.word_index)+1
clean_categories_embedded_matrix = embedded_matrix_values(t_clean_categories,clean_
```

In [0]:

```
t_clean_subcategories = Tokenizer()
t_clean_subcategories.fit_on_texts(x_train['clean_subcategories'])
clean_subcategories_size = len(t_clean_subcategories.word_index)+1
clean_subcategories_embedded_matrix = embedded_matrix_values(t_clean_subcategories,
```

In [0]:

```
t_teacher_prefix = Tokenizer()
t_teacher_prefix.fit_on_texts(x_train['teacher_prefix'])
teacher_prefix_size = len(t_teacher_prefix.word_index)+1
teacher_prefix_embedded_matrix = embedded_matrix_values(t_teacher_prefix,teacher_pr
```

In [0]:

```
from tensorflow.keras.layers import Dense,Input,Embedding,Flatten,LSTM,Dropout
```

In [0]:

```
from tensorflow.keras.layers import concatenate
```

In [0]:

```
from tensorflow.keras.models import Model
```

In [0]:

```
from sklearn.preprocessing import StandardScaler # used to normalize (mean centerin
```

In [0]:

```
remaining_data = x_train[['price','teacher_number_of_previously_posted_projects']]
standardised_rem_data = StandardScaler().fit_transform(remaining_data) # normalizin
```

In [0]:

```
remaining_data_cv = x_cv[['price','teacher_number_of_previously_posted_projects']]
standardised_rem_data_cv = StandardScaler().fit_transform(remaining_data_cv)
```

In [0]:

```
remaining_data_test = x_test[['price','teacher_number_of_previously_posted_projects
standardised_rem_data_test = StandardScaler().fit_transform(remaining_data_test)
```

In [0]:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences  # as we know tha
# rows of the word vector matrix will not be same so we need to pad the zeers so th
```

In [0]:

```
encoded_essay = t_essay.texts_to_sequences(x_train['essay']) # converting the words
max_length = 100 # considering the maximum length of each row as 100
padded_essay = pad_sequences(encoded_essay,maxlen=max_length,padding='post') # limi
```

In [0]:

```
encoded_essay_cv = t_essay.texts_to_sequences(x_cv['essay'])
padded_essay_cv = pad_sequences(encoded_essay_cv,maxlen=max_length,padding='post')
```

In [0]:

```
encoded_essay_test = t_essay.texts_to_sequences(x_test['essay'])
padded_essay_test = pad_sequences(encoded_essay_test,maxlen=max_length,padding='pos
```

In [0]:

```
encoded_school_state = t_school_state.texts_to_sequences(x_train['school_state'])
padded_school_state = pad_sequences(encoded_school_state,maxlen=max_length,padding=
```

In [0]:

```
encoded_school_state_cv = t_school_state.texts_to_sequences(x_cv['school_state'])
padded_school_state_cv = pad_sequences(encoded_school_state_cv,maxlen=max_length,pa
```

In [0]:

```
encoded_school_state_test = t_school_state.texts_to_sequences(x_test['school_state'
padded_school_state_test = pad_sequences(encoded_school_state_test,maxlen=max_lengt
```

In [0]:

```
encoded_project_grade_category = t_project_grade_category.texts_to_sequences(x_trai
padded_project_grade_category = pad_sequences(encoded_project_grade_category,maxlen
```

In [0]:

```
encoded_project_grade_category_cv = t_project_grade_category.texts_to_sequences(x_c
padded_project_grade_category_cv = pad_sequences(encoded_project_grade_category_cv,
```

In [0]:

```
encoded_project_grade_category_test = t_project_grade_category.texts_to_sequences(x
padded_project_grade_category_test = pad_sequences(encoded_project_grade_category_t
```

In [0]:

```
encoded_clean_categories = t_clean_categories.texts_to_sequences(x_train['clean_cat
padded_clean_categories = pad_sequences(encoded_clean_categories,maxlen=max_length,
```

In [0]:

```
encoded_clean_categories_cv = t_clean_categories.texts_to_sequences(x_cv['clean_cat
padded_clean_categories_cv = pad_sequences(encoded_clean_categories_cv,maxlen=max_l
```

In [0]:

```
encoded_clean_categories_test = t_clean_categories.texts_to_sequences(x_test['clean
padded_clean_categories_test = pad_sequences(encoded_clean_categories_test,maxlen=m
```

In [0]:

```
encoded_clean_subcategories = t_clean_subcategories.texts_to_sequences(x_train['cle
padded_clean_subcategories = pad_sequences(encoded_clean_subcategories,maxlen=max_l
```

In [0]:

```
encoded_clean_subcategories_cv = t_clean_subcategories.texts_to_sequences(x_cv['cle
padded_clean_subcategories_cv = pad_sequences(encoded_clean_subcategories_cv,maxlen
```

In [0]:

```
encoded_clean_subcategories_test = t_clean_subcategories.texts_to_sequences(x_test[
padded_clean_subcategories_test = pad_sequences(encoded_clean_subcategories_test,ma
```

In [0]:

```
padded_clean_subcategories_test.shape
```

Out[44]:

```
(21850, 100)
```

In [0]:

```
encoded_teacher_prefix = t_teacher_prefix.texts_to_sequences(x_train['teacher_prefi
padded_teacher_prefix = pad_sequences(encoded_teacher_prefix,maxlen=max_length,padd
```

In [0]:

```
encoded_teacher_prefix_cv = t_teacher_prefix.texts_to_sequences(x_cv['teacher_prefi
padded_teacher_prefix_cv = pad_sequences(encoded_teacher_prefix_cv,maxlen=max_lengt
```

In [0]:

```
encoded_teacher_prefix_test = t_teacher_prefix.texts_to_sequences(x_test['teacher_p
padded_teacher_prefix_test = pad_sequences(encoded_teacher_prefix_test,maxlen=max_l
```

In [0]:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

In [0]:

```
class EndOfEpoch(tf.keras.callbacks.Callback):
    def on_epoch_end(self,epoch,logs={}): # at the end of each epoch this function
        y_pred = self.model.predict([padded_essay_test,padded_school_state_test,pad
        # the above line will predict the class label of the x_test data points
        print('   AUC score at the end of the epoch is:-',roc_auc_score(y_test,y_pr
        # the above line will print the AUC of x_test data points
```

In [0]:

```
from tensorflow.keras.callbacks import ModelCheckpoint # used to store the best mod
```

In [0]:

```
import datetime
```

In [0]:

```
!rm -rf ./logs # removes the folder logs
```

In [0]:

```
from tensorflow.keras import backend
```

In [0]:

```
backend.clear_session()
# defining a model with multiple inputs and single ouput
# we are not training the word vectors we are just uisng the pre trained weights
input_essay = Input(shape=(max_length,)) # input one
embedding_essay = Embedding(essay_size,300,input_length=100,weights=[essay_embedded
# the above embedding layer is having the weights as word vectors of the essays
lstm_essay = LSTM(32)(embedding_essay)
flatten_essay = Flatten()(lstm_essay)
input_school_state = Input(shape=(max_length,)) # input two
embedding_school_state = Embedding(school_state_size,300,input_length=100,weights=[
# the above embedding layer is having the weights as word vectors of the  school st
flatten_school_state = Flatten()(embedding_school_state)
input_project_grade_category = Input(shape=(max_length,)) # input three
embedding_project_grade_category = Embedding(project_grade_category_size,300,input_
# the above embedding layer is having the weights as word vectors of the project gr
flatten_project_grade_category = Flatten()(embedding_project_grade_category)
input_clean_categories = Input(shape=(max_length,)) # input four
embedding_clean_categories = Embedding(clean_categories_size,300,input_length=100,w
# the above embedding layer is having the weights as word vectors of the clean cate
flatten_clean_categories = Flatten()(embedding_clean_categories)
input_clean_subcategories = Input(shape=(max_length,)) # input five
embedding_clean_subcategories = Embedding(clean_subcategories_size,300,input_length
# the above embedding layer is having the weights as word vectors of the clean subc
flatten_clean_subcategories = Flatten()(embedding_clean_subcategories)
input_teacher_prefix = Input(shape=(max_length,)) # input six
embedding_teacher_prefix = Embedding(teacher_prefix_size,300,input_length=100,weigh
# the above embedding layer is having the weights as word vectors of the teacher pr
flatten_teacher_prefix = Flatten()(embedding_teacher_prefix)
input2_layer = Input(shape=(standardised_rem_data.shape[1],)) # input seven
remaining_dense_data = Dense(100,activation='relu',kernel_initializer=tf.keras.init
concat = concatenate([flatten_essay,flatten_school_state,flatten_project_grade_cate
dense_after_concat = Dense(80,activation='relu',kernel_initializer=tf.keras.initial
droupout_layer1 = Dropout(0.2)(dense_after_concat)
dense_after_dropout1 = Dense(60,activation='relu',kernel_initializer=tf.keras.initi
dropout_layer2 = Dropout(0.2)(dense_after_dropout1)
dense_after_dropout2 = Dense(40,activation='relu',kernel_initializer=tf.keras.initi
output_layer = Dense(2,activation='softmax',kernel_initializer=tf.keras.initializer
# defining all the seven inputs and the output of the model
model = Model(inputs=[input_essay,input_school_state,input_project_grade_category,i
# filepath to save the best model
filepath = '/content/drive/My Drive/savedmodels/weights-{epoch:02d}-{val_acc:04f}.h
checkpoint = ModelCheckpoint(filepath=filepath,monitor='val_loss',verbose=1,save_be
# object of the newly created class which executes the end of the epoch function
eoe = EndOfEpoch()
# logs for graph generation in tensorboard
log_dir = "logs/fit/"+datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
# callback used to initilize the tensorboard
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_fre
model.compile(optimizer=tf.keras.optimizers.Adam(),loss='categorical_crossentropy',
model.fit([padded_essay,padded_school_state,padded_project_grade_category,padded_cl
```

```
Train on 65548 samples, validate on 21850 samples
Epoch 1/10
65400/65548 [============================>.] - ETA: 0s - loss: 0.429
3 - acc: 0.8474   AUC score at the end of the epoch is:- 0.615434814
5376898
```

```
Epoch 00001: val_loss improved from inf to 0.41553, saving model to
/content/drive/My Drive/savedmodels/weights-01-0.848604.hdf5
65548/65548 [==============================] - 25s 378us/sample - lo
ss: 0.4292 - acc: 0.8474 - val_loss: 0.4155 - val_acc: 0.8486
Epoch 2/10
65500/65548 [============================>.] - ETA: 0s - loss: 0.400
7 - acc: 0.8486   AUC score at the end of the epoch is:- 0.725658607
1146076

Epoch 00002: val_loss improved from 0.41553 to 0.39954, saving model
to /content/drive/My Drive/savedmodels/weights-02-0.848604.hdf5
65548/65548 [==============================] - 23s 348us/sample - lo
ss: 0.4007 - acc: 0.8486 - val_loss: 0.3995 - val_acc: 0.8486
Epoch 3/10
65500/65548 [============================>.] - ETA: 0s - loss: 0.380
8 - acc: 0.8485   AUC score at the end of the epoch is:- 0.736465066
9481627

Epoch 00003: val_loss improved from 0.39954 to 0.37909, saving model
to /content/drive/My Drive/savedmodels/weights-03-0.848604.hdf5
65548/65548 [==============================] - 23s 353us/sample - lo
ss: 0.3807 - acc: 0.8486 - val_loss: 0.3791 - val_acc: 0.8486
Epoch 4/10
65500/65548 [============================>.] - ETA: 0s - loss: 0.373
7 - acc: 0.8486   AUC score at the end of the epoch is:- 0.742261129
3498034

Epoch 00004: val_loss improved from 0.37909 to 0.37834, saving model
to /content/drive/My Drive/savedmodels/weights-04-0.848604.hdf5
65548/65548 [==============================] - 22s 337us/sample - lo
ss: 0.3737 - acc: 0.8486 - val_loss: 0.3783 - val_acc: 0.8486
Epoch 5/10
65500/65548 [============================>.] - ETA: 0s - loss: 0.369
3 - acc: 0.8486   AUC score at the end of the epoch is:- 0.745572556
6931464

Epoch 00005: val_loss did not improve from 0.37834
65548/65548 [==============================] - 21s 326us/sample - lo
ss: 0.3694 - acc: 0.8486 - val_loss: 0.3852 - val_acc: 0.8486
Epoch 6/10
65500/65548 [============================>.] - ETA: 0s - loss: 0.363
9 - acc: 0.8486   AUC score at the end of the epoch is:- 0.749006440
9947756

Epoch 00006: val_loss did not improve from 0.37834
65548/65548 [==============================] - 21s 327us/sample - lo
ss: 0.3640 - acc: 0.8486 - val_loss: 0.3895 - val_acc: 0.8486
Epoch 7/10
65500/65548 [============================>.] - ETA: 0s - loss: 0.360
3 - acc: 0.8525   AUC score at the end of the epoch is:- 0.748463395
8092011

Epoch 00007: val_loss did not improve from 0.37834
65548/65548 [==============================] - 21s 323us/sample - lo
ss: 0.3604 - acc: 0.8525 - val_loss: 0.3931 - val_acc: 0.8438
Epoch 8/10
65500/65548 [============================>.] - ETA: 0s - loss: 0.354
8 - acc: 0.8558   AUC score at the end of the epoch is:- 0.749168835
4163974

Epoch 00008: val_loss did not improve from 0.37834
```

```
65548/65548 [==============================] - 21s 324us/sample - lo
ss: 0.3549 - acc: 0.8558 - val_loss: 0.3874 - val_acc: 0.8514
Epoch 9/10
65500/65548 [============================>.] - ETA: 0s - loss: 0.350
9 - acc: 0.8581   AUC score at the end of the epoch is:- 0.748696513
0768237

Epoch 00009: val_loss improved from 0.37834 to 0.37592, saving model
to /content/drive/My Drive/savedmodels/weights-09-0.847551.hdf5
65548/65548 [==============================] - 22s 332us/sample - lo
ss: 0.3510 - acc: 0.8581 - val_loss: 0.3759 - val_acc: 0.8476
Epoch 10/10
65500/65548 [============================>.] - ETA: 0s - loss: 0.344
5 - acc: 0.8619   AUC score at the end of the epoch is:- 0.748203299
0553276

Epoch 00010: val_loss did not improve from 0.37592
65548/65548 [==============================] - 22s 330us/sample - lo
ss: 0.3444 - acc: 0.8619 - val_loss: 0.3779 - val_acc: 0.8450
```

Out[57]:

```
<tensorflow.python.keras.callbacks.History at 0x7f87b20aa518>
```

In [0]:

```
%tensorboard --logdir logs/fit  # command used to initilize the tensorboard
```

```
<IPython.core.display.Javascript object>
```

Model - 2

In [0]:

```
essay = data['essay']
```

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer # tfiddf vectorizer use
```

In [0]:

```
vectorizer = TfidfVectorizer() # initilizing the tfidf vectorizer
tfidf = vectorizer.fit_transform(essay)

vectorizer.idf_ # getting the idf values of the unique words present in the column
```

Out[83]:

```
array([ 7.18528456,  5.91178569, 11.90823778, ..., 11.50277267,
       11.50277267, 11.90823778])
```

In [0]:

```
vectorizer.idf_.shape
```

Out[84]:

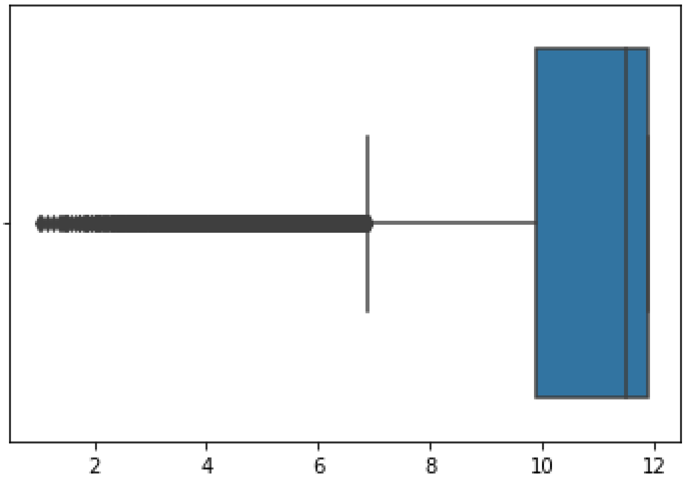
```
(56345,)
```

In [0]:

```
import seaborn as sb
import matplotlib.pyplot as plt
```

In [0]:

```
sb.boxplot(vectorizer.idf_)
plt.show()
```



from the above box plot it is clear that most of the words lies between 9 and 12 idf values

In [0]:

```
np.percentile(vectorizer.idf_,75)
```

Out[87]:

```
11.908237779037922
```

In [0]:

```
np.percentile(vectorizer.idf_,25)
```

Out[88]:

```
9.893334758495657
```

In [0]:

```
len(vectorizer.get_feature_names())
```

Out[89]:

```
56345
```

In [0]:

```
low_percentile = np.percentile(vectorizer.idf_,25) # taking the 25 precentile value
high_percentile = np.percentile(vectorizer.idf_,75) # taking teh 75 percentile valu
feature_names = vectorizer.get_feature_names() # all the unique words in the essay
```

In [0]:

```
# vectorizer.transform(x_train['essay'])
# feature_names = vectorizer.get_feature_names()
# vectorizer.idf_.shape
```

In [0]:

```
idf_values = vectorizer.idf_ # considering the idf values
not_important_features = []
for i in range(len(feature_names)):
    if idf_values[i]<=low_percentile or idf_values[i]>=high_percentile:  # ignoring
        not_important_features.append(feature_names[i])
        # del idf_values[i]
# new_feature_names = np.delete(feature_names,remove_index)
```

In [0]:

```
new_train_essay = []
for i in x_train['essay'].values: # for each row in the train essay column
    dummy_list = []
    for j in i.split(): # spliting the sentence into words
        if j not in not_important_features: # if a word is not in the not_important
            dummy_list.append(j)
    new_train_essay.append(dummy_list)
```

In [0]:

```
new_cv_essay = []
for i in x_cv['essay'].values:
    dummy_essay = []
    for j in i.split():
        if j not in not_important_features:
            dummy_essay.append(j)
    new_cv_essay.append(dummy_essay)
```

In [0]:

```
new_test_essay = []
for i in x_test['essay'].values:
    dummy_essay = []
    for j in i.split():
        if j not in not_important_features:
            dummy_essay.append(j)
    new_test_essay.append(dummy_essay)
```

In [0]:

```
# t_essay_model_2 = Tokenizer()
t_essay.fit_on_texts(new_train_essay) # fitting the new essay
new_essay_size = len(t_essay.word_index)+1
new_essay_embedded_matrix = embedded_matrix_values(t_essay,new_essay_size,embedded_
```

In [0]:

```
encoded_essay_model_2 = t_essay.texts_to_sequences(new_train_essay)
padded_essay_model_2 = pad_sequences(encoded_essay_model_2,maxlen=max_length,paddin
```

In [0]:

```
encoded_essay_model_2_cv = t_essay.texts_to_sequences(new_cv_essay)
padded_essay_model_2_cv = pad_sequences(encoded_essay_model_2_cv,maxlen=max_length,
```

In [0]:

```
encoded_essay_model_2_test = t_essay.texts_to_sequences(new_test_essay)
padded_essay_model_2_test = pad_sequences(encoded_essay_model_2_test,maxlen=max_len
```

In [0]:

```
class EndOfEpoch_model2(tf.keras.callbacks.Callback):
    def on_epoch_end(self,epoch,logs={}):
        # predicting the class label
        y_pred = self.model.predict([padded_essay_model_2_test,padded_school_state_
        # printing the AUC of the test data
        print('   AUC score at the end of the epoch is:-',roc_auc_score(y_test,y_pr
```

In [0]:

```
%rm -rf ./logs/fit_model_2
```

In [0]:

```
backend.clear_session()

input_essay = Input(shape=(max_length,))
embedding_essay = Embedding(new_essay_size,300,input_length=100,weights=[new_essay_
lstm_essay = LSTM(32)(embedding_essay)
flatten_essay = Flatten()(lstm_essay)
input_school_state = Input(shape=(max_length,))
embedding_school_state = Embedding(school_state_size,300,input_length=100,weights=[
flatten_school_state = Flatten()(embedding_school_state)
input_project_grade_category = Input(shape=(max_length,))
embedding_project_grade_category = Embedding(project_grade_category_size,300,input_
flatten_project_grade_category = Flatten()(embedding_project_grade_category)
input_clean_categories = Input(shape=(max_length,))
embedding_clean_categories = Embedding(clean_categories_size,300,input_length=100,w
flatten_clean_categories = Flatten()(embedding_clean_categories)
input_clean_subcategories = Input(shape=(max_length,))
embedding_clean_subcategories = Embedding(clean_subcategories_size,300,input_length
flatten_clean_subcategories = Flatten()(embedding_clean_subcategories)
input_teacher_prefix = Input(shape=(max_length,))
embedding_teacher_prefix = Embedding(teacher_prefix_size,300,input_length=100,weigh
flatten_teacher_prefix = Flatten()(embedding_teacher_prefix)
input2_layer = Input(shape=(standardised_rem_data.shape[1],))
remaining_dense_data = Dense(100,activation='relu',kernel_initializer=tf.keras.init
concat = concatenate([flatten_essay,flatten_school_state,flatten_project_grade_cate
dense_after_concat = Dense(80,activation='relu',kernel_initializer=tf.keras.initial
droupout_layer1 = Dropout(0.2)(dense_after_concat)
dense_after_dropout1 = Dense(60,activation='relu',kernel_initializer=tf.keras.initi
dropout_layer2 = Dropout(0.2)(dense_after_dropout1)
dense_after_dropout2 = Dense(40,activation='relu',kernel_initializer=tf.keras.initi
output_layer = Dense(2,activation='softmax',kernel_initializer=tf.keras.initializer
# giving all the inputs and output to the model
model = Model(inputs=[input_essay,input_school_state,input_project_grade_category,i
# filepath to store the best model weigths
filepath = '/content/drive/My Drive/savedmodels/weights-{epoch:02d}-{val_acc:04f}.h
checkpoint = ModelCheckpoint(filepath=filepath,monitor='val_loss',verbose=1,save_be
# creating the object of the class which invocks the end of each epoch
eoe = EndOfEpoch_model2()
# path to store the logs so that tensorboard can plot the graphs
log_dir = "logs/fit_model_2/"+datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
# tensorboard callback used to initilize the tensorboard
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_fre
model.compile(optimizer=tf.keras.optimizers.Adam(),loss='categorical_crossentropy',
model.fit([padded_essay_model_2,padded_school_state,padded_project_grade_category,p
```

```
Train on 65548 samples, validate on 21850 samples
Epoch 1/5
65400/65548 [============================>.] - ETA: 0s - loss: 0.433
3 - acc: 0.8467   AUC score at the end of the epoch is:- 0.606888600
7925816

Epoch 00001: val_loss improved from inf to 0.42547, saving model to
/content/drive/My Drive/savedmodels/weights-01-0.848604.hdf5
65548/65548 [==============================] - 27s 419us/sample - lo
ss: 0.4333 - acc: 0.8467 - val_loss: 0.4255 - val_acc: 0.8486
Epoch 2/5
65500/65548 [============================>.] - ETA: 0s - loss: 0.419
9 - acc: 0.8486   AUC score at the end of the epoch is:- 0.608360920
```

```
3775664

Epoch 00002: val_loss improved from 0.42547 to 0.41810, saving model
to /content/drive/My Drive/savedmodels/weights-02-0.848604.hdf5
65548/65548 [==============================] - 26s 392us/sample - lo
ss: 0.4199 - acc: 0.8486 - val_loss: 0.4181 - val_acc: 0.8486
Epoch 3/5
65500/65548 [============================>.] - ETA: 0s - loss: 0.417
6 - acc: 0.8486   AUC score at the end of the epoch is:- 0.606470164
0784695

Epoch 00003: val_loss did not improve from 0.41810
65548/65548 [==============================] - 22s 328us/sample - lo
ss: 0.4176 - acc: 0.8486 - val_loss: 0.4206 - val_acc: 0.8486
Epoch 4/5
65500/65548 [============================>.] - ETA: 0s - loss: 0.416
9 - acc: 0.8486   AUC score at the end of the epoch is:- 0.602655421
1962742

Epoch 00004: val_loss did not improve from 0.41810
65548/65548 [==============================] - 22s 331us/sample - lo
ss: 0.4170 - acc: 0.8486 - val_loss: 0.4212 - val_acc: 0.8486
Epoch 5/5
65500/65548 [============================>.] - ETA: 0s - loss: 0.416
5 - acc: 0.8486   AUC score at the end of the epoch is:- 0.598792485
2338962

Epoch 00005: val_loss did not improve from 0.41810
65548/65548 [==============================] - 22s 331us/sample - lo
ss: 0.4165 - acc: 0.8486 - val_loss: 0.4181 - val_acc: 0.8486
```

Out[102]:

```
<tensorflow.python.keras.callbacks.History at 0x7f85cf6dc128>
```

In [0]:

```
%tensorboard --logdir logs/fit_model_2
```

```
<IPython.core.display.Javascript object>
```

Model - 3

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

one hot encoding the categorical features

In [0]:

```
word_vectors = CountVectorizer(binary=True) # initilizing the count vectorizer
one_hot_school_state_train = word_vectors.fit_transform(x_train['school_state']) #
one_hot_school_state_cv = word_vectors.transform(x_cv['school_state']) # transformi
one_hot_school_state_test = word_vectors.transform(x_test['school_state']) # transf
```

In [0]:

```
one_hot_teacher_prefix_train = word_vectors.fit_transform(x_train['teacher_prefix']
one_hot_teacher_prefix_cv = word_vectors.transform(x_cv['teacher_prefix'])
one_hot_teacher_prefix_test = word_vectors.transform(x_test['teacher_prefix'])
```

In [0]:

```
one_hot_project_grade_category_train = word_vectors.fit_transform(x_train['project_
one_hot_project_grade_category_cv = word_vectors.transform(x_cv['project_grade_cate
one_hot_project_grade_category_test = word_vectors.transform(x_test['project_grade_
```

In [0]:

```
one_hot_clean_categories_train = word_vectors.fit_transform(x_train['clean_categori
one_hot_clean_categories_cv = word_vectors.transform(x_cv['clean_categories'])
one_hot_clean_categories_test = word_vectors.transform(x_test['clean_categories'])
```

In [0]:

```
one_hot_clean_subcategories_train = word_vectors.fit_transform(x_train['clean_subca
one_hot_clean_subcategories_cv = word_vectors.transform(x_cv['clean_subcategories']
one_hot_clean_subcategories_test = word_vectors.transform(x_test['clean_subcategori
```

In [0]:

```
# concatenating all the one hot encoded columns and the numerical columns except th
other_text_data_train = np.concatenate((one_hot_school_state_train.todense(),one_ho
```

In [0]:

```
other_text_data_cv = np.concatenate((one_hot_school_state_cv.todense(),one_hot_proj
other_text_data_test = np.concatenate((one_hot_school_state_test.todense(),one_hot_
```

In [0]:

```
# padding the data so that all the inputs will be of same shape
other_text_data_train_new = pad_sequences(other_text_data_train,maxlen=max_length,p
other_text_data_cv_new = pad_sequences(other_text_data_cv,maxlen=max_length,padding
other_text_data_test_new = pad_sequences(other_text_data_test,maxlen=max_length,pad
```

In [0]:

```
from tensorflow.keras.layers import Conv1D
```

In [0]:

```
class EndOfEpoch_model3(tf.keras.callbacks.Callback):
    def on_epoch_end(self,epoch,logs={}):
        # predicting the class label of the test data
        y_pred = self.model.predict([other_text_data_test_new,padded_essay_test])
        # printing the AUC value of test data
        print('   AUC score at the end of the epoch is:-',roc_auc_score(y_test,y_pr
```

In [0]:

```
%rm -rf ./logs/fit_model_3
```

In [0]:

```
backend.clear_session()

input_layer = Input(shape=(other_text_data_train_new.shape[1],other_text_data_train
conv_layer = Conv1D(250,3,strides=1,padding='same',activation='relu',kernel_initial
conv_layer1 = Conv1D(225,3,strides=1,padding='same',activation='relu',kernel_initia
conv_layer2 = Conv1D(200,3,strides=1,padding='same',activation='relu',kernel_initia
conv_layer3 = Conv1D(180,3,strides=1,padding='same',activation='relu',kernel_initia
conv_layer4 = Conv1D(150,3,strides=1,padding='same',activation='relu',kernel_initia
conv_layer5 = Conv1D(130,3,strides=1,padding='same',activation='relu',kernel_initia
conv_layer6 = Conv1D(120,3,strides=1,padding='same',activation='relu',kernel_initia

flatten_layer = Flatten()(conv_layer6)

input_essay = Input(shape=(max_length,))
embedding_essay = Embedding(essay_size,300,input_length=100,weights=[essay_embedded
lstm_essay = LSTM(32)(embedding_essay)

flatten_essay = Flatten()(lstm_essay)

concate_layer1 = concatenate([flatten_layer,flatten_essay])

dense_layer1 = Dense(100,activation='relu',kernel_initializer='he_normal')(concate_
droupout_layer1 = Dropout(0.3)(dense_layer1)
dense_layer2 = Dense(80,activation='relu',kernel_initializer='he_normal')(droupout_
droupout_layer2 = Dropout(0.3)(dense_layer2)

dense_layer3 = Dense(40,activation='relu',kernel_initializer='he_normal')(droupout_
output = Dense(2,activation='softmax',kernel_initializer='he_normal')(dense_layer3)
# giving the inputs and output to the model
model = Model(inputs=[input_layer,input_essay],outputs=output)
# filepath to save the best model weights
filepath = '/content/drive/My Drive/savedmodels/weights-{epoch:02d}-{val_acc:04f}.h
checkpoint = ModelCheckpoint(filepath=filepath,monitor='val_loss',verbose=1,save_be
# object of the class which calls the function endofepoch at the end of each epoch
eoe = EndOfEpoch_model3()
log_dir = "logs/fit_model_3/"+datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
# tensorboard callback used to initilize the tensorboard
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir,histogram_fre
model.compile(optimizer=tf.keras.optimizers.Adam(),loss='categorical_crossentropy',
model.fit([other_text_data_train_new,padded_essay],y_train,validation_data=([other_
```

```
Train on 65548 samples, validate on 21850 samples
Epoch 1/10
65300/65548 [============================>.] - ETA: 0s - loss: 0.451
3 - acc: 0.8449   AUC score at the end of the epoch is:- 0.578167187
5365971

Epoch 00001: val_loss improved from inf to 0.42016, saving model to
/content/drive/My Drive/savedmodels/weights-01-0.848604.hdf5
65548/65548 [==============================] - 30s 454us/sample - lo
ss: 0.4511 - acc: 0.8450 - val_loss: 0.4202 - val_acc: 0.8486
Epoch 2/10
65500/65548 [============================>.] - ETA: 0s - loss: 0.413
1 - acc: 0.8486   AUC score at the end of the epoch is:- 0.719766790
1521134

Epoch 00002: val_loss improved from 0.42016 to 0.39471, saving model
to /content/drive/My Drive/savedmodels/weights-02-0.848604.hdf5
```

```
65548/65548 [==============================] - 28s 426us/sample - lo
ss: 0.4133 - acc: 0.8486 - val_loss: 0.3947 - val_acc: 0.8486
Epoch 3/10
65400/65548 [============================>.] - ETA: 0s - loss: 0.384
5 - acc: 0.8486   AUC score at the end of the epoch is:- 0.737750331
2067093

Epoch 00003: val_loss improved from 0.39471 to 0.37895, saving model
to /content/drive/My Drive/savedmodels/weights-03-0.848604.hdf5
65548/65548 [==============================] - 29s 437us/sample - lo
ss: 0.3845 - acc: 0.8486 - val_loss: 0.3789 - val_acc: 0.8486
Epoch 4/10
65400/65548 [============================>.] - ETA: 0s - loss: 0.375
5 - acc: 0.8485   AUC score at the end of the epoch is:- 0.747793215
7541163

Epoch 00004: val_loss improved from 0.37895 to 0.37716, saving model
to /content/drive/My Drive/savedmodels/weights-04-0.848604.hdf5
65548/65548 [==============================] - 30s 452us/sample - lo
ss: 0.3754 - acc: 0.8486 - val_loss: 0.3772 - val_acc: 0.8486
Epoch 5/10
65500/65548 [============================>.] - ETA: 0s - loss: 0.369
8 - acc: 0.8485   AUC score at the end of the epoch is:- 0.748761058
4721136

Epoch 00005: val_loss did not improve from 0.37716
65548/65548 [==============================] - 25s 389us/sample - lo
ss: 0.3697 - acc: 0.8485 - val_loss: 0.3808 - val_acc: 0.8486
Epoch 6/10
65300/65548 [============================>.] - ETA: 0s - loss: 0.365
4 - acc: 0.8493   AUC score at the end of the epoch is:- 0.752644164
8705199

Epoch 00006: val_loss did not improve from 0.37716
65548/65548 [==============================] - 25s 385us/sample - lo
ss: 0.3656 - acc: 0.8492 - val_loss: 0.3790 - val_acc: 0.8486
Epoch 7/10
65500/65548 [============================>.] - ETA: 0s - loss: 0.359
2 - acc: 0.8525   AUC score at the end of the epoch is:- 0.754209533
3255455

Epoch 00007: val_loss did not improve from 0.37716
65548/65548 [==============================] - 26s 391us/sample - lo
ss: 0.3592 - acc: 0.8525 - val_loss: 0.3793 - val_acc: 0.8544
Epoch 8/10
65300/65548 [============================>.] - ETA: 0s - loss: 0.353
1 - acc: 0.8550   AUC score at the end of the epoch is:- 0.752782542
2113439

Epoch 00008: val_loss improved from 0.37716 to 0.37404, saving model
to /content/drive/My Drive/savedmodels/weights-08-0.849565.hdf5
65548/65548 [==============================] - 26s 392us/sample - lo
ss: 0.3530 - acc: 0.8550 - val_loss: 0.3740 - val_acc: 0.8496
Epoch 9/10
65400/65548 [============================>.] - ETA: 0s - loss: 0.347
2 - acc: 0.8583   AUC score at the end of the epoch is:- 0.751550996
5100011

Epoch 00009: val_loss improved from 0.37404 to 0.37174, saving model
to /content/drive/My Drive/savedmodels/weights-09-0.853959.hdf5
65548/65548 [==============================] - 26s 389us/sample - lo
```

```
ss: 0.3474 - acc: 0.8581 - val_loss: 0.3717 - val_acc: 0.8540
Epoch 10/10
65500/65548 [============================>.] - ETA: 0s - loss: 0.342
2 - acc: 0.8608   AUC score at the end of the epoch is:- 0.745695889
7410781

Epoch 00010: val_loss did not improve from 0.37174
65548/65548 [==============================] - 25s 384us/sample - lo
ss: 0.3422 - acc: 0.8608 - val_loss: 0.3833 - val_acc: 0.8514
```

Out[72]:

```
<tensorflow.python.keras.callbacks.History at 0x7f5daf290240>
```

In [0]:

```
%tensorboard --logdir logs/fit_model_3
```

```
<IPython.core.display.Javascript object>
```

In [0]:

```

```