In [0]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.model_selection import train_test_split
```

In [442]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [0]:
```
# Getting data into a dataframe
path="/content/drive/My Drive/Colab_Notebooks/ass14/preprocessed_data.csv"
df = pd.read_csv(path)
```

In [444]:
```
df.head(2)
```

Out[444]:

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

In [445]:
```
df.columns
```

Out[445]:
```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

In [446]:
```
df.shape
```

Out[446]:
```
(109248, 9)
```

In [0]:
```
#df = df.sample(n=40000)
#project_data=project_data.tail(1000)
#project_data.shape
```

Spliting dataset

In [448]:
```
y=df['project_is_approved']
y.shape
```

Out[448]:
```
(109248,)
```

In [449]:
```
features = df.drop(["project_is_approved"],axis=1)
features.shape
```

Out[449]:
```
(109248, 8)
```

In [0]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_

#split the data into train and test fo bag of words

x_train,x_test,y_train,y_test=model_selection.train_test_split(features,y,test_s
#split train into cross val train and cross val test
#x_train,x_cv,y_train,y_cv=model_selection.train_test_split(x_t,y_t,test_size=0.
```

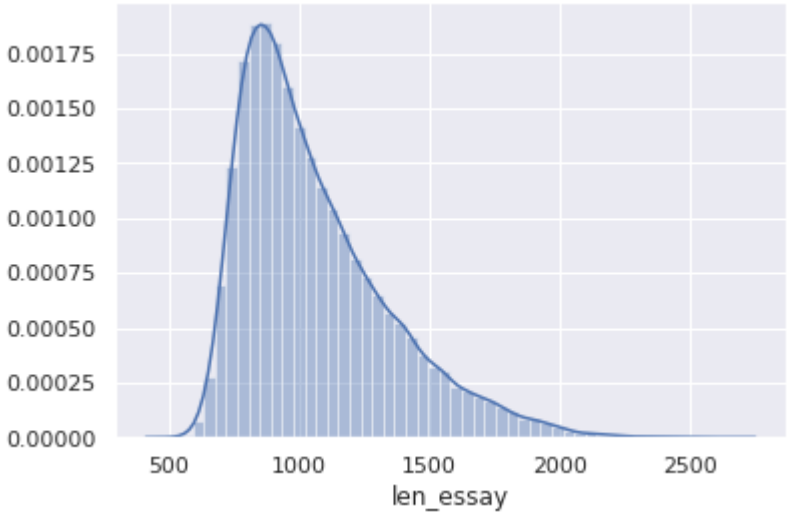
In [451]:
```
print(x_train.shape)
print("++++++++++++++")
print(x_test.shape)
```

```
(73196, 8)
++++++++++++++
(36052, 8)
```

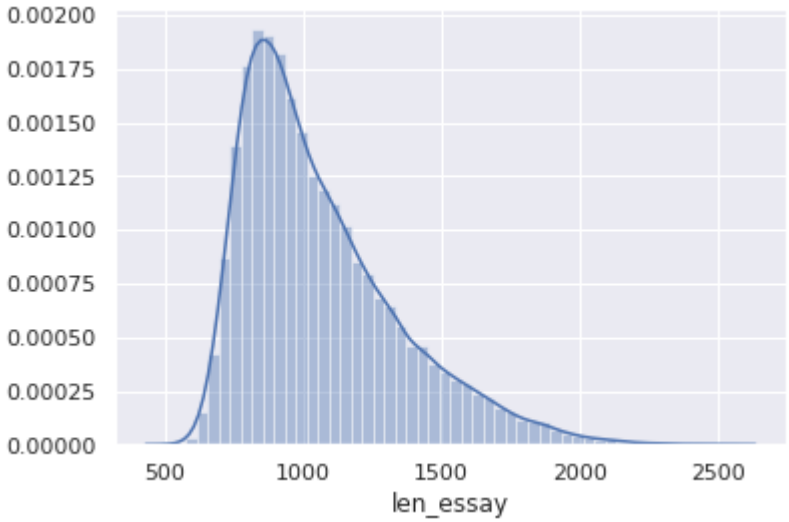
In [0]:
```
# Preparing Text Data As per Our Model
x_train["len_essay"] = x_train["essay"].apply(len)
x_test["len_essay"] = x_test["essay"].apply(len)
```

Distribution plot of essay dataset

In [453]:
```
sns.set()
ax = sns.distplot(x_train["len_essay"])
```



In [454]:
```
ax = sns.distplot(x_test["len_essay"])
```



In [0]:
```
from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem.porter import PorterStemmer
import re
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
```

Calculate IDF value

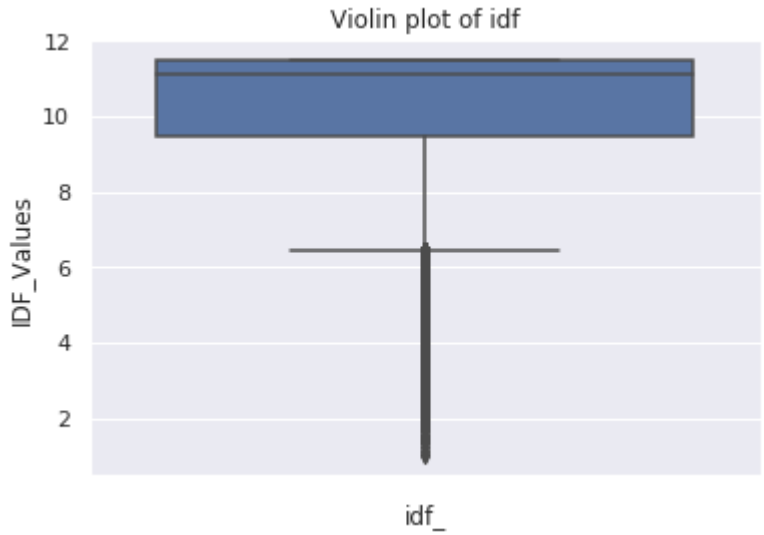
In [0]:
```
# Filtering Text Data based on idf values

tfidf = TfidfVectorizer()
combine_tfidf = tfidf.fit_transform(x_train["essay"])

# converting to dictionary
combine_dict = dict(zip(tfidf.get_feature_names(),list(tfidf.idf_)))
tfidf_df = pd.DataFrame(list(combine_dict.items()), columns=['Words', 'IDF_Value
tfidf_df = tfidf_df.sort_values(by ='IDF_Values' )
```

In [457]:
```
print(tfidf_df["IDF_Values"].min())
print(tfidf_df["IDF_Values"].max())
```

```
1.0080242390926728
11.50776253494305
```

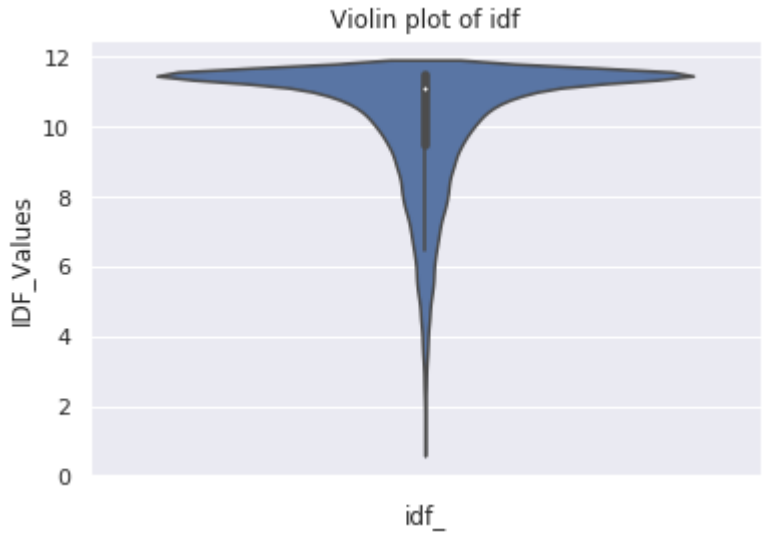
In [458]:
```
sns.boxplot(x = "IDF_Values",data=tfidf_df,orient="v")
plt.xlabel("idf_")
plt.title("Violin plot of idf")
```

Out[458]: Text(0.5, 1.0, 'Violin plot of idf')



In [459]:
```
sns.violinplot(x = "IDF_Values",data=tfidf_df,orient="v")
plt.xlabel("idf_")
plt.title("Violin plot of idf")
```

Out[459]: Text(0.5, 1.0, 'Violin plot of idf')



In [460]:
```
print("\nQuantiles:")
print(np.percentile(tfidf_df['IDF_Values'],np.arange(0, 100, 25)))
```

```
Quantiles:
[ 1.00802424  9.49285951 11.10229743 11.50776253]
```

In [461]:
```
print("\n25th Percentiles:")
print(np.percentile(tfidf_df['IDF_Values'],25))
print("\n75th Percentiles:")
print(np.percentile(tfidf_df['IDF_Values'],75))

print("\n90th Percentiles:")
print(np.percentile(tfidf_df['IDF_Values'],90))
```

```
25th Percentiles:
9.492859514400784

75th Percentiles:
11.50776253494305

90th Percentiles:
11.507762534943051
```

Consider words that have idf values between 25th and 75th percentile because most important and most rare words both are not good for model

In [462]:
```
print(tfidf_df.shape)
tfidf_filtered = tfidf_df[tfidf_df["IDF_Values"] <=  np.percentile(tfidf_df['IDF_
print("dimension after removing words", tfidf_filtered.shape)
```

```
(48463, 2)
dimension after removing words (12374, 2)
```

In [463]:
```
#selecting important words between 25th and 75th percentile
corpus = tfidf_filtered["Words"].tolist()
corpus[:10]
```

Out[463]:
```
['students',
 'nannan',
 'school',
 'my',
 'learning',
 'classroom',
 'not',
 'learn',
 'the',
 'they']
```

In [464]:
```
len(corpus)
```

Out[464]:
```
12374
```

In [0]:
```
# convert the sentences (strings) into integers
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(corpus)
sequences_train = tokenizer.texts_to_sequences(x_train["essay"])
sequences_test = tokenizer.texts_to_sequences(x_test["essay"])
```

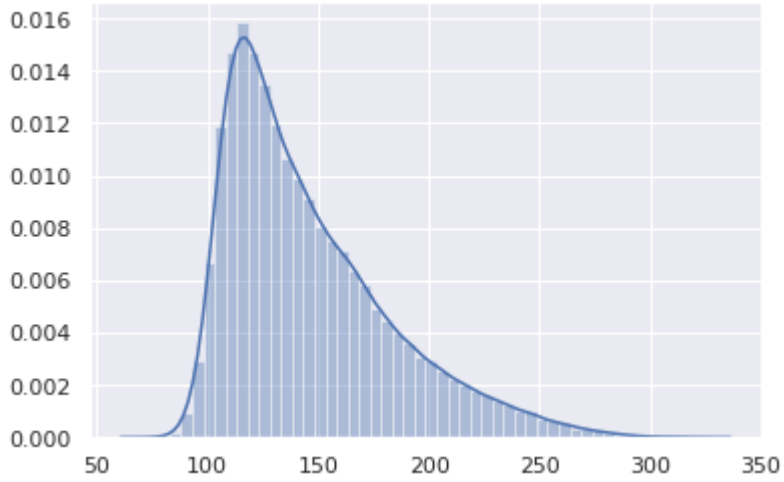
In [466]:
```
sequences_train[:2]
```

Out[466]:
```
[[24,
  6,
  1770,
  26,
  927,
  15,
  3063,
  6387,
  7132,
  12,
  1,
  30,
  741,
  1632,
  3,
  10,
  140,
  19,
  1259,
```

In [0]:
```
length = []
for i in sequences_train:
  length.append(len(i))
```

In [468]:
```
sns.set()
sns.distplot(length)
```

Out[468]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7efdb1c4ee48>
```



In [469]:
```
print("No. of datapoints in X_train :",len(x_train))
print("No. of datapoints in X_test :",len(x_test))
print("Shape of Y_train :",y_train.shape)
print("Shape of Y_test :",y_test.shape)
```

```
No. of datapoints in X_train : 73196
No. of datapoints in X_test : 36052
Shape of Y_train : (73196,)
Shape of Y_test : (36052,)
```

In [470]:
```
# get word -> integer mapping
word2idx = tokenizer.word_index
print('Found %s unique tokens.' % len(word2idx))
```

```
Found 12374 unique tokens.
```

In [0]:
```
# importing required libraries
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
from keras.layers import Input, Embedding, LSTM, Dropout, BatchNormalization, De
from keras.preprocessing.text import Tokenizer, one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Model, load_model
from keras import regularizers
from keras.optimizers import *
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard, ReduceLR
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics import roc_auc_score
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib inline
import re
from tqdm import tqdm
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
import pickle
```

In [472]:
```
# truncate and/or pad input sequences
max_review_length = 800
encoded_train = pad_sequences(sequences_train,maxlen=max_review_length,padding='
encoded_test = pad_sequences(sequences_test, maxlen=max_review_length,padding='p
print('Shape of train data tensor:', encoded_train.shape)
print('Shape of test data tensor:', encoded_test.shape)

print(encoded_train[1])
```

```
Shape of train data tensor: (73196, 800)
Shape of test data tensor: (36052, 800)
[   24     3  4340  2491  1674  2692  2784    24     3  4873  1405  4268
  1041  1463  2692    24     1    18   113   153  4537   828   874   143
   366   265   307    38   143   366    14   989   443  2692     7  2062
   257     3   989   340    24  1531    13  1480   874  2674    14    94
   874  2674  2643  1761    11    38  4537   168  4774  4705   724     3
   724   168  4461    24     3  4294  2491  1674  1565     3   989   181
   874  9576   443    24     1  5857    98  1257   221     3    73  5427
    10    21   168  4774  4705   724   228   656  5427  7632  1742   141
     1    15  3114   172   168    24     1  2785  9576    15    38   366
  4537   838   955    51    10  3699    13  2189  2571 10551  1480    29
   989   999   544    43     3     2     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     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     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     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    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]
```

In [0]:

```
# Loading Embedding File
pickle_in = open("glove_vectors","rb")
glove_words = pickle.load(pickle_in)
```

In [474]:

```
MAX_VOCAB_SIZE=5000
num_words = min(MAX_VOCAB_SIZE, len(word2idx) + 1)
embedding_matrix = np.zeros((num_words, 300))
for word, i in word2idx.items():
  if i < MAX_VOCAB_SIZE:
    embedding_vector = glove_words.get(word)
    if embedding_vector is not None:
      # words not found in embedding index will be all zeros.
      embedding_matrix[i] = embedding_vector


print(num_words)
print("+++++++++")
print(embedding_matrix.shape)
```

```
5000
+++++++++
(5000, 300)
```

In [0]:
```
# load pre-trained word embeddings into an Embedding layer
# note that we set trainable = False so as to keep the embeddings fixed
MAX_SEQUENCE_LENGTH=800
embedding_layer = Embedding(
  num_words,
  300,
  weights=[embedding_matrix],
  input_length=MAX_SEQUENCE_LENGTH,
  trainable=False
)
input_text = Input(shape=(MAX_SEQUENCE_LENGTH,),name="input_text")
x = embedding_layer(input_text)
x = LSTM(128,recurrent_dropout=0.5,kernel_regularizer=regularizers.l2(0.001),retu
# x = SpatialDropout1D(0.5)(x)
flatten_1 = Flatten()(x)
```

caregorical variable

In [0]:
```
vect = CountVectorizer(binary=True)
vect.fit(x_train["teacher_prefix"])

train_prefix = vect.transform(x_train["teacher_prefix"])
test_prefix = vect.transform(x_test["teacher_prefix"])
```

In [0]:
```
# School State
vect = CountVectorizer(binary=True)
vect.fit(x_train["school_state"])

train_state = vect.transform(x_train["school_state"])
test_state = vect.transform(x_test["school_state"])
```

In [0]:
```
# For project_grade_category
vect = CountVectorizer(binary=True)
vect.fit(x_train["project_grade_category"])

train_grade = vect.transform(x_train["project_grade_category"])
test_grade = vect.transform(x_test["project_grade_category"])
```

In [0]:
```
# For clean_categories
vect = CountVectorizer(binary=True)
vect.fit(x_train["clean_categories"])

train_subcat = vect.transform(x_train["clean_categories"])
test_subcat = vect.transform(x_test["clean_categories"])
```

In [0]:
```
# For clean_subcategories
vect = CountVectorizer(binary=True)
vect.fit(x_train["clean_subcategories"])

train_subcat_1 = vect.transform(x_train["clean_subcategories"])
test_subcat_1 = vect.transform(x_test["clean_subcategories"])
```

numerical data

In [0]:
```
# Now we will prepare numerical features for our model
num_train_1=x_train['len_essay'].values.reshape(-1, 1)
num_train_2=x_train['price'].values.reshape(-1, 1)
num_train_3=x_train['teacher_number_of_previously_posted_projects'].values.resha

num_test_1=x_test['len_essay'].values.reshape(-1, 1)
num_test_2=x_test['price'].values.reshape(-1, 1)
num_test_3=x_test['teacher_number_of_previously_posted_projects'].values.reshape


num_train=np.concatenate((num_train_1,num_train_2,num_train_3),axis=1)

num_test=np.concatenate((num_test_1,num_test_2,num_test_3),axis=1)
```

In [0]:
```
from sklearn.preprocessing import StandardScaler
norm=StandardScaler()
norm_train=norm.fit_transform(num_train)
norm_test=norm.transform(num_test)
```

In [0]:
```
from scipy.sparse import hstack
cat_train = hstack([train_prefix,train_state,train_grade,train_subcat,train_subca
cat_test = hstack([test_prefix,test_state,test_grade,test_subcat,test_subcat_1])
```

In [0]:
```
x_train = np.hstack((cat_train,norm_train))
x_test = np.hstack((cat_test,norm_test))
```

In [485]:
```
x_train.shape
```

Out[485]:
```
(73196, 102)
```

In [0]:
```
expend_train = np.expand_dims(x_train,2)
expend_test = np.expand_dims(x_test,2)
```

In [487]:
```
expend_train.shape
```

Out[487]:
```
(73196, 102, 1)
```

In [488]:
```
print(expend_train.shape)
print("+++++++++++++++")
print(expend_test.shape)
```

```
(73196, 102, 1)
+++++++++++++++
(36052, 102, 1)
```

In [0]:
```
inp_conv =  Input(shape=(102, 1),name='features_all')
x1 = Conv1D(filters=128, kernel_size=3, activation='relu',kernel_initializer="he_
x1 = Conv1D(filters=128, kernel_size=3, activation='relu',kernel_initializer="he_
# x1 = Conv1D(filters=256, kernel_size=5, activation='relu',kernel_initializer="h
# x1 = Conv1D(filters=128, kernel_size=5, activation='relu',kernel_initializer="h
x2 = Flatten()(x1)
```

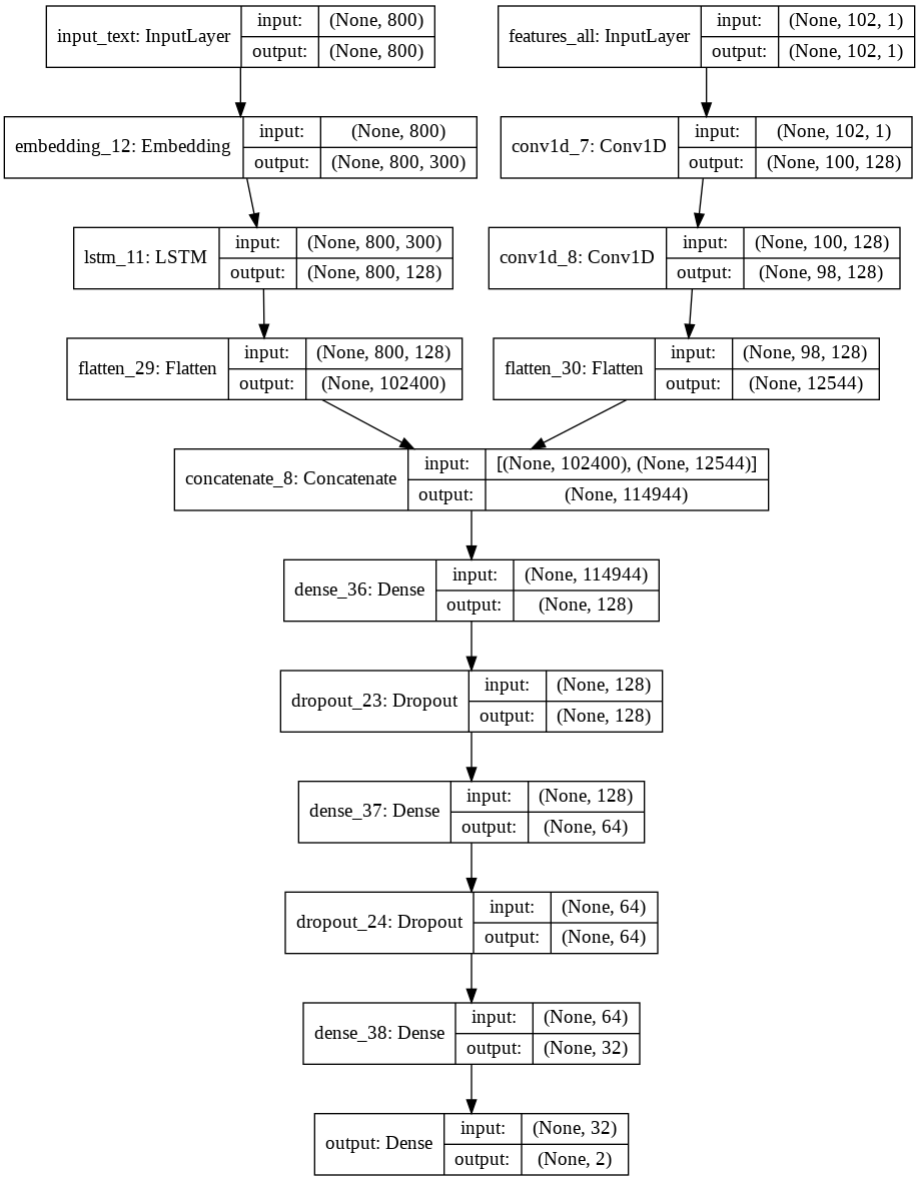
In [490]:
```
print("Building Model-3")
x_concatenate = concatenate([flatten_1,x2])
# x_concatenate = BatchNormalization()(x_concatenate)
x = Dense(128,activation="relu",kernel_initializer="he_normal",kernel_regularizer
# x=LeakyReLU(alpha=0.3)(x)
x=Dropout(0.5)(x)
x = Dense(64,activation="relu",kernel_initializer="he_normal",kernel_regularizer=
# x=LeakyReLU(alpha=0.3)(x)
x=Dropout(0.3)(x)
x = Dense(32,activation="relu",kernel_initializer="he_normal",kernel_regularizer=
# x = BatchNormalization()(x)
# x=LeakyReLU(alpha=0.3)(x)
output = Dense(2, activation='softmax', name='output')(x)
model_3 = Model(inputs=[input_text,inp_conv],outputs=[output])
```

```
Building Model-3
```

In [0]:
```
train_data_3 = [encoded_train,expend_train]
test_data_3 = [encoded_test,expend_test]

from keras.utils import np_utils
Y_train = np_utils.to_categorical(y_train, 2)
Y_test = np_utils.to_categorical(y_test, 2)
```

In [492]:
```
# https://github.com/mmortazavi/EntityEmbedding-Working_Example/blob/master/Entit
from keras.utils import plot_model
import pydot_ng as pydot
plot_model(model_3, show_shapes=True, show_layer_names=True, to_file='model_3.png
from IPython.display import Image
Image(retina=True, filename='model_3.png')
```

Out[492]:



In [0]:
```
checkpoint_3 = ModelCheckpoint("model_3.h5",
                               monitor="val_auroc",
                               mode="max",
                               save_best_only = True,
                               verbose=1)

# earlystop_3 = EarlyStopping(monitor = 'val_loss',
#                             mode="min",
#                             min_delta = 0,
#                             patience = 2,
#                             verbose = 1,
#                             restore_best_weights = True)

# reduce_lr_3 = ReduceLROnPlateau(monitor = 'val_auroc', factor = 0.2, patience =

tensorboard_3 = TensorBoard(log_dir='graph_3', histogram_freq=0, batch_size=512,

callbacks_3 = [tensorboard_3,checkpoint_3]
```

In [0]:
```
# Defining Custom ROC-AUC Metrics
from sklearn.metrics import roc_auc_score

def auc1(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 auroc(y_true, y_pred):
    return tf.py_func(auc1, (y_true, y_pred), tf.double)
```

In [0]:
```

adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=
rms = RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
```

In [0]:
```
model_3.compile(optimizer=adam, loss='categorical_crossentropy', metrics=[auroc]
```

In [497]:
```
history_3 = model_3.fit(train_data_3,Y_train,batch_size=512,epochs=30,validation_
```

```
Train on 73196 samples, validate on 36052 samples
Epoch 1/30
73196/73196 [==============================] - 327s 4ms/step - loss: 2.2392 -
auroc: 0.5115 - val_loss: 0.6345 - val_auroc: 0.6141

Epoch 00001: val_auroc improved from -inf to 0.61413, saving model to model_
3.h5
Epoch 2/30
73196/73196 [==============================] - 320s 4ms/step - loss: 0.4983 -
auroc: 0.6402 - val_loss: 0.4804 - val_auroc: 0.7195

Epoch 00002: val_auroc improved from 0.61413 to 0.71945, saving model to mode
l_3.h5
Epoch 3/30
73196/73196 [==============================] - 319s 4ms/step - loss: 0.4454 -
auroc: 0.7099 - val_loss: 0.4493 - val_auroc: 0.7365

Epoch 00003: val_auroc improved from 0.71945 to 0.73651, saving model to mode
l_3.h5
```

In [498]:
```
print(model_2.summary())
```

```
Model: "model_7"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_text (InputLayer)         (None, 800)          0
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 800, 300)     1500000     input_text[0][0]
__________________________________________________________________________________________________
teacher_prefix (InputLayer)     (None, 1)            0
__________________________________________________________________________________________________
school_prefix (InputLayer)      (None, 1)            0
__________________________________________________________________________________________________
grade_cat (InputLayer)          (None, 1)            0
__________________________________________________________________________________________________
sub_cat (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
sub_cat_1 (InputLayer)          (None, 1)            0
__________________________________________________________________________________________________
lstm_3 (LSTM)                   (None, 800, 128)     219648      embedding_3[0][0]
__________________________________________________________________________________________________
emb_pre (Embedding)             (None, 1, 3)         15          teacher_prefix[0][0]
__________________________________________________________________________________________________
emb_state (Embedding)           (None, 1, 26)        1326        school_prefix[0][0]
__________________________________________________________________________________________________
emb_grade (Embedding)           (None, 1, 2)         8           grade_cat[0][0]
__________________________________________________________________________________________________
emb_subcat (Embedding)          (None, 1, 26)        1326        sub_cat[0][0]
__________________________________________________________________________________________________
emb_subcat_1 (Embedding)        (None, 1, 50)        19550       sub_cat_1[0][0]
__________________________________________________________________________________________________
numerical_features (InputLayer) (None, 3)            0
__________________________________________________________________________________________________
flatten_13 (Flatten)            (None, 102400)       0           lstm_3[0][0]
```

flatten_14 (Flatten)	(None, 3)	0	emb_pre[0][0]
flatten_15 (Flatten)	(None, 26)	0	emb_state[0][0]
flatten_16 (Flatten)	(None, 2)	0	emb_grade[0][0]
flatten_17 (Flatten)	(None, 26)	0	emb_subcat[0][0]
flatten_18 (Flatten)	(None, 50)	0	emb_subcat_1[0][0]
dense_8 (Dense)	(None, 100)	400	numerical_features[0][0]
concatenate_3 (Concatenate)	(None, 102607)	0	flatten_13[0][0] flatten_14[0][0] flatten_15[0][0] flatten_16[0][0] flatten_17[0][0] flatten_18[0][0] dense_8[0][0]
dense_21 (Dense)	(None, 128)	13133824	concatenate_3[0][0]
dropout_13 (Dropout)	(None, 128)	0	dense_21[0][0]
dense_22 (Dense)	(None, 256)	33024	dropout_13[0][0]
dropout_14 (Dropout)	(None, 256)	0	dense_22[0][0]
dense_23 (Dense)	(None, 64)	16448	dropout_14[0][0]

```
____________________
batch_normalization_7 (BatchNor (None, 64)           256         dense_23[0][0]
_____________________________________________________________________________
____________________
output (Dense)                  (None, 2)            130         batch_normaliz
ation_7[0][0]
=============================================================================
==================
Total params: 14,925,955
Trainable params: 13,425,827
Non-trainable params: 1,500,128
_____________________________________________________________________________
____________________
None
```

In [0]:
```
my_model = load_model("model_2.h5",custom_objects={"auroc":auroc})
```

In [0]:
```
project_status = {0:"Rejected",1:"Approved"}
```

In [0]:
```
Y_pred = my_model.predict(test_data_1,batch_size=512)
```

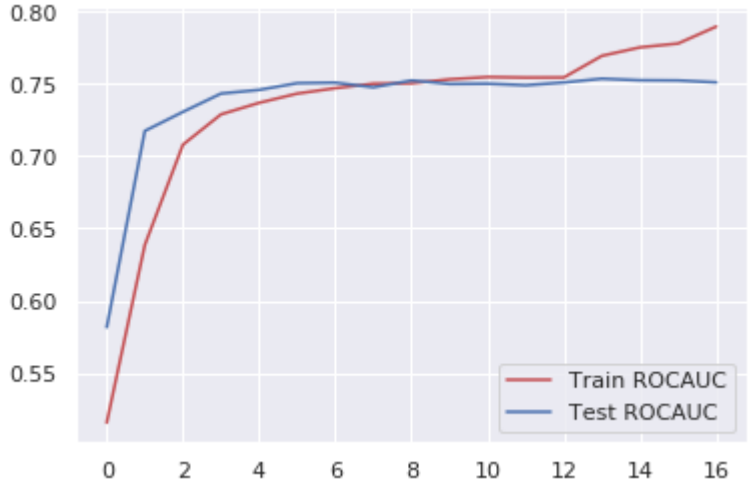
In [0]:
```
# took the function from https://nbviewer.jupyter.org/github/pranaya-mathur/Humar
def confusion_matrix(Y_true, Y_pred):
    Y_true = pd.Series([project_status[y] for y in np.argmax(Y_test, axis=1)])
    Y_pred = pd.Series([project_status[y] for y in np.argmax(Y_pred, axis=1)])

    return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])
```

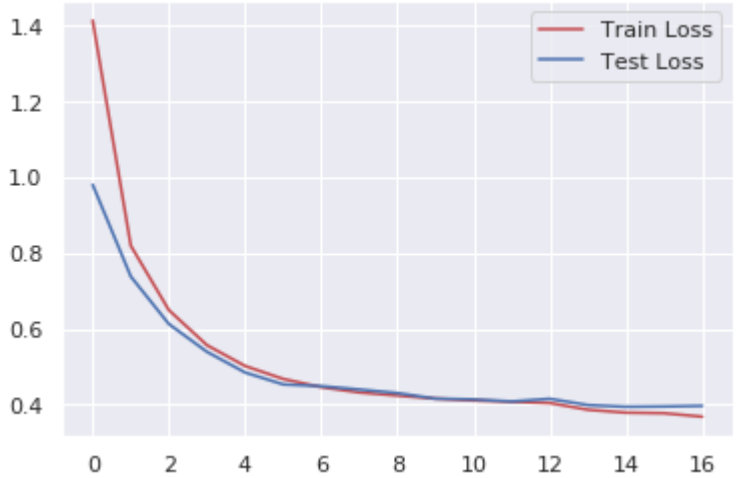
In [503]:
```
results = confusion_matrix(Y_test,Y_pred)
results
```

Out[503]:

Pred	Approved	Rejected
True		
Approved	30134	459
Rejected	4801	658

In [504]:
```
plt.plot(history_2.history['auroc'], 'r')
plt.plot(history_2.history['val_auroc'], 'b')
plt.legend({'Train ROCAUC': 'r', 'Test ROCAUC':'b'})
plt.show()


plt.plot(history_2.history['loss'], 'r')
plt.plot(history_2.history['val_loss'], 'b')
plt.legend({'Train Loss': 'r', 'Test Loss':'b'})
plt.show()
```





Model is little bit overfitting because at end, train AUC(0.7636) hs higher than test AUC(0.7595)

In [0]: