```
from google.colab import drive
drive.mount('/content/drive')
%cd ./drive/My Drive/LSTM
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473

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

```
import numpy as np
import pandas as pd
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Input , Dropout
from keras.layers import Flatten
from keras.layers import concatenate
from keras.layers.embeddings import Embedding
from keras.models import Model
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
import matplotlib.pyplot as plt
import pickle

import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
from sklearn.preprocessing import LabelEncoder
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
from tqdm import tqdm
import os
```

```
from chart_studio import plotly
import plotly.offline as offline
from keras.layers import LSTM
from keras.preprocessing.text import text_to_word_sequence
import tensorflow as tf
from keras.callbacks import ModelCheckpoint,TensorBoard,ReduceLROnPlateau, EarlyStopping
from keras.layers.normalization import BatchNormalization
from sklearn.feature_extraction.text import TfidfVectorizer
import seaborn as sns
from keras.regularizers import l2
from sklearn.metrics import roc_auc_score
from keras.models import load_model
from IPython.display import Image
from scipy.sparse import hstack
from keras.layers import Conv1D
from sklearn.feature_extraction.text import CountVectorizer
from prettytable import PrettyTable
```

```
Using TensorFlow backend.
```

```
X = pd.read_csv('preprocessed_data.csv')
X=X[0:50000]
print(X.columns)
X.head(2)
```

```
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')
```

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

```
Y=X['project_is_approved']
X=X.drop(['project_is_approved'],axis=1)
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,stratify=Y)
x_train,x_cv,y_train,y_cv=train_test_split(x_train,y_train,test_size=0.25,stratify=y_train
```

```
x_train.head(2)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previou
2247	fl	ms	grades_prek_2	
6874	tx	ms	grades_6_8	

```
print(x_train.shape, y_train.shape)
print(x_cv.shape, y_cv.shape)
print(x_test.shape, y_test.shape)
```

```
(30000, 8) (30000,)
(10000, 8) (10000,)
(10000, 8) (10000,)
```

```
#https://stackoverflow.com/questions/21057621/sklearn-labelencoder-with-never-seen-before-

class LabelEncoderExt(object):
    def __init__(self):
        """
        It differs from LabelEncoder by handling new classes and providing a value for it
        Unknown will be added in fit and transform will take care of new item. It gives un
        """
        self.label_encoder = LabelEncoder()
        # self.classes_ = self.label_encoder.classes_

    def fit(self, data_list):
        """
        This will fit the encoder for all the unique values and introduce unknown value
        :param data_list: A list of string
        :return: self
        """
        self.label_encoder = self.label_encoder.fit(list(data_list) + ['Unknown'])
        self.classes_ = self.label_encoder.classes_

        return self

    def transform(self, data_list):
        """
        This will transform the data_list to id list where the new values get assigned to
        :param data_list:
        :return:
        """
        new_data_list = list(data_list)
        for unique_item in np.unique(data_list):
            if unique_item not in self.label_encoder.classes_:
                new_data_list = ['Unknown' if x==unique_item else x for x in new_data_list

        return self.label_encoder.transform(new_data_list)
```

```
x_train.columns
```

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

```
label_encoder = LabelEncoderExt()
label_encoder.fit(x_train['teacher_prefix'].values)
x_train_teacher_ohe=label_encoder.transform(x_train['teacher_prefix'].values)
x_cv_teacher_ohe=label_encoder.transform(x_cv['teacher_prefix'].values)
x_test_teacher_ohe=label_encoder.transform(x_test['teacher_prefix'].values)


label_encoder = LabelEncoderExt()
label_encoder.fit(x_train['school_state'].values)
x_train_school_ohe=label_encoder.transform(x_train['school_state'].values)
x_cv_school_ohe=label_encoder.transform(x_cv['school_state'].values)
x_test_school_ohe=label_encoder.transform(x_test['school_state'].values)

label_encoder = LabelEncoderExt()
label_encoder.fit(x_train['school_state'].values)
x_train_project_ohe=label_encoder.transform(x_train['project_grade_category'].values)
x_cv_project_ohe=label_encoder.transform(x_cv['project_grade_category'].values)
x_test_project_ohe=label_encoder.transform(x_test['project_grade_category'].values)


label_encoder = LabelEncoderExt()
label_encoder.fit(x_train['school_state'].values)
x_train_clean_cat_ohe=label_encoder.transform(x_train['clean_categories'].values)
x_cv_clean_cat_ohe=label_encoder.transform(x_cv['clean_categories'].values)
x_test_clean_cat_ohe=label_encoder.transform(x_test['clean_categories'].values)

label_encoder = LabelEncoderExt()
label_encoder.fit(x_train['school_state'].values)
x_train_clean_subcat_ohe=label_encoder.transform(x_train['clean_subcategories'].values)
x_cv_clean_subcat_ohe=label_encoder.transform(x_cv['clean_subcategories'].values)
x_test_clean_subcat_ohe=label_encoder.transform(x_test['clean_subcategories'].values)
```

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(x_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1

x_train_teacher_no = normalizer.transform(x_train['teacher_number_of_previously_posted_pro
x_cv_teacher_no = normalizer.transform(x_cv['teacher_number_of_previously_posted_projects'
x_test_teacher_no = normalizer.transform(x_test['teacher_number_of_previously_posted_proje

print("After vectorizations")
print(x_train_teacher_no.shape, y_train.shape)
print(x_cv_teacher_no.shape, y_cv.shape)
print(x_test_teacher_no.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(30000, 1) (30000,)
(10000, 1) (10000,)
(10000, 1) (10000,)
=====================================================================================
```

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(x_train['price'].values.reshape(1,-1))

x_train_price_norm = normalizer.transform(x_train['price'].values.reshape(-1,1))
x_cv_price_norm = normalizer.transform(x_cv['price'].values.reshape(-1,1))
x_test_price_norm = normalizer.transform(x_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(x_train_price_norm.shape, y_train.shape)
print(x_cv_price_norm.shape, y_cv.shape)
print(x_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(30000, 1) (30000,)
(10000, 1) (10000,)
(10000, 1) (10000,)
=====================================================================================
```

```
remaining_train = np.hstack((x_train_price_norm,x_train_teacher_no))
remaining_cv = np.hstack((x_cv_price_norm,x_cv_teacher_no))
remaining_test = np.hstack((x_test_price_norm,x_test_teacher_no))
```

```
max_length=300
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
def padded(encoded_docs):
  max_length = 300
  padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
  return padded_docs
```

```
#https://stackoverflow.com/posts/51956230/revisions
t = Tokenizer()
t.fit_on_texts(x_train.essay)
vocab_size = len(t.word_index) + 1
# integer encode the documents
encoded_docs = t.texts_to_sequences(x_train.essay)
essay_padded_train = padded(encoded_docs)
encoded_docs = t.texts_to_sequences(x_cv.essay)
essay_padded_cv = padded(encoded_docs)
encoded_docs = t.texts_to_sequences(x_test.essay)
essay_padded_test = padded(encoded_docs)


with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())

# for train
embedding_matrix= np.zeros((vocab_size, 300))
for word, i in t.word_index.items():
    if word in glove_words:
        embedding_vector = model[word]
        embedding_matrix[i] = embedding_vector
print("embedding matrix shape",embedding_matrix.shape)
```

```
embedding matrix shape (33868, 300)
```

```
y_train = to_categorical(y_train, num_classes=2)
y_cv = to_categorical(y_cv, num_classes=2)
y_test = to_categorical(y_test, num_classes=2)


from tensorboardcolab import *
from keras.regularizers import l2
from keras.layers import LeakyReLU
import keras.backend as K
#K.clear_session()



#auc
def auroc(y_true, y_pred):
    return tf.py_func(roc_auc_score, (y_true, y_pred), tf.double)
```

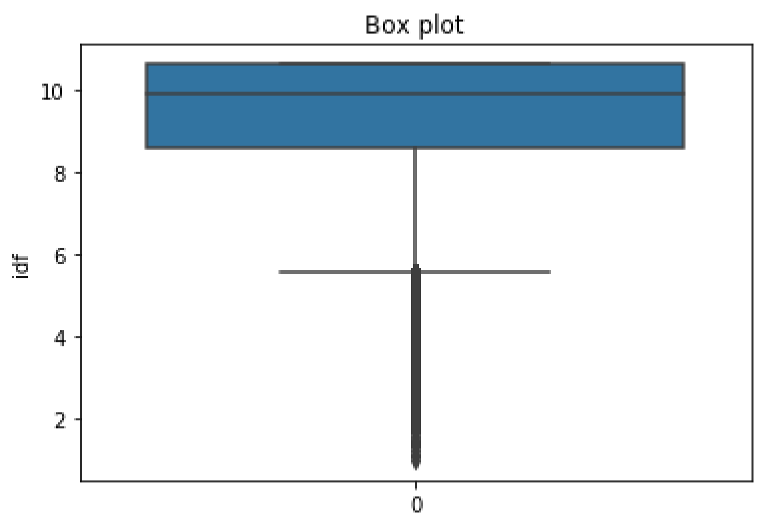
Model 2

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
vectorizer.fit_transform(x_train.essay)
```

```
<30000x33831 sparse matrix of type '<class 'numpy.float64'>'
        with 3261629 stored elements in Compressed Sparse Row format>
```

```
idf=vectorizer.idf
```

```
import seaborn as sns
sns.boxplot(data=idf)
plt.title('Box plot')
plt.ylabel('idf')
plt.show()
```



```
arg=[]#indices
for i in range(len(idf)):
  if idf[i]>=2and idf[i]<=11:
    arg.append(i)
```

```
import numpy as np
features= np.asarray(vectorizer.get_feature_names())

words = []
for i in arg:
    words.append(features[i])
```

```
# train_data
x_train_essay = []
for sentence in x_train['essay']:
    sent = []
    for word in sentence.split():
        if word in words:
            sent.append(word)
    x_train_essay.append(' '.join(sent))

#cv_data
x_cv_essay = []
for sentence in x_cv['essay']:
    sent = []
    for word in sentence.split():
        if word in words:
            sent.append(word)
    x_cv_essay.append(' '.join(sent))

#test_data
x_test_essay = []
for sentence in x_test['essay']:
    sent = []
    for word in sentence.split():
        if word in words:
            sent.append(word)
    x_test_essay.append(' '.join(sent))


max_length=300
#https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
def padded(encoded_docs):
  max_length = 300
  padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
  return padded_docs
#https://stackoverflow.com/posts/51956230/revisions
t = Tokenizer()
t.fit_on_texts(x_train_essay)
vocab_size = len(t.word_index) + 1
# integer encode the documents
encoded_docs = t.texts_to_sequences(x_train_essay)
essay_padded_train = padded(encoded_docs)
encoded_docs = t.texts_to_sequences(x_cv_essay)
essay_padded_cv = padded(encoded_docs)
encoded_docs = t.texts_to_sequences(x_test_essay)
essay_padded_test = padded(encoded_docs)


with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())

# for train
embedding_matrix= np.zeros((vocab_size, 300))
for word, i in t.word_index.items():
    if word in glove_words:
        embedding_vector = model[word]
        embedding_matrix[i] = embedding_vector
print("embedding matrix shape",embedding_matrix.shape)
```

```
embedding matrix shape (33806, 300)
```

```
    K.clear_session()
    essay_input = Input(shape=(300,), name='essay_input')

    x = Embedding(vocab_size, 300, weights=[embedding_matrix], input_length=300)(essay_input)
    lstm_out = LSTM(100,recurrent_dropout=0.5,return_sequences=True)(x)
    flatten_1 = Flatten()(lstm_out)

    state = Input(shape=(1,), name='school_state')
    x = Embedding(52, 10, input_length=1)(state)
    flatten_2 = Flatten()(x)
```

```
project_grade_category = Input(shape=(1,), name='project_grade_category')
x = Embedding(5, 10, input_length=1)(project_grade_category)
flatten_3 = Flatten()(x)


clean_categories = Input(shape=(1,), name='clean_categories')
x = Embedding(51, 10, input_length=1)(clean_categories)
flatten_4 = Flatten()(x)


clean_sub_categories = Input(shape=(1,), name='clean_sub_categories')
x = Embedding(393, 10, input_length=1)(clean_sub_categories)
flatten_5 = Flatten()(x)


teacher_prefix = Input(shape=(1,), name='teacher_prefix')
x = Embedding(6, 10, input_length=1)(teacher_prefix)
flatten_6 = Flatten()(x)


remaining_input = Input(shape=(2,), name='remaining_input')
dense_1 = Dense(1, activation='relu',kernel_initializer="he_normal",kernel_regularizer=l2(



x = concatenate([flatten_1,flatten_2,flatten_3,flatten_4,flatten_5,flatten_6,dense_1])



x= Dense(256,kernel_initializer='glorot_normal',kernel_regularizer=l2(0.001))(x)
x= LeakyReLU(alpha = 0.3)(x)
x= Dropout(0.6)(x)
x=BatchNormalization()(x)
x= Dense(128,kernel_initializer='glorot_normal',kernel_regularizer=l2(0.001))(x)
x= LeakyReLU(alpha = 0.3)(x)
x= Dropout(0.5)(x)
x= Dense(64,kernel_initializer='glorot_normal',kernel_regularizer=l2(0.001))(x)
x= LeakyReLU(alpha = 0.3)(x)
x= Dropout(0.4)(x)

x=BatchNormalization()(x)
x= Dense(32,kernel_initializer='glorot_normal',kernel_regularizer=l2(0.001))(x)
x= LeakyReLU(alpha = 0.3)(x)
x= Dropout(0.5)(x)

x= Dense(16,activation='relu',kernel_initializer='glorot_normal',kernel_regularizer=l2(0.0
x= LeakyReLU(alpha = 0.3)(x)
final_output = Dense(2, activation='sigmoid')(x)

model = Model(inputs=[essay_input,state,project_grade_category,clean_categories,clean_sub_
print(model.summary())
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:Large dropout rate: 0.6 (>0.5). In TensorFlow 2.x, dropout() uses
Model: "model_1"
________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
========================================================================================
essay_input (InputLayer)        (None, 300)          0
________________________________________________________________________________________
embedding_1 (Embedding)         (None, 300, 300)     10141800    essay_input[0][0]
________________________________________________________________________________________
school_state (InputLayer)       (None, 1)            0
________________________________________________________________________________________
project_grade_category (InputLa (None, 1)            0
________________________________________________________________________________________
clean_categories (InputLayer)   (None, 1)            0
________________________________________________________________________________________
clean_sub_categories (InputLaye (None, 1)            0
________________________________________________________________________________________
teacher_prefix (InputLayer)     (None, 1)            0
________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 300, 100)     160400      embedding_1[0][0]
________________________________________________________________________________________
embedding_2 (Embedding)         (None, 1, 10)        520         school_state[0][0]
________________________________________________________________________________________
embedding_3 (Embedding)         (None, 1, 10)        50          project_grade_catego
________________________________________________________________________________________
embedding_4 (Embedding)         (None, 1, 10)        510         clean_categories[0][
________________________________________________________________________________________
embedding_5 (Embedding)         (None, 1, 10)        3930        clean_sub_categories
________________________________________________________________________________________
embedding_6 (Embedding)         (None, 1, 10)        60          teacher_prefix[0][0]
________________________________________________________________________________________
remaining_input (InputLayer)    (None, 2)            0
________________________________________________________________________________________
flatten_1 (Flatten)             (None, 30000)        0           lstm_1[0][0]
```

```
________________________________________________________________________________________
flatten_2 (Flatten)             (None, 10)           0           embedding_2[0][0]
________________________________________________________________________________________
flatten_3 (Flatten)             (None, 10)           0           embedding_3[0][0]
________________________________________________________________________________________
flatten_4 (Flatten)             (None, 10)           0           embedding_4[0][0]
________________________________________________________________________________________
flatten_5 (Flatten)             (None, 10)           0           embedding_5[0][0]
________________________________________________________________________________________
flatten_6 (Flatten)             (None, 10)           0           embedding_6[0][0]
________________________________________________________________________________________
dense_1 (Dense)                 (None, 1)            3           remaining_input[0][0
________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 30051)        0           flatten_1[0][0]
                                                                 flatten_2[0][0]
                                                                 flatten_3[0][0]
                                                                 flatten_4[0][0]
                                                                 flatten_5[0][0]
                                                                 flatten_6[0][0]
                                                                 dense_1[0][0]
________________________________________________________________________________________
dense_2 (Dense)                 (None, 256)          7693312     concatenate_1[0][0]
________________________________________________________________________________________
leaky_re_lu_1 (LeakyReLU)       (None, 256)          0           dense_2[0][0]
________________________________________________________________________________________
dropout_1 (Dropout)             (None, 256)          0           leaky_re_lu_1[0][0]
________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 256)          1024        dropout_1[0][0]
________________________________________________________________________________________
dense_3 (Dense)                 (None, 128)          32896       batch_normalization_
________________________________________________________________________________________
leaky_re_lu_2 (LeakyReLU)       (None, 128)          0           dense_3[0][0]
________________________________________________________________________________________
dropout_2 (Dropout)             (None, 128)          0           leaky_re_lu_2[0][0]
________________________________________________________________________________________
dense_4 (Dense)                 (None, 64)           8256        dropout_2[0][0]
________________________________________________________________________________________
leaky_re_lu_3 (LeakyReLU)       (None, 64)           0           dense_4[0][0]
________________________________________________________________________________________
dropout_3 (Dropout)             (None, 64)           0           leaky_re_lu_3[0][0]
________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 64)           256         dropout_3[0][0]
________________________________________________________________________________________
dense_5 (Dense)                 (None, 32)           2080        batch_normalization_
________________________________________________________________________________________
leaky_re_lu_4 (LeakyReLU)       (None, 32)           0           dense_5[0][0]
________________________________________________________________________________________
dropout_4 (Dropout)             (None, 32)           0           leaky_re_lu_4[0][0]
________________________________________________________________________________________
dense_6 (Dense)                 (None, 16)           528         dropout_4[0][0]
________________________________________________________________________________________
leaky_re_lu_5 (LeakyReLU)       (None, 16)           0           dense_6[0][0]
________________________________________________________________________________________
dense_7 (Dense)                 (None, 2)            34          leaky_re_lu_5[0][0]
========================================================================================
Total params: 18,045,659
Trainable params: 18,045,019
Non-trainable params: 640
________________________________________________________________________________________
None
```

```
checkpoint_2 = ModelCheckpoint("model_2.h5",
                             monitor="val_auroc",
                             mode="max",
                             save_best_only = True,
                             verbose=1)
earlystop_2 = EarlyStopping(monitor = 'val_auroc',
                            mode="max",
                            min_delta = 0,
                            patience = 3,
                            verbose = 1,)
#reduce_lr_2 = ReduceLROnPlateau(monitor = 'val_loss', factor = 0.2, patience = 2, verbose

tensorboard_2 = TensorBoard(log_dir='graph_2', histogram_freq=0, batch_size=512, write_gra

callbacks_2 = [checkpoint_2,earlystop_2,tensorboard_2]


train = [essay_padded_train,x_train_school_ohe,x_train_project_ohe,x_train_clean_cat_ohe,x
cv=[essay_padded_cv,x_cv_school_ohe,x_cv_project_ohe,x_cv_clean_cat_ohe,x_cv_clean_subcat_


from keras.optimizers import adam
optim=adam(lr=0.001)
model.compile(optimizer=optim, loss='categorical_crossentropy', metrics=[auroc])
history_2 = model.fit(train, y_train, batch_size=512, epochs=10, verbose=1,callbacks=callb
```

```
Train on 30000 samples, validate on 10000 samples
Epoch 1/10
30000/30000 [==============================] - 400s 13ms/step - loss: 0.8068 - auroc:

Epoch 00001: val_auroc improved from -inf to 0.70813, saving model to model_2.h5
Epoch 2/10
30000/30000 [==============================] - 399s 13ms/step - loss: 0.7065 - auroc:

Epoch 00002: val_auroc improved from 0.70813 to 0.72363, saving model to model_2.h5
Epoch 3/10
30000/30000 [==============================] - 398s 13ms/step - loss: 0.6410 - auroc:

Epoch 00003: val_auroc improved from 0.72363 to 0.73101, saving model to model_2.h5
Epoch 4/10
30000/30000 [==============================] - 399s 13ms/step - loss: 0.5933 - auroc:

Epoch 00004: val_auroc did not improve from 0.73101
Epoch 5/10
30000/30000 [==============================] - 395s 13ms/step - loss: 0.5371 - auroc:

Epoch 00005: val_auroc did not improve from 0.73101
Epoch 6/10
30000/30000 [==============================] - 394s 13ms/step - loss: 0.4911 - auroc:

Epoch 00006: val_auroc did not improve from 0.73101
Epoch 00006: early stopping
```

```
test=[essay_padded_test,x_test_school_ohe,x_test_project_ohe,x_test_clean_cat_ohe,x_test_c
y_pred=model.predict(test)
a=roc_auc_score(y_test,y_pred)
```

```
print('Test auc score',a)
```

```
Test auc score 0.7020389201920845
```

```
%load_ext tensorboard
```

```
%tensorboard --logdir  graph_2
```

TensorBoard SCALARS GRAPHS

☐ Show data download links
☐ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis

STEP RELATIVE WALL

Runs

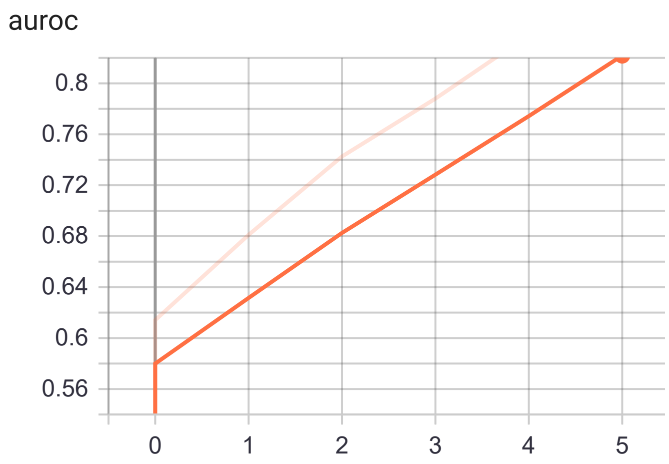
Write a regex to filter runs

TOGGLE ALL RUNS

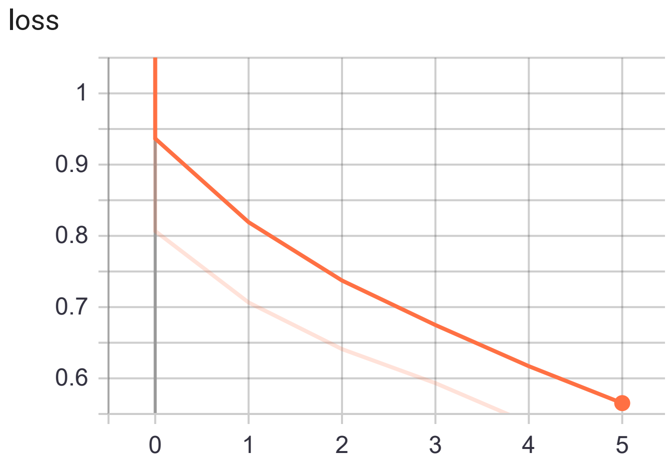
graph_2

Filter tags (regular expressions supported)

auroc



loss


loss