LSTM Assignment - 14 [Model 1 and 2]

In [1]:

```
# Importing all necessary files
%matplotlib inline
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 pickle import load,dump

from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from sklearn.metrics import roc_auc_score


import pickle
from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

Model 1

In [2]:

```
# keras imports
from sklearn.preprocessing import StandardScaler
from keras.layers import BatchNormalization,Dense,Dropout,Input,Embedding,LSTM,Flatten
from keras.layers import Conv1D
from keras.models import Model,Sequential
from keras.layers.merge import concatenate
from keras.preprocessing.sequence import pad_sequences
```

```
Using TensorFlow backend.
```

In [3]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [4]:

```
project_data.isnull().sum()
```

Out[4]:

```
Unnamed: 0                                          0
id                                                  0
teacher_id                                          0
teacher_prefix                                      3
school_state                                        0
project_submitted_datetime                          0
project_grade_category                              0
project_subject_categories                          0
project_subject_subcategories                       0
project_title                                       0
project_essay_1                                     0
project_essay_2                                     0
project_essay_3                                105490
project_essay_4                                105490
project_resource_summary                            0
teacher_number_of_previously_posted_projects        0
project_is_approved                                 0
dtype: int64
```

In [5]:

```
#filling 3 null teacher prefix values with Teacher

project_data["teacher_prefix"].fillna("Teacher",inplace = True)
project_data.isnull().sum()
```

Out[5]:

```
Unnamed: 0                                          0
id                                                  0
teacher_id                                          0
teacher_prefix                                      0
school_state                                        0
project_submitted_datetime                          0
project_grade_category                              0
project_subject_categories                          0
project_subject_subcategories                       0
project_title                                       0
project_essay_1                                     0
project_essay_2                                     0
project_essay_3                                105490
project_essay_4                                105490
project_resource_summary                            0
teacher_number_of_previously_posted_projects        0
project_is_approved                                 0
dtype: int64
```

In [6]:

```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [7]:

```
# comibining total text_data
project_data["combine"] = project_data["essay"] + project_data["project_title"]
```

In [8]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [9]:

```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109248 entries, 0 to 109247
```

```
Data columns (total 21 columns):
Unnamed: 0                                      109248 non-null int64
id                                              109248 non-null object
teacher_id                                      109248 non-null object
teacher_prefix                                  109248 non-null object
school_state                                    109248 non-null object
project_submitted_datetime                      109248 non-null object
project_grade_category                          109248 non-null object
project_subject_categories                      109248 non-null object
project_subject_subcategories                   109248 non-null object
project_title                                   109248 non-null object
project_essay_1                                 109248 non-null object
project_essay_2                                 109248 non-null object
project_essay_3                                 3758 non-null object
project_essay_4                                 3758 non-null object
project_resource_summary                        109248 non-null object
teacher_number_of_previously_posted_projects    109248 non-null int64
project_is_approved                             109248 non-null int64
essay                                           109248 non-null object
combine                                         109248 non-null object
price                                           109248 non-null float64
quantity                                        109248 non-null int64
dtypes: float64(1), int64(4), object(16)
memory usage: 18.3+ MB
```

In [10]:

```
project_data.columns
```

Out[10]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'project_subject_categories', 'project_subject_subcategories',
       'project_title', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'essay', 'combine', 'price', 'quantity'],
      dtype='object')
```

In [11]:

```
from sklearn.utils import resample
p_d = resample(project_data)


#splitting data as 30% to test
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer

p_d = resample(p_d,n_samples = 60000)

y = p_d["project_is_approved"]
X = p_d.drop("project_is_approved",axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/6, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=1/5, random_state=42)


print(X_train.shape," ",y_train.shape)
print(X_test.shape," ",y_test.shape)
print(X_val.shape," ",y_val.shape)
```

```
(40000, 20)   (40000,)
(10000, 20)   (10000,)
(10000, 20)   (10000,)
```

Preprocessing Text Data

In [12]:

```
#using function and stopwords form assignemnt
```

```
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase

# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [13]:

```
from tqdm import tqdm

#for train data
preprocessed_combine = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['combine'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_combine.append(sent.lower().strip())

test_preprocessed_combine = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['combine'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    test_preprocessed_combine.append(sent.lower().strip())
```

```
val_preprocessed_combine = []
# tqdm is for printing the status bar
for sentance in tqdm(X_val['combine'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    val_preprocessed_combine.append(sent.lower().strip())
```

```
100%|
| 40000/40000 [00:26<00:00,
1536.79it/s]
100%|
| 10000/10000 [00:06<00:00,
1536.26it/s]
100%|
| 10000/10000 [00:06<00:00,
1533.90it/s]
```

Embedding Essay

In [14]:

```
from keras.preprocessing.text import Tokenizer

# considering all the words.
token = Tokenizer()

# fitting on train data.
token.fit_on_texts(preprocessed_combine)
```

Caution:

Tokenizer considers only top words provided by num_words while converting to sequences. i.e. if word is not present the it skips the word. But token.index_word keeps all the record of words in whole vocabulary.So, remoove unnecessary words while using token.index_words as a dictionary.

In [15]:

```
# index to words are assigned according to frequency. i.e the most frequent word has index of 1
# MAX_WORDS = 47743
ix_to_word = token.index_word
MAX_WORDS = len(ix_to_word)+1

print("Total words considering for this assignment {}".format(MAX_WORDS))
```

```
Total words considering for this assignment 38180
```

In [16]:

```
for k in list(ix_to_word):
    if k>=MAX_WORDS:
        ix_to_word.pop(k, None)

word_to_ix = dict()
for k,v in ix_to_word.items():
    word_to_ix[v] = k
```

In [17]:

```
print(len(word_to_ix))
print(len(ix_to_word))
```

```
38179
38179
```

In [18]:

```
combine_sequence = token.texts_to_sequences(preprocessed_combine)
val_combine_sequence = token.texts_to_sequences(val_preprocessed_combine)
test_combine_sequence = token.texts_to_sequences(test_preprocessed_combine)

print(len(combine_sequence))
print(len(val_combine_sequence))
print(len(test_combine_sequence))
```

```
40000
10000
10000
```

- MAX_WORDS = 47742
- Sequence_length = 335

In [19]:

```
sequence_length_1 = 335

combine_sequence = pad_sequences(combine_sequence,maxlen=sequence_length_1,padding="post")
val_combine_sequence = pad_sequences(val_combine_sequence,maxlen=sequence_length_1,padding="post")
test_combine_sequence =
pad_sequences(test_combine_sequence,maxlen=sequence_length_1,padding="post")

print(combine_sequence.shape)
print(test_combine_sequence.shape)
print(val_combine_sequence.shape)
```

```
(40000, 335)
(10000, 335)
(10000, 335)
```

In [20]:

```
MAX_LENGTH = sequence_length_1
print("Maximum sequence length is {}".format(MAX_LENGTH))
print(combine_sequence.shape)
```

```
Maximum sequence length is 335
(40000, 335)
```

In [21]:

```
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    glove = load(f)
    glove_words =  set(glove.keys())
```

In [22]:

```
EMBEDDING_SIZE = 300
VOCAB_SIZE = MAX_WORDS

# Get 300-dim dense vector for each of the words in vocabulary
embedding_matrix = np.zeros((VOCAB_SIZE,EMBEDDING_SIZE))
embedding_matrix.shape
```

Out[22]:

```
(38180, 300)
```

In [23]:

```
# code for embedding matrix. considering top 5000 words and using already present glove vectors

# Get 300-dim dense vector for each of the words in vocabulary
embedding_matrix = np.zeros(((VOCAB_SIZE),EMBEDDING_SIZE))

for word, i in word_to_ix.items():
    embedding_vector = np.zeros(300)
    if word in glove_words:
        embedding_vector = glove[word]
        embedding_matrix[i] = embedding_vector
    else:
        # Words not found in the embedding index will be all zeros
        embedding_matrix[i] = embedding_vector
```

In [24]:

```
embedding_matrix.shape
```

Out[24]:

```
(38180, 300)
```

In [25]:

```
# save the embedding matrix to file
with open("embedding_matrix.pkl","wb") as f:
    dump(embedding_matrix,f)
```

Functional API for Essay

In [26]:

```
# functional api for essay
LSTM_units = 4

input_ess = Input(shape=(MAX_LENGTH,))
em1 = Embedding(MAX_WORDS,EMBEDDING_SIZE,input_length=MAX_LENGTH)(input_ess)
lstm = LSTM(LSTM_units,input_shape = (1,MAX_LENGTH), return_sequences=True,)(em1)
flt_ess  = Flatten()(lstm)
```

```
WARNING:tensorflow:From C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

Categorical Embedding

In [27]:

```
def generate_sequence(category , category_dict,df):

    """
    This function takes i/p as category name and its dictionary.

    convert them to sequences using pad_sequences from keras.

    Returns MAX_Length,VOCAB_SIZE and normalized_sequence for give category

    """
    word_to_ix = dict()
    ix_to_word = dict()

    # sorting dictionary in descending order
    category_dict = dict(sorted(category_dict.items(), key=lambda kv: kv[1],reverse = True))

    count = 1
    for k in category_dict.keys():
```

```
        ix_to_word[count] = k
        count += 1

    for k,v in ix_to_word.items():
        word_to_ix[v] = k

    print(word_to_ix)
    print("*"*50)
    print(ix_to_word)

    category_sequence = []
    for ec in df[category].values:
        temp = []
        for c in ec.split():
            temp.append(word_to_ix.get(c,0))
        category_sequence.append(temp)

    # conveting sequnce to same length using pad sequence in keras
    category_sequence = pad_sequences(category_sequence,padding="post",dtype="float32")
    #normalized_sequence = normalize(category_sequence)

    MAX_LENGTH = category_sequence.shape[1]
    VOCAB_SIZE = len(word_to_ix) + 1

    print("\nMaximum length of sequence is {}".format(MAX_LENGTH))
    print("\nVocabulary size is {}".format(VOCAB_SIZE))
    print("\nShape of {} category sequence is {}".format(category,category_sequence.shape))

    return category_sequence,MAX_LENGTH,VOCAB_SIZE
```

For categories and sub-categories i/p sequence can be vary.

for e.g. category can be [History Math].

So we are converting categories to max length sequence

project categories

In [28]:

```
catogories = list(X_train['project_subject_categories'].values)

cat_list = []
for i in catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp+=j.strip()+" "
        temp = temp.replace('&','_')
    cat_list.append(temp.strip())

X_train['clean_categories'] = cat_list
X_train.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in X_train['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))




# project subject categories for test data

catogories = list(X_test['project_subject_categories'].values)

cat_list = []
for i in catogories:
    temp = ""
```

```
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp+=j.strip()+" "
        temp = temp.replace('&','_')
    cat_list.append(temp.strip())

X_test['clean_categories'] = cat_list
X_test.drop(['project_subject_categories'], axis=1, inplace=True)


# project subject categories for val data

catogories = list(X_val['project_subject_categories'].values)

cat_list = []
for i in catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp+=j.strip()+" "
        temp = temp.replace('&','_')
    cat_list.append(temp.strip())

X_val['clean_categories'] = cat_list
X_val.drop(['project_subject_categories'], axis=1, inplace=True)
```

In [29]:

```
category_sequence,MAX_LENGTH,VOCAB_SIZE = generate_sequence("clean_categories",sorted_cat_dict,X_t
rain)
test_category_sequence,MAX_LENGTH,VOCAB_SIZE =
generate_sequence("clean_categories",sorted_cat_dict,X_test)
val_category_sequence,MAX_LENGTH,VOCAB_SIZE = generate_sequence("clean_categories",sorted_cat_dict
,X_val)
```

```
{'Literacy_Language': 1, 'Math_Science': 2, 'Health_Sports': 3, 'SpecialNeeds': 4,
'AppliedLearning': 5, 'Music_Arts': 6, 'History_Civics': 7, 'Warmth': 8, 'Care_Hunger': 9}
**************************************************
{1: 'Literacy_Language', 2: 'Math_Science', 3: 'Health_Sports', 4: 'SpecialNeeds', 5:
'AppliedLearning', 6: 'Music_Arts', 7: 'History_Civics', 8: 'Warmth', 9: 'Care_Hunger'}

Maximum length of sequence is 3

Vocabulary size is 10

Shape of clean_categories category sequence is (40000, 3)
{'Literacy_Language': 1, 'Math_Science': 2, 'Health_Sports': 3, 'SpecialNeeds': 4,
'AppliedLearning': 5, 'Music_Arts': 6, 'History_Civics': 7, 'Warmth': 8, 'Care_Hunger': 9}
**************************************************
{1: 'Literacy_Language', 2: 'Math_Science', 3: 'Health_Sports', 4: 'SpecialNeeds', 5:
'AppliedLearning', 6: 'Music_Arts', 7: 'History_Civics', 8: 'Warmth', 9: 'Care_Hunger'}

Maximum length of sequence is 3

Vocabulary size is 10

Shape of clean_categories category sequence is (10000, 3)
{'Literacy_Language': 1, 'Math_Science': 2, 'Health_Sports': 3, 'SpecialNeeds': 4,
'AppliedLearning': 5, 'Music_Arts': 6, 'History_Civics': 7, 'Warmth': 8, 'Care_Hunger': 9}
**************************************************
{1: 'Literacy_Language', 2: 'Math_Science', 3: 'Health_Sports', 4: 'SpecialNeeds', 5:
'AppliedLearning', 6: 'Music_Arts', 7: 'History_Civics', 8: 'Warmth', 9: 'Care_Hunger'}

Maximum length of sequence is 3

Vocabulary size is 10

Shape of clean_categories category sequence is (10000, 3)
```

In [30]:

```
# Embdeeing layer for category

input_pc = Input(shape=(MAX_LENGTH,))
em = Embedding(VOCAB_SIZE,10,input_length=MAX_LENGTH)(input_pc)
flt_pc  = Flatten()(em)

print("Output will contain shape of {}x{}".format(MAX_LENGTH,10))
```

```
Output will contain shape of 3x10
```

project subject sub_categories

In [31]:

```
sub_catogories = list(X_train['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','')
        temp +=j.strip()+" "
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

X_train['clean_subcategories'] = sub_cat_list
X_train.drop(['project_subject_subcategories'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in X_train['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))




sub_catogories = list(X_test['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','')
        temp +=j.strip()+" "
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

X_test['clean_subcategories'] = sub_cat_list
X_test.drop(['project_subject_subcategories'], axis=1, inplace=True)

sub_catogories = list(X_val['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','')
        temp +=j.strip()+" "
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

X_val['clean_subcategories'] = sub_cat_list
```

```
X_val.drop(['project_subject_subcategories'], axis=1, inplace=True)
```

In [32]:

```
subcategory_sequence,MAX_LENGTH,VOCAB_SIZE =
generate_sequence("clean_subcategories",sorted_sub_cat_dict,X_train)
val_subcategory_sequence,MAX_LENGTH,VOCAB_SIZE =
generate_sequence("clean_subcategories",sorted_sub_cat_dict,X_val)
test_subcategory_sequence,MAX_LENGTH,VOCAB_SIZE =
generate_sequence("clean_subcategories",sorted_sub_cat_dict,X_test)
```

```
{'Literacy': 1, 'Mathematics': 2, 'Literature_Writing': 3, 'SpecialNeeds': 4, 'AppliedSciences': 5
, 'Health_Wellness': 6, 'VisualArts': 7, 'EnvironmentalScience': 8, 'Gym_Fitness': 9, 'EarlyDevelop
ment': 10, 'Health_LifeScience': 11, 'ESL': 12, 'Music': 13, 'History_Geography': 14,
'College_CareerPrep': 15, 'Other': 16, 'CharacterEducation': 17, 'TeamSports': 18,
'PerformingArts': 19, 'SocialSciences': 20, 'NutritionEducation': 21, 'Warmth': 22, 'Care_Hunger':
23, 'ForeignLanguages': 24, 'Extracurricular': 25, 'Civics_Government': 26, 'ParentInvolvement': 2
7, 'FinancialLiteracy': 28, 'CommunityService': 29, 'Economics': 30}
**************************************************
{1: 'Literacy', 2: 'Mathematics', 3: 'Literature_Writing', 4: 'SpecialNeeds', 5:
'AppliedSciences', 6: 'Health_Wellness', 7: 'VisualArts', 8: 'EnvironmentalScience', 9:
'Gym_Fitness', 10: 'EarlyDevelopment', 11: 'Health_LifeScience', 12: 'ESL', 13: 'Music', 14: 'Hist
ory_Geography', 15: 'College_CareerPrep', 16: 'Other', 17: 'CharacterEducation', 18: 'TeamSports',
19: 'PerformingArts', 20: 'SocialSciences', 21: 'NutritionEducation', 22: 'Warmth', 23:
'Care_Hunger', 24: 'ForeignLanguages', 25: 'Extracurricular', 26: 'Civics_Government', 27:
'ParentInvolvement', 28: 'FinancialLiteracy', 29: 'CommunityService', 30: 'Economics'}

Maximum length of sequence is 3

Vocabulary size is 31

Shape of clean_subcategories category sequence is (40000, 3)
{'Literacy': 1, 'Mathematics': 2, 'Literature_Writing': 3, 'SpecialNeeds': 4, 'AppliedSciences': 5
, 'Health_Wellness': 6, 'VisualArts': 7, 'EnvironmentalScience': 8, 'Gym_Fitness': 9, 'EarlyDevelop
ment': 10, 'Health_LifeScience': 11, 'ESL': 12, 'Music': 13, 'History_Geography': 14,
'College_CareerPrep': 15, 'Other': 16, 'CharacterEducation': 17, 'TeamSports': 18,
'PerformingArts': 19, 'SocialSciences': 20, 'NutritionEducation': 21, 'Warmth': 22, 'Care_Hunger':
23, 'ForeignLanguages': 24, 'Extracurricular': 25, 'Civics_Government': 26, 'ParentInvolvement': 2
7, 'FinancialLiteracy': 28, 'CommunityService': 29, 'Economics': 30}
**************************************************
{1: 'Literacy', 2: 'Mathematics', 3: 'Literature_Writing', 4: 'SpecialNeeds', 5:
'AppliedSciences', 6: 'Health_Wellness', 7: 'VisualArts', 8: 'EnvironmentalScience', 9:
'Gym_Fitness', 10: 'EarlyDevelopment', 11: 'Health_LifeScience', 12: 'ESL', 13: 'Music', 14: 'Hist
ory_Geography', 15: 'College_CareerPrep', 16: 'Other', 17: 'CharacterEducation', 18: 'TeamSports',
19: 'PerformingArts', 20: 'SocialSciences', 21: 'NutritionEducation', 22: 'Warmth', 23:
'Care_Hunger', 24: 'ForeignLanguages', 25: 'Extracurricular', 26: 'Civics_Government', 27:
'ParentInvolvement', 28: 'FinancialLiteracy', 29: 'CommunityService', 30: 'Economics'}

Maximum length of sequence is 3

Vocabulary size is 31

Shape of clean_subcategories category sequence is (10000, 3)
{'Literacy': 1, 'Mathematics': 2, 'Literature_Writing': 3, 'SpecialNeeds': 4, 'AppliedSciences': 5
, 'Health_Wellness': 6, 'VisualArts': 7, 'EnvironmentalScience': 8, 'Gym_Fitness': 9, 'EarlyDevelop
ment': 10, 'Health_LifeScience': 11, 'ESL': 12, 'Music': 13, 'History_Geography': 14,
'College_CareerPrep': 15, 'Other': 16, 'CharacterEducation': 17, 'TeamSports': 18,
'PerformingArts': 19, 'SocialSciences': 20, 'NutritionEducation': 21, 'Warmth': 22, 'Care_Hunger':
23, 'ForeignLanguages': 24, 'Extracurricular': 25, 'Civics_Government': 26, 'ParentInvolvement': 2
7, 'FinancialLiteracy': 28, 'CommunityService': 29, 'Economics': 30}
**************************************************
{1: 'Literacy', 2: 'Mathematics', 3: 'Literature_Writing', 4: 'SpecialNeeds', 5:
'AppliedSciences', 6: 'Health_Wellness', 7: 'VisualArts', 8: 'EnvironmentalScience', 9:
'Gym_Fitness', 10: 'EarlyDevelopment', 11: 'Health_LifeScience', 12: 'ESL', 13: 'Music', 14: 'Hist
ory_Geography', 15: 'College_CareerPrep', 16: 'Other', 17: 'CharacterEducation', 18: 'TeamSports',
19: 'PerformingArts', 20: 'SocialSciences', 21: 'NutritionEducation', 22: 'Warmth', 23:
'Care_Hunger', 24: 'ForeignLanguages', 25: 'Extracurricular', 26: 'Civics_Government', 27:
'ParentInvolvement', 28: 'FinancialLiteracy', 29: 'CommunityService', 30: 'Economics'}

Maximum length of sequence is 3

Vocabulary size is 31

Shape of clean_subcategories category sequence is (10000, 3)
```

In [33]:

```
# Embdeeing layer for category

input_psc = Input(shape=(MAX_LENGTH,))
em = Embedding(VOCAB_SIZE,10,input_length=MAX_LENGTH)(input_psc)
flt_psc  = Flatten()(em)

print("Output will contain shape of {}x{}".format(MAX_LENGTH,10))
```

```
Output will contain shape of 3x10
```

Teacher Prefix

In [34]:

```
#preprocessing teacher prefix
prefix = list(X_train['teacher_prefix'].values)
prefix_list = []
for i in prefix:
    temp = ""
    if "." in i:
            i=i.replace('.','')
    temp+=i.strip()+" "
    prefix_list.append(temp.strip())

X_train['clean_prefix'] = prefix_list

my_counter = Counter()
for word in X_train['clean_prefix'].values:
  my_counter.update(word.split())

prefix_dict = dict(my_counter)
sorted_prefix_dict = dict(sorted(prefix_dict.items(), key=lambda kv: kv[1]))
print(sorted_prefix_dict)


#preprocessing teacher prefix for test data
prefix = list(X_test['teacher_prefix'].values)
prefix_list = []
for i in prefix:
    temp = ""
    if "." in i:
            i=i.replace('.','')
    temp+=i.strip()+" "
    prefix_list.append(temp.strip())

X_test['clean_prefix'] = prefix_list

#preprocessing teacher prefix for val data
prefix = list(X_val['teacher_prefix'].values)
prefix_list = []
for i in prefix:
    temp = ""
    if "." in i:
            i=i.replace('.','')
    temp+=i.strip()+" "
    prefix_list.append(temp.strip())

X_val['clean_prefix'] = prefix_list
```

```
{'Dr': 1, 'Teacher': 866, 'Mr': 3775, 'Ms': 14404, 'Mrs': 20954}
```

In [35]:

```
prefix_sequence,MAX_LENGTH,VOCAB_SIZE =
generate_sequence("clean_prefix",sorted_prefix_dict,X_train)
val_prefix_sequence,MAX_LENGTH,VOCAB_SIZE =
generate_sequence("clean_prefix",sorted_prefix_dict,X_val)
test_prefix_sequence,MAX_LENGTH,VOCAB_SIZE = generate_sequence("clean_prefix",sorted_prefix_dict,X
_test)
```

```
{'Mrs': 1, 'Ms': 2, 'Mr': 3, 'Teacher': 4, 'Dr': 5}
**************************************************
{1: 'Mrs', 2: 'Ms', 3: 'Mr', 4: 'Teacher', 5: 'Dr'}

Maximum length of sequence is 1

Vocabulary size is 6

Shape of clean_prefix category sequence is (40000, 1)
{'Mrs': 1, 'Ms': 2, 'Mr': 3, 'Teacher': 4, 'Dr': 5}
**************************************************
{1: 'Mrs', 2: 'Ms', 3: 'Mr', 4: 'Teacher', 5: 'Dr'}

Maximum length of sequence is 1

Vocabulary size is 6

Shape of clean_prefix category sequence is (10000, 1)
{'Mrs': 1, 'Ms': 2, 'Mr': 3, 'Teacher': 4, 'Dr': 5}
**************************************************
{1: 'Mrs', 2: 'Ms', 3: 'Mr', 4: 'Teacher', 5: 'Dr'}

Maximum length of sequence is 1

Vocabulary size is 6

Shape of clean_prefix category sequence is (10000, 1)
```

In [36]:

```
np.unique(prefix_sequence)
```

Out[36]:

```
array([1., 2., 3., 4., 5.], dtype=float32)
```

In [37]:

```
# Embdeeing layer for grade

input_tp = Input(shape=(1,))
em = Embedding(VOCAB_SIZE,10,input_length=MAX_LENGTH)(input_tp)
flt_tp  = Flatten()(em)
```

Grade Category

In [38]:

```
grade = list(X_train['project_grade_category'].values)
grade_list = []
for i in grade:
    temp = ""
    if "Grades" in i:
      i = i.replace("Grades","")
    if "6-8" in i:
      i = i.replace("6-8","six_eight")
    if "3-5" in i:
      i = i.replace("3-5","three_five")
    if "9-12" in i:
      i = i.replace("9-12","nine_twelve")
    if "PreK-2" in i:
      i = i.replace("PreK-2","prek_two")
    temp+=i.strip()+" "
    grade_list.append(temp.strip())

X_train['clean_grade'] = grade_list

my_counter = Counter()
for word in X_train['clean_grade'].values:
  my_counter.update(word.split())

grade_dict = dict(my_counter)
sorted_grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))
```

```
print(sorted_grade_dict)



# preprocessing of grade category for test data

grade = list(X_test['project_grade_category'].values)
grade_list = []
for i in grade:
    temp = ""
    if "Grades" in i:
      i = i.replace("Grades","")
    if "6-8" in i:
      i = i.replace("6-8","six_eight")
    if "3-5" in i:
      i = i.replace("3-5","three_five")
    if "9-12" in i:
      i = i.replace("9-12","nine_twelve")
    if "PreK-2" in i:
      i = i.replace("PreK-2","prek_two")
    temp+=i.strip()+" "
    grade_list.append(temp.strip())

X_test['clean_grade'] = grade_list

# preprocessing of grade category for val data

grade = list(X_val['project_grade_category'].values)
grade_list = []
for i in grade:
    temp = ""
    if "Grades" in i:
      i = i.replace("Grades","")
    if "6-8" in i:
      i = i.replace("6-8","six_eight")
    if "3-5" in i:
      i = i.replace("3-5","three_five")
    if "9-12" in i:
      i = i.replace("9-12","nine_twelve")
    if "PreK-2" in i:
      i = i.replace("PreK-2","prek_two")
    temp+=i.strip()+" "
    grade_list.append(temp.strip())

X_val['clean_grade'] = grade_list
```

```
{'nine_twelve': 3954, 'six_eight': 6125, 'three_five': 13666, 'prek_two': 16255}
```

In [39]:

```
grade_sequence,MAX_LENGTH,VOCAB_SIZE = generate_sequence("clean_grade",sorted_grade_dict,X_train)
val_grade_sequence,MAX_LENGTH,VOCAB_SIZE = generate_sequence("clean_grade",sorted_grade_dict,X_val
)
test_grade_sequence,MAX_LENGTH,VOCAB_SIZE =
generate_sequence("clean_grade",sorted_grade_dict,X_test)
```

```
{'prek_two': 1, 'three_five': 2, 'six_eight': 3, 'nine_twelve': 4}
**************************************************
{1: 'prek_two', 2: 'three_five', 3: 'six_eight', 4: 'nine_twelve'}

Maximum length of sequence is 1

Vocabulary size is 5

Shape of clean_grade category sequence is (40000, 1)
{'prek_two': 1, 'three_five': 2, 'six_eight': 3, 'nine_twelve': 4}
**************************************************
{1: 'prek_two', 2: 'three_five', 3: 'six_eight', 4: 'nine_twelve'}

Maximum length of sequence is 1

Vocabulary size is 5

Shape of clean_grade category sequence is (10000, 1)
```

```
{'prek_two': 1, 'three_five': 2, 'six_eight': 3, 'nine_twelve': 4}
**************************************************
{1: 'prek_two', 2: 'three_five', 3: 'six_eight', 4: 'nine_twelve'}

Maximum length of sequence is 1

Vocabulary size is 5

Shape of clean_grade category sequence is (10000, 1)
```

In [40]:

```
# Embdeeing layer for grade

print("Input vocab size for grade category is {}".format(VOCAB_SIZE))

input_gc = Input(shape=(1,))
em = Embedding(VOCAB_SIZE,10,input_length=MAX_LENGTH)(input_gc)
flt_gc  = Flatten()(em)
```

```
Input vocab size for grade category is 5
```

School State

In [41]:

```
#no need of preprocessing on school state

state = X_train["school_state"].value_counts()
sorted_state = dict(state)
sorted_state_dict = dict(sorted(sorted_state.items(), key=lambda kv: kv[1]))
X_train["clean_state"] = X_train["school_state"]
print(sorted_state_dict)

#similarly for X_test
X_test["clean_state"] = X_test["school_state"]

#similarly for X_val
X_val["clean_state"] = X_val["school_state"]
```

```
{'VT': 25, 'WY': 34, 'ND': 52, 'MT': 91, 'SD': 94, 'NE': 101, 'RI': 106, 'AK': 110, 'NH': 143,
'DE': 149, 'WV': 175, 'NM': 183, 'DC': 196, 'ME': 196, 'HI': 200, 'KS': 229, 'IA': 255, 'ID': 274,
'CO': 408, 'AR': 422, 'KY': 428, 'MN': 439, 'OR': 479, 'MS': 482, 'NV': 500, 'MD': 546, 'UT': 583,
'CT': 613, 'AL': 646, 'WI': 672, 'TN': 681, 'VA': 733, 'AZ': 790, 'OK': 860, 'NJ': 870, 'WA': 890,
'MA': 907, 'MO': 913, 'LA': 914, 'OH': 926, 'IN': 928, 'MI': 1124, 'PA': 1204, 'GA': 1420, 'SC': 14
26, 'IL': 1592, 'NC': 1790, 'FL': 2318, 'NY': 2597, 'TX': 2732, 'CA': 5554}
```

In [42]:

```
state_sequence,MAX_LENGTH,VOCAB_SIZE = generate_sequence("clean_state",sorted_state_dict,X_train)
val_state_sequence,MAX_LENGTH,VOCAB_SIZE = generate_sequence("clean_state",sorted_state_dict,X_val
)
test_state_sequence,MAX_LENGTH,VOCAB_SIZE =
generate_sequence("clean_state",sorted_state_dict,X_test)
```

```
{'CA': 1, 'TX': 2, 'NY': 3, 'FL': 4, 'NC': 5, 'IL': 6, 'SC': 7, 'GA': 8, 'PA': 9, 'MI': 10, 'IN': 1
1, 'OH': 12, 'LA': 13, 'MO': 14, 'MA': 15, 'WA': 16, 'NJ': 17, 'OK': 18, 'AZ': 19, 'VA': 20, 'TN':
21, 'WI': 22, 'AL': 23, 'CT': 24, 'UT': 25, 'MD': 26, 'NV': 27, 'MS': 28, 'OR': 29, 'MN': 30, 'KY'
: 31, 'AR': 32, 'CO': 33, 'ID': 34, 'IA': 35, 'KS': 36, 'HI': 37, 'DC': 38, 'ME': 39, 'NM': 40, 'W
V': 41, 'DE': 42, 'NH': 43, 'AK': 44, 'RI': 45, 'NE': 46, 'SD': 47, 'MT': 48, 'ND': 49, 'WY': 50,
'VT': 51}
**************************************************
{1: 'CA', 2: 'TX', 3: 'NY', 4: 'FL', 5: 'NC', 6: 'IL', 7: 'SC', 8: 'GA', 9: 'PA', 10: 'MI', 11: 'IN
', 12: 'OH', 13: 'LA', 14: 'MO', 15: 'MA', 16: 'WA', 17: 'NJ', 18: 'OK', 19: 'AZ', 20: 'VA', 21: '
TN', 22: 'WI', 23: 'AL', 24: 'CT', 25: 'UT', 26: 'MD', 27: 'NV', 28: 'MS', 29: 'OR', 30: 'MN', 31:
'KY', 32: 'AR', 33: 'CO', 34: 'ID', 35: 'IA', 36: 'KS', 37: 'HI', 38: 'DC', 39: 'ME', 40: 'NM', 41
: 'WV', 42: 'DE', 43: 'NH', 44: 'AK', 45: 'RI', 46: 'NE', 47: 'SD', 48: 'MT', 49: 'ND', 50: 'WY',
51: 'VT'}

Maximum length of sequence is 1

Vocabulary size is 52
```

```
Shape of clean_state category sequence is (40000, 1)
{'CA': 1, 'TX': 2, 'NY': 3, 'FL': 4, 'NC': 5, 'IL': 6, 'SC': 7, 'GA': 8, 'PA': 9, 'MI': 10, 'IN': 1
1, 'OH': 12, 'LA': 13, 'MO': 14, 'MA': 15, 'WA': 16, 'NJ': 17, 'OK': 18, 'AZ': 19, 'VA': 20, 'TN':
21, 'WI': 22, 'AL': 23, 'CT': 24, 'UT': 25, 'MD': 26, 'NV': 27, 'MS': 28, 'OR': 29, 'MN': 30, 'KY'
: 31, 'AR': 32, 'CO': 33, 'ID': 34, 'IA': 35, 'KS': 36, 'HI': 37, 'DC': 38, 'ME': 39, 'NM': 40, 'W
V': 41, 'DE': 42, 'NH': 43, 'AK': 44, 'RI': 45, 'NE': 46, 'SD': 47, 'MT': 48, 'ND': 49, 'WY': 50,
'VT': 51}
**************************************************
{1: 'CA', 2: 'TX', 3: 'NY', 4: 'FL', 5: 'NC', 6: 'IL', 7: 'SC', 8: 'GA', 9: 'PA', 10: 'MI', 11: 'IN
', 12: 'OH', 13: 'LA', 14: 'MO', 15: 'MA', 16: 'WA', 17: 'NJ', 18: 'OK', 19: 'AZ', 20: 'VA', 21: '
TN', 22: 'WI', 23: 'AL', 24: 'CT', 25: 'UT', 26: 'MD', 27: 'NV', 28: 'MS', 29: 'OR', 30: 'MN', 31:
'KY', 32: 'AR', 33: 'CO', 34: 'ID', 35: 'IA', 36: 'KS', 37: 'HI', 38: 'DC', 39: 'ME', 40: 'NM', 41
: 'WV', 42: 'DE', 43: 'NH', 44: 'AK', 45: 'RI', 46: 'NE', 47: 'SD', 48: 'MT', 49: 'ND', 50: 'WY',
51: 'VT'}

Maximum length of sequence is 1

Vocabulary size is 52

Shape of clean_state category sequence is (10000, 1)
{'CA': 1, 'TX': 2, 'NY': 3, 'FL': 4, 'NC': 5, 'IL': 6, 'SC': 7, 'GA': 8, 'PA': 9, 'MI': 10, 'IN': 1
1, 'OH': 12, 'LA': 13, 'MO': 14, 'MA': 15, 'WA': 16, 'NJ': 17, 'OK': 18, 'AZ': 19, 'VA': 20, 'TN':
21, 'WI': 22, 'AL': 23, 'CT': 24, 'UT': 25, 'MD': 26, 'NV': 27, 'MS': 28, 'OR': 29, 'MN': 30, 'KY'
: 31, 'AR': 32, 'CO': 33, 'ID': 34, 'IA': 35, 'KS': 36, 'HI': 37, 'DC': 38, 'ME': 39, 'NM': 40, 'W
V': 41, 'DE': 42, 'NH': 43, 'AK': 44, 'RI': 45, 'NE': 46, 'SD': 47, 'MT': 48, 'ND': 49, 'WY': 50,
'VT': 51}
**************************************************
{1: 'CA', 2: 'TX', 3: 'NY', 4: 'FL', 5: 'NC', 6: 'IL', 7: 'SC', 8: 'GA', 9: 'PA', 10: 'MI', 11: 'IN
', 12: 'OH', 13: 'LA', 14: 'MO', 15: 'MA', 16: 'WA', 17: 'NJ', 18: 'OK', 19: 'AZ', 20: 'VA', 21: '
TN', 22: 'WI', 23: 'AL', 24: 'CT', 25: 'UT', 26: 'MD', 27: 'NV', 28: 'MS', 29: 'OR', 30: 'MN', 31:
'KY', 32: 'AR', 33: 'CO', 34: 'ID', 35: 'IA', 36: 'KS', 37: 'HI', 38: 'DC', 39: 'ME', 40: 'NM', 41
: 'WV', 42: 'DE', 43: 'NH', 44: 'AK', 45: 'RI', 46: 'NE', 47: 'SD', 48: 'MT', 49: 'ND', 50: 'WY',
51: 'VT'}

Maximum length of sequence is 1

Vocabulary size is 52

Shape of clean_state category sequence is (10000, 1)
```

In [43]:

```
# Embdeeing layer for school state

print("Input vocab size for school state is {}".format(VOCAB_SIZE))

input_ss = Input(shape=(MAX_LENGTH,))
em = Embedding(VOCAB_SIZE,10,input_length=MAX_LENGTH)(input_ss)
flt_ss  = Flatten()(em)
```

```
Input vocab size for school state is 52
```

Preprocessing Numerical Feature

In [44]:

```
from sklearn.preprocessing import StandardScaler


price_scalar = StandardScaler()
price_scalar.fit(project_data['price'].values.reshape(-1,1))
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

#train data price standardization
price_standardized = price_scalar.transform(X_train['price'].values.reshape(-1, 1))

#val data price stanardization. Fit method applied on X_train
val_price_standardized = price_scalar.transform(X_val['price'].values.reshape(-1, 1))

#test data price stanardization. Fit method applied on X_train
test_price_standardized = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
```

```
Mean : 298.1193425966608, Standard deviation : 367.49634838483496
```

In [45]:

```
warnings.filterwarnings("ignore")
price_scalar = StandardScaler()
price_scalar.fit(X_train["quantity"].values.reshape(-1, 1))
print(f"Mean of Quantity : {price_scalar.mean_[0]}, Standard deviation of Quantity :
{np.sqrt(price_scalar.var_[0])}")

#train data quantity standardization
quantity_standardized = price_scalar.transform(X_train["quantity"].values.reshape(-1, 1))

#val data quantity stanardization. Fit method applied on X_train
val_quantity_standardized = price_scalar.transform(X_val["quantity"].values.reshape(-1, 1))


#test data quantity stanardization. Fit method applied on X_train
test_quantity_standardized = price_scalar.transform(X_test["quantity"].values.reshape(-1, 1))
```

```
Mean of Quantity : 16.9709, Standard deviation of Quantity : 25.709789053782607
```

In [46]:

```
price_scalar = StandardScaler()
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

#train data ppp standardization
number_ppp_standardized =
price_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
1))

#val data price stanardization. Fit method applied on X_train
val_number_ppp_standardized =
price_scalar.transform(X_val['teacher_number_of_previously_posted_projects'].values.reshape(-1,
1))


#test data price stanardization. Fit method applied on X_train
test_number_ppp_standardized =
price_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)
)
```

```
Mean : 11.265275, Standard deviation : 28.110076559383025
```

In [47]:

```
numerical_feature = np.hstack((number_ppp_standardized,quantity_standardized,price_standardized))
print(numerical_feature.shape)

val_numerical_feature =
np.hstack((val_number_ppp_standardized,val_quantity_standardized,val_price_standardized))
print(val_numerical_feature.shape)

test_numerical_feature =
np.hstack((test_number_ppp_standardized,test_quantity_standardized,test_price_standardized))
print(test_numerical_feature.shape)
```

```
(40000, 3)
(10000, 3)
(10000, 3)
```

Model Preperation

Function layer

In [48]:

```
# last funtional model for numerical features
input_nf = Input(shape=(3,))
d_nf = Dense(1,activation="relu")(input_nf)
```

Main layer

In [49]:

```
# merging into main one
# essay + school_state + grade + teacher_prefix + category + sub_categpry + numerical
# 335+10+10+10+10+10+1 ==> 386

cnt = concatenate([flt_ess,flt_ss,flt_gc,flt_tp,flt_pc,flt_psc,d_nf])
dense = Dense(8,activation="relu")(cnt)

# dropout + dense
dp = Dropout(0.2)(dense)
dense2 = Dense(4,activation="relu")(dp)

# dropout + dense

dp = Dropout(0.2)(dense2)
dense3 = Dense(2,activation="relu")(dp)

output = Dense(1,activation="sigmoid")(dense3)
```

```
WARNING:tensorflow:From C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

In [50]:

```
# input sequence essay + school_state + grade + teacher_prefix + category + sub_categpry + numeric
al
model = Model(inputs =[input_ess,input_ss,input_gc,input_tp,input_pc,input_psc,input_nf],outputs =
output)
```

In [51]:

```
# Freezing essay embedding_layer from training

model.layers[1].set_weights([embedding_matrix])
model.layers[1].trainable = False
```

In [52]:

```
model.summary()
```

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_1 (InputLayer)         (None, 335)               0
_________________________________________________________________________________________________
embedding_1 (Embedding)      (None, 335, 300)          11454000    input_1[0][0]
_________________________________________________________________________________________________
input_6 (InputLayer)         (None, 1)                 0
_________________________________________________________________________________________________
input_5 (InputLayer)         (None, 1)                 0
_________________________________________________________________________________________________
input_4 (InputLayer)         (None, 1)                 0
_________________________________________________________________________________________________
input_2 (InputLayer)         (None, 3)                 0
_________________________________________________________________________________________________
input_3 (InputLayer)         (None, 3)                 0
_________________________________________________________________________________________________
lstm_1 (LSTM)                (None, 335, 4)            4880        embedding_1[0][0]
_________________________________________________________________________________________________
```

embedding_6 (Embedding)	(None, 1, 10)	520	input_6[0][0]
embedding_5 (Embedding)	(None, 1, 10)	50	input_5[0][0]
embedding_4 (Embedding)	(None, 1, 10)	60	input_4[0][0]
embedding_2 (Embedding)	(None, 3, 10)	100	input_2[0][0]
embedding_3 (Embedding)	(None, 3, 10)	310	input_3[0][0]
input_7 (InputLayer)	(None, 3)	0	
flatten_1 (Flatten)	(None, 1340)	0	lstm_1[0][0]
flatten_6 (Flatten)	(None, 10)	0	embedding_6[0][0]
flatten_5 (Flatten)	(None, 10)	0	embedding_5[0][0]
flatten_4 (Flatten)	(None, 10)	0	embedding_4[0][0]
flatten_2 (Flatten)	(None, 30)	0	embedding_2[0][0]
flatten_3 (Flatten)	(None, 30)	0	embedding_3[0][0]
dense_1 (Dense)	(None, 1)	4	input_7[0][0]
concatenate_1 (Concatenate)	(None, 1431)	0	flatten_1[0][0] flatten_6[0][0] flatten_5[0][0] flatten_4[0][0] flatten_2[0][0] flatten_3[0][0] dense_1[0][0]
dense_2 (Dense)	(None, 8)	11456	concatenate_1[0][0]
dropout_1 (Dropout)	(None, 8)	0	dense_2[0][0]
dense_3 (Dense)	(None, 4)	36	dropout_1[0][0]
dropout_2 (Dropout)	(None, 4)	0	dense_3[0][0]
dense_4 (Dense)	(None, 2)	10	dropout_2[0][0]
dense_5 (Dense)	(None, 1)	3	dense_4[0][0]

Total params: 11,471,429
Trainable params: 17,429
Non-trainable params: 11,454,000

Input Set

essay + school_state + grade + teacher_prefix + category + sub_categpry + numerical

In [53]:

```
numerical = np.hstack((number_ppp_standardized,quantity_standardized,price_standardized))
#set_ =
np.hstack((essay_sequence,grade_sequence,prefix_sequence,category_sequence,subcategory_sequence,nu
al))
print(numerical.shape)

test_numerical =
np.hstack((test_number_ppp_standardized,test_quantity_standardized,test_price_standardized))
```

(40000, 3)

In [54]:

```
# model compilation
model.compile(loss='binary_crossentropy', optimizer='adam')
```

generator function

In [55]:

```
# please refer https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8
# data generator, intended to be used in a call to model.fit_generator()
from numpy import array

def data_generator(df,batch_size,data_type = 'Train'):
    X1, X2,X3,X4,X5,X6,X7, y = list(), list(), list(),list(), list(), list(),list(),list()
    flag = True
    if data_type == 'Val':
        flag = False
    n=0
    # loop for ever over images
    while 1:
        for i in range(len(df)):
            n+=1
            if flag:
                X1.append(combine_sequence[i])
                X2.append(state_sequence[i])
                X3.append(grade_sequence[i])
                X4.append(prefix_sequence[i])
                X5.append(category_sequence[i])
                X6.append(subcategory_sequence[i])
                X7.append(numerical_feature[i])
                y.append(df.iloc[i])
            else:
                X1.append(val_combine_sequence[i])
                X2.append(val_state_sequence[i])
                X3.append(val_grade_sequence[i])
                X4.append(val_prefix_sequence[i])
                X5.append(val_category_sequence[i])
                X6.append(val_subcategory_sequence[i])
                X7.append(val_numerical_feature[i])
                y.append(df.iloc[i])
            if n==batch_size:
                yield [[array(X1), array(X2),array(X3), array(X4),array(X5), array(X6),array(X7)], array(y)]
                X1, X2,X3,X4,X5,X6,X7, y = list(), list(), list(),list(), list(), list(),list(),list()
                n=0
```

AUC Function

In [56]:

```
# https://datascience.stackexchange.com/questions/35775/how-to-find-auc-metric-value-for-keras-model
from sklearn import metrics
from keras import backend as K
from sklearn.metrics import roc_auc_score
import tensorflow as tf

# https://stackoverflow.com/questions/41032551/how-to-compute-receiving-operating-characteristic-roc-and-auc-in-keras

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

In [57]:

```
epochs = 10
batch_size = 64
val_batch_size = 32
steps = len(y_train)//batch_size
val_steps = len(y_val)//val_batch_size
```

In [58]:

```
# using tensorboard instance for callbacks
from time import time
from datetime import datetime
from tensorflow.python.keras.callbacks import TensorBoard
tensorboard = TensorBoard(log_dir="model1_logs_1/{}".format(time()))

# model compilation
model.compile(loss='binary_crossentropy', optimizer='adam',metrics=[auroc])
```

```
WARNING:tensorflow:From <ipython-input-56-e95ed8574a9a>:10: py_func (from
tensorflow.python.ops.script_ops) is deprecated and will be removed in a future version.
Instructions for updating:
tf.py_func is deprecated in TF V2. Instead, use
    tf.py_function, which takes a python function which manipulates tf eager
    tensors instead of numpy arrays. It's easy to convert a tf eager tensor to
    an ndarray (just call tensor.numpy()) but having access to eager tensors
    means `tf.py_function`s can use accelerators such as GPUs as well as
    being differentiable using a gradient tape.
```

In [59]:

```
# callbacks
import keras
filepath = "weights.{epoch:02d}-{val_loss:.2f}.hdf5"
history = keras.callbacks.History()
model_check = keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_on
ly=True, save_weights_only=False, mode='auto', period=1)

early = keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=0, verbose=1, mode=
'auto', baseline=None, restore_best_weights=True)
```

In [60]:

```
for i in range(epochs):
    print("Epoch {} start at time ".format(i),datetime.now())
    generator = data_generator(y_train,batch_size)
    val_generator = data_generator(y_val,batch_size,"Val")
    model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=2,callbacks=[tensorboar
d,history,model_check,early],validation_data=val_generator,validation_steps=val_steps)
    model.save_weights("model_1_epoch_{}.h5".format(i))
```

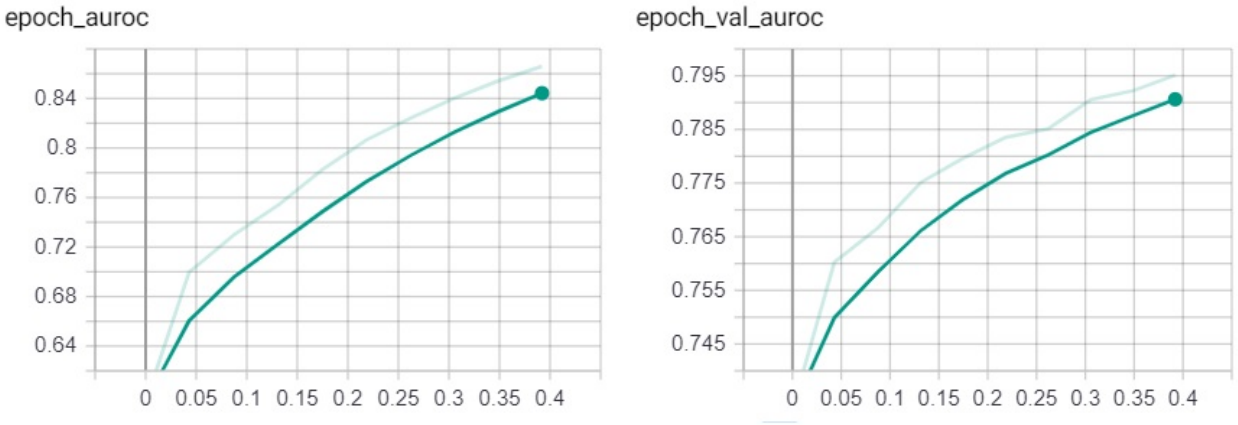
```
Epoch 0 start at time  2019-08-05 22:24:58.487459
WARNING:tensorflow:From C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/1
 - 153s - loss: 0.4422 - auroc: 0.5949 - val_loss: 0.3973 - val_auroc: 0.7327

Epoch 00001: val_loss improved from inf to 0.39731, saving model to weights.01-0.40.hdf5
Epoch 1 start at time  2019-08-05 22:27:35.152148
Epoch 1/1
 - 152s - loss: 0.4001 - auroc: 0.6996 - val_loss: 0.3828 - val_auroc: 0.7602

Epoch 00001: val_loss improved from 0.39731 to 0.38283, saving model to weights.01-0.38.hdf5
Epoch 2 start at time  2019-08-05 22:30:08.930159
Epoch 1/1
 - 161s - loss: 0.3880 - auroc: 0.7303 - val_loss: 0.3769 - val_auroc: 0.7667

Epoch 00001: val_loss improved from 0.38283 to 0.37685, saving model to weights.01-0.38.hdf5
Epoch 3 start at time  2019-08-05 22:32:50.839596
Epoch 1/1
 - 156s - loss: 0.3792 - auroc: 0.7540 - val_loss: 0.3720 - val_auroc: 0.7751

Epoch 00001: val_loss improved from 0.37685 to 0.37198, saving model to weights.01-0.37.hdf5
Epoch 4 start at time  2019-08-05 22:35:28.093063
Epoch 1/1
 - 155s - loss: 0.3680 - auroc: 0.7826 - val_loss: 0.3692 - val_auroc: 0.7796

Epoch 00001: val_loss improved from 0.37198 to 0.36920, saving model to weights.01-0.37.hdf5
Epoch 5 start at time  2019-08-05 22:38:04.521350
```

```
Epoch 1/1
 - 155s - loss: 0.3548 - auroc: 0.8066 - val_loss: 0.3666 - val_auroc: 0.7835

Epoch 00001: val_loss improved from 0.36920 to 0.36663, saving model to weights.01-0.37.hdf5
Epoch 6 start at time  2019-08-05 22:40:41.109184
Epoch 1/1
 - 156s - loss: 0.3427 - auroc: 0.8241 - val_loss: 0.3690 - val_auroc: 0.7851

Epoch 00001: val_loss did not improve from 0.36663
Epoch 7 start at time  2019-08-05 22:43:17.860101
Epoch 1/1
 - 155s - loss: 0.3331 - auroc: 0.8402 - val_loss: 0.3709 - val_auroc: 0.7905

Epoch 00001: val_loss did not improve from 0.36663
Epoch 8 start at time  2019-08-05 22:45:53.516107
Epoch 1/1
 - 157s - loss: 0.3216 - auroc: 0.8543 - val_loss: 0.3771 - val_auroc: 0.7922

Epoch 00001: val_loss did not improve from 0.36663
Epoch 9 start at time  2019-08-05 22:48:31.153699
Epoch 1/1
 - 153s - loss: 0.3115 - auroc: 0.8661 - val_loss: 0.3845 - val_auroc: 0.7951

Epoch 00001: val_loss did not improve from 0.36663
```

Loss and AUC

Epoch Loss And Validation Loss

Epoch AUC and Validation AUC



Model Prediction

In [61]:

```
test_set =
[test_combine_sequence,test_state_sequence,test_grade_sequence,test_prefix_sequence,test_category_s
equence,test_subcategory_sequence,test_numerical]
```

AUC Score

In [62]:

```
#generator = data_generator(y_test,1,1,batch_size)
history = model.predict(test_set)


# converting probabilistic values to class label. Threshold = 0.5
y_pred = (history > 0.7).astype(np.int)


print("AUC score is {}".format(roc_auc_score(y_test,y_pred)))
```

```
AUC score is 0.7073105970217028
```

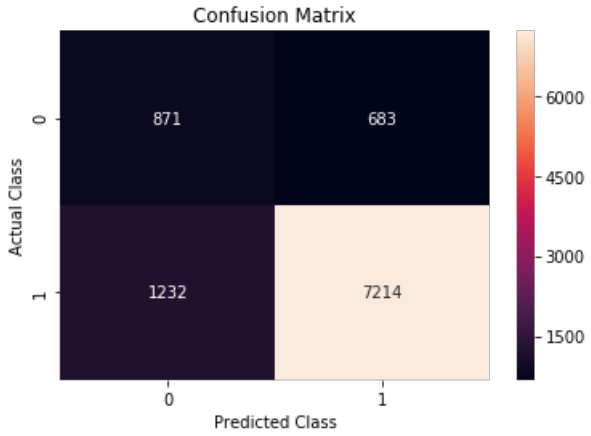
Confusion Matrix

In [63]:

```
from sklearn.metrics import confusion_matrix
cm1 = confusion_matrix(y_test,y_pred)
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.heatmap(cm1, annot=True, fmt="d")
plt.ylabel("Actual Class")
plt.xlabel("Predicted Class")
plt.title("Confusion Matrix")
```

Out[63]:

```
Text(0.5, 1.0, 'Confusion Matrix')
```



Model 2

In [64]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(min_df=10)
combine_tfidf = tfidf.fit_transform(preprocessed_combine)

# converting to dictionary
combine_dict = dict(zip(tfidf.get_feature_names(),list(tfidf.idf_)))
```

In [65]:

```
from collections import Counter
cnt =  Counter(list(tfidf.idf_))

cnt_idf = dict()
for k,v in cnt.items():
    cnt_idf[v] = k
```

In [66]:

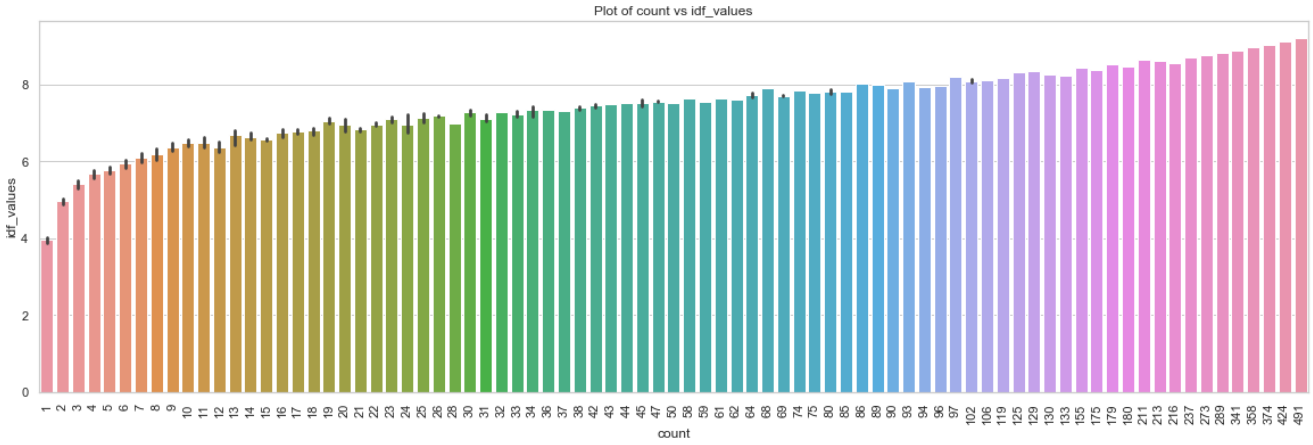
```
# getting key and values in list
x = []
y = []

for k,v in cnt.items():
    x.append(k)
    y.append(v)

# converting to dataframe
df = pd.DataFrame(data=x,columns=["idf_values"])
df["count"] = y
```

In [67]:

```
sns.set(style="whitegrid")
plt.figure(figsize=(20,6))
sns.barplot(x = "count",y = "idf_values",data=df,)
plt.xticks(rotation='vertical',)
plt.title("Plot of count vs idf_values")
```

Out[67]:

```
Text(0.5, 1.0, 'Plot of count vs idf_values')
```

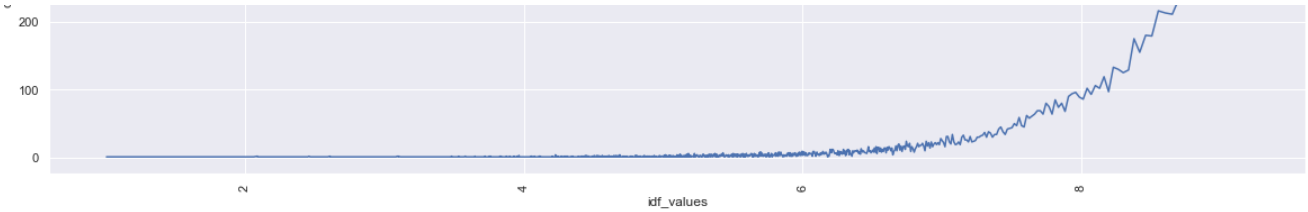


In [68]:

```
# plotting idf_ values count
sns.set(style="darkgrid")
plt.figure(figsize=(20,6))
sns.lineplot(x="idf_values",y ="count" ,data=df,markers=True, dashes=False,)
plt.xticks(rotation='vertical')
plt.title("Distribution of count over idf_values")
```

Out[68]:

```
Text(0.5, 1.0, 'Distribution of count over idf_values')
```



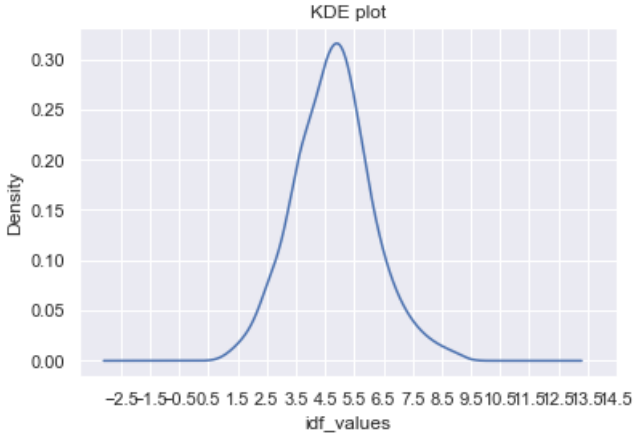
1. Idf values plot is skewed towards left.
2. Approx. idf_values greater than 8.5 occur more than 100 times.

In [69]:

```
# kde for idf_values
df["idf_values"].plot.kde()
plt.xticks(np.arange(-2.5,15,1.0))
plt.xlabel("idf_values")
plt.title("KDE plot")
plt.figure(figsize=(8,6))
```

Out[69]:

```
<Figure size 576x432 with 0 Axes>
```




```
<Figure size 576x432 with 0 Axes>
```

From above plot we can see that

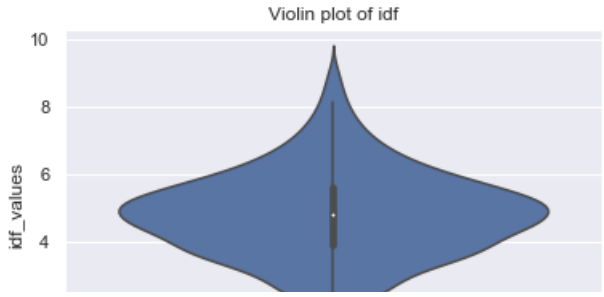
1. 10% of idf_ values lie below 3.0
2. **10-90 %** of idf_values lie in between range of 3.0 to 7.0

In [70]:

```
sns.violinplot(x = "idf_values",data=df,orient="v")
plt.xlabel("idf_")
plt.title("Violin plot of idf")
```

Out[70]:

```
Text(0.5, 1.0, 'Violin plot of idf')
```



We will consider idf*values from 3.0 to 7.0 for our model. Remoove all idf* values from essay_dict

In [71]:

```
combine_dict
```

Out[71]:


```
{'00': 7.202210578111139,
 '000': 5.916487123766512,
 '05': 8.461165516854429,
 '10': 4.473793074184496,
 '100': 3.7606851510620127,
 '1000': 7.142312436530071,
 '100th': 8.888609531681368,
 '101': 7.704839434672952,
 '102': 9.198764459985208,
 '103': 8.95760240316832,
 '104': 9.031710375322042,
 '107': 9.111753082995579,
 '10th': 6.752472646324987,
 '11': 5.638235039753797,
 '110': 7.553608464949028,
 '1100': 9.031710375322042,
 '115': 8.95760240316832,
 '11th': 6.981539215942319,
 '12': 4.7964896644813795,
 '120': 6.834485797985822,
 '1200': 8.130923829983853,
 '123': 8.824071010543797,
 '125': 8.130923829983853,
 '12th': 6.009411074383329,
 '13': 6.055396187625153,
 '130': 7.812470098865317,
 '13th': 8.888609531681368,
 '14': 5.879632031377357,
 '140': 8.229363902797104,
 '1400': 8.505617279425262,
 '14th': 9.111753082995579,
 '15': 5.181562773611983,
 '150': 6.684004847047526,
 '1500': 8.95760240316832,
 '16': 5.902927593980879,
 '160': 8.130923829983853,
 '17': 6.0019483531817395,
 '170': 8.888609531681368,
 '175': 9.111753082995579,
 '18': 5.429143241895237,
 '180': 7.130751614128995,
 '19': 6.226021704655916,
 '1960': 8.95760240316832,
 '1960s': 8.888609531681368,
 '1st': 4.8824892028741065,
 '20': 4.4884155930520375,
 '200': 6.216762379243119,
 '2000': 8.338563194762097,
 '2003': 8.652220753617138,
 '2004': 9.198764459985208,
 '2005': 9.198764459985208,
 '2008': 8.264455222608374,
 '2009': 8.652220753617138,
 '2010': 8.600927459229588,
 '2011': 7.985741820139354,
 '2012': 8.600927459229588,
 '2013': 8.505617279425262,
 '2014': 8.30082286677925,
 '2015': 7.064060239630322,
 '2016': 5.549287553737301,
 '2017': 5.987187937598619,
```

```
'2018': 7.933098086653932,
'209': 8.95760240316832,
'20th': 7.858990114500211,
'21': 5.630512993659886,
'21st': 3.966684025755789,
'22': 5.331358520045868,
'23': 5.825218609653563,
'24': 5.238817466275479,
'25': 4.778829161329428,
'250': 7.725458721875688,
'26': 5.837757958906298,
'27': 6.323660174219832,
'270': 9.031710375322042,
'28': 6.141338617425877,
'280': 8.763446388727363,
'29': 6.869271914071238,
'2d': 7.469525347738487,
'2nd': 4.5755757684944385,
'30': 4.61101791514437,
'300': 6.365551115928992,
'3000': 7.812470098865317,
'307': 9.198764459985208,
'31': 6.896179366991162,
'32': 6.47269575338032,
'320': 8.763446388727363,
'33': 7.042782841183038,
'34': 7.319993613767523,
'35': 6.268783563993997,
'350': 7.704839434672952,
'36': 6.971686919499308,
'360': 7.626367819231457,
'365': 9.031710375322042,
'37': 7.985741820139354,
'38': 7.589326547551107,
'380': 9.031710375322042,
'39': 7.985741820139354,
'3d': 5.482977552951348,
'3doodler': 7.333979855742263,
'3doodlers': 8.552137295060156,
'3rd': 4.398476155681635,
'40': 5.34475584961769,
'400': 6.308392702089043,
'41': 7.835459617090017,
'42': 7.607675686219304,
'420': 8.706287974887413,
'425': 8.824071010543797,
'43': 7.907780278669642,
'44': 7.422272462887942,
'45': 6.035978101768051,
'450': 6.905311850554435,
'46': 7.362553228186319,
'47': 7.407004990757153,
'475': 9.111753082995579,
'48': 7.252854310929894,
'480': 8.824071010543797,
'49': 7.789997243013259,
'4cs': 8.888609531681368,
'4k': 8.229363902797104,
'4th': 4.416589858480783,
'50': 4.829316607518186,
'500': 5.930233044671146,
'504': 7.74651213107352,
'51': 8.100152171317099,
'52': 7.502315170561478,
'520': 8.763446388727363,
'53': 7.725458721875688,
'54': 7.985741820139354,
'55': 7.165842933940265,
'550': 7.704839434672952,
'56': 8.041311671294164,
'560': 8.461165516854429,
'57': 8.100152171317099,
'58': 7.959073573057193,
'5k': 9.198764459985208,
'5th': 4.111168124752824,
'60': 4.564035471755572,
'600': 6.039831671084041,
```

```
'61': 8.229363902797104,
'62': 7.835459617090017,
'63': 8.195462351121424,
'64': 7.626367819231457,
'65': 6.349635660623092,
'650': 7.607675686219304,
'66': 7.74651213107352,
'67': 7.959073573057193,
'68': 7.469525347738487,
'69': 8.162672528298433,
'693': 9.031710375322042,
'6th': 4.5561233425676235,
'70': 5.350552967302016,
'700': 6.619925990363004,
'71': 7.684636727355432,
'72': 7.536216722237159,
'73': 8.01314079432747,
'74': 7.042782841183038,
'75': 5.38405363703206,
'750': 7.422272462887942,
'76': 7.858990114500211,
'77': 8.130923829983853,
'78': 7.607675686219304,
'79': 8.100152171317099,
'7th': 5.024377190089571,
'80': 5.054187772276774,
'800': 6.515255367799115,
'81': 7.858990114500211,
'82': 7.907780278669642,
'83': 7.883087666079271,
'84': 7.252854310929894,
'85': 5.764777255500062,
'850': 8.100152171317099,
'86': 7.7680183362944835,
'86th': 9.111753082995579,
'87': 7.5191222888877859,
'88': 7.7680183362944835,
'89': 7.835459617090017,
'8th': 4.673031104645151,
'90': 4.629692594169596,
'900': 6.744629468863962,
'91': 7.5191222888877859,
'92': 7.202210578111139,
'93': 7.57130804204843,
'94': 7.306200291635188,
'95': 5.638235039753797,
'96': 6.9426993826260555,
'97': 6.559707130369949,
'98': 5.803646124399434,
'99': 6.158580423860383,
'9th': 5.965447950962213,
'aba': 9.111753082995579,
'abandoned': 8.264455222608374,
'abc': 7.085800226266729,
'abcmouse': 9.198764459985208,
'abcs': 8.824071010543797,
'abcya': 8.377783907915378,
'abdominal': 8.30082286677925,
'abilities': 3.854691832962893,
'ability': 3.272808601444761,
'able': 1.8309411099923465,
'abled': 8.763446388727363,
'aboard': 8.824071010543797,
'abound': 8.652220753617138,
'about': 5.092371559246933,
'above': 7.6454160142021514,
'abraham': 8.461165516854429,
'abreast': 8.706287974887413,
'abroad': 8.552137295060156,
'absence': 8.461165516854429,
'absences': 8.600927459229588,
'absent': 7.074871155734538,
'absentee': 9.198764459985208,
'absenteeism': 8.763446388727363,
'absolute': 6.381723975174593,
'absolutely': 5.083429621871272,
'absorb': 6.259121653082261,
```

```
'absorbed': 8.888609531681368,
'absorbing': 8.195462351121424,
'abstract': 5.876347956176167,
'abstraction': 9.198764459985208,
'abstractly': 8.600927459229588,
'abundance': 6.760377825832101,
'abundant': 7.607675686219304,
'abuse': 6.729125282327996,
'abused': 8.377783907915378,
'abusive': 9.111753082995579,
'academia': 8.195462351121424,
'academic': 2.957780023110742,
'academically': 4.244859863725802,
'academics': 4.331929554853711,
'academy': 5.612723452096388,
'accelerate': 7.684636727355432,
'accelerated': 5.979888635117008,
'acceleration': 8.338563194762097,
'accents': 8.95760240316832,
'accept': 6.466761017860505,
'acceptable': 7.907780278669642,
'acceptance': 6.431873758860065,
'accepted': 6.303354908059086,
'accepting': 6.809167990001533,
'accepts': 8.95760240316832,
'access': 2.6118407369075696,
'accessed': 7.485785868610267,
'accessibility': 7.17781912498698,
'accessible': 4.913298787017304,
'accessing': 6.546803725534041,
'accessories': 6.2785397389393625,
'accessory': 8.30082286677925,
'accident': 8.162672528298433,
'accidentally': 7.812470098865317,
'accidents': 7.883087666079271,
'acclimate': 8.706287974887413,
'accolades': 9.031710375322042,
'accommodate': 5.696762379201087,
'accommodated': 8.763446388727363,
'accommodates': 8.229363902797104,
'accommodating': 7.858990114500211,
'accommodation': 8.041311671294164,
'accommodations': 6.515255367799115,
'accompanied': 8.30082286677925,
'accompaniment': 8.461165516854429,
'accompaniments': 9.031710375322042,
'accompany': 7.001539882648989,
'accompanying': 7.704839434672952,
'accomplish': 4.797603870724783,
'accomplished': 6.21216466999449,
'accomplishing': 7.279171619247268,
'accomplishment': 6.3871735799421 57,
'accomplishments': 6.193982350911299,
'according': 5.656488480063147,
'accordingly': 8.338563194762097,
'account': 6.713857810197208,
'accountability': 6.744629468863962,
'accountable': 6.3981627015177525,
'accounting': 8.95760240316832,
'accounts': 6.729125282327996,
'accreditation': 9.198764459985208,
'accredited': 8.162672528298433,
'accumulate': 8.706287974887413,
'accumulated': 8.195462351121424,
'accuracy': 7.154008476293262,
'accurate': 6.825975108317913,
'accurately': 7.042782841183038,
'accustomed': 7.306200291635188,
'ace': 8.264455222608374,
'acer': 8.552137295060156,
'aches': 9.031710375322042,
'achievable': 8.041311671294164,
'achieve': 3.7223009080536973,
'achieved': 6.684004847047526,
'achievement': 4.379216301087046,
'achievements': 6.443368138285799,
'achievers': 6.691384954345149,
```

```
'achieves': 8.706287974887413,
'achieving': 5.1717907088781905,
'acid': 8.552137295060156,
'acids': 8.95760240316832,
'acknowledge': 7.883087666079271,
'acknowledged': 8.95760240316832,
'acknowledging': 8.888609531681368,
'acquire': 5.38405363703206,
'acquired': 6.592713426838119,
'acquiring': 6.478665920366823,
'acquisition': 6.376303907705254,
'across': 4.469768923884771,
'acrylic': 7.469525347738487,
'acrylics': 9.198764459985208,
'act': 5.480767607300544,
'acting': 6.6694060476263735,
'action': 5.433344928748937,
'actions': 6.132827927757969,
'activate': 7.34816449073422,
'activated': 8.824071010543797,
'activates': 9.198764459985208,
'activating': 8.552137295060156,
'active': 3.2014082121725838,
'actively': 4.66226252285502,
'actives': 8.888609531681368,
'activism': 8.418605902435633,
'activist': 9.031710375322042,
'activists': 8.377783907915378,
'activites': 8.888609531681368,
'activities': 2.573854484654238,
'activity': 3.661072347194379,
'actor': 9.031710375322042,
'actors': 7.407004990757153,
'acts': 7.74651213107352,
'actual': 5.987187937598619,
'actually': 4.852600546472232,
'ad': 8.505617279425262,
'adapt': 6.283453753741791,
'adaptability': 8.600927459229588,
'adaptable': 8.100152171317099,
'adaptation': 8.95760240316832,
'adaptations': 7.502315170561478,
'adapted': 6.640832675182318,
'adapter': 8.505617279425262,
'adapters': 9.031710375322042,
'adapting': 7.437776649423907,
'adaptive': 6.923830898321673,
'add': 3.9304378071208537,
'added': 5.203068978832947,
'addicted': 8.706287974887413,
'addiction': 8.338563194762097,
'adding': 4.740197748188992,
'addition': 3.4610198294291927,
'additional': 4.160042467549353,
'additionally': 5.007733255250059,
'additions': 7.165842933940265,
'address': 5.705015520957808,
'addressed': 7.085800226266729,
'addresses': 7.57130804204843,
'addressing': 7.469525347738487,
'adds': 6.6062271460048425,
'adept': 8.600927459229588,
'adequate': 6.035978101768051,
'adequately': 7.279171619247268,
'adhd': 4.897159392621901,
'adhere': 8.95760240316832,
'adhesive': 8.888609531681368,
'adjacent': 8.888609531681368,
'adjectives': 7.933098086653932,
'adjust': 6.392653045706783,
'adjustable': 7.279171619247268,
'adjusted': 7.789997243013259,
'adjusting': 7.7680183362944835,
'adjustment': 8.600927459229588,
'adjustments': 8.100152171317099,
'adjusts': 9.198764459985208,
'administered': 8.706287974887413,
```

```
'administration': 6.32880157372025,
'administrative': 8.505617279425262,
'administrator': 8.162672528298433,
'administrators': 6.540413927435271,
'admirable': 8.377783907915378,
'admire': 7.812470098865317,
'admission': 8.706287974887413,
'admissions': 8.706287974887413,
'admit': 8.505617279425262,
'admitted': 8.505617279425262,
'adobe': 8.763446388727363,
'adolescence': 8.552137295060156,
'adolescent': 8.461165516854429,
'adolescents': 7.119322918305372,
'adopt': 8.461165516854429,
'adopted': 6.454996176280919,
'adopting': 8.824071010543797,
'adoption': 8.264455222608374,
'adorable': 7.422272462887942,
'adore': 7.142312436530071,
'adult': 5.308801172621794,
'adulthood': 7.119322918305372,
'adults': 4.592685596060899,
'advance': 5.083429621871272,
'advanced': 5.049874322023054,
'advancement': 7.437776649423907,
'advancements': 8.041311671294164,
'advances': 7.165842933940265,
'advancing': 7.391967113392613,
'advantage': 5.348616858275149,
'advantaged': 7.985741820139354,
'advantages': 6.736847328421907,
'adventure': 5.194742536056393,
'adventures': 5.633080389165133,
'adventurous': 7.319993613767523,
'adverse': 8.377783907915378,
'adversities': 7.907780278669642,
'adversity': 6.21216466999449,
'advertisements': 8.888609531681368,
'advertising': 8.95760240316832,
'advice': 8.100152171317099,
'advise': 8.95760240316832,
'advisor': 8.824071010543797,
'advisory': 8.162672528298433,
'advocacy': 7.789997243013259,
'advocate': 6.760377825832101,
'advocates': 7.536216722237159,
'advocating': 8.461165516854429,
'aerial': 9.111753082995579,
'aerobic': 8.377783907915378,
'aerogarden': 9.111753082995579,
'aesthetic': 8.264455222608374,
'aesthetically': 8.552137295060156,
'affect': 5.6858630887430515,
'affected': 6.6694060476263735,
'affecting': 7.907780278669642,
'affection': 8.652220753617138,
'affectionate': 9.198764459985208,
'affective': 8.763446388727363,
'affects': 6.387173579942157,
'affiliated': 9.031710375322042,
'affirm': 8.888609531681368,
'affirmation': 8.763446388727363,
'affluent': 5.97264222659624,
'afford': 4.44674289665147,
'affordable': 7.536216722237159,
'afforded': 6.706310604561825,
'affording': 8.130923829983853,
'affords': 7.985741820139354,
'afghanistan': 8.338563194762097,
'afloat': 8.95760240316832,
'aforementioned': 8.652220753617138,
'afraid': 6.365551115928992,
'africa': 6.713857810197208,
'african': 4.712173080740795,
'after': 4.499110882168786,
'afternoon': 6.198497031265826,
```

```
'afternoons': 7.933098086653932,
'afterschool': 7.407004990757153,
'afterthought': 8.888609531681368,
'afterward': 9.031710375322042,
'afterwards': 8.162672528298433,
'again': 6.887129531471245,
'against': 8.377783907915378,
'age': 3.7012237258406135,
'aged': 6.515255367799115,
'agencies': 8.763446388727363,
'agency': 8.461165516854429,
'agenda': 8.229363902797104,
'agents': 7.725458721875688,
'ages': 5.183200775616221,
'aggression': 8.706287974887413,
'aggressive': 8.461165516854429,
'agility': 7.553608464949028,
'aging': 8.338563194762097,
'agitated': 8.888609531681368,
'ago': 5.410451108883085,
'agree': 6.825975108317913,
'agreed': 7.279171619247268,
'agreement': 9.031710375322042,
'agricultural': 6.914528505659359,
'agriculture': 6.952268833642206,
'ah': 8.30082286677925,
'aha': 8.418605902435633,
'ahead': 4.897159392621901,
'aid': 5.098377583307145,
'aide': 6.47269575338032,
'aided': 8.377783907915378,
'aides': 7.202210578111139,
'aiding': 8.377783907915378,
'aids': 6.431873758860065,
'aim': 6.013163424001879,
'aimed': 7.858990114500211,
'aiming': 7.907780278669642,
'aims': 7.202210578111139,
'air': 5.471976341889373,
'airplanes': 8.461165516854429,
'airplay': 9.198764459985208,
'airport': 9.198764459985208,
'ais': 8.95760240316832,
'aka': 8.888609531681368,
'al': 8.505617279425262,
'alabama': 6.923830898321673,
'alarm': 8.505617279425262,
'alarming': 8.888609531681368,
'alaska': 7.536216722237159,
'alaskan': 8.706287974887413,
'albert': 6.896179366991162,
'alcohol': 8.01314079432747,
'aleks': 8.888609531681368,
'alert': 7.074871155734538,
'alertness': 8.706287974887413,
'alexa': 8.763446388727363,
'alexander': 8.763446388727363,
'algebra': 6.240073458111566,
'algebraic': 8.652220753617138,
'algeria': 8.95760240316832,
'algorithms': 8.100152171317099,
'alice': 9.031710375322042,
'align': 7.422272462887942,
'aligned': 6.426175737745427,
'aligns': 8.505617279425262,
'alike': 6.851727604420328,
'alive': 5.297710485927636,
'all': 3.0929571315498405,
'allergies': 8.706287974887413,
'alleviate': 6.540413927435271,
'alleviates': 9.198764459985208,
'alleviating': 8.95760240316832,
'alliance': 9.031710375322042,
'alliteration': 8.824071010543797,
'allocated': 8.100152171317099,
'allotted': 7.704839434672952,
'allow': 2.164136270588311,
```

```
'allowance': 9.031710375322042,
'allowed': 4.886136623331151,
'allowing': 3.9877891035923194,
'allows': 3.539599050817814,
'ally': 9.031710375322042,
'alma': 8.95760240316832,
'almost': 4.442044375869916,
'aloha': 8.652220753617138,
'alone': 5.288561291274048,
'along': 3.835766536932555,
'alongs': 8.552137295060156,
'alongside': 6.6062271460048425,
'alot': 8.264455222608374,
'aloud': 5.125860229000977,
'alouds': 6.10359828944303,
'alpha': 9.111753082995579,
'alphabet': 5.503089962738443,
'already': 3.926231210592886,
'also': 1.862182722981733,
'alter': 8.461165516854429,
'altered': 8.95760240316832,
'altering': 9.198764459985208,
'alternate': 6.509063397551195,
'alternative': 4.393998536260341,
'alternatives': 6.809167990001533,
'although': 4.114540809231464,
'alto': 8.552137295060156,
'altogether': 8.338563194762097,
'aluminum': 9.031710375322042,
'alumni': 8.377783907915378,
'always': 2.799867045108916,
'am': 7.626367819231457,
'amaze': 5.913079965444897,
'amazed': 5.853656544974096,
'amazement': 8.95760240316832,
'amazes': 7.165842933940265,
'amazing': 2.9965970635450465,
'amazingly': 6.843069541677214,
'amazon': 6.768345995481277,
'ambassadors': 7.306200291635188,
'ambition': 7.407004990757153,
'ambitions': 8.130923829983853,
'ambitious': 6.360217769953629,
'amenities': 9.111753082995579,
'america': 5.429143241895237,
'american': 4.273489014840109,
'americans': 6.083230986618596,
'amharic': 8.824071010543797,
'among': 4.936084582943892,
'amongst': 6.6694060476263735,
'amount': 4.367545854990277,
'amounts': 6.698819932832667,
'amp': 8.95760240316832,
'ample': 6.426175737745427,
'amplification': 8.888609531681368,
'amplified': 9.198764459985208,
'amplifier': 9.111753082995579,
'amplify': 8.652220753617138,
'amusing': 8.600927459229588,
'an': 5.035629066887005,
'analysis': 6.020710629637263,
'analytical': 7.227211880316557,
'analyze': 5.400215604989058,
'analyzed': 8.763446388727363,
'analyzing': 6.572779211937302,
'anatomy': 7.362553228186319,
'ancestors': 8.652220753617138,
'anchor': 6.128599591648447,
'anchorage': 8.763446388727363,
'anchors': 8.418605902435633,
'ancient': 6.760377825832101,
'and': 4.4617688812176945,
'anderson': 8.505617279425262,
'android': 8.377783907915378,
'andy': 8.418605902435633,
'angeles': 5.976258867066428,
'angelou': 8.338563194762097,
```

```
'angels': 8.763446388727363,
'anger': 6.869271914071238,
'angle': 7.907780278669642,
'angles': 7.377152027607472,
'anglo': 9.111753082995579,
'angry': 7.933098086653932,
'animal': 5.892877258127378,
'animals': 5.414574826066947,
'animate': 8.100152171317099,
'animated': 7.858990114500211,
'animation': 7.064060239630322,
'animations': 7.704839434672952,
'ann': 9.111753082995579,
'anne': 8.706287974887413,
'anniversary': 8.461165516854429,
'annotate': 6.905311850554435,
'annotating': 7.907780278669642,
'annotation': 8.888609531681368,
'annotations': 9.198764459985208,
'announce': 8.706287974887413,
'announcement': 8.229363902797104,
'announcements': 7.279171619247268,
'annual': 6.851727604420328,
'annually': 8.100152171317099,
'anonymous': 8.229363902797104,
'another': 3.381923899401276,
'ans': 9.198764459985208,
'answer': 4.973923408833739,
'answered': 7.553608464949028,
'answering': 7.053364950513575,
'answers': 5.21484371537748,
'ant': 8.888609531681368,
'anti': 9.111753082995579,
'anticipate': 7.684636727355432,
'anticipated': 8.162672528298433,
'anticipating': 8.195462351121424,
'anticipation': 7.812470098865317,
'antiquated': 8.100152171317099,
'antonio': 8.418605902435633,
'ants': 8.041311671294164,
'antsy': 7.835459617090017,
'anxieties': 8.377783907915378,
'anxiety': 5.620308823485645,
'anxious': 6.154242022261785,
'anxiously': 7.858990114500211,
'any': 5.643416398495794,
'anybody': 8.552137295060156,
'anymore': 6.640832675182318,
'anyone': 5.730191675850282,
'anything': 4.1582762027392715,
'anytime': 6.971686919499308,
'anyway': 7.959073573057193,
'anywhere': 5.568381212552881,
'ap': 5.916487123766512,
'apart': 5.390083806058651,
'apartment': 7.108023363051439,
'apartments': 6.640832675182318,
'apathy': 8.824071010543797,
'app': 5.46109484170184,
'appalachia': 8.652220753617138,
'appalachian': 7.704839434672952,
'apparatus': 8.95760240316832,
'apparel': 9.031710375322042,
'apparent': 6.9426993826260555,
'appeal': 6.527755530563347,
'appealing': 6.466761017860505,
'appeals': 8.100152171317099,
'appear': 7.142312436530071,
'appearance': 7.907780278669642,
'appears': 7.933098086653932,
'appetite': 7.933098086653932,
'apple': 5.675081313139763,
'apples': 8.461165516854429,
'applesauce': 8.229363902797104,
'appliances': 8.461165516854429,
'applicable': 7.2659263924972475,
'applicants': 9.111753082995579,
```

```
'application': 5.667070589393684,
'applications': 5.125860229000977,
'applied': 6.409273926942824,
'applies': 7.34816449073422,
'apply': 4.617514457714769,
'applying': 6.1243890591121035,
'appreciate': 4.61844599015288,
'appreciated': 5.266938827260883,
'appreciates': 8.552137295060156,
'appreciating': 8.100152171317099,
'appreciation': 5.919905930515297,
'appreciative': 6.0019483531817395,
'apprehensive': 8.95760240316832,
'approach': 4.928431484366175,
'approachable': 8.652220753617138,
'approached': 8.461165516854429,
'approaches': 6.800869187186837,
'approaching': 7.252854310929894,
'appropriate': 4.234014462365755,
'appropriately': 6.254325480818768,
'approval': 8.763446388727363,
'approve': 8.763446388727363,
'approved': 7.485785868610267,
'approximately': 4.726606320985454,
'apps': 4.066179737538042,
'april': 7.57130804204843,
'apron': 8.763446388727363,
'aprons': 7.959073573057193,
'aps': 8.763446388727363,
'apt': 7.985741820139354,
'aptitude': 8.763446388727363,
'aquaponics': 8.95760240316832,
'aquarium': 7.959073573057193,
'aquariums': 9.111753082995579,
'aquatic': 8.461165516854429,
'ar': 6.684004847047526,
'arabia': 9.111753082995579,
'arabic': 6.559707130369949,
'archery': 9.111753082995579,
'architect': 8.338563194762097,
'architects': 7.227211880316557,
'architectural': 8.824071010543797,
'architecture': 7.607675686219304,
'arctic': 8.763446388727363,
'arduino': 8.01314079432747,
'arduinos': 9.198764459985208,
'arduous': 8.888609531681368,
'are': 5.456775180557323,
'area': 2.8946481043747996,
'areas': 3.488638511406832,
'arena': 8.824071010543797,
'arguably': 8.706287974887413,
'argue': 7.519122288877859,
'arguing': 8.763446388727363,
'argument': 7.7680183362944835,
'argumentative': 8.763446388727363,
'arguments': 7.883087666079271,
'arise': 7.032311541315742,
'arises': 8.338563194762097,
'aristotle': 8.652220753617138,
'arithmetic': 7.883087666079271,
'arizona': 6.721462409582427,
'arkansas': 7.032311541315742,
'arledge': 8.888609531681368,
'arm': 7.704839434672952,
'armed': 8.763446388727363,
'arms': 7.021948754280196,
'army': 8.30082286677925,
'around': 2.726699222259626,
'arrange': 7.607675686219304,
'arranged': 7.858990114500211,
'arrangement': 6.515255367799115,
'arrangements': 6.691384954345149,
'arranging': 9.111753082995579,
'array': 6.0751988149213325,
'arrays': 8.763446388727363,
'arrival': 8.461165516854429,
```

```
'arrival': 8.401105910054429,
'arrivals': 8.377783907915378,
'arrive': 5.290384445835563,
'arrived': 6.896179366991162,
'arrives': 8.600927459229588,
'arriving': 7.407004990757153,
'art': 3.5176614740984244,
'artful': 9.198764459985208,
'arthur': 8.461165516854429,
'article': 6.553234615864332,
'articles': 5.523615198683173,
'articulate': 8.01314079432747,
'articulation': 7.6454160142021514,
'artifacts': 7.664834100059253,
'artificial': 9.111753082995579,
'artist': 6.013163424001879,
'artistic': 5.390083806058651,
'artistically': 7.6454160142021514,
'artists': 5.093869686867955,
'arts': 3.7670293436333857,
'artwork': 5.831468629998734,
'artworks': 7.835459617090017,
'as': 2.658259326310569,
'asd': 7.6454160142021514,
'asia': 7.553608464949028,
'asian': 5.812834550453841,
'aside': 6.47269575338032,
'ask': 3.9982604034596148,
'asked': 4.241018629809325,
'asking': 3.907746395918783,
'asks': 7.664834100059253,
'asl': 8.418605902435633,
'asleep': 8.95760240316832,
'aspect': 5.578066518287344,
'aspects': 5.318138308617734,
'asperger': 8.706287974887413,
'aspiration': 8.95760240316832,
'aspirations': 7.154008476293262,
'aspire': 6.760377825832101,
'aspiring': 7.292594639579409,
'assemble': 7.502315170561478,
'assembled': 8.763446388727363,
'assemblies': 7.664834100059253,
'assembling': 8.763446388727363,
'assembly': 8.01314079432747,
'asses': 9.031710375322042,
'assess': 6.2883920353823735,
'assessed': 7.725458721875688,
'assesses': 9.031710375322042,
'assessing': 8.01314079432747,
'assessment': 5.837757958906298,
'assessments': 5.362249007065207,
'asset': 6.083230986618596,
'assets': 8.264455222608374,
'assign': 6.721462409582427,
'assigned': 5.856866820604345,
'assigning': 8.706287974887413,
'assignment': 5.733028557185482,
'assignments': 4.356727141463109,
'assimilate': 8.888609531681368,
'assist': 4.455414610433087,
'assistance': 5.121227016079489,
'assistant': 7.704839434672952,
'assistants': 8.30082286677925,
'assisted': 7.835459617090017,
'assisting': 7.085800226266729,
'assistive': 7.607675686219304,
'assists': 7.858990114500211,
'associate': 7.469525347738487,
'associated': 6.633815102523672,
'associating': 9.111753082995579,
'association': 7.279171619247268,
'assorted': 8.600927459229588,
'assortment': 7.032311541315742,
'assume': 7.835459617090017,
'assumed': 9.111753082995579,
'assure': 7.142312436530071,
'assured': 8.264455222608374,
```

```
'assured': 8.264455222608374,
'assuredly': 9.111753082995579,
'asthma': 8.461165516854429,
'astonished': 8.95760240316832,
'astonishing': 9.031710375322042,
'astound': 8.888609531681368,
'astounded': 8.95760240316832,
'astounding': 8.162672528298433,
'astronaut': 8.706287974887413,
'astronauts': 7.933098086653932,
'astronomy': 8.338563194762097,
'asus': 8.888609531681368,
'at': 3.9187962321053678,
'ate': 7.835459617090017,
'athlete': 7.57130804204843,
'athletes': 5.716126746382878,
'athletic': 6.47269575338032,
'athletically': 8.888609531681368,
'athletics': 6.933220638671512,
'atlanta': 6.825975108317913,
'atlantic': 9.198764459985208,
'atlas': 8.95760240316832,
'atlases': 8.888609531681368,
'atmosphere': 5.1799274502712525,
'atpe': 7.985741820139354,
'attach': 7.34816449073422,
'attached': 6.834485797985822,
'attaching': 8.505617279425262,
'attachment': 9.111753082995579,
'attack': 8.162672528298433,
'attain': 6.760377825832101,
'attainable': 7.589326547551107,
'attained': 8.377783907915378,
'attaining': 8.195462351121424,
'attainment': 9.198764459985208,
'attempt': 6.484671944427035,
'attempted': 9.198764459985208,
'attempting': 7.108023363051439,
'attempts': 8.041311671294164,
'attend': 3.750460917286153,
'attendance': 5.812834550453841,
'attended': 6.149922361117269,
'attending': 5.4207924626778174,
'attends': 8.377783907915378,
'attention': 3.744998554894313,
'attentive': 6.640832675182318,
'attentiveness': 8.130923829983853,
'attire': 8.377783907915378,
'attitude': 5.274094492856294,
'attitudes': 5.8346083500003402,
'attract': 7.519122288877859,
'attracted': 8.824071010543797,
'attractive': 7.536216722237159,
'attracts': 8.30082286677925,
'attribute': 8.461165516854429,
'attributes': 7.485785868610267,
'atypical': 8.888609531681368,
'audible': 8.264455222608374,
'audience': 6.303354908059086,
'audiences': 7.725458721875688,
'audio': 5.4354524110885025,
'audiobook': 8.552137295060156,
'audiobooks': 7.536216722237159,
'audition': 8.130923829983853,
'auditorily': 8.95760240316832,
'auditorium': 7.959073573057193,
'auditory': 6.141338617425877,
'augment': 9.111753082995579,
'augmentative': 8.505617279425262,
'august': 6.1673141038291375,
'aunt': 8.418605902435633,
'aunts': 8.070299208167416,
'aural': 8.763446388727363,
'austin': 8.505617279425262,
'authentic': 5.348616858275149,
'authentically': 8.95760240316832,
'author': 5.710555701333423,
'authority': 9.031710375322042,
```

```
'authority': 9.031710375322042,
'authorized': 9.031710375322042,
'authors': 5.350552967302016,
'autism': 4.255175880420418,
'autistic': 6.216762379243119,
'auto': 8.95760240316832,
'autobiography': 8.763446388727363,
'automatic': 8.552137295060156,
'automatically': 7.469525347738487,
'automaticity': 8.505617279425262,
'autonomous': 8.763446388727363,
'autonomy': 7.292594639579409,
'av': 9.111753082995579,
'availability': 6.662185799652887,
'available': 3.469255170090498,
'avenue': 6.490714258882998,
'avenues': 6.768345995481277,
'average': 5.281301731261244,
'averages': 8.229363902797104,
'avid': 6.024505700605814,
'avoid': 6.540413927435271,
'avoided': 9.111753082995579,
'avoiding': 8.763446388727363,
'await': 7.57130804204843,
'awaiting': 8.338563194762097,
'awaits': 7.7680183362944835,
'awake': 7.607675686219304,
'awaken': 7.907780278669642,
'award': 6.162937729229339,
'awarded': 7.453525006392046,
'awards': 6.834485797985822,
'aware': 5.491866500368594,
'awareness': 5.204742619390976,
'away': 4.2276893305887855,
'awe': 6.905311850554435,
'awesome': 4.4029739143884665,
'awesomeness': 8.824071010543797,
'awful': 8.552137295060156,
'awhile': 8.30082286677925,
'awkward': 8.130923829983853,
'az': 9.111753082995579,
'babe': 9.198764459985208,
'babies': 6.502909531976816,
'baby': 6.706310604561825,
'babysitting': 8.706287974887413,
'baccalaureate': 6.540413927435271,
'back': 3.7535957160915245,
'backbone': 8.418605902435633,
'backdrop': 9.111753082995579,
'backdrops': 8.95760240316832,
'backed': 9.111753082995579,
'background': 4.187523288863451,
'backgrounds': 2.796696513276581,
'backing': 9.198764459985208,
'backpack': 5.318138308617734,
'backpacks': 5.8283387369898065,
'backpatter': 8.706287974887413,
'backs': 6.869271914071238,
'backup': 9.198764459985208,
'backwards': 8.01314079432747,
'backyard': 8.195462351121424,
'backyards': 8.763446388727363,
'bacteria': 8.229363902797104,
'bad': 5.951212835140341,
'badge': 9.198764459985208,
'badges': 8.418605902435633,
'badly': 7.835459617090017,
'badminton': 8.229363902797104,
'bag': 5.1799274502712525,
'baggage': 7.485785868610267,
'baggies': 8.505617279425262,
'bags': 5.071630074940116,
'bake': 8.01314079432747,
'baked': 8.763446388727363,
'baker': 8.95760240316832,
'baking': 8.100152171317099,
'balance': 4.580947312296349,
'balanced': 6.250121653082261,
```

```
 'balanced': 6.259121653082261,
 'balances': 8.505617279425262,
 'balancing': 6.706310604561825,
 'ball': 4.582744257973051,
 'ballet': 8.01314079432747,
 'balloon': 8.162672528298433,
 'balloons': 8.30082286677925,
 'balls': 4.087872562149302,
 'baltimore': 7.064060239630322,
 'banana': 8.824071010543797,
 'bananas': 9.198764459985208,
 'band': 5.382051634361387,
 'bands': 5.1653286508501,
 'bang': 8.418605902435633,
 'bangladesh': 8.600927459229588,
 'bank': 7.108023363051439,
 ...}
```

In [72]:

```
for k in list(combine_dict):
    if combine_dict[k] <= 3 or combine_dict[k] >= 7:
        combine_dict.pop(k,None)
```

In [73]:

```
print("We hvae {} number of features after preprocesing..".format(len(combine_dict)))
```

```
We hvae 3729 number of features after preprocesing..
```

In [74]:

```
# sorting combine_dict according to k:v  where key is feature name and value is index given
combine_dict = dict(sorted(combine_dict.items(), key=lambda kv: kv[1],reverse = True))
#combine_dict_n = dict()

# assigning integers to keys as per their idf_ values
n = 1
for k in list(combine_dict):
    combine_dict[k] = n
    n += 1
```

In [75]:

```
combine_dict["start_seq"] = 0
```

In [76]:

```
# from sklearn.feature_extraction.text import CountVectorizer
# cnt  = CountVectorizer(vocabulary=combine_dict_n)

# combine_idf = cnt.fit_transform(preprocessed_combine)
# test_combine_idf = cnt.transform(test_preprocessed_combine)

# print(combine_idf.shape)
# print(test_combine_idf.shape)
```

In [77]:

```
len(combine_dict)
```

Out[77]:

```
3730
```

In [78]:

```
combine_sequence_idf = []
```

```
for el in preprocessed_combine:
    seq = [combine_dict[word] for word in el.split(' ') if word in combine_dict]
    combine_sequence_idf.append(seq)


# for val data

val_combine_sequence_idf = []

for el in val_preprocessed_combine:
    seq = [combine_dict[word] for word in el.split(' ') if word in combine_dict]
    val_combine_sequence_idf.append(seq)


# for test data

test_combine_sequence_idf = []

for el in test_preprocessed_combine:
    seq = [combine_dict[word] for word in el.split(' ') if word in combine_dict]
    test_combine_sequence_idf.append(seq)
```

- Max_len = 206

In [79]:

```
Max_len = 206

combine_sequence = pad_sequences(combine_sequence_idf,maxlen=Max_len,padding="post")

# for test and val data

val_combine_sequence = pad_sequences(val_combine_sequence_idf,maxlen=Max_len,padding="post")
test_combine_sequence = pad_sequences(test_combine_sequence_idf,maxlen=Max_len,padding="post")
```

In [80]:

```
print("shape of combine sequence idf {}".format(combine_sequence.shape))
print("shape of tval combine sequence idf {}".format(val_combine_sequence.shape))
print("shape of test combine sequence idf {}".format(test_combine_sequence.shape))
print("Maximum words are = {}".format(combine_sequence.shape[1]))
```

```
shape of combine sequence idf (40000, 206)
shape of tval combine sequence idf (10000, 206)
shape of test combine sequence idf (10000, 206)
Maximum words are = 206
```

In [81]:

```
MAX_LENGTH = Max_len
MAX_WORDS = len(combine_dict)

print(MAX_LENGTH)
print(MAX_WORDS)
```

```
206
3730
```

- MAX_WORDS = 3731
- MAX_LENGTH = 206

1. Preparing essay for model2
2. using keras tokenizer

In [82]:

```
word_to_ix = combine_dict
ix_to_word = dict()
```

In [83]:

```
for k,v in word_to_ix.items():
    ix_to_word[v] = k

print(len(word_to_ix))
print(len(ix_to_word))
```

```
3730
3730
```

In [84]:

```
MAX_LENGTH = combine_sequence.shape[1]
print("Maximum sequence length is {}".format(MAX_LENGTH))
print(combine_sequence.shape)
```

```
Maximum sequence length is 206
(40000, 206)
```

In [85]:

```
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    glove = load(f)
    glove_words =  set(glove.keys())


EMBEDDING_SIZE = 300
VOCAB_SIZE = MAX_WORDS

# Get 300-dim dense vector for each of the words in vocabulary
embedding_matrix_2 = np.zeros((VOCAB_SIZE,EMBEDDING_SIZE))
embedding_matrix_2.shape
```

Out[85]:

```
(3730, 300)
```

In [86]:

```
# code for embedding matrix. considering top 5000 words and using already present glove vectors

# Get 300-dim dense vector for each of the words in vocabulary
embedding_matrix_2 = np.zeros(((VOCAB_SIZE),EMBEDDING_SIZE))

for word, i in word_to_ix.items():
    embedding_vector = np.zeros(300)
    if word in glove_words:
        embedding_vector = glove[word]
        embedding_matrix_2[i] = embedding_vector
    else:
        # Words not found in the embedding index will be all zeros
        embedding_matrix_2[i] = embedding_vector

print("Shape of embedding matrix {}".format(embedding_matrix_2.shape))
```

```
Shape of embedding matrix (3730, 300)
```

Model Preperation

In [87]:

```
# save the embedding matrix to file
with open("embedding_matrix_2_idf.pkl","wb") as f:
    dump(embedding_matrix_2,f)

# functional api for combine
```

```
LSTM_UINTS = 8

input_ess = Input(shape=(MAX_LENGTH,))
em1 = Embedding(MAX_WORDS,EMBEDDING_SIZE,input_length=MAX_LENGTH)(input_ess)
lstm = LSTM(LSTM_UINTS,input_shape = (1,MAX_LENGTH), return_sequences=True,)(em1)
flt_ess  = Flatten()(lstm)
```

In [88]:

```
cnt = concatenate([flt_ess,flt_ss,flt_gc,flt_tp,flt_pc,flt_psc,d_nf])

dense = Dense(8,activation="relu")(cnt)

# dropout + dense
dp = Dropout(0.2)(dense)
dense2 = Dense(4,activation="relu")(dp)

# dropout + dense

dp = Dropout(0.2)(dense2)
dense3 = Dense(2,activation="relu")(dp)

output = Dense(1,activation="sigmoid")(dense3)
```

In [89]:

```
# input sequence essay + school_state + grade + teacher_prefix + category + sub_categpry + numeric
al
model = Model(inputs =[input_ess,input_ss,input_gc,input_tp,input_pc,input_psc,input_nf],outputs =
output)
```

In [90]:

```
# Freezing essay embedding_layer from training

model.layers[1].set_weights([embedding_matrix_2])
model.layers[1].trainable = False
```

In [91]:

```
model.summary()
```

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_8 (InputLayer)         (None, 206)               0
_________________________________________________________________________________________________
embedding_7 (Embedding)      (None, 206, 300)          1119000     input_8[0][0]
_________________________________________________________________________________________________
input_6 (InputLayer)         (None, 1)                 0
_________________________________________________________________________________________________
input_5 (InputLayer)         (None, 1)                 0
_________________________________________________________________________________________________
input_4 (InputLayer)         (None, 1)                 0
_________________________________________________________________________________________________
input_2 (InputLayer)         (None, 3)                 0
_________________________________________________________________________________________________
input_3 (InputLayer)         (None, 3)                 0
_________________________________________________________________________________________________
lstm_2 (LSTM)                (None, 206, 8)            9888        embedding_7[0][0]
_________________________________________________________________________________________________
embedding_6 (Embedding)      (None, 1, 10)             520         input_6[0][0]
_________________________________________________________________________________________________
embedding_5 (Embedding)      (None, 1, 10)             50          input_5[0][0]
_________________________________________________________________________________________________
embedding_4 (Embedding)      (None, 1, 10)             60          input_4[0][0]
_________________________________________________________________________________________________
embedding_2 (Embedding)      (None, 3, 10)             100         input_2[0][0]
_________________________________________________________________________________________________
embedding_3 (Embedding)      (None, 3, 10)             310         input_3[0][0]
_________________________________________________________________________________________________
input_7 (InputLayer)         (None, 3)                 0
```

flatten_7 (Flatten)	(None, 1648)	0	lstm_2[0][0]
flatten_6 (Flatten)	(None, 10)	0	embedding_6[0][0]
flatten_5 (Flatten)	(None, 10)	0	embedding_5[0][0]
flatten_4 (Flatten)	(None, 10)	0	embedding_4[0][0]
flatten_2 (Flatten)	(None, 30)	0	embedding_2[0][0]
flatten_3 (Flatten)	(None, 30)	0	embedding_3[0][0]
dense_1 (Dense)	(None, 1)	4	input_7[0][0]
concatenate_2 (Concatenate)	(None, 1739)	0	flatten_7[0][0] flatten_6[0][0] flatten_5[0][0] flatten_4[0][0] flatten_2[0][0] flatten_3[0][0] dense_1[0][0]
dense_6 (Dense)	(None, 8)	13920	concatenate_2[0][0]
dropout_3 (Dropout)	(None, 8)	0	dense_6[0][0]
dense_7 (Dense)	(None, 4)	36	dropout_3[0][0]
dropout_4 (Dropout)	(None, 4)	0	dense_7[0][0]
dense_8 (Dense)	(None, 2)	10	dropout_4[0][0]
dense_9 (Dense)	(None, 1)	3	dense_8[0][0]

Total params: 1,143,901
Trainable params: 24,901
Non-trainable params: 1,119,000

In [92]:

```
# model compilation
model.compile(loss='binary_crossentropy', optimizer='adam',metrics=[auroc])

# using tensorboard instance for callbacks
tensorboard = TensorBoard(log_dir="model2_logs_2/{}".format(time()))
```

In [94]:

```
# callbacks
import keras
filepath = "weights_2.{epoch:02d}-{val_loss:.2f}.hdf5"
history_2 = keras.callbacks.History()
model_check_2 = keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_
only=True, save_weights_only=False, mode='auto', period=1)

early_2 = keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=0, verbose=1, mod
e='auto', baseline=None, restore_best_weights=True)
```

In [95]:

```
epochs = 10
batch_size = 64
val_batch_size = 32
steps = len(y_train)//batch_size
val_steps = len(y_val)//val_batch_size
```

In [96]:

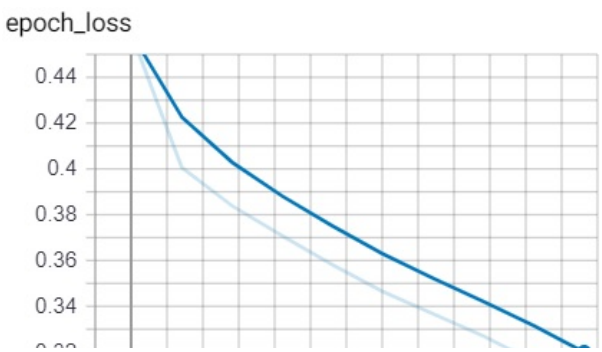
```
for i in range(epochs):
    print("Epoch {} start at time ".format(i),datetime.now())
    generator = data_generator(y_train,batch_size)
```

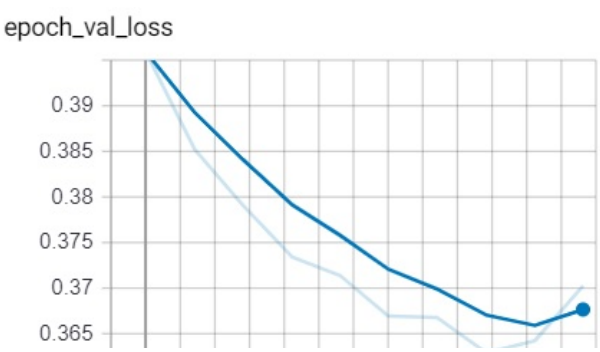
```
    generator = data_generator(y_train,batch_size)
    val_generator = data_generator(y_val,batch_size,"Val")
    model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=2,callbacks=[tensorboar
d,history_2,model_check_2,early_2],validation_data=val_generator,validation_steps=val_steps)
    model.save_weights("model_2_epoch_{}.h5".format(i))
```

```
Epoch 0 start at time  2019-08-05 22:58:30.762575
Epoch 1/1
 - 102s - loss: 0.4593 - auroc: 0.5834 - val_loss: 0.3960 - val_auroc: 0.7169

Epoch 00001: val_loss improved from inf to 0.39596, saving model to weights_2.01-0.40.hdf5
Epoch 1 start at time  2019-08-05 23:00:15.385022
Epoch 1/1
 - 100s - loss: 0.4005 - auroc: 0.6839 - val_loss: 0.3852 - val_auroc: 0.7412

Epoch 00001: val_loss improved from 0.39596 to 0.38522, saving model to weights_2.01-0.39.hdf5
Epoch 2 start at time  2019-08-05 23:01:56.514355
Epoch 1/1
 - 100s - loss: 0.3838 - auroc: 0.7279 - val_loss: 0.3790 - val_auroc: 0.7545

Epoch 00001: val_loss improved from 0.38522 to 0.37904, saving model to weights_2.01-0.38.hdf5
Epoch 3 start at time  2019-08-05 23:03:37.816354
Epoch 1/1
 - 100s - loss: 0.3706 - auroc: 0.7549 - val_loss: 0.3734 - val_auroc: 0.7598

Epoch 00001: val_loss improved from 0.37904 to 0.37341, saving model to weights_2.01-0.37.hdf5
Epoch 4 start at time  2019-08-05 23:05:18.603354
Epoch 1/1
 - 99s - loss: 0.3581 - auroc: 0.7788 - val_loss: 0.3714 - val_auroc: 0.7680

Epoch 00001: val_loss improved from 0.37341 to 0.37138, saving model to weights_2.01-0.37.hdf5
Epoch 5 start at time  2019-08-05 23:06:58.833874
Epoch 1/1
 - 99s - loss: 0.3464 - auroc: 0.7988 - val_loss: 0.3669 - val_auroc: 0.7726

Epoch 00001: val_loss improved from 0.37138 to 0.36694, saving model to weights_2.01-0.37.hdf5
Epoch 6 start at time  2019-08-05 23:08:39.064763
Epoch 1/1
 - 100s - loss: 0.3368 - auroc: 0.8096 - val_loss: 0.3668 - val_auroc: 0.7776

Epoch 00001: val_loss improved from 0.36694 to 0.36678, saving model to weights_2.01-0.37.hdf5
Epoch 7 start at time  2019-08-05 23:10:20.190019
Epoch 1/1
 - 102s - loss: 0.3269 - auroc: 0.8233 - val_loss: 0.3629 - val_auroc: 0.7802

Epoch 00001: val_loss improved from 0.36678 to 0.36288, saving model to weights_2.01-0.36.hdf5
Epoch 8 start at time  2019-08-05 23:12:02.858536
Epoch 1/1
 - 100s - loss: 0.3158 - auroc: 0.8363 - val_loss: 0.3642 - val_auroc: 0.7812

Epoch 00001: val_loss did not improve from 0.36288
Epoch 9 start at time  2019-08-05 23:13:43.744050
Epoch 1/1
 - 99s - loss: 0.3032 - auroc: 0.8499 - val_loss: 0.3703 - val_auroc: 0.7792

Epoch 00001: val_loss did not improve from 0.36288
```

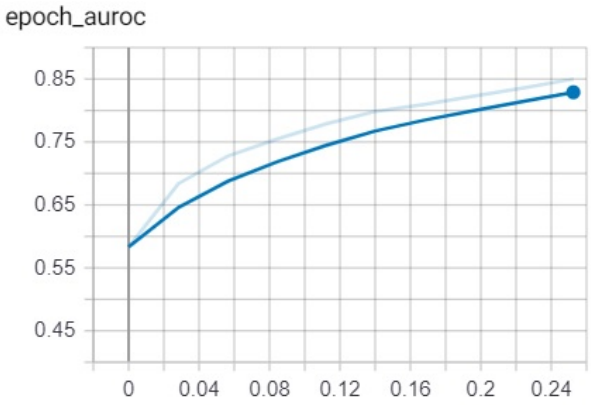
Loss and AUC

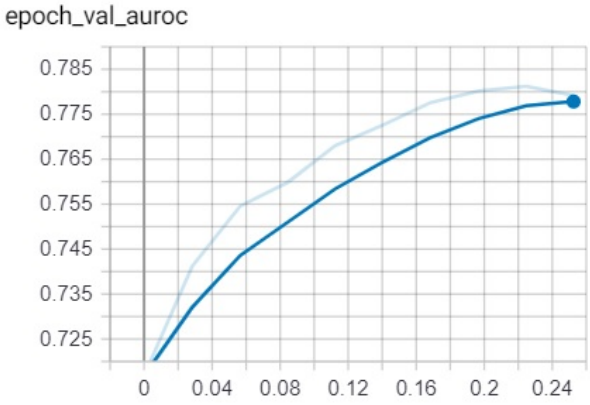
1. TensorBoard Epoch Loss And Validation Loss





1. TensorBoard Epoch AUC and Validation AUC





Model Prediction

In [97]:

```
test_set =
[test_combine_sequence,test_state_sequence,test_grade_sequence,test_prefix_sequence,test_category_s
equence,test_subcategory_sequence,test_numerical]
#generator = data_generator(y_test,1,1,batch_size)
history = model.predict(test_set)
```

AUC Score

In [98]:

```
from sklearn.metrics import roc_auc_score

# AUC for test data
print("AUC score is {}".format(roc_auc_score(y_test,history)))
```

```
AUC score is 0.770582115893506
```

ConFusion Matrix

In [101]:

```
# converting probabilistic values to class label. Threshold = 0.5
y_pred = (history > 0.5).astype(np.int)
```

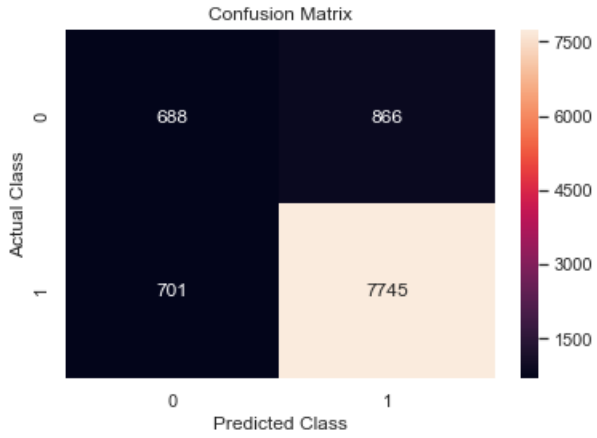
In [102]:

```
from sklearn.metrics import confusion_matrix
cm1 = confusion_matrix(y_test,y_pred)
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.heatmap(cm1, annot=True, fmt="d")
plt.ylabel("Actual Class")
plt.xlabel("Predicted Class")
plt.title("Confusion Matrix")
```

Out[102]:

```
Text(0.5, 1.0, 'Confusion Matrix')
```

```
text(0.5, 1.0, 'Confusion Matrix')
```



Observation

==========

- At start faced the problem of overfitting. Overfitting was avoided by reducing number of parameters.
- Observed **first dicrease and then increase in Validation loss as model complexity was increased.**
- While Training on more epoch's , the same above phenomenon was observed.
- Tried and tested **various model configurations to get best result.**
- Best AUC results were obtained in case of model 3 which is **0.794**
- parameters value are modified as per convinience
- Training model 3 took most of the time

Summary

In [1]:

```
from prettytable import PrettyTable
summary = PrettyTable()
```

In [2]:

```
summary.field_names = ["Model","Total params","Trainable params","Non-trainable params","Train AUC","Test AUC"]
```

In [3]:

```
summary.add_row(["Model1","11,471,429","33,157","11,454,000","0.8066"," 0.7073"])
summary.add_row(["Model2 TFIDF","1,143,901","24,901","1,119,000","0.8233","0.77"])
summary.add_row(["Model3","1,563,629","63,629","15,00,000","0.804","0.79"])
```

In [4]:

```
print(summary)
```

```
+--------------+--------------+------------------+----------------------+-----------+----------+
|    Model     | Total params | Trainable params | Non-trainable params | Train AUC | Test AUC |
+--------------+--------------+------------------+----------------------+-----------+----------+
|    Model1    |  11,471,429  |      33,157      |      11,454,000      |   0.8066  |  0.7073  |
| Model2 TFIDF |  1,143,901   |      24,901      |      1,119,000       |   0.8233  |   0.77   |
|    Model3    |  1,563,629   |      63,629      |      15,00,000       |   0.804   |   0.79   |
+--------------+--------------+------------------+----------------------+-----------+----------+
```

In [ ]:

```

```