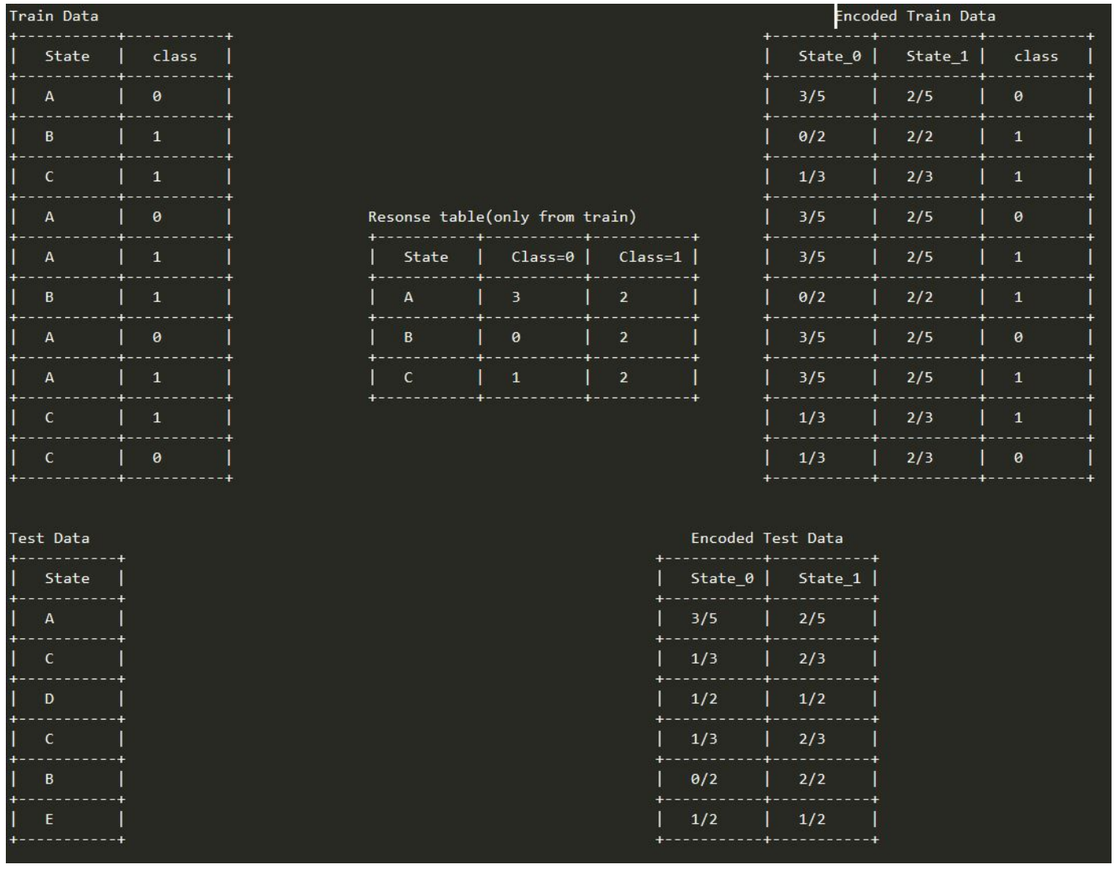
Assignment 9: GBDT

Response Coding: Example



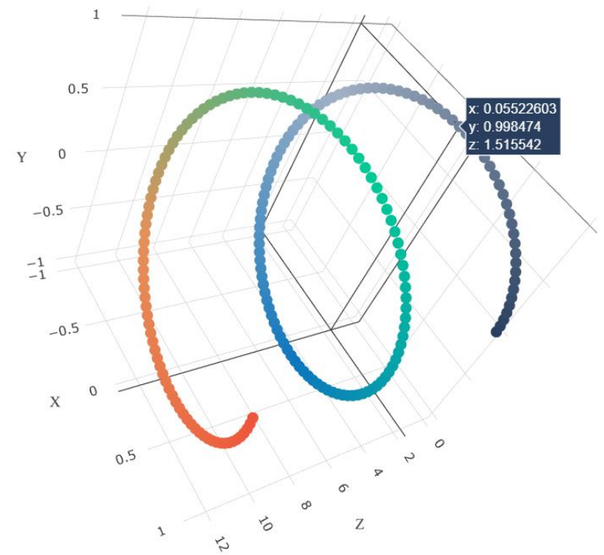
> The response tabel is built only on train dataset. For a category which is not there in train data and present in test data, we will encode them with default values Ex: in our test data if have State: D then we encode it as [0.5, 0.05]

1. **Apply GBDT on these feature sets**
   - Set 1: categorical(instead of one hot encoding, try response coding (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check the bellow example, include all 4 values as 4 features)
   - Set 2: categorical(instead of one hot encoding, try response coding (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (Consider any two hyper parameters)**
   - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value

- find the best hyper paramter using k-fold cross validation/simple cross validation data
- use gridsearch cv or randomsearch cv or you can write your own for loops to do this task

3. **Representation of results**

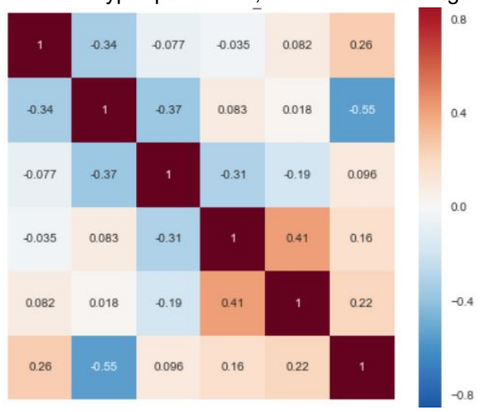
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

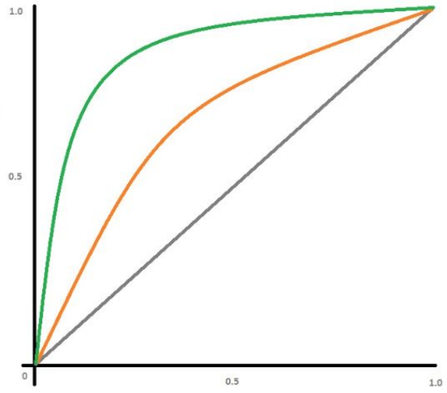
or

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



seaborn heat maps (https://seaborn.pydata.org/generated/seaborn.heatmap.html) with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**

- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.



- Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

4. You need to summarize the results at the end of the notebook, summarize it in the table format

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

In [192]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

import nltk
nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students with the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelligences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of different backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a caring community of successful \
learners which can be seen through collaborative student project based learning in and out of the classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities to practice a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with real food i will take their idea \
and create common core cooking lessons where we learn important math and writing concepts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cookbooks to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoyment for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')


# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\User\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!

neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

In [193]:
```
ss
```

Out[193]:
```
{'neg': 0.01, 'neu': 0.745, 'pos': 0.245, 'compound': 0.9975}
```

1. GBDT (xgboost/lightgbm)

1.1 Loading Data

In [194]:
```
#Required Packages
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve,accuracy_score
from sklearn.metrics import auc as AUC_score
from tqdm import tqdm
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from scipy.sparse import hstack,csr_matrix
import pickle
```

In [195]:
```
#Reading the Input file
data = pd.read_csv('preprocessed_data.csv')
```

In [196]:
```
Neg_sentiment,Neu_sentiment,Pos_sentiment,Compound_val_sentiment =[],[],[],[]


for sen in tqdm(range(len(data['essay']))):
    for_sentiment = data['essay'][sen]
    ss = sid.polarity_scores(for_sentiment)
    Neg_sentiment.append(ss['neg'])
    Neu_sentiment.append(ss['neu'])
    Pos_sentiment.append(ss['pos'])
    Compound_val_sentiment.append(ss['compound'])
```

```
  0%|                                                                                | 0/109248 [00:00<?, ?it/s]
  0%|                                                                              | 16/109248 [00:00<11:35, 156.96it/s]
  0%|                                                                              | 33/109248 [00:00<11:20, 160.57it/s]
  0%|                                                                              | 52/109248 [00:00<10:50, 167.98it/s]
  0%|                                                                              | 69/109248 [00:00<10:53, 166.95it/s]
  0%|                                                                              | 87/109248 [00:00<10:39, 170.57it/s]
  0%|                                                                             | 104/109248 [00:00<10:44, 169.34it/s]
  0%|                                                                             | 121/109248 [00:00<10:48, 168.22it/s]
  0%|                                                                             | 140/109248 [00:00<10:32, 172.58it/s]
  0%|                                                                             | 160/109248 [00:00<10:09, 179.03it/s]
  0%|▏                                                                            | 182/109248 [00:01<09:35, 189.53it/s]
  0%|▏                                                                            | 201/109248 [00:01<09:38, 188.65it/s]
  0%|▏                                                                            | 220/109248 [00:01<10:01, 181.28it/s]
  0%|▏                                                                            | 240/109248 [00:01<09:44, 186.45it/s]
  0%|▏                                                                            | 259/109248 [00:01<10:10, 178.51it/s]
  0%|▏                                                                            | 278/109248 [00:01<10:02, 180.74it/s]
  0%|▏                                                                            | 297/109248 [00:01<10:41, 169.76it/s]
  0%|▏                                                                            | 317/109248 [00:01<10:17, 176.45it/s]
```

In [197]:
```
data['Neg_sentiment'] = Neg_sentiment
data['Neu_sentiment'] = Neu_sentiment
data['Pos_sentiment'] = Pos_sentiment
data['Compound_val_sentiment'] = Compound_val_sentiment
```

In [198]:
```
data.head(2)
```

Out[198]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_subcategories	essay	price	Neg_sentiment	Neu_sentiment	Pos_sentiment	C
0	ca	mrs	grades_prek_2	53	1	math_science	appliedsciences health_lifescience	i fortunate enough use fairy tale stem kits cl...	725.05	0.013	0.783	0.205	
1	ut	ms	grades_3_5	4	1	specialneeds	specialneeds	imagine 8 9 years old you third grade classroo...	213.03	0.072	0.680	0.248	

In [199]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109248 entries, 0 to 109247
Data columns (total 13 columns):
 #   Column                                        Non-Null Count   Dtype  
---  ------                                        --------------   -----  
 0   school_state                                  109248 non-null  object 
 1   teacher_prefix                                109248 non-null  object 
 2   project_grade_category                        109248 non-null  object 
 3   teacher_number_of_previously_posted_projects  109248 non-null  int64  
 4   project_is_approved                           109248 non-null  int64  
 5   clean_categories                              109248 non-null  object 
 6   clean_subcategories                           109248 non-null  object 
 7   essay                                         109248 non-null  object 
 8   price                                         109248 non-null  float64
 9   Neg_sentiment                                 109248 non-null  float64
 10  Neu_sentiment                                 109248 non-null  float64
 11  Pos_sentiment                                 109248 non-null  float64
 12  Compound_val_sentiment                        109248 non-null  float64
dtypes: float64(5), int64(2), object(6)
memory usage: 10.8+ MB
```

In [200]:
```
X  =  data.drop('project_is_approved',axis=1) #Independent features
y  =  data['project_is_approved'] # Dependent feature
```

In [ ]:
```

```

1.2 Splitting data into Train and cross validation(or test): Stratified Sampling

In [201]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y,random_state=123)
```

In [202]:
```
X_train = X_train.reset_index(drop=True)
X_test = X_test.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
```

In [203]:
```
# Intializse the Variable to store all the feature name
Feature_names =[]
```

1.3 Make Data Model Ready: encoding eassay, and project_title

In [204]:
```
# Before Enoding just check the shape of X-train and X_CV
print('Before Encoding X_train ', X_train.shape)
print('Before Encoding X_test ', X_test.shape)

#Initializse TFIDF
vectorizer  = TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=3000)
vectorizer.fit(X_test['essay'].values)
# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)
#X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
#print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("-"*100)

Feature_names.extend(vectorizer.get_feature_names())
```

```
Before Encoding X_train  (73196, 12)
Before Encoding X_test  (36052, 12)
After vectorizations
(73196, 3000) (73196,)
(36052, 3000) (36052,)
----------------------------------------------------------------------------------------------------
```

In [ ]:

1.4 Make Data Model Ready: encoding numerical, categorical features

In [205]:
```
X_train['project_is_approved'] = y_train.values
X_test['project_is_approved'] =  y_test.values
```

In [206]:
```
State_Encoding = ['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', '
Teachers_pref = ['dr', 'mr', 'mrs', 'ms', 'teacher']
Project_grade_category = ['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
Clean_category= ['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
Clean_subcategory = ['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmental
```

In [207]:
```
#function used to find Negative and Postive prob of both train and test data.
def response_coding(X_train,cat_feature_names):
    col_dict = dict(X_train[cat_feature_names].value_counts())
    class_approve_x1 = dict()
    class_disapprove_x2 = dict()
    print(col_dict)
    for val in list(col_dict.keys()):
        #print(val)
        N = col_dict[val]
        index_val = X_train.index[X_train[cat_feature_names] == val].tolist()
        value_count_target = X_train.iloc[index_val]['project_is_approved'].value_counts()
        zero_index_val = X_train.index[X_train['project_is_approved'] == 0].tolist()
        one_index_val = X_train.index[X_train['project_is_approved'] == 1].tolist()
        #print(value_count_target)
        #print('##'*50)
        if len(value_count_target) !=2:
            if list(value_count_target.keys()) == [0]:
                value_count_target[1]= 0
            else:
                value_count_target[0]= [0]

        Approved_count = value_count_target[1]
        dis_Approve = value_count_target[0]
        if Approved_count !=0:
            class_approve = Approved_count / N
        else:
            class_approve = 0
        if dis_Approve !=0:
            class_disapprove =dis_Approve / N
        else:
            class_disapprove = 0
        class_approve_x1[val] = class_approve,one_index_val
        class_disapprove_x2[val]=class_disapprove,zero_index_val
    return class_approve_x1, class_disapprove_x2
```

In [208]:
```
class_one,class_zero = response_coding(X_train,'school_state') # Reasponse coding State

X_train['State_class_0']  = X_train['school_state'].apply(lambda row: class_zero[row][0] if (row in list(class_zero.keys())) else False)

X_train['State_class_1']  = X_train['school_state'].apply(lambda row: class_one[row][0] if (row in list(class_one.keys())) else False)

X_test['State_class_0']  = X_test['school_state'].apply(lambda row: class_zero[row][0] if (row in list(class_zero.keys())) else 0.5)

X_test['State_class_1']  = X_test['school_state'].apply(lambda row: class_one[row][0] if (row in list(class_one.keys())) else 0.5)
```

```
{'ca': 10360, 'tx': 4984, 'ny': 4903, 'fl': 4087, 'nc': 3421, 'il': 2934, 'ga': 2692, 'sc': 2594, 'mi': 2152, 'pa': 2044, 'in': 1739, 'mo': 1704, 'ma': 1652, 'oh': 1652, 'la': 1577, 'w
a': 1537, 'ok': 1535, 'nj': 1493, 'az': 1425, 'va': 1366, 'wi': 1213, 'al': 1200, 'ut': 1190, 'ct': 1114, 'tn': 1106, 'md': 1020, 'nv': 917, 'ms': 893, 'ky': 847, 'or': 828, 'mn': 819,
'co': 732, 'ar': 717, 'id': 478, 'ia': 442, 'ks': 427, 'nm': 384, 'dc': 352, 'wv': 349, 'hi': 340, 'me': 327, 'de': 236, 'ak': 234, 'nh': 223, 'ne': 214, 'sd': 207, 'ri': 190, 'mt': 13
8, 'nd': 88, 'wy': 64, 'vt': 56}
```

In [209]:
```
class_one,class_zero = response_coding(X_train,'teacher_prefix') # Reasponse coding Teacher Prefix

X_train['teacher_prefix_class_0']  = X_train['teacher_prefix'].apply(lambda row: class_zero[row][0] if (row in list(class_zero.keys())) else False)

X_train['teacher_prefix_class_1']  = X_train['teacher_prefix'].apply(lambda row: class_one[row][0] if (row in list(class_one.keys())) else False)

X_test['teacher_prefix_class_0']  = X_test['teacher_prefix'].apply(lambda row: class_zero[row][0] if (row in list(class_zero.keys())) else 0.5)

X_test['teacher_prefix_class_1']  = X_test['teacher_prefix'].apply(lambda row: class_one[row][0] if (row in list(class_one.keys())) else 0.5)
```

```
{'mrs': 38452, 'ms': 26027, 'mr': 7101, 'teacher': 1606, 'dr': 10}
```

In [210]:
```
class_one,class_zero = response_coding(X_train,'project_grade_category') # Reasponse coding Project Grade

X_train['project_grade_category_class_0']  = X_train['project_grade_category'].apply(lambda row: class_zero[row][0] if (row in list(class_zero.keys())) else False)

X_train['project_grade_category_class_1']  = X_train['project_grade_category'].apply(lambda row: class_one[row][0] if (row in list(class_one.keys())) else False)

X_test['project_grade_category_class_0']  = X_test['project_grade_category'].apply(lambda row: class_zero[row][0] if (row in list(class_zero.keys())) else 0.5)

X_test['project_grade_category_class_1']  = X_test['project_grade_category'].apply(lambda row: class_one[row][0] if (row in list(class_one.keys())) else 0.5)
```

```
{'grades_prek_2': 29579, 'grades_3_5': 24947, 'grades_6_8': 11295, 'grades_9_12': 7375}
```

In [211]:
```
class_one,class_zero = response_coding(X_train,'clean_categories') # Reasponse coding clean_categories

X_train['clean_categories_class_0']  = X_train['clean_categories'].apply(lambda row: class_zero[row][0] if (row in list(class_zero.keys())) else False)

X_train['clean_categories_class_1']  = X_train['clean_categories'].apply(lambda row: class_one[row][0] if (row in list(class_one.keys())) else False)

X_test['clean_categories_class_0']  = X_test['clean_categories'].apply(lambda row: class_zero[row][0] if (row in list(class_zero.keys())) else 0.5)

X_test['clean_categories_class_1']  = X_test['clean_categories'].apply(lambda row: class_one[row][0] if (row in list(class_one.keys())) else 0.5)
```

```
{'literacy_language': 15824, 'math_science': 11434, 'literacy_language math_science': 9799, 'health_sports': 6822, 'music_arts': 3496, 'specialneeds': 2808, 'literacy_language specialneeds': 2680, 'appliedlearning': 2590, 'math_science literacy_language': 1583, 'appliedlearning literacy_language': 1453, 'history_civics': 1238, 'math_science specialneeds': 1221, 'literacy_language music_arts': 1176, 'math_science music_arts': 1095, 'appliedlearning specialneeds': 951, 'health_sports specialneeds': 933, 'history_civics literacy_language': 911, 'warmth care_hunger': 882, 'math_science appliedlearning': 808, 'appliedlearning math_science': 710, 'literacy_language history_civics': 545, 'health_sports literacy_language': 534, 'appliedlearning music_arts': 501, 'math_science history_civics': 441, 'appliedlearning health_sports': 421, 'literacy_language appliedlearning': 418, 'math_science health_sports': 266, 'history_civics music_arts': 224, 'history_civics math_science': 220, 'specialneeds music_arts': 202, 'health_sports math_science': 182, 'history_civics specialneeds': 160, 'health_sports appliedlearning': 128, 'appliedlearning history_civics': 115, 'music_arts specialneeds': 97, 'health_sports music_arts': 95, 'literacy_language health_sports': 50, 'history_civics appliedlearning': 29, 'health_sports history_civics': 27, 'specialneeds health_sports': 25, 'health_sports warmth care_hunger': 22, 'specialneeds warmth care_hunger': 19, 'music_arts health_sports': 15, 'music_arts history_civics': 12, 'math_science warmth care_hunger': 10, 'appliedlearning warmth care_hunger': 6, 'music_arts appliedlearning': 6, 'literacy_language warmth care_hunger': 6, 'history_civics health_sports': 5, 'music_arts warmth care_hunger': 1}
```

In [212]:
```
class_one,class_zero = response_coding(X_train,'clean_subcategories') # Reasponse coding Clean SubCategories

X_train['clean_subcategories_class_0']  = X_train['clean_subcategories'].apply(lambda row: class_zero[row][0] if (row in list(class_zero.keys())) else False)

X_train['clean_subcategories_class_1']  = X_train['clean_subcategories'].apply(lambda row: class_one[row][0] if (row in list(class_one.keys())) else False)

X_test['clean_subcategories_class_0']  = X_test['clean_subcategories'].apply(lambda row: class_zero[row][0] if (row in list(class_zero.keys())) else 0.5)

X_test['clean_subcategories_class_1']  = X_test['clean_subcategories'].apply(lambda row: class_one[row][0] if (row in list(class_one.keys())) else 0.5)
```

```
{'literacy': 6295, 'literacy mathematics': 5633, 'literature_writing mathematics': 3922, 'literacy literature_writing': 3744, 'mathematics': 3639, 'literature_writing': 3084, 'special
needs': 2808, 'health_wellness': 2412, 'appliedsciences mathematics': 2271, 'appliedsciences': 1662, 'literacy specialneeds': 1633, 'gym_fitness health_wellness': 1515, 'visualarts':
1510, 'esl literacy': 1476, 'music': 1016, 'literature_writing specialneeds': 900, 'warmth care_hunger': 882, 'mathematics specialneeds': 804, 'health_wellness specialneeds': 800, 'gy
m_fitness': 799, 'teamsports': 736, 'environmentalscience': 726, 'appliedsciences environmentalscience': 668, 'music performingarts': 629, 'environmentalscience health_lifescience': 6
25, 'earlydevelopment': 616, 'other': 582, 'environmentalscience mathematics': 548, 'health_lifescience': 542, 'health_wellness nutritioneducation': 517, 'earlydevelopment specialneed
s': 503, 'esl literature_writing': 484, 'earlydevelopment literacy': 472, 'literature_writing visualarts': 461, 'appliedsciences visualarts': 410, 'appliedsciences health_lifescienc
e': 403, 'gym_fitness teamsports': 390, 'appliedsciences literacy': 389, 'history_geography literature_writing': 386, 'literacy visualarts': 367, 'history_geography': 356, 'health_lif
escience mathematics': 350, 'mathematics visualarts': 328, 'history_geography literacy': 327, 'health_wellness literacy': 315, 'environmentalscience literacy': 306, 'college_careerpre
p': 305, 'esl': 290, 'appliedsciences literature_writing': 290, 'appliedsciences college_careerprep': 272, 'literacy socialsciences': 253, 'performingarts': 251, 'literature_writing s
ocialsciences': 234, 'health_wellness teamsports': 232, 'charactereducation': 229, 'appliedsciences specialneeds': 225, 'charactereducation literacy': 225, 'foreignlanguages': 223, 'h
ealth_lifescience literacy': 216, 'college_careerprep mathematics': 216, 'college_careerprep literature_writing': 208, 'history_geography socialsciences': 208, 'earlydevelopment healt
h_wellness': 206, 'other specialneeds': 204, 'environmentalscience literature_writing': 202, 'specialneeds visualarts': 202, 'health_wellness literature_writing': 193, 'earlydevelopme
nt mathematics': 187, 'nutritioneducation': 174, 'esl mathematics': 166, 'civics_government history_geography': 165, 'earlydevelopment literature_writing': 155, 'health_wellness mathe
matics': 155, 'college_careerprep literacy': 153, 'foreignlanguages literacy': 142, 'environmentalscience visualarts': 137, 'socialsciences': 134, 'esl specialneeds': 129, 'health_lif
escience literature_writing': 122, 'history_geography visualarts': 120, 'charactereducation specialneeds': 117, 'literacy other': 116, 'charactereducation literature_writing': 116, 'a
ppliedsciences earlydevelopment': 114, 'health_wellness other': 112, 'gym_fitness specialneeds': 111, 'charactereducation earlydevelopment': 110, 'earlydevelopment visualarts': 108,
'financialliteracy': 105, 'environmentalscience history_geography': 104, 'health_lifescience health_wellness': 104, 'environmentalscience specialneeds': 103, 'earlydevelopment other':
100, 'civics_government literacy': 100, 'literacy parentinvolvement': 97, 'literature_writing other': 95, 'extracurricular': 91, 'literacy performingarts': 90, 'college_careerprep vis
ualarts': 90, 'health_lifescience specialneeds': 89, 'appliedsciences extracurricular': 86, 'financialliteracy mathematics': 86, 'music specialneeds': 85, 'charactereducation college_
```

In [213]:
```
# Train CSR Matrix
X_tr_state_0 = csr_matrix(X_train['State_class_0']).reshape(-1,1)
X_tr_state_1= csr_matrix(X_train['State_class_1']).reshape(-1,1)

X_tr_teacher_prefix_class_0 = csr_matrix(X_train['teacher_prefix_class_0']).reshape(-1,1)
X_tr_teacher_prefix_class_1= csr_matrix(X_train['teacher_prefix_class_1']).reshape(-1,1)

X_tr_project_grade_category_class_0 = csr_matrix(X_train['project_grade_category_class_0']).reshape(-1,1)
X_tr_project_grade_category_class_1= csr_matrix(X_train['project_grade_category_class_1']).reshape(-1,1)

X_tr_clean_categories_class_0 = csr_matrix(X_train['clean_categories_class_0'].astype('float64')).reshape(-1,1)
X_tr_clean_categories_class_1= csr_matrix(X_train['clean_categories_class_1'].astype('float64')).reshape(-1,1)

X_tr_clean_subcategories_class_0 = csr_matrix(X_train['clean_subcategories_class_0'].astype('float64')).reshape(-1,1)
X_tr_clean_subcategories_class_1= csr_matrix(X_train['clean_subcategories_class_1'].astype('float64')).reshape(-1,1)

X_tr_Nega_sent = csr_matrix(X_train['Neg_sentiment']).reshape(-1,1)

X_tr_Neu_sent= csr_matrix(X_train['Neu_sentiment']).reshape(-1,1)

X_tr_Pos_sent = csr_matrix(X_train['Pos_sentiment']).reshape(-1,1)

X_tr_comp_sent= csr_matrix(X_train['Compound_val_sentiment']).reshape(-1,1)
```

In [214]:
```
X_train_essay_bow
```

Out[214]:
```
<73196x3000 sparse matrix of type '<class 'numpy.float64'>'
        with 8672705 stored elements in Compressed Sparse Row format>
```

In [215]:
```
# Test CSR Matrix
X_te_state_0 = csr_matrix(X_test['State_class_0']).reshape(-1,1)
X_te_state_1= csr_matrix(X_test['State_class_1']).reshape(-1,1)

X_te_teacher_prefix_class_0 = csr_matrix(X_test['teacher_prefix_class_0']).reshape(-1,1)
X_te_teacher_prefix_class_1= csr_matrix(X_test['teacher_prefix_class_1']).reshape(-1,1)

X_te_project_grade_category_class_0 = csr_matrix(X_test['project_grade_category_class_0']).reshape(-1,1)
X_te_project_grade_category_class_1= csr_matrix(X_test['project_grade_category_class_1']).reshape(-1,1)

X_te_clean_categories_class_0 = csr_matrix(X_test['clean_categories_class_0'].astype('float64')).reshape(-1,1)
X_te_clean_categories_class_1= csr_matrix(X_test['clean_categories_class_1'].astype('float64')).reshape(-1,1)

X_te_clean_subcategories_class_0 = csr_matrix(X_test['clean_subcategories_class_0'].astype('float64')).reshape(-1,1)
X_te_clean_subcategories_class_1= csr_matrix(X_test['clean_subcategories_class_1'].astype('float64')).reshape(-1,1)

X_te_Nega_sent = csr_matrix(X_test['Neg_sentiment']).reshape(-1,1)

X_te_Neu_sent= csr_matrix(X_test['Neu_sentiment']).reshape(-1,1)

X_te_Pos_sent = csr_matrix(X_test['Pos_sentiment']).reshape(-1,1)

X_te_comp_sent= csr_matrix(X_test['Compound_val_sentiment']).reshape(-1,1)
```

Numerical Encoding- TFIDF

In [216]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

X_train['price']= normalizer.fit_transform(X_train['price'].values.reshape(-1,1))
X_train['teacher_number_of_previously_posted_projects']= normalizer.fit_transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
```

In [217]:
```
X_train.drop('project_is_approved',axis=1,inplace=True)
X_test.drop('project_is_approved',axis=1,inplace=True)
```

In [218]:
```
# NumericalFeatures
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

print('1. Price')
normalizer.fit(X_train['price'].values.reshape(-1,1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))



print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
#print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)

print('1. Teacher_number_of_previously_posted_projects')
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_previous_prsub_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
#X_cv_previous_prsub_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_test_previous_prsub_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))



print("After vectorizations")
print(X_train_previous_prsub_norm.shape, y_train.shape)
#print(X_cv_previous_prsub_norm.shape, y_cv.shape)
print(X_test_previous_prsub_norm.shape, y_test.shape)
print("="*100)

Feature_names.append(['Price'])
Feature_names.append(['teacher_number_of_previously_posted_projects'])
```

```
1. Price
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
1. Teacher_number_of_previously_posted_projects
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

In [219]:
```
from scipy.sparse import hstack,csr_matrix
X_tr = hstack((X_train_essay_bow,X_train_price_norm,X_train_previous_prsub_norm,
               X_tr_state_0,X_tr_state_1,X_tr_teacher_prefix_class_0,X_tr_teacher_prefix_class_1,
               X_tr_project_grade_category_class_0,X_tr_project_grade_category_class_1,
               X_tr_clean_categories_class_0,X_tr_clean_categories_class_1,
               X_tr_clean_subcategories_class_0,X_tr_clean_subcategories_class_1,
               X_tr_Nega_sent,X_tr_Neu_sent,X_tr_Pos_sent,X_tr_comp_sent)).tocsr()
```

In [220]:
```
X_tr.shape
```

Out[220]:
```
(73196, 3016)
```

In [221]:
```
X_te = hstack((X_test_essay_bow,X_test_price_norm,X_test_previous_prsub_norm,
               X_te_state_0,X_te_state_1,X_te_teacher_prefix_class_0,X_te_teacher_prefix_class_1,
               X_te_project_grade_category_class_0,X_te_project_grade_category_class_1,
               X_te_clean_categories_class_0,X_te_clean_categories_class_1,
               X_te_clean_subcategories_class_0,X_te_clean_subcategories_class_1,
               X_te_Nega_sent,X_te_Neu_sent,X_te_Pos_sent,X_te_comp_sent)).tocsr()
```

In [223]:
```
X_test_essay_bow
```

Out[223]:
```
<36052x3000 sparse matrix of type '<class 'numpy.float64'>'
        with 4277441 stored elements in Compressed Sparse Row format>
```

1.5 Appling Models on different kind of featurization as mentioned in the instructions

In [ ]:
```
```

Apply GBDT on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

In [224]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_auc_score
```

In [225]:
```
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
n_estimators=[5,10,50, 75, 100]

# Initializse - store AUC score
Learn_rate  = []
n_estimator  = []
roc_auc_values  = []

for alpha in tqdm(learning_rate):
    for n_est in tqdm(n_estimators):
        DTC = GradientBoostingClassifier(learning_rate=alpha,n_estimators=n_est)
        DTC.fit(X_tr,y_train)
        y_pred_prob = DTC.predict_proba(X_te)[:,1]
        result_roc_auc = roc_auc_score(y_test,y_pred_prob)
        Learn_rate.append(alpha)
        n_estimator.append(n_est)
        roc_auc_values.append(result_roc_auc)
```

```
  0%|                                                                                                  | 0/6 [00:00<?, ?it/s]
  0%|                                                                                                  | 0/5 [00:00<?, ?it/s]
 20%|████████████████                                                                  | 1/5 [00:42<02:48, 42.10s/it]
 40%|████████████████████████████████                                                  | 2/5 [02:07<02:45, 55.14s/it]
 60%|████████████████████████████████████████████████                                  | 3/5 [09:03<05:26, 163.32s/it]
 80%|████████████████████████████████████████████████████████████████                  | 4/5 [19:21<04:59, 299.88s/it]
100%|████████████████████████████████████████████████████████████████████████████████| 5/5 [33:06<00:00, 397.38s/it]
 17%|█████████████▋                                                                   | 1/6 [33:06<2:45:34, 1986.91s/it]
  0%|                                                                                                  | 0/5 [00:00<?, ?it/s]
 20%|████████████████                                                                  | 1/5 [00:42<02:49, 42.44s/it]
 40%|████████████████████████████████                                                  | 2/5 [02:03<02:42, 54.06s/it]
 60%|████████████████████████████████████████████████                                  | 3/5 [08:56<05:23, 161.78s/it]
 80%|████████████████████████████████████████████████████████████████                  | 4/5 [17:32<04:28, 268.02s/it]
100%|████████████████████████████████████████████████████████████████████████████████| 5/5 [27:35<00:00, 331.01s/it]
 33%|██████████████████████████                                                       | 2/6 [1:00:41<2:05:49, 1887.36s/it]
  0%|                                                                                                  | 0/5 [00:00<?, ?it/s]
 20%|████████████████                                                                  | 1/5 [00:31<02:06, 31.64s/it]
 40%|████████████████████████████████                                                  | 2/5 [01:32<02:01, 40.51s/it]
 60%|████████████████████████████████████████████████                                  | 3/5 [06:34<03:57, 118.71s/it]
```

```
 80%|████████████████████████████████████████████████████████▊                 | 4/5 [14:05<03:38, 218.64s/it]
100%|████████████████████████████████████████████████████████████████████████| 5/5 [24:09<00:00, 289.93s/it]
 50%|████████████████████████████                                              | 3/6 [1:24:51<1:27:48, 1756.06s/it]
  0%|                                                                                  | 0/5 [00:00<?, ?it/s]
 20%|████████████▊                                                              | 1/5 [00:30<02:01, 30.41s/it]
 40%|█████████████████████████                                                  | 2/5 [01:31<01:58, 39.55s/it]
 60%|██████████████████████████████████                                         | 3/5 [06:34<03:57, 118.66s/it]
 80%|████████████████████████████████████████████████████████▊                 | 4/5 [14:11<03:40, 220.13s/it]
100%|████████████████████████████████████████████████████████████████████████| 5/5 [24:29<00:00, 293.83s/it]
 67%|███████████████████████████████████████████                               | 4/6 [1:49:20<55:39, 1669.99s/it]
  0%|                                                                                  | 0/5 [00:00<?, ?it/s]
 20%|████████████▊                                                              | 1/5 [00:30<02:01, 30.33s/it]
 40%|█████████████████████████                                                  | 2/5 [01:30<01:58, 39.42s/it]
 60%|██████████████████████████████████                                         | 3/5 [08:29<05:06, 153.19s/it]
 80%|████████████████████████████████████████████████████████▊                 | 4/5 [19:16<05:01, 301.20s/it]
100%|████████████████████████████████████████████████████████████████████████| 5/5 [32:31<00:00, 390.35s/it]
 83%|████████████████████████████████████████████████████████▊                 | 5/6 [2:21:52<29:14, 1754.52s/it]
  0%|                                                                                  | 0/5 [00:00<?, ?it/s]
 20%|████████████▊                                                              | 1/5 [00:38<02:32, 38.01s/it]
 40%|█████████████████████████                                                  | 2/5 [01:52<02:27, 49.01s/it]
 60%|██████████████████████████████████                                         | 3/5 [08:08<04:54, 147.10s/it]
 80%|████████████████████████████████████████████████████                      | 4/5 [7:15:53<2:10:02, 7802.42s/it]
100%|████████████████████████████████████████████████████████████████████████| 5/5 [7:28:26<00:00, 5381.26s/it]
100%|████████████████████████████████████████████████████████████████████████| 6/6 [9:50:18<00:00, 5903.16s/it]
```

In [226]:
```
print((Learn_rate))
print((n_estimator))
print((roc_auc_values))
best_result_acu = pd.DataFrame(data=[Learn_rate,n_estimator,roc_auc_values]).T
best_result_acu.rename(columns ={0:'Alpha_values',1:'n_estimator',2:'AUC_score'},inplace=True)
best_result_acu.sort_values(by=['AUC_score'],inplace=True,ignore_index=True,ascending=False)
best_result_acu.head()
```

```
[0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.01, 0.01, 0.01, 0.01, 0.01, 0.1, 0.1, 0.1, 0.1, 0.1, 0.2, 0.2, 0.2, 0.2, 0.2, 0.3, 0.3, 0.3, 0.3, 0.3]
[5, 10, 50, 75, 100, 5, 10, 50, 75, 100, 5, 10, 50, 75, 100, 5, 10, 50, 75, 100, 5, 10, 50, 75, 100, 5, 10, 50, 75, 100]
[0.5913755226593931, 0.5913755226593931, 0.5913755226593931, 0.5913755226593931, 0.5913755226593931, 0.5913755226593931, 0.5913755226593931, 0.5910688741796483, 0.5942537401099989, 0.59
5308272571527, 0.5910688741796483, 0.5942972741646142, 0.6050569488365791, 0.6126993085632896, 0.6182807300382827, 0.6010409809489217, 0.6162368090182849, 0.6639487347331945, 0.67476331
1234025, 0.6815902270122064, 0.6155872920606704, 0.6372966062831775, 0.6818614608483885, 0.6910963119209953, 0.6936277299251798, 0.6255786135718818, 0.6482085229062626, 0.68464487998351
83, 0.69135747134044, 0.694147063862587]
```

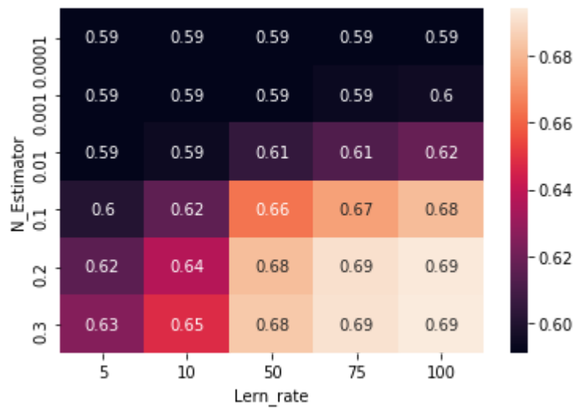
Out[226]:

	Alpha_values	n_estimator	AUC_score
0	0.3	100.0	0.694147
1	0.2	100.0	0.693628
2	0.3	75.0	0.691357
3	0.2	75.0	0.691096
4	0.3	50.0	0.684645

In [227]:
```
#Best Hyperparameter for As part of AUC
print('Best Learning is ',str(0.03))
print('Best n_estimator ',str(100.0))
print('Best AUC_Score value is ',str(0.69))
```

```
Best Learning is  0.03
Best n_estimator  100.0
Best AUC_Score value is  0.69
```

In [228]:
```
n_estimator_GBT = Learn_rate
Learn_rate_GBT = n_estimator
roc_values = roc_auc_values
data = pd.DataFrame({'N_Estimator': n_estimator_GBT, 'Lern_rate': Learn_rate_GBT, 'AUC': roc_values})
data_pivoted = data.pivot("N_Estimator", "Lern_rate", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.show()
```



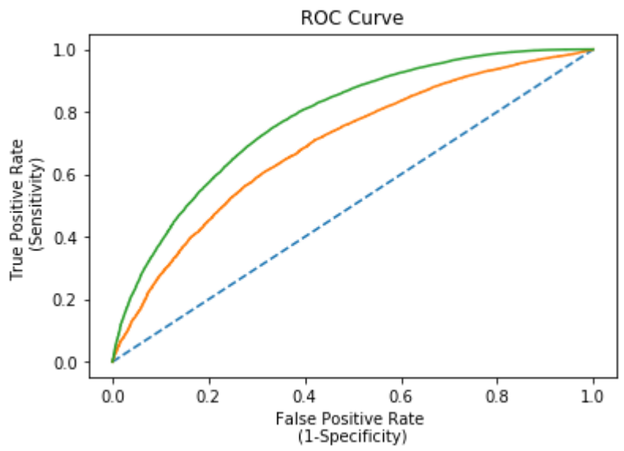
In [230]:
```
# Apply best Decision Tree on test data set
DTC = GradientBoostingClassifier(learning_rate=0.3,n_estimators=100)
DTC.fit(X_tr,y_train)


y_pred_prob = DTC.predict_proba(X_te)[:,1]
fpr,tpr,thershold = roc_curve(y_test,y_pred_prob)
y_pred_prob_train = DTC.predict_proba(X_tr)[:,1]
fpr_tr,tpr_tr,thershold_tr = roc_curve(y_train,y_pred_prob_train)
#print(fpr)
#print(tpr)
#print(thershold)
#fpr - False Postive rate
#tpr - True Postive rate

#Plot ROC curve
plt.plot([0,1],[0,1],'--')
plt.plot(fpr, tpr, label='GBT_Decision_Tree ')
plt.plot(fpr_tr, tpr_tr, label='GBT_Decision_Tree')
plt.xlabel('False Positive Rate \n (1-Specificity) ')
plt.ylabel('True Positive Rate \n (Sensitivity)')
plt.title('ROC Curve')
plt.show()


predict = DTC.predict(X_tr)
print('Confusion Metrics-Train')
print(confusion_matrix(y_train,predict)) # confusion matrix
print('Accuracy_Score: ',accuracy_score(y_train,predict)) # score  - 0.842 # Good Score


predict = DTC.predict(X_te)
print('Confusion Metrics-Test')
print(confusion_matrix(y_test,predict)) # confusion matrix
print('Accuracy_Score: ',accuracy_score(y_test,predict)) # score  - 0.842 # Good Score
```



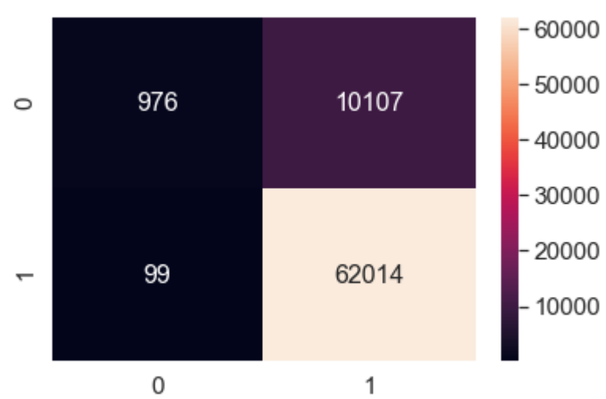

```
Confusion Metrics-Train
```

```
[[  976 10107]
 [   99 62014]]
Accuracy_Score:  0.8605661511558008
Confusion Metrics-Test
[[  139  5320]
 [  266 30327]]
Accuracy_Score:  0.8450571396871186
```

In [276]:
```
array = [[976,10107],
         [99,62014]]

df_cm = pd.DataFrame(array, range(2), range(2))
# plt.figure(figsize=(10,7))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16},fmt='d') # font size
```

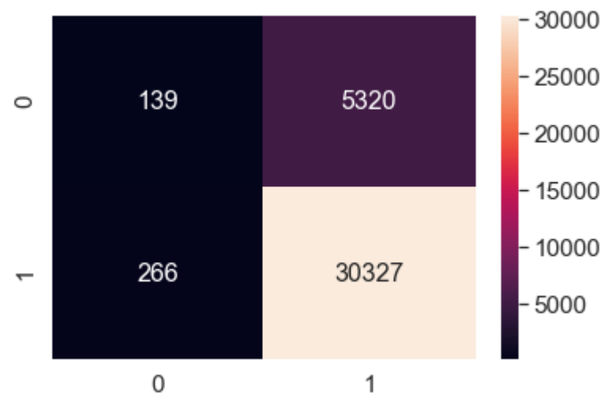
Out[276]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x27e1ac68888>
```



In [280]:
```
array = [[139,5320],
         [266,30327]]

df_cm = pd.DataFrame(array, range(2), range(2))
# plt.figure(figsize=(10,7))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16},fmt='d') # font size
```

Out[280]: <matplotlib.axes._subplots.AxesSubplot at 0x27e1a2dad48>



3. Summary

as mentioned in the step 4 of instructions

In [237]:
```
from prettytable import PrettyTable
x1 = PrettyTable()
x1.field_names = ["Vectorizer", "Model", "Learning_Rate",'N_estimator','AU_Score']
x1.add_row(['TFIDF','Decision_Tree','0.03','100.','0.69'])
print(x1)
```

```
+------------+---------------+---------------+-------------+----------+
| Vectorizer |     Model     | Learning_Rate | N_estimator | AU_Score |
+------------+---------------+---------------+-------------+----------+
|   TFIDF    | Decision_Tree |      0.03     |     100.    |   0.69   |
+------------+---------------+---------------+-------------+----------+
```

TFIDF- Avg W2V

In [295]:
```
#Reading the Input file
preprocessed_essays_train = X_train['essay'].values
preprocessed_essays_test = X_test['essay'].values
```

In [296]:
```
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [240]:
```
tfidf_model = TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=3000)
tfidf_model.fit(preprocessed_essays_train)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [241]:
```
len(glove_words)
```

Out[241]: 51510

In [242]:
```
# average Word2Vec- Train
# compute average word2vec for each review.
tfidf_w2v_vectors_tr = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_tr.append(vector)

print(len(tfidf_w2v_vectors_tr))
print(len(tfidf_w2v_vectors_tr[0]))
```

```
  0%|                                                                                | 0/73196 [00:00<?, ?it/s]
  0%|                                                                              | 15/73196 [00:00<08:11, 148.91it/s]
  0%|                                                                              | 30/73196 [00:00<08:17, 147.16it/s]
  0%|                                                                              | 49/73196 [00:00<07:46, 156.77it/s]
  0%|                                                                              | 64/73196 [00:00<08:06, 150.17it/s]
  0%|                                                                              | 77/73196 [00:00<08:35, 141.74it/s]
  0%|                                                                             | 104/73196 [00:00<07:22, 165.08it/s]
  0%|▏                                                                            | 130/73196 [00:00<06:34, 185.09it/s]
  0%|▏                                                                            | 157/73196 [00:00<05:58, 203.57it/s]
  0%|▏                                                                            | 179/73196 [00:01<07:21, 165.53it/s]
  0%|▏                                                                            | 202/73196 [00:01<07:34, 160.44it/s]
  0%|▏                                                                            | 220/73196 [00:01<07:42, 157.69it/s]
  0%|▏                                                                            | 248/73196 [00:01<06:44, 180.18it/s]
  0%|▏                                                                            | 275/73196 [00:01<06:04, 199.83it/s]
  0%|▏                                                                            | 297/73196 [00:01<05:56, 204.49it/s]
  0%|▏                                                                            | 319/73196 [00:01<06:14, 194.68it/s]
  0%|▏                                                                            | 340/73196 [00:01<07:23, 164.23it/s]
  0%|▏                                                                            | 358/73196 [00:02<07:33, 160.67it/s]
```

In [297]:
```
X_tr_w2v = csr_matrix(tfidf_w2v_vectors_tr)
```

In [298]:
```
X_train_w2c = hstack((X_tr_w2v,X_train_price_norm,X_train_previous_prsub_norm,
                X_tr_state_0,X_tr_state_1,X_tr_teacher_prefix_class_0,X_tr_teacher_prefix_class_1,
                X_tr_project_grade_category_class_0,X_tr_project_grade_category_class_1,
                X_tr_clean_categories_class_0,X_tr_clean_categories_class_1,
                X_tr_clean_subcategories_class_0,X_tr_clean_subcategories_class_1,
                       X_tr_Nega_sent,X_tr_Neu_sent,X_tr_Pos_sent,X_tr_comp_sent
                )).tocsr()
```

In [247]:
```
# Average w2V Test data
tfidf_w2v_vectors_te = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_test): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_te.append(vector)
```

```
  0%|                                                                                | 0/36052 [00:00<?, ?it/s]
  0%|                                                                               | 12/36052 [00:00<05:08, 116.82it/s]
  0%|                                                                               | 31/36052 [00:00<04:34, 131.28it/s]
  0%|                                                                               | 58/36052 [00:00<03:52, 154.74it/s]
  0%|                                                                               | 79/36052 [00:00<03:35, 167.18it/s]
  0%|                                                                              | 107/36052 [00:00<03:09, 189.88it/s]
  0%|▏                                                                             | 137/36052 [00:00<02:48, 212.60it/s]
  0%|▏                                                                             | 160/36052 [00:00<02:48, 213.47it/s]
  1%|▏                                                                             | 183/36052 [00:00<03:02, 197.07it/s]
  1%|▏                                                                             | 204/36052 [00:00<03:10, 188.01it/s]
  1%|▏                                                                             | 224/36052 [00:01<03:16, 182.70it/s]
  1%|▏                                                                            | 253/36052 [00:01<03:06, 192.17it/s]
  1%|▏                                                                            | 274/36052 [00:01<03:02, 196.24it/s]
  1%|▏                                                                             | 294/36052 [00:01<03:05, 192.39it/s]
  1%|▏                                                                             | 317/36052 [00:01<02:56, 201.93it/s]
  1%|▏                                                                             | 338/36052 [00:01<02:58, 199.79it/s]
  1%|▏                                                                             | 360/36052 [00:01<02:56, 202.21it/s]
  1%|▏                                                                             | 381/36052 [00:01<03:14, 183.75it/s]
```

In [299]:
```
X_te_w2v = csr_matrix(tfidf_w2v_vectors_te)
```

In [300]:
```
X_test_w2v = hstack((X_te_w2v,X_test_price_norm,X_test_previous_prsub_norm,
                X_te_state_0,X_te_state_1,X_te_teacher_prefix_class_0,X_te_teacher_prefix_class_1,
                X_te_project_grade_category_class_0,X_te_project_grade_category_class_1,
                X_te_clean_categories_class_0,X_te_clean_categories_class_1,
                X_te_clean_subcategories_class_0,X_te_clean_subcategories_class_1,
                 X_te_Nega_sent,X_te_Neu_sent,X_te_Pos_sent,X_te_comp_sent
                )).tocsr()
```

In [274]:
```
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
n_estimators=[5,10,50, 75, 100]

# Initializse - store AUC score
Learn_rate  = []
n_estimator  = []
roc_auc_values  = []

for alpha in tqdm(learning_rate):
    for n_est in tqdm(n_estimators):
        DTC = GradientBoostingClassifier(learning_rate=alpha,n_estimators=n_est)
        DTC.fit(X_train_w2c,y_train)
        y_pred_prob = DTC.predict_proba(X_test_w2v)[:,1]
        result_roc_auc = roc_auc_score(y_test,y_pred_prob)
        Learn_rate.append(alpha)
        n_estimator.append(n_est)
        roc_auc_values.append(result_roc_auc)
```

```
100%|██████████| 5/5 [35:24<00:00, 424.90s/it]

83%|████████▎ | 5/6 [4:03:56<44:50, 2690.04s/it]

 0%|          | 0/5 [00:00<?, ?it/s]

20%|██        | 1/5 [00:45<03:00, 45.02s/it]
```

In [275]:
```
print((Learn_rate))
print((n_estimator))
print((roc_auc_values))
best_result_acu = pd.DataFrame(data=[Learn_rate,n_estimator,roc_auc_values]).T
best_result_acu.rename(columns ={0:'Alpha_values',1:'n_estimator',2:'AUC_score'},inplace=True)
best_result_acu.sort_values(by=['AUC_score'],inplace=True,ignore_index=True,ascending=False)
best_result_acu.head()
```

```
[0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.01, 0.01, 0.01, 0.01, 0.01, 0.1, 0.1, 0.1, 0.1, 0.1, 0.2, 0.2, 0.2, 0.2, 0.2, 0.3, 0.3, 0.3, 0.3, 0.3]
[5, 10, 50, 75, 100, 5, 10, 50, 75, 100, 5, 10, 50, 75, 100, 5, 10, 50, 75, 100, 5, 10, 50, 75, 100, 5, 10, 50, 75, 100]
[0.5915172171602411, 0.5915172171602411, 0.59519227756348, 0.5954906898707301, 0.5997059276257375, 0.5951604226469607, 0.5954906898707301, 0.6002254202389506, 0.6046285960136554, 0.6033653928917442, 0.5995709424169872, 0.6038904421520495, 0.6198743979802498, 0.6272014419355498, 0.6322844387529263, 0.6160693671225059, 0.6320687546817 98, 0.6679427484758484, 0.6730716924176443, 0.6765380761727338, 0.6257370408855518, 0.6434154716946403, 0.6766782198421197, 0.679083239094375, 0.6788822297809255, 0.6321983406618303, 0.6503059056973399, 0.6734900576464412, 0.6751876013575392, 0.6746214161430071]
```

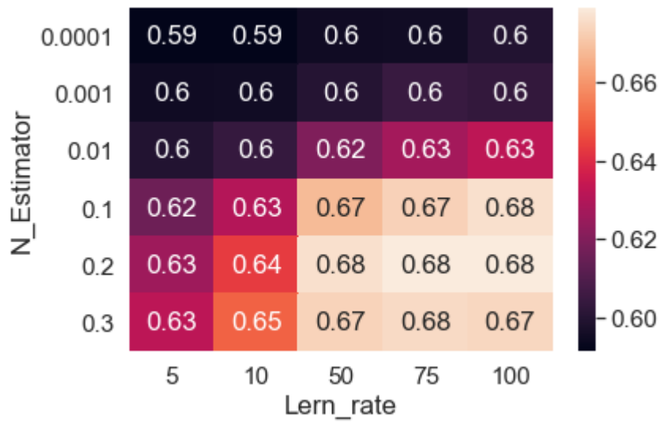
Out[275]:

	Alpha_values	n_estimator	AUC_score
0	0.2	75.0	0.679083
1	0.2	100.0	0.678882
2	0.2	50.0	0.676678
3	0.1	100.0	0.676538
4	0.3	75.0	0.675188

In [279]:
```
#Best Hyperparameter for As part of AUC
print('Best Learning is ',str(0.2))
print('Best n_estimator ',str(75.0))
print('Best AUC_Score value is ',str(0.68))
```

```
Best Learning is  0.2
Best n_estimator  75.0
Best AUC_Score value is  0.68
```

In [278]:
```
n_estimator_GBT = Learn_rate
Learn_rate_GBT = n_estimator
roc_values = roc_auc_values
data = pd.DataFrame({'N_Estimator': n_estimator_GBT, 'Lern_rate': Learn_rate_GBT, 'AUC': roc_values})
data_pivoted = data.pivot("N_Estimator", "Lern_rate", "AUC")
ax = sns.heatmap(data_pivoted,annot=True)
plt.show()
```



In [302]:
```
type(X_train_w2c)
```

Out[302]: scipy.sparse.csr.csr_matrix

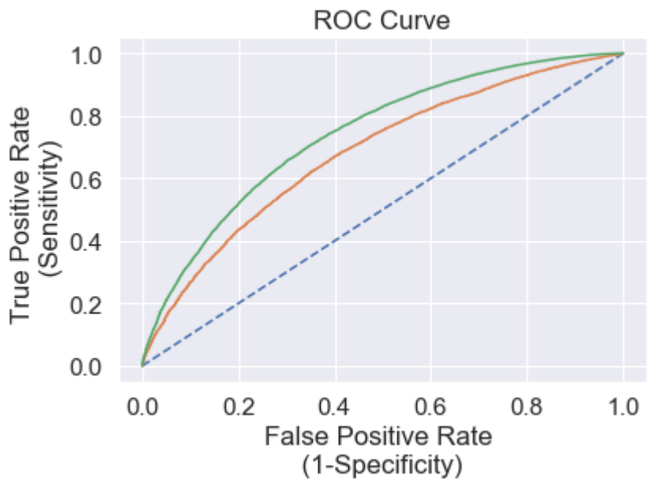
In [305]:
```
# Apply best Decision Tree on test data set
DTC = GradientBoostingClassifier(learning_rate=0.2,n_estimators=75)
DTC.fit(X_train_w2c,y_train)


y_pred_prob = DTC.predict_proba(X_test_w2v)[:,1]
fpr,tpr,thershold = roc_curve(y_test,y_pred_prob)
y_pred_prob_train = DTC.predict_proba(X_train_w2c)[:,1]
fpr_tr,tpr_tr,thershold_tr = roc_curve(y_train,y_pred_prob_train)
#print(fpr)
#print(tpr)
#print(thershold)
#fpr - False Postive rate
#tpr - True Postive rate

#Plot ROC curve
plt.plot([0,1],[0,1],'--')
plt.plot(fpr, tpr, label='Decision_tree_Classifier ')
plt.plot(fpr_tr, tpr_tr, label='Decision_tree_Classifier ')
plt.xlabel('False Positive Rate \n (1-Specificity) ')
plt.ylabel('True Positive Rate \n (Sensitivity)')
plt.title('ROC Curve')
plt.show()

predict = DTC.predict(X_train_w2c)
print('Confusion Metrics-Train')
print(confusion_matrix(y_train,predict)) # confusion matrix
print('Accuracy_Score: ',accuracy_score(y_train,predict)) # score  - 0.842 # Good Score


predict = DTC.predict(X_test_w2v)
print('Confusion Metrics')
print(confusion_matrix(y_test,predict)) # confusion matrix
print('Accuracy_Score: ',accuracy_score(y_test,predict)) # score  - 0.842 # Good Score
```




```
Confusion Metrics-Train
[[  207 10876]
 [   44 62069]]
```

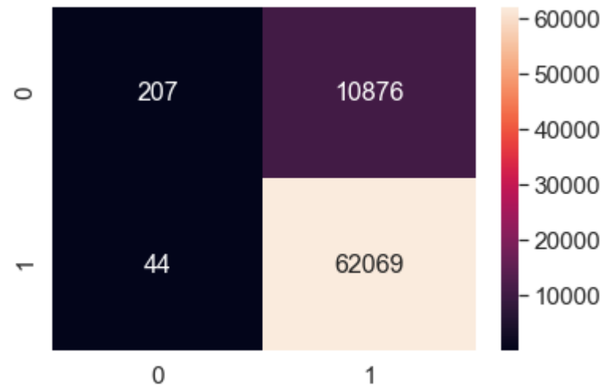
```
Accuracy_Score:  0.8508115197551779
Confusion Metrics
[[   25  5434]
 [   60 30533]]
Accuracy_Score:  0.8476090092089205
```

In [306]:
```
print('Train result- HeatMap')
array = [[207 ,10876],
         [44,62069]]

df_cm = pd.DataFrame(array, range(2), range(2))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16},fmt='d') # font size
```

```
Train result- HeatMap
```

Out[306]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x27d35643ec8>
```

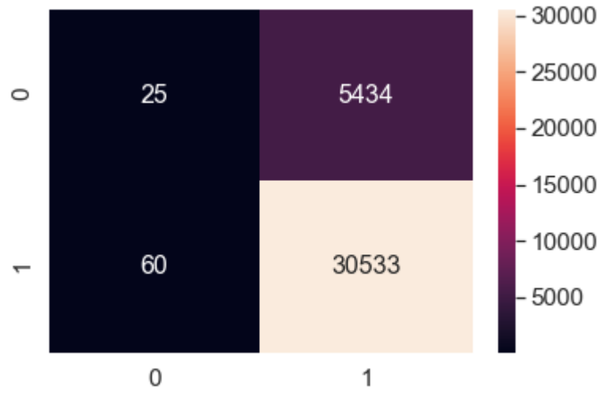


In [307]:
```
print('Test result- HeatMap')
array = [[25 ,5434],
         [60,30533]]

df_cm = pd.DataFrame(array, range(2), range(2))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16},fmt='d') # font size
```

```
Test result- HeatMap
```

Out[307]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x27d133f5c48>
```



In [308]:
```
from prettytable import PrettyTable
x1 = PrettyTable()
x1.field_names = ["Vectorizer", "Model", "Learning_rate",'N_estimator','AU_Score']
x1.add_row(['TFIDF_W2V','Gradient Descent DT','0.2','75','0.68'])
print(x1)
```

```
+------------+---------------------+---------------+-------------+----------+
| Vectorizer |        Model        | Learning_rate | N_estimator | AU_Score |
+------------+---------------------+---------------+-------------+----------+
| TFIDF_W2V  | Gradient Descent DT |      0.2      |      75     |   0.68   |
+------------+---------------------+---------------+-------------+----------+
```

In [ ]:
```

```