Donors Choose - Random Forest and GBDT

Objective : Predict whether teachers' project proposals are accepted

Importing packages

In [1]:

```
%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 sklearn.feature_extraction.text import TfidfTransformer
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, auc
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 gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

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

Reading the data

In [2]:

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

Project data

In [3]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("Attributes :", project_data.columns.values)
project_data.head(2)
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
Attributes : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_stat
e'
 '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']
```

Out[3]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	proj
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	

Handling Missing Value in "Teacher prefix" column

In [4]:

```
a = project_data['teacher_prefix'].mode().values
```

In [5]:

```
#Replacin nan with the most frequently occured value in that column
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(a[0])
```

Total number of null values in each column

In [6]:

```
#Total number of null values in each column
project_data.isnull().sum(axis = 0)
```

Out[6]:

```
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
```

Resource data

In [7]:

```
print("Number of data points in train data", resource_data.shape)
print('-'*50)
print("Attributes: ", resource_data.columns.values)
resource_data.head(2)
```

```
Number of data points in train data (1541272, 4)
--------------------------------------------------
Attributes:  ['id' 'description' 'quantity' 'price']
```

Out[7]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

Replacing date-time with date

In [8]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.colum


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]


project_data.head(10)
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_stat
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	C
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	U
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	C
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	G
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	W
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	C
81565	95963	p155767	e50367a62524e11fbd2dc79651b6df21	Mrs.	C
79026	139722	p182545	22460c54072bd0cf958cc8349fac8b8f	Ms.	C

	Unnamed: 0	id	teacher_id	teacher_prefix	school_stat
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	C
86551	114684	p049177	679f50f18ce50aabcc602d17f7627206	Mrs.	I

In [9]:

```
# https://matplotlib.org/gallery/pie_and_polar_charts/pie_and_donut_labels.html#sphx-glr-

y_value_counts = project_data['project_is_approved'].value_counts()
print("Number of projects thar are approved for funding ", y_value_counts[1], ", (", (y_v
print("Number of projects thar are not approved for funding ", y_value_counts[0], ", (",
```

```
Number of projects thar are approved for funding  92706 , ( 84.85830404217
927 %)
Number of projects thar are not approved for funding  16542 , ( 15.1416959
57820739 %)
```

NOTE: This is an imbalance dataset that containes 85% approved project's data and 15% not approved project's data

Sampling the data (50k random points)

In [10]:

```
approved_project=project_data[project_data["project_is_approved"]==1].sample(n=35000,rand
rejected_project=project_data[project_data["project_is_approved"]==0].sample(n=15000,rand
project_data=pd.concat([approved_project,rejected_project])
```

Preprocessing Categorical Data

Project Subject Categories

In [11]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/4736

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth",
        if 'The' in j.split(): # this will split each of the catogory based on space "Mat
            j=j.replace('The','') # if we have the words "The" we are going to replace it
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Mat
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing space
        temp = temp.replace('&','_') # we are replacing the & value into
        temp = temp.replace('-','_') # we are replacing - & _
        temp = temp.lower()
    cat_list.append(temp.strip())

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

from collections import Counter
my_counter = Counter()
for word in project_data['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]))
```

In [12]:

```
print(sorted_cat_dict)
```

```
{'warmth': 584, 'care_hunger': 584, 'history_civics': 2719, 'music_arts':
4792, 'appliedlearning': 5723, 'specialneeds': 6308, 'health_sports': 651
7, 'math_science': 19091, 'literacy_language': 23541}
```

In [13]:

```
print(project_data['clean_categories'])
```

```
51310                literacy_language
71601               warmth care_hunger
3220                     specialneeds
92053                   health_sports
68739                    math_science
                    ...
84723                   health_sports
62033                 appliedlearning
24070                literacy_language
43322    literacy_language specialneeds
56331        health_sports math_science
Name: clean_categories, Length: 50000, dtype: object
```

Project Subject Sub-Categories

Project Subject Sub-Categories

In [14]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/4736

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth",
        if 'The' in j.split(): # this will split each of the catogory based on space "Mat
            j=j.replace('The','') # if we have the words "The" we are going to replace it
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Mat
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing space
        temp = temp.replace('&','_')
        temp = temp.replace('-','_') # we are replacing - & _
        temp = temp.lower()
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
project_data.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 project_data['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]))
```

In [15]:

```
print(sorted_sub_cat_dict)
```

```
{'economics': 126, 'communityservice': 218, 'financialliteracy': 268, 'par
entinvolvement': 321, 'civics_government': 354, 'extracurricular': 375, 'f
oreignlanguages': 426, 'warmth': 584, 'care_hunger': 584, 'nutritioneducat
ion': 649, 'socialsciences': 883, 'performingarts': 903, 'charactereducati
on': 991, 'teamsports': 1087, 'other': 1136, 'college_careerprep': 1208,
'music': 1414, 'history_geography': 1476, 'esl': 1972, 'earlydevelopment':
1979, 'health_lifescience': 2011, 'gym_fitness': 2085, 'environmentalscien
ce': 2596, 'visualarts': 2980, 'health_wellness': 4595, 'appliedsciences':
5077, 'specialneeds': 6308, 'literature_writing': 10024, 'mathematics': 12
845, 'literacy': 15112}
```

Teacher Prefix

In [16]:

```
prefix = list(project_data['teacher_prefix'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/4736

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

prefix_list = []
for i in prefix:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth",
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Mat
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing space
        temp = temp.replace('.','')
        temp = temp.lower()
    prefix_list.append(temp.strip())

project_data['prefix_teacher'] = prefix_list
project_data.drop(['teacher_prefix'], axis=1, inplace=True)

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

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

In [17]:

```
print(sorted_prefix_dict)
```

```
{'dr': 8, 'teacher': 1119, 'mr': 4898, 'ms': 17913, 'mrs': 26062}
```

Project Grade categories

In [18]:

```
grades = list(project_data["project_grade_category"].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/4736

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

grades_list = []
for i in grades:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth",
        j = j.replace(' ','_') # we are placeing all the ' '(space) with ''(empty) ex:"Ma
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing space
        temp = temp.replace('-','_')
        temp = temp.lower()
    grades_list.append(temp.strip())

project_data['project_grade'] = grades_list
project_data.drop(["project_grade_category"], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['project_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]))
```

In [19]:

```
print(sorted_grade_dict)
```

```
{'grades_9_12': 5078, 'grades_6_8': 7751, 'grades_3_5': 16936, 'grades_pre
k_2': 20235}
```

Preprocessing Text Data

Project Essay

In [20]:

```
# 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 [21]:

```
# https://stackoverflow.com/a/47091490/4084039
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
```

In [22]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'h
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'the
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had'
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as',
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through',
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now'
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", '
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'was
            'won', "won't", 'wouldn', "wouldn't"]


from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('nannan', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = sent.lower()
    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_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████
██████| 50000/50000 [00:40<00:00, 1241.49it/s]
```

In [23]:

```
# placing the preprocessed essay into the dataframe
project_data['clean_essays'] = preprocessed_essays
project_data.drop(['essay'], axis=1, inplace=True)
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [24]:

```
#Printing random cleaned essay
project_data['clean_essays'].values[23]
```

Out[24]:

```
'students attend high poverty school students receive free breakfast major
ity free reduced lunch despite socioeconomic backgrounds students high ach
ievers progress throughout school years becomes evident successfully closi
ng socioeconomic achievement gap engaged curious always active students lo
ve learning world around meaningful ways build foundation future learning
no matter bad day may going always count little friends bring smile face b
ig books provide materials shared reading book activity sets storage cente
r allow students participate variety literacy experiences students chance
manipulate learn hands comprehension increases mouse paint allow content i
ntegrated across curriculum shared reading affords teachers opportunities
teach reading strategies larger group students big books activity sets all
ow provide instruction group setting well making materials accessible stud
ents use literacy centers build comprehension allow interact text practici
ng strategies'
```

Project title

In [25]:

```
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = sent.lower()
    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_titles.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████
████| 50000/50000 [00:01<00:00, 25914.44it/s]
```

In [26]:

```
# placing the preprocessed essay into the dataframe
project_data['clean_titles'] = preprocessed_titles
project_data.drop(['project_title'], axis=1, inplace=True)
```

In [27]:

```
#Printing random cleaned title
project_data['clean_titles'].values[12]
```

Out[27]:

```
'learning outside box'
```

Merging Price and quantity data to Project data (left joining price data)

In [28]:

```
# reference : https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-inde
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_inc
price_data.head(2)
```

Out[28]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [29]:

```
# join two dataframes(project_data and price_data) in python
# reference : https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

Splitting Data and Starifying the sampling

In [30]:

```
y = project_data['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
X = project_data

print(X.shape)
print(y.shape)
```

```
(50000, 15)
(50000,)
```

In [31]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_sp
#https://stackoverflow.com/questions/34842405/parameter-stratify-from-method-train-test-s
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, shuffle=Flase
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify = y) #
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratif

print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(22445, 15) (22445,)
(11055, 15) (11055,)
(16500, 15) (16500,)
```

Response coding with Laplace smooting categorical data

In [32]:

```
#dataframe of train data
train_data=pd.DataFrame(X_train)
train_data['project_is_approved']=y_train

#dataframe of cv data
cv_data=pd.DataFrame(X_cv)
cv_data['project_is_approved']=y_cv

#dataframe of test data
test_data=pd.DataFrame(X_test)
test_data['project_is_approved']=y_test

#approved data datframe
approved_train=train_data[train_data["project_is_approved"]==1]
#rejected data datframe
rejected_train=train_data[train_data["project_is_approved"]==0]

#Addivitve laplace smoothing : alpha 1

# response code with laplace smoothinh: (no. of occurance + 1*10)/(total occurances + 2*1
```

Function for response coding

In [33]:

```
def response_code(feature):
    #feature and number of occurance in total train data
    dic = (train_data[feature].value_counts()).to_dict()

    #feature and number of occurance in approved train data
    app_dict = (approved_train[feature].value_counts()).to_dict()

    #feature and number of occurance in rejected train data
    rej_dict = (rejected_train[feature].value_counts()).to_dict()

    dic_cv = (test_data[feature].value_counts()).to_dict()

    dic_test = (test_data[feature].value_counts()).to_dict()

    for f in dic.keys():
        for i in range(0,X_train.shape[0]):
            if f in app_dict.keys() and f in rej_dict.keys():
                occ = app_dict.get(f) + rej_dict.get(f)
                if(X_train[feature].values[i]==f and train_data['project_is_approved'].va
                    X_train[feature].values[i] = (app_dict.get(f) + 1*10)/(occ+2*1*10)
                elif(X_train[feature].values[i]==f and train_data['project_is_approved'].
                    X_train[feature].values[i] = (rej_dict.get(f) + 1*10)/(occ+2*1*10)
            elif f in app_dict.keys() and f not in rej_dict.keys():
                occ = app_dict.get(f)
                if(X_train[feature].values[i]==f and train_data['project_is_approved'].va
                    X_train[feature].values[i] = (app_dict.get(f) + 1*10)/(occ+2*1*10)

            elif f not in app_dict.keys() and f in rej_dict.keys():
                occ = rej_dict.get(f)
                if(X_train[feature].values[i]==f and train_data['project_is_approved'].va
                    X_train[feature].values[i] = (rej_dict.get(f) + 1*10)/(occ+2*1*10)



#iterating through whole train data and replace categories with their probability value
    for f in dic_cv.keys():
        for i in range(0,X_cv.shape[0]):
            if f in app_dict.keys() and f in rej_dict.keys():
                occ = app_dict.get(f) + rej_dict.get(f)
                if(X_cv[feature].values[i]==f and cv_data['project_is_approved'].values[i
                    X_cv[feature].values[i] = (app_dict.get(f) + 1*10)/(occ+len(dic_cv)*1
                elif(X_cv[feature].values[i]==f and cv_data['project_is_approved'].values
                    X_cv[feature].values[i] =(rej_dict.get(f) + 1*10)/(occ+2*1*10)
            elif f in app_dict.keys() and f not in rej_dict.keys():
                occ = app_dict.get(f)
                if(X_cv[feature].values[i]==f and cv_data['project_is_approved'].values[i
                    X_cv[feature].values[i] = (app_dict.get(f) + 1*10)/(occ+2*1*10)

            elif f not in app_dict.keys() and f in rej_dict.keys():
                occ = rej_dict.get(f)
                if(X_cv[feature].values[i]==f and cv_data['project_is_approved'].values[i
                    X_cv[feature].values[i] = (rej_dict.get(f) + 1*10)/(occ+2*1*10)

            elif f not in app_dict.keys() and f not in rej_dict.keys():
                X_cv[feature].values[i] = 0.5

    for f in dic_test.keys():
        for i in range(0,X_test.shape[0]):
            if f in app_dict.keys() and f in rej_dict.keys():
```

```
            occ = app_dict.get(f) + rej_dict.get(f)
            if(X_test[feature].values[i]==f and test_data['project_is_approved'].valu
                X_test[feature].values[i] = (app_dict.get(f) + 1*10)/(occ+2*1*10)
            elif(X_test[feature].values[i]==f and test_data['project_is_approved'].va
                X_test[feature].values[i] = (rej_dict.get(f) + 1*10)/(occ+2*1*10)
        elif f in app_dict.keys() and f not in rej_dict.keys():
            occ = app_dict.get(f)
            if(X_test[feature].values[i]==f and test_data['project_is_approved'].valu
                X_test[feature].values[i] = (app_dict.get(f) + 1*10)/(occ+2*1*10)

        elif f not in app_dict.keys() and f in rej_dict.keys():
            occ = rej_dict.get(f)
            if(X_test[feature].values[i]==f and test_data['project_is_approved'].valu
                X_test[feature].values[i] = (rej_dict.get(f) + 1*10)/(occ+2*1*10)

        elif f not in dic.keys():
            X_test[feature].values[i] = 0.5
```

Project grade

In [34]:

```
#Response encoding "Project grade"
response_code('project_grade')

grade_train = (X_train['project_grade'].values.reshape(-1,1))
grade_cv = (X_cv['project_grade'].values.reshape(-1,1))
grade_test = (X_test['project_grade'].values.reshape(-1,1))

print(grade_train.shape, y_train.shape)
print(grade_cv.shape, y_cv.shape)
print(grade_test.shape, y_test.shape)
```

```
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Teacher prefix

In [35]:

```
#Response encoding "Teacher prefix"
response_code('prefix_teacher')

prefix_train = (X_train['prefix_teacher'].values.reshape(-1,1))
prefix_cv = (X_cv['prefix_teacher'].values.reshape(-1,1))
prefix_test = (X_test['prefix_teacher'].values.reshape(-1,1))

print(prefix_train.shape, y_train.shape)
print(prefix_cv.shape, y_cv.shape)
print(prefix_test.shape, y_test.shape)
```

```
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

School state

In [36]:

```
#Response encoding "School state"
response_code('school_state')

state_train = (X_train['school_state'].values.reshape(-1,1))
state_cv = (X_cv['school_state'].values.reshape(-1,1))
state_test = (X_test['school_state'].values.reshape(-1,1))

print(state_train.shape, y_train.shape)
print(state_cv.shape, y_cv.shape)
print(state_test.shape, y_test.shape)
```

```
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Subject category

In [38]:

```
#Response encoding "School state"
response_code('clean_categories')

cat_train = (X_train['clean_categories'].values.reshape(-1,1))
cat_cv = (X_cv['clean_categories'].values.reshape(-1,1))
cat_test = (X_test['clean_categories'].values.reshape(-1,1))

print(cat_train.shape, y_train.shape)
print(cat_cv.shape, y_cv.shape)
print(cat_test.shape, y_test.shape)
```

```
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Subject subcategories

In [39]:

```
#Response encoding "School state"
response_code('clean_subcategories')

subcat_train = (X_train['clean_subcategories'].values.reshape(-1,1))
subcat_cv = (X_cv['clean_subcategories'].values.reshape(-1,1))
subcat_test = (X_test['clean_subcategories'].values.reshape(-1,1))

print(subcat_train.shape, y_train.shape)
print(subcat_cv.shape, y_cv.shape)
print(subcat_test.shape, y_test.shape)
```

```
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Numerical data

Price

In [40]:

```
price_train = X_train['price'].values.reshape(-1,1)
price_cv = X_cv['price'].values.reshape(-1,1)
price_test = X_test['price'].values.reshape(-1,1)

print(price_train.shape, y_train.shape)
print(price_cv.shape, y_cv.shape)
print(price_test.shape, y_test.shape)
```

```
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Resource Quantity

In [41]:

```
quantity_train = X_train['quantity'].values.reshape(-1,1)
quantity_cv = X_cv['quantity'].values.reshape(-1,1)
quantity_test = X_test['quantity'].values.reshape(-1,1)

print(quantity_train.shape, y_train.shape)
print(quantity_cv.shape, y_cv.shape)
print(quantity_test.shape, y_test.shape)
```

```
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Number of previously posted assignmnets by the teaxhers

In [42]:

```
number_projects_train = X_train['teacher_number_of_previously_posted_projects'].values.re
number_projects_cv = X_cv['teacher_number_of_previously_posted_projects'].values.reshape(
number_projects_test = X_test['teacher_number_of_previously_posted_projects'].values.resh

print(number_projects_train.shape, y_train.shape)
print(number_projects_cv.shape, y_cv.shape)
print(number_projects_test.shape, y_test.shape)
```

```
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Text Vectorization: Making data ready for models

BoW on Clean Essay

In [43]:

```
# We are considering only the words which appeared in at least 10 documents(rows or proje
vectorizer_bow_essay = CountVectorizer(min_df=10)
vectorizer_bow_essay.fit(X_train['clean_essays'].values)

X_train_essay_bow = vectorizer_bow_essay.transform(X_train['clean_essays'].values)
X_cv_essay_bow = vectorizer_bow_essay.transform(X_cv['clean_essays'].values)
X_test_essay_bow = vectorizer_bow_essay.transform(X_test['clean_essays'].values)

print("After vectorizing")
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)
```

```
After vectorizing
(22445, 8746) (22445,)
(11055, 8746) (11055,)
(16500, 8746) (16500,)
```

BoW on Clean Title

In [44]:

```
vectorizer_bow_title = CountVectorizer(min_df=10)
vectorizer_bow_title.fit(X_train['clean_titles'].values)

X_train_titles_bow = vectorizer_bow_title.transform(X_train['clean_titles'].values)
X_cv_titles_bow = vectorizer_bow_title.transform(X_cv['clean_titles'].values)
X_test_titles_bow = vectorizer_bow_title.transform(X_test['clean_titles'].values)

print("After vectorizing")
print(X_train_titles_bow.shape, y_train.shape)
print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)
```

```
After vectorizing
(22445, 1129) (22445,)
(11055, 1129) (11055,)
(16500, 1129) (16500,)
```

Tfidf on Clean Essay

In [45]:

```
from sklearn.feature_extraction.text import TfidfVectorizer


vectorizer_tfidf_essay = TfidfVectorizer(min_df=10)
vectorizer_tfidf_essay.fit(X_train['clean_essays'].values)

X_train_essay_tfidf = vectorizer_tfidf_essay.transform(X_train['clean_essays'])
X_cv_essay_tfidf = vectorizer_tfidf_essay.transform(X_cv['clean_essays'])
X_test_essay_tfidf = vectorizer_tfidf_essay.transform(X_test['clean_essays'])

print("After vectorizing")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
After vectorizing
(22445, 8746) (22445,)
(11055, 8746) (11055,)
(16500, 8746) (16500,)
```

Tfidf on Clean Title

In [46]:

```
vectorizer_tfidf_title = TfidfVectorizer(min_df=10)
vectorizer_tfidf_title.fit(X_train['clean_titles'].values)

X_train_title_tfidf = vectorizer_tfidf_title.transform(X_train['clean_titles'])
X_cv_title_tfidf = vectorizer_tfidf_title.transform(X_cv['clean_titles'])
X_test_title_tfidf = vectorizer_tfidf_title.transform(X_test['clean_titles'])

print("After vectorizing")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
```

```
After vectorizing
(22445, 1129) (22445,)
(11055, 1129) (11055,)
(16500, 1129) (16500,)
```

Avg W2V on Clean Essay

In [47]:

```
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')


words = []
for i in preprocessed_essays:
    words.extend(i.split(' '))

for i in preprocessed_titles:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and our coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pic

import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_courpus, f)

# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pic
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

```
Loading Glove Model

1917495it [06:14, 5122.33it/s]

Done. 1917495  words loaded!
all the words in the coupus 6982353
the unique words in the coupus 43045
The number of words that are present in both glove vectors and our coupus
39465 ( 91.683 %)
word 2 vec length 39465
```

In [48]:

```
train_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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 words_glove:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_essay_avg_w2v.append(vector)

print(len(train_essay_avg_w2v))
print(len(train_essay_avg_w2v[0]))


cv_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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 words_glove:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_essay_avg_w2v.append(vector)

print(len(cv_essay_avg_w2v))
print(len(cv_essay_avg_w2v[0]))


test_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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 words_glove:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_essay_avg_w2v.append(vector)

print(len(test_essay_avg_w2v))
print(len(test_essay_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██████| 22445/22445 [00:09<00:00, 2300.88it/s]

22445
300

100%|████████████████████████████████████████████████████████████████
██████| 11055/11055 [00:04<00:00, 2453.40it/s]

11055
300
```

```
100%|████████████████████████████████████████████████████████████
███████| 16500/16500 [00:06<00:00, 2452.80it/s]

16500
300
```

In [49]:

```
# Changing list to numpy arrays
train_essay_avg_w2v = np.array(train_essay_avg_w2v)
cv_essay_avg_w2v = np.array(cv_essay_avg_w2v)
test_essay_avg_w2v = np.array(test_essay_avg_w2v)

print("After vectorization")
print(train_essay_avg_w2v.shape, y_train.shape)
print(cv_essay_avg_w2v.shape, y_cv.shape)
print(test_essay_avg_w2v.shape, y_test.shape)
```

```
After vectorization
(22445, 300) (22445,)
(11055, 300) (11055,)
(16500, 300) (16500,)
```

Avg W2V on Clean Title

In [50]:

```
train_title_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_title_avg_w2v.append(vector)

print(len(train_title_avg_w2v))
print(len(train_title_avg_w2v[0]))


cv_title_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_title_avg_w2v.append(vector)

print(len(cv_title_avg_w2v))
print(len(cv_title_avg_w2v[0]))


test_title_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_title_avg_w2v.append(vector)

print(len(test_title_avg_w2v))
print(len(test_title_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
████| 22445/22445 [00:00<00:00, 42566.16it/s]

22445
300

100%|████████████████████████████████████████████████████████████████████
████| 11055/11055 [00:00<00:00, 47714.44it/s]

11055
300
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 16500/16500 [00:00<00:00, 49172.85it/s]

16500
300
```

In [51]:

```
# Changing list to numpy arrays
train_title_avg_w2v = np.array(train_title_avg_w2v)
cv_title_avg_w2v = np.array(cv_title_avg_w2v)
test_title_avg_w2v = np.array(test_title_avg_w2v)

print("After vectorization")
print(train_title_avg_w2v.shape, y_train.shape)
print(cv_title_avg_w2v.shape, y_cv.shape)
print(test_title_avg_w2v.shape, y_test.shape)
```

```
After vectorization
(22445, 300) (22445,)
(11055, 300) (11055,)
(16500, 300) (16500,)
```

Tfidf W2V on Clean essay

In [52]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['clean_essays'])
# 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 [53]:

```
train_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_essays']): # 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((sente
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gett
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_essay_tfidf_w2v.append(vector)

print(len(train_essay_tfidf_w2v))
print(len(train_essay_tfidf_w2v[0]))


cv_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_essays']): # 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((sente
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gett
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_essay_tfidf_w2v.append(vector)

print(len(cv_essay_tfidf_w2v))
print(len(cv_essay_tfidf_w2v[0]))


test_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_essays']): # 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((sente
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gett
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_essay_tfidf_w2v.append(vector)

print(len(test_essay_tfidf_w2v))
print(len(test_essay_tfidf_w2v[0]))
```

```
100%|██████████| 22445/22445 [01:03<00:00, 352.25it/s]
```

```
22445
300

100%|████████████████████████████████████████████████████████████
██████| 11055/11055 [00:31<00:00, 355.85it/s]

11055
300

100%|████████████████████████████████████████████████████████████
██████| 16500/16500 [00:46<00:00, 355.07it/s]

16500
300
```

In [54]:

```
# Changing list to numpy arrays
train_essay_tfidf_w2v = np.array(train_essay_tfidf_w2v)
cv_essay_tfidf_w2v = np.array(cv_essay_tfidf_w2v)
test_essay_tfidf_w2v = np.array(test_essay_tfidf_w2v)
```

Tfidf W2V on Clean Title

In [55]:

```
train_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles']): # 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((sente
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gett
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_title_tfidf_w2v.append(vector)

print(len(train_title_tfidf_w2v))
print(len(train_title_tfidf_w2v[0]))


cv_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles']): # 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((sente
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gett
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_title_tfidf_w2v.append(vector)

print(len(cv_title_tfidf_w2v))
print(len(cv_title_tfidf_w2v[0]))



test_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles']): # 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((sente
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gett
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_title_tfidf_w2v.append(vector)

print(len(test_title_tfidf_w2v))
print(len(test_title_tfidf_w2v[0]))
```

100%|

```
100%|██████| 22445/22445 [00:00<00:00, 23411.47it/s]

22445
300

100%|████████████████████████████████████████████████████████████
████| 11055/11055 [00:00<00:00, 23255.82it/s]

11055
300

100%|████████████████████████████████████████████████████████████
████| 16500/16500 [00:00<00:00, 23468.89it/s]

16500
300
```

In [56]:

```
# Changing list to numpy arrays
train_title_tfidf_w2v = np.array(train_title_tfidf_w2v)
cv_title_tfidf_w2v = np.array(cv_title_tfidf_w2v)
test_title_tfidf_w2v = np.array(test_title_tfidf_w2v)
```

Applying Decision Tree (2 Class Classification)

Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay(BOW)

Hstacking features

In [57]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :
X_train_bow = hstack((state_train, prefix_train, grade_train, cat_train, subcat_train, X_
X_cv_bow = hstack((state_cv,prefix_cv,grade_cv,cat_cv,subcat_cv,X_cv_essay_bow, quantity_
X_test_bow = hstack((state_test,prefix_test,grade_test,cat_test,subcat_test,X_test_essay_

print('Final matrix')
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
```

```
Final matrix
(22445, 9883) (22445,)
(11055, 9883) (11055,)
(16500, 9883) (16500,)
```

Hyperparameter tuning using simple for loop

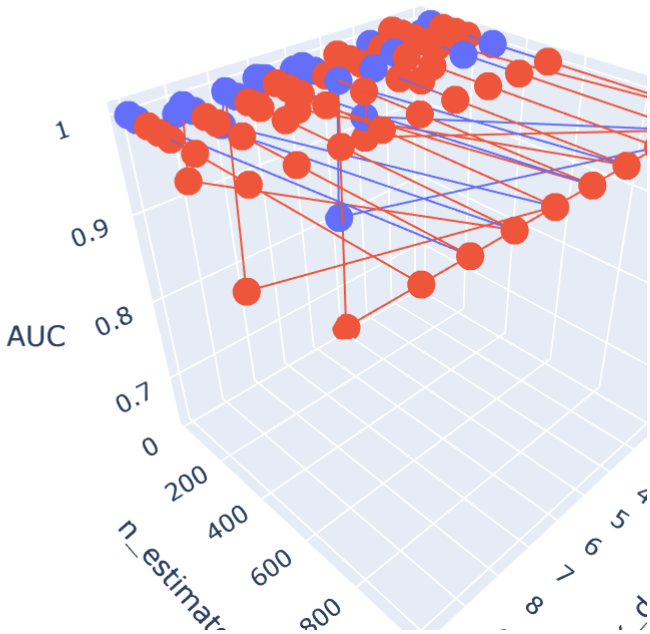
In [58]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifi
#https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.ht
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

train_auc = []
cv_auc = []


n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]
max_depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]


for i in (max_depth):
    for j in (n_estimators):

        rf_bow = RandomForestClassifier(n_estimators=j , criterion='gini', max_depth =i

        rf_bow.fit(X_train_bow, y_train)


        y_train_pred = rf_bow.predict_proba(X_train_bow)[:,1]
        y_cv_pred = rf_bow.predict_proba(X_cv_bow)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))

        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

3-D AUC plot

In [59]:

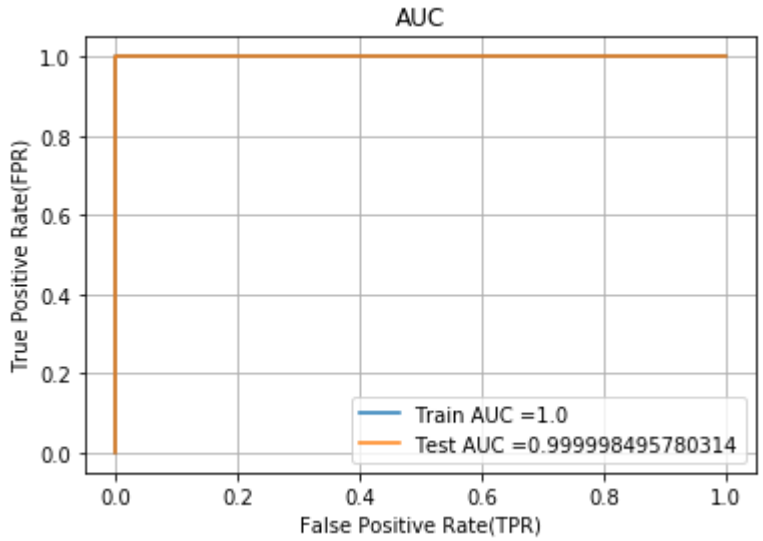
```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scat
from mpl_toolkits.mplot3d import Axes3D
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np

max_depth_val = []
n_estimators_val = []

for i in (max_depth):
    for j in (n_estimators):
        max_depth_val.append(i)
        n_estimators_val.append(j)

#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scat

trace1 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=train_auc, name = 'train auv')
trace2 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth_val'),
        yaxis = dict(title='n_estimators'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-auc error')
```



Training model with optimal value of hyperparameter

In [60]:

```
from sklearn.metrics import roc_curve, auc

rf_bow = RandomForestClassifier(n_estimators=200, criterion='gini', max_depth = 4, class_
rf_bow.fit(X_train_bow, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs
y_train_pred = rf_bow.predict_proba(X_train_bow)[:,1]
y_test_pred = rf_bow.predict_proba(X_test_bow)[:,1]


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Getting confusion metrix for both train and test set

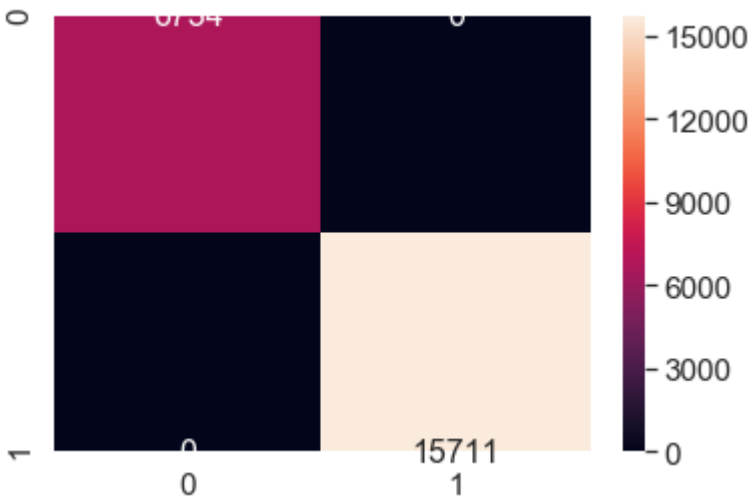
In [61]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

from mlxtend.plotting import plot_confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
```

In [62]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 0.479
Train confusion matrix
[[ 6734     0]
 [    0 15711]]
```



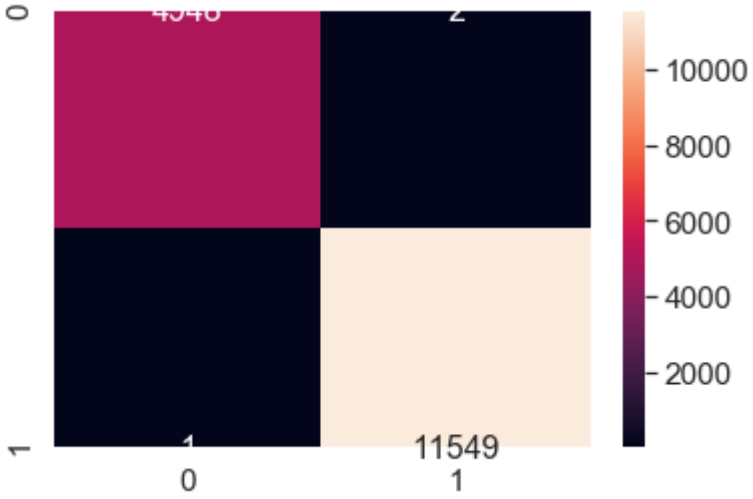
In [63]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 0.9995094144912327 for threshold 0.479
Test confusion matrix
```

Out[63]:

```
array([[ 4948,     2],
       [    1, 11549]], dtype=int64)
```



Evaluating model performance

In [64]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = rf_bow.predict(X_test_bow)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 98.406%
Precision on test set: 1.000
Recall on test set: 0.977
F1-Score on test set: 0.988
```

Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay(TFIDF)

Hstacking features

In [65]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :
X_train_tfidf = hstack((state_train, prefix_train, grade_train, cat_train, subcat_train,X
X_cv_tfidf = hstack((state_cv, prefix_cv, grade_cv, cat_cv, subcat_cv, X_cv_essay_tfidf,
X_test_tfidf = hstack((state_test, prefix_test, grade_test, cat_test, subcat_test,X_test_

print('Final matrix')
print(X_train_tfidf.shape, y_train.shape)
print(X_cv_tfidf.shape, y_cv.shape)
print(X_test_tfidf.shape, y_test.shape)
```

```
Final matrix
(22445, 9883) (22445,)
(11055, 9883) (11055,)
(16500, 9883) (16500,)
```

Hyperparameter tuning

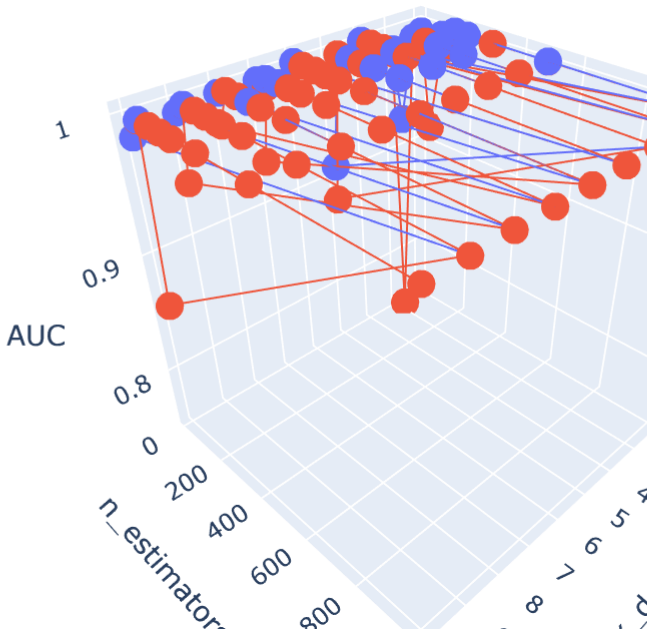
In [66]:

```
train_auc = []
cv_auc = []

n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]
max_depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]


for i in (max_depth):
    for j in (n_estimators):

        rf_tfidf = RandomForestClassifier(n_estimators=j , criterion='gini', max_depth =

        rf_tfidf.fit(X_train_tfidf, y_train)


        y_train_pred = rf_tfidf.predict_proba(X_train_tfidf)[:,1]
        y_cv_pred = rf_tfidf.predict_proba(X_cv_tfidf)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

3-D AUC plot

In [67]:

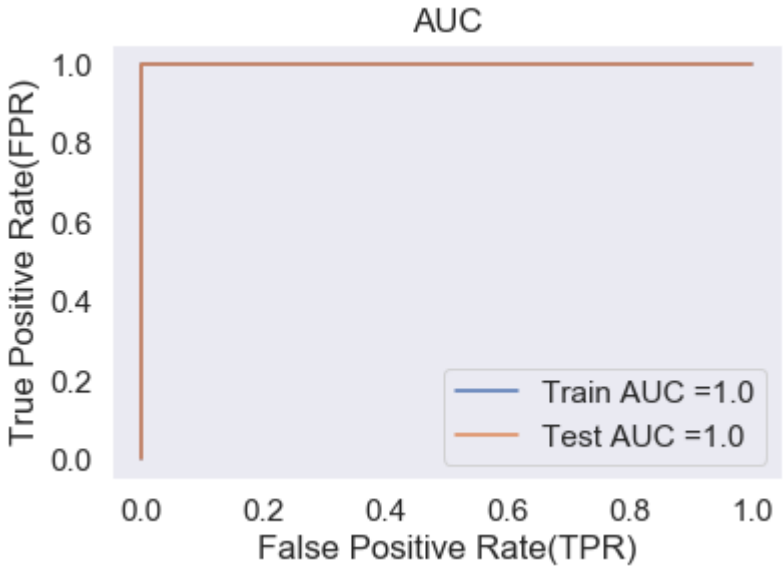
```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scat

trace1 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=train_auc, name = 'train auv')
trace2 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth_val'),
        yaxis = dict(title='n_estimators_val'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-auc error')
```



Training model on the best hyperparameters

In [68]:

```
from sklearn.metrics import roc_curve, auc

rf_tfidf = RandomForestClassifier(n_estimators= 300, criterion='gini', max_depth = 10 , c
rf_tfidf.fit(X_train_tfidf, y_train)

y_train_pred = rf_tfidf.predict_proba(X_train_tfidf)[:,1]
y_test_pred = rf_tfidf.predict_proba(X_test_tfidf)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

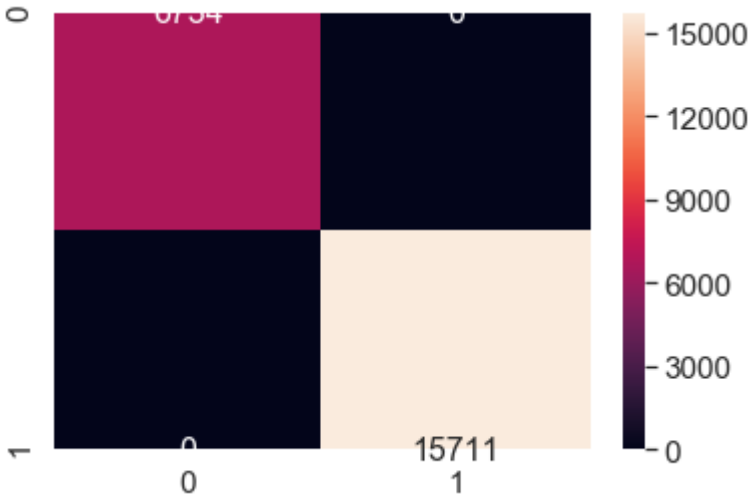


Confusion matrix

In [69]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 0.499
Train confusion matrix
[[ 6734     0]
 [    0 15711]]
```



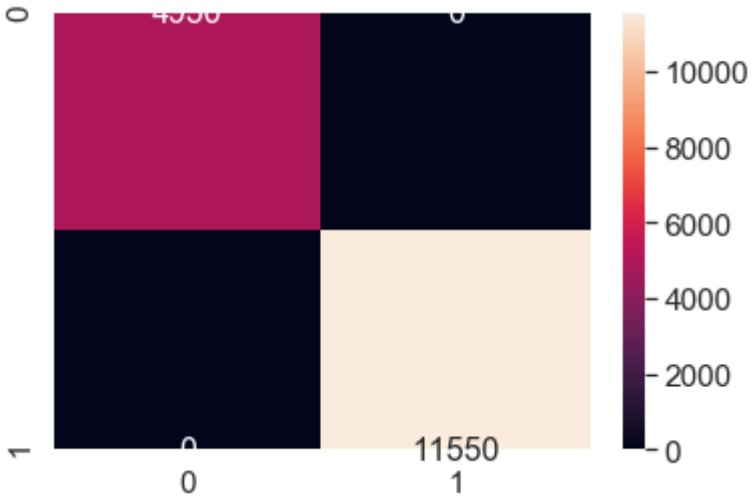
In [70]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 0.462
Test confusion matrix
```

Out[70]:

```
array([[ 4950,     0],
       [    0, 11550]], dtype=int64)
```



Evaluating model performance

In [71]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = rf_tfidf.predict(X_test_tfidf)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 99.794%
Precision on test set: 1.000
Recall on test set: 0.997
F1-Score on test set: 0.999
```

Set 3: categorical, numerical features + project_title(AVG W2V)+

preprocessed_eassay (AVG W2V) - 30K points

Hstacking features

In [72]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :
X_train_avg = np.hstack((train_essay_avg_w2v, quantity_train, train_title_avg_w2v, price_
X_cv_avg = np.hstack((cv_essay_avg_w2v, quantity_cv, cv_title_avg_w2v, price_cv, number_p
X_test_avg = np.hstack(( subcat_test,test_essay_avg_w2v, quantity_test,test_title_avg_w2v

#Considering only 30K datapoints

X_train_avg = X_train_avg[0:18000,:]
X_cv_avg = X_cv_avg[0:4000,:]
X_test_avg = X_test_avg[0:8000,:]
y_train_avg  = y_train[0:18000]
y_cv_avg  = y_cv[0:4000]
y_test_avg  = y_test[0:8000]

print('Final matrix')
print(X_train_avg.shape, y_train_avg.shape)
print(X_cv_avg.shape, y_cv_avg.shape)
print(X_test_avg.shape, y_test_avg.shape)
```

```
Final matrix
(18000, 608) (18000,)
(4000, 608) (4000,)
(8000, 608) (8000,)
```

Hyperparameter Tuning

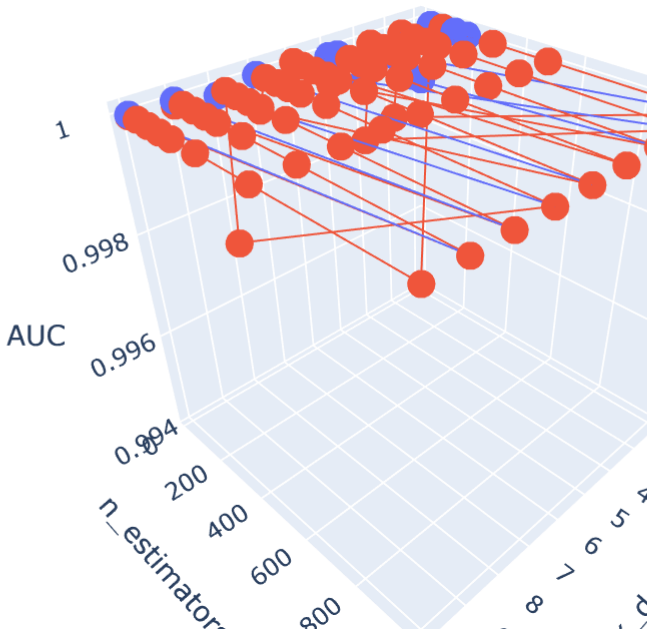
In [73]:

```
train_auc = []
cv_auc = []

n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]
max_depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]


for i in (max_depth):
    for j in (n_estimators):

        rf_avg = RandomForestClassifier(n_estimators=j , criterion='gini', max_depth =i ,

        rf_avg.fit(X_train_avg, y_train_avg)


        y_train_pred = rf_avg.predict_proba(X_train_avg)[:,1]
        y_cv_pred = rf_avg.predict_proba(X_cv_avg)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train_avg,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv_avg, y_cv_pred))
```

3-D AUC plot

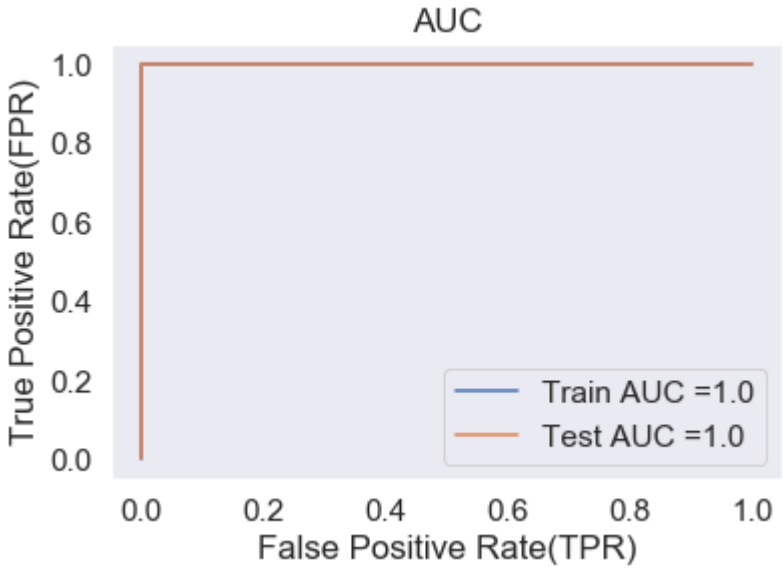
In [74]:

```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scat

trace1 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=train_auc, name = 'train auv')
trace2 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth_val'),
        yaxis = dict(title='n_estimators_val'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-auc error')
```



In [121]:

```
from sklearn.metrics import roc_curve, auc

rf_avg = RandomForestClassifier(n_estimators= 300, criterion='gini', max_depth =2 , class

rf_avg.fit(X_train_avg, y_train_avg)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs

y_train_pred = rf_avg.predict_proba(X_train_avg)[:,1]
y_test_pred = rf_avg.predict_proba(X_test_avg)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_avg, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_avg, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

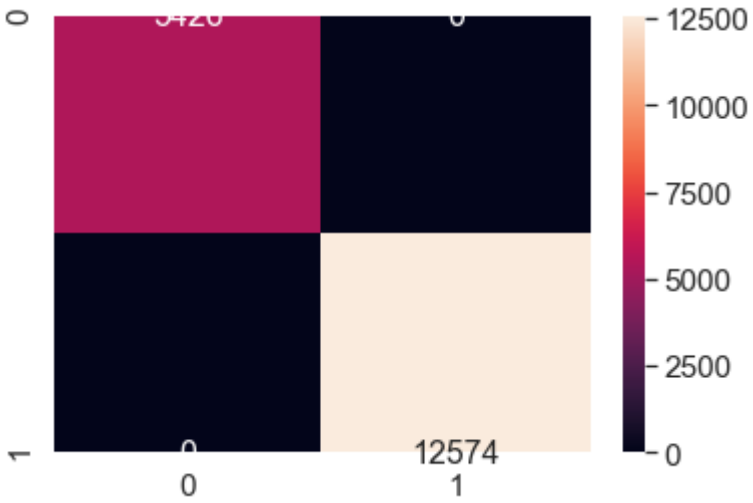


Confusion matrix

In [82]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train_avg, predict_with_best_t(y_train_pred, be
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train_avg, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 0.541
Train confusion matrix
[[ 5426     0]
 [    0 12574]]
```



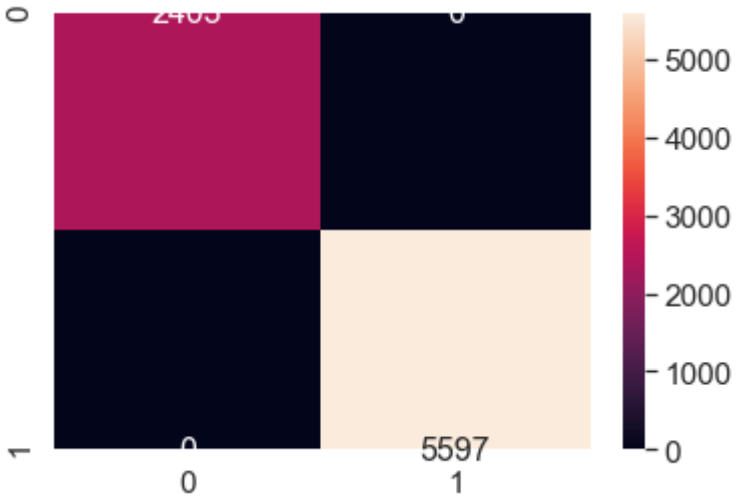
In [83]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test_avg, predict_with_best_t(y_test_pred, best
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test_avg, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 0.531
Test confusion matrix
```

Out[83]:

```
array([[2403,    0],
       [   0, 5597]], dtype=int64)
```



Evaluating model performance

In [84]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = rf_avg.predict(X_test_avg)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test_avg, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test_avg, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test_avg, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test_avg, y_pred_new)))
```

```
Accuracy on test set: 100.000%
Precision on test set: 1.000
Recall on test set: 1.000
F1-Score on test set: 1.000
```

Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V) - 30K points

Hstacking features

In [85]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :
X_train_tfidf_w2v = np.hstack((state_train, prefix_train, grade_train, cat_train, subcat_
X_cv_tfidf_w2v = np.hstack((state_cv, prefix_cv, grade_cv, cat_cv, subcat_cv, cv_essay_tf
X_test_tfidf_w2v = np.hstack((state_test, prefix_test, grade_test, cat_test, subcat_test,

#Considering only 30K datapoint
X_train_tfidf_w2v = X_train_avg[0:18000,:]
X_cv_tfidf_w2v = X_cv_avg[0:4000,:]
X_test_tfidf_w2v = X_test_avg[0:8000,:]
y_train_tfidf_w2v  = y_train[0:18000]
y_cv_tfidf_w2v  = y_cv[0:4000]
y_test_tfidf_w2v  = y_test[0:8000]

print('Final matrix')
print(X_train_tfidf_w2v.shape, y_train_tfidf_w2v.shape)
print(X_cv_tfidf_w2v.shape, y_cv_tfidf_w2v.shape)
print(X_test_tfidf_w2v.shape, y_test_tfidf_w2v.shape)
```

```
Final matrix
(18000, 608) (18000,)
(4000, 608) (4000,)
(8000, 608) (8000,)
```

Hyperparameter tuning

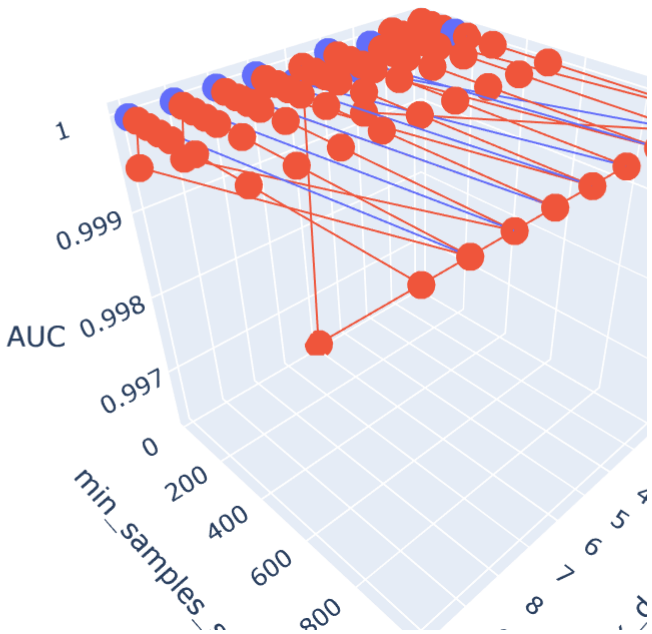
In [88]:

```
train_auc = []
cv_auc = []

n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]
max_depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]


for i in (max_depth):
    for j in (n_estimators):

        rf_tfidf_w2v = RandomForestClassifier(n_estimators=j , criterion='gini', max_dept

        rf_tfidf_w2v.fit(X_train_tfidf_w2v, y_train_tfidf_w2v)

        y_train_pred = rf_tfidf_w2v.predict_proba(X_train_tfidf_w2v)[:,1]
        y_cv_pred = rf_tfidf_w2v.predict_proba(X_cv_tfidf_w2v)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train_tfidf_w2v,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv_tfidf_w2v, y_cv_pred))
```

3-D AUC plot

In [89]:

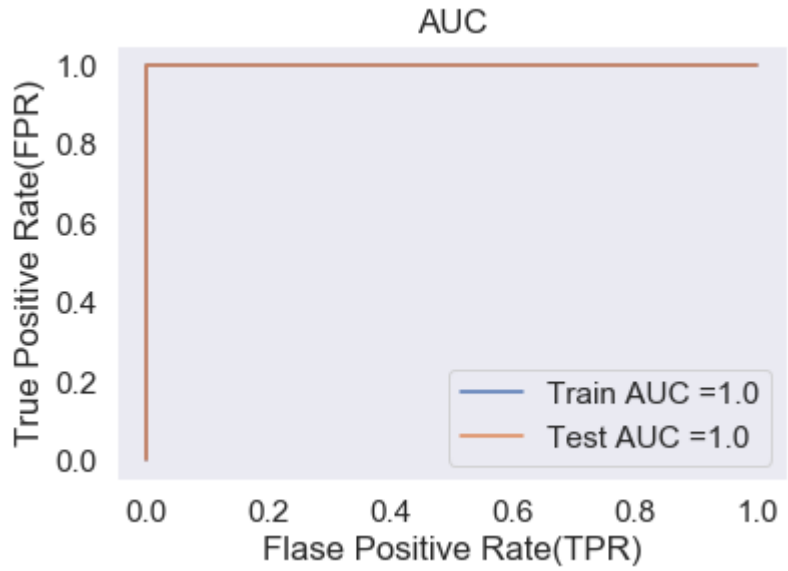
```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scat

trace1 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=train_auc, name = 'train auv')
trace2 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth_val'),
        yaxis = dict(title='min_samples_split_val'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-auc error')
```



Training model on the best hyperparameters

In [90]:

```
from sklearn.metrics import roc_curve, auc

rf_tfidf_w2v = RandomForestClassifier(n_estimators= 200 , criterion='gini', max_depth = 5

rf_tfidf_w2v.fit(X_train_tfidf_w2v, y_train_tfidf_w2v)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs

y_train_pred = rf_tfidf_w2v.predict_proba(X_train_tfidf_w2v)[:,1]
y_test_pred = rf_tfidf_w2v.predict_proba(X_test_tfidf_w2v)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_tfidf_w2v, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_tfidf_w2v, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Flase Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

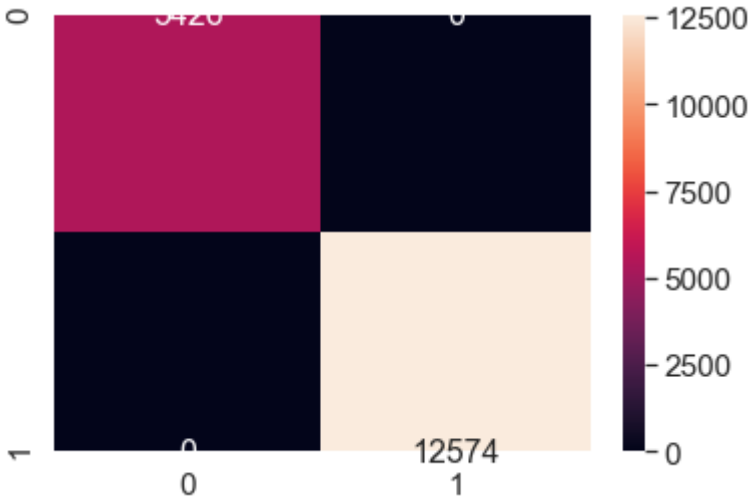


Confusion matrix

In [91]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train_tfidf_w2v, predict_with_best_t(y_train_pr
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train_tfidf_w2v, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 0.533
Train confusion matrix
[[ 5426     0]
 [    0 12574]]
```



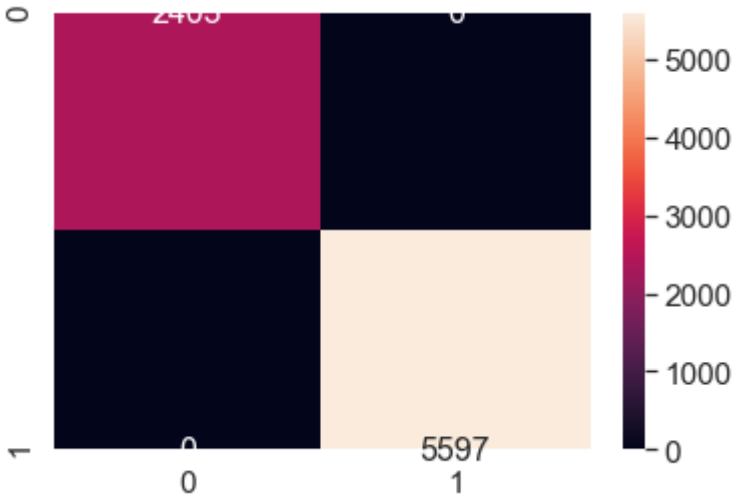
In [92]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test_tfidf_w2v, predict_with_best_t(y_test_pred
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test_tfidf_w2v, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 0.521
Test confusion matrix
```

Out[92]:

```
array([[2403,    0],
       [   0, 5597]], dtype=int64)
```



Evaluating Model performance

In [93]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = rf_tfidf_w2v.predict(X_test_tfidf_w2v)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test_tfidf_w2v, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test_tfidf_w2v, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test_tfidf_w2v, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test_tfidf_w2v, y_pred_new)))
```

```
Accuracy on test set: 100.000%
Precision on test set: 1.000
Recall on test set: 1.000
F1-Score on test set: 1.000
```

Conclusion: Random Forest models

In [124]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "max depth","n estimators", "Train AUC","Test AUC","F1 Sc

x.add_row(["BoW (set 1)",4,200, 1, 0.99,0.98,"98.4%"])
x.add_row(["TFIDF (set 2)",10,300, 1 , 1, 1,"100%"])
x.add_row(["AVG W2V (set 3)",2,200, 1, 1 ,1 ,"100%"])
x.add_row(["TFIDF W2V (set 4)",5,300, 1 , 1 , 1 ,"100%"])

print(x)
```

```
+------------------+-----------+--------------+-----------+----------+---
-------+----------------------+
|     Vectorizer    | max depth | n estimators | Train AUC | Test AUC | F1
Score | Accuracy on test set |
+------------------+-----------+--------------+-----------+----------+---
-------+----------------------+
|    BoW (set 1)    |     4     |     200      |     1     |   0.99   |
0.98   |        98.4%         |
|   TFIDF (set 2)   |     10    |     300      |     1     |    1     |
1      |         100%         |
|  AVG W2V (set 3)  |     2     |     200      |     1     |    1     |
1      |         100%         |
| TFIDF W2V (set 4) |     5     |     300      |     1     |    1     |
1      |         100%         |
+------------------+-----------+--------------+-----------+----------+---
-------+----------------------+
```

XGBoost: Boosting Ensemble Model

Set 1: categorical, numerical features + project_title(Bow)+ preprocessed_eassay (Bow)

Hyperparameter Tuning

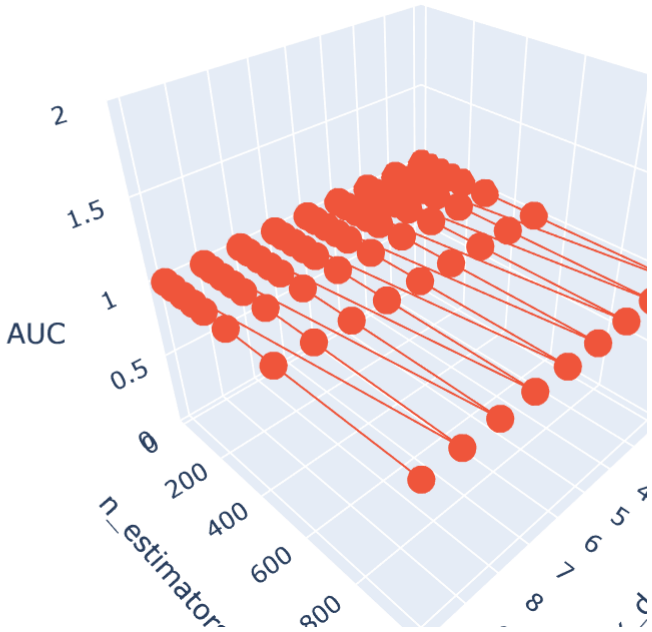
In [95]:

```
import xgboost as xgb

train_auc = []
cv_auc = []

n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]
max_depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]


for i in (max_depth):
    for j in (n_estimators):

        xg_bow = xgb.XGBClassifier(n_estimators=j, max_depth =i,class_weight='balanced')

        xg_bow.fit(X_train_bow,y_train)

        y_train_pred = xg_bow.predict_proba(X_train_bow)[:,1]
        y_cv_pred = xg_bow.predict_proba(X_cv_bow)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

3-D AUC plot

In [96]:

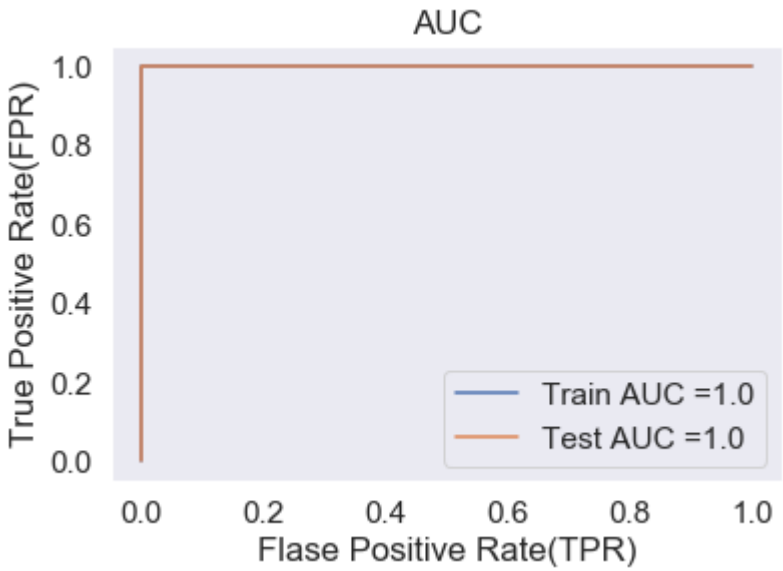
```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scat

trace1 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=train_auc, name = 'train auv')
trace2 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth_val'),
        yaxis = dict(title='n_estimators_val'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-auc error')
```



Training model on the best hyperparameters

In [97]:

```
xg_bow = xgb.XGBClassifier(n_estimators=200, max_depth =10,class_weight='balanced')

xg_bow.fit(X_train_bow,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs

y_train_pred = xg_bow.predict_proba(X_train_bow)[:,1]
y_test_pred = xg_bow.predict_proba(X_test_bow)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Flase Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

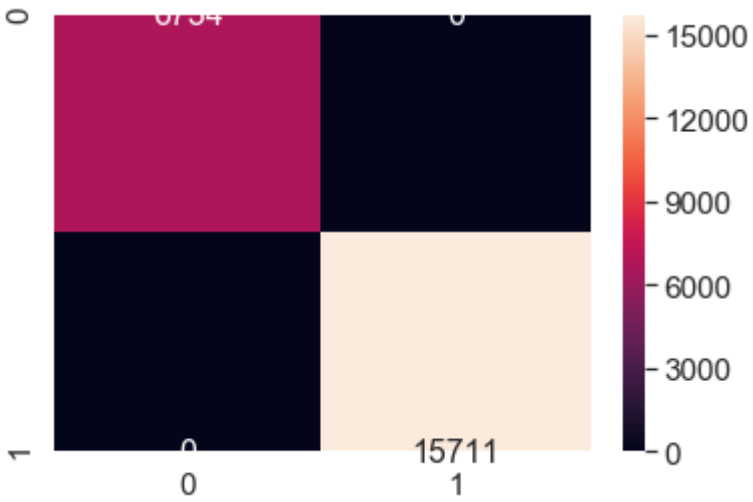


Confusion matrix

In [98]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Train confusion matrix
[[ 6734     0]
 [    0 15711]]
```



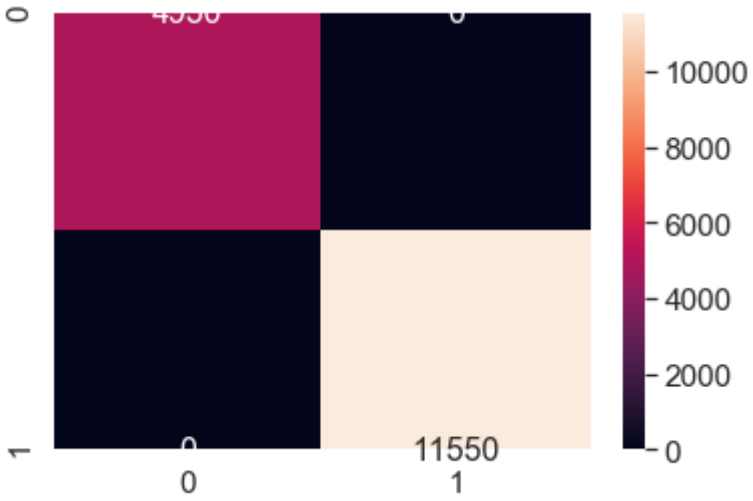
In [99]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Test confusion matrix
```

Out[99]:

```
array([[ 4950,     0],
       [    0, 11550]], dtype=int64)
```



Evaluating Model performance

In [100]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = xg_bow.predict(X_test_bow)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 100.000%
Precision on test set: 1.000
Recall on test set: 1.000
F1-Score on test set: 1.000
```

Set 2: categorical, numerical features + project_title(Tfidf)+ preprocessed_eassay (Tfidf)

Hyperparameter tuning

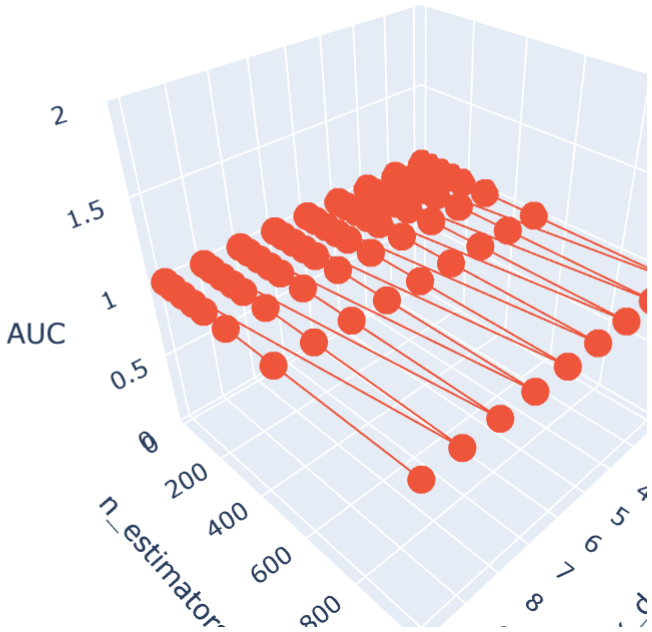
In [102]:

```
import xgboost as xgb

train_auc = []
cv_auc = []

n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]
max_depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]


for i in (max_depth):
    for j in (n_estimators):

        xg_tfidf = xgb.XGBClassifier(n_estimators=j, max_depth =i,class_weight='balanced'

        xg_tfidf.fit(X_train_tfidf,y_train)

        y_train_pred = xg_tfidf.predict_proba(X_train_tfidf)[:,1]
        y_cv_pred = xg_tfidf.predict_proba(X_cv_tfidf)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

3D scatter AUC plot

In [103]:

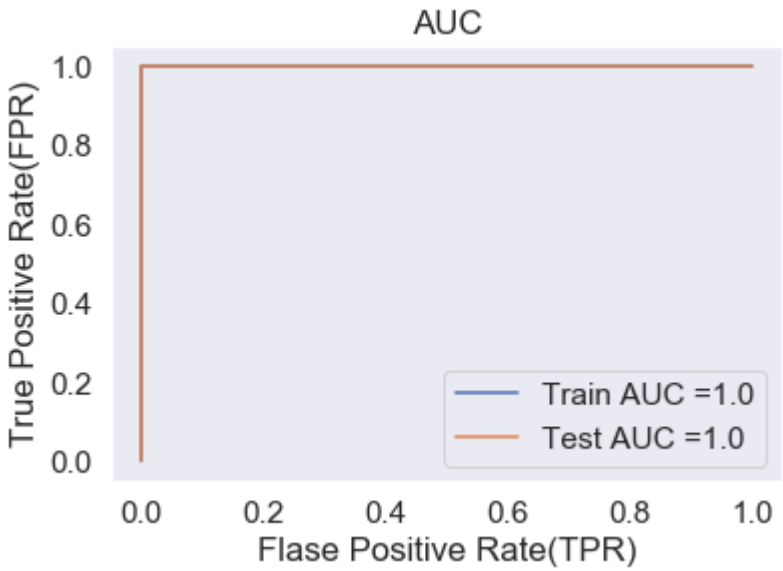
```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scat

trace1 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=train_auc, name = 'train auv')
trace2 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth_val'),
        yaxis = dict(title='n_estimators_val'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-auc error')
```



Training model on the best hyperparameters

In [104]:

```
xg_tfidf = xgb.XGBClassifier(n_estimators=300, max_depth =10,class_weight='balanced')

xg_tfidf.fit(X_train_tfidf,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs

y_train_pred = xg_tfidf.predict_proba(X_train_tfidf)[:,1]
y_test_pred = xg_tfidf.predict_proba(X_test_tfidf)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Flase Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

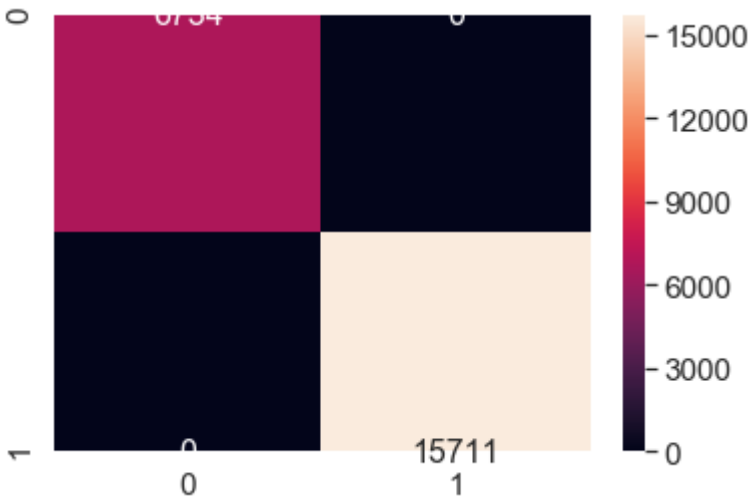


Confusion matrix

In [105]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Train confusion matrix
[[ 6734     0]
 [    0 15711]]
```



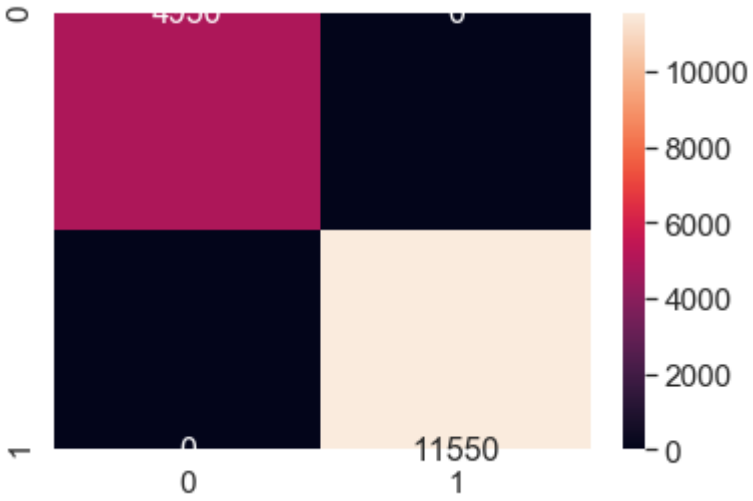
In [106]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Test confusion matrix
```

Out[106]:

```
array([[ 4950,     0],
       [    0, 11550]], dtype=int64)
```



Evaluating Model performance

In [107]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = xg_tfidf.predict(X_test_tfidf)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 100.000%
Precision on test set: 1.000
Recall on test set: 1.000
F1-Score on test set: 1.000
```

Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V) - 30K points

Hyperparameter tuning

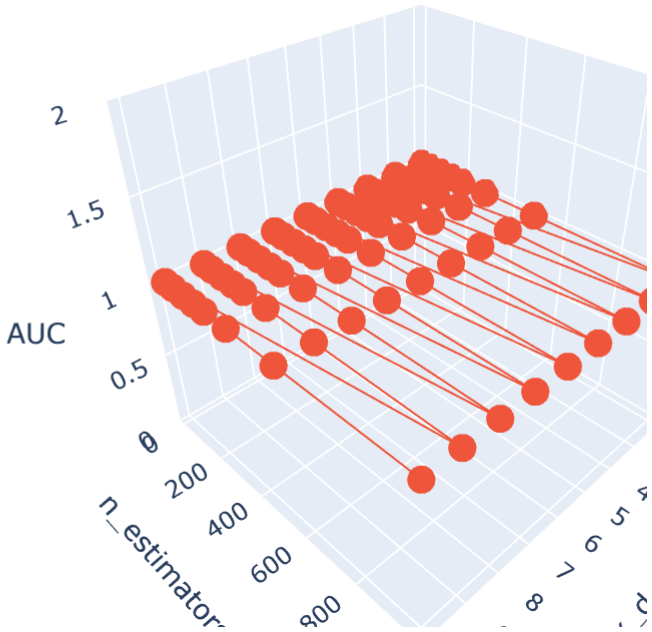
In [108]:

```
import xgboost as xgb

train_auc = []
cv_auc = []

n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]
max_depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]


for i in (max_depth):
    for j in (n_estimators):

        xg_avg = xgb.XGBClassifier(n_estimators=j, max_depth =i, class_weight='balanced')

        xg_avg.fit(X_train_avg,y_train_avg)

        y_train_pred = xg_avg.predict_proba(X_train_avg)[:,1]
        y_cv_pred = xg_avg.predict_proba(X_cv_avg)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train_avg,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv_avg, y_cv_pred))
```

3D Scatter AUC plot

In [109]:

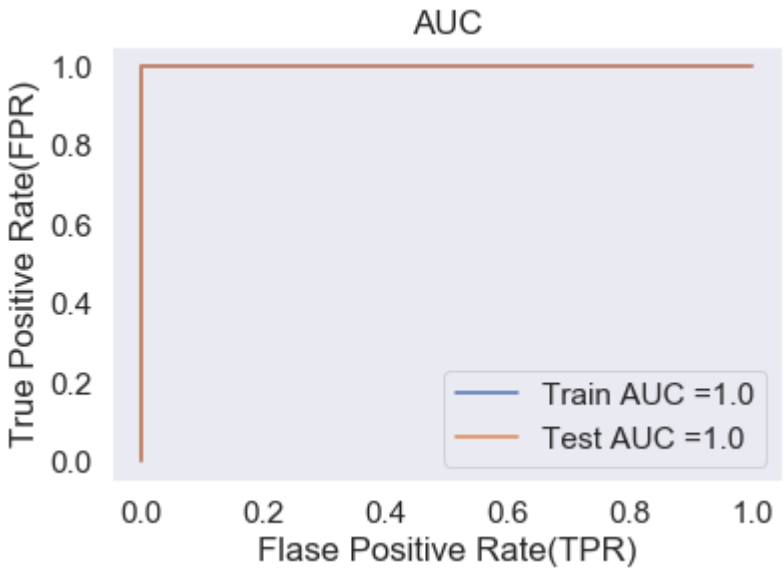
```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scat

trace1 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=train_auc, name = 'train auv')
trace2 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth_val'),
        yaxis = dict(title='n_estimators_val'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-auc error')
```



Training model with best set of hyperparameters

In [110]:

```
xg_avg = xgb.XGBClassifier(n_estimators=200, max_depth =10,class_weight='balanced')

xg_avg.fit(X_train_avg,y_train_avg)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs

y_train_pred = xg_avg.predict_proba(X_train_avg)[:,1]
y_test_pred = xg_avg.predict_proba(X_test_avg)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_avg, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_avg, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Flase Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

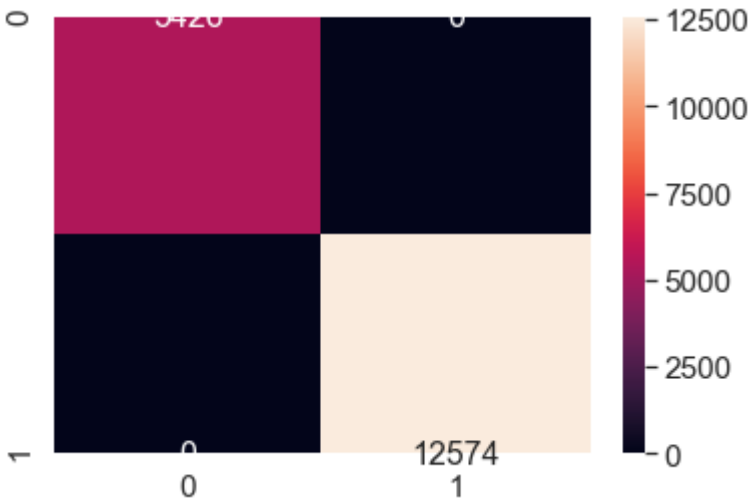


Confusion matrix

In [111]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train_avg, predict_with_best_t(y_train_pred, be
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train_avg, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Train confusion matrix
[[ 5426     0]
 [    0 12574]]
```



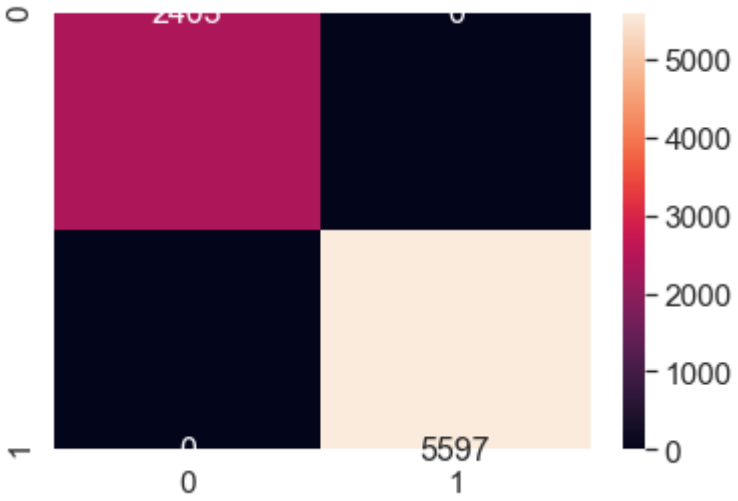
In [112]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test_avg, predict_with_best_t(y_test_pred, best
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test_avg, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Test confusion matrix
```

Out[112]:

```
array([[2403,    0],
       [   0, 5597]], dtype=int64)
```



Evaluating Model performance

In [113]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = xg_avg.predict(X_test_avg)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test_avg, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test_avg, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test_avg, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test_avg, y_pred_new)))
```

```
Accuracy on test set: 100.000%
Precision on test set: 1.000
Recall on test set: 1.000
F1-Score on test set: 1.000
```

Set 4: categorical, numerical features + project_title(Tfidf W2V)+ preprocessed_eassay (Tfidf W2V) - 30K points

Hyperparameter tuning

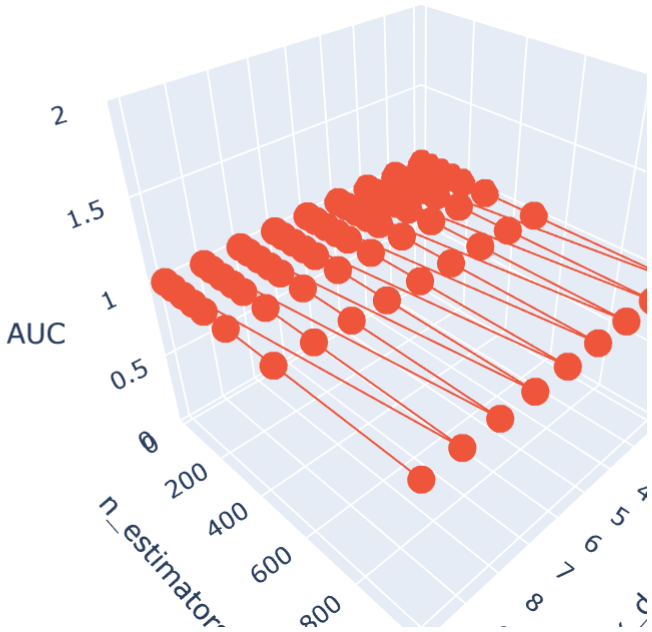
In [114]:

```
import xgboost as xgb

train_auc = []
cv_auc = []

n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]
max_depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]


for i in (max_depth):
    for j in (n_estimators):

        xg_tfidf_w2v = xgb.XGBClassifier(n_estimators=j, max_depth =i, class_weight='bala

        xg_tfidf_w2v.fit(X_train_tfidf_w2v,y_train_tfidf_w2v)

        y_train_pred = xg_tfidf_w2v.predict_proba(X_train_tfidf_w2v)[:,1]
        y_cv_pred = xg_tfidf_w2v.predict_proba(X_cv_tfidf_w2v)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train_tfidf_w2v,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv_tfidf_w2v, y_cv_pred))
```

3D Scatter AUC plot

In [115]:

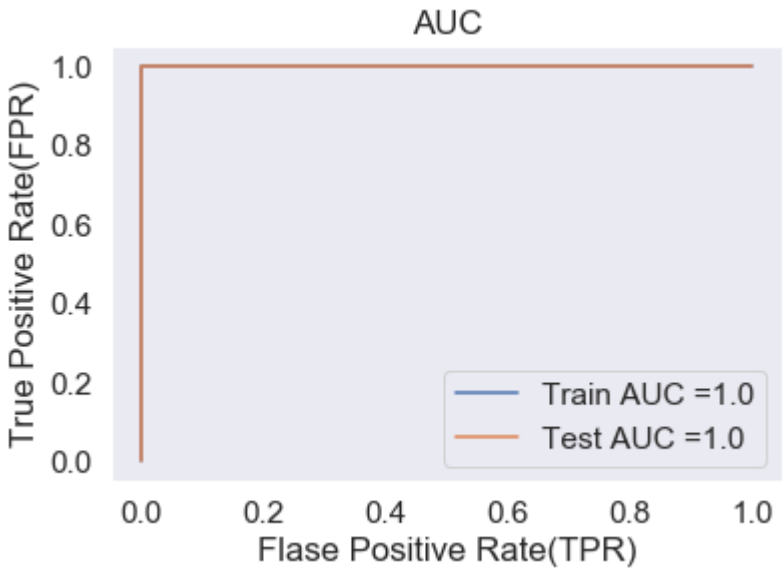
```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scat

trace1 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=train_auc, name = 'train auv')
trace2 = go.Scatter3d(x=max_depth_val,y=n_estimators_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='max_depth_val'),
        yaxis = dict(title='n_estimators_val'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-auc error')
```



Training model with best set of hyperparameters

In [116]:

```
xg_tfidf_w2v = xgb.XGBClassifier(n_estimators=150, max_depth =7,class_weight='balanced')

xg_tfidf_w2v.fit(X_train_tfidf_w2v,y_train_tfidf_w2v)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs

y_train_pred = xg_tfidf_w2v.predict_proba(X_train_tfidf_w2v)[:,1]
y_test_pred = xg_tfidf_w2v.predict_proba(X_test_tfidf_w2v)[:,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_tfidf_w2v, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_tfidf_w2v, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Flase Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

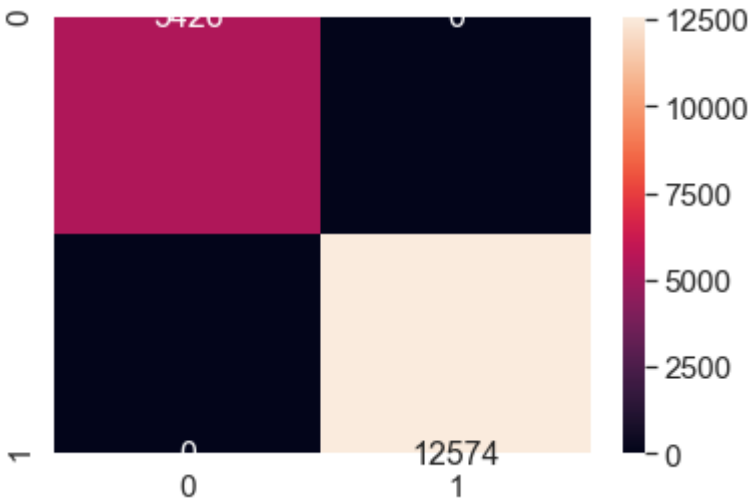


Confusion matrix

In [117]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train_tfidf_w2v, predict_with_best_t(y_train_pr
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train_tfidf_w2v, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Train confusion matrix
[[ 5426     0]
 [    0 12574]]
```



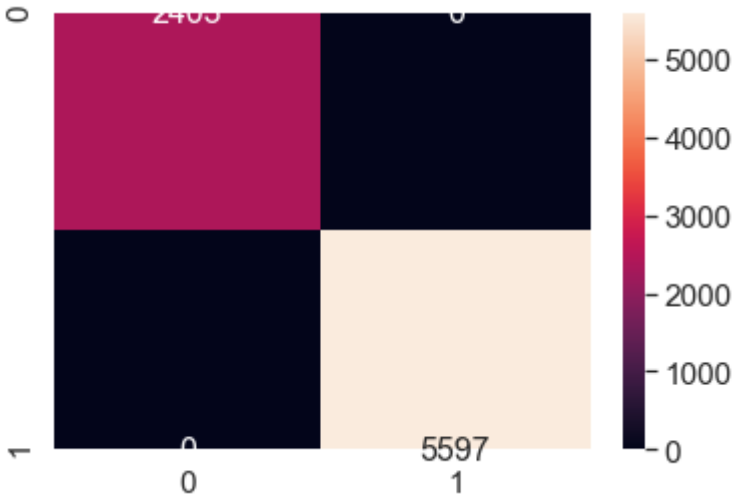
In [118]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test_tfidf_w2v, predict_with_best_t(y_test_pred
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test_tfidf_w2v, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Test confusion matrix
```

Out[118]:

```
array([[2403,    0],
       [   0, 5597]], dtype=int64)
```



Evaluating Model performance

In [119]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = xg_tfidf_w2v.predict(X_test_tfidf_w2v)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test_tfidf_w2v, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test_tfidf_w2v, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test_tfidf_w2v, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test_tfidf_w2v, y_pred_new)))
```

```
Accuracy on test set: 100.000%
Precision on test set: 1.000
Recall on test set: 1.000
F1-Score on test set: 1.000
```

Conclusion: GBDT models (XGBoost implementation)

In [125]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "max depth","min sample spit", "Train AUC","Test AUC","F1

x.add_row(["BoW (set 1)",10,200, 1, 1,1,"100%"])
x.add_row(["TFIDF (set 2)",10,300, 1, 1,1,"100%"])
x.add_row(["AVG W2V (set 3)",10,200, 1, 1,1,"100%"])
x.add_row(["TFIDF W2V (set 4)",7,150, 1, 1, 1,"100%"])

print(x)
```

```
+------------------+-----------+-----------------+-----------+----------+
----------+----------------------+
|     Vectorizer    | max depth | min sample spit | Train AUC | Test AUC |
F1 Score | Accuracy on test set |
+------------------+-----------+-----------------+-----------+----------+
----------+----------------------+
|    BoW (set 1)    |     10    |       200       |     1     |    1     |
1     |         100%         |
|   TFIDF (set 2)   |     10    |       300       |     1     |    1     |
1     |         100%         |
|  AVG W2V (set 3)  |     10    |       200       |     1     |    1     |
1     |         100%         |
| TFIDF W2V (set 4) |     7     |       150       |     1     |    1     |
1     |         100%         |
+------------------+-----------+-----------------+-----------+----------+
----------+----------------------+
```