DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located ([Two-letter U.S. postal code (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)](https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!</code`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*

Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

- **project_essay_1:** "Introduce us to your classroom"
- **project_essay_2:** "Tell us more about your students"
- **project_essay_3:** "Describe how your students will use the materials you're requesting"
- **project_essay_3:** "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

- **project_essay_1:** "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- **project_essay_2:** "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

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

1.1 Reading Data

In [2]:
```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [3]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix'
'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:
```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[4]:

	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

1.2 preprocessing of `project_subject_categories`

In [5]:
```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverf

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

1.3 preprocessing of `project_subject_subcategories`

In [6]:

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

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunge
    for j in i.split(','): # it will split it in three parts ["Math & Scien
        if 'The' in j.split(): # this will split each of the catogory based
            j=j.replace('The','') # if we have the words "The" we are going
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the
        temp = temp.replace('&','_')
    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/2289
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
```

1.3 Text preprocessing

In [7]:

```
# 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 [8]:
```
project_data.head(2)
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25

In [9]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [10]:
```python
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

```
My students are English learners that are working on English as their seco
nd or third languages. We are a melting pot of refugees, immigrants, and n
ative-born Americans bringing the gift of language to our school. \r\n\r\n
We have over 24 languages represented in our English Learner program with
students at every level of mastery.  We also have over 40 countries repres
ented with the families within our school.  Each student brings a wealth o
f knowledge and experiences to us that open our eyes to new cultures, beli
efs, and respect.\"The limits of your language are the limits of your worl
d.\"-Ludwig Wittgenstein  Our English learner's have a strong support syst
em at home that begs for more resources.  Many times our parents are learn
ing to read and speak English along side of their children.  Sometimes thi
s creates barriers for parents to be able to help their child learn phonet
ics, letter recognition, and other reading skills.\r\n\r\nBy providing the
se dvd's and players, students are able to continue their mastery of the E
nglish language even if no one at home is able to assist.  All families wi
th students within the Level 1 proficiency status, will be a offered to be
a part of this program.  These educational videos will be specially chosen
by the English Learner Teacher and will be sent home regularly to watch.
The videos are to help the child develop early reading skills.\r\n\r\nPare
nts that do not have access to a dvd player will have the opportunity to c
heck out a dvd player to use for the year.  The plan is to use these video
s and educational dvd's for the years to come for other EL students.\r\nna
nnan
==================================================
The 51 fifth grade students that will cycle through my classroom this year
all love learning, at least most of the time. At our school, 97.3% of the
students receive free or reduced price lunch. Of the 560 students, 97.3% a
re minority students. \r\nThe school has a vibrant community that loves to
get together and celebrate. Around Halloween there is a whole school parad
e to show off the beautiful costumes that students wear. On Cinco de Mayo
we put on a big festival with crafts made by the students, dances, and gam
es. At the end of the year the school hosts a carnival to celebrate the ha
rd work put in during the school year, with a dunk tank being the most pop
ular activity.My students will use these five brightly colored Hokki stool
s in place of regular, stationary, 4-legged chairs. As I will only have a
total of ten in the classroom and not enough for each student to have an i
ndividual one, they will be used in a variety of ways. During independent
reading time they will be used as special chairs students will each use on
occasion. I will utilize them in place of chairs at my small group tables
during math and reading times. The rest of the day they will be used by th
e students who need the highest amount of movement in their life in order
to stay focused on school.\r\n\r\nWhenever asked what the classroom is mis
sing, my students always say more Hokki Stools. They can't get their fill
of the 5 stools we already have. When the students are sitting in group wi
th me on the Hokki Stools, they are always moving, but at the same time do
ing their work. Anytime the students get to pick where they can sit, the H
okki Stools are the first to be taken. There are always students who head
over to the kidney table to get one of the stools who are disappointed as
there are not enough of them. \r\n\r\nWe ask a lot of students to sit for
7 hours a day. The Hokki stools will be a compromise that allow my student
s to do desk work and move at the same time. These stools will help studen
ts to meet their 60 minutes a day of movement by allowing them to activate
their core muscles for balance while they sit. For many of my students, th
ese chairs will take away the barrier that exists in schools for a child w
ho can't sit still.nannan
==================================================
How do you remember your days of school? Was it in a sterile environment w
```

In [11]:

```
# 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 [12]:

```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and
language delays, cognitive delays, gross/fine motor delays, to autism. The
y are eager beavers and always strive to work their hardest working past t
heir limitations. \r\n\r\nThe materials we have are the ones I seek out fo
r my students. I teach in a Title I school where most of the students rece
ive free or reduced price lunch.  Despite their disabilities and limitatio
ns, my students love coming to school and come eager to learn and explore.
Have you ever felt like you had ants in your pants and you needed to groov
e and move as you were in a meeting? This is how my kids feel all the tim
e. The want to be able to move as they learn or so they say.Wobble chairs
are the answer and I love then because they develop their core, which enha
nces gross motor and in Turn fine motor skills. \r\nThey also want to lear
n through games, my kids do not want to sit and do worksheets. They want t
o learn to count by jumping and playing. Physical engagement is the key to
our success. The number toss and color and shape mats can make that happe
n. My students will forget they are doing work and just have the fun a 6 y
ear old deserves.nannan
==================================================
```

In [13]:

```
# \r \n \t remove from string python: http://texthandler.com/info/remove-li
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and
language delays, cognitive delays, gross/fine motor delays, to autism. The
y are eager beavers and always strive to work their hardest working past t
heir limitations.     The materials we have are the ones I seek out for my
students. I teach in a Title I school where most of the students receive f
ree or reduced price lunch.  Despite their disabilities and limitations, m
y students love coming to school and come eager to learn and explore.Have
you ever felt like you had ants in your pants and you needed to groove and
move as you were in a meeting? This is how my kids feel all the time. The
want to be able to move as they learn or so they say.Wobble chairs are the
answer and I love then because they develop their core, which enhances gro
ss motor and in Turn fine motor skills.   They also want to learn through
games, my kids do not want to sit and do worksheets. They want to learn to
count by jumping and playing. Physical engagement is the key to our succes
s. The number toss and color and shape mats can make that happen. My stude
nts will forget they are doing work and just have the fun a 6 year old des
erves.nannan
```

In [14]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and
language delays cognitive delays gross fine motor delays to autism They ar
e eager beavers and always strive to work their hardest working past their
limitations The materials we have are the ones I seek out for my students
I teach in a Title I school where most of the students receive free or red
uced price lunch Despite their disabilities and limitations my students lo
ve coming to school and come eager to learn and explore Have you ever felt
like you had ants in your pants and you needed to groove and move as you w
ere in a meeting This is how my kids feel all the time The want to be able
to move as they learn or so they say Wobble chairs are the answer and I lo
ve then because they develop their core which enhances gross motor and in
Turn fine motor skills They also want to learn through games my kids do no
t want to sit and do worksheets They want to learn to count by jumping and
playing Physical engagement is the key to our success The number toss and
color and shape mats can make that happen My students will forget they are
doing work and just have the fun a 6 year old deserves nannan
```

In [15]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'y
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', '
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this',
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have'
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'be
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on'
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', '
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shou
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't",
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "sho
            'won', "won't", 'wouldn', "wouldn't"]
```

In [16]:
```
# Combining all the above stundents
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('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
project_data['essays']=preprocessed_essays
```

```
100%|██████████| 109248/109248 [02:17<00:00, 791.88it/s]
```

In [17]:

```
# after preprocesing
preprocessed_essays[20000]
```

Out[17]:

```
'my kindergarten students varied disabilities ranging speech language dela
ys cognitive delays gross fine motor delays autism they eager beavers alwa
ys strive work hardest working past limitations the materials ones i seek
students i teach title i school students receive free reduced price lunch
despite disabilities limitations students love coming school come eager le
arn explore have ever felt like ants pants needed groove move meeting this
kids feel time the want able move learn say wobble chairs answer i love de
velop core enhances gross motor turn fine motor skills they also want lear
n games kids not want sit worksheets they want learn count jumping playing
physical engagement key success the number toss color shape mats make happ
en my students forget work fun 6 year old deserves nannan'
```

1.3.1 Essays word count

In [18]:

```
essay_word_count=[]
for i in project_data['essays']:
    essay_word_count.append(len(i.split()))
project_data['essay_word_count']=essay_word_count
```

1.3.2 Computing Sentiment scores for essays

In [19]:

```
"""
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

import nltk
nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()
negative=[]
neutral=[]
positive=[]
compound=[]
for i in tqdm(project_data['essay']):
    negative.append(sid.polarity_scores(i)['neg'])
    neutral.append(sid.polarity_scores(i)['neu'])
    positive.append(sid.polarity_scores(i)['pos'])
    compound.append(sid.polarity_scores(i)['compound'])
project_data['negative']=negative
project_data['neutral']=neutral
project_data['positive']=positive
project_data['compound']=compound
print(project_data['negative'])
"""
```

Out[19]:

```
"\nimport nltk\nfrom nltk.sentiment.vader import SentimentIntensityAnalyze
r\n\nimport nltk\nnltk.download('vader_lexicon')\n\nsid = SentimentIntensi
tyAnalyzer()\nnegative=[]\nneutral=[]\npositive=[]\ncompound=[]\nfor i in
tqdm(project_data['essay']):\n    negative.append(sid.polarity_scores(i)['
neg'])\n    neutral.append(sid.polarity_scores(i)['neu'])\n    positive.ap
pend(sid.polarity_scores(i)['pos'])\n    compound.append(sid.polarity_scor
es(i)['compound'])\nproject_data['negative']=negative\nproject_data['neutr
al']=neutral\nproject_data['positive']=positive\nproject_data['compound']=
compound\nprint(project_data['negative'])\n"
```

In [20]:

```
import pickle

with open("Sneutral.dat", "rb") as f:
    project_data['neutral']=pickle.load(f)
with open("Snegative.dat", "rb") as f:
    project_data['negative']=pickle.load(f)
with open("Spositive.dat", "rb") as f:
    project_data['positive']=pickle.load(f)
with open("Scompound.dat", "rb") as f:
    project_data['compound']=pickle.load(f)
```

In [21]:

```
"""
import pickle
PIK = "Sneutral.dat"
d={}
with open(PIK, "wb") as f:
    pickle.dump(project_data['neutral'], f)
with open(PIK, "rb") as f:
    d['neutral']=pickle.load(f)
    print(d['neutral'])
"""
```

Out[21]:

```
'\nimport pickle\nPIK = "Sneutral.dat"\nd={}\nwith open(PIK, "wb") as f:\n
pickle.dump(project_data[\'neutral\'], f)\nwith open(PIK, "rb") as f:\n
d[\'neutral\']=pickle.load(f)\n    print(d[\'neutral\'])\n'
```

1.4 Preprocessing of `project_title`

In [22]:
```
# similarly you can preprocess the titles also
print(project_data['project_title'].values[0])
print("="*50)
print(project_data['project_title'].values[150])
print("="*50)
print(project_data['project_title'].values[1000])
print("="*50)
print(project_data['project_title'].values[20000])
print("="*50)
print(project_data['project_title'].values[99999])
print("="*50)
preprocessed_titles=[]
for sentance in tqdm(project_data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
project_data['project_title']=preprocessed_titles
print(project_data['project_title'].values[99999])
```

```
  4%|▍         | 3983/109248 [00:00<00:05, 19830.25it/s]

Educational Support for English Learners at Home
==================================================
More Movement with Hokki Stools
==================================================
Sailing Into a Super 4th Grade Year
==================================================
We Need To Move It While We Input It!
==================================================
Inspiring Minds by Enhancing the Educational Experience
==================================================

100%|██████████| 109248/109248 [00:05<00:00, 20418.72it/s]

inspiring minds enhancing educational experience
```

1.4.1 project title word count

In [23]:
```
title_word_count=[]
for i in project_data['project_title']:
    title_word_count.append(len(i.split()))
project_data['title_word_count']=title_word_count
```

1.5 Preparing data for models

In [24]:
```
project_data.columns
```

Out[24]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_ti
tle',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       'clean_categories', 'clean_subcategories', 'essay', 'essays',
       'essay_word_count', 'neutral', 'negative', 'positive', 'compound',
       'title_word_count'],
      dtype='object')
```

we are going to consider

```
   - school_state : categorical data
   - clean_categories : categorical data
   - clean_subcategories : categorical data
   - project_grade_category : categorical data
   - teacher_prefix : categorical data

   - project_title : text data
   - text : text data
   - project_resource_summary: text data (optinal)

   - quantity : numerical (optinal)
   - teacher_number_of_previously_posted_projects : numerical
   - price : numerical
```

1.5.1 Vectorizing Categorical data

- https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/ (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)

In [25]:
```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowerc
categories_one_hot = vectorizer.fit_transform(project_data['clean_categories
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning
', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (109248, 9)
```

In [26]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lo
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subca
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement
', 'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionE
ducation', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'C
haracterEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Music',
'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym
_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'Appli
edSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literac
y']
Shape of matrix after one hot encodig  (109248, 30)
```

In [27]:

```
"""
# you can do the similar thing with state, teacher_prefix and project_grade_
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(' ')#
newvocab=list(set(list(project_data['teacher_prefix'])))#remove duplicate fr
newvocab = list(filter(None,newvocab))#remove space from list
#print(newvocab)
vectorizer = CountVectorizer(vocabulary=newvocab, lowercase=False, binary=Tr
#for removing duplicates in the list 'teacher_prefix' i 1st converted it int
vectorizer.fit(project_data['teacher_prefix'].values.astype('U'))
print(vectorizer.get_feature_names())
teacher_one_hot = vectorizer.transform(project_data['teacher_prefix'].values
print("Shape of matrix after one hot encodig ",teacher_one_hot.shape)
"""
```

Out[27]:

```
'\n# you can do the similar thing with state, teacher_prefix and project_g
rade_category also\nproject_data[\'teacher_prefix\'] = project_data[\'teac
her_prefix\'].fillna(\' \')#replace nan with space\nnewvocab=list(set(list
(project_data[\'teacher_prefix\'])))#remove duplicate from list\nnewvocab
= list(filter(None,newvocab))#remove space from list\n#print(newvocab)\nve
ctorizer = CountVectorizer(vocabulary=newvocab, lowercase=False, binary=Tr
ue)\n#for removing duplicates in the list \'teacher_prefix\' i 1st convert
ed it into set and then again into list.\nvectorizer.fit(project_data[\'te
acher_prefix\'].values.astype(\'U\'))\nprint(vectorizer.get_feature_names
())\nteacher_one_hot = vectorizer.transform(project_data[\'teacher_prefi
x\'].values.astype(\'U\'))\nprint("Shape of matrix after one hot encodig
",teacher_one_hot.shape)\n'
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [28]:

```
# We are considering only the words which appeared in at least 10 documents

vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_bow.shape)
```

```
Shape of matrix after one hot encodig  (109248, 16623)
```

In [29]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
```

1.5.2.2 TFIDF vectorizer

In [30]:

```
'''
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
text_tfidf = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_tfidf.shape)
'''
```

Out[30]:

```
'\nfrom sklearn.feature_extraction.text import TfidfVectorizer\nvectorizer
= TfidfVectorizer(min_df=10)\ntext_tfidf = vectorizer.fit_transform(prepro
cessed_essays)\nprint("Shape of matrix after one hot encodig ",text_tfidf.
shape)\n'
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [31]:

```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/408
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')

# ============================
Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

# ============================

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

for i in preproced_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 c
      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/

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


'''
```

Out[31]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349
/4084039\ndef (https://stackoverflow.com/a/38230349/4084039\ndef) loadGlov
eModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveF
ile,\'r\', encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n
splitLine = line.split()\n        word = splitLine[0]\n        embedding =
np.array([float(val) for val in splitLine[1:]])\n        model[word] = emb
edding\n    print ("Done.",len(model)," words loaded!")\n    return model\
nmodel = loadGloveModel(\'glove.42B.300d.txt\')\n\n#
============================\nOutput:\n    \nLoading Glove Model\n1917495i
t [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
============================\n\nwords = []\nfor i in preproced_texts:\n
words.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n    words.exte
nd(i.split(\' \'))\nprint("all the words in the coupus", len(words))\nword
```

In [32]:

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

In [33]:

```
# average Word2Vec
# compute average word2vec for each review.
'''
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in tl
for sentence in tqdm(preprocessed_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 glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors.append(vector)

print(len(avg_w2v_vectors))
print(len(avg_w2v_vectors[0]))
'''
```

Out[33]:

```
'\navg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored
in this list\nfor sentence in tqdm(preprocessed_essays): # for each review
/sentence\n    vector = np.zeros(300) # as word vectors are of zero lengt
h\n    cnt_words =0; # num of words with a valid vector in the sentence/re
view\n    for word in sentence.split(): # for each word in a review/senten
ce\n        if word in glove_words:\n            vector += model[word]\n
cnt_words += 1\n    if cnt_words != 0:\n        vector /= cnt_words\n    a
vg_w2v_vectors.append(vector)\n\nprint(len(avg_w2v_vectors))\nprint(len(av
g_w2v_vectors[0]))\n'
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [34]:

```
'''
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_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())
'''
```

Out[34]:

```
'\n# S = ["abc def pqr", "def def def abc", "pqr pqr def"]\ntfidf_model =
TfidfVectorizer()\ntfidf_model.fit(preprocessed_essays)\n# we are converti
ng a dictionary with word as a key, and the idf as a value\ndictionary = d
ict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))\ntfidf_w
ords = set(tfidf_model.get_feature_names())\n'
```

In [35]:

```
'''
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(preprocessed_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/re
    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 t
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.sp
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

print(len(tfidf_w2v_vectors))
print(len(tfidf_w2v_vectors[0]))
'''
```

Out[35]:

```
'\n# average Word2Vec\n# compute average word2vec for each review.\ntfidf_
w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this
list\nfor sentence in tqdm(preprocessed_essays): # for each review/sentenc
e\n    vector = np.zeros(300) # as word vectors are of zero length\n    tf
_idf_weight =0; # num of words with a valid vector in the sentence/review\
n    for word in sentence.split(): # for each word in a review/sentence\n
if (word in glove_words) and (word in tfidf_words):\n            vec = mod
el[word] # getting the vector for each word\n            # here we are mul
tiplying idf value(dictionary[word]) and the tf value((sentence.count(wor
d)/len(sentence.split())))\n            tf_idf = dictionary[word]*(sentenc
e.count(word)/len(sentence.split())) # getting the tfidf value for each wo
rd\n            vector += (vec * tf_idf) # calculating tfidf weighted w2v\
n            tf_idf_weight += tf_idf\n    if tf_idf_weight != 0:\n
vector /= tf_idf_weight\n    tfidf_w2v_vectors.append(vector)\n\nprint(len
(tfidf_w2v_vectors))\nprint(len(tfidf_w2v_vectors[0]))\n'
```

In [36]:

```
# Similarly you can vectorize for title also
```

1.5.3 Vectorizing Numerical features

In [37]:

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

In [38]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generate
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
price_standardized = price_scalar.transform(project_data['price'].values.re
```

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

In [39]:
```
price_standardized
```

Out[39]:
```
array([[-0.3905327 ],
       [ 0.00239637],
       [ 0.59519138],
       ...,
       [-0.15825829],
       [-0.61243967],
       [-0.51216657]])
```

1.5.4 Merging all the above features

- we need to merge all the numerical vectors i.e catogorical, text, numerical vectors

In [40]:
```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
```

```
(109248, 9)
(109248, 30)
(109248, 16623)
(109248, 1)
```

In [41]:
```
# 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
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_star
X.shape
```

Out[41]:
```
(109248, 16663)
```

Computing Sentiment Scores

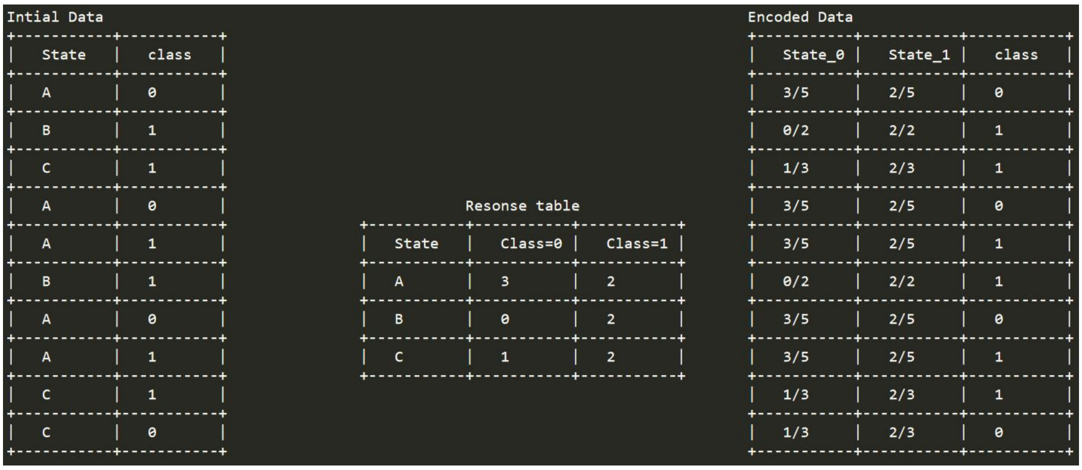
In [42]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

import nltk
nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach t
for learning my students learn in many different ways using all of our sense
of techniques to help all my students succeed students in my class come from
for wonderful sharing of experiences and cultures including native americans
learners which can be seen through collaborative student project based learn
in my class love to work with hands on materials and have many different opp
mastered having the social skills to work cooperatively with friends is a cr
montana is the perfect place to learn about agriculture and nutrition my stu
in the early childhood classroom i have had several kids ask me can we try c
and create common core cooking lessons where we learn important math and wri
food for snack time my students will have a grounded appreciation for the wo
of where the ingredients came from as well as how it is healthy for their bo
nutrition and agricultural cooking recipes by having us peel our own apples
and mix up healthy plants from our classroom garden in the spring we will al
shared with families students will gain math and literature skills as well a
nannan'
ss = sid.polarity_scores(for_sentiment)

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

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

```
/home/ankit/anaconda3/lib/python3.6/site-packages/nltk/twitter/__init__.p
y:20: UserWarning:

The twython library has not been installed. Some functionality from the tw
itter package will not be available.


[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /home/ankit/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

Assignment 9: RF and GBDT

Response Coding: Example



Intial Data

State	class
A	0
B	1
C	1
A	0
A	1
B	1
A	0
A	1
C	1
C	0

Resonse table

State	Class=0	Class=1
A	3	2
B	0	2
C	1	2

Encoded Data

State_0	State_1	class
3/5	2/5	0
0/2	2/2	1
1/3	2/3	1
3/5	2/5	0
3/5	2/5	1
0/2	2/2	1
3/5	2/5	0
3/5	2/5	1
1/3	2/3	1
1/3	2/3	0

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

1. **Apply both Random Forrest and GBDT on these feature sets**

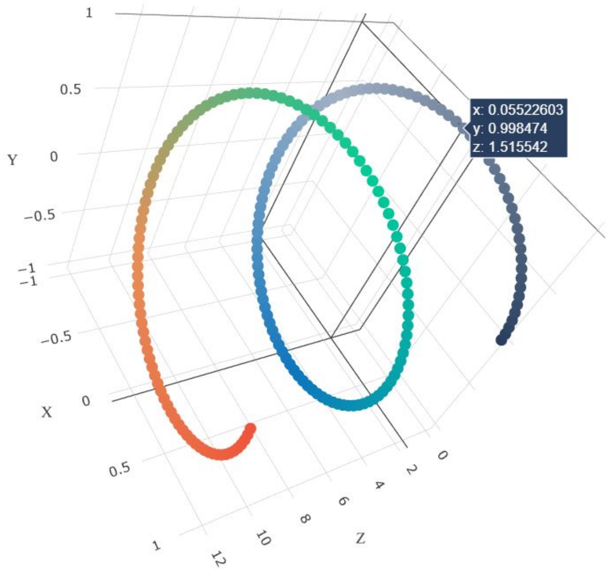
   - Set 1: categorical(instead of one hot encoding, try response coding (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): use probability values), numerical features + project_title(BOW) + preprocessed_eassay (BOW)
   - Set 2: categorical(instead of one hot encoding, try response coding (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
   - Set 3: categorical(instead of one hot encoding, try response coding (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): use probability values), numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
   - Set 4: categorical(instead of one hot encoding, try response coding (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

2. **The hyper paramter tuning (Consider any two hyper parameters preferably n_estimators, max_depth)**

   - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - find the best hyper paramter using k-fold cross validation/simple cross validation data
   - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task

3. **Representation of results**

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



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

or

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakage, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

2.Random Forest and GBDT

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

In [43]:

```
from sklearn import cross_validation
print(project_data.shape)
# create design matrix X and target vector y
#Xsp = (project_data.loc[:, project_data.columns != 'project_is_approved'])
ysp = (project_data['project_is_approved'] )
#print(Xsp.shape)
# split the data set into train and test
X_1, X_test, y_1, y_test = cross_validation.train_test_split(project_data, y

# split the train data set into cross validation train and cross validation
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_s
#print(X_tr.shape)
```

```
/home/ankit/anaconda3/lib/python3.6/site-packages/sklearn/cross_validatio
n.py:41: DeprecationWarning:

This module was deprecated in version 0.18 in favor of the model_selection
module into which all the refactored classes and functions are moved. Also
note that the interface of the new CV iterators are different from that of
this module. This module will be removed in 0.20.


(109248, 27)
```

In [44]:

```
print(X_tr.shape, y_tr.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(49041, 27) (49041,)
(24155, 27) (24155,)
(36052, 27) (36052,)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [45]:

```
X_train_one = X_tr.loc[X_tr['project_is_approved'] == 1]
X_train_zero = X_tr.loc[X_tr['project_is_approved'] == 0]
X_cv_one = X_cv.loc[X_cv['project_is_approved'] == 1]
X_cv_zero = X_cv.loc[X_cv['project_is_approved'] == 0]
X_test_one = X_test.loc[X_test['project_is_approved'] == 1]
X_test_zero = X_test.loc[X_test['project_is_approved'] == 0]
```

2.2.1 Clean categories

Train

Making a dictionary which contain key's as distinct elements present in the feature and value's as the no of times that element occurs in the feature

In [46]:
```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_tr['clean_categories']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_train_one['clean_categories']:
    clean_one[element] +=1       #counting occurence of element in feature
for element in X_train_zero['clean_categories']:
    clean_zero[element] +=1      #counting occurence of element in feature
#clean_one["History_Civics Warmth Care_Hunger"]
#list(clean_one.values())[1]
clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculatin
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_tr["clean_categories"]:            # mapping probabilities to or
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])

X_tr["clean_cat_0"] = clean_cat_0
X_tr["clean_cat_1"] = clean_cat_1
```

Cross validation

In [47]:
```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_cv['clean_categories']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_cv_one['clean_categories']:
    clean_one[element] +=1       #counting occurence of element in feature
for element in X_cv_zero['clean_categories']:
    clean_zero[element] +=1      #counting occurence of element in feature
#clean_one["History_Civics Warmth Care_Hunger"]
#list(clean_one.values())[1]
clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculatin
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_cv["clean_categories"]:            # mapping probabilities to or
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])

X_cv["clean_cat_0"] = clean_cat_0
X_cv["clean_cat_1"] = clean_cat_1
```

Test

In [48]:
```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_test['clean_categories']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_test_one['clean_categories']:
    clean_one[element] +=1       #counting occurence of element in feature
for element in X_test_zero['clean_categories']:
    clean_zero[element] +=1      #counting occurence of element in feature
#clean_one["History_Civics Warmth Care_Hunger"]
#list(clean_one.values())[1]
clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculatin
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_test["clean_categories"]:            # mapping probabilities to
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])

X_test["clean_cat_0"] = clean_cat_0
X_test["clean_cat_1"] = clean_cat_1
```

Normalization

In [49]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
Train_categ_0 = normalizer.fit_transform(X_tr["clean_cat_0"].values.reshape
Train_categ_1 = normalizer.fit_transform(X_tr["clean_cat_1"].values.reshape
CV_categ_0 = normalizer.transform(X_cv['clean_cat_0'].values.reshape(-1, 1)
CV_categ_1 = normalizer.transform(X_cv['clean_cat_1'].values.reshape(-1, 1)
Test_categ_0 = normalizer.transform(X_test['clean_cat_0'].values.reshape(-1
Test_categ_1 = normalizer.transform(X_test['clean_cat_1'].values.reshape(-1
```

2.2.2 Clean subcategories

Train

In [50]:

```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_tr['clean_subcategories']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_train_one['clean_subcategories']:
    clean_one[element] +=1       #counting occurence of element in feature
for element in X_train_zero['clean_subcategories']:
    clean_zero[element] +=1      #counting occurence of element in feature
#clean_one["History_Civics Warmth Care_Hunger"]
#list(clean_one.values())[1]
clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculating
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_tr["clean_subcategories"]:            # mapping probabilities to
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])

X_tr["sub_clean_cat_0"] = clean_cat_0
X_tr["sub_clean_cat_1"] = clean_cat_1
```

Cross Validation

In [51]:

```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_cv['clean_subcategories']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_cv_one['clean_subcategories']:
    clean_one[element] +=1       #counting occurence of element in feature
for element in X_cv_zero['clean_subcategories']:
    clean_zero[element] +=1      #counting occurence of element in feature
#clean_one["History_Civics Warmth Care_Hunger"]
#list(clean_one.values())[1]
clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculating
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_cv["clean_subcategories"]:            # mapping probabilities to
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])

X_cv["sub_clean_cat_0"] = clean_cat_0
X_cv["sub_clean_cat_1"] = clean_cat_1
```

Test

In [52]:
```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_test['clean_subcategories']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_test_one['clean_subcategories']:
    clean_one[element] +=1       #counting occurence of element in feature 
for element in X_test_zero['clean_subcategories']:
    clean_zero[element] +=1      #counting occurence of element in feature 
#clean_one["History_Civics Warmth Care_Hunger"]
#list(clean_one.values())[1]
clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculatin
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_test["clean_subcategories"]:           # mapping probabilities 
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])

X_test["sub_clean_cat_0"] = clean_cat_0
X_test["sub_clean_cat_1"] = clean_cat_1
```

Normalization

In [53]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
Train_sub_categ_0 = normalizer.fit_transform(X_tr["sub_clean_cat_0"].values
Train_sub_categ_1 = normalizer.fit_transform(X_tr["sub_clean_cat_1"].values
CV_sub_categ_0 = normalizer.transform(X_cv['sub_clean_cat_0'].values.reshape
CV_sub_categ_1 = normalizer.transform(X_cv['sub_clean_cat_1'].values.reshape
Test_sub_categ_0 = normalizer.transform(X_test['sub_clean_cat_0'].values.res
Test_sub_categ_1 = normalizer.transform(X_test['sub_clean_cat_1'].values.res
```

2.2.3 Teacher prefix

Train

In [54]:
```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_tr['teacher_prefix']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_train_one['teacher_prefix']:
    clean_one[element] +=1       #counting occurence of element in feature w
for element in X_train_zero['teacher_prefix']:
    clean_zero[element] +=1      #counting occurence of element in feature w

clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculating
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_tr["teacher_prefix"]:           # mapping probabilities to orig
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])
clean_one_prob[np.nan]=0
clean_zero_prob[np.nan]=0
X_tr["teacher_prefix_0"] = clean_cat_0
X_tr["teacher_prefix_1"] = clean_cat_1
```

Cross validation

In [55]:
```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_cv['teacher_prefix']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_cv_one['teacher_prefix']:
    clean_one[element] +=1       #counting occurence of element in feature w
for element in X_cv_zero['teacher_prefix']:
    clean_zero[element] +=1      #counting occurence of element in feature w
#clean_one["History_Civics Warmth Care_Hunger"]
#list(clean_one.values())[1]
clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculating
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_cv["teacher_prefix"]:           # mapping probabilities to orig
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])
clean_one_prob[np.nan]=0
clean_zero_prob[np.nan]=0
X_cv["teacher_prefix_0"] = clean_cat_0
X_cv["teacher_prefix_1"] = clean_cat_1
```

Test

In [56]:
```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_test['teacher_prefix']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_test_one['teacher_prefix']:
    clean_one[element] +=1       #counting occurence of element in feature
for element in X_test_zero['teacher_prefix']:
    clean_zero[element] +=1      #counting occurence of element in feature
#clean_one["History_Civics Warmth Care_Hunger"]
#list(clean_one.values())[1]
clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculatin
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_test["teacher_prefix"]:             # mapping probabilities to or
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])
clean_one_prob[np.nan]=0
clean_zero_prob[np.nan]=0
X_test["teacher_prefix_0"] = clean_cat_0
X_test["teacher_prefix_1"] = clean_cat_1
```

Normalization

In [57]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
Train_teach_0 = normalizer.fit_transform(X_tr["teacher_prefix_0"].values.re
Train_teach_1 = normalizer.fit_transform(X_tr["teacher_prefix_1"].values.re
CV_teach_0 = normalizer.transform(X_cv['teacher_prefix_0'].values.reshape(-
CV_teach_1 = normalizer.transform(X_cv['teacher_prefix_1'].values.reshape(-
Test_teach_0 = normalizer.transform(X_test['teacher_prefix_0'].values.resha
Test_teach_1 = normalizer.transform(X_test['teacher_prefix_1'].values.resha
```

2.2.4 Project grade

Train

In [58]:
```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_tr['project_grade_category']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_train_one['project_grade_category']:
    clean_one[element] +=1       #counting occurence of element in feature
for element in X_train_zero['project_grade_category']:
    clean_zero[element] +=1      #counting occurence of element in feature

clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculatin
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_tr["project_grade_category"]:            # mapping probabilities
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])

X_tr["project_grade_category_0"] = clean_cat_0
X_tr["project_grade_category_1"] = clean_cat_1
```

Cross Validation

In [59]:
```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_cv['project_grade_category']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_cv_one['project_grade_category']:
    clean_one[element] +=1       #counting occurence of element in feature
for element in X_cv_zero['project_grade_category']:
    clean_zero[element] +=1      #counting occurence of element in feature
#clean_one["History_Civics Warmth Care_Hunger"]
#list(clean_one.values())[1]
clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculatin
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_cv["project_grade_category"]:            # mapping probabilities
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])

X_cv["project_grade_category_0"] = clean_cat_0
X_cv["project_grade_category_1"] = clean_cat_1
```

Test

In [60]:
```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_test['project_grade_category']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_test_one['project_grade_category']:
    clean_one[element] +=1       #counting occurence of element in feature
for element in X_test_zero['project_grade_category']:
    clean_zero[element] +=1      #counting occurence of element in feature
#clean_one["History_Civics Warmth Care_Hunger"]
#list(clean_one.values())[1]
clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculatin
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_test["project_grade_category"]:           # mapping probabiliti
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])

X_test["project_grade_category_0"] = clean_cat_0
X_test["project_grade_category_1"] = clean_cat_1
```

In [61]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
Train_pgrade_0 = normalizer.fit_transform(X_tr["project_grade_category_0"].
Train_pgrade_1 = normalizer.fit_transform(X_tr["project_grade_category_1"].
CV_pgrade_0 = normalizer.transform(X_cv['project_grade_category_0'].values.
CV_pgrade_1 = normalizer.transform(X_cv['project_grade_category_1'].values.
Test_pgrade_0 = normalizer.transform(X_test['project_grade_category_0'].val
Test_pgrade_1 = normalizer.transform(X_test['project_grade_category_1'].val
```

2.2.5 Price

In [62]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
Train_price = normalizer.fit_transform(X_tr['price'].values.reshape(-1, 1))
#print(f"Mean : {Train_price.mean[0]}, Standard deviation : {np.sqrt(Train_
#print(np.mean(Train_price,axis=0),np.std(Train_price,axis=0))
print('Training data shape',Train_price.shape)
CV_price = normalizer.transform(X_cv['price'].values.reshape(-1, 1))
print('cv data shape',CV_price.shape)
Test_price = normalizer.transform(X_test['price'].values.reshape(-1, 1))
print('Test data shape',Test_price.shape)
```

```
Training data shape (49041, 1)
cv data shape (24155, 1)
Test data shape (36052, 1)
```

2.2.6 Quantity

In [63]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
Train_quantity = normalizer.fit_transform(X_tr['quantity'].values.reshape(-1
#print(f"Mean : {Train_price.mean[0]}, Standard deviation : {np.sqrt(Train_p
#print(np.mean(Train_quantity,axis=0),np.std(Train_quantity,axis=0))
print('Training data shape',Train_quantity.shape)
CV_quantity = normalizer.transform(X_cv['quantity'].values.reshape(-1, 1))
print('cv data shape',CV_quantity.shape)
Test_quantity = normalizer.transform(X_test['quantity'].values.reshape(-1, 1
print('Test data shape',Test_quantity.shape)
```

```
Training data shape (49041, 1)
cv data shape (24155, 1)
Test data shape (36052, 1)
```

2.2.7 School state

Train

In [64]:

```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_tr['school_state']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_train_one['school_state']:
    clean_one[element] +=1       #counting occurence of element in feature w
for element in X_train_zero['school_state']:
    clean_zero[element] +=1      #counting occurence of element in feature w

clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculating
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_tr["school_state"]:            # mapping probabilities to origina
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])

X_tr["school_state_0"] = clean_cat_0
X_tr["school_state_1"] = clean_cat_1
```

Cross validation

In [65]:

```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_cv['school_state']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_cv_one['school_state']:
    clean_one[element] +=1       #counting occurence of element in feature
for element in X_cv_zero['school_state']:
    clean_zero[element] +=1      #counting occurence of element in feature
#clean_one["History_Civics Warmth Care_Hunger"]
#list(clean_one.values())[1]
clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculatin
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_cv["school_state"]:            # mapping probabilities to origin
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])

X_cv["school_state_0"] = clean_cat_0
X_cv["school_state_1"] = clean_cat_1
```

Test

In [66]:

```
clean_one={}      #initializing empty dictionary
clean_zero={}
for element in X_test['school_state']:
    clean_one[element]=0
    clean_zero[element]=0
for element in X_test_one['school_state']:
    clean_one[element] +=1       #counting occurence of element in feature
for element in X_test_zero['school_state']:
    clean_zero[element] +=1      #counting occurence of element in feature
#clean_one["History_Civics Warmth Care_Hunger"]
#list(clean_one.values())[1]
clean_one_prob={}
clean_zero_prob={}
for i in list(clean_one.keys()):
    clean_one_prob[i]=clean_one[i]/(clean_one[i]+clean_zero[i]) #calculatin
for i in list(clean_zero.keys()):
    clean_zero_prob[i]=clean_zero[i]/(clean_one[i]+clean_zero[i])
clean_cat_1 = []
clean_cat_0 = []
for i in X_test["school_state"]:            # mapping probabilities to orig
    clean_cat_1.append(clean_one_prob[i])
    clean_cat_0.append(clean_zero_prob[i])

X_test["school_state_0"] = clean_cat_0
X_test["school_state_1"] = clean_cat_1
```

Normalization

In [67]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
Train_school_0 = normalizer.fit_transform(X_tr["school_state_0"].values.resh
Train_school_1 = normalizer.fit_transform(X_tr["school_state_1"].values.resh
CV_school_0 = normalizer.transform(X_cv['school_state_0'].values.reshape(-1
CV_school_1 = normalizer.transform(X_cv['school_state_1'].values.reshape(-1
Test_school_0 = normalizer.transform(X_test['school_state_0'].values.reshape
Test_school_1 = normalizer.transform(X_test['school_state_1'].values.reshape
```

2.2.8 teacher_number_of_previously_posted_projects

In [68]:
```
norm = Normalizer()
Train_teach = norm.fit_transform(X_tr['teacher_number_of_previously_posted_p
CV_teach=norm.transform(X_cv['teacher_number_of_previously_posted_projects']
Test_teach=norm.transform(X_test['teacher_number_of_previously_posted_proje
print('Training data shape',Train_teach.shape)
print('cv data shape',CV_teach.shape)
print('Test data shape',Test_teach.shape)
```

```
Training data shape (49041, 1)
cv data shape (24155, 1)
Test data shape (36052, 1)
```

2.2.9 number of words in the title

In [69]:
```
norm = Normalizer()
Train_t_word = norm.fit_transform(X_tr['title_word_count'].values.reshape(-1
CV_t_word=norm.transform(X_cv['title_word_count'].values.reshape(-1, 1))
Test_t_word=norm.transform(X_test['title_word_count'].values.reshape(-1, 1)
print('Training data shape',Train_t_word.shape)
print('cv data shape',CV_t_word.shape)
print('Test data shape',Test_t_word.shape)
```

```
Training data shape (49041, 1)
cv data shape (24155, 1)
Test data shape (36052, 1)
```

2.2.10 number of words in essays

In [70]:
```
norm = Normalizer()
Train_e_word = norm.fit_transform(X_tr['essay_word_count'].values.reshape(-1
CV_e_word=norm.transform(X_cv['essay_word_count'].values.reshape(-1, 1))
Test_e_word=norm.transform(X_test['essay_word_count'].values.reshape(-1, 1)
print('Training data shape',Train_e_word.shape)
print('cv data shape',CV_e_word.shape)
print('Test data shape',Test_e_word.shape)
```

```
Training data shape (49041, 1)
cv data shape (24155, 1)
Test data shape (36052, 1)
```

2.2.11 Positive sentiments essay

In [71]:
```
norm = Normalizer()
Train_e_pos_word = norm.fit_transform(X_tr['positive'].values.reshape(-1, 1
CV_e_pos_word=norm.transform(X_cv['positive'].values.reshape(-1, 1))
Test_e_pos_word=norm.transform(X_test['positive'].values.reshape(-1, 1))
print('Training data shape',Train_e_pos_word.shape)
print('cv data shape',CV_e_pos_word.shape)
print('Test data shape',Test_e_pos_word.shape)
```

```
Training data shape (49041, 1)
cv data shape (24155, 1)
Test data shape (36052, 1)
```

2.2.12 Negative sentiments essay

In [72]:
```
norm = Normalizer()
Train_e_neg_word = norm.fit_transform(X_tr['negative'].values.reshape(-1, 1
CV_e_neg_word=norm.transform(X_cv['negative'].values.reshape(-1, 1))
Test_e_neg_word=norm.transform(X_test['negative'].values.reshape(-1, 1))
print('Training data shape',Train_e_neg_word.shape)
print('cv data shape',CV_e_neg_word.shape)
print('Test data shape',Test_e_neg_word.shape)
```

```
Training data shape (49041, 1)
cv data shape (24155, 1)
Test data shape (36052, 1)
```

2.2.13 Neutral sentiments essay

In [73]:
```
norm = Normalizer()
Train_e_neu_word = norm.fit_transform(X_tr['neutral'].values.reshape(-1, 1)
CV_e_neu_word=norm.transform(X_cv['neutral'].values.reshape(-1, 1))
Test_e_neu_word=norm.transform(X_test['neutral'].values.reshape(-1, 1))
print('Training data shape',Train_e_neu_word.shape)
print('cv data shape',CV_e_neu_word.shape)
print('Test data shape',Test_e_neu_word.shape)
```

```
Training data shape (49041, 1)
cv data shape (24155, 1)
Test data shape (36052, 1)
```

2.2.14 compound sentiments essay

In [74]:
```
norm = Normalizer()
Train_e_comp_word = norm.fit_transform(X_tr['compound'].values.reshape(-1,
CV_e_comp_word=norm.transform(X_cv['compound'].values.reshape(-1, 1))
Test_e_comp_word=norm.transform(X_test['compound'].values.reshape(-1, 1))
print('Training data shape',Train_e_comp_word.shape)
print('cv data shape',CV_e_comp_word.shape)
print('Test data shape',Test_e_comp_word.shape)
```

```
Training data shape (49041, 1)
cv data shape (24155, 1)
Test data shape (36052, 1)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

2.3.1.1 Eassay bow

In [75]:
```
vectorizer = CountVectorizer(min_df=10)
Train_essays= vectorizer.fit_transform(X_tr['essay'])
CV_essays=vectorizer.transform(X_cv['essay'])
Test_essays=vectorizer.transform(X_test['essay'])
print(Train_essays.shape)
print(CV_essays.shape)
print(Test_essays.shape)
v5=vectorizer
#print(v5.get_feature_names())
```

```
(49041, 12565)
(24155, 12565)
(36052, 12565)
```

2.3.1.2 Essays tfidf

In [76]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(min_df=10)
Train_essays_tfidf = vectorizer.fit_transform(X_tr['essay'])
CV_essays_tfidf = vectorizer.transform(X_cv['essay'])
Test_essays_tfidf = vectorizer.transform(X_test['essay'])
print(Train_essays_tfidf.shape)
print(CV_essays_tfidf.shape)
print(Test_essays_tfidf.shape)
vt5=vectorizer
```

```
(49041, 12565)
(24155, 12565)
(36052, 12565)
```

In [77]:

```
i=0
list_of_sentance=[]
for sentnc in preprocessed_essays:
    list_of_sentance.append(sentnc.split())
w2v_model=Word2Vec(list_of_sentance,min_count=5,size=30, workers=4)
#print(w2v_model.wv.most_similar('teacher'))
#print('='*50)
#print(w2v_model.wv.most_similar('student'))
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  23129
sample words  ['my', 'students', 'english', 'learners', 'working', 'second
', 'third', 'languages', 'we', 'melting', 'pot', 'refugees', 'immigrants',
'native', 'born', 'americans', 'bringing', 'gift', 'language', 'school',
'24', 'represented', 'learner', 'program', 'every', 'level', 'mastery', 'a
lso', '40', 'countries', 'families', 'within', 'each', 'student', 'brings
', 'wealth', 'knowledge', 'experiences', 'us', 'open', 'eyes', 'new', 'cul
tures', 'beliefs', 'respect', 'the', 'limits', 'world', 'ludwig', 'our']
```

2.3.1.3 Essays wordtovec

In [78]:

```
def avg_w2v_essays(ESSAYS):
    avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored
    for sentence in tqdm(ESSAYS): # for each review/sentence
        vector = np.zeros(30) # as word vectors are of zero length
        cnt_words =0; # num of words with a valid vector in the sentence/re
        for word in sentence.split(): # for each word in a review/sentence
            if word in w2v_model:
                vector += w2v_model.wv[word]
                cnt_words += 1
                cnt_words += 1
        if cnt_words != 0:
            vector /= cnt_words
        avg_w2v_vectors.append(vector)
    return(avg_w2v_vectors)
Train_essays_w2v=avg_w2v_essays(X_tr['essay'])
CV_essays_w2v=avg_w2v_essays(X_cv['essay'])
Test_essays_w2v=avg_w2v_essays(X_test['essay'])
print(len(Train_essays_w2v))
print(len(CV_essays_w2v))
print(len(Test_essays_w2v))
```

```
100%|██████████| 49041/49041 [03:16<00:00, 249.48it/s]
100%|██████████| 24155/24155 [01:35<00:00, 253.62it/s]
100%|██████████| 36052/36052 [02:23<00:00, 251.91it/s]

49041
24155
36052
```

In [79]:

```
Train_essays_w2v=np.array(Train_essays_w2v)
CV_essays_w2v=np.array(CV_essays_w2v)
Test_essays_w2v=np.array(Test_essays_w2v)
```

2.3.1.4 Essays TFIDF W2V

In [80]:
```
def essay_tfidf_w2v(ESSAYS,tfidf_model,dictionary,tfidf_words):

    tfidf_model.transform(ESSAYS)

    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored
    for sentence in tqdm(ESSAYS): # for each review/sentence
        vector = np.zeros(30) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence
        for word in sentence.split(): # for each word in a review/sentence
            if (word in w2v_model) and (word in tfidf_words):
                vec = w2v_model.wv[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and t
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)
    return(tfidf_w2v_vectors)
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_tr['essay'])
# 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())
Train_essays_tfidf_w2v=essay_tfidf_w2v(X_tr['essay'],tfidf_model,dictionary
CV_essays_tfidf_w2v=essay_tfidf_w2v(X_cv['essay'],tfidf_model,dictionary,tf
Test_essays_tfidf_w2v=essay_tfidf_w2v(X_test['essay'],tfidf_model,dictionary
print(len(Train_essays_tfidf_w2v))
print(len(CV_essays_tfidf_w2v))
print(len(Test_essays_tfidf_w2v))
```

```
100%|██████████| 49041/49041 [08:59<00:00, 90.86it/s]
100%|██████████| 24155/24155 [04:25<00:00, 91.11it/s]
100%|██████████| 36052/36052 [06:34<00:00, 91.41it/s]

49041
24155
36052
```

In [81]:
```
Train_essays_tfidf_w2v=np.array(Train_essays_tfidf_w2v)
CV_essays_tfidf_w2v=np.array(CV_essays_tfidf_w2v)
Test_essays_tfidf_w2v=np.array(Test_essays_tfidf_w2v)
```

2.3.2.1 Project title bow

In [82]:
```
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_tr['project_title'])
Train_ptitle=vectorizer.transform(X_tr['project_title'])
CV_ptitle = vectorizer.transform(X_cv['project_title'])
Test_ptitle = vectorizer.transform(X_test['project_title'])
#print(len(X_cv['project_title']))
print(Train_ptitle.shape)
print(CV_ptitle.shape)
print(Test_ptitle.shape)
v6=vectorizer
```

```
(49041, 2128)
(24155, 2128)
(36052, 2128)
```

2.3.2.2 Project title tfidf

In [83]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

Tr_ptitle_tfidf=X_tr['project_title']
vectorizer = TfidfVectorizer(min_df=10)
Train_ptitle_tfidf = vectorizer.fit_transform(Tr_ptitle_tfidf)
C_ptitle_tfidf=X_cv['project_title']
CV_ptitle_tfidf = vectorizer.transform(C_ptitle_tfidf)
Te_ptitle_tfidf=X_test['project_title']
Test_ptitle_tfidf = vectorizer.transform(Te_ptitle_tfidf)
print(Train_ptitle_tfidf.shape)
print(CV_ptitle_tfidf.shape)
print(Test_ptitle_tfidf.shape)
vt6=vectorizer
```

```
(49041, 2128)
(24155, 2128)
(36052, 2128)
```

2.3.2.3 Project title w2v

In [84]:
```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentnc in preprocessed_titles:
    list_of_sentance.append(sentnc.split())
w2v_model=Word2Vec(list_of_sentance,min_count=5,size=30, workers=4)
#print(w2v_model.wv.most_similar('teacher'))
#print('='*50)
#print(w2v_model.wv.most_similar('student'))
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  5177
sample words  ['educational', 'support', 'english', 'learners', 'home', 'w
anted', 'projector', 'hungry', 'soccer', 'equipment', 'awesome', 'middle',
'school', 'students', 'techie', 'kindergarteners', 'interactive', 'math',
'tools', 'flexible', 'seating', 'mrs', 'terrific', 'third', 'graders', 'ch
romebooks', 'special', 'education', 'reading', 'program', 'it', '21st', 'c
entury', 'targeting', 'more', 'success', 'class', 'just', 'for', 'love', '
pure', 'pleasure', 'changes', 'lives', 'elevating', 'academics', 'parent',
'through', 'technology', 'building']
```

In [85]:

```
def avg_w2v_ptitle(ptitle):
    i=0
    list_of_sentance=[]
    for sentnc in ptitle:
        list_of_sentance.append(sentnc.split())
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=30, workers=4)
    w2v_words = list(w2v_model.wv.vocab)
    sent_vectors = []; # the avg-w2v for each sentence
    for sent in (list_of_sentance): # for each sentence
        sent_vec = np.zeros(30) # as word vectors are of zero length 50, yo
        cnt_words =0; # num of words with a valid vector in the sentence
        for word in sent: # for each word in a sentence
            if word in w2v_words:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    return(sent_vectors)
Train_ptitle_w2v=avg_w2v_ptitle(X_tr['project_title'])
CV_ptitle_w2v=avg_w2v_ptitle(X_cv['project_title'])
Test_ptitle_w2v=avg_w2v_ptitle(X_test['project_title'])
print(len(Train_ptitle_w2v))
print(len(CV_ptitle_w2v))
print(len(Test_ptitle_w2v))
```

```
49041
24155
36052
```

In [86]:

```
Train_ptitle_w2v=np.array(Train_ptitle_w2v)
CV_ptitle_w2v=np.array(CV_ptitle_w2v)
Test_ptitle_w2v=np.array(Test_ptitle_w2v)
```

2.3.2.4 Project title tfidf w2v

In [87]:

```
def ptitle_tfidf_w2v(ptitle,tfidf_model,dictionary,tfidf_words):
    #tfidf_model = TfidfVectorizer()
    tfidf_model.transform(ptitle)
    # we are converting a dictionary with word as a key, and the idf as a va
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is store
    for sentence in (ptitle): # for each review/sentence
        vector = np.zeros(30) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence
        for word in sentence.split(): # for each word in a review/sentence
            if (word in w2v_words) and (word in tfidf_words):
                vec = w2v_model.wv[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and t
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)
    return(tfidf_w2v_vectors)

Tmodel = TfidfVectorizer()
Tmodel.fit(X_tr['project_title'])
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_
tfidf_words = set(tfidf_model.get_feature_names())
Train_ptitle_tfidf_w2v=ptitle_tfidf_w2v(X_tr['project_title'],Tmodel,diction
CV_ptitle_tfidf_w2v=ptitle_tfidf_w2v(X_cv['project_title'],Tmodel,dictionary
Test_ptitle_tfidf_w2v=ptitle_tfidf_w2v(X_test['project_title'],Tmodel,dictio
print(len(Train_ptitle_tfidf_w2v))
print(len(CV_ptitle_tfidf_w2v))
print(len(Test_ptitle_tfidf_w2v))
```

```
49041
24155
36052
```

In [88]:

```
Train_ptitle_tfidf_w2v=np.array(Train_ptitle_tfidf_w2v)
CV_ptitle_tfidf_w2v=np.array(CV_ptitle_tfidf_w2v)
Test_ptitle_tfidf_w2v=np.array(Test_ptitle_tfidf_w2v)
```

2.4 Applying Random Forest

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

2.4.1 Applying Random Forests on BOW, SET 1

In [89]:

```
import matplotlib.pyplot as plt
from scipy.sparse import hstack
#from sklearn import datasets, neighbors
from matplotlib.colors import ListedColormap
#https://pythonspot.com/k-nearest-neighbors/
#from mlxtend.plotting import plot_decision_regions
```

In [90]:
```
# Please write all the code with proper documentation


Xh = hstack((Train_categ_0,Train_categ_1,Train_sub_categ_0,Train_sub_categ_1
              Train_pgrade_0,Train_pgrade_1,Train_price,Train_essays,Train_pt
Xh_test=hstack((Test_categ_0,Test_categ_1,Test_sub_categ_0,Test_sub_categ_1
                 Test_pgrade_0,Test_pgrade_1,Test_price,Test_essays,Test_ptit
Xh_cross=hstack((CV_categ_0,CV_categ_1,CV_sub_categ_0,CV_sub_categ_1,CV_teac
              CV_pgrade_0,CV_pgrade_1,CV_price,CV_essays,CV_ptitle)).tocsr()

print(Xh.shape)
print(Xh_test.shape)
print(Xh_cross.shape)
```

```
(49041, 14702)
(36052, 14702)
(24155, 14702)
```

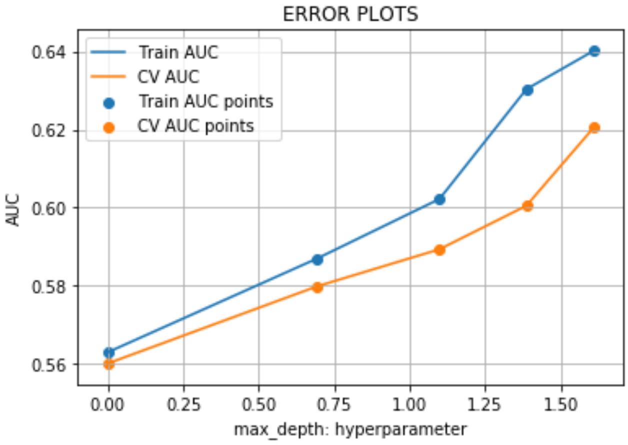
In [91]:
```
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.grid_search import GridSearchCV
```

```
/home/ankit/anaconda3/lib/python3.6/site-packages/sklearn/grid_search.p
y:42: DeprecationWarning:

This module was deprecated in version 0.18 in favor of the model_selection
module into which all the refactored classes and functions are moved. This
module will be removed in 0.20.
```

In [92]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 4
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [414]:

```
import math
train_auc = []
cv_auc = []
K = {'max_depth':  [1,2,3,4,5]}
for i in K['max_depth']:
    neigh = RandomForestClassifier(max_depth=i,class_weight='balanced')
    neigh.fit(Xh, y_tr)

    y_train_pred = batch_predict(neigh, Xh)
    y_cv_pred = batch_predict(neigh, Xh_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['max_depth']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

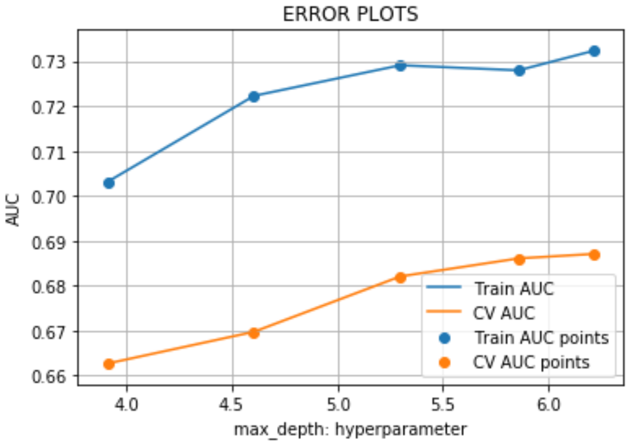


Take max_depth =3

In [479]:

```
max_depth1=3
```

In [411]:
```
import math
train_auc = []
cv_auc = []
K = {'n_estimators':  [50,100,200,350,500]}
for i in K['n_estimators']:
    neigh = RandomForestClassifier(n_estimators=i,max_depth=max_depth1,clas
    neigh.fit(Xh, y_tr)

    y_train_pred = batch_predict(neigh, Xh)
    y_cv_pred = batch_predict(neigh, Xh_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['n_estimators']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [480]:
```
n_estimators1=350
```

In [478]:
```
"""
a=[1,2]
b=[2,3]
l=[]
l.append(a)
l.append(b)
print(l)
sns.heatmap(l,annot=True)
plt.show()
"""
```

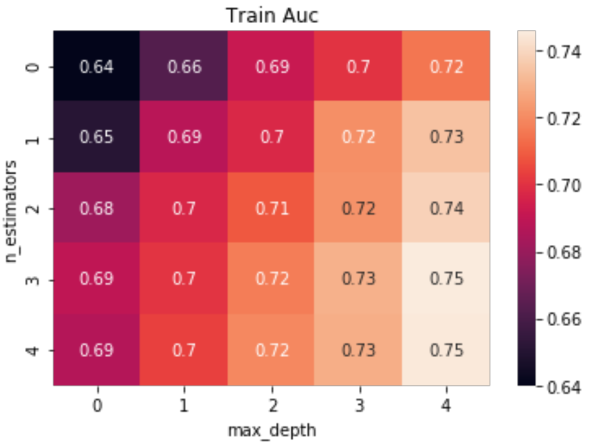
Out[478]:
```
'\na=[1,2]\nb=[2,3]\nl=[]\nl.append(a)\nl.append(b)\nprint(l)\nsns.heatmap
(l,annot=True)\nplt.show()\n'
```

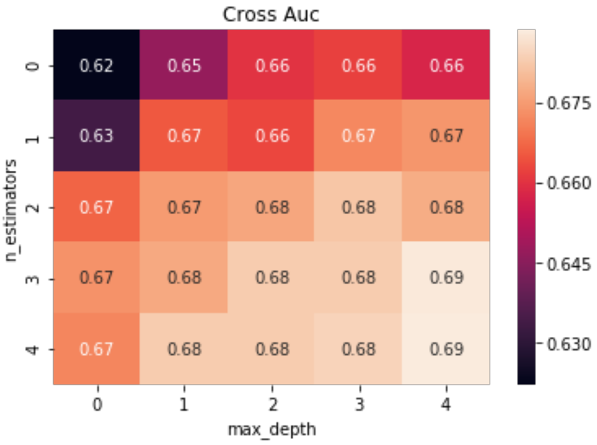
In [450]:
```
"""
X=K['max_depth']
data=pd.DataFrame({'max_depth':K['max_depth'],'n_estimators':X,'AUC':cv_auc}
data_p=data.pivot('max_depth','n_estimators','AUC')
sns.heatmap(data_p,annot=True,)
plt.show()
"""
```

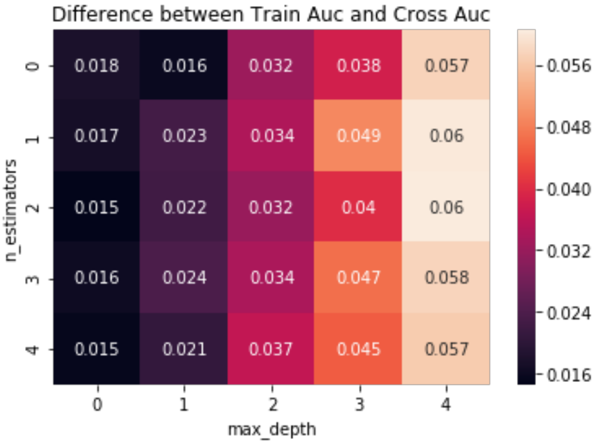
Out[450]:
```
"\nX=K['max_depth']\ndata=pd.DataFrame({'max_depth':K['max_depth'],'n_esti
mators':X,'AUC':cv_auc})\ndata_p=data.pivot('max_depth','n_estimators','AU
C')\nsns.heatmap(data_p,annot=True,)\nplt.show()\n"
```

In [476]:
```
from sklearn.metrics import roc_curve, auc
K = {'n_estimators':  [50,100,200,350,500],'max_depth':  [1,2,3,4,5]}
diff=[]
t_auc=[]
c_auc=[]
for e in K['n_estimators']:
    l_dif=[]
    l_train=[]
    l_cross=[]
    for d in K['max_depth']:
        neigh = RandomForestClassifier(max_depth=d,n_estimators=e,class_weig
        neigh.fit(Xh, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

        y_train_pred = batch_predict(neigh, Xh)
        y_cv_pred = batch_predict(neigh, Xh_cross)

        train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
        cv_fpr, cv_tpr, te_thresholds = roc_curve(y_cv, y_cv_pred)
        l_train.append(auc(train_fpr, train_tpr))
        l_cross.append(auc(cv_fpr, cv_tpr))
        no=auc(train_fpr, train_tpr)-auc(cv_fpr, cv_tpr)
        l_dif.append(no)

    diff.append(l_dif)
    t_auc.append(l_train)
    c_auc.append(l_cross)
```

In [481]:
```
sns.heatmap(t_auc,annot=True)
plt.title(" Train Auc ")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(c_auc,annot=True)
plt.title("Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(diff,annot=True)
plt.title("Difference between Train Auc and Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()
```

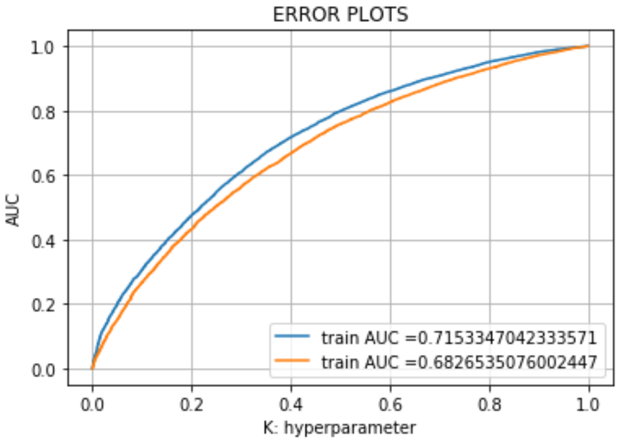






In [482]:

```
from sklearn.metrics import roc_curve, auc

neigh = RandomForestClassifier(max_depth=max_depth1,n_estimators=n_estimato
neigh.fit(Xh, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

y_train_pred = batch_predict(neigh, Xh)
y_test_pred = batch_predict(neigh, Xh_test)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, 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_
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



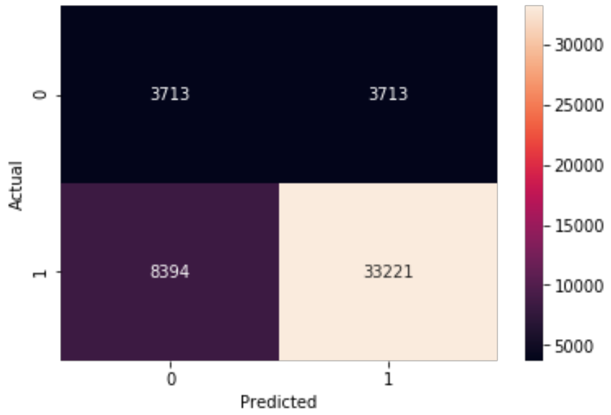
In [93]:

```
def predict(proba, 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

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for thresho
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [484]:
```
import seaborn as sns
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_tr, predict(y_train_pred, tr_thresholds, tra
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.496
```

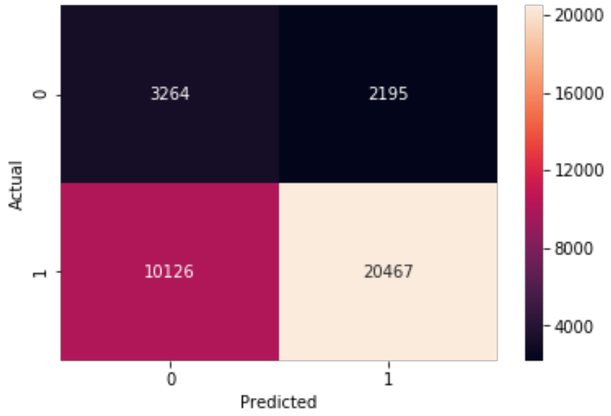
Out[484]:
```
Text(33,0.5,'Actual')
```



In [485]:
```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.499
```

Out[485]:
```
Text(33,0.5,'Actual')
```



2.4.1 Applying Random Forests on TFIDF, SET 2

In [94]:
```
# Please write all the code with proper documentation
from sklearn.metrics import accuracy_score
from scipy.sparse import hstack

Xh2 = hstack((Train_categ_0,Train_categ_1,Train_sub_categ_0,Train_sub_categ_
             Train_pgrade_0,Train_pgrade_1,Train_price,Train_essays_tfidf,Tr
Xh2_test=hstack((Test_categ_0,Test_categ_1,Test_sub_categ_0,Test_sub_categ_1
                Test_pgrade_0,Test_pgrade_1,Test_price,Test_essays_tfidf,Tes
Xh2_cross=hstack((CV_categ_0,CV_categ_1,CV_sub_categ_0,CV_sub_categ_1,CV_tea
             CV_pgrade_0,CV_pgrade_1,CV_price,CV_essays_tfidf,CV_ptitle_tfi
```

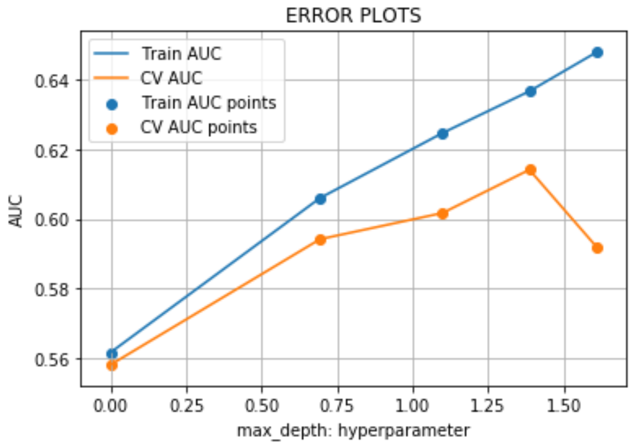
In [496]:

```
print(Xh2.shape)
print(Xh2_test.shape)
print(Xh2_cross.shape)
```

```
(49041, 14652)
(36052, 14652)
(24155, 14652)
```

In [497]:

```
import math
train_auc = []
cv_auc = []
K = {'max_depth':  [1,2,3,4,5]}
for i in K['max_depth']:
    neigh = RandomForestClassifier(max_depth=i,class_weight='balanced')
    neigh.fit(Xh2, y_tr)

    y_train_pred = batch_predict(neigh, Xh2)
    y_cv_pred = batch_predict(neigh, Xh2_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['max_depth']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

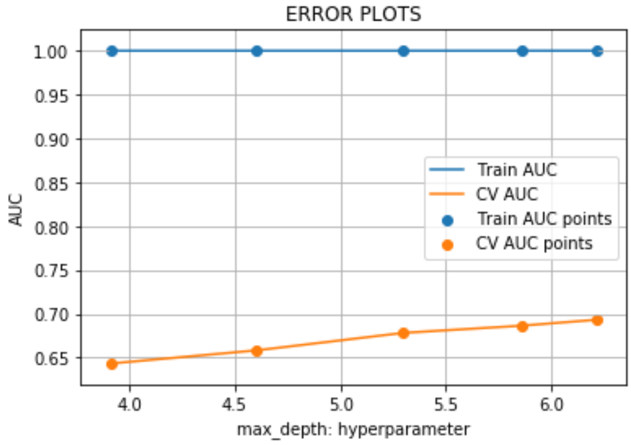


In [563]:

```
max_depth2=4
```

In [498]:

```
import math
train_auc = []
cv_auc = []
K = {'n_estimators':  [50,100,200,350,500]}
for i in K['n_estimators']:
    neigh = RandomForestClassifier(n_estimators=i,class_weight='balanced')
    neigh.fit(Xh2, y_tr)

    y_train_pred = batch_predict(neigh, Xh2)
    y_cv_pred = batch_predict(neigh, Xh2_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['n_estimators']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



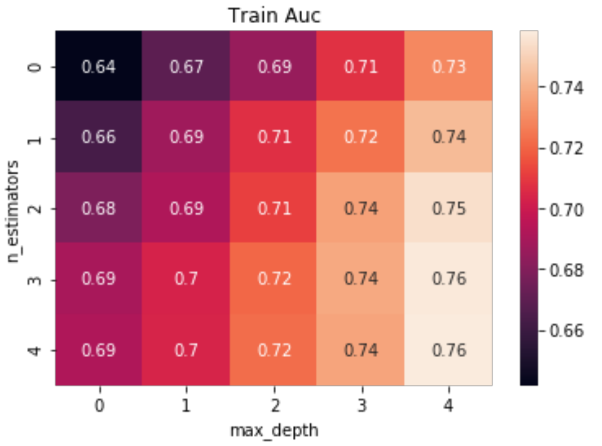
In [564]:

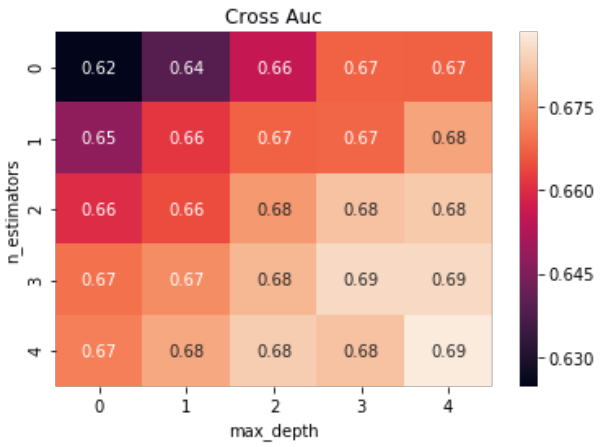
```
n_estimators2=500
```

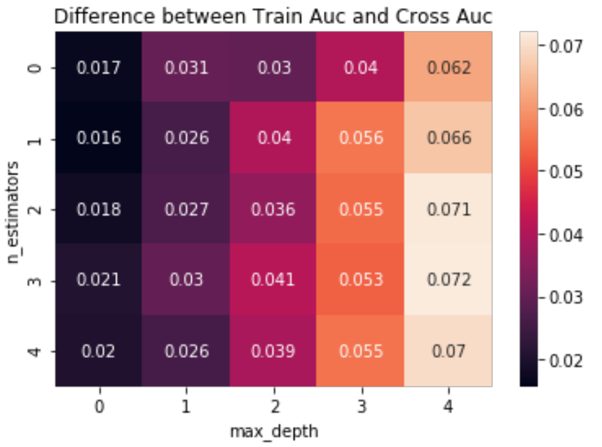
In [499]:

```
from sklearn.metrics import roc_curve, auc
K = {'n_estimators':  [50,100,200,350,500],'max_depth':  [1,2,3,4,5]}
diff=[]
t_auc=[]
c_auc=[]
for e in K['n_estimators']:
    l_dif=[]
    l_train=[]
    l_cross=[]
    for d in K['max_depth']:
        neigh = RandomForestClassifier(max_depth=d,n_estimators=e,class_weig
        neigh.fit(Xh2, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

        y_train_pred = batch_predict(neigh, Xh2)
        y_cv_pred = batch_predict(neigh, Xh2_cross)

        train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
        cv_fpr, cv_tpr, te_thresholds = roc_curve(y_cv, y_cv_pred)
        l_train.append(auc(train_fpr, train_tpr))
        l_cross.append(auc(cv_fpr, cv_tpr))
        no=auc(train_fpr, train_tpr)-auc(cv_fpr, cv_tpr)
        l_dif.append(no)

    diff.append(l_dif)
    t_auc.append(l_train)
    c_auc.append(l_cross)
```

In [500]:
```
sns.heatmap(t_auc,annot=True)
plt.title(" Train Auc ")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(c_auc,annot=True)
plt.title("Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(diff,annot=True)
plt.title("Difference between Train Auc and Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()
```

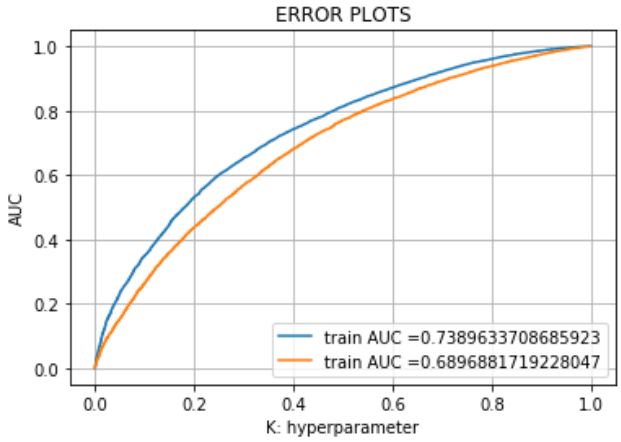






In [565]:

```
from sklearn.metrics import roc_curve, auc

neigh = RandomForestClassifier(max_depth=max_depth2,n_estimators=n_estimato
neigh.fit(Xh2, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

y_train_pred = batch_predict(neigh, Xh2)
y_test_pred = batch_predict(neigh, Xh2_test)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, 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_
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



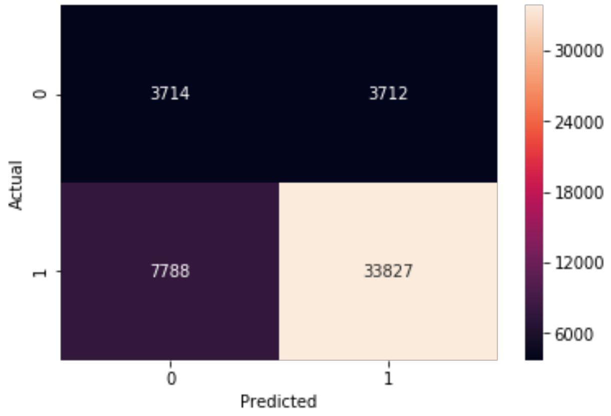
In [566]:

```
import seaborn as sns
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_tr, predict(y_train_pred, tr_thresholds, tra
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499999818661462 for threshold 0.496
```

Out[566]:

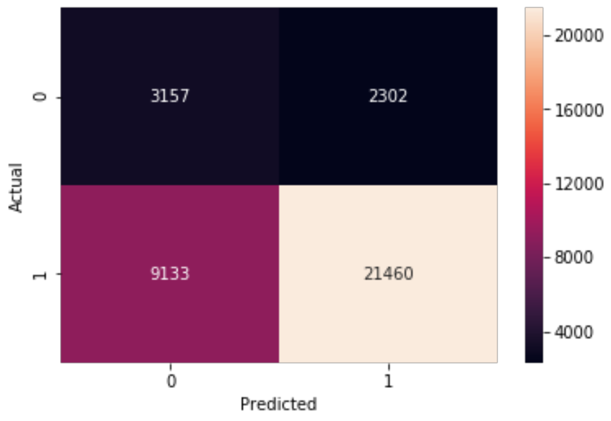
```
Text(33,0.5,'Actual')
```



In [567]:
```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.5
```

Out[567]:
```
Text(33,0.5,'Actual')
```

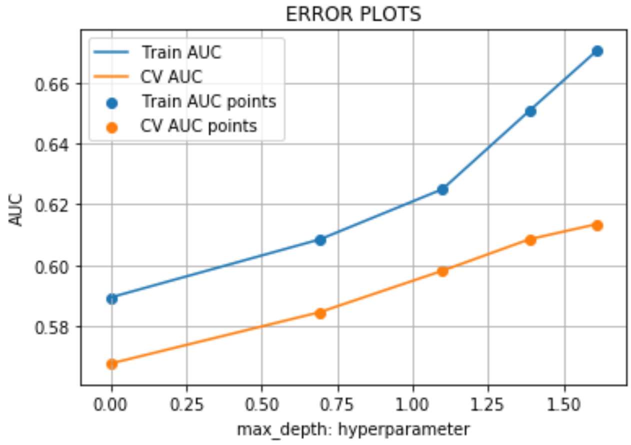


2.4.3 Applying Random Forests on AVG W2V, SET 3

In [95]:
```
Xh3 = np.hstack((Train_categ_0,Train_categ_1,Train_sub_categ_0,Train_sub_ca
             Train_pgrade_0,Train_pgrade_1,Train_price,Train_essays_w2v,Tra
Xh3_test=np.hstack((Test_categ_0,Test_categ_1,Test_sub_categ_0,Test_sub_cate
                Test_pgrade_0,Test_pgrade_1,Test_price,Test_essays_w2v,Test_
Xh3_cross=np.hstack((CV_categ_0,CV_categ_1,CV_sub_categ_0,CV_sub_categ_1,CV_
                  CV_pgrade_1,CV_price,CV_essays_w2v,CV_ptitle_w2v))
```

In [553]:

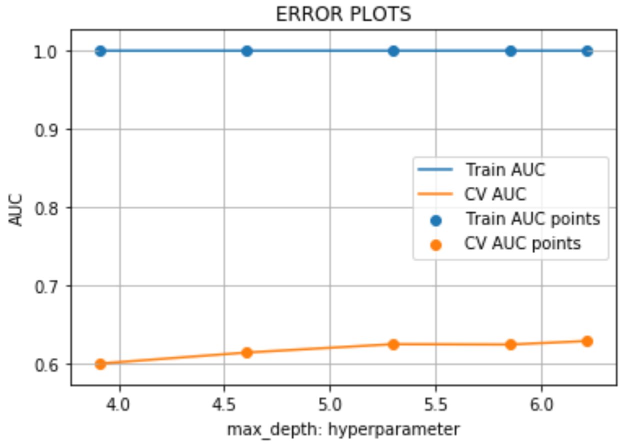
```
import math
train_auc = []
cv_auc = []
K = {'max_depth':  [1,2,3,4,5]}
for i in K['max_depth']:
    neigh = RandomForestClassifier(max_depth=i,class_weight='balanced')
    neigh.fit(Xh3, y_tr)

    y_train_pred = batch_predict(neigh, Xh3)
    y_cv_pred = batch_predict(neigh, Xh3_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['max_depth']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [568]:

```
max_depth3=3
```

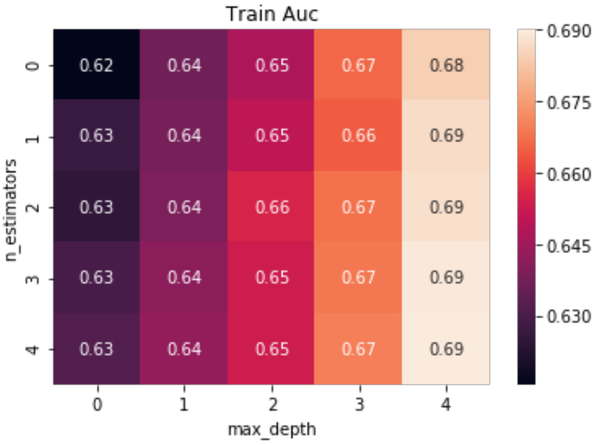
In [554]:
```
import math
train_auc = []
cv_auc = []
K = {'n_estimators':  [50,100,200,350,500]}
for i in K['n_estimators']:
    neigh = RandomForestClassifier(n_estimators=i,class_weight='balanced')
    neigh.fit(Xh3, y_tr)

    y_train_pred = batch_predict(neigh, Xh3)
    y_cv_pred = batch_predict(neigh, Xh3_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['n_estimators']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

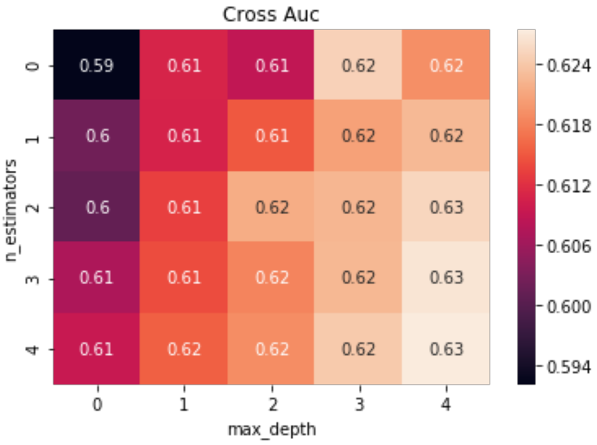


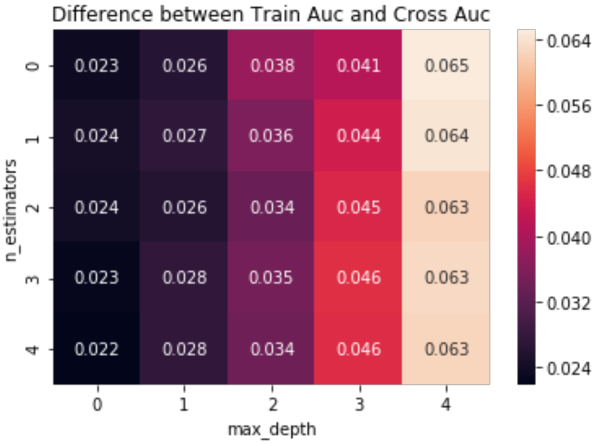
In [555]:

```
from sklearn.metrics import roc_curve, auc
K = {'n_estimators':  [50,100,200,350,500],'max_depth':  [1,2,3,4,5]}
diff=[]
t_auc=[]
c_auc=[]
for e in K['n_estimators']:
    l_dif=[]
    l_train=[]
    l_cross=[]
    for d in K['max_depth']:
        neigh = RandomForestClassifier(max_depth=d,n_estimators=e,class_weig
        neigh.fit(Xh3, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

        y_train_pred = batch_predict(neigh, Xh3)
        y_cv_pred = batch_predict(neigh, Xh3_cross)

        train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
        cv_fpr, cv_tpr, te_thresholds = roc_curve(y_cv, y_cv_pred)
        l_train.append(auc(train_fpr, train_tpr))
        l_cross.append(auc(cv_fpr, cv_tpr))
        no=auc(train_fpr, train_tpr)-auc(cv_fpr, cv_tpr)
        l_dif.append(no)

    diff.append(l_dif)
    t_auc.append(l_train)
    c_auc.append(l_cross)
```

In [556]:
```
sns.heatmap(t_auc,annot=True)
plt.title(" Train Auc ")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(c_auc,annot=True)
plt.title("Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(diff,annot=True)
plt.title("Difference between Train Auc and Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()
```

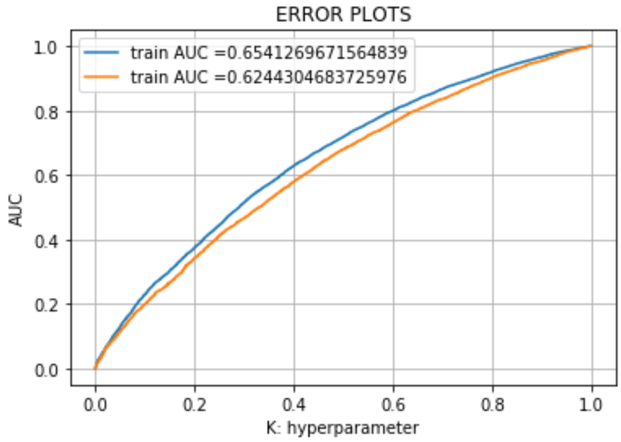






In [569]:
```
n_estimators3=500
```

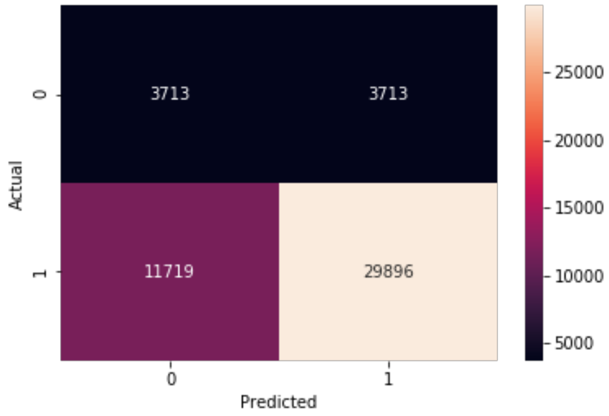
In [570]:
```
from sklearn.metrics import roc_curve, auc

neigh = RandomForestClassifier(max_depth=max_depth3,n_estimators=n_estimato
neigh.fit(Xh3, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

y_train_pred = batch_predict(neigh, Xh3)
y_test_pred = batch_predict(neigh, Xh3_test)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, 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_
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [571]:
```
import seaborn as sns
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_tr, predict(y_train_pred, tr_thresholds, tra
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.491
```

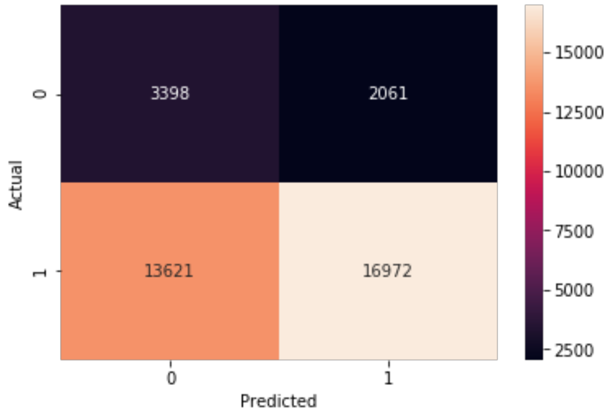
Out[571]:
```
Text(33,0.5,'Actual')
```



```
In [572]: print("Test confusion matrix")
          sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
          plt.xlabel("Predicted")
          plt.ylabel("Actual")
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.504
```

```
Out[572]: Text(33,0.5,'Actual')
```

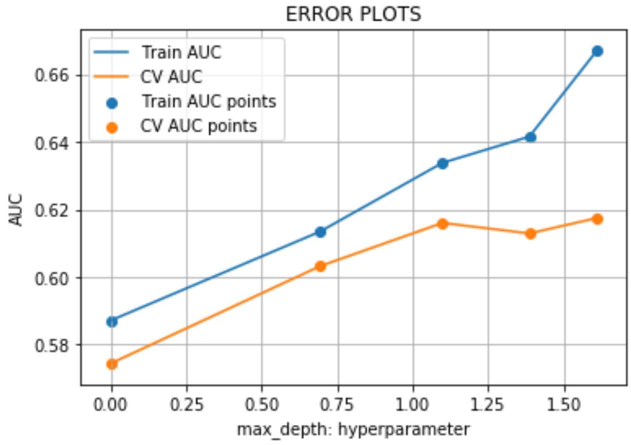


2.4.4 Applying Random Forests on TFIDF W2V, SET 4

```
In [96]: Xh4 = np.hstack((Train_categ_0,Train_categ_1,Train_sub_categ_0,Train_sub_ca
                      Train_pgrade_0,Train_pgrade_1,Train_price,Train_essays_tfidf_w2
         Xh4_test=np.hstack((Test_categ_0,Test_categ_1,Test_sub_categ_0,Test_sub_cate
                         Test_pgrade_0,Test_pgrade_1,Test_price,Test_essays_tfidf_w2v
         Xh4_cross=np.hstack((CV_categ_0,CV_categ_1,CV_sub_categ_0,CV_sub_categ_1,CV_
                      CV_pgrade_0,CV_pgrade_1,CV_price,CV_essays_tfidf_w2v,CV_ptitle
```

In [559]:

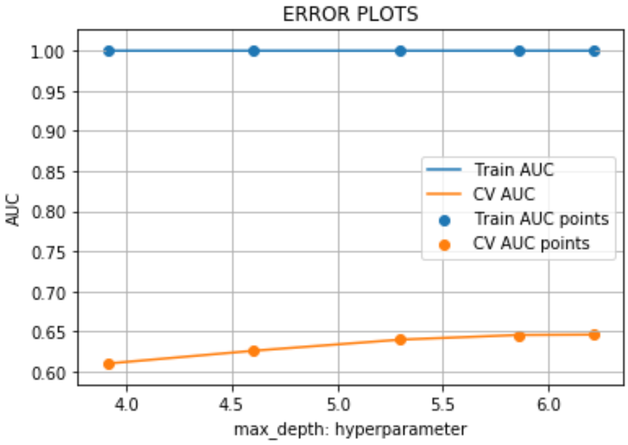
```
import math
train_auc = []
cv_auc = []
K = {'max_depth':  [1,2,3,4,5]}
for i in K['max_depth']:
    neigh = RandomForestClassifier(max_depth=i,class_weight='balanced')
    neigh.fit(Xh4, y_tr)

    y_train_pred = batch_predict(neigh, Xh4)
    y_cv_pred = batch_predict(neigh, Xh4_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['max_depth']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [573]:

```
max_depth4=3
```

In [560]:
```
import math
train_auc = []
cv_auc = []
K = {'n_estimators':  [50,100,200,350,500]}
for i in K['n_estimators']:
    neigh = RandomForestClassifier(n_estimators=i,class_weight='balanced')
    neigh.fit(Xh4, y_tr)

    y_train_pred = batch_predict(neigh, Xh4)
    y_cv_pred = batch_predict(neigh, Xh4_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['n_estimators']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



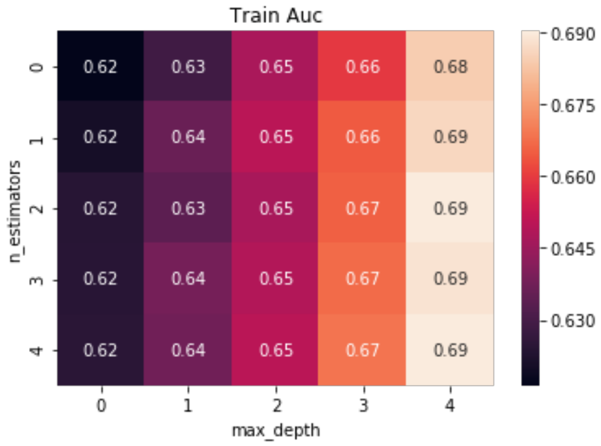
In [577]:
```
n_estimators4=500
```

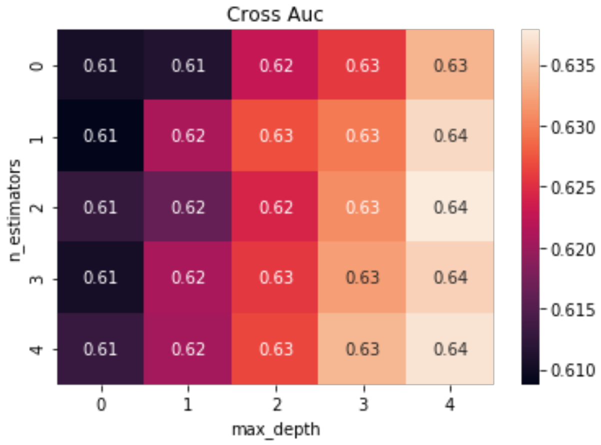
In [561]:

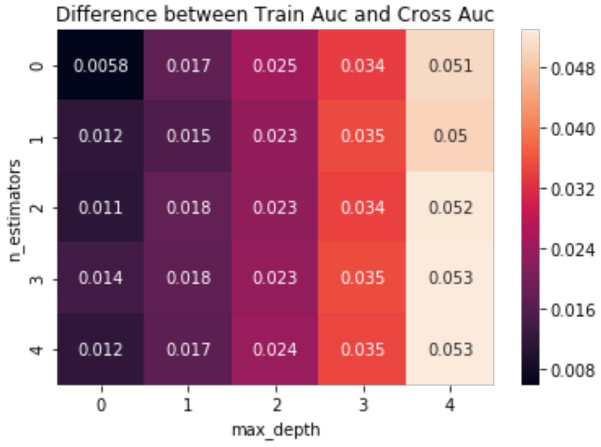
```
from sklearn.metrics import roc_curve, auc
K = {'n_estimators':  [50,100,200,350,500],'max_depth':  [1,2,3,4,5]}
diff=[]
t_auc=[]
c_auc=[]
for e in K['n_estimators']:
    l_dif=[]
    l_train=[]
    l_cross=[]
    for d in K['max_depth']:
        neigh = RandomForestClassifier(max_depth=d,n_estimators=e,class_weig
        neigh.fit(Xh4, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

        y_train_pred = batch_predict(neigh, Xh4)
        y_cv_pred = batch_predict(neigh, Xh4_cross)

        train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
        cv_fpr, cv_tpr, te_thresholds = roc_curve(y_cv, y_cv_pred)
        l_train.append(auc(train_fpr, train_tpr))
        l_cross.append(auc(cv_fpr, cv_tpr))
        no=auc(train_fpr, train_tpr)-auc(cv_fpr, cv_tpr)
        l_dif.append(no)

    diff.append(l_dif)
    t_auc.append(l_train)
    c_auc.append(l_cross)
```

In [562]:

```
sns.heatmap(t_auc,annot=True)
plt.title(" Train Auc ")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(c_auc,annot=True)
plt.title("Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(diff,annot=True)
plt.title("Difference between Train Auc and Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()
```

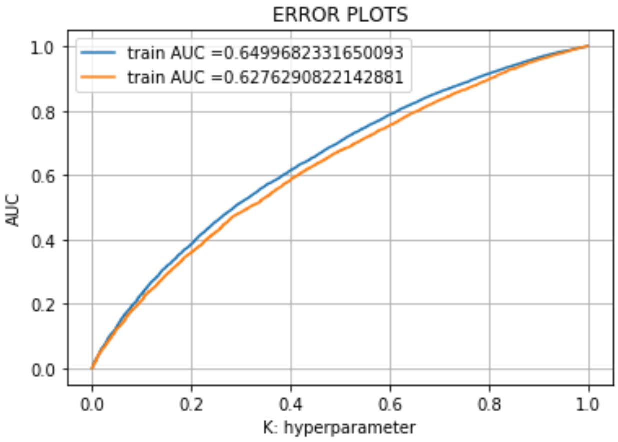






In [578]:

```
from sklearn.metrics import roc_curve, auc

neigh = RandomForestClassifier(max_depth=max_depth4,n_estimators=n_estimato
neigh.fit(Xh4, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

y_train_pred = batch_predict(neigh, Xh4)
y_test_pred = batch_predict(neigh, Xh4_test)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, 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_
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



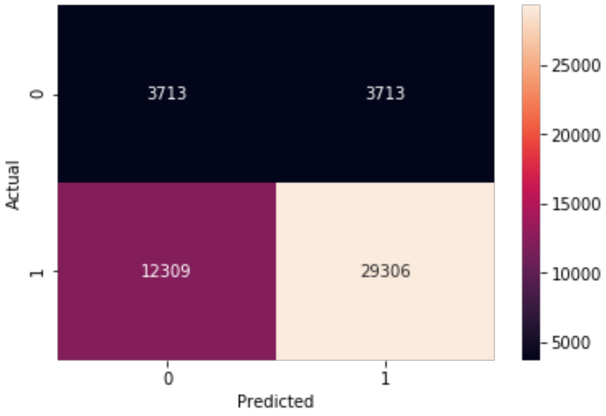
In [579]:

```
import seaborn as sns
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_tr, predict(y_train_pred, tr_thresholds, tra
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.489
```

Out[579]:

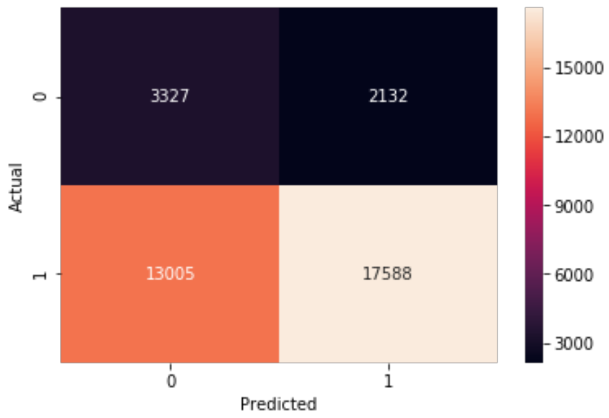
```
Text(33,0.5,'Actual')
```



In [580]:
```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.501
```

Out[580]:
```
Text(33,0.5,'Actual')
```



2.5 Applying GBDT

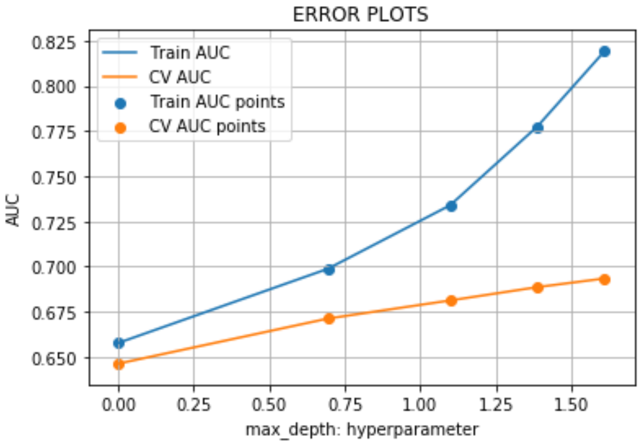
Apply GBDT on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

2.5.1 Applying XGBOOST on BOW, SET 1

In [98]:
```
from xgboost import XGBClassifier
```

In [582]:

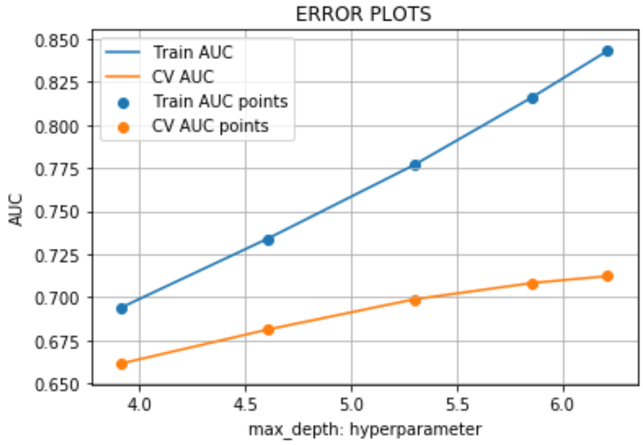
```
import math
train_auc = []
cv_auc = []
K = {'max_depth':  [1,2,3,4,5]}
for i in K['max_depth']:
    neigh = XGBClassifier(max_depth=i)
    neigh.fit(Xh, y_tr)

    y_train_pred = batch_predict(neigh, Xh)
    y_cv_pred = batch_predict(neigh, Xh_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['max_depth']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [128]:

```
max_depth1=2
```

In [584]:
```
import math
train_auc = []
cv_auc = []
K = {'n_estimators':  [50,100,200,350,500]}
for i in K['n_estimators']:
    neigh = XGBClassifier(n_estimators=i)
    neigh.fit(Xh, y_tr)

    y_train_pred = batch_predict(neigh, Xh)
    y_cv_pred = batch_predict(neigh, Xh_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['n_estimators']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

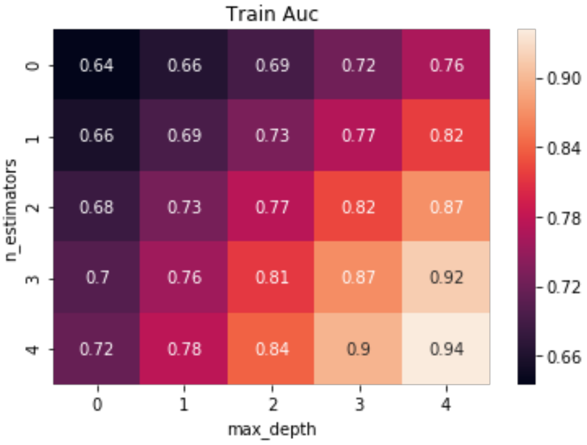


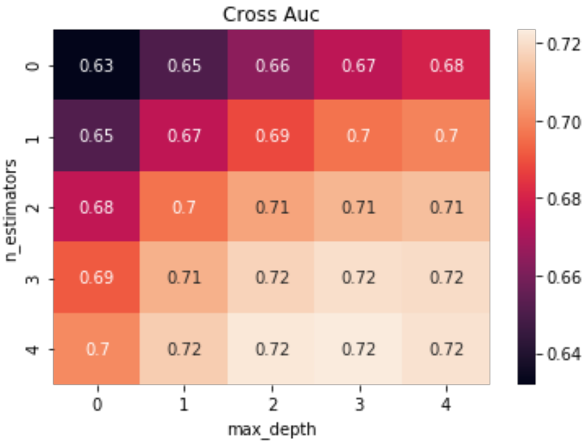
In [129]:
```
n_estimators1=50
```

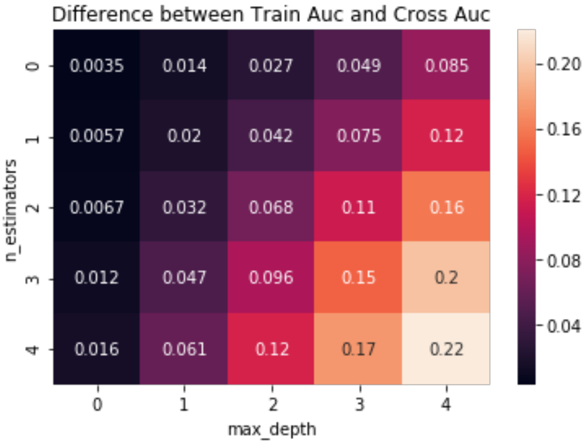
In [99]:

```
from sklearn.metrics import roc_curve, auc
K = {'n_estimators':  [50,100,200,350,500],'max_depth':  [1,2,3,4,5]}
diff=[]
t_auc=[]
c_auc=[]
for e in K['n_estimators']:
    l_dif=[]
    l_train=[]
    l_cross=[]
    for d in K['max_depth']:
        neigh = XGBClassifier(max_depth=d,n_estimators=e)
        neigh.fit(Xh, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

        y_train_pred = batch_predict(neigh, Xh)
        y_cv_pred = batch_predict(neigh, Xh_cross)

        train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
        cv_fpr, cv_tpr, te_thresholds = roc_curve(y_cv, y_cv_pred)
        l_train.append(auc(train_fpr, train_tpr))
        l_cross.append(auc(cv_fpr, cv_tpr))
        no=auc(train_fpr, train_tpr)-auc(cv_fpr, cv_tpr)
        l_dif.append(no)

    diff.append(l_dif)
    t_auc.append(l_train)
    c_auc.append(l_cross)
```

In [100]:
```
sns.heatmap(t_auc,annot=True)
plt.title(" Train Auc ")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(c_auc,annot=True)
plt.title("Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(diff,annot=True)
plt.title("Difference between Train Auc and Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()
```

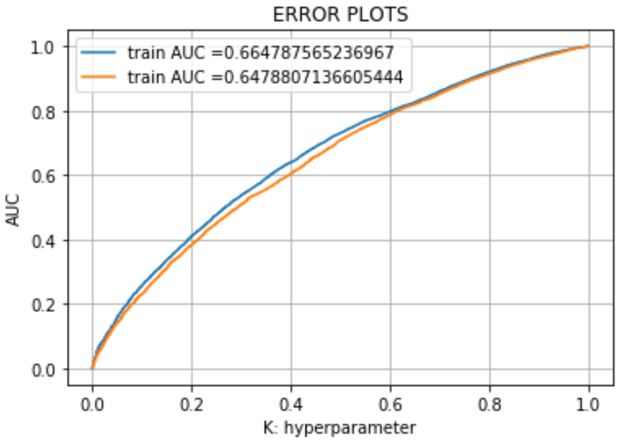

Train Auc


Cross Auc


Difference between Train Auc and Cross Auc

In [130]:

```
from sklearn.metrics import roc_curve, auc

neigh = XGBClassifier(max_depth=max_depth1,n_estimators=n_estimators1)
neigh.fit(Xh, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

y_train_pred = batch_predict(neigh, Xh)
y_test_pred = batch_predict(neigh, Xh_test)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, 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_
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



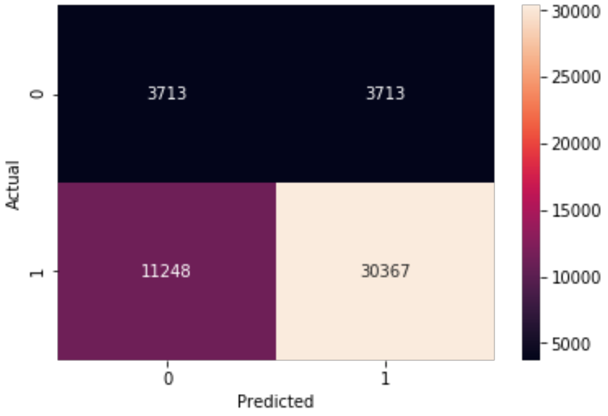
In [131]:

```
import seaborn as sns
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_tr, predict(y_train_pred, tr_thresholds, tra
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.829
```

Out[131]:

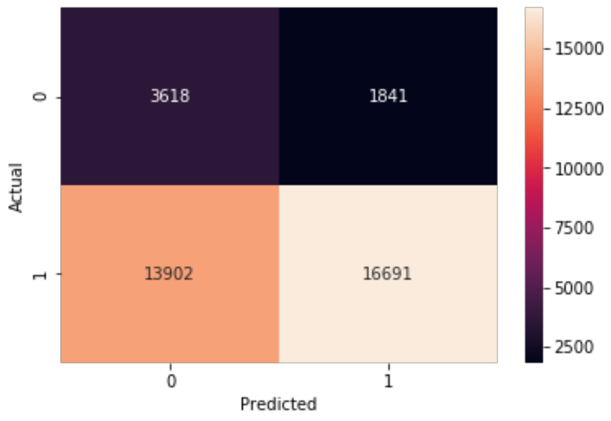
```
Text(33,0.5,'Actual')
```



In [132]:
```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, te
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.847
```

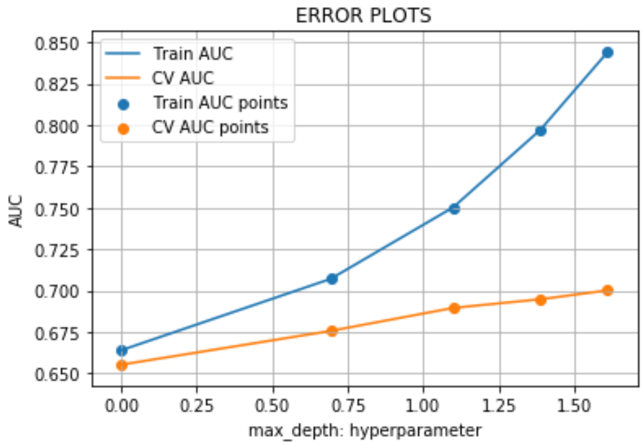
Out[132]:
```
Text(33,0.5,'Actual')
```



2.5.2 Applying XGBOOST on TFIDF, SET 2

In [101]:

```
from xgboost import XGBClassifier

import math
train_auc = []
cv_auc = []
K = {'max_depth':  [1,2,3,4,5]}
for i in K['max_depth']:
    neigh = XGBClassifier(max_depth=i)
    neigh.fit(Xh2, y_tr)

    y_train_pred = batch_predict(neigh, Xh2)
    y_cv_pred = batch_predict(neigh, Xh2_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['max_depth']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

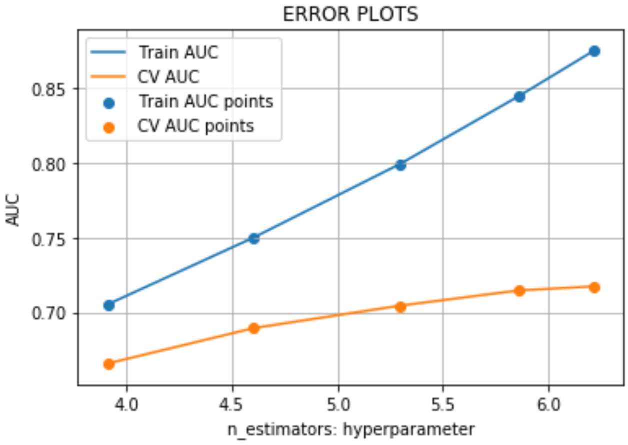


In [123]:

```
max_depth2=2
```

In [102]:

```
import math
train_auc = []
cv_auc = []
K = {'n_estimators':  [50,100,200,350,500]}
for i in K['n_estimators']:
    neigh = XGBClassifier(n_estimators=i)
    neigh.fit(Xh2, y_tr)

    y_train_pred = batch_predict(neigh, Xh2)
    y_cv_pred = batch_predict(neigh, Xh2_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['n_estimators']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("n_estimators: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



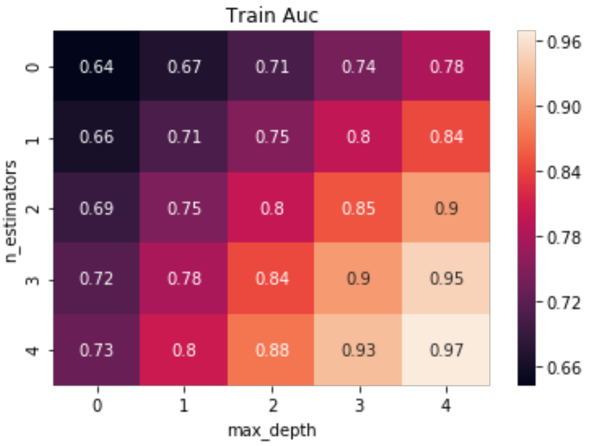
In [124]:

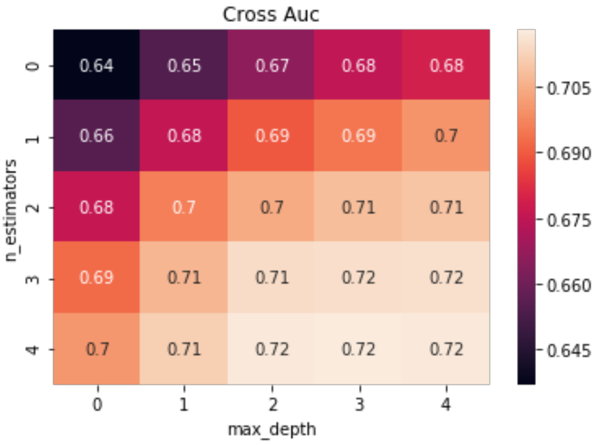
```
n_estimators2=50
```

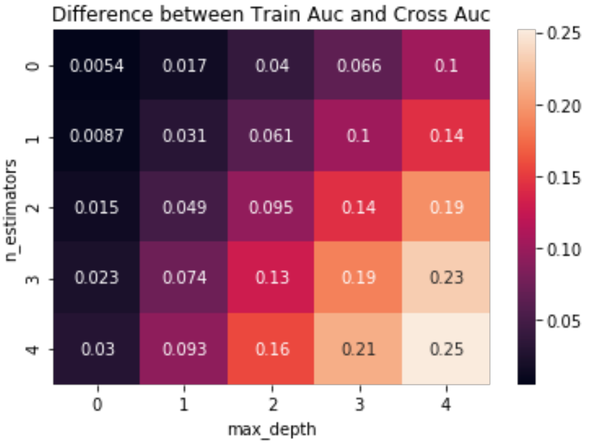
In [103]:
```
from sklearn.metrics import roc_curve, auc
K = {'n_estimators':  [50,100,200,350,500],'max_depth':  [1,2,3,4,5]}
diff=[]
t_auc=[]
c_auc=[]
for e in K['n_estimators']:
    l_dif=[]
    l_train=[]
    l_cross=[]
    for d in K['max_depth']:
        neigh = XGBClassifier(max_depth=d,n_estimators=e)
        neigh.fit(Xh2, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

        y_train_pred = batch_predict(neigh, Xh2)
        y_cv_pred = batch_predict(neigh, Xh2_cross)

        train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
        cv_fpr, cv_tpr, te_thresholds = roc_curve(y_cv, y_cv_pred)
        l_train.append(auc(train_fpr, train_tpr))
        l_cross.append(auc(cv_fpr, cv_tpr))
        no=auc(train_fpr, train_tpr)-auc(cv_fpr, cv_tpr)
        l_dif.append(no)

    diff.append(l_dif)
    t_auc.append(l_train)
    c_auc.append(l_cross)
```

In [104]:

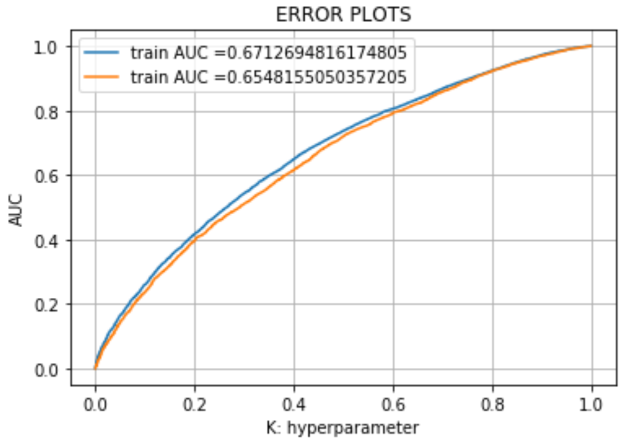
```
sns.heatmap(t_auc,annot=True)
plt.title(" Train Auc ")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(c_auc,annot=True)
plt.title("Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(diff,annot=True)
plt.title("Difference between Train Auc and Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()
```







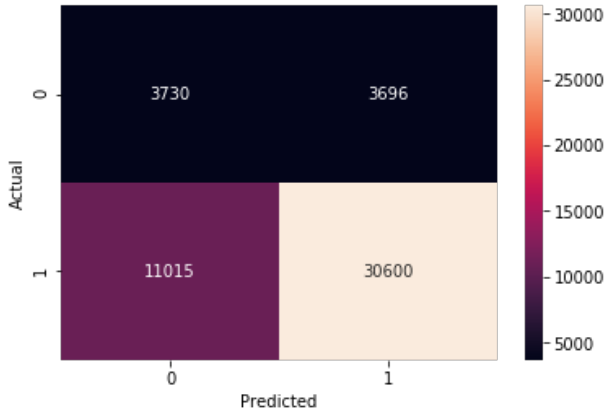
In [125]:
```
from sklearn.metrics import roc_curve, auc

neigh = XGBClassifier(max_depth=max_depth2,n_estimators=n_estimators2)
neigh.fit(Xh2, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

y_train_pred = batch_predict(neigh, Xh2)
y_test_pred = batch_predict(neigh, Xh2_test)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, 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_
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [126]:
```
import seaborn as sns
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_tr, predict(y_train_pred, tr_thresholds, tra
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2499947593162493 for threshold 0.832
```

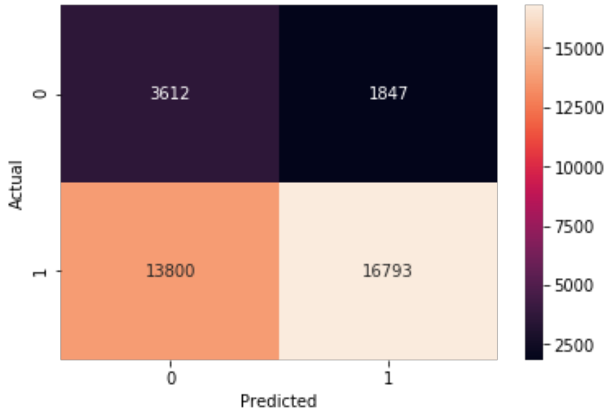
Out[126]:
```
Text(33,0.5,'Actual')
```



In [127]:
```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.851
```

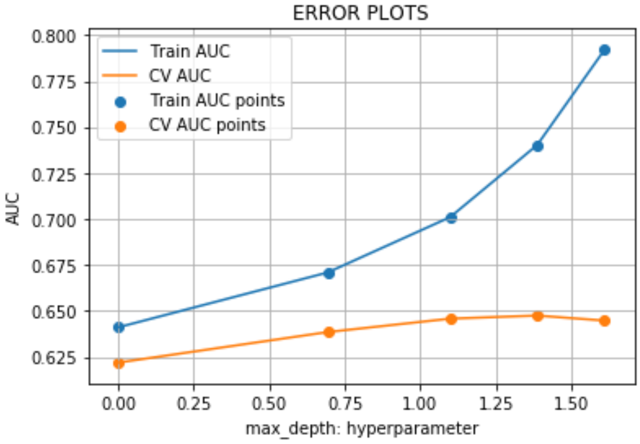
Out[127]:
```
Text(33,0.5,'Actual')
```



2.5.3 Applying XGBOOST on AVG W2V, SET 3

In [105]:

```
from xgboost import XGBClassifier

import math
train_auc = []
cv_auc = []
K = {'max_depth':  [1,2,3,4,5]}
for i in K['max_depth']:
    neigh = XGBClassifier(max_depth=i)
    neigh.fit(Xh3, y_tr)

    y_train_pred = batch_predict(neigh, Xh3)
    y_cv_pred = batch_predict(neigh, Xh3_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['max_depth']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

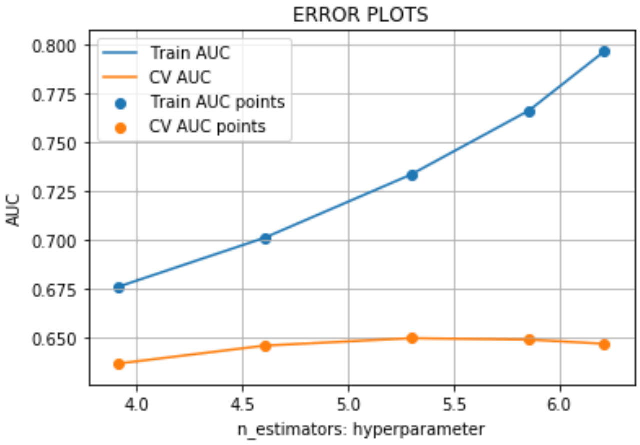


In [119]:

```
max_depth3=2
```

In [106]:

```
import math
train_auc = []
cv_auc = []
K = {'n_estimators':  [50,100,200,350,500]}
for i in K['n_estimators']:
    neigh = XGBClassifier(n_estimators=i)
    neigh.fit(Xh3, y_tr)

    y_train_pred = batch_predict(neigh, Xh3)
    y_cv_pred = batch_predict(neigh, Xh3_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['n_estimators']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("n_estimators: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



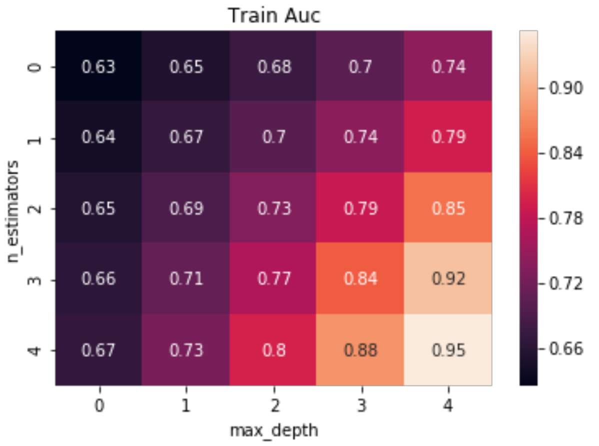
In [118]:

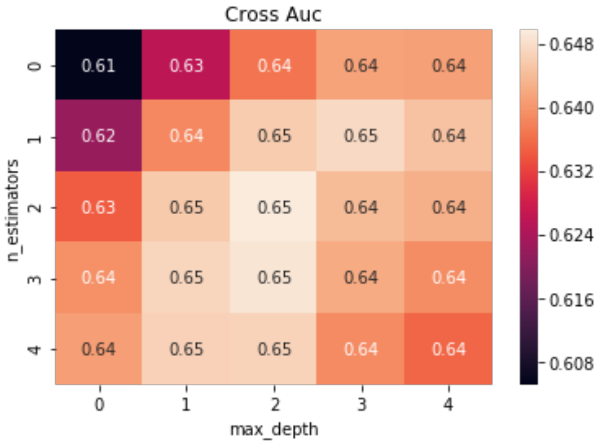
```
n_estimators3=50
```

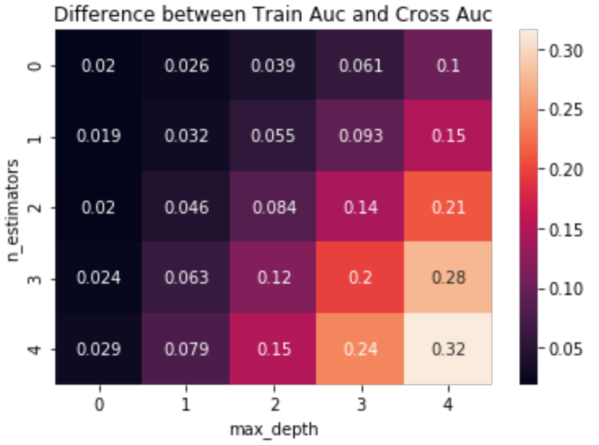
In [107]:

```
from sklearn.metrics import roc_curve, auc
K = {'n_estimators':  [50,100,200,350,500],'max_depth':  [1,2,3,4,5]}
diff=[]
t_auc=[]
c_auc=[]
for e in K['n_estimators']:
    l_dif=[]
    l_train=[]
    l_cross=[]
    for d in K['max_depth']:
        neigh = XGBClassifier(max_depth=d,n_estimators=e)
        neigh.fit(Xh3, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

        y_train_pred = batch_predict(neigh, Xh3)
        y_cv_pred = batch_predict(neigh, Xh3_cross)

        train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
        cv_fpr, cv_tpr, te_thresholds = roc_curve(y_cv, y_cv_pred)
        l_train.append(auc(train_fpr, train_tpr))
        l_cross.append(auc(cv_fpr, cv_tpr))
        no=auc(train_fpr, train_tpr)-auc(cv_fpr, cv_tpr)
        l_dif.append(no)

    diff.append(l_dif)
    t_auc.append(l_train)
    c_auc.append(l_cross)
```

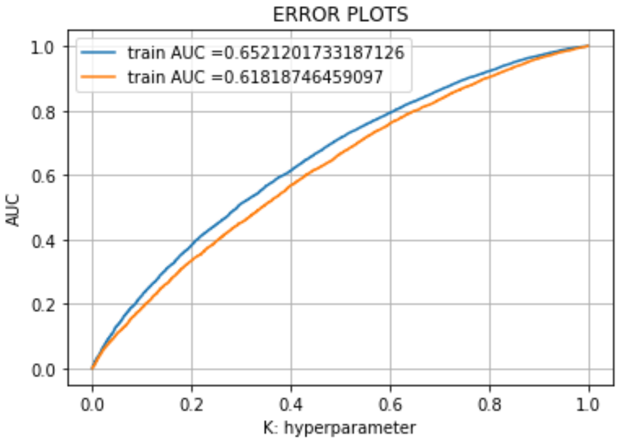
In [108]:
```
sns.heatmap(t_auc,annot=True)
plt.title(" Train Auc ")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(c_auc,annot=True)
plt.title("Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(diff,annot=True)
plt.title("Difference between Train Auc and Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()
```







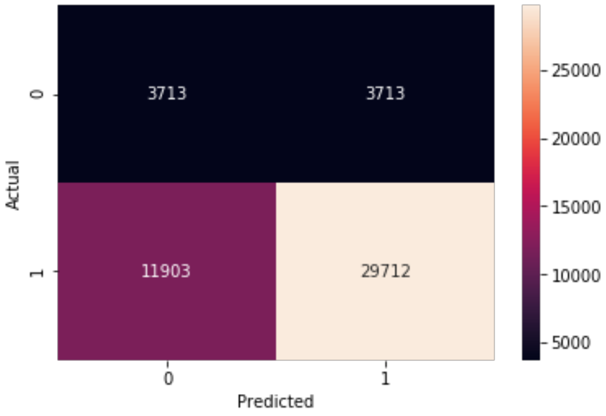
In [120]:
```
from sklearn.metrics import roc_curve, auc

neigh = XGBClassifier(max_depth=max_depth3,n_estimators=n_estimators3)
neigh.fit(Xh3, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

y_train_pred = batch_predict(neigh, Xh3)
y_test_pred = batch_predict(neigh, Xh3_test)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, 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_
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [121]:
```
import seaborn as sns
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_tr, predict(y_train_pred, tr_thresholds, tra
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.833
```

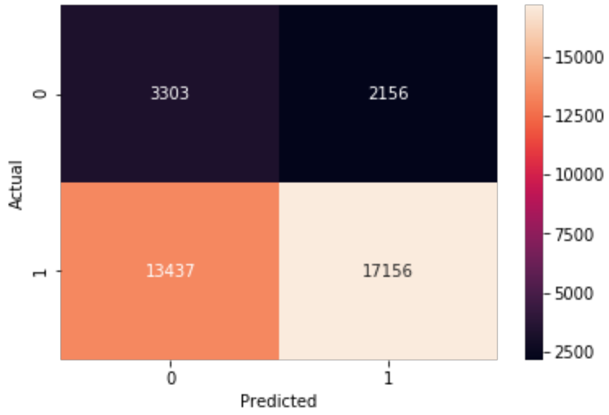
Out[121]:
```
Text(33,0.5,'Actual')
```



In [122]:
```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, tes
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

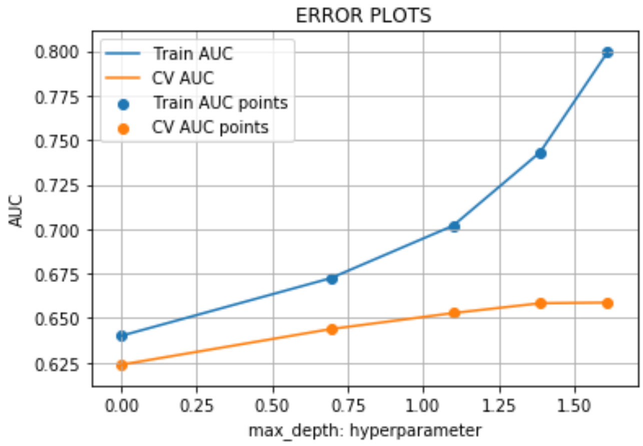
```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.853
```

Out[122]:
```
Text(33,0.5,'Actual')
```



2.5.4 Applying XGBOOST on TFIDF W2V, SET 4

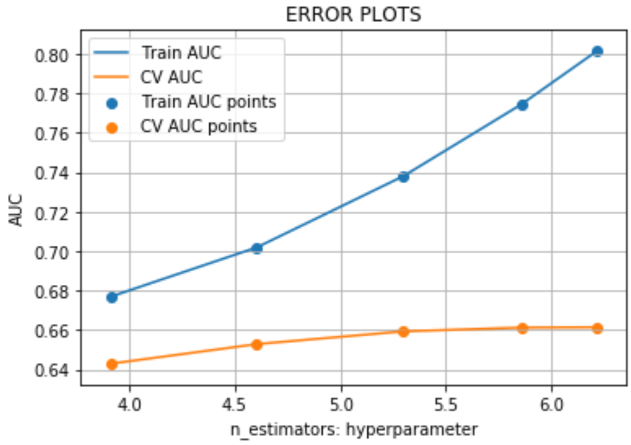
In [109]:
```
from xgboost import XGBClassifier

import math
train_auc = []
cv_auc = []
K = {'max_depth':  [1,2,3,4,5]}
for i in K['max_depth']:
    neigh = XGBClassifier(max_depth=i)
    neigh.fit(Xh4, y_tr)

    y_train_pred = batch_predict(neigh, Xh4)
    y_cv_pred = batch_predict(neigh, Xh4_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['max_depth']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("max_depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [113]:
```
max_depth4=2
```

In [110]:

```
import math
train_auc = []
cv_auc = []
K = {'n_estimators':  [50,100,200,350,500]}
for i in K['n_estimators']:
    neigh = XGBClassifier(n_estimators=i)
    neigh.fit(Xh4, y_tr)

    y_train_pred = batch_predict(neigh, Xh4)
    y_cv_pred = batch_predict(neigh, Xh4_cross)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_tr,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
log_K=[]
for l in K['n_estimators']:
    log_K.append(math.log(l))
plt.plot(log_K, train_auc, label='Train AUC')
plt.plot(log_K, cv_auc, label='CV AUC')

plt.scatter(log_K, train_auc, label='Train AUC points')
plt.scatter(log_K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("n_estimators: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



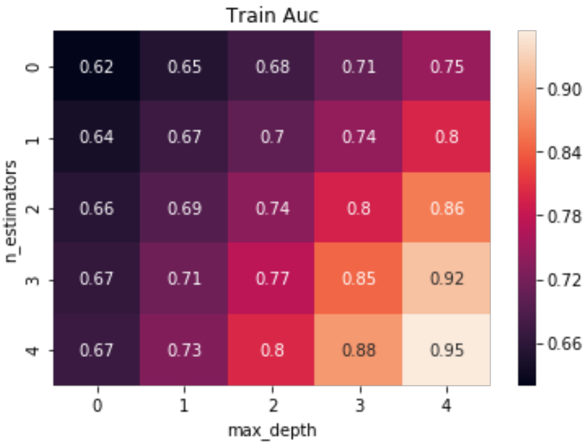
In [114]:

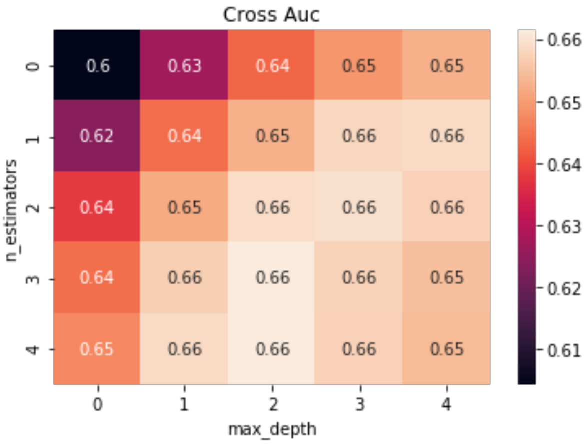
```
n_estimators4=50
```

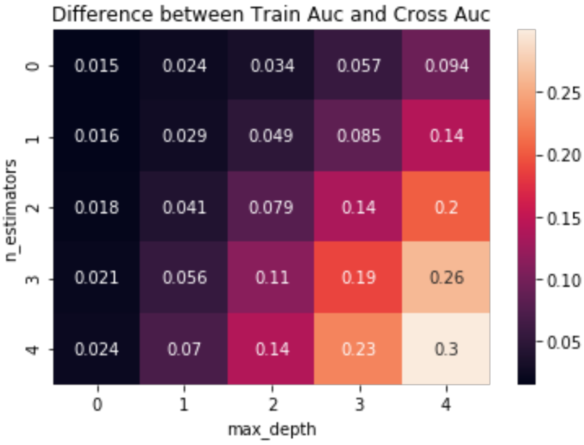
```
In [111]: from sklearn.metrics import roc_curve, auc
K = {'n_estimators':  [50,100,200,350,500],'max_depth':  [1,2,3,4,5]}
diff=[]
t_auc=[]
c_auc=[]
for e in K['n_estimators']:
    l_dif=[]
    l_train=[]
    l_cross=[]
    for d in K['max_depth']:
        neigh = XGBClassifier(max_depth=d,n_estimators=e)
        neigh.fit(Xh4, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

        y_train_pred = batch_predict(neigh, Xh4)
        y_cv_pred = batch_predict(neigh, Xh4_cross)

        train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
        cv_fpr, cv_tpr, te_thresholds = roc_curve(y_cv, y_cv_pred)
        l_train.append(auc(train_fpr, train_tpr))
        l_cross.append(auc(cv_fpr, cv_tpr))
        no=auc(train_fpr, train_tpr)-auc(cv_fpr, cv_tpr)
        l_dif.append(no)

    diff.append(l_dif)
    t_auc.append(l_train)
    c_auc.append(l_cross)
```

In [112]:

```
sns.heatmap(t_auc,annot=True)
plt.title(" Train Auc ")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(c_auc,annot=True)
plt.title("Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()

sns.heatmap(diff,annot=True)
plt.title("Difference between Train Auc and Cross Auc")
plt.xlabel("max_depth")
plt.ylabel("n_estimators")
plt.show()
```

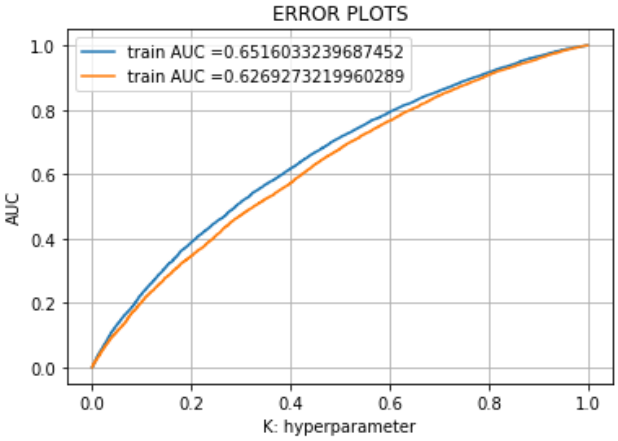






In [115]:

```
from sklearn.metrics import roc_curve, auc

neigh = XGBClassifier(max_depth=max_depth4,n_estimators=n_estimators4)
neigh.fit(Xh4, y_tr)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
# not the predicted outputs

y_train_pred = batch_predict(neigh, Xh4)
y_test_pred = batch_predict(neigh, Xh4_test)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, 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_
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



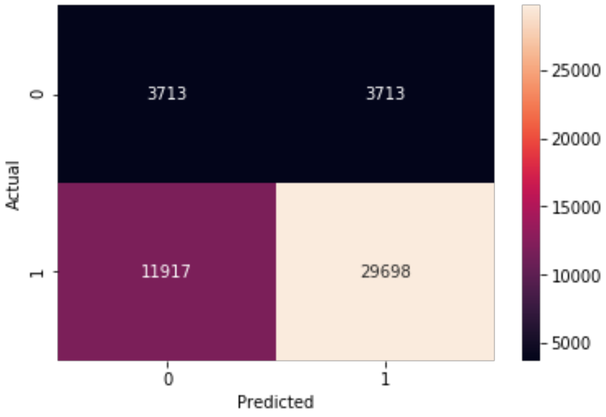
In [116]:

```
import seaborn as sns
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_tr, predict(y_train_pred, tr_thresholds, tra
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.834
```

Out[116]:

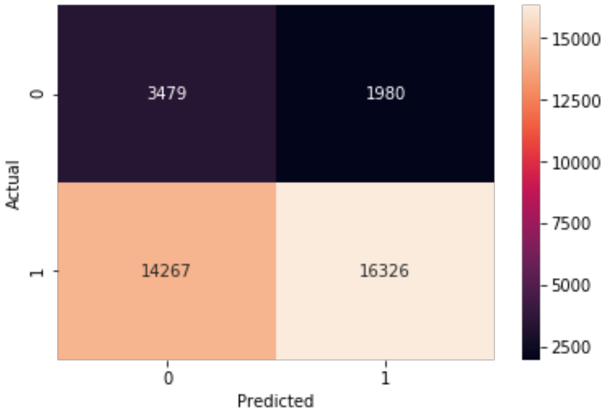
```
Text(33,0.5,'Actual')
```



```
In [117]: print("Test confusion matrix")
          sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, te
          plt.xlabel("Predicted")
          plt.ylabel("Actual")
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.85
```

```
Out[117]: Text(33,0.5,'Actual')
```



3. Conclusion

In [133]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Set","Vectorizer", "Model", "max_depth","min_samples_spli

x.add_row(["set 1","BOW", "Random Forest",3,350, 0.682])
x.add_row(["set 2","TFIDF", "Random Forest",4,500, 0.689])
x.add_row(["set 3","W2V", "Random Forest",3,500, 0.624])
x.add_row(["set 4","TFIDF W2V", "Random Forest", 3,500, 0.627])
x.add_row(["set 1","BOW", "GBDT", 2,50, 0.647])
x.add_row(["set 2","TFIDF", "GBDT",2,50, 0.654])
x.add_row(["set 3","W2V", "GBDT",2,50, 0.618])
x.add_row(["set 4","TFIDF W2V", "GBDT",2,50, 0.626])
print(x)
```

```
+-------+------------+---------------+-----------+-------------------+----
---+
|  Set  | Vectorizer |     Model     | max_depth | min_samples_split |  AU
C  |
+-------+------------+---------------+-----------+-------------------+----
---+
| set 1 |    BOW     | Random Forest |     3     |        350        |
0.682 |
| set 2 |   TFIDF    | Random Forest |     4     |        500        |
0.689 |
| set 3 |    W2V     | Random Forest |     3     |        500        |
0.624 |
| set 4 | TFIDF W2V  | Random Forest |     3     |        500        |
0.627 |
| set 1 |    BOW     |      GBDT     |     2     |         50        |
0.647 |
| set 2 |   TFIDF    |      GBDT     |     2     |         50        |
0.654 |
| set 3 |    W2V     |      GBDT     |     2     |         50        |
0.618 |
| set 4 | TFIDF W2V  |      GBDT     |     2     |         50        |
0.626 |
+-------+------------+---------------+-----------+-------------------+----
---+
```

In [ ]: