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). **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!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*

Feature	Description
`project_essay_4`	Fourth application essay
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

* See the section **Notes on the Essay Data** for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p036502`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved.

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',nrows=50000)
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 (50000, 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

In [5]:

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


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


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


project_data.head(2)
```

Out[5]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	L

In [6]:

```
project_grade_category = []

for i in range(len(project_data)):
    a = project_data["project_grade_category"][i].replace(" ", "_")
    project_grade_category.append(a)
```

In [7]:

```
project_grade_category[0:5]
```

Out[7]:

```
['Grades_PreK-2', 'Grades_6-8', 'Grades_6-8', 'Grades_PreK-2', 'Grades_PreK-2']
```

In [8]:

```
project_data.drop(['project_grade_category'], axis=1, inplace=True)
```

In [9]:

```
project_data["project_grade_category"] = project_grade_category
```

In [10]:

```
project_data.head(5)
```

Out[10]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_subject_categories	proje
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Applied Learning	
						2016-		

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_subject_categories	proje
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	04-27 01:05:25	Literacy & Language	
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Math & Science, History & Civics	Ma
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Literacy & Language	
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	IL	2016-04-27 07:19:44	Literacy & Language	

1.2 preprocessing of `project_subject_categories`

In [11]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        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 [12]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
```

```
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        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/22898595/4084039
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

1.4 Clean Titles (Text preprocessing)

In [13]:

```
# 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', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [14]:

```
# 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 [15]:

```
clean_titles = []

for titles in tqdm(project_data["project_title"]):
    title = decontracted(titles)
    title = title.replace('\\r', ' ')
    title = title.replace('\\"', ' ')
    title = title.replace('\\n', ' ')
    title = re.sub('[^A-Za-z0-9]+', ' ', title)
    title = ' '.join(f for f in title.split() if f not in stopwords)
    clean_titles.append(title.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000
[00:03<00:00, 12729.16it/s]
```

In [16]:

```
project_data["clean_titles"] = clean_titles
```

In [17]:

```
project_data.drop(['project_title'], axis=1, inplace=True)
```

1.5 Introducing new feature "Number of Words in Title"

In [18]:

```
title_word_count = []
```

In [19]:

```
for a in project_data["clean_titles"] :
    b = len(a.split())
    title_word_count.append(b)
```

In [20]:

```
project_data["title_word_count"] = title_word_count
```

In [21]:

```
project_data.head(5)
```

Out[21]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_essay_1	project_essay_2
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	I recently read an article about giving studen...	I teach at a low-income (Title 1) school. Ever...
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	My students crave challenge, they eat obstacle...	We are an urban, public k-5 elementary school....
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	It's the end of the school year. Routines have...	My students desire challenges, movement, and c...
						2016-	Never has	Our Language

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_essay_1	project_essay_2
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	04-27 02:04:15	society so rapidly changed Technolo...	Arts and Social Justice Magnet Sc...
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	IL	2016-04-27 07:19:44	My students yearn for a classroom environment ...	I have the privilege of teaching an incredible...

1.6 Combine 4 Project essays into 1 Essay

In [22]:

```
# 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)
```

1.7 Clean Essays (Text preprocessing)

In [23]:

```
clean_essay = []

for ess in tqdm(project_data["essay"]):
    ess = decontracted(ess)
    ess = ess.replace('\\r', ' ')
    ess = ess.replace('\\"', ' ')
    ess = ess.replace('\\n', ' ')
    ess = re.sub('[^A-Za-z0-9]+', ' ', ess)
    ess = ' '.join(f for f in ess.split() if f not in stopwords)
    clean_essay.append(ess.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████████| 50000/50000 [01:
09<00:00, 715.03it/s]
```

In [24]:

```
project_data["clean_essays"] = clean_essay
```

In [25]:

```
project_data.drop(['essay'], axis=1, inplace=True)
```

1.8 Introducing new feature "Number of Words in Essay"

In [26]:

```
essay_word_count = []
```

In [27]:

```
for ess in project_data["clean_essays"] :
    c = len(ess.split())
    essay_word_count.append(c)
```

In [28]:

```
project_data["essay_word_count"] = essay_word_count
```

In [29]:

```
project_data.head(5)
```

Out[29]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_essay_1	project_essay_2
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	I recently read an article about giving studen...	I teach at a low-income (Title 1) school. Ever...
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	My students crave challenge, they eat obstacle...	We are an urban, public k-5 elementary school....
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	It's the end of the school year. Routines have...	My students desire challenges, movement, and c...
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Never has society so rapidly changed. Technolo...	Our Language Arts and Social Justice Magnet Sc...
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	IL	2016-04-27 07:19:44	My students yearn for a classroom environment ...	I have the privilege of teaching an incredible...

1.10 Test - Train Split

In [30]:

```
# train test split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(project_data,
project_data['project_is_approved'], test_size=0.33, stratify = project_data['project_is_approved'
])
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

Preparing data for models

In [31]:

```
project_data.columns
```

Out[31]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'project_grade_category', 'clean_categories', 'clean_subcategories',
       'clean_titles', 'title_word_count', 'clean_essays', 'essay_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
```

2.1 Vectorizing Categorical data

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

Response coding

In [32]:

```
def get_gv_fea_dict(alpha, feature, df):

    value_count = X_train[feature].value_counts()

    # gv_dict :
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in whole data
    for i, denominator in value_count.items():

        vec = []
        for k in range(1,3):

            cls_cnt = X_train.loc[(X_train['project_is_approved']==k) & (X_train[feature]==i)]

            # cls_cnt.shape[0](numerator) will contain the number of time that particular feature occured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))


        gv_dict[i]=vec
    return gv_dict


def get_gv_feature(alpha, feature, df):

    gv_dict = get_gv_fea_dict(alpha, feature, df)

    value_count = X_train[feature].value_counts()

    gv_fea = []

    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/2,1/2])

    return gv_fea
```

Response Coding - Clean Categories of Projects

In [33]:

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_categories_responseCoding = np.array(get_gv_feature(alpha, "clean_categories", X_train))
# test gene feature
test_categories_feature_responseCoding = np.array(get_gv_feature(alpha, "clean_categories", X_test)
)
# cross validation gene feature
cv_categories_feature_responseCoding = np.array(get_gv_feature(alpha, "clean_categories", X_cv))
```

In [34]:

```
print("Shape of matrix of Train data after response coding ",train_categories_responseCoding.shape
)
print("Shape of matrix of Test data after response coding ",test_categories_feature_responseCoding
.shape)
print("Shape of matrix of CV data after response coding  ",cv_categories_feature_responseCoding.sh
ape)
```

```
Shape of matrix of Train data after response coding  (22445, 2)
Shape of matrix of Test data after response coding  (16500, 2)
Shape of matrix of CV data after response coding   (11055, 2)
```

Response Coding - Clean Sub Categories of Projects

In [35]:

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
import numpy as np
alpha = 1
# train gene feature
train_subcategories_responseCoding = np.array(get_gv_feature(alpha, "clean_subcategories", X_train)
)
# test gene feature
test_subcategories_feature_responseCoding = np.array(get_gv_feature(alpha, "clean_subcategories", X
_test))
# cross validation gene feature
cv_subcategories_feature_responseCoding = np.array(get_gv_feature(alpha, "clean_subcategories", X_c
v))
```

In [36]:

```
print("Shape of matrix of Train data after response coding ",train_subcategories_responseCoding.sh
ape)
print("Shape of matrix of Test data after response coding 
",test_subcategories_feature_responseCoding.shape)
print("Shape of matrix of CV data after response coding  ",cv_subcategories_feature_responseCoding
.shape)
```

```
Shape of matrix of Train data after response coding  (22445, 2)
Shape of matrix of Test data after response coding  (16500, 2)
Shape of matrix of CV data after response coding   (11055, 2)
```

Response Coding - School States

In [37]:

```
# alpha is used for laplace smoothing
import numpy as np

alpha = 1
# train gene feature
train_state_categories_responseCoding = np.array(get_gv_feature(alpha, "school_state", X_train))
# test gene feature
test_state_categories_feature_responseCoding = np.array(get_gv_feature(alpha, "school_state", X_tes
t))
# cross validation gene feature
cv_state_categories_feature_responseCoding = np.array(get_gv_feature(alpha, "school_state", X_cv))
```

In [38]:

```
print("Shape of matrix of Train data after response coding ",train_state_categories_responseCoding
.shape)
print("Shape of matrix of Test data after response coding 
",test_state_categories_feature_responseCoding.shape)
print("Shape of matrix of CV data after response coding 
",cv_state_categories_feature_responseCoding.shape)
```

```
Shape of matrix of Train data after response coding  (22445, 2)
Shape of matrix of Test data after response coding  (16500, 2)
Shape of matrix of CV data after response coding   (11055, 2)
```

Response Coding - Project Grade Category

In [39]:

```
# alpha is used for laplace smoothing
import numpy as np

alpha = 1
# train gene feature
train_grade_categories_responseCoding = np.array(get_gv_feature(alpha, "project_grade_category", X_
train))
# test gene feature
test_grade_categories_feature_responseCoding = np.array(get_gv_feature(alpha,
"project_grade_category", X_test))
# cross validation gene feature
cv_grade_categories_feature_responseCoding = np.array(get_gv_feature(alpha,
"project_grade_category", X_cv))
```

In [40]:

```
print("Shape of matrix of Train data after response coding ",train_grade_categories_responseCoding
.shape)
print("Shape of matrix of Test data after response coding
",test_grade_categories_feature_responseCoding.shape)
print("Shape of matrix of CV data after response coding
",cv_grade_categories_feature_responseCoding.shape)
```

```
Shape of matrix of Train data after response coding  (22445, 2)
Shape of matrix of Test data after response coding  (16500, 2)
Shape of matrix of CV data after response coding   (11055, 2)
```

Response Coding - Teacher Prefix Category

In [41]:

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
import numpy as np

alpha = 1
# train gene feature
train_teacher_prefix_responseCoding = np.array(get_gv_feature(alpha, "teacher_prefix", X_train))
# test gene feature
test_teacher_prefix_feature_responseCoding = np.array(get_gv_feature(alpha, "teacher_prefix", X_tes
t))
# cross validation gene feature
cv_teacher_prefix_feature_responseCoding = np.array(get_gv_feature(alpha, "teacher_prefix", X_cv))
```

In [42]:

```
print("Shape of matrix of Train data after response coding ",train_teacher_prefix_responseCoding.s
hape)
print("Shape of matrix of Test data after response coding
",test_teacher_prefix_feature_responseCoding.shape)
print("Shape of matrix of CV data after response coding
",cv_teacher_prefix_feature_responseCoding.shape)
```

```
Shape of matrix of Train data after response coding  (22445, 2)
Shape of matrix of Test data after response coding  (16500, 2)
Shape of matrix of CV data after response coding   (11055, 2)
```

2.2 Vectorizing Text data

A) Bag of Words (BOW) with min_df=10

Bag of words - Train Data - Essays

In [43]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).

vectorizer_bow_essay = CountVectorizer(min_df=10)

vectorizer_bow_essay.fit(X_train["clean_essays"])

text_bow_train = vectorizer_bow_essay.transform(X_train["clean_essays"])

print("Shape of matrix after one hot encoding ",text_bow_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 12113)
```

Bag of words - Test Data - Essays

In [44]:

```
text_bow_test = vectorizer_bow_essay.transform(X_test["clean_essays"])
print("Shape of matrix after one hot encoding ",text_bow_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 12113)
```

Bag of words - Cross Validation Data - Essays

In [45]:

```
text_bow_cv = vectorizer_bow_essay.transform(X_cv["clean_essays"])
print("Shape of matrix after one hot encoding ",text_bow_cv.shape)
```

```
Shape of matrix after one hot encoding  (24155, 12113)
```

Bag of words - Train Data - Titles

In [46]:

```
vectorizer_bow_title = CountVectorizer(min_df=10)

vectorizer_bow_title.fit(X_train["clean_titles"])

title_bow_train = vectorizer_bow_title.transform(X_train["clean_titles"])
print("Shape of matrix after one hot encoding ",title_bow_train.shape)
```

```
Shape of matrix after one hot encoding  (49041, 2089)
```

Bag of words - Test Data - Titles

In [47]:

```
title_bow_test = vectorizer_bow_title.transform(X_test["clean_titles"])
print("Shape of matrix after one hot encoding ",title_bow_test.shape)
```

```
Shape of matrix after one hot encoding  (36052, 2089)
```

Bag of words - Cross Validation Data - Titles

In [48]:

```
title_bow_cv = vectorizer_bow_title.transform(X_cv["clean_titles"])
print("Shape of matrix after one hot encoding ",title_bow_cv.shape)
```

```
Shape of matrix after one hot encoding  (24155, 2089)
```

B) TFIDF vectorizer with min_df=10

TFIDF - Train Data - Essays

In [146]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

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

text_tfidf_train = vectorizer_tfidf_essay.transform(X_train["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (22445, 8908)
```

TFIDF - Test Data - Essays

In [147]:

```
text_tfidf_test = vectorizer_tfidf_essay.transform(X_test["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (16500, 8908)
```

TFIDF - Cross Validation Data - Essays

In [148]:

```
text_tfidf_cv = vectorizer_tfidf_essay.transform(X_cv["clean_essays"])
print("Shape of matrix after one hot encoding ",text_tfidf_cv.shape)
```

```
Shape of matrix after one hot encoding  (11055, 8908)
```

TFIDF - Train Data - Titles

In [149]:

```
vectorizer_tfidf_titles = TfidfVectorizer(min_df=10)

vectorizer_tfidf_titles.fit(X_train["clean_titles"])
title_tfidf_train = vectorizer_tfidf_titles.transform(X_train["clean_titles"])
print("Shape of matrix after one hot encoding ",title_tfidf_train.shape)
```

```
Shape of matrix after one hot encoding  (22445, 1227)
```

TFIDF - Test Data - Titles

In [150]:

```
title_tfidf_test = vectorizer_tfidf_titles.transform(X_test["clean_titles"])
print("Shape of matrix after one hot encoding ",title_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (16500, 1227)
```

TFIDF - Cross Validation Data - Titles

In [151]:

```
title_tfidf_cv = vectorizer_tfidf_titles.transform(X_cv["clean_titles"])
print("Shape of matrix after one hot encoding ",title_tfidf_cv.shape)
```

```
Shape of matrix after one hot encoding  (11055, 1227)
```

C) Using Pretrained Models : AVG W2V

In [43]:

```
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039

def loadGloveModel(gloveFile):

    print ("Loading Glove Model")

    f = open(gloveFile,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
```

In [44]:

```
model = loadGloveModel('glove.42B.300d.txt')
```

```
Loading Glove Model
```

```
1917495it [12:02, 2653.89it/s]
```

```
Done. 1917495  words loaded!
```

In [45]:

```
words_train_essays = []

for i in X_train["clean_essays"] :
    words_train_essays.extend(i.split(' '))
```

In [46]:

```
## Find the total number of words in the Train data of Essays.

print("All the words in the corpus", len(words_train_essays))
```

```
All the words in the corpus 3384119
```

In [47]:

```
## Find the unique words in this set of words

words_train_essay = set(words_train_essays)
print("the unique words in the corpus", len(words_train_essay))
```

```
the unique words in the corpus 30590
```

In [48]:

```
## Find the words present in both Glove Vectors as well as our corpus.

inter_words = set(model.keys()).intersection(words_train_essay)

print("The number of words that are present in both glove vectors and our corpus are {} which \
is nearly {}% ".format(len(inter_words), np.round((float(len(inter_words))/len(words_train_essay))
*100)))
```

```
The number of words that are present in both glove vectors and our corpus are 28901 which is
nearly 94.0%
```

In [49]:

```
words_corpus_train_essay = {}

words_glove = set(model.keys())

for i in words_train_essay:
    if i in words_glove:
        words_corpus_train_essay[i] = model[i]

print("word 2 vec length", len(words_corpus_train_essay))
```

```
word 2 vec length 28901
```

In [50]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/

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

In [51]:

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

Train - Essays

In [52]:

```
# average Word2Vec
# compute average word2vec for each review.

avg_w2v_vectors_train = [];

for sentence in tqdm(X_train["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
print(len(avg_w2v_vectors_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████| 22445/22445
[00:19<00:00, 1176.38it/s]
```

```
22445
300
```

Test - Essays

In [53]:

```
# average Word2Vec
# compute average word2vec for each review.

avg_w2v_vectors_test = [];

for sentence in tqdm(X_test["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)

print(len(avg_w2v_vectors_test))
print(len(avg_w2v_vectors_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████| 16500/16500
[00:15<00:00, 1098.25it/s]
```

```
16500
300
```

Cross-Validation - Essays

In [54]:

```
avg_w2v_vectors_cv = [];

for sentence in tqdm(X_cv["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv.append(vector)

print(len(avg_w2v_vectors_cv))
print(len(avg_w2v_vectors_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████| 11055/11055
[00:09<00:00, 1164.33it/s]
```

```
11055
300
```

```
300
```

Train - Titles

In [55]:

```
# Similarly you can vectorize for title also

avg_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train["clean_titles"]): # for each title
    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_titles_train.append(vector)

print(len(avg_w2v_vectors_titles_train))
print(len(avg_w2v_vectors_titles_train[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████████| 22445/22445
[00:01<00:00, 21913.39it/s]
```

```
22445
300
```

Test - Titles

In [56]:

```
# Similarly you can vectorize for title also

avg_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test["clean_titles"]): # for each title
    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_titles_test.append(vector)

print(len(avg_w2v_vectors_titles_test))
print(len(avg_w2v_vectors_titles_test[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████████| 16500/16500
[00:01<00:00, 15671.55it/s]
```

```
16500
300
```

Cross-Validation - Titles

In [57]:

```
# Similarly you can vectorize for title also

avg_w2v_vectors_titles_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv["clean_titles"]): # for each title
    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_titles_cv.append(vector)

print(len(avg_w2v_vectors_titles_cv))
print(len(avg_w2v_vectors_titles_cv[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 11055/11055
[00:01<00:00, 9948.42it/s]
```

```
11055
300
```

In [ ]:

```

```

D) Using Pretrained Models: TFIDF weighted W2V

Train - Essays

In [73]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train["clean_essays"])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [74]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|███████████████████████████████████████████████████████████████████████████| 22445/22445 [03:
21<00:00, 120.39it/s]
```

```
22445
300
```

Test - Essays

In [75]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

print(len(tfidf_w2v_vectors_test))
print(len(tfidf_w2v_vectors_test[0]))
```

```
100%|███████████████████████████████████████████████████████████████████████████| 16500/16500 [02:
11<00:00, 115.88it/s]
```

```
16500
300
```

Cross-Validation - Essays

In [76]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv["clean_essays"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_cv.append(vector)

print(len(tfidf_w2v_vectors_cv))
print(len(tfidf_w2v_vectors_cv[0]))
```

```
100%|███████████████████████████████████████████████████████████████████████████| 11055/11055 [01:
28<00:00, 125.59it/s]
```

```
11055
300
```

Train - Titles

In [77]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train["clean_titles"])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [78]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_train = [];

for sentence in tqdm(X_train["clean_titles"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_train.append(vector)

print(len(tfidf_w2v_vectors_titles_train))
print(len(tfidf_w2v_vectors_titles_train[0]))
```

```
100%|███████████████████████████████████████████████████████████████████████████| 22445/22445
[00:07<00:00, 2955.66it/s]
```

```
22445
300
```

Test - Titles

In [79]:

```
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_test = [];

for sentence in tqdm(X_test["clean_titles"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_test.append(vector)

print(len(tfidf_w2v_vectors_titles_test))
print(len(tfidf_w2v_vectors_titles_test[0]))
```

```
100%|███████████████████████████████████████████████████████████████████████████| 16500/16500
[00:05<00:00, 2850.02it/s]
```

```
16500
300
```

Cross-Validation - Titles

In [80]:

```
# compute average word2vec for each review.
# compute average word2vec for each review.

tfidf_w2v_vectors_titles_cv = [];

for sentence in tqdm(X_cv["clean_titles"]): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_cv.append(vector)

print(len(tfidf_w2v_vectors_titles_cv))
print(len(tfidf_w2v_vectors_titles_cv[0]))
```

```
100%|██████████████████████████████████████| 11055/11055
[00:03<00:00, 2790.26it/s]
```

```
11055
300
```

2.3 Vectorizing Numerical features

In [58]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in
-one-step
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)
```

Out[58]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [59]:

```
# join two dataframes in python:
X_train = pd.merge(X_train, price_data, on='id', how='left')
X_test = pd.merge(X_test, price_data, on='id', how='left')
X_cv = pd.merge(X_cv, price_data, on='id', how='left')
```

A) Price

In [60]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer()

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

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

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

print("After vectorizations")
print(price_train.shape, y_train.shape)
print(price_cv.shape, y_cv.shape)
print(price_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

B) Quantity

In [61]:

```
normalizer = Normalizer()

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

normalizer.fit(X_train['quantity'].values.reshape(-1,1))

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

print("After vectorizations")
print(quantity_train.shape, y_train.shape)
print(quantity_cv.shape, y_cv.shape)
print(quantity_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

C) Number of Projects previously proposed by Teacher

In [62]:

```
normalizer = Normalizer()

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

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

prev_projects_train = normalizer.transform(X_train['teacher_number_of_previously_posted_projects']
.values.reshape(-1,1))
prev_projects_cv =
normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
prev_projects_test = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].v
alues.reshape(-1,1))
```

```
print("After vectorizations")
print(prev_projects_train.shape, y_train.shape)
print(prev_projects_cv.shape, y_cv.shape)
print(prev_projects_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

D) Title word Count

In [63]:

```
normalizer = Normalizer()

normalizer.fit(X_train['title_word_count'].values.reshape(-1,1))

title_word_count_train = normalizer.transform(X_train['title_word_count'].values.reshape(-1,1))
title_word_count_cv = normalizer.transform(X_cv['title_word_count'].values.reshape(-1,1))
title_word_count_test = normalizer.transform(X_test['title_word_count'].values.reshape(-1,1))

print("After vectorizations")
print(title_word_count_train.shape, y_train.shape)
print(title_word_count_cv.shape, y_cv.shape)
print(title_word_count_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

E) Essay word Count

In [64]:

```
normalizer = Normalizer()

normalizer.fit(X_train['essay_word_count'].values.reshape(-1,1))

essay_word_count_train = normalizer.transform(X_train['essay_word_count'].values.reshape(-1,1))
essay_word_count_cv = normalizer.transform(X_cv['essay_word_count'].values.reshape(-1,1))
essay_word_count_test = normalizer.transform(X_test['essay_word_count'].values.reshape(-1,1))

print("After vectorizations")
print(essay_word_count_train.shape, y_train.shape)
print(essay_word_count_cv.shape, y_cv.shape)
print(essay_word_count_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Assignment 9: RF and GBDT

> 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: use probability values), numerical features + project_title(BOW) + preprocessed_eassay (BOW)
    - Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
    - Set 3: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
    - Set 4: categorical(instead of one hot encoding, try response coding: 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 value
    - find the best hyper paramter using k-fold cross validation/simple cross validation data
    - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task

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

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

or

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

      seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**
    - You can choose either of the plotting techniques: 3d plot or heat map
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

4. **Conclusion**
    - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

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 [ ]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((train_categories_responseCoding, train_subcategories_responseCoding,
train_state_categories_responseCoding, train_grade_categories_responseCoding,
train_teacher_prefix_responseCoding, price_train, quantity_train, prev_projects_train,
title_word_count_train, essay_word_count_train, title_bow_train, text_bow_train)).tocsr()
X_te = hstack((test_categories_feature_responseCoding, test_subcategories_feature_responseCoding,
test_state_categories_feature_responseCoding, test_grade_categories_feature_responseCoding,
test_teacher_prefix_feature_responseCoding, price_test, quantity_test, prev_projects_test,
title_word_count_test, essay_word_count_test, title_bow_test, text_bow_test)).tocsr()
X_cv = hstack((cv_categories_feature_responseCoding, cv_subcategories_feature_responseCoding,
cv_state_categories_feature_responseCoding, cv_grade_categories_feature_responseCoding,
cv_teacher_prefix_feature_responseCoding, price_cv, quantity_cv, prev_projects_cv,
title_word_count_cv, essay_word_count_cv,  title_bow_cv, text_bow_cv)).tocsr()
```

In [57]:

```
print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 14217) (49041,)
(24155, 14217) (24155,)
(36052, 14217) (36052,)
====================================================================================================
```

Hyperparameter Tunning GridSearch

In [96]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
import seaborn as sea
```

In [97]:

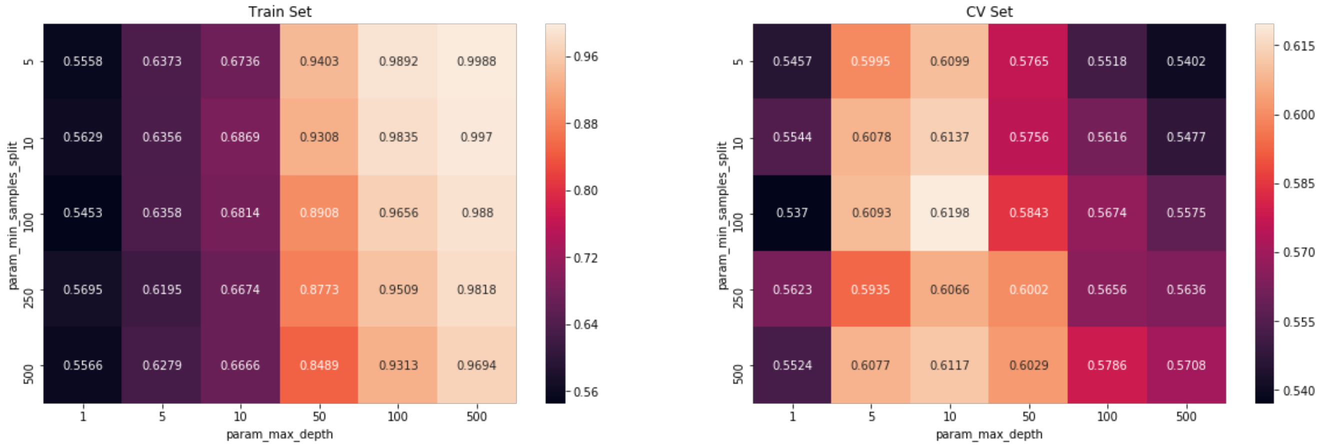
```
RF = RandomForestClassifier()

parameters = {'max_depth': [1, 5, 10, 50, 100, 500], 'min_samples_split': [5, 10, 100,250, 500]}

classifier = GridSearchCV(RF, parameters, cv=3, scoring='roc_auc')
classifier.fit(X_tr, y_train)
```

Out[97]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'max_depth': [1, 5, 10, 50, 100, 500], 'min_samples_split': [5, 10, 100, 250, 5
00]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [98]:

```
max_scores = pd.DataFrame(classifier.cv_results_).groupby(['param_min_samples_split',
'param_max_depth']).max().unstack()[['mean_test_score', 'mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(20,6))

sea.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sea.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])

ax[0].set_title('Train Set')
ax[1].set_title('CV Set')

plt.show()
```

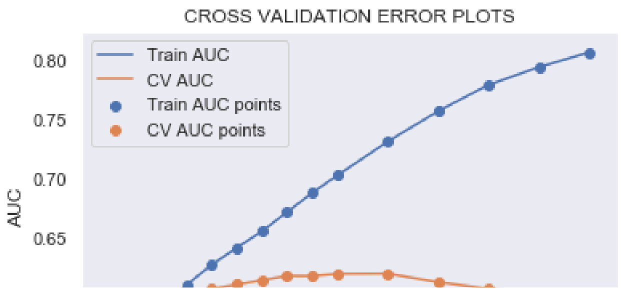


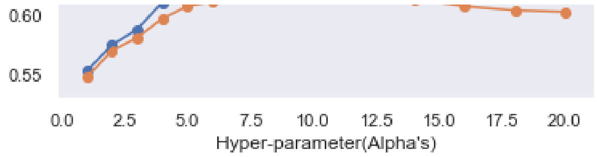
Observations :
From above we can conclude the best hyperparameter of max depth is '10' and sample split is '100'</b>

Trainng Model Using Best HyperParameter

In [109]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
import numpy as np
from sklearn import tree
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.
y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.
"""
train_auc = []
cv_auc = []
depth = [1, 2, 3, 4, 5, 6, 7,8,9,10,12,14,16,18,20]
for i in depth:
    clf = tree.DecisionTreeClassifier(class_weight='balanced',max_depth = i, min_samples_split=100)
    clf.fit(X_tr, y_train)
    y_train_pred = clf.predict_proba( X_tr)[:,1]
    y_cv_pred = clf.predict_proba( X_cv)[:,1]
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(depth, train_auc, label='Train AUC')
plt.plot(depth, cv_auc, label='CV AUC')
plt.scatter(depth, train_auc, label='Train AUC points')
plt.scatter(depth, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("Hyper-parameter(Alpha's)")
plt.ylabel("AUC")
plt.title("CROSS VALIDATION ERROR PLOTS")
plt.grid()
plt.show()
```



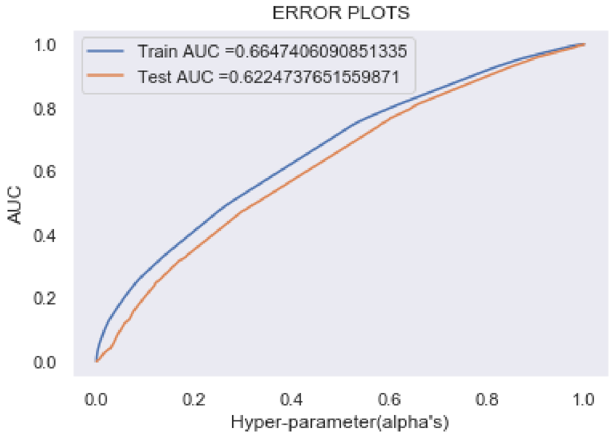


In [110]:

```
i=8
```

In [111]:

```
#https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.r
rve
from sklearn.metrics import roc_curve, auc
from sklearn import tree

clf = tree.DecisionTreeClassifier(class_weight='balanced',max_depth = i, min_samples_split=250)
clf.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = clf.predict_proba(X_tr)[:, 1]
y_test_pred = clf.predict_proba(X_te)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Hyper-parameter(alpha's)")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix for Train and Test Data

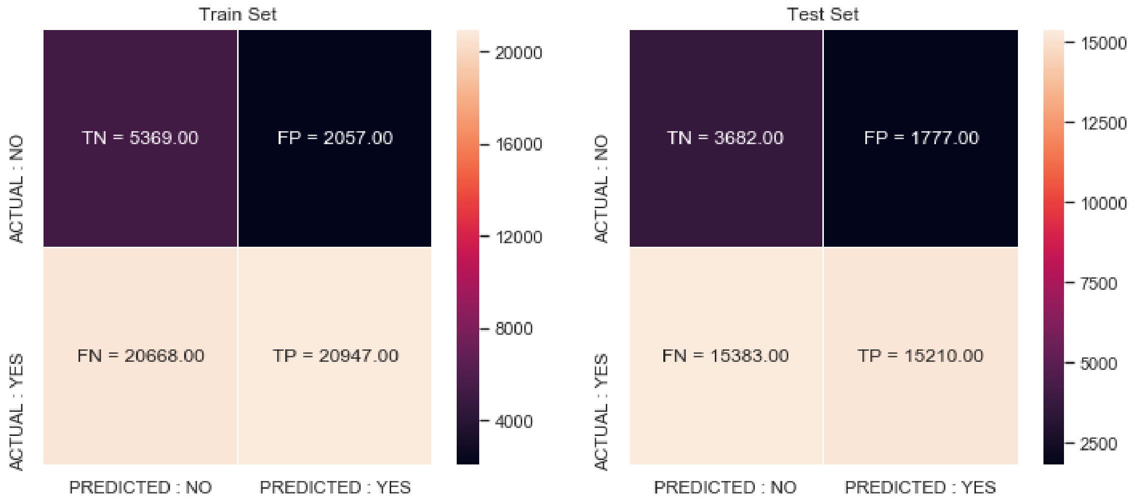
In [107]:

```
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    global predictions1
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1 = predictions
    return predictions
    return predictions
```

In [108]:

```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()

con_m_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
train_tpr))
con_m_test = confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten()
, con_m_train.flatten())])).reshape(2,2)
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(),
con_m_test.flatten())])).reshape(2,2)

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_train, fmt = '', ax=ax[0])
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = '', ax=ax[1])

ax[0].set_title('Train Set')
ax[1].set_title('Test Set')

plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.36392374484031625 for threshold 0.515
the maximum value of tpr*(1-fpr) 0.3353341913363285 for threshold 0.515
```



2.4.2 Applying Random Forests on TFIDF, SET 2

In [159]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr_tfidf = hstack((train_categories_responseCoding, train_subcategories_responseCoding,
train_state_categories_responseCoding, train_grade_categories_responseCoding,
train_teacher_prefix_responseCoding, price_train, quantity_train, prev_projects_train,
title_word_count_train, essay_word_count_train, title_tfidf_train, text_tfidf_train)).tocsr()
X_te_tfidf = hstack((test_categories_feature_responseCoding,
test_subcategories_feature_responseCoding, test_state_categories_feature_responseCoding,
test_grade_categories_feature_responseCoding, test_teacher_prefix_feature_responseCoding,
price_test, quantity_test, prev_projects_test, title_word_count_test, essay_word_count_test, title_
tfidf_test,text_tfidf_test)).tocsr()
X_cv_tfidf = hstack((cv_categories_feature_responseCoding, cv_subcategories_feature_responseCoding
, cv_state_categories_feature_responseCoding, cv_grade_categories_feature_responseCoding,
cv_teacher_prefix_feature_responseCoding, price_cv, quantity_cv, prev_projects_cv,
title_word_count_cv, essay_word_count_cv,  title_tfidf_cv, text_tfidf_cv)).tocsr()
```

In [160]:

```
print("Final Data matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_cv_tfidf.shape, y_cv.shape)
print(X_te_tfidf.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 10150) (22445,)
(11055, 10150) (11055,)
(16500, 10150) (16500,)
====================================================================================================
```

Hyperparameter Tunning GridSearch

In [120]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
import seaborn as sea
```

In [121]:

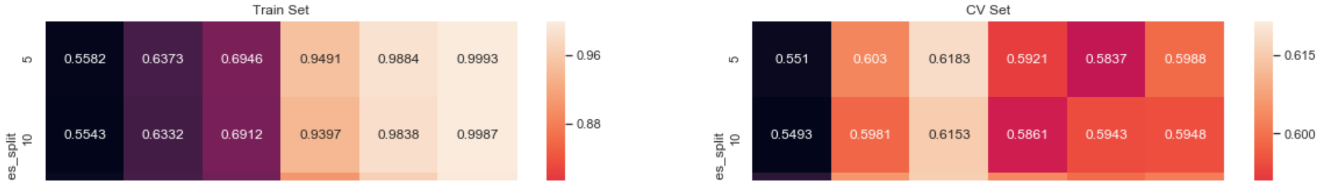
```
RF = RandomForestClassifier()

parameters = {'max_depth': [1, 5, 10, 50, 100, 500], 'min_samples_split': [5, 10, 100,250, 500]}

classifier = GridSearchCV(RF, parameters, cv=10, scoring='roc_auc')
classifier.fit(X_tr_tfidf, y_train)
```

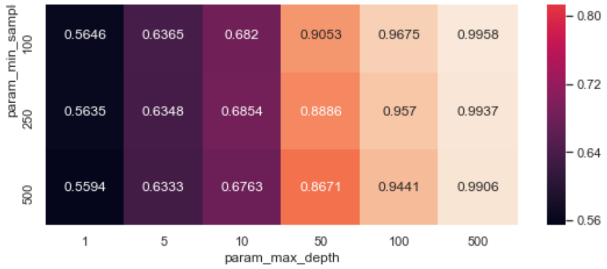
Out[121]:

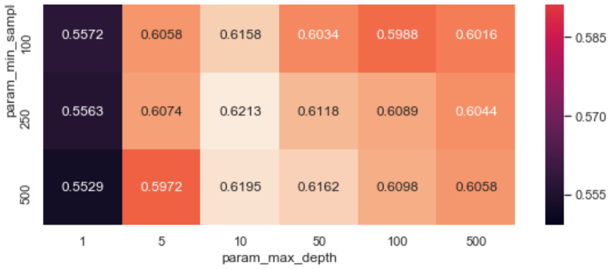
```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'max_depth': [1, 5, 10, 50, 100, 500], 'min_samples_split': [5, 10, 100, 250, 5
00]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [122]:

```
max_scores = pd.DataFrame(classifier.cv_results_).groupby(['param_min_samples_split',
'param_max_depth']).max().unstack()[['mean_test_score', 'mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(20,6))

sea.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sea.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])

ax[0].set_title('Train Set')
ax[1].set_title('CV Set')

plt.show()
```



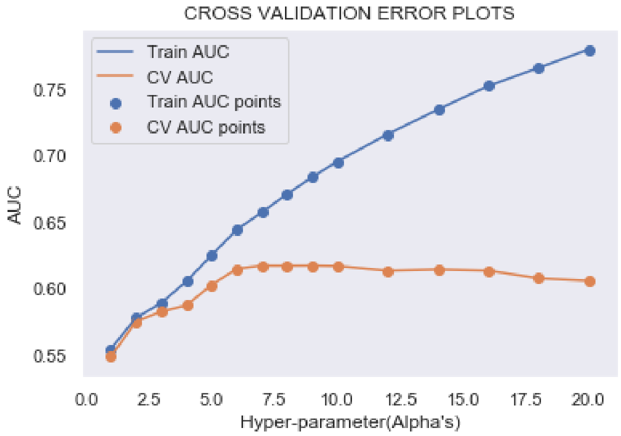




In [ ]:

In [123]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
import numpy as np
from sklearn import tree
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.
y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.
"""
train_auc = []
cv_auc = []
depth = [1, 2, 3, 4, 5, 6, 7,8,9,10,12,14,16,18,20]
for i in depth:
    clf = tree.DecisionTreeClassifier(class_weight='balanced',max_depth = i, min_samples_split=250)
    clf.fit(X_tr_tfidf, y_train)
    y_train_pred = clf.predict_proba( X_tr_tfidf)[:,1]
    y_cv_pred = clf.predict_proba( X_cv_tfidf)[:,1]
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(depth, train_auc, label='Train AUC')
plt.plot(depth, cv_auc, label='CV AUC')
plt.scatter(depth, train_auc, label='Train AUC points')
plt.scatter(depth, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("Hyper-parameter(Alpha's)")
plt.ylabel("AUC")
plt.title("CROSS VALIDATION ERROR PLOTS")
plt.grid()
plt.show()
```

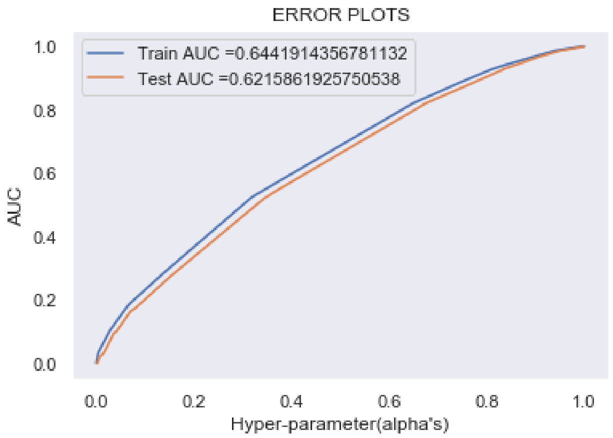


In [124]:

```
i=6
```

In [125]:

```
#https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.ro
rve
from sklearn.metrics import roc_curve, auc
from sklearn import tree

clf = tree.DecisionTreeClassifier(class_weight='balanced',max_depth = i, min_samples_split=250)
clf.fit(X_tr_tfidf, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = clf.predict_proba(X_tr_tfidf)[:, 1]
y_test_pred = clf.predict_proba(X_te_tfidf)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Hyper-parameter(alpha's)")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix for Train and Test Data

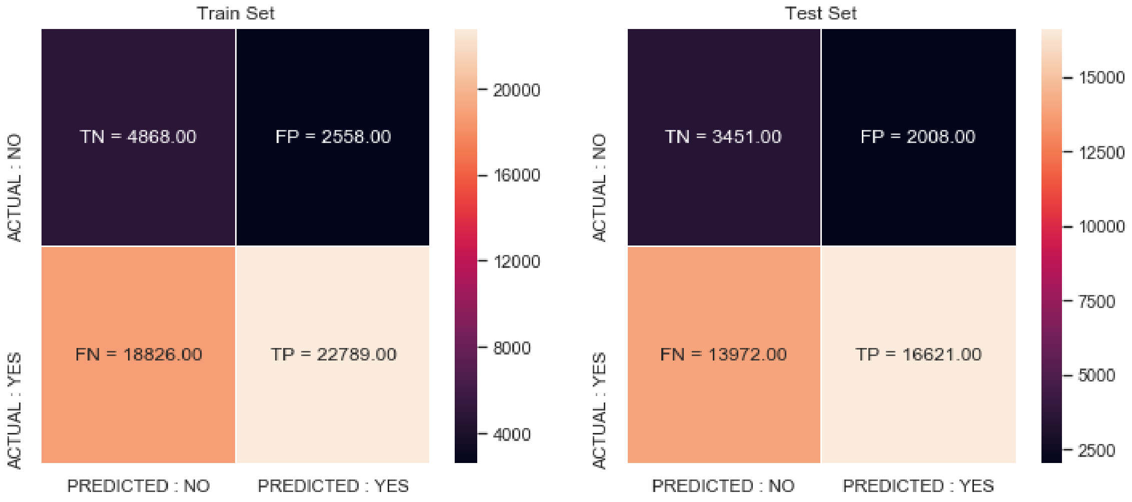
In [126]:

```
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    global predictions1
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1 = predictions
    return predictions
```

In [127]:

```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()

con_m_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
train_tpr))
con_m_test = confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr))
con_m_test = confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten()
, con_m_train.flatten())])).reshape(2,2)
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(),
con_m_test.flatten())])).reshape(2,2)

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_train, fmt = '', ax=ax[0])
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = '', ax=ax[1])

ax[0].set_title('Train Set')
ax[1].set_title('Test Set')

plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.35898061239351825 for threshold 0.476
the maximum value of tpr*(1-fpr) 0.3434527102118066 for threshold 0.476
```



2.4.3 Applying Random Forests on AVG W2V, SET 3

In [66]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
import numpy as np

X_tr_avg_w2v = np.hstack((train_categories_responseCoding, train_subcategories_responseCoding,
train_state_categories_responseCoding, train_grade_categories_responseCoding,
train_teacher_prefix_responseCoding, avg_w2v_vectors_titles_train, avg_w2v_vectors_train))
X_te_avg_w2v = np.hstack((test_categories_feature_responseCoding,
test_subcategories_feature_responseCoding, test_state_categories_feature_responseCoding,
test_grade_categories_feature_responseCoding, test_teacher_prefix_feature_responseCoding,
avg_w2v_vectors_titles_test, avg_w2v_vectors_test))
X_cv_avg_w2v = np.hstack((cv_categories_feature_responseCoding,
cv_subcategories_feature_responseCoding, cv_state_categories_feature_responseCoding,
cv_grade_categories_feature_responseCoding, cv_teacher_prefix_feature_responseCoding,
avg_w2v_vectors_titles_cv, avg_w2v_vectors_cv))
```

In [67]:

```
print("Final Data matrix")
print(X_tr_avg_w2v.shape, y_train.shape)
print(X_cv_avg_w2v.shape, y_cv.shape)
print(X_te_avg_w2v.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
```

```
(22445, 610) (22445,)
(11055, 610) (11055,)
(16500, 610) (16500,)
==================================================================================================
```

Hyperparameter Tunning GridSearch

In [71]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
import seaborn as sea
```

In [72]:

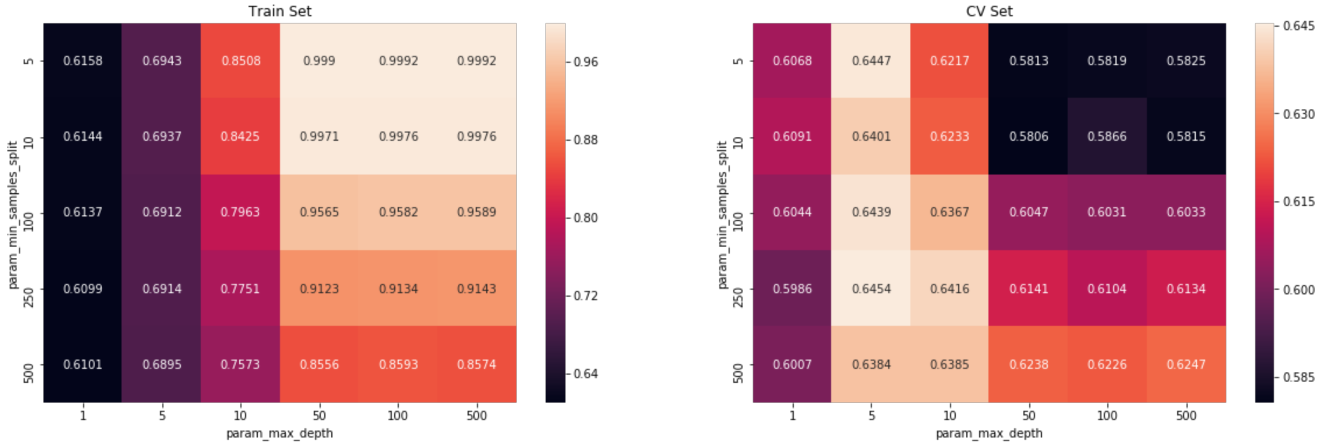
```
RF = RandomForestClassifier()

parameters = {'max_depth': [1, 5, 10, 50, 100, 500], 'min_samples_split': [5, 10, 100,250, 500]}

classifier = GridSearchCV(RF, parameters, cv=10, scoring='roc_auc')
classifier.fit(X_tr_avg_w2v, y_train)
```

Out[72]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'max_depth': [1, 5, 10, 50, 100, 500], 'min_samples_split': [5, 10, 100, 250, 5
00]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

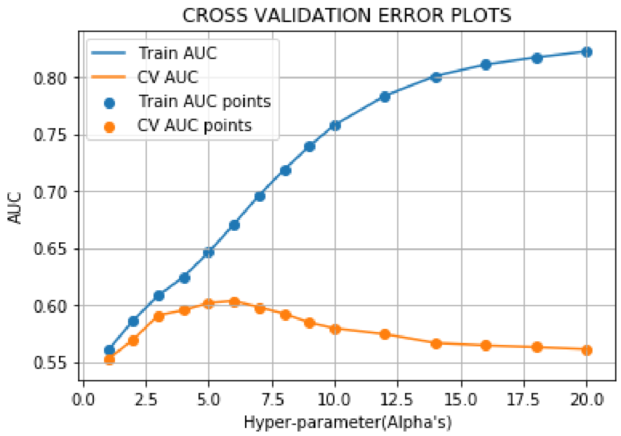
In [73]:

```
max_scores = pd.DataFrame(classifier.cv_results_).groupby(['param_min_samples_split',
'param_max_depth']).max().unstack()[['mean_test_score', 'mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(20,6))

sea.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sea.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])

ax[0].set_title('Train Set')
ax[1].set_title('CV Set')

plt.show()
```



In [74]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
import numpy as np
from sklearn import tree
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.
y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.
"""
train_auc = []
cv_auc = []
depth = [1, 2, 3, 4, 5, 6, 7,8,9,10,12,14,16,18,20]
for i in depth:
    clf = tree.DecisionTreeClassifier(class_weight='balanced',max_depth = i, min_samples_split=250)
    clf.fit(X_tr_avg_w2v, y_train)
    y_train_pred = clf.predict_proba( X_tr_avg_w2v)[:,1]
    y_cv_pred = clf.predict_proba( X_cv_avg_w2v)[:,1]
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(depth, train_auc, label='Train AUC')
plt.plot(depth, cv_auc, label='CV AUC')
plt.scatter(depth, train_auc, label='Train AUC points')
plt.scatter(depth, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("Hyper-parameter(Alpha's)")
plt.ylabel("AUC")
plt.title("CROSS VALIDATION ERROR PLOTS")
plt.grid()
plt.show()
```



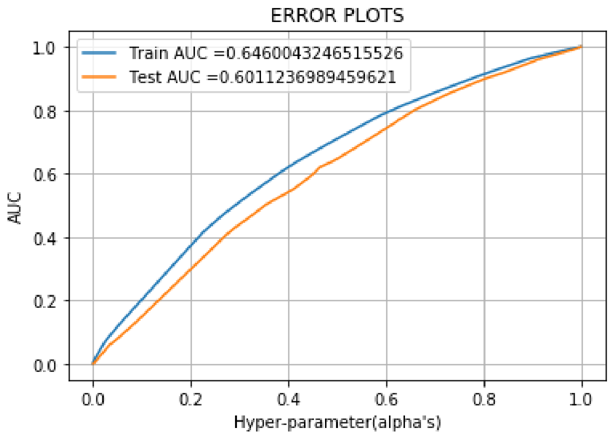
In [79]:

```
i=5
```

In [80]:

```
#https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.ro
rve
from sklearn.metrics import roc_curve, auc
from sklearn import tree

clf = tree.DecisionTreeClassifier(class_weight='balanced',max_depth = i, min_samples_split=250)
clf.fit(X_tr_avg_w2v, y_train)

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

```
y_train_pred = clf.predict_proba(X_tr_avg_w2v)[:, 1]
y_test_pred = clf.predict_proba(X_te_avg_w2v)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Hyper-parameter(alpha's)")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix for Train and Test Data

In [81]:

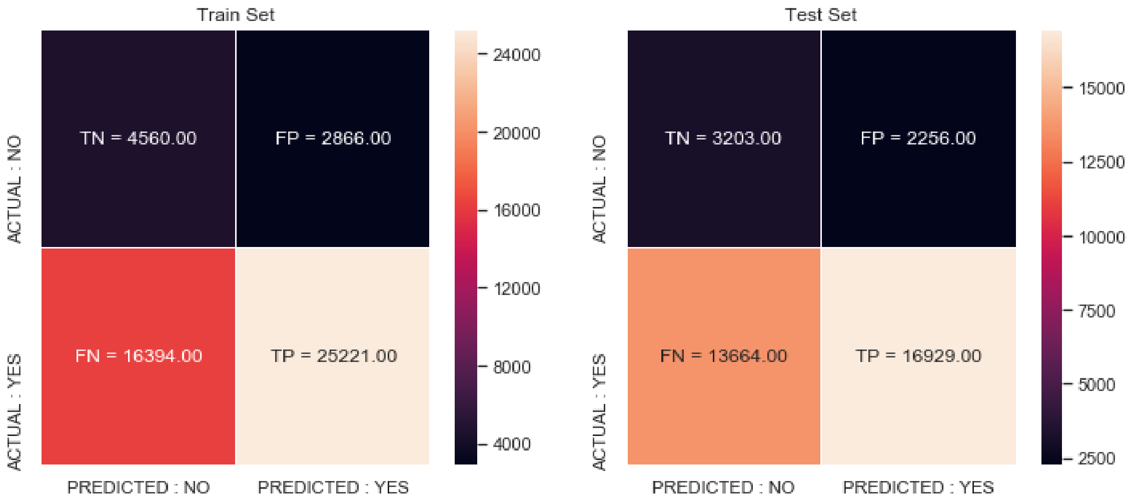
```
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    global predictions1
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1 = predictions
    return predictions
```

In [82]:

```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()

con_m_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
train_tpr))
con_m_test = confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten()
, con_m_train.flatten())])).reshape(2,2)
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(),
con_m_test.flatten())])).reshape(2,2)

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_train, fmt = '', ax=ax[0])
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = '', ax=ax[1])

ax[0].set_title('Train Set')
```

```
ax[1].set_title('Test Set')

plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.3721536655358382 for threshold 0.524
the maximum value of tpr*(1-fpr) 0.33183395873855415 for threshold 0.525
```



2.4.4 Applying Random Forests on TFIDF W2V, SET 4

In [81]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
import numpy as np

X_tr_tfidf_w2v = np.hstack((train_categories_responseCoding, train_subcategories_responseCoding,
train_state_categories_responseCoding, train_grade_categories_responseCoding,
train_teacher_prefix_responseCoding, tfidf_w2v_vectors_train, tfidf_w2v_vectors_titles_train))
X_te_tfidf_w2v = np.hstack((test_categories_feature_responseCoding,
test_subcategories_feature_responseCoding, test_state_categories_feature_responseCoding,
test_grade_categories_feature_responseCoding, test_teacher_prefix_feature_responseCoding,
tfidf_w2v_vectors_test, tfidf_w2v_vectors_titles_test))
X_cv_tfidf_w2v = np.hstack((cv_categories_feature_responseCoding,
cv_subcategories_feature_responseCoding, cv_state_categories_feature_responseCoding,
cv_grade_categories_feature_responseCoding, cv_teacher_prefix_feature_responseCoding,
tfidf_w2v_vectors_cv, tfidf_w2v_vectors_titles_cv))
```

In [82]:

```
print("Final Data matrix")
print(X_tr_tfidf_w2v.shape, y_train.shape)
print(X_cv_tfidf_w2v.shape, y_cv.shape)
print(X_te_tfidf_w2v.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 610) (22445,)
(11055, 610) (11055,)
(16500, 610) (16500,)
====================================================================================================
```

In [94]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
import seaborn as sea
```

In [95]:

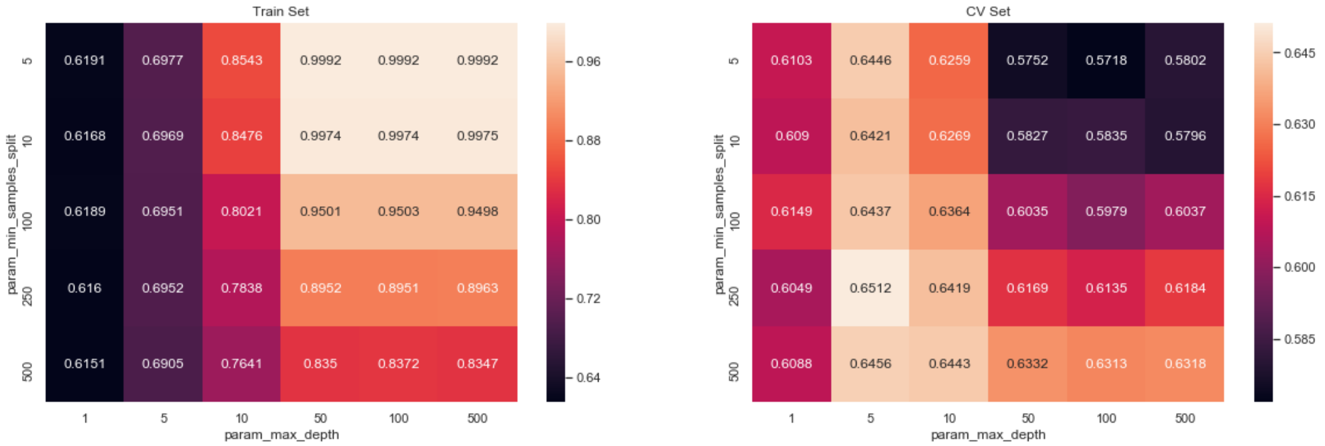
```
RF = RandomForestClassifier()

parameters = {'max_depth': [1, 5, 10, 50, 100, 500], 'min_samples_split': [5, 10, 100,250, 500]}

classifier = GridSearchCV(RF, parameters, cv=10, scoring='roc_auc')
classifier.fit(X_tr_tfidf_w2v, y_train)
```

Out[95]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'max_depth': [1, 5, 10, 50, 100, 500], 'min_samples_split': [5, 10, 100, 250, 5
00]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [96]:

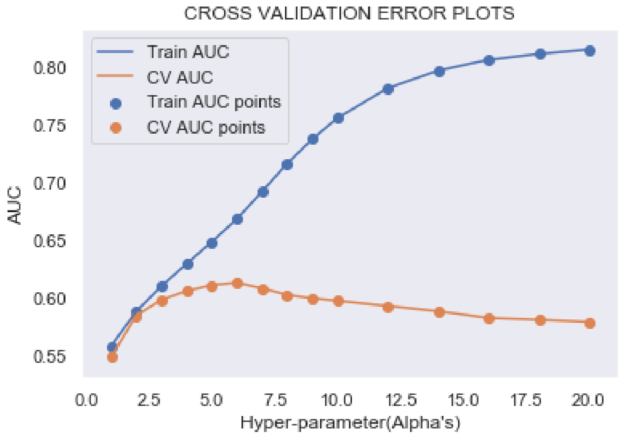
```
max_scores = pd.DataFrame(classifier.cv_results_).groupby(['param_min_samples_split',
'param_max_depth']).max().unstack()[['mean_test_score', 'mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(20,6))

sea.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sea.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])

ax[0].set_title('Train Set')
ax[1].set_title('CV Set')

plt.show()
```



In [97]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
import numpy as np
from sklearn import tree
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.
y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.
"""
train_auc = []
cv_auc = []
depth = [1, 2, 3, 4, 5, 6, 7,8,9,10,12,14,16,18,20]
```

```
for i in depth:
    clf = tree.DecisionTreeClassifier(class_weight='balanced',max_depth = i, min_samples_split=250)
    clf.fit(X_tr_tfidf_w2v, y_train)
    y_train_pred = clf.predict_proba( X_tr_tfidf_w2v)[:,1]
    y_cv_pred = clf.predict_proba( X_cv_tfidf_w2v)[:,1]
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(depth, train_auc, label='Train AUC')
plt.plot(depth, cv_auc, label='CV AUC')
plt.scatter(depth, train_auc, label='Train AUC points')
plt.scatter(depth, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("Hyper-parameter(Alpha's)")
plt.ylabel("AUC")
plt.title("CROSS VALIDATION ERROR PLOTS")
plt.grid()
plt.show()
```

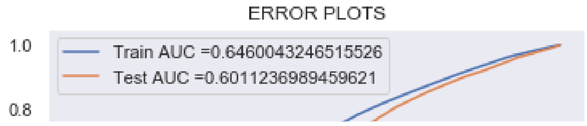


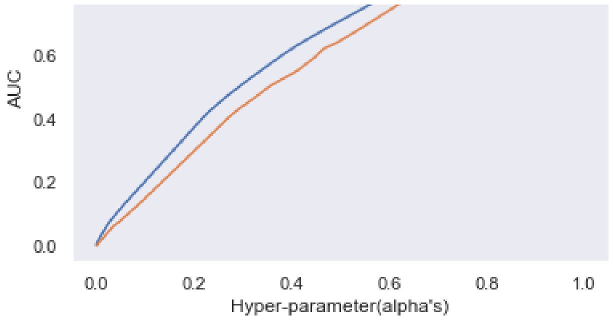
In [101]:

```
i=5
```

In [102]:

```
#https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.rc
rve
from sklearn.metrics import roc_curve, auc
from sklearn import tree

clf = tree.DecisionTreeClassifier(class_weight='balanced',max_depth = i, min_samples_split=250)
clf.fit(X_tr_tfidf_w2v, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = clf.predict_proba(X_tr_tfidf_w2v)[:, 1]
y_test_pred = clf.predict_proba(X_te_tfidf_w2v)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Hyper-parameter(alpha's)")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```





Confusion Matrix for Train and Test Data

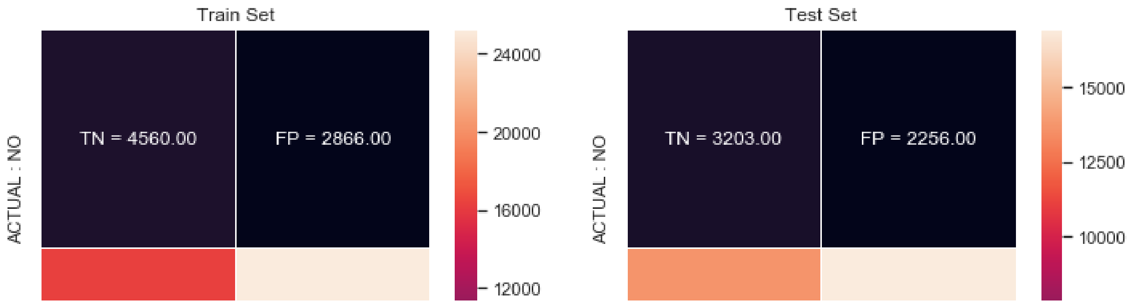
In [103]:

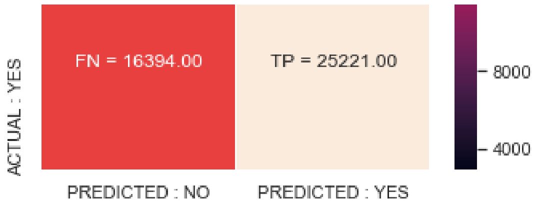
```
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    global predictions1
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1 = predictions
    return predictions
```

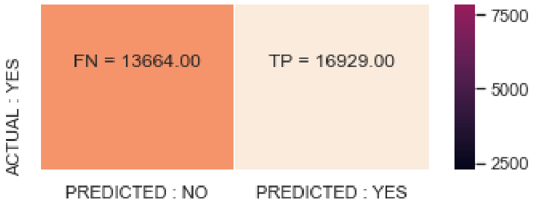
In [104]:

```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()

con_m_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
train_tpr))
con_m_test = confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten()
, con_m_train.flatten())])).reshape(2,2)
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(),
con_m_test.flatten())])).reshape(2,2)

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_train, fmt = '', ax=ax[0])
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = '', ax=ax[1])

ax[0].set_title('Train Set')
ax[1].set_title('Test Set')

plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.3721536655358382 for threshold 0.524
the maximum value of tpr*(1-fpr) 0.33183395873855415 for threshold 0.525
```



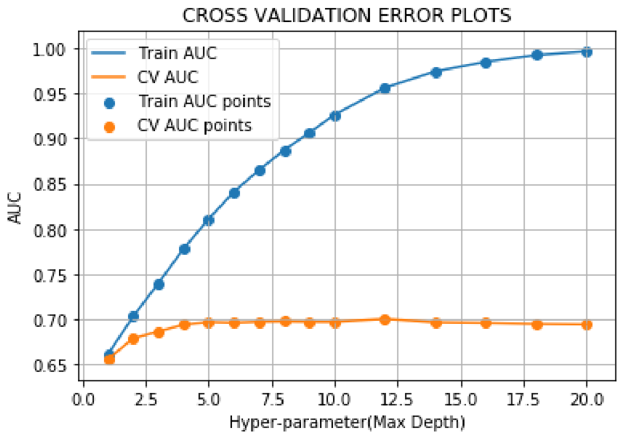




2.5.2.1 Applying GBDT on Set - 1

In [58]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
import numpy as np
from sklearn import tree
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.
y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.
"""
train_auc = []
cv_auc = []
depth = [1, 2, 3, 4, 5, 6, 7,8,9,10,]
for i in depth:
    clf = GradientBoostingClassifier(max_depth = i, min_samples_split=250)
    clf.fit(X_tr, y_train)
    y_train_pred = clf.predict_proba( X_tr)[:,1]
    y_cv_pred = clf.predict_proba( X_cv)[:,1]
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(depth, train_auc, label='Train AUC')
plt.plot(depth, cv_auc, label='CV AUC')
plt.scatter(depth, train_auc, label='Train AUC points')
plt.scatter(depth, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("Hyper-parameter(Max Depth)")
plt.ylabel("AUC")
plt.title("CROSS VALIDATION ERROR PLOTS")
plt.grid()
plt.show()
```

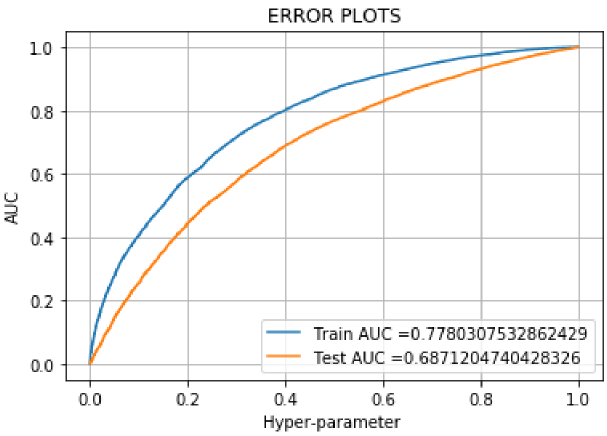


In [59]:

```
i=4
```

In [60]:

```
#https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.ro
rve
from sklearn.metrics import roc_curve, auc
from sklearn import tree
from sklearn.ensemble import GradientBoostingClassifier


clf = GradientBoostingClassifier(max_depth = i, min_samples_split=250)
clf.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = clf.predict_proba(X_tr)[:, 1]
y_test_pred = clf.predict_proba(X_te)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Hyper-parameter ")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix

In [61]:

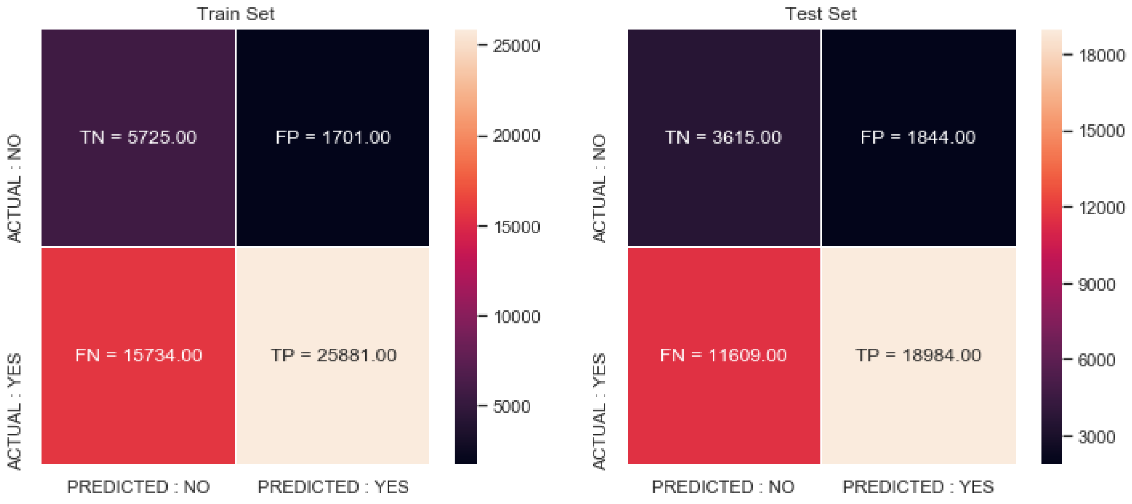
```
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    global predictions1
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1 = predictions
    return predictions
```

In [62]:

```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()

con_m_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
train_tpr))
con_m_test = confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr))
```

```
key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten()
, con_m_train.flatten())])).reshape(2,2)
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(),
con_m_test.flatten())])).reshape(2,2)

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_train, fmt = '', ax=ax[0])
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = '', ax=ax[1])

ax[0].set_title('Train Set')
ax[1].set_title('Test Set')

plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.5019841894549835 for threshold 0.849
the maximum value of tpr*(1-fpr) 0.414720355717386 for threshold 0.846
```

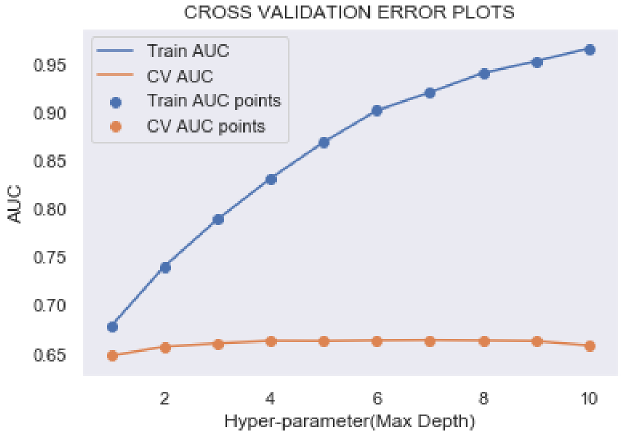


2.5.2.2 Applying GBDT on Set - 2

In [162]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
import numpy as np
from sklearn import tree
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.
y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.
"""
train_auc = []
cv_auc = []
depth = [1, 2, 3, 4, 5, 6, 7,8,9,10,]
for i in depth:
    clf = GradientBoostingClassifier(max_depth = i, min_samples_split=150)
    clf.fit(X_tr_tfidf, y_train)
    y_train_pred = clf.predict_proba( X_tr_tfidf)[:,1]
    y_cv_pred = clf.predict_proba( X_cv_tfidf)[:,1]
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
plt.plot(depth, train_auc, label='Train AUC')
plt.plot(depth, cv_auc, label='CV AUC')
plt.scatter(depth, train_auc, label='Train AUC points')
plt.scatter(depth, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("Hyper-parameter(Max Depth)")
plt.ylabel("AUC")
plt.title("CROSS VALIDATION ERROR PLOTS")
plt.grid()
plt.show()
```

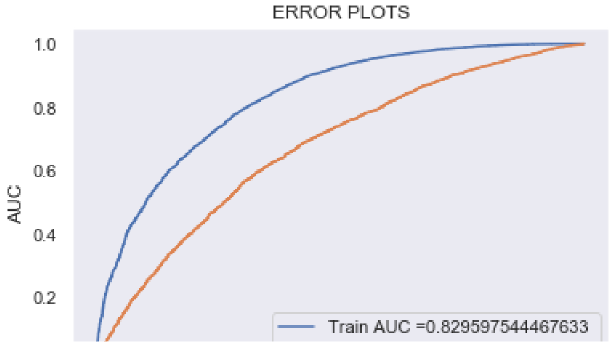


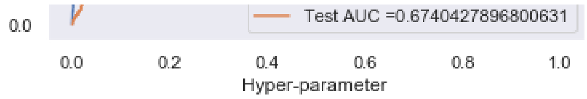
In [163]:

```
i=4
```

In [164]:

```
#https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.r
rve
from sklearn.metrics import roc_curve, auc
from sklearn import tree
from sklearn.ensemble import GradientBoostingClassifier


clf = GradientBoostingClassifier(max_depth = i, min_samples_split=250)
clf.fit(X_tr_tfidf, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = clf.predict_proba(X_tr_tfidf)[:, 1]
y_test_pred = clf.predict_proba(X_te_tfidf)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Hyper-parameter ")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```





Confusion Matrix

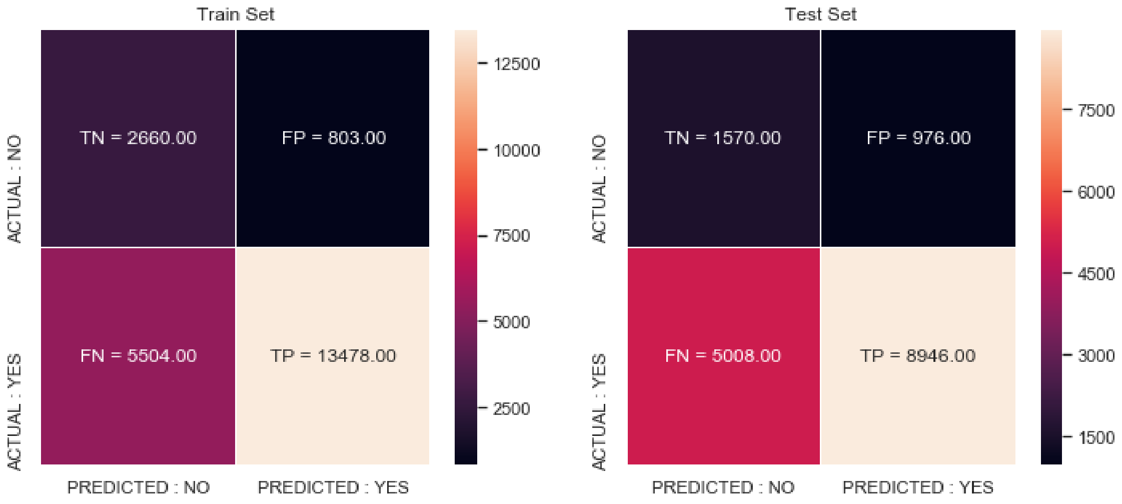
In [165]:

```
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    global predictions1
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1 = predictions
    return predictions
```

In [166]:

```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()

con_m_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
train_tpr))
con_m_test = confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten()
, con_m_train.flatten())])).reshape(2,2)
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(),
con_m_test.flatten())])).reshape(2,2)

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_train, fmt = '', ax=ax[0])
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = '', ax=ax[1])

ax[0].set_title('Train Set')
ax[1].set_title('Test Set')

plt.show()
```

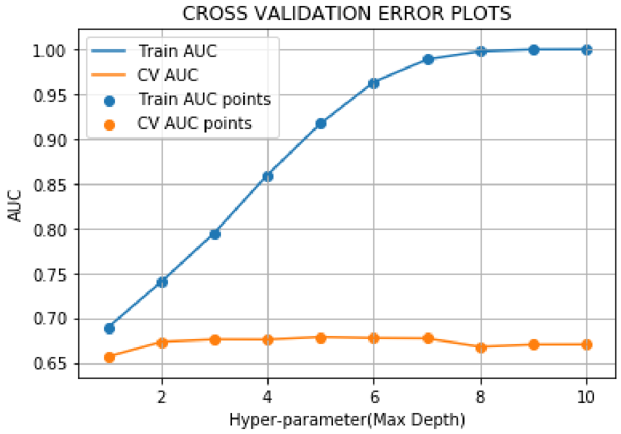
```
the maximum value of tpr*(1-fpr) 0.5576215295594565 for threshold 0.845
the maximum value of tpr*(1-fpr) 0.40025970192038235 for threshold 0.85
```



2.5.2.3 Applying GBDT on Set - 3

In [68]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
import numpy as np
from sklearn import tree
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.
y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.
"""
train_auc = []
cv_auc = []
depth = [1, 2, 3, 4, 5, 6, 7,8,9,10,]
for i in depth:
    clf = GradientBoostingClassifier(max_depth = i, min_samples_split=50)
    clf.fit(X_tr_avg_w2v, y_train)
    y_train_pred = clf.predict_proba( X_tr_avg_w2v)[:,1]
    y_cv_pred = clf.predict_proba( X_cv_avg_w2v)[:,1]
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(depth, train_auc, label='Train AUC')
plt.plot(depth, cv_auc, label='CV AUC')
plt.scatter(depth, train_auc, label='Train AUC points')
plt.scatter(depth, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("Hyper-parameter(Max Depth)")
plt.ylabel("AUC")
plt.title("CROSS VALIDATION ERROR PLOTS")
plt.grid()
plt.show()
```



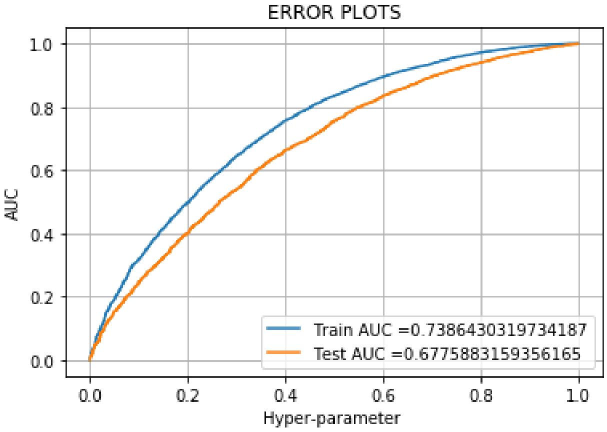
In [69]:

```
i=2
```

In [70]:

```
#https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.ro
rve
from sklearn.metrics import roc_curve, auc
from sklearn import tree
from sklearn.ensemble import GradientBoostingClassifier
```

```
clf = GradientBoostingClassifier(max_depth = i, min_samples_split=150)
clf.fit(X_tr_avg_w2v, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = clf.predict_proba(X_tr_avg_w2v)[:, 1]
y_test_pred = clf.predict_proba(X_te_avg_w2v)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Hyper-parameter ")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [71]:

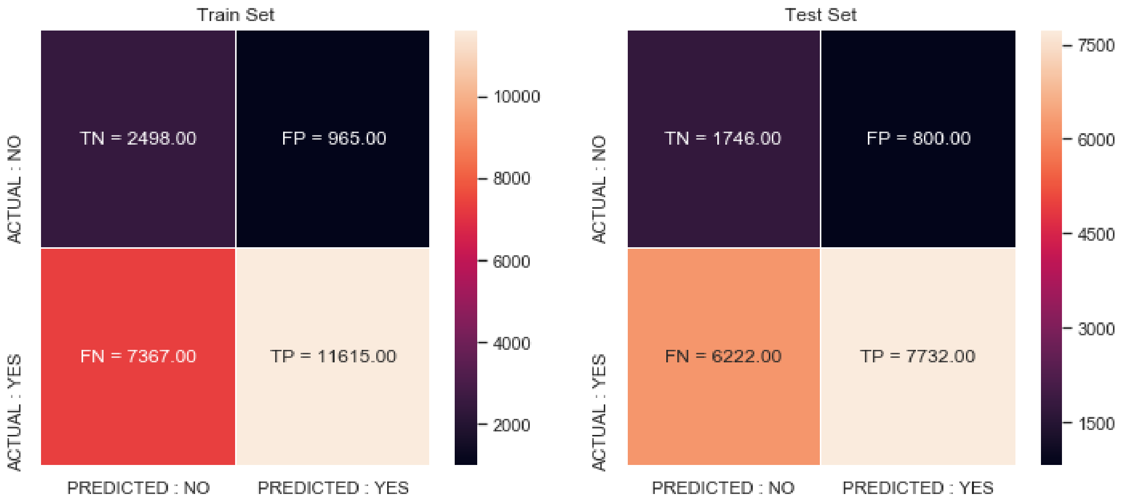
```
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    global predictions1
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1 = predictions
    return predictions
```

In [72]:

```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()

con_m_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
train_tpr))
con_m_test = confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten()
, con_m_train.flatten())])).reshape(2,2)
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(),
con_m_test.flatten())])).reshape(2,2)

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_train, fmt = '', ax=ax[0])
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = '', ax=ax[1])
```

```
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = '', ax=ax[1])

ax[0].set_title('Train Set')
ax[1].set_title('Test Set')

plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.45719395607790875 for threshold 0.853
the maximum value of tpr*(1-fpr) 0.39871802435586523 for threshold 0.859
```

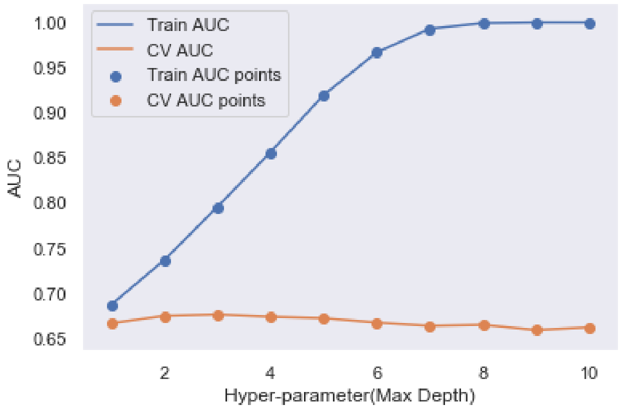


2.5.2.4 Applying GBDT on Set - 4

In [83]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
import numpy as np
from sklearn import tree
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.
y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.
"""
train_auc = []
cv_auc = []
depth = [1, 2, 3, 4, 5, 6, 7,8,9,10,]
for i in depth:
    clf = GradientBoostingClassifier(max_depth = i, min_samples_split=50)
    clf.fit(X_tr_tfidf_w2v, y_train)
    y_train_pred = clf.predict_proba( X_tr_tfidf_w2v)[:,1]
    y_cv_pred = clf.predict_proba( X_cv_tfidf_w2v)[:,1]
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(depth, train_auc, label='Train AUC')
plt.plot(depth, cv_auc, label='CV AUC')
plt.scatter(depth, train_auc, label='Train AUC points')
plt.scatter(depth, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("Hyper-parameter(Max Depth)")
plt.ylabel("AUC")
plt.title("CROSS VALIDATION ERROR PLOTS")
plt.grid()
plt.show()
```

CROSS VALIDATION ERROR PLOTS

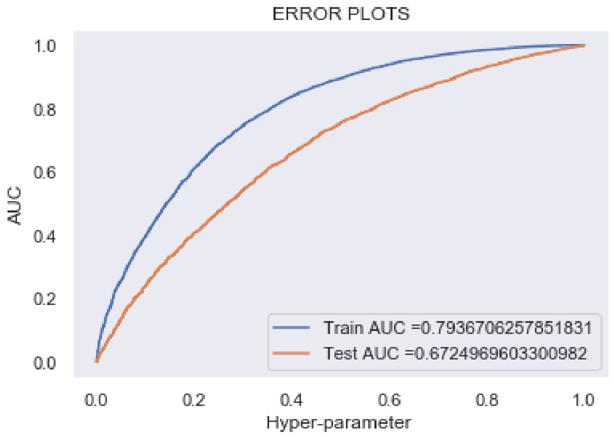


In [84]:

```
i=3
```

In [85]:

```
#https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.r
rve
from sklearn.metrics import roc_curve, auc
from sklearn import tree
from sklearn.ensemble import GradientBoostingClassifier


clf = GradientBoostingClassifier(max_depth = i, min_samples_split=150)
clf.fit(X_tr_tfidf_w2v, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred = clf.predict_proba(X_tr_tfidf_w2v)[:, 1]
y_test_pred = clf.predict_proba(X_te_tfidf_w2v)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Hyper-parameter ")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix

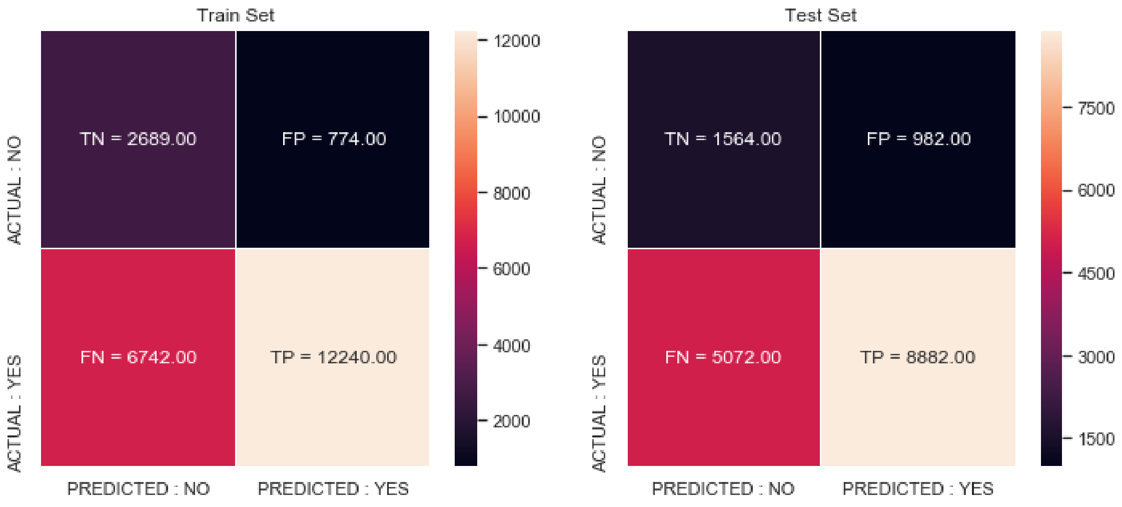
In [86]:

```
def predict(proba, threshould, fpr, tpr):
    t = threshould[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    global predictions1
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1 = predictions
    return predictions
```

In [87]:

```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()

con_m_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr,
train_tpr))
con_m_test = confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten()
, con_m_train.flatten())])).reshape(2,2)
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(),
con_m_test.flatten())])).reshape(2,2)

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_train, fmt = '', ax=ax[0])
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = '', ax=ax[1])

ax[0].set_title('Train Set')
ax[1].set_title('Test Set')

plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.5237033379008877 for threshold 0.853
the maximum value of tpr*(1-fpr) 0.39662583411480723 for threshold 0.848
```



3. Conclusion

In [92]:

```
# Please compare all your models using Prettytable library
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
TB = PrettyTable()
```

```
TB.field_names = ["Random Forest - MODEL", "HyperparameterS",  "Test_Auc"]
TB.title = "Decision Tree"
TB.add_row(["BOW-ENC-RF", "Depth:8   | Samp_Split:250",0.623])
TB.add_row(["TFIDF-ENC-RF", "Depth:6   | Samp_Split:250", 0.622])
TB.add_row(["AvgW2V-ENC-RF", "Depth:5   | Samp_Split:250", 0.6011])
TB.add_row(["Tf-Idf-ENC-RF", "Depth:5   | Samp_Split:250",  0.6012])
print(TB)

TB1 = PrettyTable()

TB1.field_names = ["GBDT - MODEL", "HyperparameterS", "Test_Auc"]
TB1.title = "Gradient Boosting Decision Tree"
TB1.add_row(["BOW-ENC-GBDT", "Depth:4   | Samp_Split:250", 0.697])
TB1.add_row(["TFIDF-ENC-GBDT", "Depth:4   | Samp_Split:250", 0.674])
TB1.add_row(["AvgW2V-ENC-GBDT", "Depth:2   | Samp_Split:150",  0.678])
TB1.add_row(["Tf-Idf-ENC-GBDT", "Depth:3   | Samp_Split:150", 0.6724])
print(TB1)
```

```
+-----------------------+----------------------------+----------+
| Random Forest - MODEL |      HyperparameterS       | Test_Auc |
+-----------------------+----------------------------+----------+
|       BOW-ENC-RF      | Depth:8   | Samp_Split:250 |  0.623   |
|      TFIDF-ENC-RF     | Depth:6   | Samp_Split:250 |  0.622   |
|     AvgW2V-ENC-RF     | Depth:5   | Samp_Split:250 |  0.6011  |
|     Tf-Idf-ENC-RF     | Depth:5   | Samp_Split:250 |  0.6012  |
+-----------------------+----------------------------+----------+
+-----------------+----------------------------+----------+
|   GBDT - MODEL  |      HyperparameterS       | Test_Auc |
+-----------------+----------------------------+----------+
|   BOW-ENC-GBDT  | Depth:4   | Samp_Split:250 |  0.697   |
|  TFIDF-ENC-GBDT | Depth:4   | Samp_Split:250 |  0.674   |
| AvgW2V-ENC-GBDT | Depth:2   | Samp_Split:150 |  0.678   |
| Tf-Idf-ENC-GBDT | Depth:3   | Samp_Split:150 |  0.6724  |
+-----------------+----------------------------+----------+
```

Observations :

1. **From Above we can say that GBDT performs better compare to Random Forest models. Among them BOW encoded models will perform better, Since BOW is High dimensional compared to others.**
2. **As GridSearch for GBDT is taking so much time I have considered 50K points only.**
3. **For both Random Forest and GBDT, BOW model is performing well.**

In [ ]: