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`

Feature	Description
	• 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*
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_4:__ "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."

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

from google.colab import drive
drive.mount('/content/gdrive')
```

```
/usr/local/lib/python3.6/dist-packages/smart_open/ssh.py:34: UserWarning: paramiko missing, openin
g SSH/SCP/SFTP paths will be disabled.  `pip install paramiko` to suppress
  warnings.warn('paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install
paramiko` to suppress')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/gdrive
```

1.1 Reading Data

In [0]:

```
project_data = pd.read_csv('gdrive/My Drive/train_data.csv')
```

```
resource_data = pd.read_csv('gdrive/My Drive/resources.csv')
```

In [0]:

```
project_data.columns
```

In [0]:

```
project_data[0:2000]['project_is_approved'].value_counts()
```

In [0]:

```
project_data[0:4000]['project_is_approved'].value_counts()
```

In [0]:

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

In [0]:

```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

1.2 Data Analysis

In [0]:

```
# PROVIDE CITATIONS TO YOUR CODE IF YOU TAKE IT FROM ANOTHER WEBSITE.
# https://matplotlib.org/gallery/pie_and_polar_charts/pie_and_donut_labels.html#sphx-glr-gallery-p
ie-and-polar-charts-pie-and-donut-labels-py


y_value_counts = project_data['project_is_approved'].value_counts()
print("Number of projects thar are approved for funding ", y_value_counts[1], ", (",
(y_value_counts[1]/(y_value_counts[1]+y_value_counts[0]))*100,"%)")
print("Number of projects thar are not approved for funding ", y_value_counts[0], ", (",
(y_value_counts[0]/(y_value_counts[1]+y_value_counts[0]))*100,"%)")

fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(aspect="equal"))
recipe = ["Accepted", "Not Accepted"]

data = [y_value_counts[1], y_value_counts[0]]

wedges, texts = ax.pie(data, wedgeprops=dict(width=0.5), startangle=-40)

bbox_props = dict(boxstyle="square,pad=0.3", fc="w", ec="k", lw=0.72)
kw = dict(xycoords='data', textcoords='data', arrowprops=dict(arrowstyle="-"),
          bbox=bbox_props, zorder=0, va="center")

for i, p in enumerate(wedges):
    ang = (p.theta2 - p.theta1)/2. + p.theta1
    y = np.sin(np.deg2rad(ang))
    x = np.cos(np.deg2rad(ang))
    horizontalalignment = {-1: "right", 1: "left"}[int(np.sign(x))]
    connectionstyle = "angle,angleA=0,angleB={}".format(ang)
    kw["arrowprops"].update({"connectionstyle": connectionstyle})
    ax.annotate(recipe[i], xy=(x, y), xytext=(1.35*np.sign(x), 1.4*y),
                horizontalalignment=horizontalalignment, **kw)

ax.set_title("Nmber of projects that are Accepted and not accepted")

plt.show()
```

1.We took the feature "project is approved" which is output for for Project Data and calculated percentages for approved and not approved

approved

2.The above pie chart shows the percentage of projects which are approved in Blue which is 84.85830404217927 % and the percentage of projects which are not approved in Orange which is 15.141695957820739 %.\

3.We calculated approved and not aprooved projects using y_value_counts which gives details of no.of.projects submitted and how many of them are approved and not approved

1.2.1 Univariate Analysis: School State

In [0]:

```
# Pandas dataframe groupby count, mean: https://stackoverflow.com/a/19385591/4084039

temp = pd.DataFrame(project_data.groupby("school_state")
["project_is_approved"].apply(np.mean)).reset_index()
# if you have data which contain only 0 and 1, then the mean = percentage (think about it)
temp.columns = ['state_code', 'num_proposals']

'''# How to plot US state heatmap: https://datascience.stackexchange.com/a/9620

scl = [[0.0, 'rgb(242,240,247)'],[0.2, 'rgb(218,218,235)'],[0.4, 'rgb(188,189,220)'],\
            [0.6, 'rgb(158,154,200)'],[0.8, 'rgb(117,107,177)'],[1.0, 'rgb(84,39,143)']]

data = [ dict(
        type='choropleth',
        colorscale = scl,
        autocolorscale = False,
        locations = temp['state_code'],
        z = temp['num_proposals'].astype(float),
        locationmode = 'USA-states',
        text = temp['state_code'],
        marker = dict(line = dict (color = 'rgb(255,255,255)',width = 2)),
        colorbar = dict(title = "% of pro")
    ) ]

layout = dict(
        title = 'Project Proposals % of Acceptance Rate by US States',
        geo = dict(
            scope='usa',
            projection=dict( type='albers usa' ),
            showlakes = True,
            lakecolor = 'rgb(255, 255, 255)',
        ),
    )

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='us-map-heat-map')
'''
```

In [0]:

```
# https://www.csi.cuny.edu/sites/default/files/pdf/administration/ops/2letterstabbrev.pdf
temp.sort_values(by=['num_proposals'], inplace=True)
print("States with lowest % approvals")
print(temp.head(5))
print('='*50)
print("States with highest % approvals")
print(temp.tail(5))
```

In [0]:

```
#stacked bar plots matplotlib:
https://matplotlib.org/gallery/lines_bars_and_markers/bar_stacked.html
def stack_plot(data, xtick, col2='project_is_approved', col3='total'):
    ind = np.arange(data.shape[0])

    plt.figure(figsize=(20,5))
    p1 = plt.bar(ind, data[col3].values)
    p2 = plt.bar(ind, data[col2].values)

    plt.ylabel('Projects')
    plt.title('Number of projects aproved vs rejected')
    plt.xticks(ind, list(data[xtick].values))
```

```
    plt.xticks(ind, list(data[xtick].values))
    plt.legend((p1[0], p2[0]), ('total', 'accepted'))
    plt.show()
```

In [0]:

```
def univariate_barplots(data, col1, col2='project_is_approved', top=False):
    # Count number of zeros in dataframe python: https://stackoverflow.com/a/51540521/4084039
    temp = pd.DataFrame(project_data.groupby(col1)[col2].agg(lambda x: x.eq(1).sum())).reset_index(
)

    # Pandas dataframe grouby count: https://stackoverflow.com/a/19385591/4084039
    temp['total'] = pd.DataFrame(project_data.groupby(col1)
[col2].agg({'total':'count'})).reset_index()['total']
    temp['Avg'] = pd.DataFrame(project_data.groupby(col1)[col2].agg({'Avg':'mean'})).reset_index()[
'Avg']

    temp.sort_values(by=['total'],inplace=True, ascending=False)

    if top:
        temp = temp[0:top]

    stack_plot(temp, xtick=col1, col2=col2, col3='total')
    print(temp.head(5))
    print("="*50)
    print(temp.tail(5))

univariate_barplots(project_data, 'school_state', 'project_is_approved', False)
```

1.we calculated no.of.projects submitted per each state and calculated how many of them are approved and rejected from that state

2.we made a barplot of how many total projects are submitted from each state and how many are accepted and rejected from that state

3.We calculated acceptance rate which is perentage of projects submitted and accepted and rejected from particular state and we sorted the acceptance rate

1. projects from CA have been accepted and rejected more

SUMMARY: Every state has greater than 80% success rate in approval

1.2.2 Univariate Analysis: teacher_prefix

In [0]:

```
univariate_barplots(project_data, 'teacher_prefix', 'project_is_approved' , top=False)
```

1.based on the prefixe's of the teacher's who submitted projects,we calculated what are the total no.of.projects submitted by a teacher of particular prefix and what is the approval rate that that project is approved.

2.we calculated the approval rate of project acceptance of a teacher of a particular prefix

3.projects submitted by teacher of prefix-MRS are accepted more

4.no.of.projects approved and rejected by teacher of certain specific prefix

1.2.3 Univariate Analysis: project_grade_category

In [0]:

```
univariate_barplots(project_data, 'project_grade_category', 'project_is_approved', top=False)
```

1.we calculated the acceptance rate based on the grades of projects submitted and how many are accepted and rejected from that specific grade

2.projects from Grade preK-2 are submitted more

1.2.4 Univariate Analysis: project_subject_categories

In [0]:

```
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())
```

In [0]:

```
project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)
project_data.head(2)
```

In [0]:

```
univariate_barplots(project_data, 'clean_categories', 'project_is_approved', top=20)
```

1. projects from literacy_language subcategories have been acceptd more
2. we calculated projects submitted of a specific sub_category
3. We calculated what are the total no.of.projets submitted from specific catgory and how many are accepted and rejected from that category

In [0]:

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

In [0]:

```
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))


ind = np.arange(len(sorted_cat_dict))
plt.figure(figsize=(20,5))
p1 = plt.bar(ind, list(sorted_cat_dict.values()))

plt.ylabel('Projects')
plt.title('% of projects aproved category wise')
plt.xticks(ind, list(sorted_cat_dict.keys()))
plt.show()
```

We calculated how many projected are approved from a specific categoryand we do plot how many are rejected and how many are submitted we can say that literacy_language projects are more accepted

In [0]:

```
for i, j in sorted_cat_dict.items():
    print("{:20} :{:10}".format(i,j))
```

1.2.5 Univariate Analysis: project_subject_subcategories

In [0]:

```
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())
```

In [0]:

```
project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)
project_data.head(2)
```

In [0]:

```
univariate_barplots(project_data, 'clean_subcategories', 'project_is_approved', top=50)
```

1. how many projects are submitted of a particular sub_category and how many are approved and rejected.

2.literacy sub_category are submitted more and are accepted more

In [0]:

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

In [0]:

```
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))


ind = np.arange(len(sorted_sub_cat_dict))
plt.figure(figsize=(20,5))
p1 = plt.bar(ind, list(sorted_sub_cat_dict.values()))

plt.ylabel('Projects')
plt.title('% of projects aproved state wise')
```

```
plt.xticks(ind, list(sorted_sub_cat_dict.keys()))
plt.show()
```

1. total no.of.projects approved for a specific sub_category
2. Literacy subcategory projects are accepted more

In [0]:

```
for i, j in sorted_sub_cat_dict.items():
    print("{:20} :{:10}".format(i,j))
```

1.2.6 Univariate Analysis: Text features (Title)

In [0]:

```
#How to calculate number of words in a string in DataFrame: https://stackoverflow.com/a/37483537/4084039
word_count = project_data['project_title'].str.split().apply(len).value_counts()
word_dict = dict(word_count)

word_dict = dict(sorted(word_dict.items(), key=lambda kv: kv[1]))


ind = np.arange(len(word_dict))
plt.figure(figsize=(20,5))
p1 = plt.bar(ind, list(word_dict.values()))

plt.ylabel('Numeber of projects')
plt.xlabel('Numeber words in project title')
plt.title('Words for each title of the project')
plt.xticks(ind, list(word_dict.keys()))
plt.show()
```

1.No.of.words in the project title and are the projects accepted based oon no.of.words in project title

2project title with more no.of.words are accepted more

In [0]:

```
approved_title_word_count = project_data[project_data['project_is_approved']==1]['project_title'].str.split().apply(len)
approved_title_word_count = approved_title_word_count.values

rejected_title_word_count = project_data[project_data['project_is_approved']==0]['project_title'].str.split().apply(len)
rejected_title_word_count = rejected_title_word_count.values
```

In [0]:

```
word_count_titles = project_data['project_title'].str.split().apply(len)
word_count_titles = word_count_titles.values
print(word_count_titles)
```

In [0]:

```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
plt.boxplot([approved_title_word_count, rejected_title_word_count])
plt.xticks([1,2],('Approved Projects','Rejected Projects'))
plt.ylabel('Words in project title')
plt.grid()
plt.show()
```

We created a box plot for numerical data fo approved projects and rejected projects under project_title category

we found few outliers for approved projects and rejected projects

we found the Iqr range for approved projects and rejected projects and found the range for approved projects under project_title category is High and the for rejected projects is low.

The mean for approved projects and rejected projects is Same though the IQR range is different.

In [0]:

```
plt.figure(figsize=(10,3))
sns.kdeplot(approved_title_word_count,label="Approved Projects", bw=0.6)
sns.kdeplot(rejected_title_word_count,label="Not Approved Projects", bw=0.6)
plt.legend()
plt.show()
```

We plotted the PDF curve for numerical data of approved projects and rejected projects under project_title category

the PDF for both is almost similar both approved and rejected projects Numerical data under project_title category

The PDF will be calcuated based on the distribution of data which is Gaussian distribution for both approved and rejected projects numerical data under project_title category

Approved and rejected projects numerical data do follow gaussian distribution as the PDF for both is Gaussian curve

we can say whether the project will be accepted or rejected based on no.of.words present in the project title as PDF is almost same for both which gives the probability whther the project is accepted or rejected.

So we do need other few features To predict whether the project will be accepted or rejected

1.2.7 Univariate Analysis: Text features (Project Essay's)

In [0]:

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

In [0]:

```
approved_word_count = project_data[project_data['project_is_approved']==1]['essay'].str.split().app
ly(len)
approved_word_count = approved_word_count.values

rejected_word_count = project_data[project_data['project_is_approved']==0]['essay'].str.split().app
ly(len)
rejected_word_count = rejected_word_count.values
```

In [0]:

```
word_count_essays = project_data['essay'].str.split().apply(len)
word_count_essays = word_count_essays.values
print(word_count_essays)
```

In [0]:

```
word_count_resource_summary = project_data['project_resource_summary'].str.split().apply(len)
word_count_resource_summary = word_count_resource_summary.values
print(word_count_resource_summary)
```

In [0]:

```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
plt.boxplot([approved_word_count, rejected_word_count])
plt.title('Words for each essay of the project')
plt.xticks([1,2],('Approved Projects','Rejected Projects'))
plt.ylabel('Words in project essays')
plt.grid()
plt.show()
```

We calculated boxplots for Approved projects and rejected projects on no.of.words present in essay

There are many outliers for both numerical data

The numerical data is whether the project is approved and rejected based on no.of.words present in the essay.

there many outliers for approved and rejected projects data on no.of.words present in essay

The IQR and median is almost similar for both approved and rejected data under no.of.words present in essay.

In [0]:

```
plt.figure(figsize=(10,3))
sns.distplot(approved_word_count, hist=False, label="Approved Projects")
sns.distplot(rejected_word_count, hist=False, label="Not Approved Projects")
plt.title('Words for each essay of the project')
plt.xlabel('Number of words in each eassay')
plt.legend()
plt.show()
```

We plotted the PDF curve for numerical data of approved projects and rejected projects under no.of.words present in the essay category.

the PDF for both is almost similar both approved and rejected projects Numerical data under no.of.words present in the essay category

The PDF will be calcuated based on the distribution of data which is Gaussian distribution for both approved and rejected projects numerical data under no.of.words present in the essay category

Approved and rejected projects numerical data under no.of.words present in the essay do follow gaussian distribution as the PDF for both is Gaussian curve

we cann say whether the project will be accepted or rejected based on no.of.words present in the essay as PDF is almost same for both which gives the probability whther the project is accepted or rejected.

So we do need other few features To predict whether the project will be accepted or rejected

1.2.8 Univariate Analysis: Cost per project

In [0]:

```
# we get the cost of the project using resource.csv file
resource_data.head(2)
```

In [0]:

```
# 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)
```

In [0]:

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

In [0]:

```
approved_price = project_data[project_data['project_is_approved']==1]['price'].values

rejected_price = project_data[project_data['project_is_approved']==0]['price'].values
```

In [0]:

```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
plt.boxplot([approved_price, rejected_price])
plt.title('Box Plots of Cost per approved and not approved Projects')
plt.xticks([1,2],('Approved Projects','Rejected Projects'))
plt.ylabel('Price')
```

```
plt.ylabel('Price')
plt.grid()
plt.show()
```

We calculated boxplots for Approved projects and rejected projects based on the price of project

There are many outliers for both numerical data for approved and rejected projects based on the price of project

The numerical data is whether the project is approved and rejected based on the price of project

there too many outliers for approved and rejected projects data based on the price of project and the IQR rane is too small for the data. As there are to many outliers in the data We could normalize and standard them to scale them under a certain range

The IQR and median is almost similar for both approved and rejected data based on the price of project.

In [0]:

```
plt.figure(figsize=(10,3))
sns.distplot(approved_price, hist=False, label="Approved Projects")
sns.distplot(rejected_price, hist=False, label="Not Approved Projects")
plt.title('Cost per approved and not approved Projects')
plt.xlabel('Cost of a project')
plt.legend()
plt.show()
```

We plotted the PDF curve for numerical data of approved projects and rejected projects based on the price of project and the plot do not follow PDF-Gaussian distribution as the graph increases and decreases linearly at a particular peak

the curve for both is almost similar both approved and rejected projects Numerical data based on the price of project

Approved and rejected projects numerical data based on the price of project do not follow gaussian distribution as the curve is linear .

we can say whether the project will be accepted or rejected based on the price of project as PDF which gives the probability whther the project is accepted or rejected but here the curve is linear, its difficult to calculate whether the project will be calculated or rejected.

So we do need other few features To predict whether the project will be accepted or rejected

In [0]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable

x = PrettyTable()
x.field_names = ["Percentile", "Approved Projects", "Not Approved Projects"]

for i in range(0,101,5):
    x.add_row([i,np.round(np.percentile(approved_price,i), 3), np.round(np.percentile(rejected_pric
e,i), 3)])
print(x)
```

Here we created a table and calculated the percentiles for the approved and rejected data based on the price of project

We can say that the percentiles are larger for rejected projects as the price is high for rejected projects when compared to same percentile of approved projects

1.2.9 Univariate Analysis: teacher_number_of_previously_posted_projects

In [0]:

```
project_data['teacher_number_of_previously_posted_projects'].head(10)
approved_ppp = project_data[project_data['project_is_approved']==1]
['teacher_number_of_previously_posted_projects'].values
rejected_ppp = project_data[project_data['project_is_approved']==0]
['teacher_number_of_previously_posted_projects'].values
```

In [0]:

```
plt.boxplot([approved_ppp, rejected_ppp])
plt.title('Box Plots of Cost per approved and not approved Projects')
plt.xticks([1,2],('Approved Projects','Rejected Projects'))
plt.ylabel('teacher_number_of_previously_posted_projects')
plt.grid()
plt.show()
```

We calculated boxplots for Approved projects and rejected projects based on No.of.previously posted projects by a teacher

There are many outliers for both numerical data for approved and rejected projects based on No.of.previously posted projects by a teacher

The numerical data is whether the project is approved and rejected based on No.of.previously posted projects by a teacher

there too many outliers for approved and rejected projects data based on No.of.previously posted projects by a teacher and the IQR rane is too small for the data. As there are to many outliers in the data We could normalize and standard them to scale them under a certain range

The IQR and median is almost similar for both approved and rejected data No.of.previously posted projects by a teacher

In [0]:

```
x = PrettyTable()
x.field_names = ["Percentile", "Approved Projects", "Not Approved Projects"]

for i in range(0,101,5):
    x.add_row([i,np.round(np.percentile(approved_ppp,i), 3), np.round(np.percentile(rejected_ppp,i)
, 3)])
print(x)
```

Here we created a table and calculated the percentiles for the approved and rejected data No.of.previously posted projects by a teacher

The No.of.previously posted projects by a teacher are very less for a smaller range and more for a larger range

This means large no.of. teachers posted large no.of. projects previously and few no.of.teachers posted few no.of.projects previously.

In [0]:

```
plt.figure(figsize=(10,3))
sns.distplot(approved_ppp, hist=False, label="Approved Projects")
sns.distplot(rejected_ppp, hist=False, label="Not Approved Projects")
plt.title('Cost per approved and not approved Projects')
plt.xlabel('Cost of a project')
plt.legend()
plt.show()
```

We plotted the PDF curve for numerical data of approved projects and rejected projects No.of.previously posted projects by a teacher and the plot do not follow PDF-Gaussian distribution as the graph increases and decreases linearly at a particular peak

the curve for both is almost similar both approved and rejected projects Numerical data No.of.previously posted projects by a teacher

Approved and rejected projects numerical data No.of.previously posted projects by a teacher do not follow gaussian distribution as the curve is linear .

we can say whether the project will be accepted or rejected No.of.previously posted projects by a teacher as PDF which gives the probability whther the project is accepted or rejected but here the curve is linear, its difficult to calculate whether the project will be calculated or rejected.

So we do need other few features To predict whether the project will be accepted or rejected

1.2.10 Univariate Analysis: project_resource_summary

Please do this on your own based on the data analysis that was done in the above cells

Check if the `presence of the numerical digits` in the `project_resource_summary` effects the acceptance of the project or not. If you observe that `presence of the numerical digits` is helpful in the classification, please include it for further process or you can ignore it.

In [0]:

```
k=project_data['project_resource_summary'].shape
n = k[0]
print(n)
b=project_data['project_resource_summary'][45]
print(b)
project_data['No.of.digits'] = 0
project_data.head(5)
```

In [0]:

```
for j in tqdm(range(70000)):
    sample = project_data['project_resource_summary'][j] #sample string
    letters = 0
    numeric = 0

    for i in sample:
                if i.isdigit():
                    numeric +=1
                elif i.isalpha():
                    letters +=1
                else:
                    pass
    project_data['No.of.digits'][j] = numeric
#count no of digits in a string        https://stackoverflow.com/questions/24878174/how-to-count-
digits-letters-spaces-for-a-string-in-python
```

CALCULATED ANALYSIS on NO.OF.DIGITS in THE PROJECT_RESOURCE_SUMMARY TEXT on 10000 points only as its taking HUGE TIME on my LAPTOP

In [0]:

```
project_data[:69999].to_pickle('gdrive/My Drive/naive_bayes_no.of.digits_original.pkl')
```

In [0]:

```
project_data = pd.read_pickle('gdrive/My Drive/naive_bayes_no.of.digits_original.pkl')
```

In [0]:

```
digits_project_resource_summary = project_data['No.of.digits']
```

In [0]:

```
rejected_np[:100]
```

As we can see the no.of.digits in the text are few and the text do contain any few digits

Approved_np and rejected_np is very sparse as we can see

In [0]:

```
plt.boxplot([approved_np[:49999], rejected_np[:49999]])
plt.title('Box Plots of No.of.digits in project_resource_summary per approved and not approved Pro
jects')
plt.xticks([1,2],('Approved Projects','Rejected Projects'))
plt.ylabel('No.of.digits in project_resource_summary')
plt.grid()
plt.show()
```

FOR 10000 DATAPOINTS ONLY

The no.of.digits in No.of.digits in project_resource_summary text are few and very less,So we can IQR is almost zero and the text which has values are treated as outliers mostly for both approved and rejected projects

In [0]:

```
plt.figure(figsize=(10,3))
sns.distplot(approved_np, hist=False, label="Approved Projects")
sns.distplot(rejected_np, hist=False, label="Not Approved Projects")
plt.title('No.of.digits in project_resource_summary per approved and not approved Projects')
plt.xlabel('No.of.digits in project_resource_summary')
plt.legend()
plt.show()
```

This do not follow any gaussian distribution and the curve for approved and rejected is non-linear and non-symmetric and do have any local maximum and local minimum

we need to look for other features in the Project_data and check whether other features data follow any distributions

1.3 Text preprocessing

1.3.1 Essay Text

In [0]:

```
project_data.shape
```

In [0]:

```
# printing some random essays.
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
```

In [0]:

```
# 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 [0]:

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

In [0]:

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

```
sent = sent.replace('\\n', ' ')
print(sent)
```

In [0]:

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

In [0]:

```
# 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","nan"]
```

In [0]:

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

In [0]:

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

1.3.2 Project title Text-Cleaning

In [0]:

```
# similarly you can preprocess the titles also
```

In [0]:

```
project_data['project_title'].values[50]
```

In [0]:

```
project_data['title'] = project_data['project_title'].map(str)
project_data['title'].values[0]
```

In [0]:

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

We are checking is decontracted applied to 20000th value of project_title

In [0]:

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

we are replacing special characters in the project_title with space

In [0]:

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

we are replacing numerical charcaters in the text with space

In [0]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = sent.replace(',',' ')
    sent = sent.replace('!',' ')
    sent = sent.replace('*',' ')
    sent = sent.replace('.',' ')
    sent = sent.replace(':',' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
```

In [0]:

```
preprocessed_titles[20000]
```

Resource_summary_Text

In [0]:

```
project_data['resources_summary'] = project_data['project_resource_summary'].map(str)
project_data['resources_summary'].values[0]
```

In [0]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_resource_summary = []
# tqdm is for printing the status bar
for sentance in project_data['project_resource_summary'].values:
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = sent.replace(',',' ')
    sent = sent.replace('!',' ')
    sent = sent.replace('*',' ')
    sent = sent.replace('.',' ')
    sent = sent.replace(':',' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_resource_summary.append(sent.lower().strip())
```

In [0]:

```
!pip install autocorrect
```

SpellingMistakes Count

Resource_summary

In [0]:

```
from autocorrect import spell
count_spellingmistake_resource = []
for i in tqdm(range(70000)):

    my_words = []

    my_words = preprocessed_resource_summary[i].split()
    my_words = list(set(my_words))
    count = 0
    for j in range(len(my_words)):


        if(my_words[j] != spell(my_words[j])):

            count += 1
        else:

            pass
    count_spellingmistake_resource.append(count)
```

Titles

In [0]:

```
from autocorrect import spell
count_spellingmistake_titles = []
for i in tqdm(range(70000)):


    my_words = []

    my_words = preprocessed_titles[i].split()
    my_words = list(set(my_words))
    count = 0
    for j in range(len(my_words)):
```

```
    for j in range(len(my_words)):


        if(my_words[j] != spell(my_words[j])):

            count += 1
        else:
            pass
    count_spellingmistake_titles.append(count)
```

Essays

In [0]:

```
from autocorrect import spell
count_spellingmistake_essays = []
for i in tqdm(range(70000)):

    my_words = []

    my_words = preprocessed_essays[i].split()
    my_words = list(set(my_words))
    count = 0
    for j in range(len(my_words)):


        if(my_words[j] != spell(my_words[j])):

            count += 1
        else:
            pass
    count_spellingmistake_essays.append(count)
```

Sentimental Analysis of Essays

In [0]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
sid = SentimentIntensityAnalyzer()


for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students w
ith the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelli
gences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of differen
t backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a carin
g community of successful \
learners which can be seen through collaborative student project based learning in and out of the
classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities to practice
a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect of the ki
ndergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role pla
y in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with real food
i will take their idea \
and create common core cooking lessons where we learn important math and writing concepts while co
oking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into maki
ng the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project woul
d expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade apple
```

```
sauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cook
books to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoymen
t for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

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

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

In [0]:

```
ss = sid.polarity_scores(preprocessed_essays[5])
for k in ss:
    print(k,ss[k])
```

In [0]:

```
negative_polarity = []
neutrality_polarity = []
positive_polarity = []
compound_polarity = []


for i in range(70000):

    ss = sid.polarity_scores(preprocessed_essays[i])

    negative_polarity.append(ss['neg'])
    neutrality_polarity.append(ss['neu'])
    positive_polarity.append(ss['pos'])
    compound_polarity.append(ss['compound'])
```

In [0]:

```
negative_polarity = pd.Series(negative_polarity)
neutrality_polarity = pd.Series(neutrality_polarity)
positive_polarity = pd.Series(positive_polarity)
compound_polarity = pd.Series(compound_polarity)
```

In [0]:

```
count_spellingmistake_essays = pd.Series(count_spellingmistake_essays)
count_spellingmistake_titles = pd.Series(count_spellingmistake_titles)
count_spellingmistake_resource = pd.Series(count_spellingmistake_resource)
```

1. 4 Preparing data for models

In [0]:

```
project_data.columns
```

1.4.1 Vectorizing Categorical data

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

School_state- One hot encoding

In [0]:

```
clean_categories  = project_data['clean_categories'].values
clean_categories = pd.Series(clean_categories.ravel().tolist())
```

```
clean_categories = pd.Series(clean_categories.ravel().tolist())
X = ['clean_categories']
```

Clean_Categories-One-Hot-Encoding

In [0]:

```
clean_subcategories = project_data['clean_subcategories'].values
clean_subcategories = pd.Series(clean_subcategories.ravel().tolist())
X = X + ['clean_subcategories']
```

Clean_SubCategories

In [0]:

```
project_grade_category = project_data['project_grade_category']
project_grade_category = pd.Series(project_grade_category.ravel().tolist())
X = X + ['project_grade_category']
```

Project_Grade_Category

In [0]:

```
school_state = project_data['school_state'].values
school_state = pd.Series(school_state.ravel().tolist())
X = X + ['school_state']
```

Teahcer_prefix-One Hot Encoding

In [0]:

```
project_data[project_data['teacher_prefix'].isnull()]
```

We contain nan values in the teacher_prefix column

In [0]:

```
#replacing nan values in pandas    https://stackoverflow.com/questions/13295735/how-can-i-replace-all-the-nan-values-with-zeros-in-a-column-of-a-pandas-datafra
project_data['teacher_prefix'].value_counts()
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('Mrs.')
project_data['teacher_prefix'].isnull().any()
```

replaced nan values in teacher_prefix with "Mrs." as Mrs. is majority vote

In [0]:

```
teacher_prefix  = project_data['teacher_prefix'].values
teacher_prefix = pd.Series(teacher_prefix.ravel().tolist())
X = X + ['teacher_prefix']
```

In [0]:

```
set5_categorical =
pd.concat([clean_categories,clean_subcategories,project_grade_category,school_state,teacher_prefix],axis=1)
```

Vectorizing Numerical Features

Prize

In [0]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
price_standardized = price_scalar.transform(project_data['price'].values.reshape(-1, 1))
```

In [0]:

```
price_standardized = pd.Series(price_standardized.ravel().tolist())
```

In [0]:

```
X = X + ['price_standardized']
```

teacher_number_of_previously_posted_projects

In [0]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

teacher_number_of_previously_posted_projects_scalar = StandardScaler()
teacher_number_of_previously_posted_projects_scalar.fit(project_data['teacher_number_of_previously_p
osted_projects'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {teacher_number_of_previously_posted_projects_scalar.mean_[0]}, Standard deviation
: {np.sqrt(teacher_number_of_previously_posted_projects_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
teacher_number_of_previously_posted_projects_standardized =
teacher_number_of_previously_posted_projects_scalar.transform(project_data['teacher_number_of_previ
ously_posted_projects'].values.reshape(-1, 1))
```

In [0]:

```
teacher_number_of_previously_posted_projects_standardized =
pd.Series(teacher_number_of_previously_posted_projects_standardized.ravel().tolist())
```

In [0]:

```
X = X + ['previously_posted_projects']
```

Quantity

In [0]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

quantity_scalar = StandardScaler()
quantity_scalar.fit(resource_data['quantity'][:109248].values.reshape(-1,1)) # finding the mean and
standard deviation of this data
print(f"Mean : {quantity_scalar.mean_[0]}, Standard deviation :
{np.sqrt(quantity_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
quantity_standardised = quantity_scalar.transform(resource_data['quantity']
[:109248].values.reshape(-1, 1))
```

In [0]:

```
quantity_standardised = pd.Series(quantity_standardised.ravel().tolist())
```

In [0]:

```
X = X + ['quantity_standardised']
```

Digits in Resource_Summary

Words in Essays

In [0]:

```
from sklearn.preprocessing import StandardScaler

essays_scalar = StandardScaler()
essays_scalar.fit(word_count_essays.reshape(-1,1)) # finding the mean and standard deviation of th
is data
print(f"Mean : {essays_scalar.mean_[0]}, Standard deviation : {np.sqrt(essays_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
essays_words_standardised = essays_scalar.transform(word_count_essays.reshape(-1, 1))
```

In [0]:

```
essays_words_standardised = pd.Series(essays_words_standardised.ravel().tolist())
```

In [0]:

```
X = X  + ['essays_words_standardised']
```

Resource_summary-No.of.Words

In [0]:

```
essays_scalar = StandardScaler()
essays_scalar.fit(word_count_resource_summary.reshape(-1,1)) # finding the mean and standard
deviation of this data
print(f"Mean : {essays_scalar.mean_[0]}, Standard deviation : {np.sqrt(essays_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
summary_words_standardised = essays_scalar.transform(word_count_resource_summary.reshape(-1, 1))
```

In [0]:

```
summary_words_standardised = pd.Series(summary_words_standardised.ravel().tolist())
```

In [0]:

```
X = X + ['summary_words_standardised']
```

Titles-No.of.Words

In [0]:

```
titles_scalar = StandardScaler()
titles_scalar.fit(word_count_titles.reshape(-1,1)) # finding the mean and standard deviation of th
is data
print(f"Mean : {titles_scalar.mean_[0]}, Standard deviation : {np.sqrt(titles_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
titles_words_standardised = titles_scalar.transform(word_count_titles.reshape(-1, 1))
```

In [0]:

```
titles_words_standardised = pd.Series(titles_words_standardised.ravel().tolist())
```

In [0]:

```
X = X + ['titles_words_standardised']
```

Sentimental_score's of Essays

In [0]:

```
neutrality_scalar = StandardScaler()
neutrality_scalar.fit(neutrality_polarity.reshape(-1,1)) # finding the mean and standard deviation
of this data
print(f"Mean : {neutrality_scalar.mean_[0]}, Standard deviation :
{np.sqrt(neutrality_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
neutrality_polarity_standardised = neutrality_scalar.transform(neutrality_polarity.reshape(-1, 1))
```

In [0]:

```
neutrality_polarity_standardised = pd.Series(neutrality_polarity_standardised.ravel().tolist())
```

In [0]:

```
X = X +['neutrality_polarity_standardised']
```

In [0]:

```
compound_scalar = StandardScaler()
compound_scalar.fit(compound_polarity.reshape(-1,1)) # finding the mean and standard deviation of
this data
print(f"Mean : {compound_scalar.mean_[0]}, Standard deviation :
{np.sqrt(compound_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
compound_polarity_standardised = compound_scalar.transform(compound_polarity.reshape(-1, 1))
```

In [0]:

```
compound_polarity_standardised = pd.Series(compound_polarity_standardised.ravel().tolist())
```

In [0]:

```
X = X + ['compound_polarity_standardised']
```

In [0]:

```
positive_scalar = StandardScaler()
positive_scalar.fit(positive_polarity.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {positive_scalar.mean_[0]}, Standard deviation : {np.sqrt(positive_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
positive_polarity_standardised = positive_scalar.transform(positive_polarity.reshape(-1, 1))
```

In [0]:

```
positive_polarity_standardised = pd.Series(positive_polarity_standardised.ravel().tolist())
```

In [0]:

```
X = X + ['positive_polarity_standardised']
```

In [0]:

```
negative_scalar = StandardScaler()
negative_scalar.fit(negative_polarity.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {negative_scalar.mean_[0]}, Standard deviation : {np.sqrt(negative_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
negative_polarity_standardised = negative_scalar.transform(negative_polarity.reshape(-1, 1))
```

In [0]:

```
negative_polarity_standardised = pd.Series(negative_polarity_standardised.ravel().tolist())
```

In [0]:

```
X = X + ['negative_polarity_standardised']
```

In [0]:

```
###  Spelling Mistakes-Count in Essays
```

In [0]:

```
essays_spelling_scalar = StandardScaler()
essays_spelling_scalar.fit(count_spellingmistake_essays.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {essays_spelling_scalar.mean_[0]}, Standard deviation : {np.sqrt(essays_spelling_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
spellingmistake_essays = essays_spelling_scalar.transform(count_spellingmistake_essays.reshape(-1, 1))
```

In [0]:

```
spellingmistake_essays = pd.Series(spellingmistake_essays.ravel().tolist())
```

In [0]:

```
X = X  + ['spellingmistake_essays']
```

Spelling Mistakes-Count in Titles

In [0]:

```
titles_spelling_scalar = StandardScaler()
titles_spelling_scalar.fit(count_spellingmistake_titles.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {titles_spelling_scalar.mean_[0]}, Standard deviation : {np.sqrt(titles_spelling_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
spellingmistake_titles = titles_spelling_scalar.transform(count_spellingmistake_titles.reshape(-1, 1))
```

In [0]:

```
spellingmistake_titles = pd.Series(spellingmistake_titles.ravel().tolist())
```

In [0]:

```
X = X + ['spellingmistake_titles']
```

Spelling Mistakes-Count in Resource_suumary

In [0]:

```
resource_spelling_scalar = StandardScaler()
resource_spelling_scalar.fit(count_spellingmistake_resource.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {resource_spelling_scalar.mean_[0]}, Standard deviation : {np.sqrt(resource_spelling_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
spellingmistake_resource = resource_spelling_scalar.transform(count_spellingmistake_resource.reshape(-1, 1))
```

In [0]:

```
spellingmistake_resource = pd.Series(spellingmistake_resource.ravel().tolist())
```

In [0]:

```
X  = X  + ['spellingmistake_resource']
```

In [0]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

digits_scalar = StandardScaler()
digits_scalar.fit(digits_project_resource_summary.values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {digits_scalar.mean_[0]}, Standard deviation : {np.sqrt(digits_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
digits_standardised = digits_scalar.transform(digits_project_resource_summary.values.reshape(-1, 1))
```

In [0]:

```
digits_standardised = pd.Series(digits_standardised.ravel().tolist())
```

In [0]:

```
X = X + ['digits_standardised']
```

Similarity Between Essay_1 and Essay_2

In [0]:

```
X_features_cn = X
```

In [0]:

```
X_cat_num = X_features_cn
```

X-features_cn conatains all feature-names of only categorial and Numerical-Features

we need to append this X_features_cn to BOW when performing BOW and to TFIDF when performing TFIDF

Converting each feature-vector to Dataframe

In [0]:

```
print(type(teacher_prefix_one_hot))
teacher_prefix_one_hot.shape
df = pd.DataFrame(teacher_prefix_one_hot.toarray().astype(np.float64))
type(df)

print(type(school_state_one_hot))
school_state_one_hot.shape
df1 = pd.DataFrame(school_state_one_hot.toarray().astype(np.float64))
type(df1)

print(type(sub_categories_one_hot))
sub_categories_one_hot.shape
df2 = pd.DataFrame(sub_categories_one_hot.toarray().astype(np.float64))
type(df2)

print(type(categories_one_hot))
categories_one_hot.shape
df3 = pd.DataFrame(categories_one_hot.toarray().astype(np.float64))
type(df3)

type(teacher_number_of_previously_posted_projects_standardized.tolist())
df4=teacher_number_of_previously_posted_projects_standardized.tolist()
type(df4)


type(price_standardized.tolist())
df5=price_standardized.tolist()
type(df5)
```

Combine all numerical and categorical features

In [0]:

```
set5_dataset =
pd.concat([set5_categorical,price_standardized[:69999],teacher_number_of_previously_posted_projects
standardized[:69999],quantity_standardised[:69999],essays_words_standardised[:69999],summary_words
_standardised[:69999],titles_words_standardised[:69999],neutrality_polarity_standardised[:69999],c
ompound_polarity_standardised[:69999],positive_polarity_standardised[:69999],negative_polarity_sta
dardised[:69999],spellingmistake_essays[:69999],spellingmistake_titles[:69999],spellingmistake_res
urce[:69999],digits_standardised[:69999]],axis=1)
```

we are combining all the categorical and numerical features into a single X_cn Sparse Matrix

We are ignoring text features here

In [0]:

```
set5_dataset = set5_dataset[:69000]
```

In [0]:

```
set5_dataset.columns = X
```

Creating Datframe for X_cn as we need to concatenate text features into the dataframe

We can not add Text columns to Sparse matrix as the type of text is 'str and numerical columns as 'int'

dk is the dataframe conatining all categorical and numerical features

In [0]:

```
y = project_data['project_is_approved']
type(y)
```

Taking the output into a series-(y)

In [0]:

```
k =
pd.DataFrame({'preprocessed_essays':preprocessed_essays[:69999],'preprocessed_titles':preprocessed_
titles[:69999],'preprocessed_resource_summary':preprocessed_resource_summary[:69999],'y':y[:69999]
})
```

k is the dataframe conatining all the text features and the output-(y) feature.

We should not be using hstack as the features are of strings and could not concatenate them

k is the dataframe conatining all text and output-y

dataset - contains all features

Dataset is the Dataframe containing all the features text,Categorical and Numerical Features

we need to vectorize the text features only after splitting Dataset into train,test,split

dataset conatins all features with text in raw format and also output-y

In [0]:

```
dataset = pd.concat([set5_dataset,k],axis=1)

dataset.to_pickle('gdrive/My Drive/dataset_RF.pkl')
```

In [0]:

```
dataset = pd.read_pickle('gdrive/My Drive/dataset_RF.pkl')
```

In [76]:

```
y = project_data['project_is_approved']
type(y)
```

Out[76]:

```
pandas.core.series.Series
```

In [0]:

```
X = dataset.columns
```

In [0]:

```
X_feature_dataset = X
```

X = Contains all the features-Names of the Dataset..We need to store them because when we convert them to sparse matrice's,We loose the column-Names

X needs to be stored in the order in which we are storing all the features(Numerical + Categorical)

Train-Test-Split of Dataset

In [0]:

```
from sklearn.model_selection import train_test_split
X_1, X_test, y_1, y_test = train_test_split(dataset[:69000], y[:69000], test_size=0.3,random_state=
0,stratify=y[:69000])

# split the train data set into cross validation train and cross validation test
X_train, X_cv, y_train, y_cv = train_test_split(X_1, y_1, test_size=0.3,random_state=0,stratify=y_1
)
```

Split the essays into Train_test_split

We are splitting the dataset Randomly and by using stratify which means train and test-set contains equal no.of.y values

Stratify means train and test contain same proportion of 1 and 0 -samples or same ratio

while stratifying using cv, we need to stratify using y_train

X_train = train set of essay after cross=validation

X_test = test set of essay

X_cv = cv set of essay

X_1 = train set before cross-Validation

Response Coding

In [86]:

```
# Clean_categories
z = X_train.groupby(['clean_categories'])['y'].value_counts() /
X_train.groupby(['clean_categories'])['y'].count()
z
```

Out[86]:

```
clean_categories                    y
AppliedLearning                     1    0.817117
                                    0    0.182883
AppliedLearning Health_Sports       1    0.809524
                                    0    0.190476
AppliedLearning History_Civics      1    0.783333
                                    0    0.216667
AppliedLearning Literacy_Language   1    0.880000
                                    0    0.120000
AppliedLearning Math_Science        1    0.814815
                                    0    0.185185
AppliedLearning Music_Arts          1    0.807531
                                    0    0.192469
AppliedLearning SpecialNeeds        1    0.799136
                                    0    0.200864
AppliedLearning Warmth Care_Hunger  1    0.800000
```

```
AppliedLearning Warmth Care_Hunger    1    0.800000
                                      0    0.200000
Health_Sports                         1    0.860583
                                      0    0.139417
Health_Sports AppliedLearning         1    0.805556
                                      0    0.194444
Health_Sports History_Civics          1    0.833333
                                      0    0.166667
Health_Sports Literacy_Language       1    0.830827
                                      0    0.169173
Health_Sports Math_Science            1    0.772152
                                      0    0.227848
Health_Sports Music_Arts              1    0.851064
                                      0    0.148936
Health_Sports SpecialNeeds            1    0.870192
                                      0    0.129808
                                              ...
Math_Science Health_Sports            0    0.204545
Math_Science History_Civics           1    0.829268
                                      0    0.170732
Math_Science Literacy_Language        1    0.874433
                                      0    0.125567
Math_Science Music_Arts               1    0.842742
                                      0    0.157258
Math_Science SpecialNeeds             1    0.826958
                                      0    0.173042
Math_Science Warmth Care_Hunger       0    0.666667
                                      1    0.333333
Music_Arts                            1    0.855598
                                      0    0.144402
Music_Arts AppliedLearning            0    0.500000
                                      1    0.500000
Music_Arts Health_Sports              1    0.571429
                                      0    0.428571
Music_Arts History_Civics             1    1.000000
Music_Arts SpecialNeeds               1    0.880000
                                      0    0.120000
Music_Arts Warmth Care_Hunger         0    1.000000
SpecialNeeds                          1    0.815686
                                      0    0.184314
SpecialNeeds Health_Sports            1    0.750000
                                      0    0.250000
SpecialNeeds Music_Arts               1    0.815217
                                      0    0.184783
SpecialNeeds Warmth Care_Hunger       1    1.000000
Warmth Care_Hunger                    1    0.908277
                                      0    0.091723
Name: y, Length: 95, dtype: float64
```

z= what is the probability such that data-point(x) belongs to y=1 and y=0 given x belongs to certain category of clean_categories.

but we need to ditribute this probabilities to all points which belongs certain and category and y=1 or y=0]

In [87]:

```
X_train=X_train.set_index(['clean_categories','y']).sort_index()
X_train.head()
```

Out[87]:

		clean_subcategories	project_grade_category	school_state	teacher_prefix	price_standardized	previ
clean_categories	**y**						
AppliedLearning	**0**	EarlyDevelopment	Grades PreK-2	NC	Ms.	0.498347	0.318
	0	EarlyDevelopment	Grades PreK-2	PA	Ms.	1.303090	-0.401
	0	CharacterEducation Other	Grades 9-12	MT	Mrs.	-0.625474	-0.365

		clean_subcategories	project_grade_category	school_state	teacher_prefix	price_standardized	previo
clean_categories	y	CharacterEducation	Grades 3-5	MO	Ms.	2.704927	-0.401
	0	EarlyDevelopment	Grades 3-5	GA	Mrs.	-0.576657	-0.365

5 rows × 26 columns

we are setting index as clean_categories and y and now we sorting the index such that clean_categories sort based on alphabetical order and ty from 0 to 1.

In [88]:

```
clean_categories_prob = pd.Series(z, index=X_train.index)
X_train = X_train.reset_index()
X_train.head()
```

Out[88]:

	clean_categories	y	clean_subcategories	project_grade_category	school_state	teacher_prefix	price_standardized	pre
0	AppliedLearning	0	EarlyDevelopment	Grades PreK-2	NC	Ms.	0.498347	0.3
1	AppliedLearning	0	EarlyDevelopment	Grades PreK-2	PA	Ms.	1.303090	-0.4
2	AppliedLearning	0	CharacterEducation Other	Grades 9-12	MT	Mrs.	-0.625474	-0.3
3	AppliedLearning	0	CharacterEducation	Grades 3-5	MO	Ms.	2.704927	-0.4
4	AppliedLearning	0	EarlyDevelopment	Grades 3-5	GA	Mrs.	-0.576657	-0.3

5 rows × 28 columns

z conatins probabilities and we need to match them with clean_categories and y, so we can make them index now,this clean_probabilities are formed

In [89]:

```
X_train.drop('clean_categories',axis=1,inplace=True)
X_train['clean_categories_Proba'] = clean_categories_prob.tolist()
X_train['clean_categories_Proba'].head()
```

Out[89]:

```
0    0.182883
1    0.182883
2    0.182883
3    0.182883
4    0.182883
Name: clean_categories_Proba, dtype: float64
```

Drop clean_Categories and replace them with probabilities as clean_categories_proba

In [90]:

```
X_y_1 = X_train[X_train['y']==1]['clean_categories_Proba']
```

```
X_y_1 = X_train[X_train['y']==1]['clean_categories_Proba']
X_y_0 = X_train[X_train['y']==0]['clean_categories_Proba']
X_train['clean_categories_Proba'] = X_y_1.add((1-X_y_0),fill_value=0)
X_train['clean_categories_Proba']
```

Out[90]:

```
0        0.817117
1        0.817117
2        0.817117
3        0.817117
4        0.817117
5        0.817117
6        0.817117
7        0.817117
8        0.817117
9        0.817117
10       0.817117
11       0.817117
12       0.817117
13       0.817117
14       0.817117
15       0.817117
16       0.817117
17       0.817117
18       0.817117
19       0.817117
20       0.817117
21       0.817117
22       0.817117
23       0.817117
24       0.817117
25       0.817117
26       0.817117
27       0.817117
28       0.817117
29       0.817117
           ...
33780    0.908277
33781    0.908277
33782    0.908277
33783    0.908277
33784    0.908277
33785    0.908277
33786    0.908277
33787    0.908277
33788    0.908277
33789    0.908277
33790    0.908277
33791    0.908277
33792    0.908277
33793    0.908277
33794    0.908277
33795    0.908277
33796    0.908277
33797    0.908277
33798    0.908277
33799    0.908277
33800    0.908277
33801    0.908277
33802    0.908277
33803    0.908277
33804    0.908277
33805    0.908277
33806    0.908277
33807    0.908277
33808    0.908277
33809    0.908277
Name: clean_categories_Proba, Length: 33810, dtype: float64
```

we are doing this because there are few probabilties such that y=0 and y =1, we need to cahnge thgem such that what is the probabiltiues usch that y=1, so we can subtract y=0 from 1 so that we can get all the probabiltiues which y=1.

In [0]:

```
# Clean SubCategories
```

```
# Clean_subcategories
z = X_train.groupby(['clean_subcategories'])['y'].value_counts() /
X_train.groupby(['clean_subcategories'])['y'].count()
X_train=X_train.set_index(['clean_subcategories','y']).sort_index()
clean_subcategories_prob = pd.Series(z, index=X_train.index)
X_train = X_train.reset_index()

X_train.drop('clean_subcategories',axis=1,inplace=True)
X_train['clean_subcategories_Proba'] = clean_subcategories_prob.tolist()
X_y_1 = X_train[X_train['y']==1]['clean_subcategories_Proba']
X_y_0 = X_train[X_train['y']==0]['clean_subcategories_Proba']
X_train['clean_subcategories_Proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
#project_grade_category
z = X_train.groupby(['project_grade_category'])['y'].value_counts() /
X_train.groupby(['project_grade_category'])['y'].count()
X_train=X_train.set_index(['project_grade_category','y']).sort_index()
project_grade_category_prob = pd.Series(z, index=X_train.index)
X_train = X_train.reset_index()
X_train.drop('project_grade_category',axis=1,inplace=True)
X_train['project_grade_category_proba'] = project_grade_category_prob.tolist()
X_y_1 = X_train[X_train['y']==1]['project_grade_category_proba']
X_y_0 = X_train[X_train['y']==0]['project_grade_category_proba']
X_train['project_grade_category_proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
# School_State
z = X_train.groupby(['school_state'])['y'].value_counts() / X_train.groupby(['school_state'])['y'].
count()
X_train=X_train.set_index(['school_state','y']).sort_index()
school_state_prob = pd.Series(z, index=X_train.index)
X_train = X_train.reset_index()
X_train.drop('school_state',axis=1,inplace=True)
X_train['school_state_proba'] = school_state_prob.tolist()
X_y_1 = X_train[X_train['y']==1]['school_state_proba']
X_y_0 = X_train[X_train['y']==0]['school_state_proba']
X_train['school_state_proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
# Teacher_Prefix
z = X_train.groupby(['teacher_prefix'])['y'].value_counts() / X_train.groupby(['teacher_prefix'])[
'y'].count()
X_train=X_train.set_index(['teacher_prefix','y']).sort_index()
teacher_prefix_prob = pd.Series(z, index=X_train.index)

X_train = X_train.reset_index()
X_train.drop('teacher_prefix',axis=1,inplace=True)
X_train['teacher_prefix_proba'] = teacher_prefix_prob.tolist()
X_y_1 = X_train[X_train['y']==1]['teacher_prefix_proba']
X_y_0 = X_train[X_train['y']==0]['teacher_prefix_proba']
X_train['teacher_prefix_proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
X = X_train.columns
```

In [0]:

```
y_train = X_train['y']
```

In [0]:

```
# Clean_categories
z = X_cv.groupby(['clean_categories'])['y'].value_counts() / X_cv.groupby(['clean_categories'])['y
'].count()
X_cv=X_cv.set_index(['clean_categories','y']).sort_index()
clean_categories_prob = pd.Series(z, index=X_cv.index)
```

```
X_cv = X_cv.reset_index()

X_cv.drop('clean_categories',axis=1,inplace=True)
X_cv['clean_categories_Proba'] = clean_categories_prob.tolist()
X_y_1 = X_cv[X_cv['y']==1]['clean_categories_Proba']
X_y_0 = X_cv[X_cv['y']==0]['clean_categories_Proba']
X_cv['clean_categories_Proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
# Clean_SubCategories
z = X_cv.groupby(['clean_subcategories'])['y'].value_counts() /
X_cv.groupby(['clean_subcategories'])['y'].count()
X_cv=X_cv.set_index(['clean_subcategories','y']).sort_index()
clean_subcategories_prob = pd.Series(z, index=X_cv.index)
X_cv = X_cv.reset_index()

X_cv.drop('clean_subcategories',axis=1,inplace=True)
X_cv['clean_subcategories_Proba'] = clean_subcategories_prob.tolist()
X_y_1 = X_cv[X_cv['y']==1]['clean_subcategories_Proba']
X_y_0 = X_cv[X_cv['y']==0]['clean_subcategories_Proba']
X_cv['clean_subcategories_Proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
#project_grade_category
z = X_cv.groupby(['project_grade_category'])['y'].value_counts() /
X_cv.groupby(['project_grade_category'])['y'].count()
X_cv=X_cv.set_index(['project_grade_category','y']).sort_index()
project_grade_category_prob = pd.Series(z, index=X_cv.index)
X_cv = X_cv.reset_index()
X_cv.drop('project_grade_category',axis=1,inplace=True)
X_cv['project_grade_category_proba'] = project_grade_category_prob.tolist()
X_y_1 = X_cv[X_cv['y']==1]['project_grade_category_proba']
X_y_0 = X_cv[X_cv['y']==0]['project_grade_category_proba']
X_cv['project_grade_category_proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
# School_State
z = X_cv.groupby(['school_state'])['y'].value_counts() / X_cv.groupby(['school_state'])['y'].count(
)
X_cv=X_cv.set_index(['school_state','y']).sort_index()
school_state_prob = pd.Series(z, index=X_cv.index)
X_cv = X_cv.reset_index()
X_cv.drop('school_state',axis=1,inplace=True)
X_cv['school_state_proba'] = school_state_prob.tolist()
X_y_1 = X_cv[X_cv['y']==1]['school_state_proba']
X_y_0 = X_cv[X_cv['y']==0]['school_state_proba']
X_cv['school_state_proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
# Teacher_Prefix
z = X_cv.groupby(['teacher_prefix'])['y'].value_counts() / X_cv.groupby(['teacher_prefix'])
['y'].count()
X_cv=X_cv.set_index(['teacher_prefix','y']).sort_index()
teacher_prefix_prob = pd.Series(z, index=X_cv.index)

X_cv = X_cv.reset_index()
X_cv.drop('teacher_prefix',axis=1,inplace=True)
X_cv['teacher_prefix_proba'] = teacher_prefix_prob.tolist()
X_y_1 = X_cv[X_cv['y']==1]['teacher_prefix_proba']
X_y_0 = X_cv[X_cv['y']==0]['teacher_prefix_proba']
X_cv['teacher_prefix_proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
y_cv = X_cv['y']
```

In [0]:

```
# Clean_categories
z = X_test.groupby(['clean_categories'])['y'].value_counts() /
X_test.groupby(['clean_categories'])['y'].count()
X_test=X_test.set_index(['clean_categories','y']).sort_index()
clean_categories_prob = pd.Series(z, index=X_test.index)
X_test = X_test.reset_index()

X_test.drop('clean_categories',axis=1,inplace=True)
X_test['clean_categories_Proba'] = clean_categories_prob.tolist()
X_y_1 = X_test[X_test['y']==1]['clean_categories_Proba']
X_y_0 = X_test[X_test['y']==0]['clean_categories_Proba']
X_test['clean_categories_Proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
# Clean_SubCategories
z = X_test.groupby(['clean_subcategories'])['y'].value_counts() /
X_test.groupby(['clean_subcategories'])['y'].count()
X_test=X_test.set_index(['clean_subcategories','y']).sort_index()
clean_subcategories_prob = pd.Series(z, index=X_test.index)
X_test = X_test.reset_index()

X_test.drop('clean_subcategories',axis=1,inplace=True)
X_test['clean_subcategories_Proba'] = clean_subcategories_prob.tolist()
X_y_1 = X_test[X_test['y']==1]['clean_subcategories_Proba']
X_y_0 = X_test[X_test['y']==0]['clean_subcategories_Proba']
X_test['clean_subcategories_Proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
#project_grade_category
z = X_test.groupby(['project_grade_category'])['y'].value_counts() / X_test.groupby(['project_grade
_category'])['y'].count()
X_test=X_test.set_index(['project_grade_category','y']).sort_index()
project_grade_category_prob = pd.Series(z, index=X_test.index)
X_test = X_test.reset_index()
X_test.drop('project_grade_category',axis=1,inplace=True)
X_test['project_grade_category_proba'] = project_grade_category_prob.tolist()
X_y_1 = X_test[X_test['y']==1]['project_grade_category_proba']
X_y_0 = X_test[X_test['y']==0]['project_grade_category_proba']
X_test['project_grade_category_proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
# School_State
z = X_test.groupby(['school_state'])['y'].value_counts() / X_test.groupby(['school_state'])['y'].co
unt()
X_test=X_test.set_index(['school_state','y']).sort_index()
school_state_prob = pd.Series(z, index=X_test.index)
X_test = X_test.reset_index()
X_test.drop('school_state',axis=1,inplace=True)
X_test['school_state_proba'] = school_state_prob.tolist()
X_y_1 = X_test[X_test['y']==1]['school_state_proba']
X_y_0 = X_test[X_test['y']==0]['school_state_proba']
X_test['school_state_proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
# Teacher_Prefix
z = X_test.groupby(['teacher_prefix'])['y'].value_counts() / X_test.groupby(['teacher_prefix'])['y'
].count()
X_test=X_test.set_index(['teacher_prefix','y']).sort_index()
teacher_prefix_prob = pd.Series(z, index=X_test.index)

X_test = X_test.reset_index()
X_test.drop('teacher_prefix',axis=1,inplace=True)
X_test['teacher_prefix_proba'] = teacher_prefix_prob.tolist()
X_y_1 = X_test[X_test['y']==1]['teacher_prefix_proba']
X_y_0 = X_test[X_test['y']==0]['teacher_prefix_proba']
X_test['teacher_prefix_proba'] = X_y_1.add((1-X_y_0),fill_value=0)
```

In [0]:

```
y_test = X_test['y']
```

1-BOW

1.1Vectorizers of train,test,split of only Raw test-Features

Essays_Vectorizers_BOW

In [0]:

```
X_train_essay =  X_train[:]['preprocessed_essays']
X_cv_essay = X_cv[:]['preprocessed_essays']
X_test_essay = X_test[:]['preprocessed_essays']
```

In [0]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
train_bow_essay = vectorizer.fit_transform(X_train_essay)
cv_bow_essay = vectorizer.transform(X_cv_essay)
test_bow_essay = vectorizer.transform(X_test_essay)

print(train_bow_essay.shape,cv_bow_essay.shape,test_bow_essay.shape)
```

```
(33810, 10476) (14490, 10476) (20700, 10476)
```

Vectorizing the train,test,cv sets of essays-Text features

min_df=10 means, we are using all the words present only in min of 10 documents

We need to fit the vectorizer with train set and then transform to cv,test using the same vectorizer

It is because test and cv should contain the same words as Train-set

In [0]:

```
X = X.tolist() + vectorizer.get_feature_names()
```

we are appending X = which contain BOW-features_names of Essays to Categorical and Numerical-Features

Titles_Vectorizers_bow

In [0]:

```
X_train_titles =  X_train[:]['preprocessed_titles']
X_cv_titles = X_cv[:]['preprocessed_titles']
X_test_titles = X_test[:]['preprocessed_titles']
X_1_titles = X_1[:]['preprocessed_titles']
```

In [0]:

```
vectorizer3 = CountVectorizer(min_df=10)
train_bow_titles = vectorizer3.fit_transform(X_train_titles)
bow_titles_cv = vectorizer3.transform(X_cv_titles)
test_bow_titles = vectorizer3.transform(X_test_titles)
```

Vectorizing the train,test,cv sets of titles-Text features

min_df=10 means, we are using all the words present only in min of 10 documents

Similar Vectorizing has to be done to Titles and titles are also the text vectors

Similar vectorizing has to be done to Titles and titles are also the text vectors

In [0]:

```
X= X + vectorizer3.get_feature_names()
```

In [0]:

```
X_train_summary =  X_train[:]['preprocessed_resource_summary']
X_cv_summary = X_cv[:]['preprocessed_resource_summary']
X_test_summary = X_test[:]['preprocessed_resource_summary']
```

In [0]:

```
vectorizer4 = CountVectorizer(min_df=10)
train_bow_summary = vectorizer4.fit_transform(X_train_summary)
bow_summary_cv = vectorizer4.transform(X_cv_summary)
test_bow_summary = vectorizer4.transform(X_test_summary)
```

In [0]:

```
X= X + vectorizer4.get_feature_names()
```

In [0]:

```
print(train_bow_essay.shape,test_bow_essay.shape,cv_bow_essay.shape)
print(train_bow_titles.shape,test_bow_titles.shape,bow_titles_cv.shape)
print(y_train.shape,y_test.shape,y_cv.shape)
```

```
(33810, 10476) (20700, 10476) (14490, 10476)
(33810, 1639) (20700, 1639) (14490, 1639)
(33810,) (20700,) (14490,)
```

we are using BOW of the text here

As we need to use fit_transform for train of essays and titles and their respective test-set/cv-set should be transformed because they both should have the same no.of.features (train/test and 1/cv-sets).

When transforming CV,Test features , they should have same no.of features/vectorizers similar to Train-set

We need to vectorize the Each train and testset separately and fit the train data and then transform the test data

1.2Extract train,test of only numerical and categorical features

In [30]:

```
import scipy
X_train_cn =
X_train.drop(['y','preprocessed_essays','preprocessed_titles','preprocessed_resource_summary'],axi
s=1)
print(X_train_cn.shape)
X_train_cn = scipy.sparse.csr_matrix(X_train_cn)
print(X_train_cn.shape)
```

```
(33810, 24)
(33810, 24)
```

In [31]:

```
X_test_cn =
X_test.drop(['y','preprocessed_essays','preprocessed_titles','preprocessed_resource_summary'],axis
=1)
print(X_test_cn.shape)
X_test_cn = scipy.sparse.csr_matrix(X_test_cn)
print(X_test_cn.shape)
```

```
(20700, 24)
```

```
(20700, 24)
```

In [32]:

```
X_cv_cn =
X_cv.drop(['y','preprocessed_essays','preprocessed_titles','preprocessed_resource_summary'],axis=1
)
print(X_cv_cn.shape)
X_cv_cn = scipy.sparse.csr_matrix(X_cv_cn)
print(X_cv_cn.shape)
```

```
(14490, 24)
(14490, 24)
```

From the original TrainTest,Cv sets of dataset, we need to drop text of essays and Titles and replace them with Vectorizers of Text of Essays and Titles

We need to keep the Categorical and Numerical columns also along with the vectors of essays and titles

1.3-Train,test,cv sets of ALL features

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler


X_train_bow = hstack((X_train_cn,train_bow_essay,train_bow_titles,train_bow_summary))
X_train_bow = X_train_bow.tocsr()
train_scalar = StandardScaler(with_mean = False)
X_train_bow = train_scalar.fit_transform(X_train_bow)
```

In [0]:

```
X_test_bow =hstack((X_test_cn,test_bow_essay,test_bow_titles,test_bow_summary))
X_test_bow = X_test_bow.tocsr()
test_scalar = StandardScaler(with_mean = False)
X_test_bow = test_scalar.fit_transform(X_test_bow)
```

In [0]:

```
X_cv_bow = hstack((X_cv_cn,cv_bow_essay,bow_titles_cv,bow_summary_cv))
X_cv_bow   = X_cv_bow.tocsr()
cv_scalar = StandardScaler(with_mean = False)
X_cv_bow = cv_scalar.fit_transform(X_cv_bow)
```

Now using hstack concatenate all train sets of categorical,numerical,vectors of essays and vectors of titles -Features

Similarly concatenate all the test sets and cv sets with their respective features

Convert COO-matrix to CSR-Sparse matrix as the input gievn to the KNN should be of Sparse Matrix and Not Dataframe as DF taes more time to Run.

1.4-Applying Randdom Forest on BOW, SET 1

For one Estimator value, we are calculating models with different Depths and we will find the best Model.

In [0]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

```python
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

1.5-AUC with trainset and CV-set using Dataset after CV-spliting

In [0]:

```python
train_auc = []
cv_auc = []

estimator_values = [5, 10, 50, 100, 200, 500, 1000]
depth_values = [2, 3, 4, 5, 6, 7, 8, 9, 10]
for j in tqdm(estimator_values):

  for i in depth_values:
      neigh = RandomForestClassifier(n_estimators=j, max_depth=i,class_weight="balanced")
      neigh.fit(X_train_bow, y_train)
      # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the po
sitive class
      # not the predicted outputs


      y_train_pred = []
      for k in range(0,X_train_bow.shape[0],1000):
          y_train_pred.extend(neigh.predict_proba(X_train_bow[k:k+1000])[:,1])

      y_cv_pred = []


      for k in range(0, X_cv_bow.shape[0],1000):
          y_cv_pred.extend(neigh.predict_proba(X_cv_bow[k:k+1000])[:,1])



      train_auc.append(roc_auc_score(y_train,y_train_pred))
      cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
  0%|          | 0/7 [00:00<?, ?it/s]
 14%|█         | 1/7 [00:04<00:27,  4.64s/it]
 29%|██        | 2/7 [00:11<00:26,  5.30s/it]
 43%|███       | 3/7 [00:35<00:43, 10.84s/it]
 57%|████      | 4/7 [01:20<01:03, 21.21s/it]
 71%|█████     | 5/7 [02:50<01:23, 41.68s/it]
 86%|██████    | 6/7 [06:23<01:33, 93.22s/it]
100%|███████   | 7/7 [13:31<00:00, 193.53s/it]
```

In [0]:

```python
cv_auc_t = np.array(cv_auc).reshape(7,9).T
train_auc_t = np.array(train_auc).reshape(7,9).T
```

In [0]:

```python
df_cv = pd.DataFrame(cv_auc_t,columns=estimator_values)
df_cv.index = depth_values
df_cv

df_train = pd.DataFrame(train_auc_t,columns=estimator_values)
df_train.index = depth_values
df_train
```

Out[0]:

	5	10	50	100	200	500	1000

	5	10	50	100	200	500	1000
2	0.57705[illegible]	0.6179[illegible]	0.6800[illegible]	0.724[illegible]	0.731[illegible]	0.739[illegible]	0.739[illegible]
3	0.611734	0.640761	0.706160	0.727680	0.749888	0.749243	0.756414
4	0.645062	0.656848	0.744035	0.755900	0.759500	0.763679	0.766450
5	0.648743	0.680902	0.748003	0.767211	0.773063	0.778334	0.783090
6	0.654763	0.701839	0.771026	0.780630	0.788488	0.794508	0.794430
7	0.660130	0.720624	0.782374	0.788125	0.801785	0.812221	0.813510
8	0.684067	0.723724	0.796471	0.817908	0.820728	0.833452	0.831753
9	0.694568	0.736933	0.815566	0.840234	0.842066	0.845787	0.849001
10	0.692495	0.747634	0.840962	0.850933	0.862798	0.865528	0.865338

Heat-Map TRAIN_AUC

X_Axis = n_Estimators

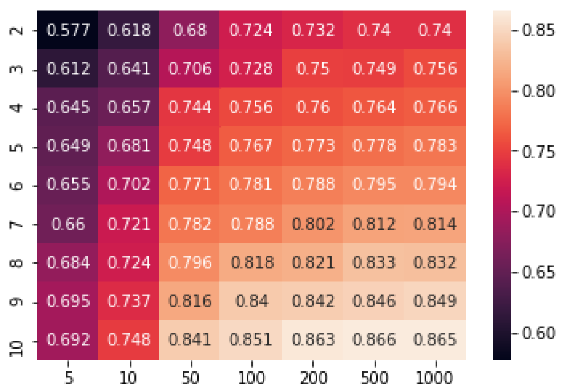
Y_Axis = Depth

In [0]:

```
import seaborn as sns
sns.heatmap(df_train,annot=True,fmt='.3g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c4bbe9550>
```



In [0]:

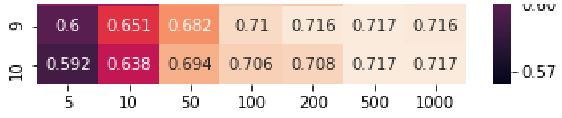
```
### Heat-Map-CV_AUC
```

In [0]:

```
import seaborn as sns
sns.heatmap(df_cv,annot=True,fmt='.3g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c4ea9e4e0>
```



The Best model is the model with highest CV_AUC and please check the hyperparameters of this model.

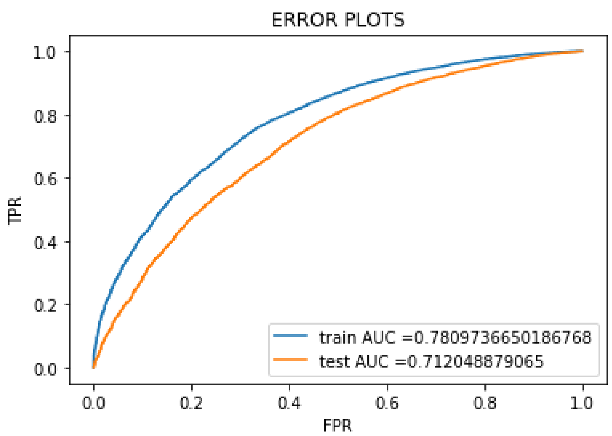
Please also check the TRAIN_AUC such that the model should not be overfitting, which means the train_auc should not be very high

Best_depth = 5 and best_N_Estimators = 75

1.7-ROC-Curve with optimal_k for train and test-sets

In [0]:

```
from sklearn.metrics import roc_curve, auc


neigh = RandomForestClassifier(n_estimators=1000, max_depth=5,class_weight="balanced")
neigh.fit(X_train_bow, 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 = []
for k in range(0,X_train_bow.shape[0],1000):
  y_train_pred.extend(neigh.predict_proba(X_train_bow[k:k+1000])[:,1])

y_test_pred = []


for k in range(0, X_test_bow.shape[0],1000):
  y_test_pred.extend(neigh.predict_proba(X_test_bow[k:k+1000])[:,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("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.show()
print("="*100)
```



==

In [0]:

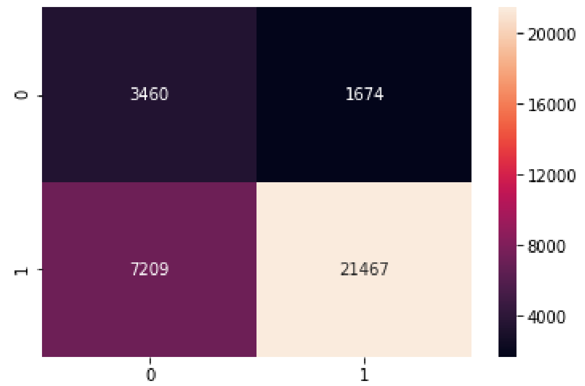
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):
```

```
    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

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

import seaborn as sns
from sklearn.metrics import confusion_matrix
print("train Confusion Matrix")
cm_train=confusion_matrix(y_train,predict( y_train_pred, tr_thresholds, train_fpr, train_tpr))
sns.heatmap(cm_train,annot=True,fmt='.5g')
```

```
train Confusion Matrix
the maximum value of tpr*(1-fpr) 0.5045137640024033 for threshold 0.498
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c4e784588>
```



In [0]:

```
y_train.value_counts()
```

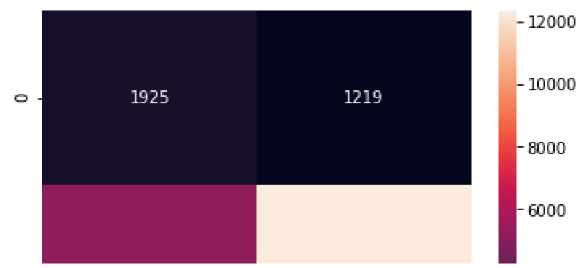
Out[0]:

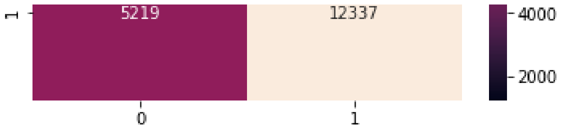
```
1    28676
0     5134
Name: y, dtype: int64
```

In [0]:

```
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict( y_test_pred, te_thresholds, test_fpr, test_tpr))
print(sns.heatmap(cm,annot=True,fmt='.5g'))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.430261204856926 for threshold 0.494
AxesSubplot(0.125,0.125;0.62x0.755)
```





In [0]:

```
y_test.value_counts()
```

Out[0]:

```
1    17556
0     3144
Name: y, dtype: int64
```

Gradient-Boosting with BOW using Train_auc and Cv_auc

In [0]:

```
train_auc = []
cv_auc = []

estimator_values = [5, 10, 50, 100, 200, 500, 1000]
depth_values = [2, 3, 4, 5, 6, 7, 8, 9, 10]

for j in tqdm(estimator_values):

  for i in depth_values:
      neigh = XGBClassifier(n_estimators=j, max_depth=i,class_weight="balanced")
      neigh.fit(X_train_bow, y_train)
      # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the po
sitive class
      # not the predicted outputs


      y_train_pred = []
      for k in range(0,X_train_bow.shape[0],1000):
          y_train_pred.extend(neigh.predict_proba(X_train_bow[k:k+1000])[:,1])

      y_cv_pred = []


      for k in range(0, X_cv_bow.shape[0],1000):
          y_cv_pred.extend(neigh.predict_proba(X_cv_bow[k:k+1000])[:,1])



      train_auc.append(roc_auc_score(y_train,y_train_pred))
      cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
  0%|          | 0/7 [00:00<?, ?it/s]

 14%|█         | 1/7 [00:55<05:33, 55.54s/it]

 29%|██        | 2/7 [02:13<05:11, 62.25s/it]

 43%|███       | 3/7 [06:19<07:49, 117.36s/it]

 57%|████      | 4/7 [13:48<10:50, 216.80s/it]

 71%|█████     | 5/7 [27:48<13:27, 403.87s/it]

 86%|██████    | 6/7 [1:01:30<14:49, 889.30s/it]

100%|███████| 7/7 [2:08:27<00:00, 1827.46s/it]
```

In [0]:

In [0]:

```
cv_auc_t = np.array(cv_auc).reshape(7,9).T
train_auc_t = np.array(train_auc).reshape(7,9).T
```

In [0]:

```
df_cv = pd.DataFrame(cv_auc_t,columns=estimator_values)
df_cv.index = depth_values
df_cv

df_train = pd.DataFrame(train_auc_t,columns=estimator_values)
df_train.index = depth_values
df_train
```

Out[0]:

	5	10	50	100	200	500	1000
2	0.652410	0.669098	0.728208	0.753750	0.779471	0.822711	0.864151
3	0.659628	0.682305	0.753001	0.783034	0.820071	0.878209	0.926833
4	0.684343	0.703859	0.780717	0.819945	0.863262	0.928498	0.969337
5	0.696097	0.723352	0.811526	0.858240	0.903465	0.959832	0.989991
6	0.721079	0.747494	0.852833	0.898437	0.939615	0.982644	0.997521
7	0.739332	0.783786	0.890362	0.934037	0.966462	0.993631	0.999661
8	0.767508	0.814100	0.926305	0.958996	0.982368	0.997989	0.999971
9	0.793246	0.836850	0.949633	0.976597	0.992526	0.999631	1.000000
10	0.811512	0.867143	0.969375	0.987900	0.997005	0.999951	1.000000

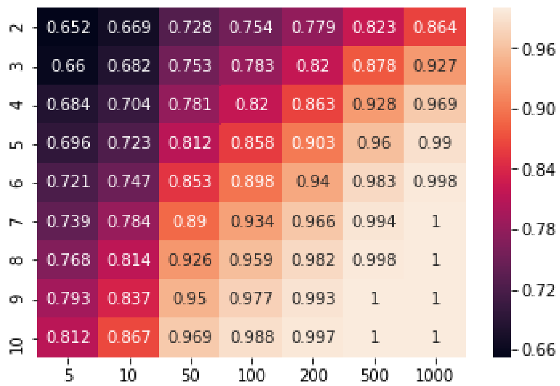
Heat_Map- Train_AUC

In [0]:

```
import seaborn as sns
sns.heatmap(df_train,annot=True,fmt='.3g')
```

Out[0]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f3c4b8ecd68>



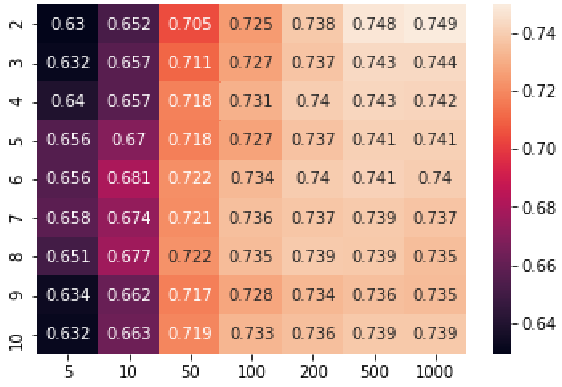
Heat-Map- CV_AUC

In [0]:

```
import seaborn as sns
sns.heatmap(df_cv,annot=True,fmt='.3g')
```

Out[0]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f3c4bd9bac8>



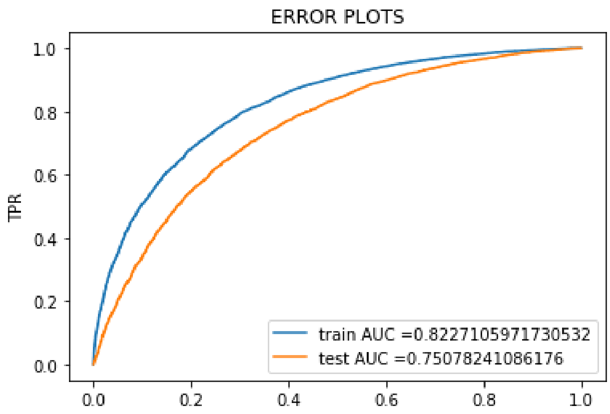
best_depth = 5

best_N_Estimators = 75

ROC-Curve with optimal_k for train and test-sets

In [0]:

```
from sklearn.metrics import roc_curve, auc


neigh = XGBClassifier(n_estimators=500, max_depth=2,class_weight="balanced")
neigh.fit(X_train_bow, 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 = []
for k in range(0,X_train_bow.shape[0],1000):
  y_train_pred.extend(neigh.predict_proba(X_train_bow[k:k+1000])[:,1])

y_test_pred = []


for k in range(0, X_test_bow.shape[0],1000):
  y_test_pred.extend(neigh.predict_proba(X_test_bow[k:k+1000])[:,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("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.show()
print("="*100)
```



FPR

```
==========================================================================================
```

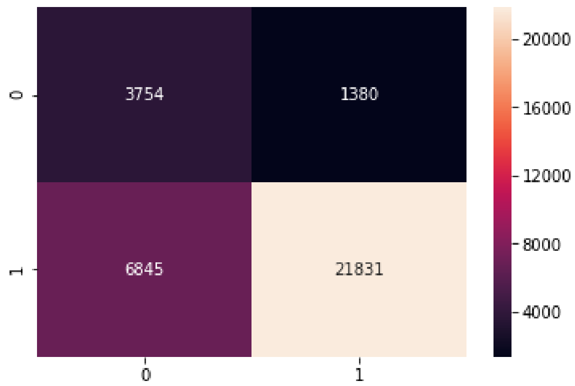
In [0]:

```python
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

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

import seaborn as sns
from sklearn.metrics import confusion_matrix
print("train Confusion Matrix")
cm_train=confusion_matrix(y_train,predict( y_train_pred, tr_thresholds, train_fpr, train_tpr))
sns.heatmap(cm_train,annot=True,fmt='.5g')
```

```
train Confusion Matrix
the maximum value of tpr*(1-fpr) 0.5566644177363441 for threshold 0.827
```

Out[0]:

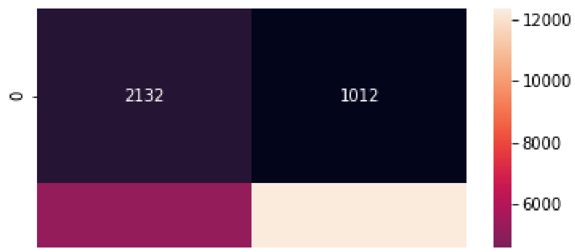
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c4bdac4a8>
```

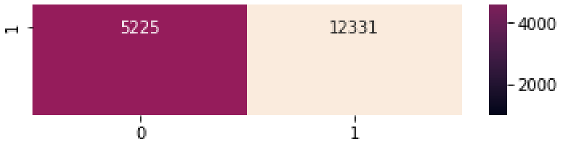


In [0]:

```python
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict( y_test_pred, te_thresholds, test_fpr, test_tpr))
print(sns.heatmap(cm,annot=True,fmt='.5g'))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.476296498243063 for threshold 0.735
AxesSubplot(0.125,0.125;0.62x0.755)
```





2-TFIDF

We can use the same encoding of categorical variables here

2.1-Vectorizer of train,test,split with TFIDF-

In [0]:

```
X = X_train.columns
```

In [0]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer3 = TfidfVectorizer(min_df=10)
train_tfidf_essay = vectorizer3.fit_transform(X_train_essay)
cv_tfidf_essay = vectorizer3.transform(X_cv_essay)
test_tfidf_essay = vectorizer3.transform(X_test_essay)
print(train_tfidf_essay.shape,cv_tfidf_essay.shape,test_tfidf_essay.shape)
```

```
(33810, 10476) (14490, 10476) (20700, 10476)
```

In [0]:

```
X= X.tolist() + vectorizer3.get_feature_names()
```

Vectorizing using TFIDF the train,test,cv sets of Essay-Text features

min_df=10 means, we are using all the words present only in min of 10 documents

In [0]:

```
vectorizer4 = TfidfVectorizer(min_df=10)
train_tfidf_titles = vectorizer4.fit_transform(X_train_titles)
tfidf_titles_cv = vectorizer4.transform(X_cv_titles)
test_tfidf_titles = vectorizer4.transform(X_test_titles)
```

In [0]:

```
X= X + vectorizer4.get_feature_names()
```

In [0]:

```
vectorizer5 = TfidfVectorizer(min_df=10)
train_tfidf_summary = vectorizer5.fit_transform(X_train_summary)
tfidf_summary_cv = vectorizer5.transform(X_cv_summary)
test_tfidf_summary = vectorizer5.transform(X_test_summary)
```

In [0]:

```
X = X + vectorizer5.get_feature_names()
```

we are using TFIDF of the text here

As we need to use fit_transform for train of essays and titles and their respective test-set/cv-set should be transformed because they both should have the same no.of.features (train/test and 1/cv-sets).

When transforming CV,Test features), they should have same no.of features/vectorizers similar to Train set

When transforming CV,Test features \, they should have same no.of features/vectorizers similar to Train-set

Vectorizing using TFIDF the train,test,cv sets of titles-Text features

min_df=10 means, we are using all the words present only in min of 10 documents

2.2-Train,test,cv sets of ALL features -Concatenating

In [0]:

```
from scipy.sparse import hstack
from sklearn import preprocessing


X_train_tfidf = hstack((X_train_cn,train_tfidf_essay,train_tfidf_titles,train_tfidf_summary))
X_train_tfidf = X_train_tfidf.tocsr()
train_scalar = StandardScaler(with_mean = False)
X_train_tfidf = train_scalar.fit_transform(X_train_tfidf)
```

In [0]:

```
X_test_tfidf =hstack((X_test_cn,test_tfidf_essay,test_tfidf_titles,test_tfidf_summary))
X_test_tfidf = X_test_tfidf.tocsr()
test_scalar = StandardScaler(with_mean = False)
X_test_tfidf = test_scalar.fit_transform(X_test_tfidf)
```

In [0]:

```
X_cv_tfidf = hstack((X_cv_cn,cv_tfidf_essay,tfidf_titles_cv,tfidf_summary_cv))
X_cv_tfidf   = X_cv_tfidf.tocsr()
cv_scalar = StandardScaler(with_mean = False)
X_cv_tfidf = cv_scalar.fit_transform(X_cv_tfidf)
```

Now using hstack concatenate all train sets of categorical,numerical,vectors of essays and vectors of titles -Features

Similarly concatenate all the test sets and cv sets with their respective features

Convert COO-matrix to CSR-Sparse matrix as the input gievn to the KNN should be of Sparse Matrix and Not Dataframe

2.3-AUC with trainset and CV-set using Dataset after CV-spliting

In [0]:

```
train_auc = []
cv_auc = []

estimator_values = [5, 10, 50, 100, 200, 500, 1000]
depth_values = [2, 3, 4, 5, 6, 7, 8, 9, 10]

for j in tqdm(estimator_values):

  for i in depth_values:
      model_tfidf = RandomForestClassifier(n_estimators=j, max_depth=i,class_weight="balanced")
      model_tfidf.fit(X_train_tfidf, y_train)
      # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the po
sitive class
      # not the predicted outputs


      y_train_pred = []
      for k in range(0,X_train_tfidf.shape[0],100):
          y_train_pred.extend(model_tfidf.predict_proba(X_train_tfidf[k:k+100])[:,1])

      y_cv_pred = []


      for k in range(0, X_cv_tfidf.shape[0],100):
          y_cv_pred.extend(model_tfidf.predict_proba(X_cv_tfidf[k:k+100])[:,1])
```

```
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
  0%|          | 0/7 [00:00<?, ?it/s]
 14%|█         | 1/7 [00:08<00:50,  8.37s/it]
 29%|██        | 2/7 [00:21<00:48,  9.69s/it]
 43%|███       | 3/7 [01:02<01:17, 19.31s/it]
 57%|████      | 4/7 [02:24<01:53, 37.87s/it]
 71%|█████     | 5/7 [05:02<02:27, 73.97s/it]
 86%|██████    | 6/7 [11:31<02:48, 168.67s/it]
100%|███████| 7/7 [24:31<00:00, 351.85s/it]
```

In [0]:

```
cv_auc_t = np.array(cv_auc).reshape(7,9).T
train_auc_t = np.array(train_auc).reshape(7,9).T

df_cv = pd.DataFrame(cv_auc_t,columns=estimator_values)
df_cv.index = depth_values
df_cv

df_train = pd.DataFrame(train_auc_t,columns=estimator_values)
df_train.index = depth_values
df_train
```

Out[0]:

	5	10	50	100	200	500	1000
2	0.597298	0.622420	0.709707	0.723709	0.731534	0.744051	0.744137
3	0.625633	0.637526	0.728524	0.745415	0.755582	0.759455	0.759771
4	0.596297	0.667129	0.747537	0.754032	0.771667	0.773621	0.775257
5	0.646986	0.698210	0.768353	0.781358	0.781392	0.794029	0.792316
6	0.673932	0.703243	0.780671	0.794842	0.805944	0.808849	0.810114
7	0.653388	0.720955	0.795804	0.811502	0.823003	0.827359	0.826291
8	0.687214	0.735563	0.824062	0.835977	0.837221	0.846419	0.851948
9	0.704676	0.764773	0.843384	0.853477	0.867546	0.867820	0.869552
10	0.723784	0.769793	0.863885	0.869240	0.882010	0.884822	0.887877

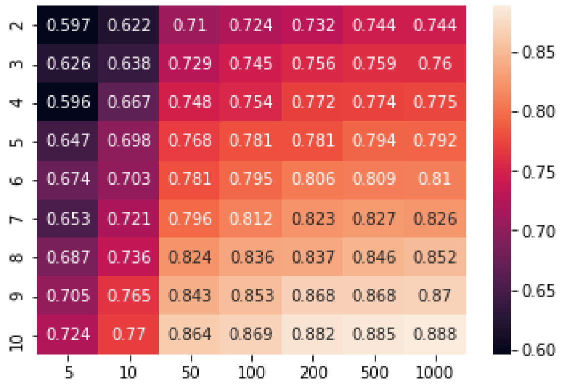
Heat_Map Train_AUC

In [0]:

```
import seaborn as sns
sns.heatmap(df_train,annot=True,fmt='.3g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c4e7ac048>
```



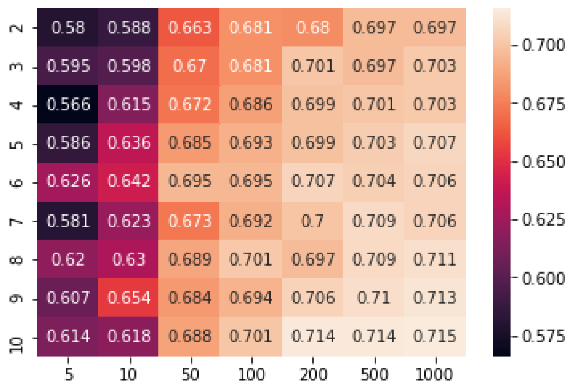
Heat_Map-CV_AUC

In [0]:

```python
import seaborn as sns
sns.heatmap(df_cv,annot=True,fmt='.3g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c4eefb0f0>
```



In [0]:

```python
Best_depth = 10
```

In [0]:

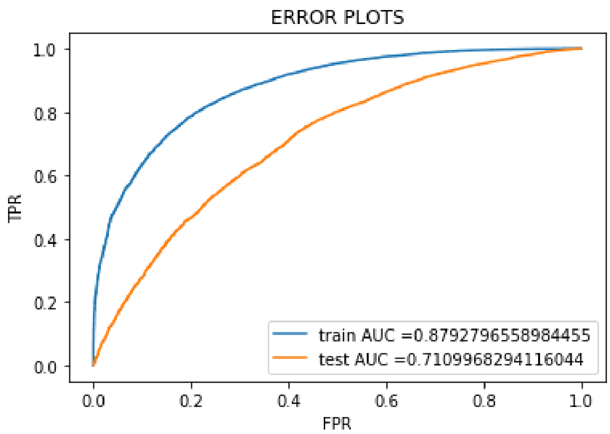
```python
Best_n_Estiomators = 200
```

2.5-ROC-Curve with optimal_k for train and test-sets

In [0]:

```python
from sklearn.metrics import roc_curve, auc


model_tfidf = RandomForestClassifier(n_estimators=200, max_depth=10,class_weight="balanced")
model_tfidf.fit(X_train_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 = []
```

```
y_train_pred = []
for k in range(0,X_train_tfidf.shape[0],100):
  y_train_pred.extend(model_tfidf.predict_proba(X_train_tfidf[k:k+100])[:,1])

y_test_pred = []


for k in range(0, X_test_tfidf.shape[0],100):
  y_test_pred.extend(model_tfidf.predict_proba(X_test_tfidf[k:k+100])[:,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("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)
```




```
====================================================================================================
```

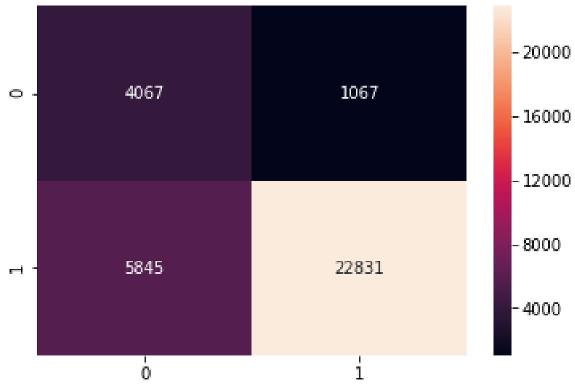
In [0]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

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

import seaborn as sns
from sklearn.metrics import confusion_matrix
print("train Confusion Matrix")
cm_train=confusion_matrix(y_train,predict( y_train_pred, tr_thresholds, train_fpr, train_tpr))
sns.heatmap(cm_train,annot=True,fmt='.5g')
```

```
train Confusion Matrix
the maximum value of tpr*(1-fpr) 0.6307026712695112 for threshold 0.509
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c4e292320>
```



In [0]:

```
y_train.value_counts()
```

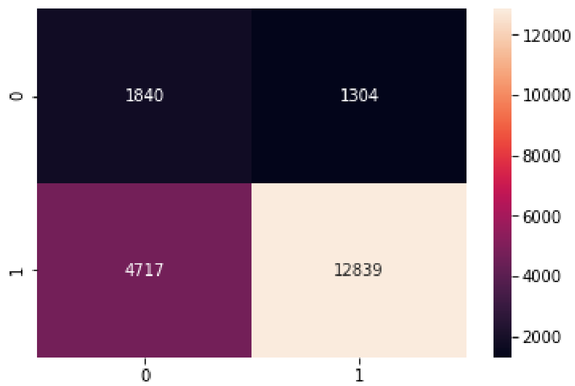
Out[0]:

```
1    28676
0     5134
Name: y, dtype: int64
```

In [0]:

```
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict( y_test_pred, te_thresholds, test_fpr, test_tpr))
print(sns.heatmap(cm,annot=True,fmt='.5g'))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.427997184726795 for threshold 0.498
AxesSubplot(0.125,0.125;0.62x0.755)
```



In [0]:

```
y_test.value_counts()
```

Out[0]:

```
1    17556
0     3144
Name: y, dtype: int64
```

Gradient-Boosting with TFIDF

In [0]:

```
train_auc = []
cv_auc = []
```

```
estimator_values = [5, 10, 50, 100, 200, 500, 1000]
depth_values = [2, 3, 4, 5, 6, 7, 8, 9, 10]

for j in tqdm(estimator_values):

  for i in depth_values:
      model_tfidf = XGBClassifier(n_estimators=j, max_depth=i,class_weight="balanced")
      model_tfidf.fit(X_train_tfidf, y_train)
      # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the po
sitive class
      # not the predicted outputs


      y_train_pred = []
      for k in range(0,X_train_tfidf.shape[0],100):
          y_train_pred.extend(model_tfidf.predict_proba(X_train_tfidf[k:k+100])[:,1])

      y_cv_pred = []


      for k in range(0, X_cv_tfidf.shape[0],100):
          y_cv_pred.extend(model_tfidf.predict_proba(X_cv_tfidf[k:k+100])[:,1])



      train_auc.append(roc_auc_score(y_train,y_train_pred))
      cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
  0%|          | 0/7 [00:00<?, ?it/s]

 14%|█         | 1/7 [01:37<09:44, 97.49s/it]

 29%|██        | 2/7 [04:00<09:15, 111.06s/it]

 43%|███       | 3/7 [12:32<15:25, 231.44s/it]

 57%|████      | 4/7 [28:31<22:29, 449.84s/it]

 71%|█████     | 5/7 [58:51<28:41, 860.89s/it]

 86%|██████    | 6/7 [2:11:56<31:58, 1918.08s/it]

100%|███████| 7/7 [4:36:15<00:00, 3940.16s/it]
```

In [0]:

```
cv_auc_t = np.array(cv_auc).reshape(7,9).T
train_auc_t = np.array(train_auc).reshape(7,9).T

df_cv = pd.DataFrame(cv_auc_t,columns=estimator_values)
df_cv.index = depth_values
df_cv

df_train = pd.DataFrame(train_auc_t,columns=estimator_values)
df_train.index = depth_values
df_train
```

Out[0]:

	5	10	50	100	200	500	1000
2	0.647851	0.671536	0.731873	0.761820	0.791514	0.846188	0.895733

	5	10	50	100	200	500	1000
2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
3	0.674888	0.689429	0.763897	0.799111	0.843257	0.909699	0.958999
4	0.677726	0.713794	0.798357	0.842271	0.891360	0.954508	0.988878
5	0.703925	0.730125	0.835376	0.882934	0.930766	0.981098	0.997937
6	0.724683	0.763335	0.876053	0.920761	0.960859	0.993698	0.999808
7	0.750245	0.787462	0.912554	0.952742	0.981004	0.998383	0.999993
8	0.771523	0.816916	0.941234	0.972830	0.991558	0.999754	1.000000
9	0.798657	0.851276	0.966012	0.987235	0.997288	0.999980	1.000000
10	0.826057	0.880859	0.981319	0.994218	0.999193	0.999999	1.000000

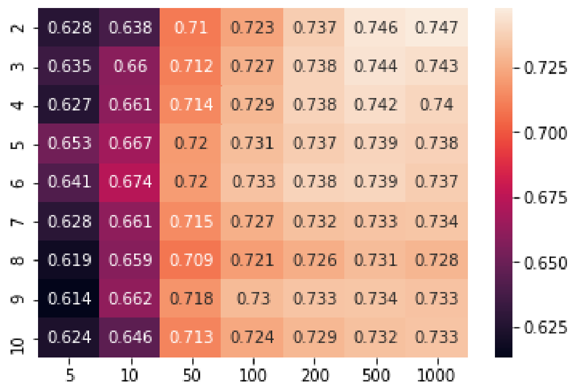
heat_Map Train_AUC

In [0]:

```python
import seaborn as sns
sns.heatmap(df_cv,annot=True,fmt='.3g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c4e326780>
```



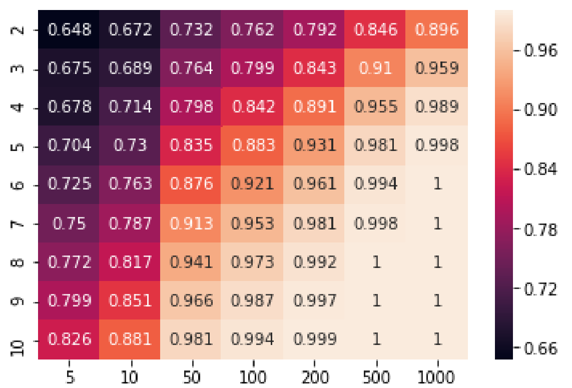
Heat_Map-CV_AUC

In [0]:

```python
import seaborn as sns
sns.heatmap(df_train,annot=True,fmt='.3g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c4e13d390>
```



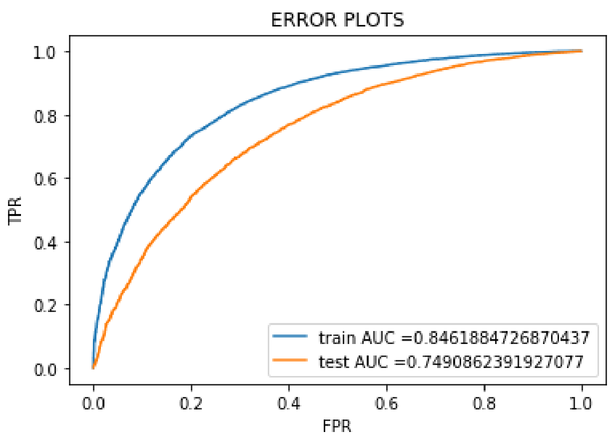
best_depth = 5

best_n_estimators = 75

ROC-Curve with optimal_k for train and test-sets

In [0]:

```
from sklearn.metrics import roc_curve, auc


model_tfidf = XGBClassifier(n_estimators=500, max_depth=2,class_weight="balanced")
model_tfidf.fit(X_train_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 = []
for k in range(0,X_train_tfidf.shape[0],100):
  y_train_pred.extend(model_tfidf.predict_proba(X_train_tfidf[k:k+100])[:,1])

y_test_pred = []


for k in range(0, X_test_tfidf.shape[0],100):
  y_test_pred.extend(model_tfidf.predict_proba(X_test_tfidf[k:k+100])[:,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("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.show()
```



In [0]:

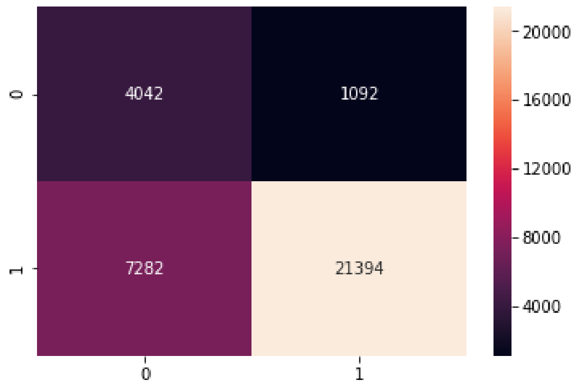
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

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

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
print("train Confusion Matrix")
cm_train=confusion_matrix(y_train,predict( y_train_pred, tr_thresholds, train_fpr, train_tpr))
sns.heatmap(cm_train,annot=True,fmt='.5g')
```

```
train Confusion Matrix
the maximum value of tpr*(1-fpr) 0.5873728449162392 for threshold 0.838
```

Out[0]:

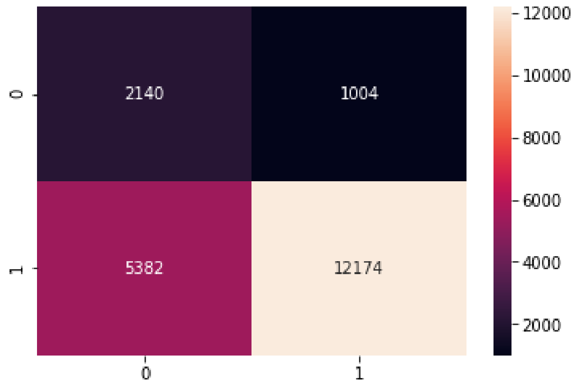
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c4bdafda0>
```



In [0]:

```
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict( y_test_pred, te_thresholds, test_fpr, test_tpr))
print(sns.heatmap(cm,annot=True,fmt='.5g'))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.47199669889505164 for threshold 0.747
AxesSubplot(0.125,0.125;0.62x0.755)
```



In [0]:

```
# TFIDF-W2v
```

In [0]:

```
# 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('gdrive/My Drive/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
```

```
tfidf_model_train = TfidfVectorizer()
tfidf_model_train.fit_transform(X_train_essay)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf_model_train.idf_)))
tfidf_words_train = set(tfidf_model_train.get_feature_names())
```

In [37]:

```
# 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_essay): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_train =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_train):
            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_train += tf_idf
    if tf_idf_weight_train != 0:
        vector /= tf_idf_weight_train
    tfidf_w2v_vectors_train.append(vector)
```

```
100%|██████████| 33810/33810 [00:58<00:00, 573.32it/s]
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train_titles = TfidfVectorizer()
tfidf_model_train_titles.fit_transform(X_train_titles)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train_titles.get_feature_names(), list(tfidf_model_train_titles.
idf_)))
tfidf_words_train_titles = set(tfidf_model_train_titles.get_feature_names())
```

In [39]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_train_titles =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_train_titles):
            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_train_titles += tf_idf
    if tf_idf_weight_train_titles != 0:
        vector /= tf_idf_weight_train_titles
    tfidf_w2v_vectors_train_titles.append(vector)

print(len(tfidf_w2v_vectors_train_titles))
print(len(tfidf_w2v_vectors_train_titles[0]))
```

```
100%|██████████| 33810/33810 [00:01<00:00, 27084.74it/s]
```

```
33810
300
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
```

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train = TfidfVectorizer()
tfidf_model_train.fit_transform(X_train_essay)

tfidf_model_train.transform(X_test_essay)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf_model_train.idf_)))
tfidf_words_test = set(tfidf_model_train.get_feature_names())
```

In [41]:

```
# average Word2Vec
# 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_essay): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_test =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_train):
            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_test += tf_idf
    if tf_idf_weight_test != 0:
        vector /= tf_idf_weight_test
    tfidf_w2v_vectors_test.append(vector)

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

```
100%|██████████| 20700/20700 [00:36<00:00, 562.73it/s]
```

```
20700
300
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train_titles = TfidfVectorizer()
tfidf_model_train_titles.fit_transform(X_train_titles)

tfidf_model_train_titles.transform(X_test_titles)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train_titles.get_feature_names(), list(tfidf_model_train_titles.
idf_)))
tfidf_words_test_titles = set(tfidf_model_train_titles.get_feature_names())
```

In [43]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_test_titles =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_train_titles):
            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_test_titles += tf_idf
    if tf_idf_weight_test_titles != 0:
        vector /= tf_idf_weight_test_titles
    tfidf_w2v_vectors_test_titles.append(vector)

print(len(tfidf_w2v_vectors_test_titles))
print(len(tfidf_w2v_vectors_test_titles[0]))
```

```
100%|██████████| 20700/20700 [00:00<00:00, 30581.44it/s]
```

```
20700
300
```

In [0]:

```python
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train = TfidfVectorizer()
tfidf_model_train.fit_transform(X_train_essay)
tfidf_model_train.transform(X_cv_essay)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf_model_train.idf_)))
tfidf_words_cv = set(tfidf_model_train.get_feature_names())
```

In [45]:

```python
# average Word2Vec
# 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_essay): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_cv =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_train):
            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_cv += tf_idf
    if tf_idf_weight_cv != 0:
        vector /= tf_idf_weight_cv
    tfidf_w2v_vectors_cv.append(vector)

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

```
100%|██████████| 14490/14490 [00:24<00:00, 579.97it/s]
```

```
14490
300
```

In [0]:

```python
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_1_titles = TfidfVectorizer()
tfidf_model_1_titles.fit_transform(X_train_titles)
tfidf_model_1_titles.transform(X_cv_titles)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_1_titles.get_feature_names(), list(tfidf_model_1_titles.idf_)))
tfidf_words_cv_titles = set(tfidf_model_1_titles.get_feature_names())
```

In [47]:

```python
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_cv_titles =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_train_titles):
            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
```

```
idf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight_cv_titles += tf_idf
        if tf_idf_weight_cv_titles != 0:
            vector /= tf_idf_weight_cv_titles
        tfidf_w2v_vectors_cv_titles.append(vector)

print(len(tfidf_w2v_vectors_cv_titles))
print(len(tfidf_w2v_vectors_cv_titles[0]))
```

```
100%|██████████| 14490/14490 [00:00<00:00, 32857.11it/s]
```

```
14490
300
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train = TfidfVectorizer()
tfidf_model_train.fit_transform(X_train_summary)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf_model_train.idf_)))
tfidf_words_train_summary = set(tfidf_model_train.get_feature_names())
```

In [49]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train_summary = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_train_summary): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_train =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_train_summary):
            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_train += tf_idf
    if tf_idf_weight_train != 0:
        vector /= tf_idf_weight_train
    tfidf_w2v_vectors_train_summary.append(vector)

print(len(tfidf_w2v_vectors_train_summary))
print(len(tfidf_w2v_vectors_train_summary[0]))
```

```
100%|██████████| 33810/33810 [00:03<00:00, 10281.79it/s]
```

```
33810
300
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_train = TfidfVectorizer()
tfidf_model_train.fit_transform(X_train_summary)

tfidf_model_train.transform(X_test_summary)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_train.get_feature_names(), list(tfidf_model_train.idf_)))
tfidf_words_test = set(tfidf_model_train.get_feature_names())
```

In [51]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test_summary = []; # the avg-w2v for each sentence/review is stored in this list
```

```
for sentence in tqdm(X_test_summary): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_test =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_train_summary):
            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_test += tf_idf
    if tf_idf_weight_test != 0:
        vector /= tf_idf_weight_test
    tfidf_w2v_vectors_test_summary.append(vector)
```

```
100%|██████████| 20700/20700 [00:01<00:00, 10563.98it/s]
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_1 = TfidfVectorizer()
tfidf_model_1.fit_transform(X_train_summary)
tfidf_model_1.transform(X_cv_summary)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_1.get_feature_names(), list(tfidf_model_1.idf_)))
tfidf_words_cv = set(tfidf_model_1.get_feature_names())
```

In [53]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv_summary = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_summary): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight_cv =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_train_summary):
            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_cv += tf_idf
    if tf_idf_weight_cv != 0:
        vector /= tf_idf_weight_cv
    tfidf_w2v_vectors_cv_summary.append(vector)

print(len(tfidf_w2v_vectors_cv_summary))
print(len(tfidf_w2v_vectors_cv_summary[0]))
```

```
100%|██████████| 14490/14490 [00:01<00:00, 10074.80it/s]
```

```
14490
300
```

Concatenating All numerical and text features

In [0]:

```
from scipy.sparse import hstack
from sklearn.preprocessing import StandardScaler

X_train_tfidf_w2v = hstack((X_train_cn,tfidf_w2v_vectors_train,tfidf_w2v_vectors_train_titles,tfid
f_w2v_vectors_train_summary))
X_train_tfidf_w2v = X_train_tfidf_w2v.tocsr()
train_scalar = StandardScaler(with_mean = False)
X_train_tfidf_w2v = train_scalar.fit_transform(X_train_tfidf_w2v)
```

In [0]:

```
X_test_tfidf_w2v
=hstack((X_test_cn,tfidf_w2v_vectors_test,tfidf_w2v_vectors_test_titles,tfidf_w2v_vectors_test_summ
ary))
X_test_tfidf_w2v = X_test_tfidf_w2v.tocsr()
test_scalar = StandardScaler(with_mean = False)
X_test_tfidf_w2v = test_scalar.fit_transform(X_test_tfidf_w2v)
```

In [0]:

```
X_cv_tfidf_w2v=
hstack((X_cv_cn,tfidf_w2v_vectors_cv,tfidf_w2v_vectors_cv_titles,tfidf_w2v_vectors_cv_summary))
X_cv_tfidf_w2v  = X_cv_tfidf_w2v.tocsr()
cv_scalar = StandardScaler(with_mean = False)
X_cv_tfidf_w2v = cv_scalar.fit_transform(X_cv_tfidf_w2v)


print(X_train_tfidf_w2v.shape,X_test_tfidf_w2v.shape,X_cv_tfidf_w2v.shape)
```

```
(33810, 924) (20700, 924) (14490, 924)
```

Random Forest for TFIDF

In [0]:

```
train_auc = []
cv_auc = []

estimator_values = [5, 10, 50, 100, 200, 500]
depth_values = [2, 3, 4, 5, 6, 7, 8, 9, 10]

for j in tqdm(estimator_values):

  for i in depth_values:
      model_tfidf = RandomForestClassifier(n_estimators=j, max_depth=i,class_weight="balanced")
      model_tfidf.fit(X_train_tfidf_w2v, y_train)
      # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the po
sitive class
      # not the predicted outputs


      y_train_pred = []
      for k in range(0,X_train_tfidf_w2v.shape[0],1000):
          y_train_pred.extend(model_tfidf.predict_proba(X_train_tfidf_w2v[k:k+1000])[:,1])

      y_cv_pred = []


      for k in range(0, X_cv_tfidf_w2v.shape[0],1000):
          y_cv_pred.extend(model_tfidf.predict_proba(X_cv_tfidf_w2v[k:k+1000])[:,1])



      train_auc.append(roc_auc_score(y_train,y_train_pred))
      cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
100%|██████████| 6/6 [2:07:32<00:00, 1878.84s/it]
```

In [0]:

```
cv_auc_t = np.array(cv_auc).reshape(6,9).T
train_auc_t = np.array(train_auc).reshape(6,9).T

df_cv = pd.DataFrame(cv_auc_t,columns=estimator_values)
df_cv.index = depth_values
df_cv

df_train = pd.DataFrame(train_auc_t,columns=estimator_values)
```

```
df_train.index = depth_values
df_train
```

Out[0]:

	5	10	50	100	200	500
2	0.634570	0.652765	0.697827	0.704176	0.703836	0.707988
3	0.665614	0.684025	0.716505	0.727170	0.723168	0.727142
4	0.672474	0.717064	0.745018	0.745017	0.747351	0.747429
5	0.707046	0.737354	0.767023	0.774834	0.780642	0.781338
6	0.734554	0.771336	0.810551	0.819624	0.821587	0.824504
7	0.764397	0.812412	0.861134	0.870311	0.875852	0.876380
8	0.806376	0.860071	0.911885	0.921436	0.924759	0.926725
9	0.843954	0.911718	0.951186	0.960822	0.965525	0.967099
10	0.884985	0.930735	0.979499	0.984335	0.986759	0.988695

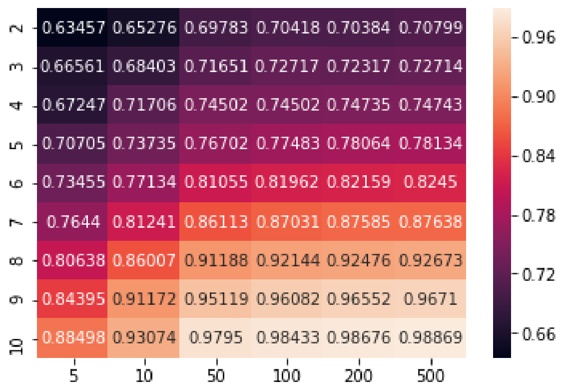
Heat_map Train_AUC

In [0]:

```
import seaborn as sns
sns.heatmap(df_train,annot=True,fmt='.5g')
```

Out[0]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f58bc39e828>



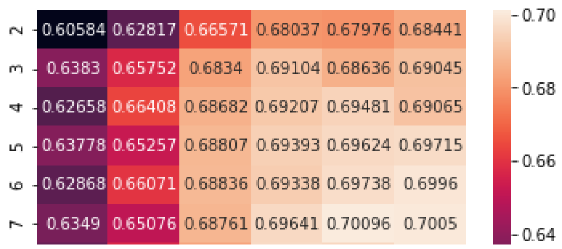
Heat_Map-CV_AUC

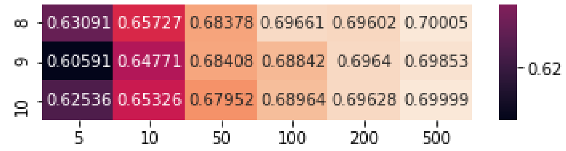
In [0]:

```
import seaborn as sns
sns.heatmap(df_cv,annot=True,fmt='.5g')
```

Out[0]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f58bb84a710>





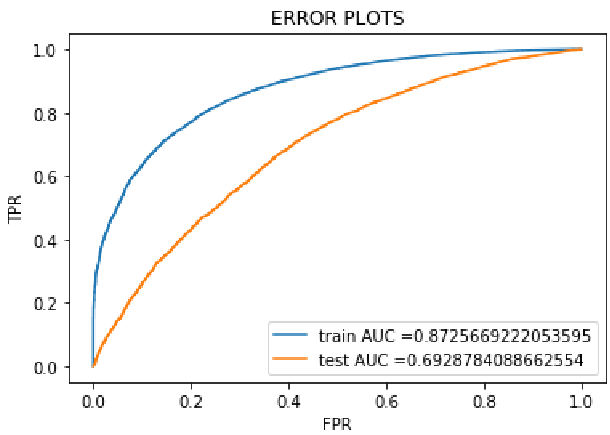
best_depth = 5

best_n_estimators = 50

ROC-Curve with optimal_k for train and test-sets

In [0]:

```
from sklearn.metrics import roc_curve, auc


model_tfidf = RandomForestClassifier(n_estimators=200, max_depth=7,class_weight="balanced")
model_tfidf.fit(X_train_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 = []
for k in range(0,X_train_tfidf_w2v.shape[0],100):
  y_train_pred.extend(model_tfidf.predict_proba(X_train_tfidf_w2v[k:k+100])[:,1])

y_test_pred = []


for k in range(0, X_test_tfidf_w2v.shape[0],100):
  y_test_pred.extend(model_tfidf.predict_proba(X_test_tfidf_w2v[k:k+100])[:,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("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)
```




```
====================================================================================================
```

In [0]:

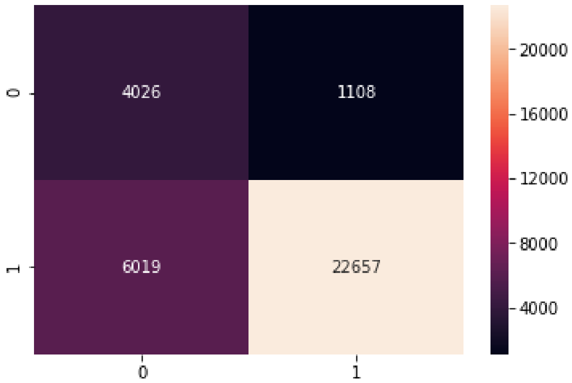
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):
```

```
    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

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

import seaborn as sns
from sklearn.metrics import confusion_matrix
print("train Confusion Matrix")
cm_train=confusion_matrix(y_train,predict( y_train_pred, tr_thresholds, train_fpr, train_tpr))
sns.heatmap(cm_train,annot=True,fmt='.5g')
```

```
train Confusion Matrix
the maximum value of tpr*(1-fpr) 0.6195862042470331 for threshold 0.508
```

Out[0]:

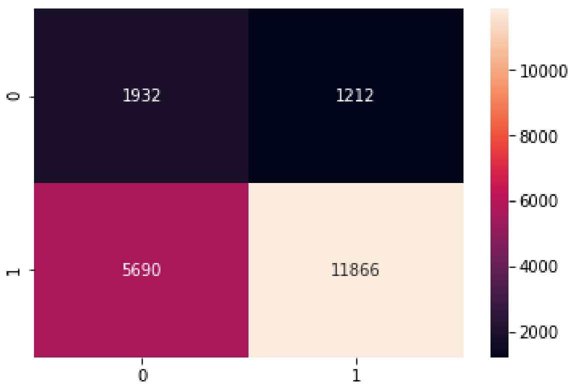
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58bbfb93c8>
```



In [0]:

```
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict( y_test_pred, te_thresholds, test_fpr, test_tpr))
print(sns.heatmap(cm,annot=True,fmt='.5g'))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4153396155204111 for threshold 0.506
AxesSubplot(0.125,0.125;0.62x0.755)
```



Gradient-Boosting TFIDF

In [0]:

```
train_auc = []
cv_auc = []

estimator_values = [5, 10, 50, 100, 200, 500]
depth_values = [2, 3, 4, 5, 6, 7, 8, 9, 10]

for j in tqdm(estimator_values):

  for i in depth_values:
      model_tfidf = XGBClassifier(n_estimators=j, max_depth=i,class_weight="balanced")
      model_tfidf.fit(X_train_tfidf_w2v, y_train)
      # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the po
sitive class
      # not the predicted outputs


      y_train_pred = []
      for k in range(0,X_train_tfidf_w2v.shape[0],100):
          y_train_pred.extend(model_tfidf.predict_proba(X_train_tfidf_w2v[k:k+100])[:,1])

      y_cv_pred = []


      for k in range(0, X_cv_tfidf_w2v.shape[0],100):
          y_cv_pred.extend(model_tfidf.predict_proba(X_cv_tfidf_w2v[k:k+100])[:,1])



      train_auc.append(roc_auc_score(y_train,y_train_pred))
      cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
100%|██████████| 6/6 [10:54:15<00:00, 9602.25s/it]
```

In [0]:

```
cv_auc_t = np.array(cv_auc).reshape(6,9).T
train_auc_t = np.array(train_auc).reshape(6,9).T

df_cv = pd.DataFrame(cv_auc_t,columns=estimator_values)
df_cv.index = depth_values
df_cv

df_train = pd.DataFrame(train_auc_t,columns=estimator_values)
df_train.index = depth_values
df_train
```

Out[0]:

	5	10	50	100	200	500
2	0.665593	0.688344	0.736403	0.763024	0.791598	0.843429
3	0.688685	0.705789	0.767901	0.803601	0.850719	0.932028
4	0.707175	0.732313	0.807504	0.858504	0.918022	0.987393
5	0.737923	0.768103	0.862096	0.919615	0.975259	0.999739
6	0.772526	0.805285	0.920523	0.971176	0.997419	1.000000
7	0.806921	0.850579	0.964018	0.995126	0.999976	1.000000
8	0.841389	0.901173	0.990963	0.999719	1.000000	1.000000
9	0.878628	0.934592	0.999080	0.999999	1.000000	1.000000
10	0.912496	0.961070	0.999977	1.000000	1.000000	1.000000

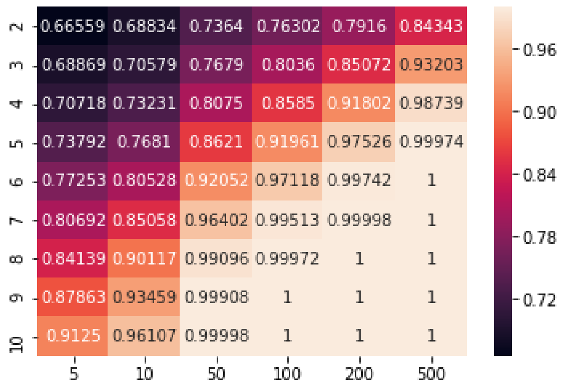
heat_map- Train_AUC

In [0]:

```
import seaborn as sns
sns.heatmap(df_train,annot=True,fmt='.5g')
```

Out[0]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7822c299e8>



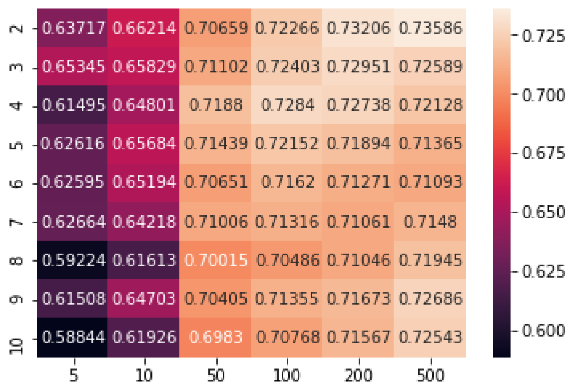
Heat_map - CV_AUC

In [0]:

```
import seaborn as sns
sns.heatmap(df_cv,annot=True,fmt='.5g')
```

Out[0]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7822a85550>



best_depth = 5

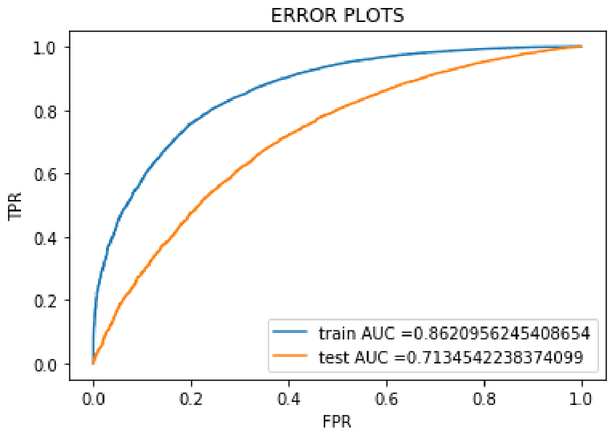
best_n_estimators = 50

ROC-Curve with optimal_k for train and test-sets

In [0]:

```
from sklearn.metrics import roc_curve, auc


model_tfidf = XGBClassifier(n_estimators=50, max_depth=5,class_weight="balanced")
model_tfidf.fit(X_train_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 = []
for k in range(0,X_train_tfidf_w2v.shape[0],1000):
```

```
for k in range(0,X_train_tfidf_w2v.shape[0],1000):
  y_train_pred.extend(model_tfidf.predict_proba(X_train_tfidf_w2v[k:k+1000])[:,1])

y_test_pred = []


for k in range(0, X_test_tfidf_w2v.shape[0],1000):
  y_test_pred.extend(model_tfidf.predict_proba(X_test_tfidf_w2v[k:k+1000])[:,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("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)
```




```
====================================================================================================
```

In [0]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

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

import seaborn as sns
from sklearn.metrics import confusion_matrix
print("train Confusion Matrix")
cm_train=confusion_matrix(y_train,predict( y_train_pred, tr_thresholds, train_fpr, train_tpr))
sns.heatmap(cm_train,annot=True,fmt='.5g')
```

In [0]:

```
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict( y_test_pred, te_thresholds, test_fpr, test_tpr))
print(sns.heatmap(cm,annot=True,fmt='.5g'))
```

AVG_W2V

In [54]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essays_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_essay): # 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) and (word in tfidf_words_train):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essays_vectors_train.append(vector)

print(len(avg_w2v_essays_vectors_train))
print(len(avg_w2v_essays_vectors_train[0]))
```

```
100%|██████████| 33810/33810 [00:09<00:00, 3546.29it/s]
```

```
33810
300
```

In [55]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_titles_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_train_titles):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_vectors_train.append(vector)

print(len(avg_w2v_titles_vectors_train))
print(len(avg_w2v_titles_vectors_train[0]))
```

```
100%|██████████| 33810/33810 [00:00<00:00, 64001.91it/s]
```

```
33810
300
```

In [56]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essays_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_essay): # 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) and (word in tfidf_words_train):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essays_vectors_test.append(vector)

print(len(avg_w2v_essays_vectors_test))
print(len(avg_w2v_essays_vectors_test[0]))
```

```
100%|██████████| 20700/20700 [00:06<00:00, 3096.52it/s]

20700
300
```

In [57]:

```python
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_titles_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_train_titles):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_vectors_test.append(vector)

print(len(avg_w2v_titles_vectors_test))
print(len(avg_w2v_titles_vectors_test[0]))
```

```
100%|██████████| 20700/20700 [00:00<00:00, 55067.66it/s]

20700
300
```

In [58]:

```python
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essays_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_essay): # 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) and (word in tfidf_words_train):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essays_vectors_cv.append(vector)

print(len(avg_w2v_essays_vectors_cv))
print(len(avg_w2v_essays_vectors_cv[0]))
```

```
100%|██████████| 14490/14490 [00:04<00:00, 3177.93it/s]

14490
300
```

In [59]:

```python
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_titles_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_train_titles):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_titles_vectors_cv.append(vector)

print(len(avg_w2v_titles_vectors_cv))
```

```
print(len(avg_w2v_titles_vectors_cv[0]))
```

```
100%|██████████| 14490/14490 [00:00<00:00, 63278.58it/s]
```

```
14490
300
```

In [60]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_summary_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train_summary): # 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) and (word in tfidf_words_train_summary):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_summary_vectors_train.append(vector)

print(len(avg_w2v_summary_vectors_train))
print(len(avg_w2v_summary_vectors_train[0]))
```

```
100%|██████████| 33810/33810 [00:01<00:00, 31008.80it/s]
```

```
33810
300
```

In [61]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_summary_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_summary): # 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) and (word in tfidf_words_train_summary):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_summary_vectors_test.append(vector)

print(len(avg_w2v_summary_vectors_test))
print(len(avg_w2v_summary_vectors_test[0]))
```

```
100%|██████████| 20700/20700 [00:00<00:00, 31184.44it/s]
```

```
20700
300
```

In [62]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_summary_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv_summary): # 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) and (word in tfidf_words_train_summary):
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
```

```
    avg_w2v_summary_vectors_cv.append(vector)

print(len(avg_w2v_summary_vectors_cv))
print(len(avg_w2v_summary_vectors_cv[0]))
```

```
100%|██████████| 14490/14490 [00:00<00:00, 31435.77it/s]
```

```
14490
300
```

Conacatenating all features Numerical and Categorical

In [0]:

```
from scipy.sparse import hstack
from sklearn.preprocessing import StandardScaler

X_train_avg_w2v = hstack((X_train_cn,avg_w2v_essays_vectors_train,avg_w2v_titles_vectors_train,avg
_w2v_summary_vectors_train))
X_train_avg_w2v = X_train_avg_w2v.tocsr()
train_scalar = StandardScaler(with_mean = False)
X_train_avg_w2v = train_scalar.fit_transform(X_train_avg_w2v)
```

In [0]:

```
X_cv_avg_w2v=
hstack((X_cv_cn,avg_w2v_essays_vectors_cv,avg_w2v_titles_vectors_cv,avg_w2v_summary_vectors_cv))
X_cv_avg_w2v  = X_cv_avg_w2v.tocsr()
cv_scalar = StandardScaler(with_mean = False)
X_cv_avg_w2v = cv_scalar.fit_transform(X_cv_avg_w2v)
```

In [0]:

```
X_test_avg_w2v
=hstack((X_test_cn,avg_w2v_essays_vectors_test,avg_w2v_titles_vectors_test,avg_w2v_summary_vectors_
test))
X_test_avg_w2v = X_test_avg_w2v.tocsr()
test_scalar = StandardScaler(with_mean = False)
X_test_avg_w2v = test_scalar.fit_transform(X_test_avg_w2v)
```

Random_Forest

In [0]:

```
train_auc = []
cv_auc = []


estimator_values = [5, 10, 50, 100, 200, 500]
depth_values = [2, 3, 4, 5, 6, 7, 8, 9, 10]

for j in tqdm(estimator_values):

  for i in depth_values:
      model_avg_w2v = RandomForestClassifier(n_estimators=j, max_depth=i,class_weight="balanced")
      model_avg_w2v.fit(X_train_avg_w2v, y_train)
      # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the po
sitive class
      # not the predicted outputs


      y_train_pred = []
      for k in range(0,X_train_avg_w2v.shape[0],100):
          y_train_pred.extend(model_avg_w2v.predict_proba(X_train_avg_w2v[k:k+100])[:,1])

      y_cv_pred = []
```

```
    for k in range(0, X_cv_avg_w2v.shape[0],100):
        y_cv_pred.extend(model_avg_w2v.predict_proba(X_cv_avg_w2v[k:k+100])[:,1])



    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
100%|██████████| 6/6 [2:11:48<00:00, 1943.38s/it]
```

In [0]:

```
cv_auc_t = np.array(cv_auc).reshape(6,9).T
train_auc_t = np.array(train_auc).reshape(6,9).T

df_cv = pd.DataFrame(cv_auc_t,columns=estimator_values)
df_cv.index = depth_values
df_cv

df_train = pd.DataFrame(train_auc_t,columns=estimator_values)
df_train.index = depth_values
df_train
```

Out[0]:

	5	10	50	100	200	500
2	0.640799	0.659519	0.704403	0.708510	0.707882	0.715399
3	0.654874	0.685531	0.717716	0.729175	0.727148	0.731157
4	0.692136	0.707574	0.740910	0.744855	0.751770	0.755705
5	0.709447	0.725132	0.776641	0.784486	0.786348	0.789141
6	0.739988	0.768470	0.819318	0.828963	0.829643	0.833225
7	0.774656	0.811349	0.873764	0.879951	0.884853	0.889067
8	0.814683	0.869703	0.921009	0.933837	0.935840	0.939026
9	0.856077	0.905770	0.961856	0.967591	0.973332	0.974071
10	0.884267	0.942982	0.984851	0.989184	0.991714	0.992284

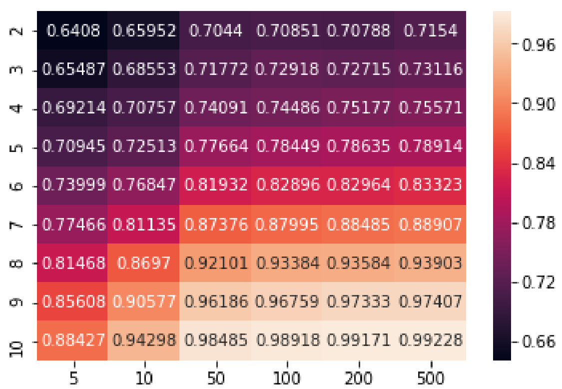
Heat_Map-Train_AUC

In [0]:

```
import seaborn as sns
sns.heatmap(df_train,annot=True,fmt='.5g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58bb811e48>
```



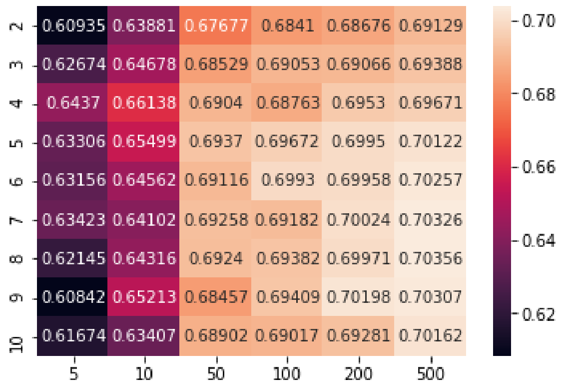
Heat_Map-CV_AUC

In [0]:

```
import seaborn as sns
sns.heatmap(df_cv,annot=True,fmt='.5g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58bc1af9b0>
```



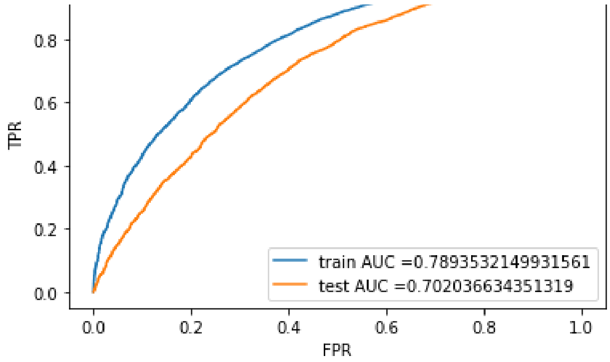
max_depth=5

n_estimators = 60

ROC-Curve with optimal_k for train and test-sets

In [0]:

```
from sklearn.metrics import roc_curve, auc


model_tfidf = RandomForestClassifier(n_estimators=500, max_depth=5,class_weight="balanced")
model_tfidf.fit(X_train_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 = []
for k in range(0,X_train_avg_w2v.shape[0],100):
  y_train_pred.extend(model_tfidf.predict_proba(X_train_avg_w2v[k:k+100])[:,1])

y_test_pred = []


for k in range(0, X_test_avg_w2v.shape[0],100):
  y_test_pred.extend(model_tfidf.predict_proba(X_test_avg_w2v[k:k+100])[:,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("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)
```




```
==================================================================================================
```

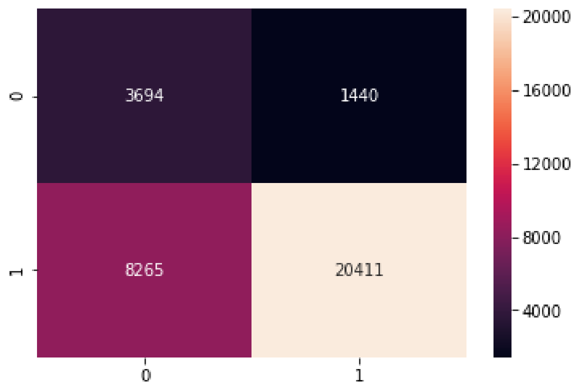
In [0]:

```python
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

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

import seaborn as sns
from sklearn.metrics import confusion_matrix
print("train Confusion Matrix")
cm_train=confusion_matrix(y_train,predict( y_train_pred, tr_thresholds, train_fpr, train_tpr))
sns.heatmap(cm_train,annot=True,fmt='.5g')
```

```
train Confusion Matrix
the maximum value of tpr*(1-fpr) 0.5121376894186288 for threshold 0.507
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58bc2fc2b0>
```

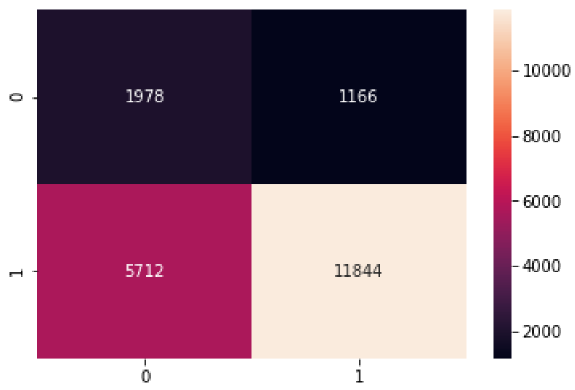


In [0]:

```python
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict( y_test_pred, te_thresholds, test_fpr, test_tpr))
print(sns.heatmap(cm,annot=True,fmt='.5g'))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.424440264436246 7 for threshold 0.493
```

```
the maximum value of tpr*(1-fpr) 0.4244402644362467 for threshold 0.495
AxesSubplot(0.125,0.125;0.62x0.755)
```



Gradient-Boosting

In [66]:

```python
train_auc = []
cv_auc = []


estimator_values = [5, 10, 50, 100, 200, 500]
depth_values = [2, 3, 4, 5, 6, 7, 8, 9, 10]

for j in tqdm(estimator_values):

  for i in depth_values:
      model_avg_w2v = XGBClassifier(n_estimators=j, max_depth=i,class_weight="balanced")
      model_avg_w2v.fit(X_train_avg_w2v, y_train)
      # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the po
sitive class
      # not the predicted outputs


      y_train_pred = []
      for k in range(0,X_train_avg_w2v.shape[0],1000):
          y_train_pred.extend(model_avg_w2v.predict_proba(X_train_avg_w2v[k:k+1000])[:,1])

      y_cv_pred = []


      for k in range(0, X_cv_avg_w2v.shape[0],1000):
          y_cv_pred.extend(model_avg_w2v.predict_proba(X_cv_avg_w2v[k:k+1000])[:,1])



      train_auc.append(roc_auc_score(y_train,y_train_pred))
      cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
100%|██████████| 6/6 [10:49:54<00:00, 9528.70s/it]
```

In [67]:

```python
cv_auc_t = np.array(cv_auc).reshape(6,9).T
train_auc_t = np.array(train_auc).reshape(6,9).T

df_cv = pd.DataFrame(cv_auc_t,columns=estimator_values)
df_cv.index = depth_values
df_cv

df_train = pd.DataFrame(train_auc_t,columns=estimator_values)
df_train.index = depth_values
df_train
```

Out[67]:

Out[67]:

	5	10	50	100	200	500
2	0.663820	0.676569	0.736907	0.764656	0.792420	0.846509
3	0.688277	0.699680	0.770801	0.806157	0.852741	0.930712
4	0.712439	0.731050	0.810805	0.858244	0.920996	0.986819
5	0.738667	0.771790	0.862357	0.921593	0.975786	0.999695
6	0.767304	0.810351	0.918602	0.969903	0.996596	1.000000
7	0.810475	0.857261	0.968209	0.994495	0.999945	1.000000
8	0.847671	0.900832	0.991348	0.999641	1.000000	1.000000
9	0.888586	0.939171	0.999323	0.999997	1.000000	1.000000
10	0.910258	0.962730	0.999949	1.000000	1.000000	1.000000

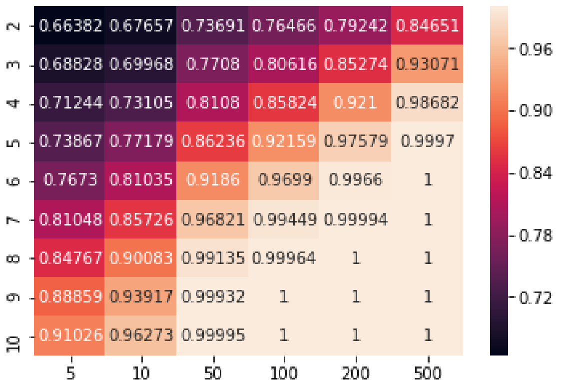
Heat_Map-train _AUC

In [68]:

```
import seaborn as sns
sns.heatmap(df_train,annot=True,fmt='.5g')
```

Out[68]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f22ecb460f0>



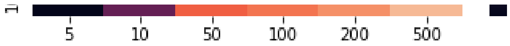
Heat_Map-CV_AUC

In [69]:

```
import seaborn as sns
sns.heatmap(df_cv,annot=True,fmt='.5g')
```

Out[69]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f22ecb62d68>



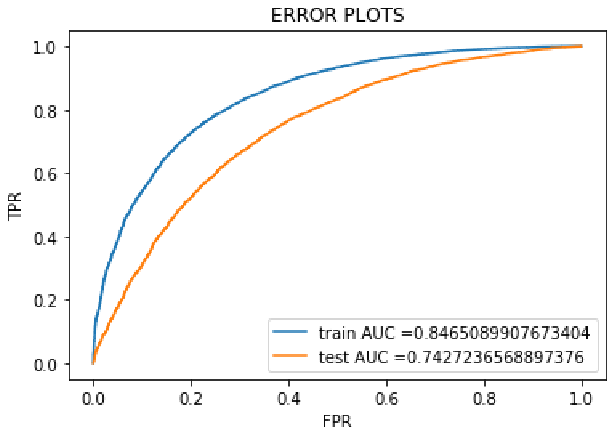
max_depth=5

n_estimators = 55

ROC-Curve with optimal_k for train and test-sets

In [70]:

```
from sklearn.metrics import roc_curve, auc


model_tfidf = XGBClassifier(n_estimators=500, max_depth=2,class_weight="balanced")
model_tfidf.fit(X_train_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 = []
for k in range(0,X_train_avg_w2v.shape[0],100):
  y_train_pred.extend(model_tfidf.predict_proba(X_train_avg_w2v[k:k+100])[:,1])

y_test_pred = []


for k in range(0, X_test_avg_w2v.shape[0],100):
  y_test_pred.extend(model_tfidf.predict_proba(X_test_avg_w2v[k:k+100])[:,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("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)
```




```
====================================================================================================
```

In [71]:

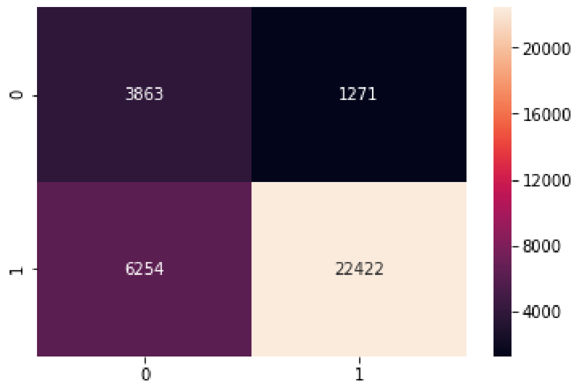
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
```

```
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

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

import seaborn as sns
from sklearn.metrics import confusion_matrix
print("train Confusion Matrix")
cm_train=confusion_matrix(y_train,predict( y_train_pred, tr_thresholds, train_fpr, train_tpr))
sns.heatmap(cm_train,annot=True,fmt='.5g')
```

```
train Confusion Matrix
the maximum value of tpr*(1-fpr) 0.5883349119860578 for threshold 0.824
```

Out[71]:

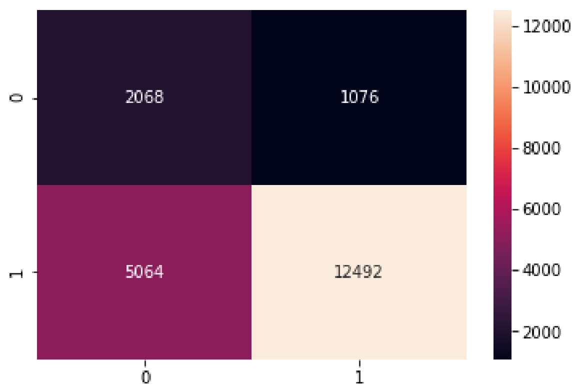
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f22ecbfb208>
```



In [72]:

```
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict( y_test_pred, te_thresholds, test_fpr, test_tpr))
print(sns.heatmap(cm,annot=True,fmt='.5g'))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4680307639327326 for threshold 0.732
AxesSubplot(0.125,0.125;0.62x0.755)
```



In [74]:

```
import numpy as np
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "max_depth","n_estimators", "Train-AUC","Test-AUC"]
```

```
x.add_row(["BOW-Random-Forest",5,1000,0.780,0.712])
x.add_row(["BOW-Gradient-Boosting",2,1000,0.822,0.750])
x.add_row(["TFidf-Random-Forest",10,200,0.879,0.710])
x.add_row(["TFidf-Gradient-Boosting",2,500,0.846,0.749])
x.add_row(["TFidf-W2V-Random-Forest",7,200,0.773,0.684])
x.add_row(["TFidf-W2V-Gradient-Boosting",5,50,0.862,0.713])
x.add_row(["AVG-W2V-Random-Forest",5,500,0.789,0.702])
x.add_row(["AVG-W2V-Gradient-Boosting",2,500,0.846,0.742])


print(x)
```

```
+-----------------------------+-----------+--------------+-----------+----------+
|            Model            | max_depth | n_estimators | Train-AUC | Test-AUC |
+-----------------------------+-----------+--------------+-----------+----------+
|      BOW-Random-Forest      |     5     |     1000     |    0.78   |  0.712   |
|    BOW-Gradient-Boosting    |     2     |     1000     |   0.822   |   0.75   |
|     TFidf-Random-Forest     |     10    |     200      |   0.879   |   0.71   |
|   TFidf-Gradient-Boosting   |     2     |     500      |   0.846   |  0.749   |
|   TFidf-W2V-Random-Forest   |     7     |     200      |   0.773   |  0.684   |
| TFidf-W2V-Gradient-Boosting |     5     |      50      |   0.862   |  0.713   |
|    AVG-W2V-Random-Forest    |     5     |     500      |   0.789   |  0.702   |
|  AVG-W2V-Gradient-Boosting  |     2     |     500      |   0.846   |  0.742   |
+-----------------------------+-----------+--------------+-----------+----------+
```