DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

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

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

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

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

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

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

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

Notes on the Essay Data

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

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_3:__ "Close by sharing why your project will make a difference"

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

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

In [222]:

```
%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

import time
from tqdm import tqdm
import os
import pickle

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

1.1 Reading Data

In [223]:

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

In [224]:

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

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

In [225]:

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


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


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


project_data.head(2)
```

Out[225]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	

In [226]:

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

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

# Join two data frames
project_data = pd.merge(project_data, price_data, on='id', how='left')
project_data.head(5)
```

```
Number of data points in train data (1541272, 4)
['id' 'description' 'quantity' 'price']
        id                                        description  quantity  \
0  p233245  LC652 - Lakeshore Double-Space Mobile Drying Rack         1
1  p069063        Bouncy Bands for Desks (Blue support pipes)         3

    price
0  149.00
1   14.95
```

Out[226]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_subje
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	M
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	
2	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Grades PreK-2	Litera
3	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	A
4	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	Litera

1.2 preprocessing of `project_subject_categories`

In [227]:

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

```
https://stackoverflow.com/a/47301924/4084039

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

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

from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

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

1.3 preprocessing of `project_subject_subcategories`

In [228]:

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

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

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

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

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

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

1.3 Text preprocessing

1.3 Text preprocessing

In [229]:

```
# 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 [230]:

```
project_data.head(2)
```

Out[230]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_title
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Engineering STEAM into the Primary Classroom
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Sensory Tools for Focus

In [231]:

```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [232]:

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

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classroom as well as the STEM j
ournals, which my students really enjoyed.  I would love to implement more of the Lakeshore STEM k
its in my classroom for the next school year as they provide excellent and engaging STEM
lessons.My students come from a variety of backgrounds, including language and socioeconomic statu
s.  Many of them don't have a lot of experience in science and engineering and these kits give me
the materials to provide these exciting opportunities for my students.Each month I try to do
several science or STEM/STEAM projects.  I would use the kits and robot to help guide my science i
nstruction in engaging and meaningful ways.  I can adapt the kits to my current language arts paci
ng guide where we already teach some of the material in the kits like tall tales (Paul Bunyan) or
Johnny Appleseed.  The following units will be taught in the next school year where I will
implement these kits: magnets, motion, sink vs. float, robots.  I often get to these units and don
't know If I am teaching the right way or using the right materials.    The kits will give me
additional ideas, strategies, and lessons to prepare my students in science.It is challenging to d
evelop high quality science activities.  These kits give me the materials I need to provide my
students with science activities that will go along with the curriculum in my classroom.  Although
I have some things (like magnets) in my classroom, I don't know how to use them effectively.  The
kits will provide me with the right amount of materials and show me how to use them in an
appropriate way.
==================================================
I teach high school English to students with learning and behavioral disabilities. My students all
vary in their ability level. However, the ultimate goal is to increase all students literacy level
s. This includes their reading, writing, and communication levels.I teach a really dynamic group o
f students. However, my students face a lot of challenges. My students all live in poverty and in
a dangerous neighborhood. Despite these challenges, I have students who have the the desire to def
```

a dangerous neighborhood. Despite these challenges, I have students who have the the desire to def
eat these challenges. My students all have learning disabilities and currently all are performing
below grade level. My students are visual learners and will benefit from a classroom that fulfills
their preferred learning style.The materials I am requesting will allow my students to be prepared
for the classroom with the necessary supplies. Too often I am challenged with students who come t
o school unprepared for class due to economic challenges. I want my students to be able to focus
on learning and not how they will be able to get school supplies. The supplies will last all year
. Students will be able to complete written assignments and maintain a classroom journal. The ch
art paper will be used to make learning more visual in class and to create posters to aid students
in their learning. The students have access to a classroom printer. The toner will be used to pr
int student work that is completed on the classroom Chromebooks.I want to try and remove all barri
ers for the students learning and create opportunities for learning. One of the biggest barriers i
s the students not having the resources to get pens, paper, and folders. My students will be able
to increase their literacy skills because of this project.

==

\"Life moves pretty fast. If you don't stop and look around once in awhile, you could miss it.\"
from the movie, Ferris Bueller's Day Off. Think back...what do you remember about your
grandparents? How amazing would it be to be able to flip through a book to see a day in their
lives?My second graders are voracious readers! They love to read both fiction and nonfiction books
. Their favorite characters include Pete the Cat, Fly Guy, Piggie and Elephant, and Mercy Watson.
They also love to read about insects, space and plants. My students are hungry bookworms! My stude
nts are eager to learn and read about the world around them. My kids love to be at school and are
like little sponges absorbing everything around them. Their parents work long hours and usually do
not see their children. My students are usually cared for by their grandparents or a family
friend. Most of my students do not have someone who speaks English at home. Thus it is difficult f
or my students to acquire language.Now think forward... wouldn't it mean a lot to your kids,
nieces or nephews or grandchildren, to be able to see a day in your life today 30 years from now?
Memories are so precious to us and being able to share these memories with future generations will
be a rewarding experience. As part of our social studies curriculum, students will be learning ab
out changes over time. Students will be studying photos to learn about how their community has ch
anged over time. In particular, we will look at photos to study how the land, buildings,
clothing, and schools have changed over time. As a culminating activity, my students will capture
a slice of their history and preserve it through scrap booking. Key important events in their
young lives will be documented with the date, location, and names.   Students will be using photos
from home and from school to create their second grade memories.   Their scrap books will preserve
their unique stories for future generations to enjoy.Your donation to this project will provide my
second graders with an opportunity to learn about social studies in a fun and creative manner. Th
rough their scrapbooks, children will share their story with others and have a historical document
for the rest of their lives.

==

\"A person's a person, no matter how small.\" (Dr.Seuss) I teach the smallest students with the bi
ggest enthusiasm for learning. My students learn in many different ways using all of our senses an
d multiple intelligences. I use a wide range of techniques to help all my students succeed. \r\nSt
udents in my class come from a variety of different backgrounds which makes for wonderful sharing
of experiences and cultures, including Native Americans.\r\nOur school is a caring community of su
ccessful learners which can be seen through collaborative student project based learning in and ou
t 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 wor
k cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is the
perfect place to learn about agriculture and nutrition. My students love to role play in our
pretend kitchen in the early childhood classroom. I have had several kids ask me, \"Can we try coo
king with REAL food?\" I will take their idea and create \"Common Core Cooking Lessons\" where we
learn important math and writing concepts while cooking delicious healthy food for snack time. My
students will have a grounded appreciation for the work that went into making the food and knowled
ge of where the ingredients came from as well as how it's healthy for their bodies. This project w
ould expand our learning of nutrition and agricultural cooking recipes by having us peel our own a
pples to make homemade applesauce, make our own bread, and mix up healthy plants from our classroo
m garden in the spring. We will also create our own cookbooks to be printed and shared with famili
es. \r\nStudents will gain math and literature skills as well as a life long enjoyment for healthy
cooking.nannan

==

My classroom consists of twenty-two amazing sixth graders from different cultures and backgrounds.
They are a social bunch who enjoy working in partners and working with groups. They are hard-worki
ng and eager to head to middle school next year. My job is to get them ready to make this
transition and make it as smooth as possible. In order to do this, my students need to come to
school every day and feel safe and ready to learn. Because they are getting ready to head to
middle school, I give them lots of choice- choice on where to sit and work, the order to complete
assignments, choice of projects, etc. Part of the students feeling safe is the ability for them to
come into a welcoming, encouraging environment. My room is colorful and the atmosphere is casual.
I want them to take ownership of the classroom because we ALL share it together. Because my time w
ith them is limited, I want to ensure they get the most of this time and enjoy it to the best of t
heir abilities.Currently, we have twenty-two desks of differing sizes, yet the desks are similar t
o the ones the students will use in middle school. We also have a kidney table with crates for sea
ting. I allow my students to choose their own spots while they are working independently or in
groups. More often than not, most of them move out of their desks and onto the crates. Believe it
or not, this has proven to be more successful than making them stay at their desks! It is because
of this that I am looking toward the “Flexible Seating” option for my classroom.\r\n The students
look forward to their work time so they can move around the room. I would like to get rid of the c

look forward to their work time so they can move around the room. I would like to get rid of the c
onstricting desks and move toward more “fun” seating options. I am requesting various seating so m
y students have more options to sit. Currently, I have a stool and a papasan chair I inherited fro
m the previous sixth-grade teacher as well as five milk crate seats I made, but I would like to gi
ve them more options and reduce the competition for the “good seats”. I am also requesting two rug
s as not only more seating options but to make the classroom more welcoming and appealing. In orde
r for my students to be able to write and complete work without desks, I am requesting a class set
of clipboards. Finally, due to curriculum that requires groups to work together, I am requesting t
ables that we can fold up when we are not using them to leave more room for our flexible seating o
ptions.\r\nI know that with more seating options, they will be that much more excited about coming
to school! Thank you for your support in making my classroom one students will remember
forever!nannan
==

In [233]:

```
# 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 [234]:

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

\"A person is a person, no matter how small.\" (Dr.Seuss) I teach the smallest students with the b
iggest enthusiasm for learning. My students learn in many different ways using all of our senses a
nd multiple intelligences. I use a wide range of techniques to help all my students succeed. \r\nS
tudents in my class come from a variety of different backgrounds which makes for wonderful sharing
of experiences and cultures, including Native Americans.\r\nOur school is a caring community of su
ccessful learners which can be seen through collaborative student project based learning in and ou
t 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 wor
k cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is the
perfect place to learn about agriculture and nutrition. My students love to role play in our
pretend kitchen in the early childhood classroom. I have had several kids ask me, \"Can we try coo
king with REAL food?\" I will take their idea and create \"Common Core Cooking Lessons\" where we
learn important math and writing concepts while cooking delicious healthy food for snack time. My
students will have a grounded appreciation for the work that went into making the food and knowled
ge of where the ingredients came from as well as how it is healthy for their bodies. This project
would expand our learning of nutrition and agricultural cooking recipes by having us peel our own
apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classro
om garden in the spring. We will also create our own cookbooks to be printed and shared with famil
ies. \r\nStudents will gain math and literature skills as well as a life long enjoyment for health
y cooking.nannan
==

In [235]:

```
# \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', ' ')
print(sent)
```

A person is a person, no matter how small. (Dr.Seuss) I teach the smallest students with the big
gest enthusiasm for learning. My students learn in many different ways using all of our senses and

```
multiple intelligences. I use a wide range of techniques to help all my students succeed.
Students in my class come from a variety of different backgrounds which makes for wonderful
sharing of experiences and cultures, including Native Americans.  Our school is a caring community
of successful learners which can be seen through collaborative student project based learning in a
nd 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 t
o work cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is
the perfect place to learn about agriculture and nutrition. My students love to role play in our p
retend kitchen in the early childhood classroom. I have had several kids ask me,  Can we try cooki
ng with REAL food?  I will take their idea and create  Common Core Cooking Lessons  where we learn
important math and writing concepts while cooking delicious healthy food for snack time. My
students will have a grounded appreciation for the work that went into making the food and knowled
ge of where the ingredients came from as well as how it is healthy for their bodies. This project
would expand our learning of nutrition and agricultural cooking recipes by having us peel our own
apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classro
om garden in the spring. We will also create our own cookbooks to be printed and shared with famil
ies.   Students will gain math and literature skills as well as a life long enjoyment for healthy
cooking.nannan
```

In [236]:

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

```
 A person is a person no matter how small Dr Seuss I teach the smallest students with the biggest
enthusiasm for learning My students learn in many different ways using all of our senses and multi
ple intelligences I use a wide range of techniques to help all my students succeed Students in my
class come from a variety of different backgrounds which makes for wonderful sharing of
experiences and cultures including Native Americans Our school is a caring community of successful
learners which can be seen through collaborative student project based learning in and out of the
classroom Kindergarteners in my class love to work with hands on materials and have many different
opportunities to practice a skill before it is mastered Having the social skills to work
cooperatively with friends is a crucial aspect of the kindergarten curriculum Montana is the
perfect place to learn about agriculture and nutrition My students love to role play in our
pretend kitchen in the early childhood classroom I have had several kids ask me Can we try cooking
with REAL food I will take their idea and create Common Core Cooking Lessons where we learn
important math and writing concepts while cooking delicious healthy food for snack time My
students will have a grounded appreciation for the work that went into making the food and knowled
ge of where the ingredients came from as well as how it is healthy for their bodies This project w
ould expand our learning of nutrition and agricultural cooking recipes by having us peel our own a
pples to make homemade applesauce make our own bread and mix up healthy plants from our classroom
garden in the spring We will also create our own cookbooks to be printed and shared with families
Students will gain math and literature skills as well as a life long enjoyment for healthy cooking
nannan
```

In [237]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
```

```
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [238]:

```python
# Create function that will filter sentance
def filterSentance(sentance):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = sent.lower()
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    return sent.strip()
```

In [239]:

```python
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    preprocessed_essays.append(filterSentance(sentance))
```

```
100%|██████████| 109248/109248 [01:09<00:00, 1576.94it/s]
```

In [240]:

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

Out[240]:

```
'person person no matter small dr seuss teach smallest students biggest enthusiasm learning
students learn many different ways using senses multiple intelligences use wide range techniques h
elp students succeed students class come variety different backgrounds makes wonderful sharing exp
eriences cultures including native americans school caring community successful learners seen coll
aborative student project based learning classroom kindergarteners class love work hands materials
many different opportunities practice skill mastered social skills work cooperatively friends cruc
ial aspect kindergarten curriculum montana perfect place learn agriculture nutrition students love
role play pretend kitchen early childhood classroom several kids ask try cooking real food take id
ea create common core cooking lessons learn important math writing concepts cooking delicious heal
thy food snack time students grounded appreciation work went making food knowledge ingredients cam
e well healthy bodies project would expand learning nutrition agricultural cooking recipes us peel
apples make homemade applesauce make bread mix healthy plants classroom garden spring also create
cookbooks printed shared families students gain math literature skills well life long enjoyment he
althy cooking nannan'
```

1.4 Preprocessing of `project_title`

In [241]:

```python
# similarly you can preprocess the titles also
# Combining all the above stundents
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_title'].values):
    preprocessed_titles.append(filterSentance(sentance))
```

```
100%|██████████| 109248/109248 [00:06<00:00, 18094.37it/s]
```

In [242]:

```python
# after preprocessing
```

```
print(preprocessed_titles[20000])
```

```
health nutritional cooking kindergarten
```

In [243]:

```
# similarly you can preprocess the project_resource_summary also
# Combining all the above stundents
from tqdm import tqdm
preprocessed_resource_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_resource_summary'].values):
    preprocessed_resource_summary.append(filterSentance(sentance))
```

```
100%|██████████| 109248/109248 [00:10<00:00, 10799.89it/s]
```

In [244]:

```
# after preprocessing
print(preprocessed_resource_summary[20000])
```

```
students need cooking supplies help us healthy learn nutrition mixer apple spiralizer kitchen
tools nutrition kit kid friendly healthy literature ink make cookbooks
```

In [245]:

```
# Preprocess teacher_prefix
from tqdm import tqdm
preprocessed_teacher_prefix = []
# tqdm is for printing the status bar
for teacher_prefix in tqdm(project_data['teacher_prefix'].values):
    teacher_prefix = str(teacher_prefix)
    clean_teacher_prefix = decontracted(teacher_prefix)
    clean_teacher_prefix = clean_teacher_prefix.replace('\\r', ' ')
    clean_teacher_prefix = clean_teacher_prefix.replace('\\"', ' ')
    clean_teacher_prefix = clean_teacher_prefix.replace('\\n', ' ')
    clean_teacher_prefix = re.sub('[^A-Za-z0-9]+', ' ', clean_teacher_prefix)
    clean_teacher_prefix = clean_teacher_prefix.lower()
    if clean_teacher_prefix in stopwords:
        continue
    preprocessed_teacher_prefix.append(clean_teacher_prefix.strip())
```

```
100%|██████████| 109248/109248 [00:02<00:00, 45904.99it/s]
```

In [246]:

```
preprocessed_teacher_prefix[0:10]
```

Out[246]:

```
['mrs', 'ms', 'mrs', 'mrs', 'mrs', 'mrs', 'mrs', 'ms', 'ms', 'mrs']
```

In [247]:

```
# Preprocess project_grade_category
from tqdm import tqdm
preprocessed_project_grade_category = []
# tqdm is for printing the status bar
for project_grade_category in tqdm(project_data['project_grade_category'].values):
    project_grade_category = str(project_grade_category)
    clean_project_grade_category = decontracted(project_grade_category)
    clean_project_grade_category = clean_project_grade_category.replace('\\r', ' ')
    clean_project_grade_category = clean_project_grade_category.replace('\\"', ' ')
    clean_project_grade_category = clean_project_grade_category.replace('\\n', ' ')
    clean_project_grade_category = re.sub('[^A-Za-z0-9]+', ' ', clean_project_grade_category)
    clean_project_grade_category = clean_project_grade_category.lower()
    if clean_project_grade_category in stopwords:
        continue
    clean_project_grade_category = clean_project_grade_category.strip()
#     whitespaces are creating problems because we are treating this as categorical feature
```

```
#     whitespace are creating problems because we are treating this as categorical feature
    preprocessed_project_grade_category.append(clean_project_grade_category.replace(' ', '_'))
```

```
100%|██████████| 109248/109248 [00:01<00:00, 58878.88it/s]
```

In [248]:

```
preprocessed_project_grade_category[0:10]
```

Out[248]:

```
['grades_prek_2',
 'grades_3_5',
 'grades_prek_2',
 'grades_prek_2',
 'grades_3_5',
 'grades_3_5',
 'grades_3_5',
 'grades_3_5',
 'grades_prek_2',
 'grades_3_5']
```

In [249]:

```
# Replace original columns with preprocessed column values
project_data['clean_essays'] = preprocessed_essays
project_data['clean_titles'] = preprocessed_titles
project_data['project_resource_summary'] = preprocessed_resource_summary
project_data['teacher_prefix'] = preprocessed_teacher_prefix
project_data['project_grade_category'] = preprocessed_project_grade_category
# Drop essays column
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [250]:

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

Out[250]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_title
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	mrs	CA	2016-04-27 00:27:36	grades_prek_2	Engineering STEAM into the Primary Classroom
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	ms	UT	2016-04-27 00:31:25	grades_3_5	Sensory Tools for Focus
2	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	mrs	CA	2016-04-27 00:46:53	grades_prek_2	Mobile Learning with a Mobile Listening Center
3	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	mrs	GA	2016-04-27 00:53:00	grades_prek_2	Flexible Seating for Flexible Learning
4	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	mrs	WA	2016-04-27 01:05:25	grades_3_5	Going Deep: The Art of Inner Thinking!

In [251]:

```
project_data.tail(5)
```

Out[251]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	projec
109243	45036	p194916	29cf137e5a40b0f141d9fd7898303a5c	mrs	HI	2017-04-30 23:11:45	grades_9_12	Na F Pro
109244	12610	p162971	22fee80f2078c694c2d244d3ecb1c390	ms	NM	2017-04-30 23:23:24	grades_prek_2	Op Organ
109245	179833	p096829	c8c81a73e29ae3bdd4140be8ad0bea00	mrs	IL	2017-04-30 23:25:42	grades_3_5	B Agri Sustai t
109246	13791	p184393	65545a295267ad9df99f26f25c978fd0	mrs	HI	2017-04-30 23:27:07	grades_9_12	M N
109247	124250	p028318	1fff5a88945be8b2c728c6a85c31930f	mrs	CA	2017-04-30 23:45:08	grades_prek_2	Ne

In [252]:

```
print(set(preprocessed_project_grade_category))
```

```
{'grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2'}
```

In [253]:

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')
```

In [254]:

```
project_data.head(2)
```

Out[254]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_title
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	mrs	CA	2016-04-27 00:27:36	grades_prek_2	Engineering STEAM into the Primary Classroom
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	ms	UT	2016-04-27 00:31:25	grades_3_5	Sensory Tools for Focus

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_title

1.5 Preparing data for models

In [255]:

```
project_data.columns
```

Out[255]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'quantity', 'price', 'clean_categories', 'clean_subcategories', 'essay',
       'clean_essays', 'clean_titles'],
      dtype='object')
```

we are going to consider

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

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

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

In [256]:

```
print(project_data.shape)

# I am taking 30% of data points for my analysis
# AVGW2V and TFIDFW2V takes more time
# I am using macbook pro with 16GB of RAM but I didn't get AVGW2V result within 12 hrs for 50% of data.
# So I switched to 30% of data
project_data = project_data.sample(frac=0.3)

print(project_data.shape)
```

```
(109248, 18)
(32774, 18)
```

In [257]:

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

Out[257]:

```
(32774, 17)
```

In [258]:

```
# Split Train, CV and Test data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)

print('Train Data Set', X_train.shape, y_train.shape)
print('Cross Validate Data Set', X_cv.shape, y_cv.shape)
print('Test Data Set', X_test.shape, y_test.shape)
```

```
Train Data Set (14711, 17) (14711,)
Cross Validate Data Set (7247, 17) (7247,)
Test Data Set (10816, 17) (10816,)
```

In [259]:

```
# Commented code as per your suggestion
# # Handle imblanced data set
# from imblearn.over_sampling import RandomOverSampler
# from collections import Counter

# ros = RandomOverSampler(sampling_strategy='minority', random_state=42)
# X_train, y_train = ros.fit_resample(X_train, y_train)
# print('Resampled Dataset Shape %s ' %Counter(y_train))

# X_train = pd.DataFrame(X_train, columns=X.columns)
# X_train.head(2)
```

In [260]:

```
print('Train Data Set', X_train.shape, y_train.shape)
print('Cross Validate Data Set', X_cv.shape, y_cv.shape)
print('Test Data Set', X_test.shape, y_test.shape)
print('*'*100)
```

```
Train Data Set (14711, 17) (14711,)
Cross Validate Data Set (7247, 17) (7247,)
Test Data Set (10816, 17) (10816,)
****************************************************************************************************
```

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

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [261]:

```
# # Sample code for bigram extraction using TFIDF
# from sklearn.feature_extraction.text import TfidfVectorizer
# corpus = [
#     'This is the first document.',
#     'This document is the second document.',
#     'And this is the third one.',
#     'Is this the first document?',
# ]
# vectorizer = TfidfVectorizer(ngram_range=(2,2))
# X = vectorizer.fit_transform(corpus)
# print(vectorizer.get_feature_names())

# print(X.shape)
```

In [262]:

```
# - project_title : text data
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("*"*100)
```

```python
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['clean_titles'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['clean_titles'].values)
X_cv_title_bow = vectorizer.transform(X_cv['clean_titles'].values)
X_test_title_bow = vectorizer.transform(X_test['clean_titles'].values)

clean_titles_bow_features = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
# print(vectorizer.get_feature_names())
print("*"*100)
```

```
(14711, 17) (14711,)
(7247, 17) (7247,)
(10816, 17) (10816,)
****************************************************************************************************

After vectorizations
(14711, 1130) (14711,)
(7247, 1130) (7247,)
(10816, 1130) (10816,)
****************************************************************************************************
```

In [263]:

```python
# - text : text data
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("*"*100)

# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['clean_essays'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['clean_essays'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['clean_essays'].values)
X_test_essay_bow = vectorizer.transform(X_test['clean_essays'].values)

easy_bow_features = vectorizer.get_feature_names()

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

```
(14711, 17) (14711,)
(7247, 17) (7247,)
(10816, 17) (10816,)
****************************************************************************************************

After vectorizations
(14711, 5000) (14711,)
(7247, 5000) (7247,)
(10816, 5000) (10816,)
****************************************************************************************************
```

In [264]:

```python
# - project_resource_summary: text data (optinal)
print(X_train.shape, y_train.shape)
```

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

print("*"*100)

# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_resource_summary'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_resource_summary_bow = vectorizer.transform(X_train['project_resource_summary'].va
lues)
X_cv_project_resource_summary_bow = vectorizer.transform(X_cv['project_resource_summary'].values)
X_test_project_resource_summary_bow =
vectorizer.transform(X_test['project_resource_summary'].values)

project_resource_summary_bow_features = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_project_resource_summary_bow.shape, y_train.shape)
print(X_cv_project_resource_summary_bow.shape, y_cv.shape)
print(X_test_project_resource_summary_bow.shape, y_test.shape)
# print(vectorizer.get_feature_names())
print("*"*100)
```

```
(14711, 17) (14711,)
(7247, 17) (7247,)
(10816, 17) (10816,)
****************************************************************************************************

After vectorizations
(14711, 4134) (14711,)
(7247, 4134) (7247,)
(10816, 4134) (10816,)
****************************************************************************************************
```

1.5.2.2 TFIDF vectorizer

In [265]:

```
# - project_title : text data
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("*"*100)

from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['clean_titles'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_tfidf = vectorizer.transform(X_train['clean_titles'].values)
X_cv_title_tfidf = vectorizer.transform(X_cv['clean_titles'].values)
X_test_title_tfidf = vectorizer.transform(X_test['clean_titles'].values)

clean_titles_tfidf_features = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_cv_title_tfidf.shape, y_test.shape)
print("*"*100)
```

```
(14711, 17) (14711,)
(7247, 17) (7247,)
(10816, 17) (10816,)
****************************************************************************************************

After vectorizations
(14711, 849) (14711,)
(7247, 849) (7247,)
```

```
(7247, 849) (7247,)
(7247, 849) (10816,)
****************************************************************************************************
```

In [266]:

```
# - text : text data
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("*"*100)

from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = TfidfVectorizer(min_df=10)#, ngram_range=(2,2), max_features=5000
vectorizer.fit(X_train['clean_essays'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['clean_essays'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['clean_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['clean_essays'].values)

easy_tfidf_features = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("*"*100)
```

```
(14711, 17) (14711,)
(7247, 17) (7247,)
(10816, 17) (10816,)
****************************************************************************************************

After vectorizations
(14711, 7337) (14711,)
(7247, 7337) (7247,)
(10816, 7337) (10816,)
****************************************************************************************************
```

In [267]:

```
# - project_resource_summary: text data (optinal)
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("*"*100)

from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['project_resource_summary'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_resource_summary_tfidf = vectorizer.transform(X_train['project_resource_summary'].
values)
X_cv_project_resource_summary_tfidf = vectorizer.transform(X_cv['project_resource_summary'].values
)
X_test_project_resource_summary_tfidf =
vectorizer.transform(X_test['project_resource_summary'].values)

project_resource_summary_tfidf_features = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_project_resource_summary_tfidf.shape, y_train.shape)
print(X_cv_project_resource_summary_tfidf.shape, y_cv.shape)
print(X_test_project_resource_summary_tfidf.shape, y_test.shape)
print("*"*100)
```

```
(14711, 17) (14711,)
(7247, 17) (7247,)
(10816, 17) (10816,)
****************************************************************************************************

After vectorizations
(14711, 1872) (14711,)
(7247, 1872) (7247,)
(10816, 1872) (10816,)
****************************************************************************************************
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [268]:

```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

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

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

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

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

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

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

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


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

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


'''
```

Out[268]:

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

```
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
============================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus",
len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",      len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\n
print("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [269]:

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

In [270]:

```python
# average Word2Vec for train text
# compute average word2vec for each review.
avg_w2v_vectors_text_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_text_train.append(vector)

print(len(avg_w2v_vectors_text_train))
print(len(avg_w2v_vectors_text_train[0]))
```

```
100%|██████████| 14711/14711 [00:05<00:00, 2782.36it/s]
```

```
14711
300
```

In [271]:

```python
# average Word2Vec for CV text
# compute average word2vec for each review.
avg_w2v_vectors_text_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_text_cv.append(vector)

print(len(avg_w2v_vectors_text_cv))
print(len(avg_w2v_vectors_text_cv[0]))
```

```
100%|██████████| 7247/7247 [00:02<00:00, 3336.93it/s]
```

```
7247
300
```

In [272]:

```
# average Word2Vec for test text
# compute average word2vec for each review.
avg_w2v_vectors_text_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_text_test.append(vector)

print(len(avg_w2v_vectors_text_test))
print(len(avg_w2v_vectors_text_test[0]))
```

```
100%|██████████| 10816/10816 [00:04<00:00, 2324.49it/s]
```

```
10816
300
```

In [273]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_title_train.append(vector)

print(len(avg_w2v_vectors_title_train))
print(len(avg_w2v_vectors_title_train[0]))
```

```
100%|██████████| 14711/14711 [00:00<00:00, 32653.38it/s]
```

```
14711
300
```

In [274]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
```

```
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_title_cv.append(vector)

print(len(avg_w2v_vectors_title_cv))
print(len(avg_w2v_vectors_title_cv[0]))
```

```
100%|██████████| 7247/7247 [00:00<00:00, 58864.66it/s]
```

```
7247
300
```

In [275]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_title_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_title_test.append(vector)

print(len(avg_w2v_vectors_title_test))
print(len(avg_w2v_vectors_title_test[0]))
```

```
100%|██████████| 10816/10816 [00:00<00:00, 38292.15it/s]
```

```
10816
300
```

In [276]:

```
# Similarly you can vectorize for project_resource_summary also
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_project_resource_summary_train = []; # the avg-w2v for each sentence/review is sto
red in this list
for sentence in tqdm(X_train['project_resource_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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_project_resource_summary_train.append(vector)

print(len(avg_w2v_vectors_project_resource_summary_train))
print(len(avg_w2v_vectors_project_resource_summary_train[0]))
```

```
100%|██████████| 14711/14711 [00:00<00:00, 26837.20it/s]
```

```
14711
300
```

In [277]:

```
# Similarly you can vectorize for project_resource_summary also
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_project_resource_summary_cv = []; # the avg-w2v for each sentence/review is stored
in this list
for sentence in tqdm(X_cv['project_resource_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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_project_resource_summary_cv.append(vector)

print(len(avg_w2v_vectors_project_resource_summary_cv))
print(len(avg_w2v_vectors_project_resource_summary_cv[0]))
```

```
100%|██████████| 7247/7247 [00:00<00:00, 10577.49it/s]
```

```
7247
300
```

In [278]:

```
# Similarly you can vectorize for project_resource_summary also
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_project_resource_summary_test = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sentence in tqdm(X_test['project_resource_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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_project_resource_summary_test.append(vector)

print(len(avg_w2v_vectors_project_resource_summary_test))
print(len(avg_w2v_vectors_project_resource_summary_test[0]))
```

```
100%|██████████| 10816/10816 [00:00<00:00, 11046.90it/s]
```

```
10816
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [279]:

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

In [280]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_text_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_essays']): # for each review/sentence
```

```
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_text_train.append(vector)

print(len(tfidf_w2v_vectors_text_train))
print(len(tfidf_w2v_vectors_text_train[0]))
```

```
100%|██████████| 14711/14711 [00:49<00:00, 447.19it/s]
```

```
14711
300
```

In [281]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_text_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_text_cv.append(vector)

print(len(tfidf_w2v_vectors_text_cv))
print(len(tfidf_w2v_vectors_text_cv[0]))
```

```
100%|██████████| 7247/7247 [00:17<00:00, 418.04it/s]
```

```
7247
300
```

In [282]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_text_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
```

```
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_text_test.append(vector)

print(len(tfidf_w2v_vectors_text_test))
print(len(tfidf_w2v_vectors_text_test[0]))
```

```
100%|██████████| 10816/10816 [00:27<00:00, 390.96it/s]
```

```
10816
300
```

In [283]:

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

In [284]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_title_train.append(vector)

print(len(tfidf_w2v_vectors_title_train))
print(len(tfidf_w2v_vectors_title_train[0]))
```

```
100%|██████████| 14711/14711 [00:00<00:00, 25966.36it/s]
```

```
14711
300
```

In [285]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
```

```
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_title_cv.append(vector)

print(len(tfidf_w2v_vectors_title_cv))
print(len(tfidf_w2v_vectors_title_cv[0]))
```

```
100%|██████████| 7247/7247 [00:00<00:00, 25684.01it/s]
```

```
7247
300
```

In [286]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_title_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_title_test.append(vector)

print(len(tfidf_w2v_vectors_title_test))
print(len(tfidf_w2v_vectors_title_test[0]))
```

```
100%|██████████| 10816/10816 [00:00<00:00, 30083.56it/s]
```

```
10816
300
```

In [287]:

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

In [288]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_project_resource_summary_train = []; # the avg-w2v for each sentence/review is s
tored in this list
for sentence in tqdm(X_train['project_resource_summary']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
```

```
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_project_resource_summary_train.append(vector)

print(len(tfidf_w2v_vectors_project_resource_summary_train))
print(len(tfidf_w2v_vectors_project_resource_summary_train[0]))
```

```
100%|██████████| 14711/14711 [00:01<00:00, 8482.62it/s]
```

```
14711
300
```

In [289]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_project_resource_summary_cv = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sentence in tqdm(X_cv['project_resource_summary']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_project_resource_summary_cv.append(vector)

print(len(tfidf_w2v_vectors_project_resource_summary_cv))
print(len(tfidf_w2v_vectors_project_resource_summary_cv[0]))
```

```
100%|██████████| 7247/7247 [00:00<00:00, 9948.22it/s]
```

```
7247
300
```

In [290]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_project_resource_summary_test = []; # the avg-w2v for each sentence/review is st
ored in this list
for sentence in tqdm(X_test['project_resource_summary']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_project_resource_summary_test.append(vector)
```

```
print(len(tfidf_w2v_vectors_project_resource_summary_test))
print(len(tfidf_w2v_vectors_project_resource_summary_test[0]))
```

```
100%|██████████| 10816/10816 [00:01<00:00, 7756.98it/s]
```

```
10816
300
```

1.5.3 Vectorizing Numerical features

In [291]:

```
# You no need to perform standardization/normalization on numerical data,
# because you will classify data by using gini impurity in decision tree classifier.
# - quantity : numerical (optional)

X_train_quantity_norm = X_train['quantity'].values.reshape(-1,1)
X_cv_quantity_norm = X_cv['quantity'].values.reshape(-1,1)
X_test_quantity_norm = X_test['quantity'].values.reshape(-1,1)

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_cv_quantity_norm.shape, y_cv.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(14711, 1) (14711,)
(7247, 1) (7247,)
(10816, 1) (10816,)
====================================================================================================
```

In [292]:

```
# You no need to perform standardization/normalization on numerical data,
# because you will classify data by using gini impurity in decision tree classifier.
# One hot encoding of numerical feature
# - teacher_number_of_previously_posted_projects : numerical
X_train_teacher_number_of_previously_posted_projects_norm =
X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)
X_cv_teacher_number_of_previously_posted_projects_norm =
X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)
X_test_teacher_number_of_previously_posted_projects_norm =
X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)

print("After vectorizations")
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.shape)
print(X_cv_teacher_number_of_previously_posted_projects_norm.shape, y_cv.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(14711, 1) (14711,)
(7247, 1) (7247,)
(10816, 1) (10816,)
====================================================================================================
```

In [293]:

```
# You no need to perform standardization/normalization on numerical data,
# because you will classify data by using gini impurity in decision tree classifier.
# - price : numerical

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

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

```
After vectorizations
(14711, 1) (14711,)
(7247, 1) (7247,)
(10816, 1) (10816,)
====================================================================================================
```

1.5.1 Vectorizing Categorical data

In [294]:

```
def calculate_proba_score_of_each_variable(data, classLabel):
    '''
    Function to return probability score
    Return Dict 'LA': {'pos': 0.8267790262172284, 'neg': 0.17322097378277154}
    '''
    positive_score_dict = {}
    negative_score_dict = {}
    distinct_feature = []
    # Collect negative and positive scores of a class label
    for i in range(len(data)):
        # Collect distinct features
        if(data[i] not in distinct_feature):
            distinct_feature.append(data[i])
        if(1 == classLabel[i]):
            if(data[i] in positive_score_dict):
                positive_score_dict[data[i]] = positive_score_dict[data[i]] + 1
            else:
                positive_score_dict[data[i]] = 1
        else:
            if(data[i] in negative_score_dict):
                negative_score_dict[data[i]] = negative_score_dict[data[i]] + 1
            else:
                negative_score_dict[data[i]] = 1

#     print(positive_score_dict, negative_score_dict)

    # Collect probability score
    prob_score = {}
    for i in range(len(distinct_feature)):
#         print(distinct_feature[i])
        pos_score = 0
        neg_score = 0
        if(distinct_feature[i] in positive_score_dict):
            pos_score = positive_score_dict[distinct_feature[i]]
        if(distinct_feature[i] in negative_score_dict):
            neg_score = negative_score_dict[distinct_feature[i]]

        prob_score[distinct_feature[i]] = {"pos" : (pos_score/(pos_score+neg_score)), "neg" : (neg_
score/(pos_score+neg_score))}
#         print(prob_score)
    return prob_score


def convert_response_encoding(data, proba_score):
    '''
    Convert feature into response encoding
    Return Lists
    '''
    result_pos = []
    result_neg = []
    for i in range(len(data)):
        if((data[i] in proba_score)):
            result_pos.append(proba_score[data[i]]['pos'])
            result_neg.append(proba_score[data[i]]['neg'])
        else:#handle missing data
            result_pos.append(0.5)
```

```
            result_neg.append(0.5)

    return [result_pos, result_neg]
```

In [295]:

```
# We are using response encoding instead of one hot encoding for categorical feature
from scipy.sparse import coo_matrix


# calculate response encoding on school_state train data i.e. fit data
proba_score_train = calculate_proba_score_of_each_variable(X_train['school_state'].values, y_train
)

# transform train data
response_encoding_train_pos, response_encoding_train_neg = convert_response_encoding(X_train['scho
ol_state'].values, proba_score_train)

response_encoding_train_pos = pd.DataFrame(response_encoding_train_pos)
response_encoding_train_neg = pd.DataFrame(response_encoding_train_neg)

# transform cv data
response_encoding_cv_pos, response_encoding_cv_neg = convert_response_encoding(X_cv['school_state'
].values, proba_score_train)

response_encoding_cv_pos = pd.DataFrame(response_encoding_cv_pos)
response_encoding_cv_neg = pd.DataFrame(response_encoding_cv_neg)

# transform test data
response_encoding_test_pos, response_encoding_test_neg =
convert_response_encoding(X_test['school_state'].values, proba_score_train)

response_encoding_test_pos = pd.DataFrame(response_encoding_test_pos)
response_encoding_test_neg = pd.DataFrame(response_encoding_test_neg)

# reshape data
X_train_school_state_pos = response_encoding_train_pos.values.reshape(-1,1)
X_train_school_state_neg = response_encoding_train_neg.values.reshape(-1,1)
X_cv_school_state_pos = response_encoding_cv_pos.values.reshape(-1,1)
X_cv_school_state_neg = response_encoding_cv_neg.values.reshape(-1,1)
X_test_school_state_pos = response_encoding_test_pos.values.reshape(-1,1)
X_test_school_state_neg = response_encoding_test_neg.values.reshape(-1,1)

# print data to do some verification
print(X_train_school_state_pos.shape, y_train.shape)
print(X_train_school_state_neg.shape, y_train.shape)
print(X_cv_school_state_pos.shape, y_cv.shape)
print(X_cv_school_state_neg.shape, y_cv.shape)
print(X_test_school_state_pos.shape, y_test.shape)
print(X_test_school_state_neg.shape, y_test.shape)
print('*'*100)
print('train data positive')
print(X_train_school_state_pos[0:5])
print('*'*100)
print('train data negative')
print(X_train_school_state_neg[0:5])
print('*'*100)
print('cv data positive')
print(X_cv_school_state_pos[0:5])
print('*'*100)
print('cv data negative')
print(X_cv_school_state_neg[0:5])
print('*'*100)
print('test data positive')
print(X_test_school_state_pos[0:5])
print('*'*100)
print('test data negative')
print(X_test_school_state_neg[0:5])
print('*'*100)


X_train_school_state_pos = coo_matrix(X_train_school_state_pos)
X_train_school_state_neg = coo_matrix(X_train_school_state_neg)
X_cv_school_state_pos = coo_matrix(X_cv_school_state_pos)
X_cv_school_state_neg = coo_matrix(X_cv_school_state_neg)
X_test_school_state_pos = coo_matrix(X_test_school_state_pos)
X_test_school_state_neg = coo_matrix(X_test_school_state_neg)
```

```
X_test_school_state_neg = coo_matrix(X_test_school_state_neg)

# print csr matrix shape
print(X_train_school_state_pos.shape, y_train.shape)
print(X_train_school_state_neg.shape, y_train.shape)
print(X_cv_school_state_pos.shape, y_cv.shape)
print(X_cv_school_state_neg.shape, y_cv.shape)
print(X_test_school_state_pos.shape, y_test.shape)
print(X_test_school_state_neg.shape, y_test.shape)
print('*'*100)
```

```
(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************

train data positive
[[0.84751773]
 [0.86019417]
 [0.84198113]
 [0.81569966]
 [0.84198113]]
****************************************************************************************************

train data negative
[[0.15248227]
 [0.13980583]
 [0.15801887]
 [0.18430034]
 [0.15801887]]
****************************************************************************************************

cv data positive
[[0.86019417]
 [0.87378641]
 [0.88663968]
 [0.86019417]
 [0.88059701]]
****************************************************************************************************

cv data negative
[[0.13980583]
 [0.12621359]
 [0.11336032]
 [0.13980583]
 [0.11940299]]
****************************************************************************************************

test data positive
[[0.81672026]
 [0.8531746 ]
 [0.84198113]
 [0.7997936 ]
 [0.86019417]]
****************************************************************************************************

test data negative
[[0.18327974]
 [0.1468254 ]
 [0.15801887]
 [0.2002064 ]
 [0.13980583]]
****************************************************************************************************

(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************
```

In [296]:

```
# We are using response encoding instead of one hot encoding for categorical feature
# - clean_categories : categorical data

# calculate response encoding on clean_categories train data i.e. fit data
proba_score_train = calculate_proba_score_of_each_variable(X_train['clean_categories'].values,
y_train)

# transform train data
response_encoding_train_pos, response_encoding_train_neg = convert_response_encoding(X_train['clea
n_categories'].values, proba_score_train)

response_encoding_train_pos = pd.DataFrame(response_encoding_train_pos)
response_encoding_train_neg = pd.DataFrame(response_encoding_train_neg)

# transform cv data
response_encoding_cv_pos, response_encoding_cv_neg =
convert_response_encoding(X_cv['clean_categories'].values, proba_score_train)

response_encoding_cv_pos = pd.DataFrame(response_encoding_cv_pos)
response_encoding_cv_neg = pd.DataFrame(response_encoding_cv_neg)

# transform test data
response_encoding_test_pos, response_encoding_test_neg =
convert_response_encoding(X_test['clean_categories'].values, proba_score_train)

response_encoding_test_pos = pd.DataFrame(response_encoding_test_pos)
response_encoding_test_neg = pd.DataFrame(response_encoding_test_neg)

# reshape data
X_train_clean_categories_pos = response_encoding_train_pos.values.reshape(-1,1)
X_train_clean_categories_neg = response_encoding_train_neg.values.reshape(-1,1)
X_cv_clean_categories_pos = response_encoding_cv_pos.values.reshape(-1,1)
X_cv_clean_categories_neg = response_encoding_cv_neg.values.reshape(-1,1)
X_test_clean_categories_pos = response_encoding_test_pos.values.reshape(-1,1)
X_test_clean_categories_neg = response_encoding_test_neg.values.reshape(-1,1)

# print data to do some verification
print(X_train_clean_categories_pos.shape, y_train.shape)
print(X_train_clean_categories_neg.shape, y_train.shape)
print(X_cv_clean_categories_pos.shape, y_cv.shape)
print(X_cv_clean_categories_neg.shape, y_cv.shape)
print(X_test_clean_categories_pos.shape, y_test.shape)
print(X_test_clean_categories_neg.shape, y_test.shape)
print('*'*100)
print('train data positive')
print(X_train_clean_categories_pos[0:5])
print('*'*100)
print('train data negative')
print(X_train_clean_categories_neg[0:5])
print('*'*100)
print('cv data positive')
print(X_cv_clean_categories_pos[0:5])
print('*'*100)
print('cv data negative')
print(X_cv_clean_categories_neg[0:5])
print('*'*100)
print('test data positive')
print(X_test_clean_categories_pos[0:5])
print('*'*100)
print('test data negative')
print(X_test_clean_categories_neg[0:5])
print('*'*100)


X_train_clean_categories_pos = coo_matrix(X_train_clean_categories_pos)
X_train_clean_categories_neg = coo_matrix(X_train_clean_categories_neg)
X_cv_clean_categories_pos = coo_matrix(X_cv_clean_categories_pos)
X_cv_clean_categories_neg = coo_matrix(X_cv_clean_categories_neg)
X_test_clean_categories_pos = coo_matrix(X_test_clean_categories_pos)
X_test_clean_categories_neg = coo_matrix(X_test_clean_categories_neg)

# print csr matrix shape
print(X_train_clean_categories_pos.shape, y_train.shape)
print(X_train_clean_categories_neg.shape, y_train.shape)
print(X_cv_clean_categories_pos.shape, y_cv.shape)
```

```
print(X_cv_clean_categories_pos.shape, y_cv.shape)
print(X_cv_clean_categories_neg.shape, y_cv.shape)
print(X_test_clean_categories_pos.shape, y_test.shape)
print(X_test_clean_categories_neg.shape, y_test.shape)
print('*'*100)
```

```
(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************

train data positive
[[0.84304933]
 [0.81161972]
 [0.84277879]
 [0.80555556]
 [0.81161972]]
****************************************************************************************************

train data negative
[[0.15695067]
 [0.18838028]
 [0.15722121]
 [0.19444444]
 [0.18838028]]
****************************************************************************************************

cv data positive
[[0.8676022 ]
 [0.88288288]
 [0.84277879]
 [0.82752458]
 [0.8676022 ]]
****************************************************************************************************

cv data negative
[[0.1323978 ]
 [0.11711712]
 [0.15722121]
 [0.17247542]
 [0.1323978 ]]
****************************************************************************************************

test data positive
[[0.82608696]
 [0.82608696]
 [0.82608696]
 [0.86263455]
 [0.84277879]]
****************************************************************************************************

test data negative
[[0.17391304]
 [0.17391304]
 [0.17391304]
 [0.13736545]
 [0.15722121]]
****************************************************************************************************

(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************
```

In [297]:

```
# We are using response encoding instead of one hot encoding for categorical feature
# - clean_categories : categorical data
# calculate response encoding on clean_subcategories train data i.e. fit data
```

```
proba_score_train = calculate_proba_score_of_each_variable(X_train['clean_subcategories'].values,
y_train)

# transform train data
response_encoding_train_pos, response_encoding_train_neg = convert_response_encoding(X_train['clea
n_subcategories'].values, proba_score_train)

response_encoding_train_pos = pd.DataFrame(response_encoding_train_pos)
response_encoding_train_neg = pd.DataFrame(response_encoding_train_neg)

# transform cv data
response_encoding_cv_pos, response_encoding_cv_neg =
convert_response_encoding(X_cv['clean_subcategories'].values, proba_score_train)

response_encoding_cv_pos = pd.DataFrame(response_encoding_cv_pos)
response_encoding_cv_neg = pd.DataFrame(response_encoding_cv_neg)

# transform test data
response_encoding_test_pos, response_encoding_test_neg =
convert_response_encoding(X_test['clean_subcategories'].values, proba_score_train)

response_encoding_test_pos = pd.DataFrame(response_encoding_test_pos)
response_encoding_test_neg = pd.DataFrame(response_encoding_test_neg)

# reshape data
X_train_clean_subcategories_pos = response_encoding_train_pos.values.reshape(-1,1)
X_train_clean_subcategories_neg = response_encoding_train_neg.values.reshape(-1,1)
X_cv_clean_subcategories_pos = response_encoding_cv_pos.values.reshape(-1,1)
X_cv_clean_subcategories_neg = response_encoding_cv_neg.values.reshape(-1,1)
X_test_clean_subcategories_pos = response_encoding_test_pos.values.reshape(-1,1)
X_test_clean_subcategories_neg = response_encoding_test_neg.values.reshape(-1,1)

# print data to do some verification
print(X_train_clean_subcategories_pos.shape, y_train.shape)
print(X_train_clean_subcategories_neg.shape, y_train.shape)
print(X_cv_clean_subcategories_pos.shape, y_cv.shape)
print(X_cv_clean_subcategories_neg.shape, y_cv.shape)
print(X_test_clean_subcategories_pos.shape, y_test.shape)
print(X_test_clean_subcategories_neg.shape, y_test.shape)
print('*'*100)
print('train data positive')
print(X_train_clean_subcategories_pos[0:5])
print('*'*100)
print('train data negative')
print(X_train_clean_subcategories_neg[0:5])
print('*'*100)
print('cv data positive')
print(X_cv_clean_subcategories_pos[0:5])
print('*'*100)
print('cv data negative')
print(X_cv_clean_subcategories_neg[0:5])
print('*'*100)
print('test data positive')
print(X_test_clean_subcategories_pos[0:5])
print('*'*100)
print('test data negative')
print(X_test_clean_subcategories_neg[0:5])
print('*'*100)


X_train_clean_subcategories_pos = coo_matrix(X_train_clean_subcategories_pos)
X_train_clean_subcategories_neg = coo_matrix(X_train_clean_subcategories_neg)
X_cv_clean_subcategories_pos = coo_matrix(X_cv_clean_subcategories_pos)
X_cv_clean_subcategories_neg = coo_matrix(X_cv_clean_subcategories_neg)
X_test_clean_subcategories_pos = coo_matrix(X_test_clean_subcategories_pos)
X_test_clean_subcategories_neg = coo_matrix(X_test_clean_subcategories_neg)

# print csr matrix shape
print(X_train_clean_subcategories_pos.shape, y_train.shape)
print(X_train_clean_subcategories_neg.shape, y_train.shape)
print(X_cv_clean_subcategories_pos.shape, y_cv.shape)
print(X_cv_clean_subcategories_neg.shape, y_cv.shape)
print(X_test_clean_subcategories_pos.shape, y_test.shape)
print(X_test_clean_subcategories_neg.shape, y_test.shape)
print('*'*100)
```

```
(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************

train data positive
[[0.72413793]
 [0.81161972]
 [0.8483965 ]
 [0.80555556]
 [0.81161972]]
****************************************************************************************************

train data negative
[[0.27586207]
 [0.18838028]
 [0.1516035 ]
 [0.19444444]
 [0.18838028]]
****************************************************************************************************

cv data positive
[[0.87652439]
 [0.93877551]
 [0.8483965 ]
 [0.84563758]
 [0.85669291]]
****************************************************************************************************

cv data negative
[[0.12347561]
 [0.06122449]
 [0.1516035 ]
 [0.15436242]
 [0.14330709]]
****************************************************************************************************

test data positive
[[0.78787879]
 [0.81818182]
 [0.78787879]
 [0.85882353]
 [0.8483965 ]]
****************************************************************************************************

test data negative
[[0.21212121]
 [0.18181818]
 [0.21212121]
 [0.14117647]
 [0.1516035 ]]
****************************************************************************************************

(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************
```

In [298]:

```
# We are using response encoding instead of one hot encoding for categorical feature
# - project_grade_category : categorical data
# calculate response encoding on project_grade_category train data i.e. fit data
proba_score_train =
calculate_proba_score_of_each_variable(X_train['project_grade_category'].values, y_train)

# transform train data
response_encoding_train_pos, response_encoding_train_neg = convert_response_encoding(X_train['proj
ect_grade_category'].values, proba_score_train)
```

```python
response_encoding_train_pos = pd.DataFrame(response_encoding_train_pos)
response_encoding_train_neg = pd.DataFrame(response_encoding_train_neg)

# transform cv data
response_encoding_cv_pos, response_encoding_cv_neg =
convert_response_encoding(X_cv['project_grade_category'].values, proba_score_train)

response_encoding_cv_pos = pd.DataFrame(response_encoding_cv_pos)
response_encoding_cv_neg = pd.DataFrame(response_encoding_cv_neg)

# transform test data
response_encoding_test_pos, response_encoding_test_neg =
convert_response_encoding(X_test['project_grade_category'].values, proba_score_train)

response_encoding_test_pos = pd.DataFrame(response_encoding_test_pos)
response_encoding_test_neg = pd.DataFrame(response_encoding_test_neg)

# reshape data
X_train_project_grade_category_pos = response_encoding_train_pos.values.reshape(-1,1)
X_train_project_grade_category_neg = response_encoding_train_neg.values.reshape(-1,1)
X_cv_project_grade_category_pos = response_encoding_cv_pos.values.reshape(-1,1)
X_cv_project_grade_category_neg = response_encoding_cv_neg.values.reshape(-1,1)
X_test_project_grade_category_pos = response_encoding_test_pos.values.reshape(-1,1)
X_test_project_grade_category_neg = response_encoding_test_neg.values.reshape(-1,1)

# print data to do some verification
print(X_train_project_grade_category_pos.shape, y_train.shape)
print(X_train_project_grade_category_neg.shape, y_train.shape)
print(X_cv_project_grade_category_pos.shape, y_cv.shape)
print(X_cv_project_grade_category_neg.shape, y_cv.shape)
print(X_test_project_grade_category_pos.shape, y_test.shape)
print(X_test_project_grade_category_neg.shape, y_test.shape)
print('*'*100)
print('train data positive')
print(X_train_project_grade_category_pos[0:5])
print('*'*100)
print('train data negative')
print(X_train_project_grade_category_neg[0:5])
print('*'*100)
print('cv data positive')
print(X_cv_project_grade_category_pos[0:5])
print('*'*100)
print('cv data negative')
print(X_cv_project_grade_category_neg[0:5])
print('*'*100)
print('test data positive')
print(X_test_project_grade_category_pos[0:5])
print('*'*100)
print('test data negative')
print(X_test_project_grade_category_neg[0:5])
print('*'*100)


X_train_project_grade_category_pos = coo_matrix(X_train_project_grade_category_pos)
X_train_project_grade_category_neg = coo_matrix(X_train_project_grade_category_neg)
X_cv_project_grade_category_pos = coo_matrix(X_cv_project_grade_category_pos)
X_cv_project_grade_category_neg = coo_matrix(X_cv_project_grade_category_neg)
X_test_project_grade_category_pos = coo_matrix(X_test_project_grade_category_pos)
X_test_project_grade_category_neg = coo_matrix(X_test_project_grade_category_neg)

# print csr matrix shape
print(X_train_project_grade_category_pos.shape, y_train.shape)
print(X_train_project_grade_category_neg.shape, y_train.shape)
print(X_cv_project_grade_category_pos.shape, y_cv.shape)
print(X_cv_project_grade_category_neg.shape, y_cv.shape)
print(X_test_project_grade_category_pos.shape, y_test.shape)
print(X_test_project_grade_category_neg.shape, y_test.shape)
print('*'*100)
```

```
(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************
```

```
****************************************************************************************************

train data positive
[[0.83940972]
 [0.84898711]
 [0.8560804 ]
 [0.84441398]
 [0.8560804 ]]
****************************************************************************************************

train data negative
[[0.16059028]
 [0.15101289]
 [0.1439196 ]
 [0.15558602]
 [0.1439196 ]]
****************************************************************************************************

cv data positive
[[0.84898711]
 [0.84898711]
 [0.8560804 ]
 [0.8560804 ]
 [0.8560804 ]]
****************************************************************************************************

cv data negative
[[0.15101289]
 [0.15101289]
 [0.1439196 ]
 [0.1439196 ]
 [0.1439196 ]]
****************************************************************************************************

test data positive
[[0.8560804 ]
 [0.8560804 ]
 [0.83940972]
 [0.8560804 ]
 [0.83940972]]
****************************************************************************************************

test data negative
[[0.1439196 ]
 [0.1439196 ]
 [0.16059028]
 [0.1439196 ]
 [0.16059028]]
****************************************************************************************************

(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************
```

In [299]:

```
# We are using response encoding instead of one hot encoding for categorical feature
# - teacher_prefix : categorical data
# calculate response encoding on teacher_prefix train data i.e. fit data
proba_score_train = calculate_proba_score_of_each_variable(X_train['teacher_prefix'].values,
y_train)

# transform train data
response_encoding_train_pos, response_encoding_train_neg = convert_response_encoding(X_train['teac
her_prefix'].values, proba_score_train)

response_encoding_train_pos = pd.DataFrame(response_encoding_train_pos)
response_encoding_train_neg = pd.DataFrame(response_encoding_train_neg)

# transform cv data
response_encoding_cv_pos, response_encoding_cv_neg =
convert_response_encoding(X_cv['teacher_prefix'].values, proba_score_train)
```

```
convert_response_encoding(X_cv['teacher_prefix'].values, proba_score_train)

response_encoding_cv_pos = pd.DataFrame(response_encoding_cv_pos)
response_encoding_cv_neg = pd.DataFrame(response_encoding_cv_neg)

# transform test data
response_encoding_test_pos, response_encoding_test_neg =
convert_response_encoding(X_test['teacher_prefix'].values, proba_score_train)

response_encoding_test_pos = pd.DataFrame(response_encoding_test_pos)
response_encoding_test_neg = pd.DataFrame(response_encoding_test_neg)

# reshape data
X_train_teacher_prefix_pos = response_encoding_train_pos.values.reshape(-1,1)
X_train_teacher_prefix_neg = response_encoding_train_neg.values.reshape(-1,1)
X_cv_teacher_prefix_pos = response_encoding_cv_pos.values.reshape(-1,1)
X_cv_teacher_prefix_neg = response_encoding_cv_neg.values.reshape(-1,1)
X_test_teacher_prefix_pos = response_encoding_test_pos.values.reshape(-1,1)
X_test_teacher_prefix_neg = response_encoding_test_neg.values.reshape(-1,1)

# print data to do some verification
print(X_train_teacher_prefix_pos.shape, y_train.shape)
print(X_train_teacher_prefix_neg.shape, y_train.shape)
print(X_cv_teacher_prefix_pos.shape, y_cv.shape)
print(X_cv_teacher_prefix_neg.shape, y_cv.shape)
print(X_test_teacher_prefix_pos.shape, y_test.shape)
print(X_test_teacher_prefix_neg.shape, y_test.shape)
print('*'*100)
print('train data positive')
print(X_train_teacher_prefix_pos[0:5])
print('*'*100)
print('train data negative')
print(X_train_teacher_prefix_neg[0:5])
print('*'*100)
print('cv data positive')
print(X_cv_teacher_prefix_pos[0:5])
print('*'*100)
print('cv data negative')
print(X_cv_teacher_prefix_neg[0:5])
print('*'*100)
print('test data positive')
print(X_test_teacher_prefix_pos[0:5])
print('*'*100)
print('test data negative')
print(X_test_teacher_prefix_neg[0:5])
print('*'*100)


X_train_teacher_prefix_pos = coo_matrix(X_train_teacher_prefix_pos)
X_train_teacher_prefix_neg = coo_matrix(X_train_teacher_prefix_neg)
X_cv_teacher_prefix_pos = coo_matrix(X_cv_teacher_prefix_pos)
X_cv_teacher_prefix_neg = coo_matrix(X_cv_teacher_prefix_neg)
X_test_teacher_prefix_pos = coo_matrix(X_test_teacher_prefix_pos)
X_test_teacher_prefix_neg = coo_matrix(X_test_teacher_prefix_neg)

# print csr matrix shape
print(X_train_teacher_prefix_pos.shape, y_train.shape)
print(X_train_teacher_prefix_neg.shape, y_train.shape)
print(X_cv_teacher_prefix_pos.shape, y_cv.shape)
print(X_cv_teacher_prefix_neg.shape, y_cv.shape)
print(X_test_teacher_prefix_pos.shape, y_test.shape)
print(X_test_teacher_prefix_neg.shape, y_test.shape)
print('*'*100)
```

```
(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************

train data positive
[[0.8539633 ]
 [0.85095057]
 [0.85095057]
 [0.85095057]
```

```
 [0.83367698]]
****************************************************************************************************

train data negative
[[0.1460367 ]
 [0.14904943]
 [0.14904943]
 [0.14904943]
 [0.16632302]]
****************************************************************************************************

cv data positive
[[0.8539633 ]
 [0.85095057]
 [0.8539633 ]
 [0.85095057]
 [0.85095057]]
****************************************************************************************************

cv data negative
[[0.1460367 ]
 [0.14904943]
 [0.1460367 ]
 [0.14904943]
 [0.14904943]]
****************************************************************************************************

test data positive
[[0.85095057]
 [0.85095057]
 [0.8539633 ]
 [0.8539633 ]
 [0.83367698]]
****************************************************************************************************

test data negative
[[0.14904943]
 [0.14904943]
 [0.1460367 ]
 [0.1460367 ]
 [0.16632302]]
****************************************************************************************************

(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************
```

1.5.4 Merging all the above features

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

In [300]:

```
# print(categories_one_hot.shape)
# print(sub_categories_one_hot.shape)
# print(text_bow.shape)
# print(price_standardized.shape)
print('Categorical Features')
print('*'*100)
print(X_train_school_state_pos.shape, y_train.shape)
print(X_train_school_state_neg.shape, y_train.shape)
print(X_cv_school_state_pos.shape, y_cv.shape)
print(X_cv_school_state_neg.shape, y_cv.shape)
print(X_test_school_state_pos.shape, y_test.shape)
print(X_test_school_state_neg.shape, y_test.shape)
print('*'*100)
print(X_train_clean_categories_pos.shape, y_train.shape)
print(X_train_clean_categories_neg.shape, y_train.shape)
```

```
print(X_cv_clean_categories_pos.shape, y_cv.shape)
print(X_cv_clean_categories_neg.shape, y_cv.shape)
print(X_test_clean_categories_pos.shape, y_test.shape)
print(X_test_clean_categories_neg.shape, y_test.shape)
print('*'*100)
print(X_train_clean_subcategories_pos.shape, y_train.shape)
print(X_train_clean_subcategories_neg.shape, y_train.shape)
print(X_cv_clean_subcategories_pos.shape, y_cv.shape)
print(X_cv_clean_subcategories_neg.shape, y_cv.shape)
print(X_test_clean_subcategories_pos.shape, y_test.shape)
print(X_test_clean_subcategories_neg.shape, y_test.shape)
print('*'*100)
print(X_train_project_grade_category_pos.shape, y_train.shape)
print(X_train_project_grade_category_neg.shape, y_train.shape)
print(X_cv_project_grade_category_pos.shape, y_cv.shape)
print(X_cv_project_grade_category_neg.shape, y_cv.shape)
print(X_test_project_grade_category_pos.shape, y_test.shape)
print(X_test_project_grade_category_neg.shape, y_test.shape)
print('*'*100)
print(X_train_teacher_prefix_pos.shape, y_train.shape)
print(X_train_teacher_prefix_neg.shape, y_train.shape)
print(X_cv_teacher_prefix_pos.shape, y_cv.shape)
print(X_cv_teacher_prefix_neg.shape, y_cv.shape)
print(X_test_teacher_prefix_pos.shape, y_test.shape)
print(X_test_teacher_prefix_neg.shape, y_test.shape)
print('*'*100)
print('Text Encoding Features')
print('*'*100)
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print('*'*100)
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print('*'*100)
print(X_train_project_resource_summary_bow.shape, y_train.shape)
print(X_cv_project_resource_summary_bow.shape, y_cv.shape)
print(X_test_project_resource_summary_bow.shape, y_test.shape)
print('*'*100)
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_cv_title_tfidf.shape, y_test.shape)
print('*'*100)
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print('*'*100)
print(X_train_project_resource_summary_tfidf.shape, y_train.shape)
print(X_cv_project_resource_summary_tfidf.shape, y_cv.shape)
print(X_test_project_resource_summary_tfidf.shape, y_test.shape)
print('*'*100)
print(len(avg_w2v_vectors_text_train))
print(len(avg_w2v_vectors_text_train[0]))
print('*'*100)
print(len(avg_w2v_vectors_text_cv))
print(len(avg_w2v_vectors_text_cv[0]))
print('*'*100)
print(len(avg_w2v_vectors_text_test))
print(len(avg_w2v_vectors_text_test[0]))
print('*'*100)
print(len(avg_w2v_vectors_title_train))
print(len(avg_w2v_vectors_title_train[0]))
print('*'*100)
print(len(avg_w2v_vectors_title_cv))
print(len(avg_w2v_vectors_title_cv[0]))
print('*'*100)
print(len(avg_w2v_vectors_title_test))
print(len(avg_w2v_vectors_title_test[0]))
print('*'*100)
print(len(avg_w2v_vectors_project_resource_summary_train))
print(len(avg_w2v_vectors_project_resource_summary_train[0]))
print('*'*100)
print(len(avg_w2v_vectors_project_resource_summary_cv))
print(len(avg_w2v_vectors_project_resource_summary_cv[0]))
print('*'*100)
print(len(avg_w2v_vectors_project_resource_summary_test))
```

```
print(len(avg_w2v_vectors_project_resource_summary_test[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_text_train))
print(len(tfidf_w2v_vectors_text_train[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_text_cv))
print(len(tfidf_w2v_vectors_text_cv[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_text_test))
print(len(tfidf_w2v_vectors_text_test[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_title_train))
print(len(tfidf_w2v_vectors_title_train[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_title_cv))
print(len(tfidf_w2v_vectors_title_cv[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_title_test))
print(len(tfidf_w2v_vectors_title_test[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_project_resource_summary_train))
print(len(tfidf_w2v_vectors_project_resource_summary_train[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_project_resource_summary_cv))
print(len(tfidf_w2v_vectors_project_resource_summary_cv[0]))
print('*'*100)
print(len(tfidf_w2v_vectors_project_resource_summary_test))
print(len(tfidf_w2v_vectors_project_resource_summary_test[0]))
print('*'*100)
print('Numerical Features')
print('*'*100)
print(X_train_quantity_norm.shape, y_train.shape)
print(X_cv_quantity_norm.shape, y_cv.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print('*'*100)
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.shape)
print(X_cv_teacher_number_of_previously_posted_projects_norm.shape, y_cv.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.shape)
print('*'*100)
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
```

```
Categorical Features
****************************************************************************************************

(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************

(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************

(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************

(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
```

```
****************************************************************************************************

(14711, 1) (14711,)
(14711, 1) (14711,)
(7247, 1) (7247,)
(7247, 1) (7247,)
(10816, 1) (10816,)
(10816, 1) (10816,)
****************************************************************************************************

Text Encoding Features
****************************************************************************************************

(14711, 1130) (14711,)
(7247, 1130) (7247,)
(10816, 1130) (10816,)
****************************************************************************************************

(14711, 5000) (14711,)
(7247, 5000) (7247,)
(10816, 5000) (10816,)
****************************************************************************************************

(14711, 4134) (14711,)
(7247, 4134) (7247,)
(10816, 4134) (10816,)
****************************************************************************************************

(14711, 849) (14711,)
(7247, 849) (7247,)
(7247, 849) (10816,)
****************************************************************************************************

(14711, 7337) (14711,)
(7247, 7337) (7247,)
(10816, 7337) (10816,)
****************************************************************************************************

(14711, 1872) (14711,)
(7247, 1872) (7247,)
(10816, 1872) (10816,)
****************************************************************************************************

14711
300
****************************************************************************************************

7247
300
****************************************************************************************************

10816
300
****************************************************************************************************

14711
300
****************************************************************************************************

7247
300
****************************************************************************************************

10816
300
****************************************************************************************************

14711
300
****************************************************************************************************

7247
300
****************************************************************************************************

10816
300
```

```
****************************************************************************************************

14711
300
****************************************************************************************************

7247
300
****************************************************************************************************

10816
300
****************************************************************************************************

14711
300
****************************************************************************************************

7247
300
****************************************************************************************************

10816
300
****************************************************************************************************

14711
300
****************************************************************************************************

7247
300
****************************************************************************************************

10816
300
****************************************************************************************************

Numerical Features
****************************************************************************************************

(14711, 1) (14711,)
(7247, 1) (7247,)
(10816, 1) (10816,)
****************************************************************************************************

(14711, 1) (14711,)
(7247, 1) (7247,)
(10816, 1) (10816,)
****************************************************************************************************

(14711, 1) (14711,)
(7247, 1) (7247,)
(10816, 1) (10816,)
```

In [301]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
# X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
# X.shape

X_train_real = X_train
X_cv_real = X_cv
X_test_real = X_test

X_train = hstack((X_train_school_state_pos, X_train_school_state_neg,
X_train_clean_categories_pos, X_train_clean_categories_neg, X_train_clean_subcategories_pos,
X_train_clean_subcategories_neg, X_train_project_grade_category_pos,
X_train_project_grade_category_neg, X_train_teacher_prefix_pos, X_train_teacher_prefix_neg,
X_train_title_bow, X_train_essay_bow, X_train_project_resource_summary_bow, X_train_title_tfidf, X
_train_essay_tfidf, X_train_project_resource_summary_tfidf, avg_w2v_vectors_text_train,
avg_w2v_vectors_title_train, avg_w2v_vectors_project_resource_summary_train,
tfidf_w2v_vectors_text_train, tfidf_w2v_vectors_title_train,
tfidf_w2v_vectors_project_resource_summary_train, X_train_quantity_norm,
```

```
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm)).tocsr()
X_cv = hstack((X_cv_school_state_pos, X_cv_school_state_neg, X_cv_clean_categories_pos,
X_cv_clean_categories_neg, X_cv_clean_subcategories_pos, X_cv_clean_subcategories_neg,
X_cv_project_grade_category_pos, X_cv_project_grade_category_neg, X_cv_teacher_prefix_pos,
X_cv_teacher_prefix_neg, X_cv_title_bow, X_cv_essay_bow, X_cv_project_resource_summary_bow,
X_cv_title_tfidf, X_cv_essay_tfidf, X_cv_project_resource_summary_tfidf, avg_w2v_vectors_text_cv,
avg_w2v_vectors_title_cv, avg_w2v_vectors_project_resource_summary_cv, tfidf_w2v_vectors_text_cv,
tfidf_w2v_vectors_title_cv, tfidf_w2v_vectors_project_resource_summary_cv, X_cv_quantity_norm, X_c
v_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocsr()
X_test = hstack((X_test_school_state_pos, X_test_school_state_neg, X_test_clean_categories_pos,
X_test_clean_categories_neg, X_test_clean_subcategories_pos, X_test_clean_subcategories_neg, X_tes
t_project_grade_category_pos,  X_test_project_grade_category_neg, X_test_teacher_prefix_pos,
X_test_teacher_prefix_neg, X_test_title_bow, X_test_essay_bow, X_test_project_resource_summary_bow
, X_test_title_tfidf, X_test_essay_tfidf, X_test_project_resource_summary_tfidf,
avg_w2v_vectors_text_test, avg_w2v_vectors_title_test,
avg_w2v_vectors_project_resource_summary_test, tfidf_w2v_vectors_text_test,
tfidf_w2v_vectors_title_test, tfidf_w2v_vectors_project_resource_summary_test,
X_test_quantity_norm, X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm)
).tocsr()

print(X_train_real.shape)
print(X_cv_real.shape)
print(X_test_real.shape)
print(X_train.shape)
print(X_cv.shape)
print(X_test.shape)
```

```
(14711, 17)
(7247, 17)
(10816, 17)
(14711, 22135)
(7247, 22135)
(10816, 22135)
```

Computing Sentiment Scores

In [302]:

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

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students 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
```

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

Assignment 9: RF and GBDT

Response Coding: Example

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

1. **Apply both Random Forrest and GBDT on these feature sets**
    - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(BOW) + preprocessed_eassay (BOW)
    - Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
    - Set 3: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
    - Set 4: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (Consider any two hyper parameters preferably n_estimators, max_depth)**
    - Consider the following range for hyperparameters **n_estimators** = [10, 50, 100, 150, 200, 300, 500, 1000], **max_depth** = [2, 3, 4, 5, 6, 7, 8, 9, 10]
    - Find the best hyper parameter which will give the maximum AUC value
    - find the best hyper paramter using k-fold cross validation/simple cross validation data
    - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task
3. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure
      with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

      or

    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure
      seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**
    - You can choose either of the plotting techniques: 3d plot or heat map
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points
4. **Conclusion**
    - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

refer to this prettytable library link

Note: Data Leakage

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

2. Random Forest and GBDT

Note: I already completed steps 2.1, 2.2 & 2.3 previously, So I didn't copy code in below cells.

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

In [303]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [304]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [305]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [306]:

```
#function to get heatmap confusion matrix
def get_confusion_matrix(clf,X_te,y_test):
```

```python
def get_confusion_matrix(clf,X_te,y_test):
    y_pred = clf.predict(X_te)
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

In [307]:

```python
# Collect bow features name
# print(10 + len(clean_titles_bow_features) + len(easy_bow_features) +
len(project_resource_summary_bow_features) + 3)

feature_names_bow = ['school_state_pos', 'school_state_neg', 'clean_categories_pos',
'clean_categories_neg', 'clean_subcategories_pos', 'clean_subcategories_neg' \
                , 'project_grade_category_pos', 'project_grade_category_neg', 'teacher_prefix_pos',
'teacher_prefix_neg']

# append title bow
for i in clean_titles_bow_features:
    feature_names_bow.append(i)

# append easy bow
for i in easy_bow_features:
    feature_names_bow.append(i)

# append project resource summary bow
for i in project_resource_summary_bow_features:
    feature_names_bow.append(i)


feature_names_bow.append('quantity')
feature_names_bow.append('teacher_number_of_previously_posted_projects_norm')
feature_names_bow.append('price')

len(feature_names_bow)
```

Out[307]:

```
10277
```

In [308]:

```python
# Collect tfidf features name
# print(10 + len(clean_titles_tfidf_features) + len(easy_tfidf_features) +
len(project_resource_summary_tfidf_features) + 3)

feature_names_tfidf = ['school_state_pos', 'school_state_neg', 'clean_categories_pos',
'clean_categories_neg', 'clean_subcategories_pos', 'clean_subcategories_neg' \
                , 'project_grade_category_pos', 'project_grade_category_neg', 'teacher_prefix_pos',
'teacher_prefix_neg']

# append title tfidf
for i in clean_titles_tfidf_features:
    feature_names_tfidf.append(i)

# append easy tfidf
for i in easy_tfidf_features:
    feature_names_tfidf.append(i)

# append project resource summary tfidf
for i in project_resource_summary_tfidf_features:
    feature_names_tfidf.append(i)


feature_names_tfidf.append('quantity')
feature_names_tfidf.append('teacher_number_of_previously_posted_projects_norm')
feature_names_tfidf.append('price')

len(feature_names_tfidf)
```

Out[308]:

```
10071
```

2.4 Applying Random Forest

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

2.4.1 Applying Random Forests on BOW, SET 1

In [309]:

```
%%time
# Please write all the code with proper documentation

# Prepare data for BOW
X_train_bow = hstack((X_train_school_state_pos, X_train_school_state_neg,
X_train_clean_categories_pos, X_train_clean_categories_neg, X_train_clean_subcategories_pos,
X_train_clean_subcategories_neg, X_train_project_grade_category_pos,
X_train_project_grade_category_neg, X_train_teacher_prefix_pos, X_train_teacher_prefix_neg,
X_train_title_bow, X_train_essay_bow, X_train_project_resource_summary_bow, X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm)).tocsr()
X_cv_bow = hstack((X_cv_school_state_pos, X_cv_school_state_neg, X_cv_clean_categories_pos,
X_cv_clean_categories_neg, X_cv_clean_subcategories_pos, X_cv_clean_subcategories_neg,
X_cv_project_grade_category_pos, X_cv_project_grade_category_neg, X_cv_teacher_prefix_pos,
X_cv_teacher_prefix_neg, X_cv_title_bow, X_cv_essay_bow, X_cv_project_resource_summary_bow,
X_cv_quantity_norm, X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocs
r()
X_test_bow = hstack((X_test_school_state_pos, X_test_school_state_neg, X_test_clean_categories_pos
, X_test_clean_categories_neg, X_test_clean_subcategories_pos, X_test_clean_subcategories_neg, X_t
est_project_grade_category_pos,  X_test_project_grade_category_neg, X_test_teacher_prefix_pos, X_t
est_teacher_prefix_neg, X_test_title_bow, X_test_essay_bow, X_test_project_resource_summary_bow, X
_test_quantity_norm, X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm))
.tocsr()

print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)

import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV

rf = RandomForestClassifier(class_weight='balanced', n_jobs=-1)
parameters = {'n_estimators':[10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6, 7
, 8, 9, 10]}
clf=GridSearchCV(rf, parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score=True)
clf.fit(X_train_bow, y_train)
```

```
(14711, 10277) (14711,)
(7247, 10277) (7247,)
(10816, 10277) (10816,)
CPU times: user 23.9 s, sys: 925 ms, total: 24.9 s
Wall time: 9min 53s
```

Out[309]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True,
                                              class_weight='balanced',
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=-1,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
```

```
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [10, 50, 100, 150, 200, 300, 500,
                                          1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [310]:

```
# Print results
clf.cv_results_
```

Out[310]:

```
{'mean_fit_time': array([ 0.61875661,  0.54400102,  1.16851274,  1.40403922,  1.90348434,
         2.59251515,  5.50109871, 11.27046665,  0.60973668,  1.05472755,
         1.3640426 ,  1.35722796,  1.63133446,  2.30017352,  3.51424289,
         8.87645125,  0.48755217,  0.67342496,  1.05075439,  1.81440131,
         2.43310356,  2.35173027,  5.59776775, 12.0179584 ,  0.43121902,
         1.04187187,  2.48384929,  2.92801706,  2.87375855,  3.17518663,
         5.06910968, 16.06109587,  0.72554898,  1.56332342,  2.18041643,
         3.40146406,  4.27123539,  4.40192731,  7.86789171, 20.26397904,
         0.69598214,  1.34199834,  3.16028961,  3.26415634,  4.51291108,
         4.22032889, 10.85717257, 19.60525632,  0.45750157,  1.63247712,
         2.28312429,  3.57134803,  5.0279309 ,  4.99995343,  9.78132232,
        18.12775858,  0.43889904,  1.50305343,  1.92915996,  3.32423393,
         4.27241826,  6.88764   , 14.18904908, 23.79648805,  0.60179226,
         1.34964101,  2.94522127,  3.54675269,  5.55132604,  9.34241199,
        15.72577167, 27.28302113]),
 'std_fit_time': array([0.04517402, 0.2138727 , 0.18975125, 0.15191877, 0.11347988,
        0.1217607 , 0.32227449, 1.2575045 , 0.09942467, 0.26069219,
        0.17556694, 0.17562906, 0.17939971, 0.0206654 , 0.33628863,
        0.4785854 , 0.09768891, 0.13900128, 0.05040857, 0.09310271,
        0.20878181, 0.08906147, 0.34161859, 1.0211632 , 0.02595257,
        0.30147732, 0.18034366, 0.6721614 , 0.20762528, 0.06701249,
        0.59460767, 2.2099849 , 0.05919553, 0.17301867, 0.35488122,
        0.47211306, 0.23586418, 0.20478353, 0.91381513, 1.32523321,
        0.15962816, 0.20836413, 0.57255453, 0.11613849, 0.19185893,
        0.08023602, 0.55349742, 1.2180666 , 0.07978664, 0.18633281,
        0.16670587, 0.34854066, 0.19015797, 0.48271274, 0.93419182,
        0.73547398, 0.02108914, 0.16500927, 0.15523879, 0.27044797,
        0.12525504, 0.60239346, 1.13098072, 1.00153275, 0.05608519,
        0.01959512, 0.14512331, 0.22890224, 0.25431089, 0.35755048,
        0.4838238 , 1.13164693]),
 'mean_score_time': array([0.16399137, 0.1382362 , 0.37878116, 0.6338671 , 0.86786254,
        1.53344965, 3.58300201, 5.96797077, 0.11860736, 0.27540088,
        0.6349024 , 0.56742938, 0.74072162, 1.15575345, 2.1236368 ,
        4.83635243, 0.11590838, 0.14958533, 0.39268327, 0.79267263,
        0.91942914, 1.52552764, 2.68520045, 7.31251224, 0.11672624,
        0.41224972, 0.79472939, 0.7824316 , 1.15431023, 1.46495406,
        2.89595811, 6.9478271 , 0.13081638, 0.32114561, 0.71150064,
        1.09067655, 1.85399866, 2.13754495, 3.69013961, 7.16727869,
        0.12445092, 0.2591819 , 0.86866633, 0.99528138, 1.38336086,
        2.21867776, 3.60977221, 6.27330057, 0.12221869, 0.28583996,
        0.52656261, 0.82279968, 1.33150069, 2.41841427, 2.85812942,
        5.52207168, 0.12068327, 0.26032424, 0.45638808, 0.87773466,
        0.93852433, 2.83900062, 2.80562202, 5.87337089, 0.11816478,
        0.22384103, 0.63641795, 0.77614252, 1.279591  , 2.42631865,
        3.45799971, 5.27976505]),
 'std_score_time': array([3.85264393e-02, 2.37272171e-02, 6.81459026e-02, 8.78821896e-02,
        4.67914097e-02, 2.72166746e-01, 5.89550667e-01, 1.02065072e+00,
        3.86493867e-03, 7.10768543e-02, 9.82556563e-02, 9.92640417e-02,
        4.46076406e-03, 8.43571884e-02, 4.82802519e-01, 3.12419123e-01,
        2.80789951e-04, 4.59608316e-02, 4.40194221e-02, 6.78408417e-02,
        5.09124934e-02, 1.62796603e-01, 1.35578088e-01, 1.77401873e+00,
        3.87545880e-03, 1.34627632e-01, 1.24147022e-01, 2.00714944e-01,
        8.56257505e-02, 1.74605777e-01, 9.61454982e-01, 3.28674970e-01,
        3.29878170e-03, 4.68618686e-02, 1.39738486e-01, 1.62134304e-01,
        6.17992327e-02, 1.61316416e-01, 9.35537859e-01, 1.02547381e+00,
        8.51409105e-03, 9.33693818e-02, 7.49062889e-02, 1.19857346e-01,
        1.45555991e-01, 8.33538885e-02, 1.11667492e+00, 7.57114084e-01,
        6.71036504e-03, 5.08804024e-02, 2.22992039e-02, 9.11634150e-02,
        1.38084840e-01, 5.41134788e-02, 9.80150134e-02, 5.18987385e-01,
        2.55769099e-03, 5.14215601e-02, 4.76320601e-02, 8.09011133e-03,
        2.25305381e-01, 1.26064348e-01, 1.43753063e-01, 6.92312070e-01,
        5.31097691e-03, 8.62510598e-02, 1.61608664e-01, 1.21296569e-01,
```

```
       3.09733074e-02, 2.00217300e-01, 5.22274577e-01, 1.40117520e+00]),
 'param_max_depth': masked_array(data=[2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4,
                    4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6,
                    6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8,
                    8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 10,
                    10, 10],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'param_n_estimators': masked_array(data=[10, 50, 100, 150, 200, 300, 500, 1000, 10, 50, 100,
                    150, 200, 300, 500, 1000, 10, 50, 100, 150, 200, 300,
                    500, 1000, 10, 50, 100, 150, 200, 300, 500, 1000, 10,
                    50, 100, 150, 200, 300, 500, 1000, 10, 50, 100, 150,
                    200, 300, 500, 1000, 10, 50, 100, 150, 200, 300, 500,
                    1000, 10, 50, 100, 150, 200, 300, 500, 1000, 10, 50,
                    100, 150, 200, 300, 500, 1000],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'params': [{'max_depth': 2, 'n_estimators': 10},
  {'max_depth': 2, 'n_estimators': 50},
  {'max_depth': 2, 'n_estimators': 100},
  {'max_depth': 2, 'n_estimators': 150},
  {'max_depth': 2, 'n_estimators': 200},
  {'max_depth': 2, 'n_estimators': 300},
  {'max_depth': 2, 'n_estimators': 500},
  {'max_depth': 2, 'n_estimators': 1000},
  {'max_depth': 3, 'n_estimators': 10},
  {'max_depth': 3, 'n_estimators': 50},
  {'max_depth': 3, 'n_estimators': 100},
  {'max_depth': 3, 'n_estimators': 150},
  {'max_depth': 3, 'n_estimators': 200},
  {'max_depth': 3, 'n_estimators': 300},
  {'max_depth': 3, 'n_estimators': 500},
  {'max_depth': 3, 'n_estimators': 1000},
  {'max_depth': 4, 'n_estimators': 10},
  {'max_depth': 4, 'n_estimators': 50},
  {'max_depth': 4, 'n_estimators': 100},
  {'max_depth': 4, 'n_estimators': 150},
  {'max_depth': 4, 'n_estimators': 200},
  {'max_depth': 4, 'n_estimators': 300},
  {'max_depth': 4, 'n_estimators': 500},
  {'max_depth': 4, 'n_estimators': 1000},
  {'max_depth': 5, 'n_estimators': 10},
  {'max_depth': 5, 'n_estimators': 50},
  {'max_depth': 5, 'n_estimators': 100},
  {'max_depth': 5, 'n_estimators': 150},
  {'max_depth': 5, 'n_estimators': 200},
  {'max_depth': 5, 'n_estimators': 300},
  {'max_depth': 5, 'n_estimators': 500},
  {'max_depth': 5, 'n_estimators': 1000},
  {'max_depth': 6, 'n_estimators': 10},
  {'max_depth': 6, 'n_estimators': 50},
  {'max_depth': 6, 'n_estimators': 100},
  {'max_depth': 6, 'n_estimators': 150},
  {'max_depth': 6, 'n_estimators': 200},
  {'max_depth': 6, 'n_estimators': 300},
  {'max_depth': 6, 'n_estimators': 500},
  {'max_depth': 6, 'n_estimators': 1000},
  {'max_depth': 7, 'n_estimators': 10},
  {'max_depth': 7, 'n_estimators': 50},
```

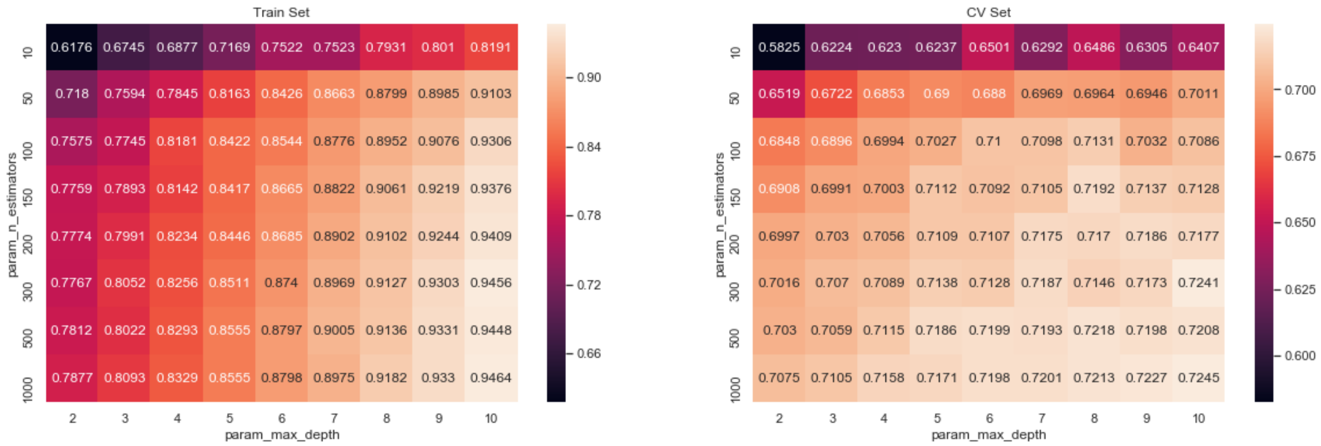
```
 {'max_depth': 7, 'n_estimators': 50},
 {'max_depth': 7, 'n_estimators': 100},
 {'max_depth': 7, 'n_estimators': 150},
 {'max_depth': 7, 'n_estimators': 200},
 {'max_depth': 7, 'n_estimators': 300},
 {'max_depth': 7, 'n_estimators': 500},
 {'max_depth': 7, 'n_estimators': 1000},
 {'max_depth': 8, 'n_estimators': 10},
 {'max_depth': 8, 'n_estimators': 50},
 {'max_depth': 8, 'n_estimators': 100},
 {'max_depth': 8, 'n_estimators': 150},
 {'max_depth': 8, 'n_estimators': 200},
 {'max_depth': 8, 'n_estimators': 300},
 {'max_depth': 8, 'n_estimators': 500},
 {'max_depth': 8, 'n_estimators': 1000},
 {'max_depth': 9, 'n_estimators': 10},
 {'max_depth': 9, 'n_estimators': 50},
 {'max_depth': 9, 'n_estimators': 100},
 {'max_depth': 9, 'n_estimators': 150},
 {'max_depth': 9, 'n_estimators': 200},
 {'max_depth': 9, 'n_estimators': 300},
 {'max_depth': 9, 'n_estimators': 500},
 {'max_depth': 9, 'n_estimators': 1000},
 {'max_depth': 10, 'n_estimators': 10},
 {'max_depth': 10, 'n_estimators': 50},
 {'max_depth': 10, 'n_estimators': 100},
 {'max_depth': 10, 'n_estimators': 150},
 {'max_depth': 10, 'n_estimators': 200},
 {'max_depth': 10, 'n_estimators': 300},
 {'max_depth': 10, 'n_estimators': 500},
 {'max_depth': 10, 'n_estimators': 1000}],
'split0_test_score': array([0.57875452, 0.66481836, 0.68259241, 0.70279088, 0.72057743,
       0.70830007, 0.72272186, 0.72270627, 0.63943763, 0.71001006,
       0.70375818, 0.71695834, 0.72547142, 0.72619543, 0.73172314,
       0.73351027, 0.64260864, 0.7112593 , 0.70062306, 0.71561036,
       0.72889952, 0.72469186, 0.73132979, 0.73446393, 0.63049807,
       0.71152028, 0.72948614, 0.72270952, 0.72900508, 0.73944562,
       0.7371011 , 0.73732003, 0.65446082, 0.69906882, 0.7293744 ,
       0.7294988 , 0.73432068, 0.72657092, 0.74054382, 0.73589247,
       0.64472676, 0.72495691, 0.72581995, 0.72204462, 0.73794757,
       0.74306276, 0.73612763, 0.74252194, 0.67125815, 0.71957375,
       0.72573225, 0.73835846, 0.74136593, 0.73660836, 0.74266908,
       0.74021381, 0.64425853, 0.70877495, 0.71995963, 0.73130315,
       0.73978148, 0.74524   , 0.73976036, 0.74446726, 0.63893546,
       0.70708136, 0.72653779, 0.7278364 , 0.73097996, 0.74268403,
       0.73867191, 0.74680853]),
'split1_test_score': array([0.60373288, 0.6368915 , 0.69010157, 0.69188749, 0.70525153,
       0.69432326, 0.69911021, 0.7048839 , 0.62676908, 0.62649352,
       0.68384882, 0.69025627, 0.69513204, 0.70679068, 0.6973661 ,
       0.70287888, 0.59537278, 0.68503987, 0.69606184, 0.69656546,
       0.69766313, 0.71157764, 0.70530196, 0.71058505, 0.62475543,
       0.69045325, 0.68339075, 0.70651643, 0.70543144, 0.70027686,
       0.72053374, 0.71227873, 0.6514801 , 0.68273391, 0.69994567,
       0.69914535, 0.70583583, 0.70984491, 0.71355046, 0.716294  ,
       0.60744981, 0.68467224, 0.70630106, 0.70743094, 0.70859173,
       0.70940279, 0.71338129, 0.71431467, 0.64318166, 0.68885831,
       0.709177  , 0.71832278, 0.70539045, 0.71188475, 0.7177606 ,
       0.71562837, 0.62754614, 0.69706484, 0.70294687, 0.70052053,
       0.71143352, 0.71089867, 0.71361748, 0.72027803, 0.63250422,
       0.6991704 , 0.70397069, 0.709584  , 0.71592051, 0.7171519 ,
       0.71296258, 0.71434004]),
'split2_test_score': array([0.56513256, 0.65411221, 0.68179874, 0.67778591, 0.67317399,
       0.7023219 , 0.68718805, 0.69485778, 0.60110337, 0.68000582,
       0.68128129, 0.69018925, 0.68827791, 0.68789012, 0.68872232,
       0.69500386, 0.63096979, 0.65944329, 0.70142529, 0.68875127,
       0.69028912, 0.69040592, 0.69777342, 0.70245269, 0.61573215,
       0.66805942, 0.69517954, 0.70428724, 0.69834308, 0.70172687,
       0.69814723, 0.7017106 , 0.64443956, 0.68211024, 0.70082179,
       0.69888313, 0.69191351, 0.70207693, 0.70571188, 0.70710886,
       0.63533121, 0.68116417, 0.69720734, 0.70207888, 0.70605998,
       0.70367594, 0.70847689, 0.7033698 , 0.63129252, 0.68089187,
       0.70451758, 0.70091646, 0.70410376, 0.69519255, 0.70484617,
       0.70815839, 0.61955221, 0.67795899, 0.68658618, 0.70913373,
       0.70453417, 0.69579474, 0.70610195, 0.70334573, 0.65070581,
       0.69700693, 0.69530121, 0.70096494, 0.70606031, 0.71252664,
       0.7106127 , 0.71248792]),
'mean_test_score': array([0.58253947, 0.65194244, 0.6848306 , 0.69082305, 0.69967049,
       0.70164931, 0.70300939, 0.70748472, 0.622439  , 0.67217495,
```

```
       0.70164951, 0.70500959, 0.70748472, 0.622459  , 0.67217495,
       0.68963135, 0.69913704, 0.70296352, 0.70696136, 0.70594069,
       0.71046747, 0.6229864 , 0.68525102, 0.69937023, 0.70031111,
       0.70562042, 0.70889396, 0.71147109, 0.71583642, 0.62366281,
       0.69001391, 0.70268912, 0.71117263, 0.71092899, 0.71381993,
       0.71859654, 0.71710587, 0.65012742, 0.6879725 , 0.71004991,
       0.70917852, 0.71069322, 0.71283279, 0.71993819, 0.7197673 ,
       0.62917137, 0.69693492, 0.7097783 , 0.71051971, 0.71753587,
       0.71871714, 0.71933089, 0.72007186, 0.64858053, 0.69644445,
       0.71314399, 0.71920184, 0.7169567 , 0.71456488, 0.72176146,
       0.72133609, 0.63045417, 0.69460152, 0.70316651, 0.71365487,
       0.71858594, 0.71731493, 0.71982931, 0.72269997, 0.64071492,
       0.70108705, 0.70860567, 0.71279716, 0.71765541, 0.72412338,
       0.7207515 , 0.72454853]),
 'std_test_score': array([0.0159832 , 0.01150422, 0.00374082, 0.01023628, 0.01975122,
       0.00572598, 0.01476674, 0.01151723, 0.01594706, 0.03454325,
       0.01004604, 0.01260416, 0.01616276, 0.01563908, 0.01857293,
       0.01661109, 0.0200937 , 0.02115504, 0.00236197, 0.01128081,
       0.01673709, 0.01412567, 0.0143774 , 0.01358616, 0.00607766,
       0.01774615, 0.01955367, 0.00821006, 0.01310772, 0.01813345,
       0.01596227, 0.01493322, 0.0042016 , 0.00785201, 0.01367194,
       0.01437194, 0.01765058, 0.01022067, 0.01492052, 0.01200511,
       0.01582968, 0.01987025, 0.01193736, 0.00843874, 0.01447314,
       0.01737645, 0.01204704, 0.01649449, 0.01675701, 0.01667835,
       0.00910397, 0.01529878, 0.01727144, 0.01701436, 0.01569863,
       0.0136951 , 0.01029408, 0.01270097, 0.013626  , 0.01296755,
       0.01525285, 0.02069009, 0.01442626, 0.01687547, 0.00753609,
       0.0043305 , 0.01316704, 0.01120336, 0.01024743, 0.01326213,
       0.01271048, 0.01576155]),
 'rank_test_score': array([72, 63, 61, 56, 50, 47, 44, 39, 71, 62, 58, 52, 45, 40, 41, 33, 70,
       60, 51, 49, 42, 37, 28, 21, 69, 57, 46, 29, 30, 23, 14, 19, 64, 59,
       34, 36, 31, 26,  8, 10, 68, 53, 35, 32, 17, 13, 11,  7, 65, 54, 25,
       12, 20, 22,  4,  5, 67, 55, 43, 24, 15, 18,  9,  3, 66, 48, 38, 27,
       16,  2,  6,  1], dtype=int32),
 'split0_train_score': array([0.59950004, 0.72507336, 0.74436852, 0.77229778, 0.77321172,
       0.75622542, 0.77339554, 0.78550258, 0.66771729, 0.7509662 ,
       0.76434484, 0.79382143, 0.80008955, 0.80244277, 0.79824117,
       0.80805656, 0.68948274, 0.77852373, 0.81751782, 0.81378486,
       0.82440773, 0.82638071, 0.82466483, 0.83137897, 0.70440945,
       0.81777357, 0.84632707, 0.83024681, 0.83150049, 0.84893746,
       0.85201089, 0.85448041, 0.74191095, 0.83563234, 0.84486941,
       0.86510702, 0.87042004, 0.86984761, 0.8793031 , 0.87549085,
       0.74833783, 0.86885022, 0.88122786, 0.88164306, 0.89308658,
       0.89563939, 0.89998691, 0.89842734, 0.78023535, 0.88928287,
       0.89005318, 0.9090829 , 0.911241  , 0.90878026, 0.91208614,
       0.920354  , 0.79112324, 0.89842888, 0.90759824, 0.92175797,
       0.92854988, 0.93291967, 0.93047723, 0.93280507, 0.81159448,
       0.90332304, 0.92169453, 0.93742717, 0.94222974, 0.94359524,
       0.94261249, 0.94595749]),
 'split1_train_score': array([0.64471913, 0.71069556, 0.76007821, 0.78826445, 0.79955111,
       0.79412822, 0.79778026, 0.80242744, 0.69657568, 0.77058834,
       0.78295518, 0.79056898, 0.81158632, 0.82359424, 0.81077372,
       0.81763961, 0.68518719, 0.79906967, 0.82816852, 0.82921127,
       0.82959494, 0.83936504, 0.83956025, 0.84252541, 0.72734203,
       0.81473418, 0.84352095, 0.84986055, 0.86114273, 0.85712059,
       0.86687403, 0.86912582, 0.75115005, 0.84920215, 0.86895069,
       0.86835222, 0.88026838, 0.88207986, 0.88669388, 0.89004962,
       0.76194832, 0.8666889 , 0.88417163, 0.89326728, 0.8936308 ,
       0.90447324, 0.9067655 , 0.90522359, 0.79229216, 0.88495083,
       0.90069539, 0.90999917, 0.91563019, 0.92491707, 0.91833602,
       0.92440264, 0.80205913, 0.90062379, 0.90921071, 0.93068096,
       0.92896229, 0.93708418, 0.94291821, 0.94276876, 0.82188439,
       0.91497281, 0.94116484, 0.94386621, 0.94536301, 0.95113161,
       0.95102271, 0.95171609]),
 'split2_train_score': array([0.60869771, 0.71824877, 0.76818767, 0.76726308, 0.75949194,
       0.7798712 , 0.7724999 , 0.77523158, 0.65928635, 0.7565178 ,
       0.77609177, 0.78362938, 0.78563485, 0.78946991, 0.79770509,
       0.80229652, 0.6883342 , 0.77605312, 0.80862693, 0.79968146,
       0.81609209, 0.81114918, 0.82380171, 0.8247185 , 0.71894756,
       0.81645435, 0.83672983, 0.84495193, 0.84126689, 0.84720672,
       0.84760543, 0.84287901, 0.76349454, 0.84298868, 0.84922998,
       0.86591116, 0.85479501, 0.87022246, 0.87309871, 0.87392334,
       0.74669796, 0.86331016, 0.86742601, 0.87163222, 0.8839839 ,
       0.89058689, 0.89485617, 0.8889091 , 0.80670103, 0.8656051 ,
       0.89475141, 0.89920549, 0.90366405, 0.90451209, 0.91034164,
       0.90980202, 0.80996729, 0.89650681, 0.90598849, 0.91329632,
       0.9156692 , 0.92088444, 0.92588026, 0.92344196, 0.82378383,
       0.91267388, 0.92886395, 0.93143342, 0.93503906, 0.94217403,
```

```
        0.91267388, 0.92886395, 0.93143342, 0.93503906, 0.94217403,
        0.9407251 , 0.9413893 ]),
 'mean_train_score': array([0.61763896, 0.7180059 , 0.7575448 , 0.77594177, 0.77741826,
        0.77674161, 0.78122523, 0.78772053, 0.67452644, 0.75935745,
        0.77446393, 0.78933993, 0.79910357, 0.80516897, 0.80223999,
        0.80933089, 0.68766804, 0.78454884, 0.81810442, 0.81422587,
        0.82336492, 0.82563165, 0.82934226, 0.8328743 , 0.71689968,
        0.8163207 , 0.84219262, 0.84168643, 0.8446367 , 0.85108826,
        0.85549678, 0.85549508, 0.75218518, 0.84260773, 0.85435003,
        0.8664568 , 0.86849448, 0.87404998, 0.87969857, 0.87982127,
        0.75232804, 0.8662831 , 0.8776085 , 0.88218086, 0.89023376,
        0.89689984, 0.90053619, 0.89752001, 0.79307618, 0.87994627,
        0.89516666, 0.90609586, 0.91017841, 0.91273647, 0.91358793,
        0.91818622, 0.80104989, 0.89851983, 0.90759915, 0.92191175,
        0.92439379, 0.9302961 , 0.9330919 , 0.93300526, 0.81908757,
        0.91032324, 0.93057444, 0.9375756 , 0.94087727, 0.94563363,
        0.94478677, 0.9463543 ]),
 'std_train_score': array([0.01951326, 0.00587223, 0.00988776, 0.00895259, 0.01662238,
        0.01563119, 0.01171188, 0.01121289, 0.01596657, 0.00825852,
        0.00768434, 0.00425068, 0.01061756, 0.01406394, 0.00603822,
        0.00632827, 0.00181581, 0.0103172 , 0.00798859, 0.01205953,
        0.00556161, 0.01153125, 0.00723379, 0.00734613, 0.00947351,
        0.00124442, 0.00402907, 0.00833356, 0.01233376, 0.00432363,
        0.00824352, 0.01073921, 0.00884181, 0.0055464 , 0.01047658,
        0.00137989, 0.01048822, 0.00568005, 0.00555724, 0.00726079,
        0.00683543, 0.00227985, 0.00729972, 0.00884066, 0.0044249 ,
        0.00573871, 0.00487745, 0.00669119, 0.01081878, 0.0102938 ,
        0.00435458, 0.00488656, 0.0049426 , 0.00878747, 0.00343211,
        0.00615462, 0.00772608, 0.00168198, 0.00131546, 0.00709808,
        0.00617151, 0.00686878, 0.00719723, 0.0078914 , 0.00535486,
        0.00503808, 0.00804021, 0.00507675, 0.00432187, 0.00393072,
        0.00447629, 0.00422522])}
```

In [311]:

```python
# Find best hyper parameter max_depth and min_samples_split
import seaborn as sns; sns.set()
max_scores = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max(
).unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [312]:

```python
# Print params
print(clf.best_estimator_)
print(clf.score(X_train_bow, y_train))
print(clf.score(X_test_bow, y_test))
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
                       criterion='gini', max_depth=10, max_features='auto',
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
```

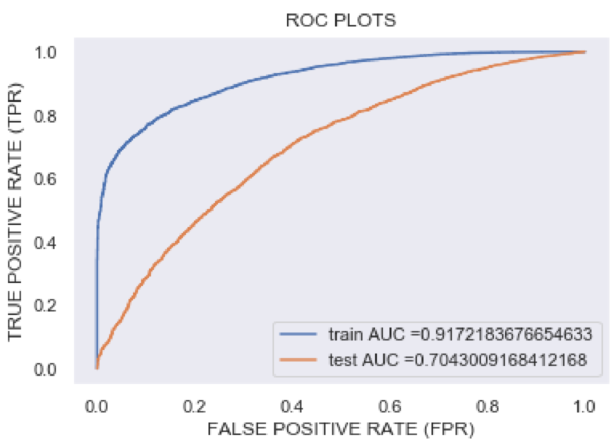
```
                              min_samples_split=2, min_weight_fraction_leaf=0.0,
                              n_estimators=1000, n_jobs=-1, oob_score=False,
                              random_state=None, verbose=0, warm_start=False)
0.9172183676654633
0.7043009168412168
```

In [313]:

```
n_estimators = 1000
max_depth = 10
```

In [314]:

```
%%time

# Create ROC Plot for Test Set
parameters = {'max_depth':[max_depth], 'n_estimators':[n_estimators]}
rf=GridSearchCV(RandomForestClassifier(class_weight='balanced', n_estimators=n_estimators,
max_depth=max_depth, n_jobs=-1), parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score
=True)
rf.fit(X_train_bow, y_train);

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

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```

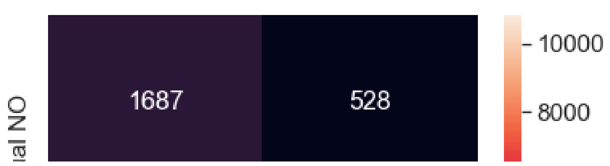


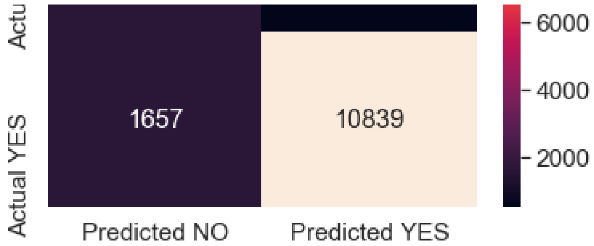

```
CPU times: user 43.5 s, sys: 828 ms, total: 44.4 s
Wall time: 1min 1s
```

In [315]:

```
%%time
get_confusion_matrix(rf,X_train_bow,y_train)
```

```
CPU times: user 13.6 s, sys: 236 ms, total: 13.9 s
Wall time: 8.6 s
```

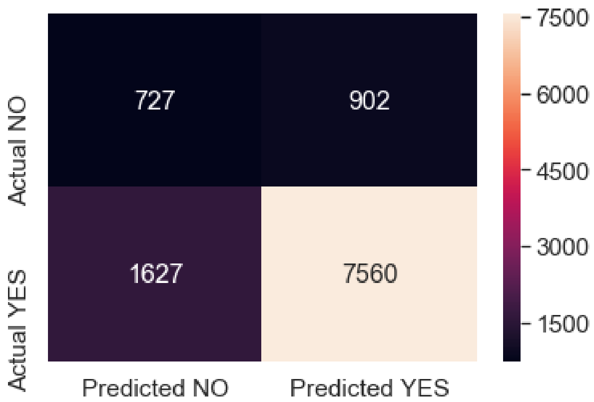




In [316]:

```
%%time
get_confusion_matrix(rf,X_test_bow,y_test)
```

```
CPU times: user 9.67 s, sys: 198 ms, total: 9.87 s
Wall time: 4.58 s
```



In [317]:

```
%%time
# Train a model on above hyperparameter to select best features
rf_feature_selection = RandomForestClassifier(class_weight='balanced', n_estimators=n_estimators,
max_depth=max_depth, n_jobs=-1)
rf_feature_selection.fit(X_train_bow, y_train)
rf_feature_selection.feature_importances_
```

```
CPU times: user 23.7 s, sys: 720 ms, total: 24.5 s
Wall time: 14.7 s
```

Out[317]:

```
array([0.00727337, 0.00735108, 0.00681443, ..., 0.02060005, 0.01528153,
       0.0187616 ])
```

In [318]:

```
# https://stackoverflow.com/questions/44101458/random-forest-feature-importance-chart-using-python

# Calculate feature importances
importances = rf_feature_selection.feature_importances_
# Sort feature importances in descending order
indices_pos = np.argsort(importances)[::-1]

# Sort feature importances in asccending order
indices_neg = np.argsort(importances)[::1]

# Rearrange feature names so they match the sorted feature importances for positive class
names_pos = [feature_names_bow[i] for i in indices_pos]

# Rearrange feature names so they match the sorted feature importances for negative class
names_neg = [feature_names_bow[i] for i in indices_neg]
```

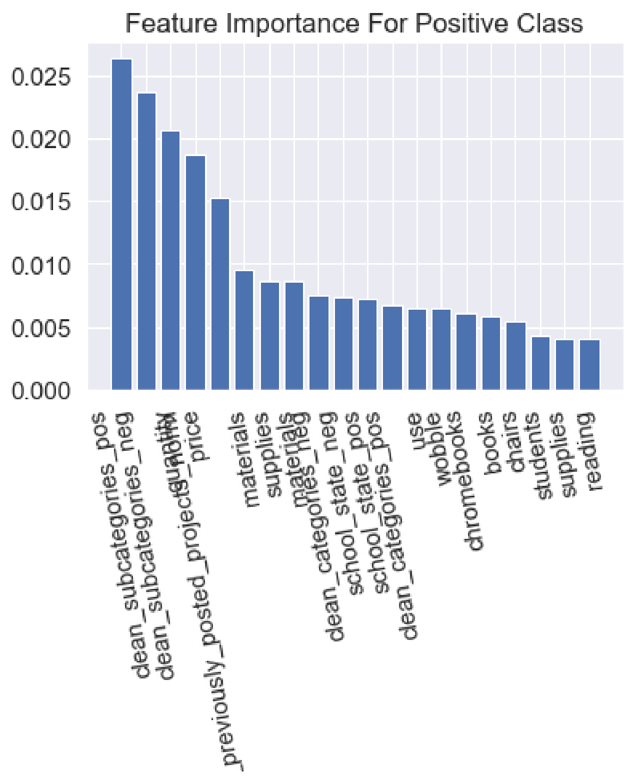
In [319]:

```python
# process for top n features
n = 20
```

In [320]:

```python
# collect n positive feature

# collect top n importance data
top_20_positive_data = []
for i in indices_pos:
    top_20_positive_data.append(importances[i])
    if(len(top_20_positive_data) >= n):
        break;

print(top_20_positive_data)
print("*"*100)

# collect top n importance label
top_20_positive_label = names_pos[0:n]

print(top_20_positive_label)
print("*"*100)

# Barplot: Add bars
plt.bar(range(n), top_20_positive_data)
# Add feature names as x-axis labels
plt.xticks(range(n), top_20_positive_label , rotation=100, fontsize = 15)
# Create plot title
plt.title("Feature Importance For Positive Class")
# Show plot
plt.show()
```

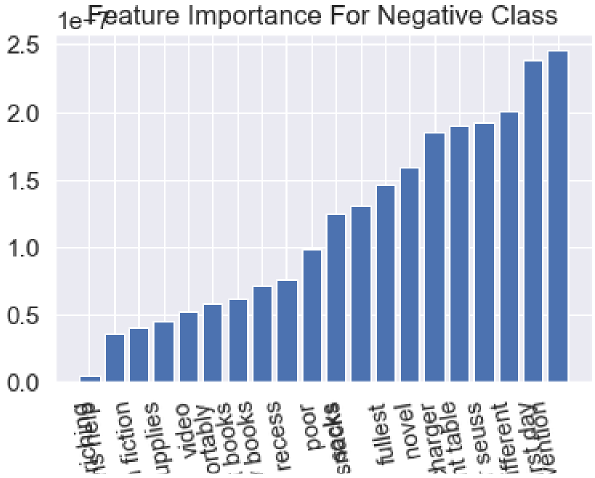
```
[0.026331933336934253, 0.02367404044767679, 0.020600053746364436, 0.018761596183043544,
0.015281527149479437, 0.00950312097978943, 0.008707417849634224, 0.008680095032457615,
0.00756095544512088, 0.007351075024355123, 0.007273367841931229, 0.006814429376444725,
0.006535722830858892, 0.006458334187670129, 0.006106948856383575, 0.005915810175130168,
0.005507670244642196, 0.004302400424120888, 0.004135010950606783, 0.004073100735093452]
****************************************************************************************************

['clean_subcategories_pos', 'clean_subcategories_neg', 'quantity', 'price',
'teacher_number_of_previously_posted_projects_norm', 'materials', 'supplies', 'materials',
'clean_categories_neg', 'school_state_neg', 'school_state_pos', 'clean_categories_pos', 'use', 'wo
bble', 'chromebooks', 'books', 'chairs', 'students', 'supplies', 'reading']
****************************************************************************************************
```

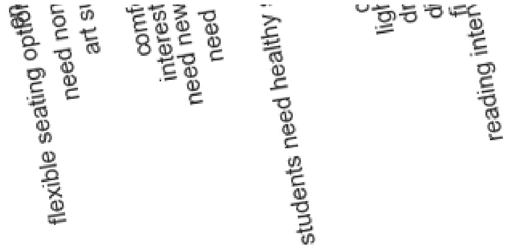


In [321]:

```
# collect n negative feature

# collect top n importance data
top_20_negative_data = []
collect_neg_indices = []
for i in indices_neg:
    if(importances[i] > 0):
        top_20_negative_data.append(importances[i])
        collect_neg_indices.append(i)
    if(len(top_20_negative_data) >= n):
        break;

print(top_20_negative_data)
print("*"*100)

# collect top n importance data
top_20_negative_label = []
for i in collect_neg_indices:
    top_20_negative_label.append(names_neg[i])

print(top_20_negative_label)
print("*"*100)

# Barplot: Add bars
plt.bar(range(n), top_20_negative_data)
# Add feature names as x-axis labels
plt.xticks(range(n), top_20_negative_label , rotation=100, fontsize = 15)
# Create plot title
plt.title("Feature Importance For Negative Class")
# Show plot
plt.show()
```

```
[4.724030407280267e-09, 3.5459442430600536e-08, 4.0799591018504394e-08, 4.530458336405595e-08,
5.2975635886083995e-08, 5.8481237432073735e-08, 6.17220916132652e-08, 7.16968200671659e-08,
7.566594213464851e-08, 9.865974997549917e-08, 1.246170478127686e-07, 1.3133319132394543e-07,
1.4675783834305262e-07, 1.589213448562151e-07, 1.8489239191015398e-07, 1.8999562174418277e-07,
1.923414231625342e-07, 2.0054795933282652e-07, 2.392824362049235e-07, 2.4565678820454505e-07]
****************************************************************************************************

['enriching', 'flexible seating options help', 'need non fiction', 'art supplies', 'video',
'comfortably', 'interest books', 'need new books', 'need recess', 'poor', 'socks', 'students need
healthy snacks', 'fullest', 'novel', 'charger', 'light table', 'dr seuss', 'different', 'first day
', 'reading intervention']
****************************************************************************************************
```





2.4.2 Applying Random Forests on TFIDF, SET 2

In [322]:

```
%%time
# Please write all the code with proper documentation
# Prepare data for TFIDF
X_train_tfidf = hstack((X_train_school_state_pos, X_train_school_state_neg,
X_train_clean_categories_pos, X_train_clean_categories_neg, X_train_clean_subcategories_pos,
X_train_clean_subcategories_neg, X_train_project_grade_category_pos,
X_train_project_grade_category_neg, X_train_teacher_prefix_pos, X_train_teacher_prefix_neg,
X_train_title_tfidf, X_train_essay_tfidf, X_train_project_resource_summary_tfidf,
X_train_quantity_norm, X_train_teacher_number_of_previously_posted_projects_norm,
X_train_price_norm)).tocsr()
X_cv_tfidf = hstack((X_cv_school_state_pos, X_cv_school_state_neg, X_cv_clean_categories_pos,
X_cv_clean_categories_neg, X_cv_clean_subcategories_pos, X_cv_clean_subcategories_neg,
X_cv_project_grade_category_pos, X_cv_project_grade_category_neg, X_cv_teacher_prefix_pos,
X_cv_teacher_prefix_neg, X_cv_title_tfidf, X_cv_essay_tfidf, X_cv_project_resource_summary_tfidf,
X_cv_quantity_norm, X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocs
r()
X_test_tfidf = hstack((X_test_school_state_pos, X_test_school_state_neg,
X_test_clean_categories_pos, X_test_clean_categories_neg, X_test_clean_subcategories_pos,
X_test_clean_subcategories_neg, X_test_project_grade_category_pos,
X_test_project_grade_category_neg, X_test_teacher_prefix_pos, X_test_teacher_prefix_neg,
X_test_title_tfidf, X_test_essay_tfidf, X_test_project_resource_summary_tfidf,
X_test_quantity_norm, X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm)
).tocsr()

print(X_train_tfidf.shape)
print(X_cv_tfidf.shape)
print(X_test_tfidf.shape)


import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV

rf = RandomForestClassifier(class_weight='balanced', n_jobs=-1)
parameters = {'n_estimators':[10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6, 7
, 8, 9, 10]}
clf=GridSearchCV(rf, parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score=True)
clf.fit(X_train_tfidf, y_train)
```

```
(14711, 10071)
(7247, 10071)
(10816, 10071)
CPU times: user 26.9 s, sys: 512 ms, total: 27.4 s
Wall time: 10min 49s
```

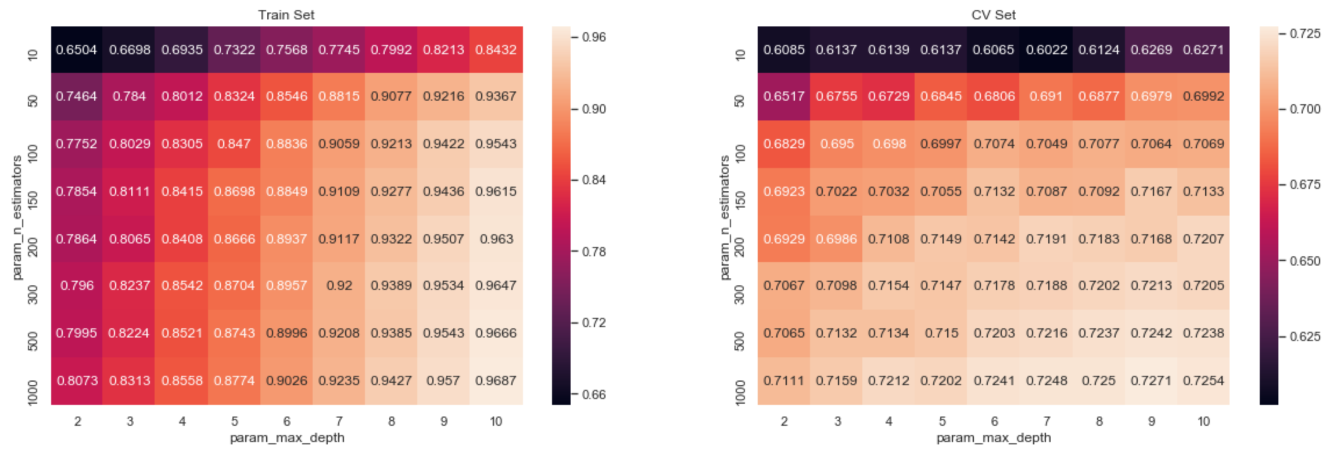
Out[322]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True,
                                              class_weight='balanced',
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=-1
```

```
                                                  n_estimators='warn', n_jobs=-1,
                                                  oob_score=False,
                                                  random_state=None, verbose=0,
                                                  warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [10, 50, 100, 150, 200, 300, 500,
                                          1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [323]:

```
# Find best hyper parameter max_depth and min_samples_split
import seaborn as sns; sns.set()
max_scores = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max(
).unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [324]:

```
# Print params
print(clf.best_estimator_)
print(clf.score(X_train_tfidf, y_train))
print(clf.score(X_test_tfidf, y_test))
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
                       criterion='gini', max_depth=9, max_features='auto',
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=1000, n_jobs=-1, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
0.9299142226641194
0.7091140809841328
```

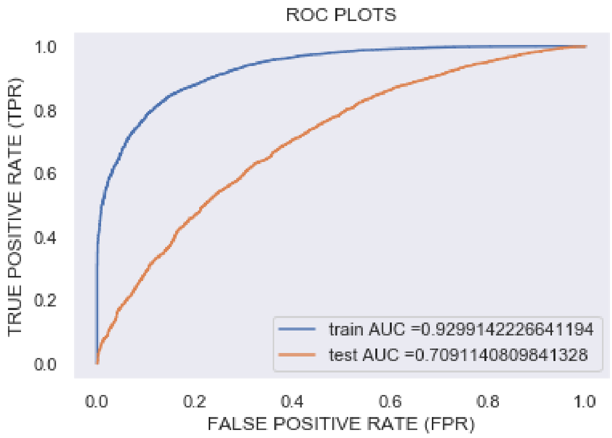
In [325]:

```
n_estimators = 1000
max_depth = 10
```

In [326]:

```
%%time

# Create ROC Plot for Test Set
parameters = {'max_depth':[max_depth], 'n_estimators':[n_estimators]}
rf=GridSearchCV(RandomForestClassifier(class_weight='balanced', n_estimators=n_estimators,
max_depth=max_depth, n_jobs=-1), parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score
=True)
```

```
rf.fit(X_train_tfidf, y_train);

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

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```

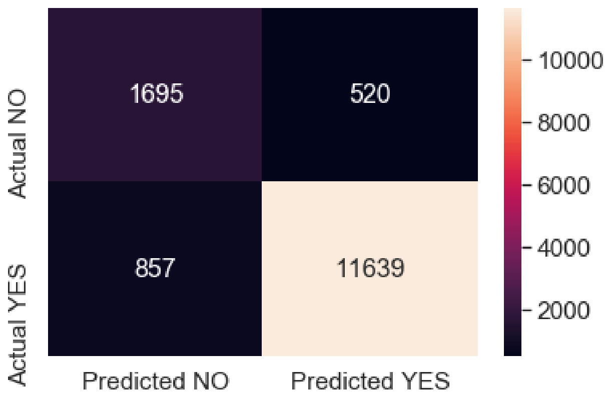



```
CPU times: user 40.2 s, sys: 726 ms, total: 40.9 s
Wall time: 46.8 s
```

In [327]:

```
%%time
get_confusion_matrix(rf,X_train_tfidf,y_train)
```

```
CPU times: user 8.24 s, sys: 121 ms, total: 8.36 s
Wall time: 3.25 s
```



In [328]:

```
%%time
get_confusion_matrix(rf,X_test_tfidf,y_test)
```

```
CPU times: user 6.38 s, sys: 114 ms, total: 6.49 s
Wall time: 2.86 s
```

In [329]:

```
%%time
# Train a model on above hyperparameter to select best features
rf_feature_selection = RandomForestClassifier(class_weight='balanced', n_estimators=n_estimators,
max_depth=max_depth, n_jobs=-1)
rf_feature_selection.fit(X_train_tfidf, y_train)
rf_feature_selection.feature_importances_
```

```
CPU times: user 26.5 s, sys: 476 ms, total: 27 s
Wall time: 10.7 s
```

Out[329]:

```
array([0.00564628, 0.00633854, 0.00641493, ..., 0.01618556, 0.01078699,
       0.01696141])
```

In [330]:

```
# https://stackoverflow.com/questions/44101458/random-forest-feature-importance-chart-using-python

# Calculate feature importances
importances = rf_feature_selection.feature_importances_
# Sort feature importances in descending order
indices_pos = np.argsort(importances)[::-1]

# Sort feature importances in asccending order
indices_neg = np.argsort(importances)[::1]

# Rearrange feature names so they match the sorted feature importances for positive class
names_pos = [feature_names_tfidf[i] for i in indices_pos]

# Rearrange feature names so they match the sorted feature importances for negative class
names_neg = [feature_names_tfidf[i] for i in indices_neg]
```

In [331]:

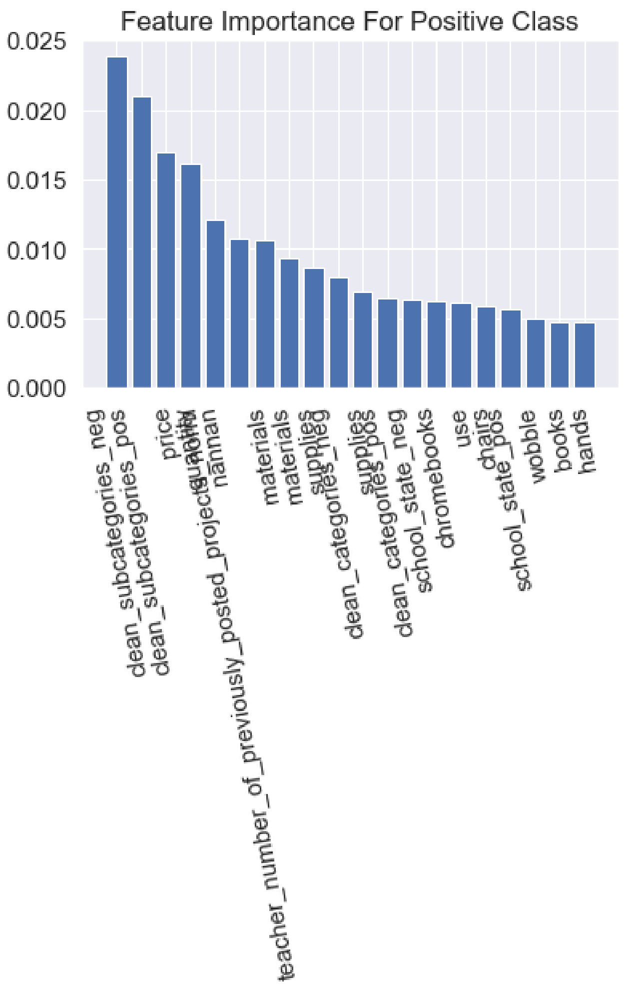
```
# process for top n features
n = 20
```

In [332]:

```
# collect n positive feature

# collect top n importance data
top_20_positive_data = []
for i in indices_pos:
    top_20_positive_data.append(importances[i])
    if(len(top_20_positive_data) >= n):
        break;

print(top_20_positive_data)
print("*"*100)

# collect top n importance label
top_20_positive_label = names_pos[0:n]

print(top_20_positive_label)
```

```
print("*"*100)

# Barplot: Add bars
plt.bar(range(n), top_20_positive_data)
# Add feature names as x-axis labels
plt.xticks(range(n), top_20_positive_label , rotation=100, fontsize = 15)
# Create plot title
plt.title("Feature Importance For Positive Class")
# Show plot
plt.show()
```

```
[0.02388569348456271, 0.02098447283244447, 0.01696141065903579, 0.016185561553855934,
0.012100496674058664, 0.010786992436094966, 0.010655334993992443, 0.009392511729585237,
0.008695271612570367, 0.007934021419129876, 0.006905389413091942, 0.00641493065688277,
0.006338544754344597, 0.006234325382121456, 0.006098518296696127, 0.005895154035009356,
0.005646276387120999, 0.005008126158841348, 0.004790594953613698, 0.004783900333657298]
****************************************************************************************************

['clean_subcategories_neg', 'clean_subcategories_pos', 'price', 'quantity', 'nannan',
'teacher_number_of_previously_posted_projects_norm', 'materials', 'materials', 'supplies',
'clean_categories_neg', 'supplies', 'clean_categories_pos', 'school_state_neg', 'chromebooks', 'us
e', 'chairs', 'school_state_pos', 'wobble', 'books', 'hands']
****************************************************************************************************
```

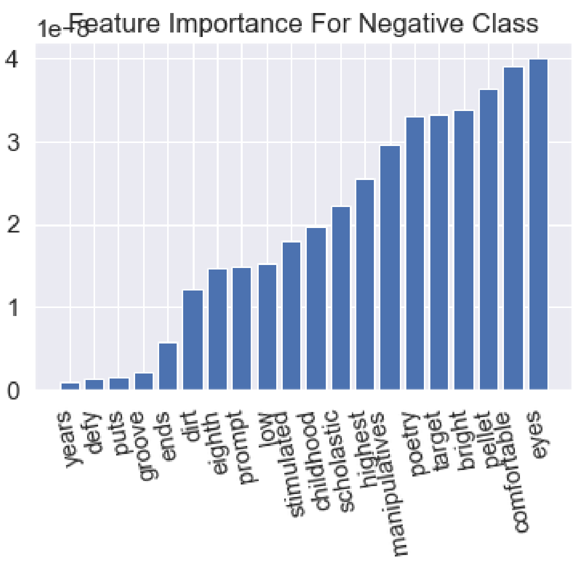


In [333]:

```
# collect n negative feature

# collect top n importance data
top_20_negative_data = []
collect_neg_indices = []
for i in indices_neg:
    if(importances[i] > 0):
        top_20_negative_data.append(importances[i])
        collect_neg_indices.append(i)
    if(len(top_20_negative_data) >= n):
```

```
        break;

print(top_20_negative_data)
print("*"*100)


# collect top n importance data
top_20_negative_label = []
for i in collect_neg_indices:
    top_20_negative_label.append(names_neg[i])

print(top_20_negative_label)
print("*"*100)

# Barplot: Add bars
plt.bar(range(n), top_20_negative_data)
# Add feature names as x-axis labels
plt.xticks(range(n), top_20_negative_label , rotation=100, fontsize = 15)
# Create plot title
plt.title("Feature Importance For Negative Class")
# Show plot
plt.show()
```

```
[1.0100386983508363e-09, 1.4496355263582026e-09, 1.6814544047211648e-09, 2.1255451547281227e-09,
5.802882451658276e-09, 1.2254253098773021e-08, 1.4722914953300233e-08, 1.4973170716250735e-08,
1.5254853225623342e-08, 1.794402183364072e-08, 1.9791987523195938e-08, 2.2276601282469166e-08,
2.5460615113996306e-08, 2.9529125076928686e-08, 3.2970829844695646e-08, 3.318136749488672e-08,
3.37809678788238e-08, 3.639388399454989e-08, 3.89090287699897e-08, 3.990171334115558e-08]
****************************************************************************************************

['years', 'defy', 'puts', 'groove', 'ends', 'dirt', 'eighth', 'prompt', 'low', 'stimulated',
'childhood', 'scholastic', 'highest', 'manipulatives', 'poetry', 'target', 'bright', 'pellet', 'co
mfortable', 'eyes']
****************************************************************************************************
```



2.4.3 Applying Random Forests on AVG W2V, SET 3

In [101]:

```
%%time
# Please write all the code with proper documentation
# Prepare data for AVGW2V
X_train_avgw2v = hstack((X_train_school_state_pos, X_train_school_state_neg,
X_train_clean_categories_pos, X_train_clean_categories_neg, X_train_clean_subcategories_pos,
X_train_clean_subcategories_neg, X_train_project_grade_category_pos,
X_train_project_grade_category_neg, X_train_teacher_prefix_pos, X_train_teacher_prefix_neg,
avg_w2v_vectors_text_train, avg_w2v_vectors_title_train,
avg_w2v_vectors_project_resource_summary_train, X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm)).tocsr()
X_cv_avgw2v = hstack((X_cv_school_state_pos, X_cv_school_state_neg, X_cv_clean_categories_pos,
X_cv_clean_categories_neg, X_cv_clean_subcategories_pos, X_cv_clean_subcategories_neg,
```

```
X_cv_project_grade_category_pos, X_cv_project_grade_category_neg, X_cv_teacher_prefix_pos,
X_cv_teacher_prefix_neg, avg_w2v_vectors_text_cv, avg_w2v_vectors_title_cv,
avg_w2v_vectors_project_resource_summary_cv, X_cv_quantity_norm,
X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocsr()
X_test_avgw2v = hstack((X_test_school_state_pos, X_test_school_state_neg,
X_test_clean_categories_pos, X_test_clean_categories_neg, X_test_clean_subcategories_pos,
X_test_clean_subcategories_neg, X_test_project_grade_category_pos,
X_test_project_grade_category_neg, X_test_teacher_prefix_pos, X_test_teacher_prefix_neg,
avg_w2v_vectors_text_test, avg_w2v_vectors_title_test,
avg_w2v_vectors_project_resource_summary_test, X_test_quantity_norm,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm)).tocsr()


print(X_train_avgw2v.shape)
print(X_cv_avgw2v.shape)
print(X_test_avgw2v.shape)


import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV

rf = RandomForestClassifier(class_weight='balanced', n_jobs=-1)
parameters = {'n_estimators':[10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6, 7
, 8, 9, 10]}
clf=GridSearchCV(rf, parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score=True)
clf.fit(X_train_avgw2v, y_train)
```

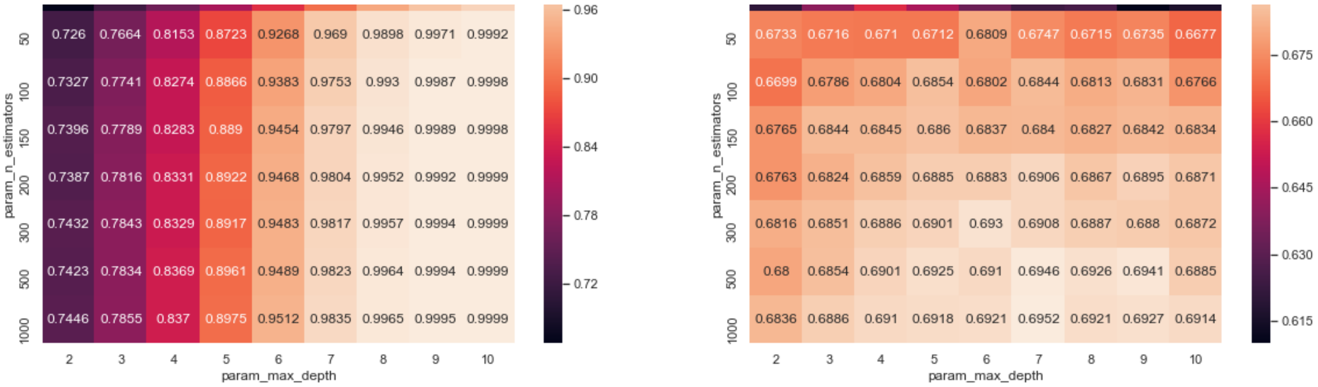
```
(14711, 913)
(7247, 913)
(10816, 913)
CPU times: user 7min 2s, sys: 1.7 s, total: 7min 4s
Wall time: 8h 58min 30s
```

Out[101]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True,
                                              class_weight='balanced',
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=-1,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [10, 50, 100, 150, 200, 300, 500,
                                          1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [102]:

```
# Find best hyper parameter max_depth and min_samples_split
import seaborn as sns; sns.set()
max_scores = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max(
).unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [103]:

```
# Print params
print(clf.best_estimator_)
print(clf.score(X_train_avgw2v, y_train))
print(clf.score(X_test_avgw2v, y_test))
```

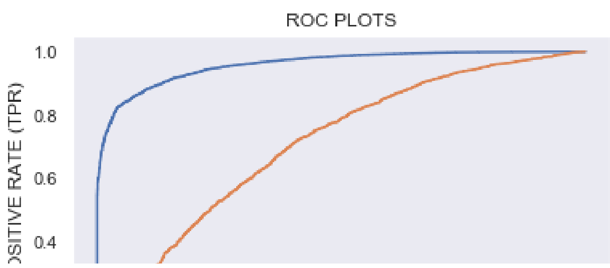
```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
                       criterion='gini', max_depth=7, max_features='auto',
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=1000, n_jobs=-1, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
0.9592071986444379
0.7023769060203968
```

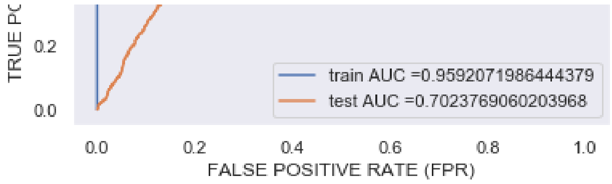
In [105]:

```
n_estimators = 1000
max_depth = 7
```

In [106]:

```
%%time

# Create ROC Plot for Test Set
parameters = {'max_depth':[max_depth], 'n_estimators':[n_estimators]}
rf=GridSearchCV(RandomForestClassifier(class_weight='balanced', n_estimators=n_estimators,
max_depth=max_depth, n_jobs=-1), parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score
=True)
rf.fit(X_train_avgw2v, y_train);

y_train_pred = clf.predict_proba(X_train_avgw2v)[:,1]
y_test_pred = clf.predict_proba(X_test_avgw2v)[:,1]

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```



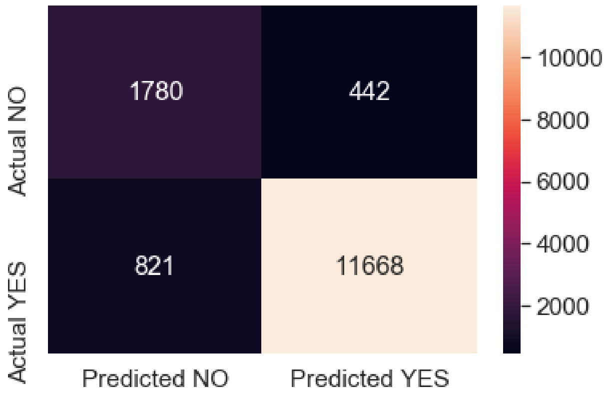



```
CPU times: user 8min 1s, sys: 2.5 s, total: 8min 3s
Wall time: 6min 34s
```

In [107]:

```
%%time
get_confusion_matrix(rf,X_train_avgw2v,y_train)
```

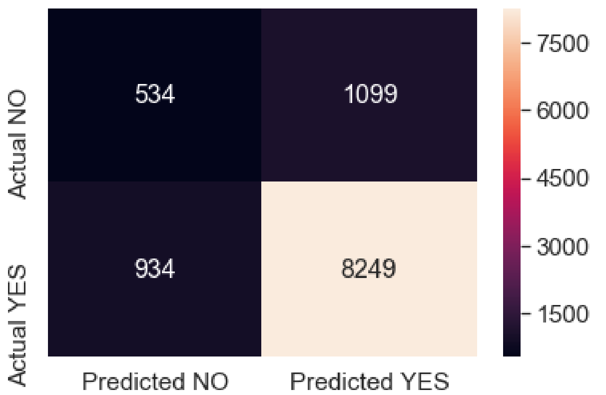
```
CPU times: user 26.6 s, sys: 266 ms, total: 26.9 s
Wall time: 7.99 s
```



In [108]:

```
%%time
get_confusion_matrix(rf,X_test_avgw2v,y_test)
```

```
CPU times: user 19.2 s, sys: 211 ms, total: 19.4 s
Wall time: 5.74 s
```



2.4.4 Applying Random Forests on TFIDF W2V, SET 4

In [132]:

```
%%time
# Please write all the code with proper documentation
# Prepare data for TFIDFW2V
X_train_tfidfw2v = hstack((X_train_school_state_pos, X_train_school_state_neg,  X_train_clean_categ
ories_pos, X_train_clean_categories_neg, X_train_clean_subcategories_pos,
X_train_clean_subcategories_neg, X_train_project_grade_category_pos,
X_train_project_grade_category_neg, X_train_teacher_prefix_pos, X_train_teacher_prefix_neg,
```

```
tfidf_w2v_vectors_text_train, tfidf_w2v_vectors_title_train,
tfidf_w2v_vectors_project_resource_summary_train, X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm)).tocsr()
X_cv_tfidfw2v = hstack((X_cv_school_state_pos, X_cv_school_state_neg, X_cv_clean_categories_pos,
X_cv_clean_categories_neg, X_cv_clean_subcategories_pos, X_cv_clean_subcategories_neg,
X_cv_project_grade_category_pos, X_cv_project_grade_category_neg, X_cv_teacher_prefix_pos,
X_cv_teacher_prefix_neg, tfidf_w2v_vectors_text_cv, tfidf_w2v_vectors_title_cv,
tfidf_w2v_vectors_project_resource_summary_cv, X_cv_quantity_norm,
X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocsr()
X_test_tfidfw2v = hstack((X_test_school_state_pos, X_test_school_state_neg,
X_test_clean_categories_pos, X_test_clean_categories_neg, X_test_clean_subcategories_pos,
X_test_clean_subcategories_neg, X_test_project_grade_category_pos,
X_test_project_grade_category_neg, X_test_teacher_prefix_pos, X_test_teacher_prefix_neg,
tfidf_w2v_vectors_text_test, tfidf_w2v_vectors_title_test,
tfidf_w2v_vectors_project_resource_summary_test, X_test_quantity_norm,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm)).tocsr()


print(X_train_tfidfw2v.shape)
print(X_cv_tfidfw2v.shape)
print(X_test_tfidfw2v.shape)

import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV

rf = RandomForestClassifier(class_weight='balanced', n_jobs=-1)
parameters = {'n_estimators':[10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6, 7
, 8, 9, 10]}
clf=GridSearchCV(rf, parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score=True)
clf.fit(X_train_tfidfw2v, y_train)
```

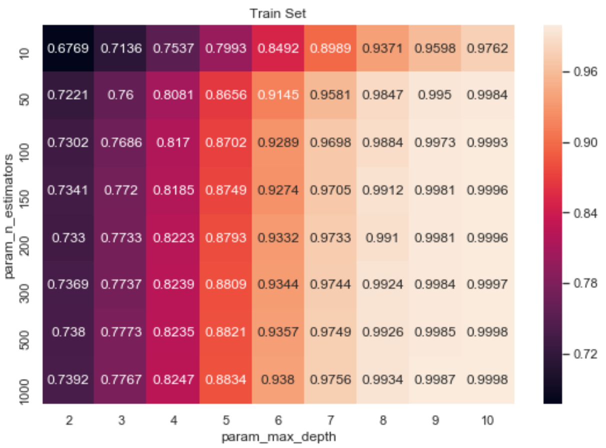
```
(14711, 913)
(7247, 913)
(10816, 913)
CPU times: user 5min 27s, sys: 2.32 s, total: 5min 30s
Wall time: 3h 33min 1s
```

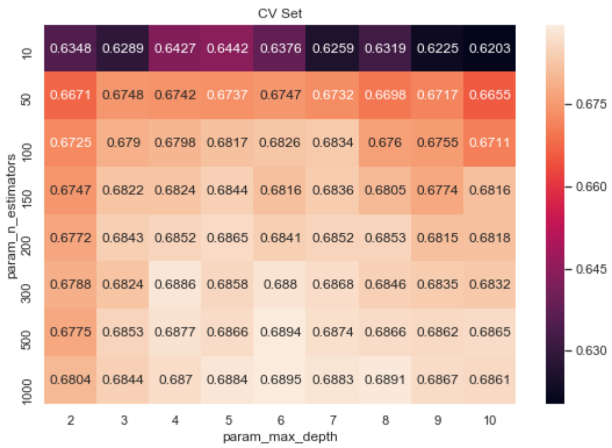
Out[132]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True,
                                              class_weight='balanced',
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=-1,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [10, 50, 100, 150, 200, 300, 500,
                                          1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [133]:

```
# Find best hyper parameter max_depth and min_samples_split
import seaborn as sns; sns.set()
max_scores = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max(
).unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```





In [134]:

```
# Print params
print(clf.best_estimator_)
print(clf.score(X_train_tfidfw2v, y_train))
print(clf.score(X_test_tfidfw2v, y_test))
```

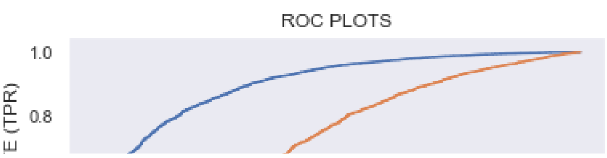
```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
                       criterion='gini', max_depth=6, max_features='auto',
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=1000, n_jobs=-1, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
0.8979032781971448
0.6924932976407655
```

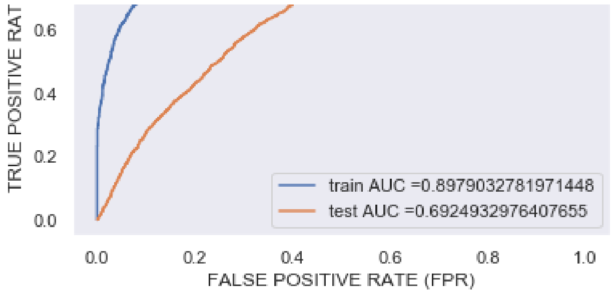
In [135]:

```
n_estimators = 1000
max_depth = 6
```

In [136]:

```
%%time

# Create ROC Plot for Test Set
parameters = {'max_depth':[max_depth], 'n_estimators':[n_estimators]}
rf=GridSearchCV(RandomForestClassifier(class_weight='balanced', n_estimators=n_estimators,
max_depth=max_depth, n_jobs=-1), parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score
=True)
rf.fit(X_train_tfidfw2v, y_train);

y_train_pred = clf.predict_proba(X_train_tfidfw2v)[:,1]
y_test_pred = clf.predict_proba(X_test_tfidfw2v)[:,1]

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```



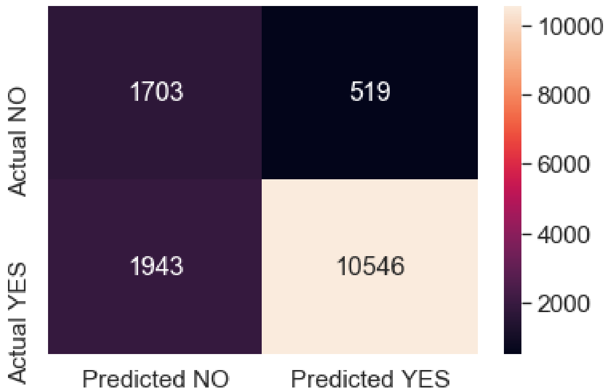



```
CPU times: user 6min 8s, sys: 1.14 s, total: 6min 10s
Wall time: 5min 16s
```

In [137]:

```
%%time
get_confusion_matrix(rf,X_train_tfidfw2v,y_train)
```

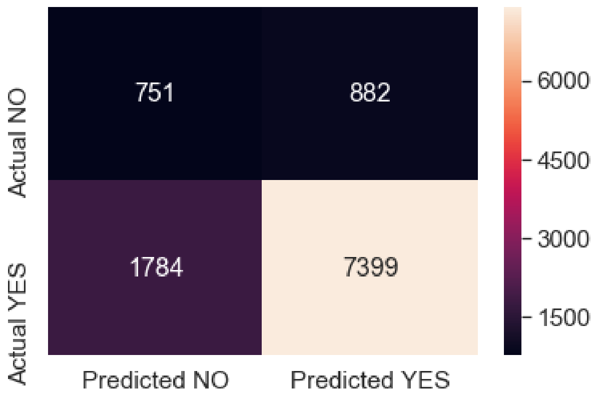
```
CPU times: user 23 s, sys: 602 ms, total: 23.6 s
Wall time: 7.17 s
```



In [138]:

```
%%time
get_confusion_matrix(rf,X_test_tfidfw2v,y_test)
```

```
CPU times: user 16.3 s, sys: 127 ms, total: 16.4 s
Wall time: 4.48 s
```



2.5 Applying GBDT

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

2.5.1 Applying XGBOOST on BOW, SET 1

In [109]:

```
%%time
# Please write all the code with proper documentation

# Prepare data for BOW
X_train_bow = hstack((X_train_school_state_pos, X_train_school_state_neg,
X_train_clean_categories_pos, X_train_clean_categories_neg, X_train_clean_subcategories_pos,
X_train_clean_subcategories_neg, X_train_project_grade_category_pos,
X_train_project_grade_category_neg, X_train_teacher_prefix_pos, X_train_teacher_prefix_neg,
X_train_title_bow, X_train_essay_bow, X_train_project_resource_summary_bow, X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm)).tocsr()
X_cv_bow = hstack((X_cv_school_state_pos, X_cv_school_state_neg, X_cv_clean_categories_pos,
X_cv_clean_categories_neg, X_cv_clean_subcategories_pos, X_cv_clean_subcategories_neg,
X_cv_project_grade_category_pos, X_cv_project_grade_category_neg, X_cv_teacher_prefix_pos,
X_cv_teacher_prefix_neg, X_cv_title_bow, X_cv_essay_bow, X_cv_project_resource_summary_bow,
X_cv_quantity_norm, X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocs
r()
X_test_bow = hstack((X_test_school_state_pos, X_test_school_state_neg, X_test_clean_categories_pos
, X_test_clean_categories_neg, X_test_clean_subcategories_pos, X_test_clean_subcategories_neg, X_t
est_project_grade_category_pos,  X_test_project_grade_category_neg, X_test_teacher_prefix_pos, X_t
est_teacher_prefix_neg, X_test_title_bow, X_test_essay_bow, X_test_project_resource_summary_bow, X
_test_quantity_norm, X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm))
.tocsr()

print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)

import matplotlib.pyplot as plt
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV

xgb = XGBClassifier(class_weight='balanced', n_jobs=-1)
parameters = {'n_estimators':[10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6, 7
, 8, 9, 10]}
clf=GridSearchCV(xgb, parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score=True)
clf.fit(X_train_bow, y_train)
```

```
(14711, 10297) (14711,)
(7247, 10297) (7247,)
(10816, 10297) (10816,)
CPU times: user 57.1 s, sys: 582 ms, total: 57.7 s
Wall time: 2h 23min 51s
```

Out[109]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     class_weight='balanced',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=-1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [10, 50, 100, 150, 200, 300, 500,
                                          1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [110]:

```
# print results
clf.cv_results_
```

Out[110]:

```
{'mean_fit_time': array([   2.79524485,    9.43027218,   16.1352423 ,   22.33804393,
          30.3830452 ,   46.51209712,   79.81964231,  160.01052785,
           3.51865443,   12.71859105,   20.17203482,   30.73062094,
          45.76393096,   61.84292674,  112.35918673,  220.19645007,
           3.22812931,   14.10237948,   28.95465748,   43.97654478,
          60.53410061,   84.12143501,  140.36739564,  310.88346863,
           4.75534431,   20.36359096,   40.32456072,   56.05845006,
          74.82564855,  120.46896005,  190.72452331,  376.97966766,
           5.337744  ,   21.84740504,   50.19408735,   65.1116937 ,
          90.90883255,  139.8322626 ,  220.12863731,  686.37745365,
           5.82620207,   26.77927597,   54.03178104,  327.46429666,
         599.71170433,  137.82622568,  218.98402007,  436.53152227,
           6.00144076,   26.13107403,   52.16250475,   75.72499569,
          96.38074795,  159.58858991,  264.78227202,  514.84462142,
           7.81324697,   31.306041  ,   58.1673487 ,   85.18502744,
         114.28311698,  175.45543496,  287.90149951, 1010.950996  ,
           7.26426832,   33.19498499,   63.13991769,   94.02897994,
        1490.63741779,  173.30822619,  298.24340638,  465.73498829]),
 'std_fit_time': array([3.97627722e-02, 6.43154060e-02, 7.76635743e-01, 1.48273923e-01,
        5.89730316e-01, 7.66073703e-01, 1.05232813e+00, 2.17574003e+00,
        1.73217743e-01, 9.74821702e-01, 2.06681789e-01, 9.35245427e-01,
        1.01063561e+00, 1.10914032e+00, 2.96907854e+00, 1.42596741e+00,
        8.96848718e-02, 1.15662850e+00, 2.43751637e-01, 1.55421138e-01,
        9.24828548e-01, 3.61119525e+00, 2.30677793e+00, 1.69635397e+00,
        1.91174765e-01, 1.50753743e+00, 2.38855678e+00, 5.47605794e-01,
        2.06382617e+00, 3.47289144e-01, 1.80059950e+00, 1.16665440e+00,
        4.24792915e-01, 1.30360445e+00, 7.53351882e-01, 3.61941675e+00,
        8.92724461e-01, 3.35756896e-01, 4.80894622e+00, 3.58776133e+02,
        1.56622195e-01, 8.44106653e-01, 1.94599150e+00, 3.60416380e+02,
        3.61386414e+02, 8.13232259e-01, 4.25488116e+00, 1.03473080e+00,
        2.34756709e-01, 1.58765464e+00, 2.22455290e+00, 4.05119322e+00,
        2.44793812e-01, 5.31165841e+00, 2.11078921e+00, 2.65572565e+00,
        8.63632582e-02, 2.12606577e+00, 4.39028182e-01, 1.57635923e-01,
        4.71378772e-01, 1.44722493e+00, 4.03831270e+00, 6.44657340e+02,
        5.75004373e-02, 4.24347053e-01, 2.18914597e-01, 3.65190129e-01,
        1.85708140e+00, 2.30535947e+00, 1.31084953e+00, 4.85325318e+01]),
 'mean_score_time': array([0.7206231 , 0.45870733, 0.36908603, 0.35308576, 0.37181052,
        0.3693707 , 0.41086968, 0.46883933, 0.45402988, 0.42309697,
        0.39210645, 0.38983107, 0.40826368, 0.40367969, 0.51116474,
        0.58821424, 0.35328341, 0.39114642, 0.43212398, 0.41899029,
        0.45055699, 0.41010992, 0.50527334, 0.6873428 , 0.43358374,
        0.50061067, 0.45588263, 0.46012831, 0.65337213, 0.50601029,
        0.56450605, 0.82946126, 0.5232319 , 0.40886656, 0.48662003,
        0.4976689 , 0.51156878, 0.60326441, 0.569086  , 0.67761668,
        0.47697703, 0.43832755, 0.49272339, 0.44393611, 0.59113765,
        0.45203797, 0.55821888, 0.70487094, 0.36044494, 0.38499093,
        0.46944507, 0.4153959 , 0.46493268, 0.7120049 , 0.72118632,
        0.83211883, 0.43160629, 0.4204673 , 0.44555759, 0.48078227,
        0.49497771, 0.58046405, 0.64363416, 1.14027317, 0.38769825,
        0.43205539, 0.47579741, 0.49978272, 0.46956944, 0.56292256,
        0.65573812, 0.68981647]),
 'std_score_time': array([0.02886101, 0.03663645, 0.00820109, 0.00277609, 0.01811138,
        0.0165542 , 0.01133442, 0.04529706, 0.07805086, 0.05448809,
        0.02872963, 0.02281892, 0.01940062, 0.02538141, 0.0196132 ,
        0.03991367, 0.00783954, 0.03237177, 0.01323294, 0.00603983,
        0.01276482, 0.01230056, 0.02490977, 0.02963749, 0.01528142,
        0.087896  , 0.03529853, 0.0079108 , 0.28216515, 0.02686692,
        0.0651535 , 0.14635523, 0.11037559, 0.00783857, 0.07812222,
        0.0379327 , 0.03146793, 0.01584939, 0.01832179, 0.06002986,
        0.04772226, 0.00425862, 0.05884968, 0.0402805 , 0.14185854,
        0.01330622, 0.04374757, 0.03661992, 0.01185891, 0.02470977,
        0.07541617, 0.0013144 , 0.04110327, 0.16354822, 0.09466761,
        0.01549273, 0.04247916, 0.0067959 , 0.00640373, 0.01328012,
        0.01089727, 0.03749297, 0.01284508, 0.31809567, 0.00657922,
        0.01798848, 0.02800506, 0.0223206 , 0.00799706, 0.02315396,
        0.04845511, 0.07633299]),
 'param_max_depth': masked_array(data=[2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4,
                    4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6,
                    6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8,
                    8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 10,
                    10, 10],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
```

```
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[10, 50, 100, 150, 200, 300, 500, 1000, 10, 50, 100,
                    150, 200, 300, 500, 1000, 10, 50, 100, 150, 200, 300,
                    500, 1000, 10, 50, 100, 150, 200, 300, 500, 1000, 10,
                    50, 100, 150, 200, 300, 500, 1000, 10, 50, 100, 150,
                    200, 300, 500, 1000, 10, 50, 100, 150, 200, 300, 500,
                    1000, 10, 50, 100, 150, 200, 300, 500, 1000, 10, 50,
                    100, 150, 200, 300, 500, 1000],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 2, 'n_estimators': 10},
  {'max_depth': 2, 'n_estimators': 50},
  {'max_depth': 2, 'n_estimators': 100},
  {'max_depth': 2, 'n_estimators': 150},
  {'max_depth': 2, 'n_estimators': 200},
  {'max_depth': 2, 'n_estimators': 300},
  {'max_depth': 2, 'n_estimators': 500},
  {'max_depth': 2, 'n_estimators': 1000},
  {'max_depth': 3, 'n_estimators': 10},
  {'max_depth': 3, 'n_estimators': 50},
  {'max_depth': 3, 'n_estimators': 100},
  {'max_depth': 3, 'n_estimators': 150},
  {'max_depth': 3, 'n_estimators': 200},
  {'max_depth': 3, 'n_estimators': 300},
  {'max_depth': 3, 'n_estimators': 500},
  {'max_depth': 3, 'n_estimators': 1000},
  {'max_depth': 4, 'n_estimators': 10},
  {'max_depth': 4, 'n_estimators': 50},
  {'max_depth': 4, 'n_estimators': 100},
  {'max_depth': 4, 'n_estimators': 150},
  {'max_depth': 4, 'n_estimators': 200},
  {'max_depth': 4, 'n_estimators': 300},
  {'max_depth': 4, 'n_estimators': 500},
  {'max_depth': 4, 'n_estimators': 1000},
  {'max_depth': 5, 'n_estimators': 10},
  {'max_depth': 5, 'n_estimators': 50},
  {'max_depth': 5, 'n_estimators': 100},
  {'max_depth': 5, 'n_estimators': 150},
  {'max_depth': 5, 'n_estimators': 200},
  {'max_depth': 5, 'n_estimators': 300},
  {'max_depth': 5, 'n_estimators': 500},
  {'max_depth': 5, 'n_estimators': 1000},
  {'max_depth': 6, 'n_estimators': 10},
  {'max_depth': 6, 'n_estimators': 50},
  {'max_depth': 6, 'n_estimators': 100},
  {'max_depth': 6, 'n_estimators': 150},
  {'max_depth': 6, 'n_estimators': 200},
  {'max_depth': 6, 'n_estimators': 300},
  {'max_depth': 6, 'n_estimators': 500},
  {'max_depth': 6, 'n_estimators': 1000},
  {'max_depth': 7, 'n_estimators': 10},
  {'max_depth': 7, 'n_estimators': 50},
  {'max_depth': 7, 'n_estimators': 100},
  {'max_depth': 7, 'n_estimators': 150},
  {'max_depth': 7, 'n_estimators': 200},
  {'max_depth': 7, 'n_estimators': 300},
  {'max_depth': 7, 'n_estimators': 500},
  {'max_depth': 7, 'n_estimators': 1000},
  {'max_depth': 8, 'n_estimators': 10},
  {'max_depth': 8, 'n_estimators': 50},
  {'max_depth': 8, 'n_estimators': 100},
  {'max_depth': 8, 'n_estimators': 150},
```

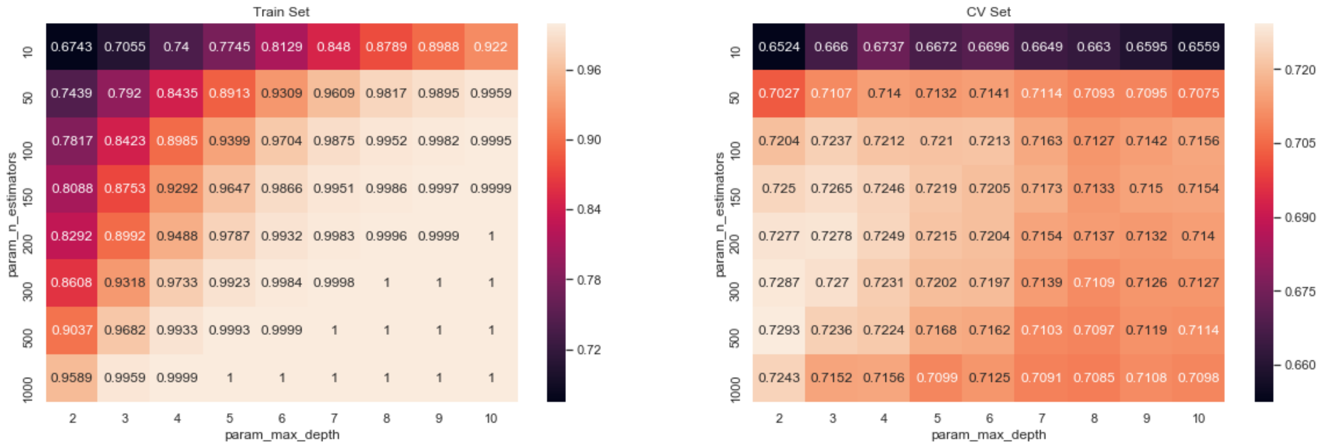
```
 {'max_depth': 8, 'n_estimators': 150},
 {'max_depth': 8, 'n_estimators': 200},
 {'max_depth': 8, 'n_estimators': 300},
 {'max_depth': 8, 'n_estimators': 500},
 {'max_depth': 8, 'n_estimators': 1000},
 {'max_depth': 9, 'n_estimators': 10},
 {'max_depth': 9, 'n_estimators': 50},
 {'max_depth': 9, 'n_estimators': 100},
 {'max_depth': 9, 'n_estimators': 150},
 {'max_depth': 9, 'n_estimators': 200},
 {'max_depth': 9, 'n_estimators': 300},
 {'max_depth': 9, 'n_estimators': 500},
 {'max_depth': 9, 'n_estimators': 1000},
 {'max_depth': 10, 'n_estimators': 10},
 {'max_depth': 10, 'n_estimators': 50},
 {'max_depth': 10, 'n_estimators': 100},
 {'max_depth': 10, 'n_estimators': 150},
 {'max_depth': 10, 'n_estimators': 200},
 {'max_depth': 10, 'n_estimators': 300},
 {'max_depth': 10, 'n_estimators': 500},
 {'max_depth': 10, 'n_estimators': 1000}],
'split0_test_score': array([0.66200037, 0.69609191, 0.71251738, 0.71897634, 0.72169388,
       0.72314989, 0.72398285, 0.72075199, 0.66684431, 0.70478458,
       0.71761482, 0.72367975, 0.72548069, 0.72480544, 0.72077744,
       0.71396805, 0.67152017, 0.70739579, 0.71787999, 0.72109335,
       0.72133648, 0.72072363, 0.72145804, 0.71618085, 0.66770645,
       0.70921326, 0.71573819, 0.71664538, 0.71835474, 0.71697815,
       0.71534254, 0.7089074 , 0.67304686, 0.70855551, 0.71566558,
       0.71674669, 0.71690569, 0.71652269, 0.71421701, 0.70915523,
       0.66601459, 0.71010311, 0.71633807, 0.71671784, 0.71582458,
       0.71493035, 0.71165281, 0.70793035, 0.66600633, 0.70521443,
       0.71291044, 0.71549636, 0.71868896, 0.71711673, 0.71293216,
       0.7094888 , 0.66000137, 0.70318625, 0.71150856, 0.71172721,
       0.7116225 , 0.71070412, 0.71145053, 0.70962415, 0.65178118,
       0.7010216 , 0.71327335, 0.71424311, 0.71538095, 0.71373302,
       0.71300461, 0.71015271]),
'split1_test_score': array([0.66364474, 0.71524464, 0.72955067, 0.73402602, 0.73485461,
       0.73340394, 0.73321835, 0.72720188, 0.67824803, 0.72048455,
       0.73047926, 0.73140153, 0.73181647, 0.72896635, 0.72741567,
       0.71980639, 0.68245708, 0.72286252, 0.72285636, 0.72573354,
       0.72609111, 0.7245735 , 0.72489637, 0.71734073, 0.67147284,
       0.72174234, 0.72982411, 0.73001764, 0.72920688, 0.72588283,
       0.72107536, 0.7125814 , 0.67317555, 0.72396827, 0.72531585,
       0.72466102, 0.72457025, 0.72281049, 0.71814257, 0.71578049,
       0.67081331, 0.71966294, 0.72155886, 0.72160538, 0.71914426,
       0.71977445, 0.71545956, 0.71662253, 0.66518763, 0.71357207,
       0.71188135, 0.71028027, 0.70870042, 0.70575613, 0.70668261,
       0.70778398, 0.66301228, 0.71390532, 0.71360757, 0.71485531,
       0.71243715, 0.71243358, 0.71063248, 0.71124808, 0.66134555,
       0.71420243, 0.71779895, 0.71721577, 0.71335747, 0.71027897,
       0.70848938, 0.70934714]),
'split2_test_score': array([0.63164444, 0.69690777, 0.71927145, 0.72202333, 0.7266297 ,
       0.72941064, 0.73061186, 0.7248239 , 0.6528715 , 0.7066868 ,
       0.72288452, 0.72439476, 0.72619878, 0.7273333 , 0.72251901,
       0.71188202, 0.66724474, 0.71184486, 0.7228704 , 0.72689037,
       0.72721709, 0.72407308, 0.72096721, 0.71342197, 0.66254196,
       0.70850007, 0.7173108 , 0.71894456, 0.71679759, 0.7177276 ,
       0.71411972, 0.7081357 , 0.66243078, 0.70974187, 0.72300738,
       0.72014513, 0.71983935, 0.71980283, 0.71637154, 0.71249359,
       0.657864  , 0.70433403, 0.71097149, 0.71348965, 0.71113948,
       0.70710652, 0.70367799, 0.70269167, 0.65789516, 0.70915092,
       0.71317235, 0.71407736, 0.71374399, 0.70986977, 0.70946092,
       0.708174  , 0.65535801, 0.71151457, 0.71742799, 0.71845197,
       0.71549802, 0.71481147, 0.71373847, 0.71161747, 0.65453091,
       0.70724692, 0.7157517 , 0.71479621, 0.71331729, 0.71411404,
       0.712658  , 0.71000416]),
'mean_test_score': array([0.65243126, 0.7027485 , 0.72044658, 0.72500877, 0.72772614,
       0.72865477, 0.72927093, 0.72425922, 0.66598884, 0.71065225,
       0.72365958, 0.72649216, 0.72783209, 0.72703501, 0.72357078,
       0.71521905, 0.67374111, 0.71403454, 0.72120214, 0.72457226,
       0.7248814 , 0.72312334, 0.72244064, 0.715648  , 0.66724073,
       0.71315221, 0.72095795, 0.72186939, 0.72145339, 0.72019636,
       0.71684606, 0.70987495, 0.66955155, 0.71408885, 0.72132949,
       0.72051764, 0.72043847, 0.719712  , 0.7162437 , 0.71247644,
       0.66489778, 0.71136717, 0.71628984, 0.71727121, 0.71536973,
       0.71393757, 0.7102639 , 0.70908195, 0.66303006, 0.70931249,
       0.71265468, 0.71328461, 0.71371112, 0.71091428, 0.70969191,
       0.70848228, 0.6594575 , 0.70953525, 0.71418115, 0.71501126,
```

```
       0.70848228, 0.6594575 , 0.70955525, 0.71418115, 0.71501126,
       0.71318573, 0.71264958, 0.71194037, 0.71082985, 0.65588597,
       0.70749033, 0.71560799, 0.7154184 , 0.71401862, 0.71270858,
       0.71138391, 0.70983466]),
 'std_test_score': array([0.01471233, 0.00884283, 0.00700351, 0.00649668, 0.00542866,
       0.00422031, 0.00388787, 0.00266334, 0.0103774 , 0.00699608,
       0.00528058, 0.00348388, 0.00283273, 0.00171178, 0.00281031,
       0.00335382, 0.00640581, 0.0065015 , 0.00234924, 0.00250501,
       0.00254856, 0.00170919, 0.00174807, 0.00164357, 0.0036608 ,
       0.00608142, 0.00630243, 0.00583792, 0.00551956, 0.00403277,
       0.00303209, 0.0019396 , 0.0050349 , 0.00700293, 0.00411458,
       0.00324184, 0.00315769, 0.00256787, 0.0016052 , 0.00270487,
       0.00534511, 0.00632141, 0.00432234, 0.0033362 , 0.00328368,
       0.00521901, 0.00490898, 0.00574516, 0.0036459 , 0.00341402,
       0.00055721, 0.00220206, 0.00407801, 0.00469653, 0.00255668,
       0.00072934, 0.00314837, 0.00459451, 0.00245036, 0.00274754,
       0.00166835, 0.00168373, 0.00131445, 0.00086583, 0.00402059,
       0.00538399, 0.00185042, 0.00129089, 0.0009635 , 0.00172511,
       0.00205173, 0.00035004]),
 'rank_test_score': array([72, 63, 21,  7,  4,  2,  1, 10, 67, 53, 11,  6,  3,  5, 12, 33, 64,
       37, 18,  9,  8, 13, 14, 29, 66, 43, 19, 15, 16, 23, 26, 55, 65, 36,
       17, 20, 22, 24, 28, 47, 68, 50, 27, 25, 32, 39, 54, 60, 69, 59, 45,
       41, 40, 51, 57, 61, 70, 58, 35, 34, 42, 46, 48, 52, 71, 62, 30, 31,
       38, 44, 49, 56], dtype=int32),
 'split0_train_score': array([0.6884685 , 0.74865568, 0.78715625, 0.81286726, 0.83233006,
       0.86349132, 0.90457307, 0.95938506, 0.71003144, 0.79576546,
       0.84695923, 0.8788992 , 0.90319789, 0.93339024, 0.96809641,
       0.99626318, 0.74359166, 0.8427795 , 0.89939964, 0.9296087 ,
       0.94823935, 0.97245744, 0.99335623, 0.99985532, 0.78067699,
       0.88883415, 0.94063717, 0.96559106, 0.97796989, 0.99205567,
       0.99929194, 0.99999992, 0.81750135, 0.92796732, 0.96938035,
       0.98599572, 0.99267396, 0.99806712, 0.99994047, 1.        ,
       0.85488734, 0.95710998, 0.98590563, 0.99440118, 0.99819736,
       0.99983083, 0.99999959, 1.        , 0.88450998, 0.97992686,
       0.99486254, 0.99854592, 0.99961227, 0.99998046, 1.        ,
       1.        , 0.90090769, 0.98722184, 0.99790014, 0.99962322,
       0.99993188, 0.99999976, 1.        , 1.        , 0.91906892,
       0.99527346, 0.99945778, 0.99992896, 0.99999424, 1.        ,
       1.        , 1.        ]),
 'split1_train_score': array([0.67295873, 0.74094114, 0.7784657 , 0.80698452, 0.82665205,
       0.85848212, 0.90177633, 0.95814994, 0.70370635, 0.79062837,
       0.84250948, 0.87306491, 0.89695098, 0.92949877, 0.96761895,
       0.99625325, 0.73633731, 0.84484786, 0.89515819, 0.92761333,
       0.94734493, 0.97158612, 0.99275133, 0.99989271, 0.76694918,
       0.89447904, 0.94281031, 0.96522381, 0.97977537, 0.99309988,
       0.99939744, 0.99999984, 0.81141573, 0.93632525, 0.97241178,
       0.98776276, 0.99406924, 0.9987837 , 0.99996245, 1.        ,
       0.84852061, 0.96523094, 0.98875556, 0.99594609, 0.99866546,
       0.99982402, 0.99999919, 1.        , 0.88165344, 0.98309806,
       0.99560426, 0.99871472, 0.99960522, 0.99998078, 1.        ,
       1.        , 0.89968632, 0.99123212, 0.99867138, 0.99976409,
       0.99996083, 0.99999992, 1.        , 1.        , 0.9210951 ,
       0.99617097, 0.99960449, 0.99994258, 0.99999416, 1.        ,
       1.        , 1.        ]),
 'split2_train_score': array([0.66157028, 0.74224755, 0.7793643 , 0.80661869, 0.82847059,
       0.86033223, 0.90467526, 0.95905551, 0.70277589, 0.7897066 ,
       0.83733037, 0.8739355 , 0.89757416, 0.93236477, 0.96893558,
       0.99504284, 0.7400382 , 0.84284207, 0.90101516, 0.9304214 ,
       0.95096487, 0.97594628, 0.99373473, 0.9999235 , 0.77579071,
       0.89072173, 0.93626651, 0.96323818, 0.97828729, 0.99173978,
       0.99919022, 1.        , 0.80976624, 0.92830877, 0.96945596,
       0.98616864, 0.99299075, 0.99826248, 0.99993914, 1.        ,
       0.84058413, 0.96021588, 0.98779047, 0.99505743, 0.99804443,
       0.99976749, 0.99999838, 1.        , 0.87055949, 0.98218003,
       0.99509106, 0.99857834, 0.99954721, 0.99996912, 1.        ,
       1.        , 0.89587112, 0.99004346, 0.99815538, 0.99967137,
       0.99993038, 0.99999968, 1.        , 1.        , 0.92576463,
       0.99633329, 0.99953522, 0.99991798, 0.99999003, 1.        ,
       1.        , 1.        ]),
 'mean_train_score': array([0.6743325 , 0.74394813, 0.78166208, 0.80882349, 0.8291509 ,
       0.86076856, 0.90367489, 0.9588635 , 0.70550456, 0.79203348,
       0.84226636, 0.87529987, 0.89924101, 0.93175126, 0.96821698,
       0.99585309, 0.73998906, 0.84348981, 0.89852433, 0.92921448,
       0.94884971, 0.97332995, 0.99328076, 0.99989051, 0.77447229,
       0.89134497, 0.93990466, 0.96468435, 0.97867752, 0.99229844,
       0.9992932 , 0.99999992, 0.81289444, 0.93086711, 0.97041603,
       0.98664238, 0.99324465, 0.9983711 , 0.99994735, 1.        ,
       0.84799736, 0.96085227, 0.98748389, 0.9951349 , 0.99830242,
```

```
        0.84799736, 0.96085227, 0.98748389, 0.9951349 , 0.99830242,
        0.99980745, 0.99999905, 1.        , 0.87890764, 0.98173498,
        0.99518596, 0.998613  , 0.99958823, 0.99997679, 1.        ,
        1.        , 0.89882171, 0.98949914, 0.9982423 , 0.99968623,
        0.99994103, 0.99999978, 1.        , 1.        , 0.92197622,
        0.99592591, 0.9995325 , 0.99992984, 0.99999281, 1.        ,
        1.        , 1.        ]),
 'std_train_score': array([1.10240345e-02, 3.37120169e-03, 3.90224446e-03, 2.86327625e-03,
        2.36742845e-03, 2.06814117e-03, 1.34312992e-03, 5.22193545e-04,
        3.22344890e-03, 2.66560669e-03, 3.93472320e-03, 2.56980681e-03,
        2.80948222e-03, 1.64685310e-03, 5.44232239e-04, 5.72944695e-04,
        2.96178041e-03, 9.60625051e-04, 2.46990522e-03, 1.17979576e-03,
        1.53956619e-03, 1.88391470e-03, 4.05004304e-04, 2.78752266e-05,
        5.68136638e-03, 2.34627514e-03, 2.72124555e-03, 1.03352957e-03,
        7.87039011e-04, 5.81192735e-04, 8.46041847e-05, 6.62159944e-08,
        3.32644959e-03, 3.86200062e-03, 1.41154727e-03, 7.95372341e-04,
        5.97245707e-04, 3.02456579e-04, 1.06895053e-05, 0.00000000e+00,
        5.85097433e-03, 3.34576802e-03, 1.18350569e-03, 6.33081618e-04,
        2.64190699e-04, 2.83908299e-05, 5.05245539e-07, 0.00000000e+00,
        6.01712010e-03, 1.33233664e-03, 3.10150756e-04, 7.31404695e-05,
        2.91487818e-05, 5.42042447e-06, 0.00000000e+00, 0.00000000e+00,
        2.14513928e-03, 1.68182519e-03, 3.20799436e-04, 5.84600350e-05,
        1.40134272e-05, 1.01068625e-07, 0.00000000e+00, 0.00000000e+00,
        2.80362011e-03, 4.66083356e-04, 5.99233148e-05, 1.00616366e-05,
        1.96594172e-06, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00])}
```

In [111]:

```python
# Find best hyper parameter max_depth and min_samples_split
import seaborn as sns; sns.set()
max_scores = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max(
).unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [112]:

```python
# Print params
print(clf.best_estimator_)
print(clf.score(X_train_bow, y_train))
print(clf.score(X_test_bow, y_test))
```

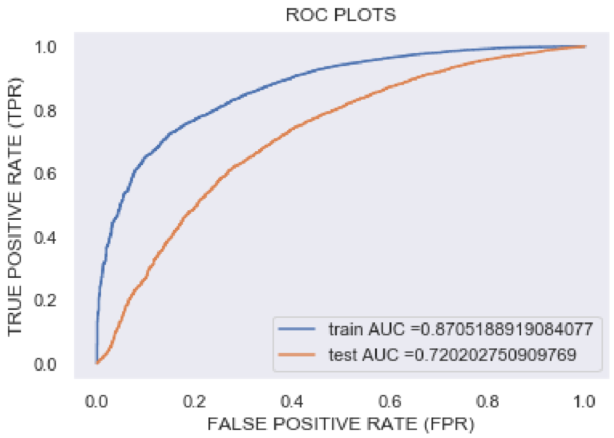
```
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=2,
              min_child_weight=1, missing=None, n_estimators=500, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
0.8705188919084077
0.720202750909769
```

In [334]:

```
n_estimators = 500
max_depth = 2
```

In [114]:

```
%%time

# Create ROC Plot for Test Set
parameters = {'max_depth':[max_depth], 'n_estimators':[n_estimators]}
xgb=GridSearchCV(XGBClassifier(class_weight='balanced', n_estimators=n_estimators,
max_depth=max_depth, n_jobs=-1), parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score
=True)
xgb.fit(X_train_bow, y_train);

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

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```




```
CPU times: user 51.3 s, sys: 394 ms, total: 51.7 s
Wall time: 1min 55s
```

In [115]:

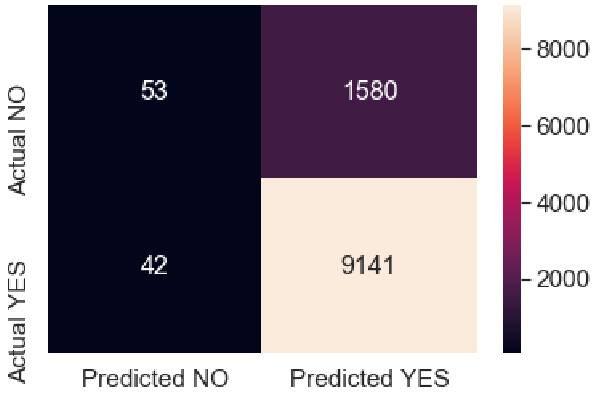
```
%%time
get_confusion_matrix(xgb,X_train_bow,y_train)
```

```
CPU times: user 634 ms, sys: 30.5 ms, total: 665 ms
Wall time: 672 ms
```

In [116]:

```
%%time
get_confusion_matrix(xgb,X_test_bow,y_test)
```

```
CPU times: user 484 ms, sys: 22 ms, total: 506 ms
Wall time: 509 ms
```



In [336]:

```
%%time
from xgboost import XGBClassifier
# Train a model on above hyperparameter to select best features
xgb_feature_selection = XGBClassifier(class_weight='balanced', n_estimators=n_estimators,
max_depth=max_depth, n_jobs=-1)
xgb_feature_selection.fit(X_train_bow, y_train)
xgb_feature_selection.feature_importances_
```

```
CPU times: user 55.7 s, sys: 442 ms, total: 56.1 s
Wall time: 59.7 s
```

Out[336]:

```
array([0.00349855, 0.        , 0.00133162, ..., 0.00711722, 0.00464577,
       0.00693135], dtype=float32)
```

In [337]:

```
# https://stackoverflow.com/questions/44101458/random-forest-feature-importance-chart-using-python

# Calculate feature importances
importances = xgb_feature_selection.feature_importances_
# Sort feature importances in descending order
indices_pos = np.argsort(importances)[::-1]

# Sort feature importances in asscending order
indices_neg = np.argsort(importances)[::1]

# Rearrange feature names so they match the sorted feature importances for positive class
names_pos = [feature_names_bow[i] for i in indices_pos]

# Rearrange feature names so they match the sorted feature importances for negative class
names_neg = [feature_names_bow[i] for i in indices_neg]
```

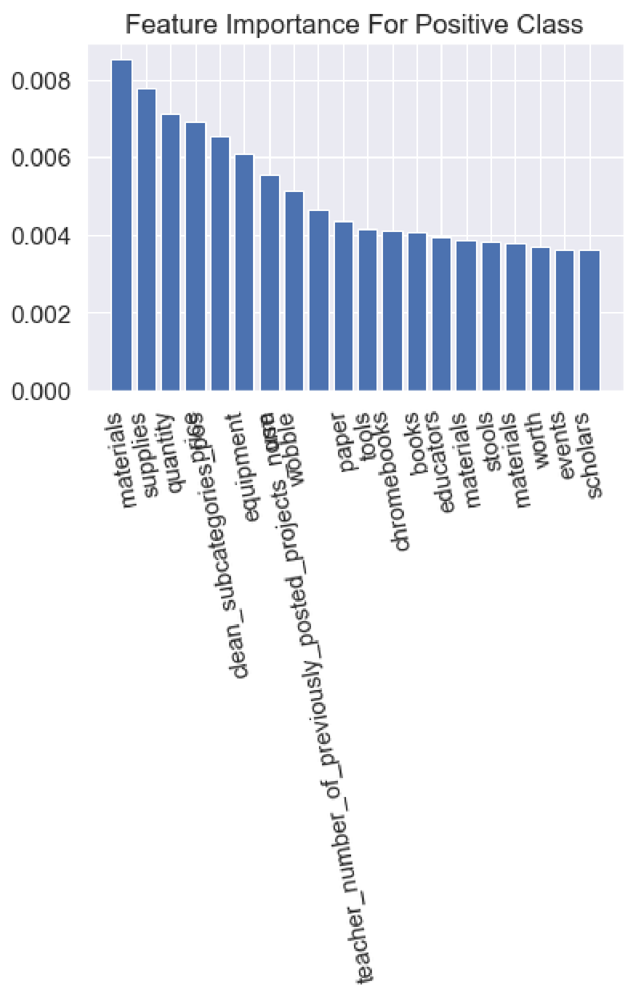
In [338]:

```
# process for top n features
n = 20
```

In [339]:

```
# collect n positive feature

# collect top n importance data
top_20_positive_data = []
for i in indices_pos:
    top_20_positive_data.append(importances[i])
    if(len(top_20_positive_data) >= n):
        break;

print(top_20_positive_data)
print("*"*100)

# collect top n importance label
top_20_positive_label = names_pos[0:n]

print(top_20_positive_label)
print("*"*100)

# Barplot: Add bars
plt.bar(range(n), top_20_positive_data)
# Add feature names as x-axis labels
plt.xticks(range(n), top_20_positive_label , rotation=100, fontsize = 15)
# Create plot title
plt.title("Feature Importance For Positive Class")
# Show plot
plt.show()
```

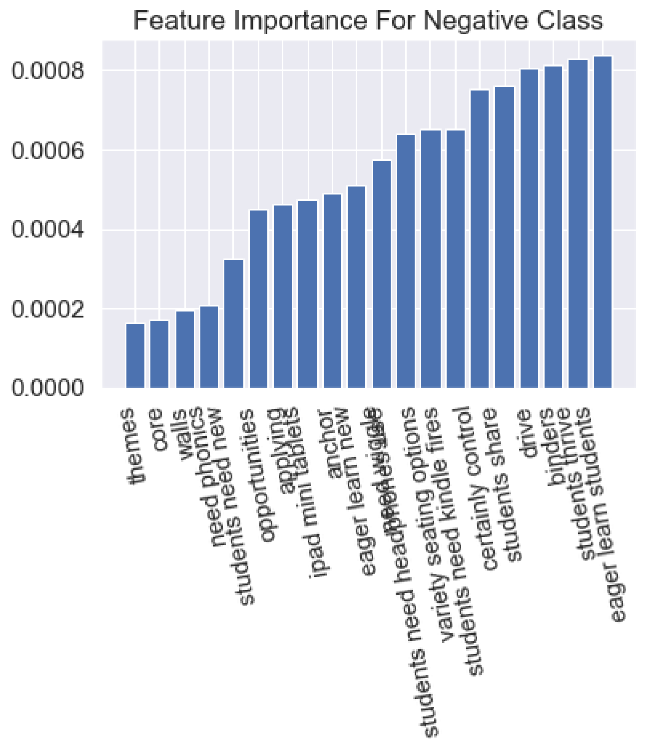
```
[0.0085176835, 0.007785931, 0.0071172155, 0.0069313548, 0.0065340674, 0.006097838, 0.00557561,
0.0051340763, 0.004645771, 0.0043550604, 0.0041668196, 0.004128658, 0.0040737395, 0.0039688707,
0.0038815043, 0.0038250992, 0.0037717978, 0.0037215052, 0.0036201498, 0.0036098238]
****************************************************************************************************

['materials', 'supplies', 'quantity', 'price', 'clean_subcategories_pos', 'equipment', 'use', 'wob
ble', 'teacher_number_of_previously_posted_projects_norm', 'paper', 'tools', 'chromebooks', 'books
', 'educators', 'materials', 'stools', 'materials', 'worth', 'events', 'scholars']
****************************************************************************************************
```



In [340]:

```
# collect n negative feature

# collect top n importance data
top_20_negative_data = []
collect_neg_indices = []
for i in indices_neg:
    if(importances[i] > 0):
        top_20_negative_data.append(importances[i])
        collect_neg_indices.append(i)
    if(len(top_20_negative_data) >= n):
        break;

print(top_20_negative_data)
print("*"*100)

# collect top n importance data
top_20_negative_label = []
for i in collect_neg_indices:
    top_20_negative_label.append(names_neg[i])

print(top_20_negative_label)
print("*"*100)

# Barplot: Add bars
plt.bar(range(n), top_20_negative_data)
# Add feature names as x-axis labels
plt.xticks(range(n), top_20_negative_label , rotation=100, fontsize = 15)
# Create plot title
plt.title("Feature Importance For Negative Class")
# Show plot
plt.show()
```

```
[0.00016181421, 0.0001717893, 0.00019614046, 0.00020639696, 0.00032587635, 0.00044913276,
0.00046302844, 0.0004734551, 0.0004897616, 0.00051184365, 0.0005776677, 0.0006421192, 0.0006526387,
0.0006545005, 0.0007516924, 0.0007634862, 0.0008068299, 0.00081551, 0.0008287739, 0.00083640416]
****************************************************************************************************

['themes', 'core', 'walls', 'need phonics', 'students need new', 'opportunities', 'applying', 'ipa
d mini tablets', 'anchor', 'eager learn new', 'need wiggle', 'students need headphones use',
'variety seating options', 'students need kindle fires', 'certainly control', 'students share', 'd
rive', 'binders', 'students thrive', 'eager learn students']
****************************************************************************************************
```



2.5.2 Applying XGBOOST on TFIDF, SET 2

In [117]:

```
%%time
# Please write all the code with proper documentation
# Prepare data for TFIDF
X_train_tfidf = hstack((X_train_school_state_pos, X_train_school_state_neg,
X_train_clean_categories_pos, X_train_clean_categories_neg, X_train_clean_subcategories_pos,
X_train_clean_subcategories_neg, X_train_project_grade_category_pos,
X_train_project_grade_category_neg, X_train_teacher_prefix_pos, X_train_teacher_prefix_neg,
X_train_title_tfidf, X_train_essay_tfidf, X_train_project_resource_summary_tfidf,
X_train_quantity_norm, X_train_teacher_number_of_previously_posted_projects_norm,
X_train_price_norm)).tocsr()
X_cv_tfidf = hstack((X_cv_school_state_pos, X_cv_school_state_neg, X_cv_clean_categories_pos,
X_cv_clean_categories_neg, X_cv_clean_subcategories_pos, X_cv_clean_subcategories_neg,
X_cv_project_grade_category_pos, X_cv_project_grade_category_neg, X_cv_teacher_prefix_pos,
X_cv_teacher_prefix_neg, X_cv_title_tfidf, X_cv_essay_tfidf, X_cv_project_resource_summary_tfidf,
X_cv_quantity_norm, X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocs
r()
X_test_tfidf = hstack((X_test_school_state_pos, X_test_school_state_neg,
X_test_clean_categories_pos, X_test_clean_categories_neg, X_test_clean_subcategories_pos,
X_test_clean_subcategories_neg, X_test_project_grade_category_pos,
X_test_project_grade_category_neg, X_test_teacher_prefix_pos, X_test_teacher_prefix_neg,
X_test_title_tfidf, X_test_essay_tfidf, X_test_project_resource_summary_tfidf,
X_test_quantity_norm, X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm)
).tocsr()

print(X_train_tfidf.shape)
print(X_cv_tfidf.shape)
print(X_test_tfidf.shape)


import matplotlib.pyplot as plt
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV

xgb = XGBClassifier(class_weight='balanced', n_jobs=-1)
parameters = {'n_estimators':[10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6, 7
, 8, 9, 10]}
clf=GridSearchCV(xgb, parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score=True)
clf.fit(X_train_tfidf, y_train)
```

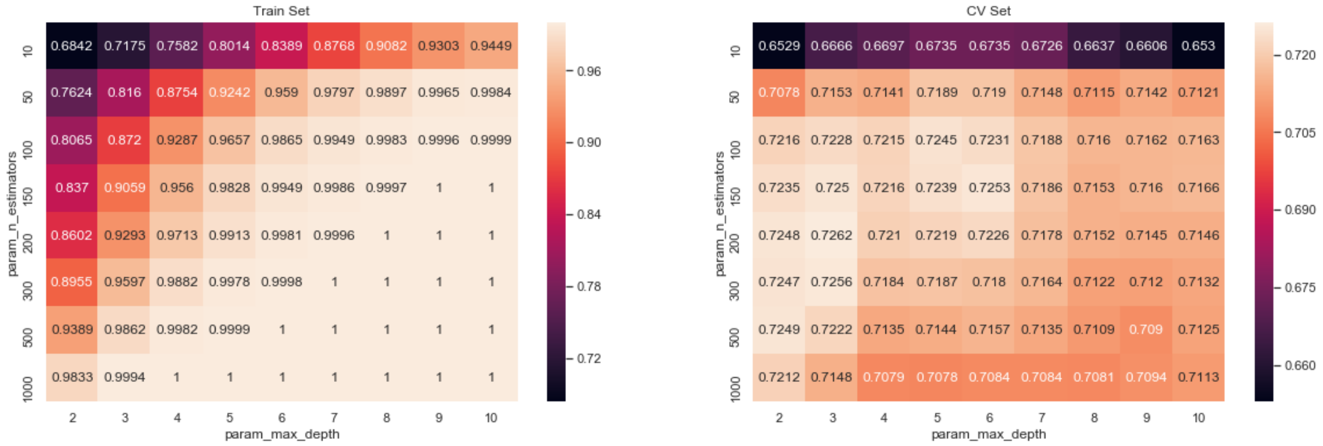
```
(14711, 10072)
(7247, 10072)
(10816, 10072)
CPU times: user 37.8 s, sys: 410 ms, total: 38.2 s
Wall time: 22h 21min 12s
```

Out[117]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     class_weight='balanced',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=-1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [10, 50, 100, 150, 200, 300, 500,
                                          1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [118]:

```
# Find best hyper parameter max_depth and min_samples_split
import seaborn as sns; sns.set()
max_scores = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max(
).unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [119]:

```
# Print params
print(clf.best_estimator_)
print(clf.score(X_train_tfidf, y_train))
print(clf.score(X_test_tfidf, y_test))
```

```
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=200, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
0.8968451733475054
0.7184091200232278
```

In [341]:

```
n_estimators = 200
max_depth = 3
```

In [ ]:

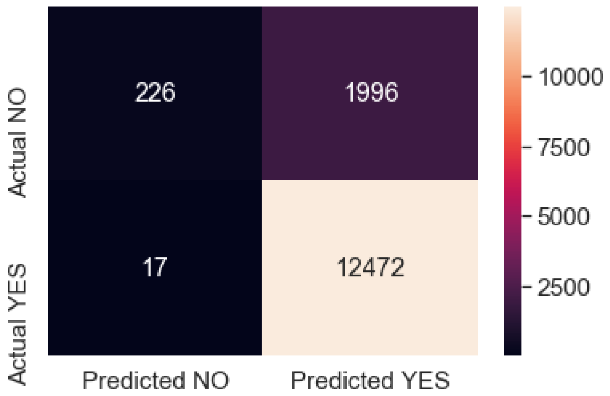
```
%%time

# Create ROC Plot for Test Set
parameters = {'max_depth':[max_depth], 'n_estimators':[n_estimators]}
xgb=GridSearchCV(XGBClassifier(class_weight='balanced', n_estimators=n_estimators,
max_depth=max_depth, n_jobs=-1), parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score
=True)
xgb.fit(X_train_tfidf, y_train);

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

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
```

```
plt.title('ROC PLOTS')
plt.grid()
plt.show()
```

In [125]:

```
%%time
get_confusion_matrix(xgb,X_train_tfidf,y_train)
```

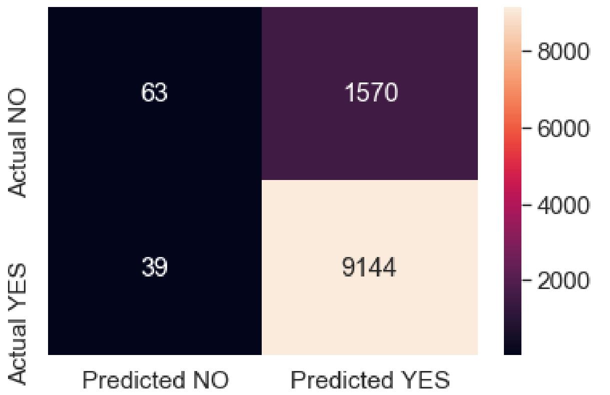
```
CPU times: user 513 ms, sys: 24.2 ms, total: 538 ms
Wall time: 541 ms
```



In [126]:

```
%%time
get_confusion_matrix(xgb,X_test_tfidf,y_test)
```

```
CPU times: user 431 ms, sys: 39.1 ms, total: 470 ms
Wall time: 494 ms
```



In [342]:

```
%%time
from xgboost import XGBClassifier
# Train a model on above hyperparameter to select best features
xgb_feature_selection = XGBClassifier(class_weight='balanced', n_estimators=n_estimators,
max_depth=max_depth, n_jobs=-1)
xgb_feature_selection.fit(X_train_tfidf, y_train)
xgb_feature_selection.feature_importances_
```

```
CPU times: user 47.8 s, sys: 425 ms, total: 48.2 s
Wall time: 51.3 s
```

Out[342]:

```
array([0.00331598, 0.        , 0.00140907, ..., 0.00608898, 0.00459591,
       0.00659823], dtype=float32)
```

In [343]:

```
# https://stackoverflow.com/questions/44101458/random-forest-feature-importance-chart-using-python

# Calculate feature importances
importances = xgb_feature_selection.feature_importances_
# Sort feature importances in descending order
indices_pos = np.argsort(importances)[::-1]

# Sort feature importances in asccending order
indices_neg = np.argsort(importances)[::1]

# Rearrange feature names so they match the sorted feature importances for positive class
names_pos = [feature_names_tfidf[i] for i in indices_pos]

# Rearrange feature names so they match the sorted feature importances for negative class
names_neg = [feature_names_tfidf[i] for i in indices_neg]
```

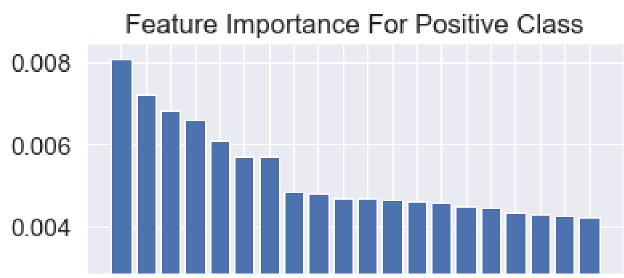
In [344]:

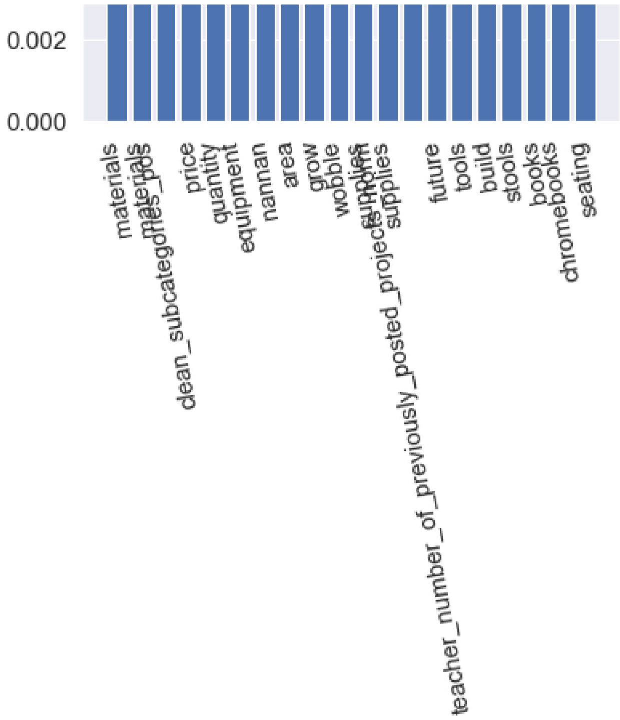
```
# process for top n features
n = 20
```

In [345]:

```
# collect n positive feature

# collect top n importance data
top_20_positive_data = []
for i in indices_pos:
    top_20_positive_data.append(importances[i])
    if(len(top_20_positive_data) >= n):
        break;

print(top_20_positive_data)
print("*"*100)

# collect top n importance label
top_20_positive_label = names_pos[0:n]

print(top_20_positive_label)
print("*"*100)

# Barplot: Add bars
plt.bar(range(n), top_20_positive_data)
# Add feature names as x-axis labels
plt.xticks(range(n), top_20_positive_label , rotation=100, fontsize = 15)
# Create plot title
plt.title("Feature Importance For Positive Class")
# Show plot
plt.show()
```

```
[0.008084392, 0.007212308, 0.0068307333, 0.0065982286, 0.006088976, 0.0057069752, 0.0056994497,
0.0048559345, 0.004795042, 0.004702217, 0.00468814, 0.0046605607, 0.004595907, 0.0045837476,
0.004491148, 0.0044378703, 0.004341614, 0.004304234, 0.004251539, 0.00422752]
****************************************************************************************************

['materials', 'materials', 'clean_subcategories_pos', 'price', 'quantity', 'equipment', 'nannan',
'area', 'grow', 'wobble', 'supplies', 'supplies',
'teacher_number_of_previously_posted_projects_norm', 'future', 'tools', 'build', 'stools',
'books', 'chromebooks', 'seating']
****************************************************************************************************
```



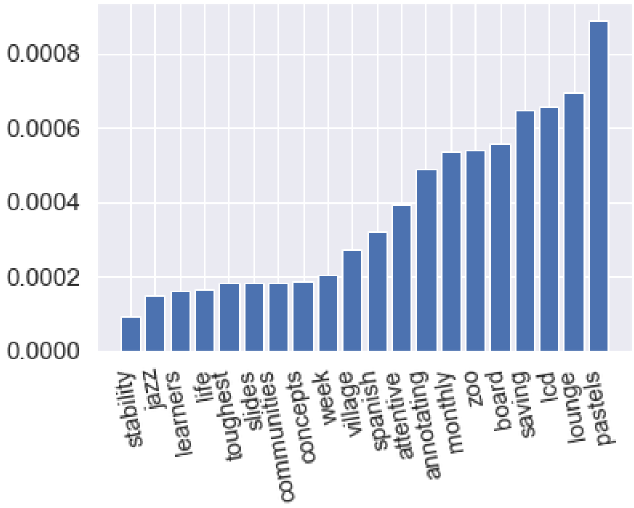


In [346]:

```
# collect n negative feature

# collect top n importance data
top_20_negative_data = []
collect_neg_indices = []
for i in indices_neg:
    if(importances[i] > 0):
        top_20_negative_data.append(importances[i])
        collect_neg_indices.append(i)
    if(len(top_20_negative_data) >= n):
        break;

print(top_20_negative_data)
print("*"*100)

# collect top n importance data
top_20_negative_label = []
for i in collect_neg_indices:
    top_20_negative_label.append(names_neg[i])

print(top_20_negative_label)
print("*"*100)

# Barplot: Add bars
plt.bar(range(n), top_20_negative_data)
# Add feature names as x-axis labels
plt.xticks(range(n), top_20_negative_label , rotation=100, fontsize = 15)
# Create plot title
plt.title("Feature Importance For Negative Class")
# Show plot
plt.show()
```

```
[9.292743e-05, 0.00015012757, 0.00016106894, 0.00016453672, 0.00018180761, 0.00018269985,
0.00018389258, 0.00018649247, 0.00020408655, 0.00027387298, 0.0003229957, 0.00039613864,
0.0004894801, 0.00053861097, 0.0005395893, 0.00055635575, 0.0006508971, 0.000658669, 0.0006948462,
0.0008917722]
****************************************************************************************************

['stability', 'jazz', 'learners', 'life', 'toughest', 'slides', 'communities', 'concepts', 'week',
'village', 'spanish', 'attentive', 'annotating', 'monthly', 'zoo', 'board', 'saving', 'lcd',
'lounge', 'pastels']
****************************************************************************************************
```

Feature Importance For Negative Class



2.5.3 Applying XGBOOST on AVG W2V, SET 3

In [120]:

```
%%time
# Please write all the code with proper documentation
# Prepare data for AVGW2V
X_train_avgw2v = hstack((X_train_school_state_pos, X_train_school_state_neg,
X_train_clean_categories_pos, X_train_clean_categories_neg, X_train_clean_subcategories_pos,
X_train_clean_subcategories_neg, X_train_project_grade_category_pos,
X_train_project_grade_category_neg, X_train_teacher_prefix_pos, X_train_teacher_prefix_neg,
avg_w2v_vectors_text_train, avg_w2v_vectors_title_train,
avg_w2v_vectors_project_resource_summary_train, X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm)).tocsr()
X_cv_avgw2v = hstack((X_cv_school_state_pos, X_cv_school_state_neg, X_cv_clean_categories_pos,
X_cv_clean_categories_neg, X_cv_clean_subcategories_pos, X_cv_clean_subcategories_neg,
X_cv_project_grade_category_pos, X_cv_project_grade_category_neg, X_cv_teacher_prefix_pos,
X_cv_teacher_prefix_neg, avg_w2v_vectors_text_cv, avg_w2v_vectors_title_cv,
avg_w2v_vectors_project_resource_summary_cv, X_cv_quantity_norm,
X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocsr()
X_test_avgw2v = hstack((X_test_school_state_pos, X_test_school_state_neg,
X_test_clean_categories_pos, X_test_clean_categories_neg, X_test_clean_subcategories_pos,
X_test_clean_subcategories_neg, X_test_project_grade_category_pos,
X_test_project_grade_category_neg, X_test_teacher_prefix_pos, X_test_teacher_prefix_neg,
avg_w2v_vectors_text_test, avg_w2v_vectors_title_test,
avg_w2v_vectors_project_resource_summary_test, X_test_quantity_norm,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm)).tocsr()


print(X_train_avgw2v.shape)
print(X_cv_avgw2v.shape)
print(X_test_avgw2v.shape)


import matplotlib.pyplot as plt
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV

xgb = XGBClassifier(class_weight='balanced', n_jobs=-1)
parameters = {'n_estimators':[10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6, 7
, 8, 9, 10]}
clf=GridSearchCV(xgb, parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score=True)
clf.fit(X_train_avgw2v, y_train)
```

```
(14711, 913)
(7247, 913)
(10816, 913)
CPU times: user 3min 43s, sys: 2.89 s, total: 3min 46s
Wall time: 1d 5h 58min 30s
```

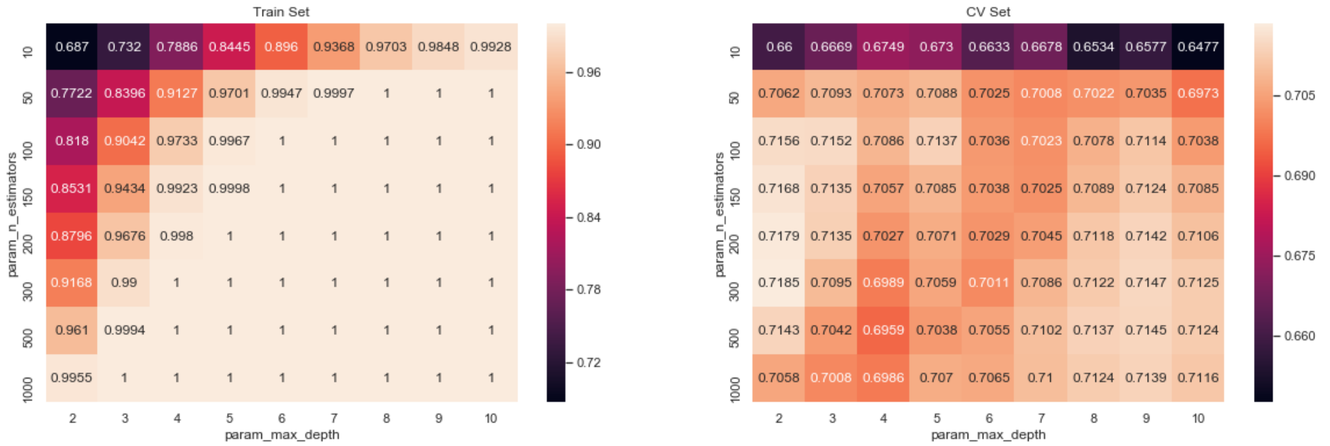
Out[120]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
```

```
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     class_weight='balanced',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=-1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [10, 50, 100, 150, 200, 300, 500,
                                          1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [121]:

```python
# Find best hyper parameter max_depth and min_samples_split
import seaborn as sns; sns.set()
max_scores = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max(
).unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [122]:

```python
# Print params
print(clf.best_estimator_)
print(clf.score(X_train_avgw2v, y_train))
print(clf.score(X_test_avgw2v, y_test))
```

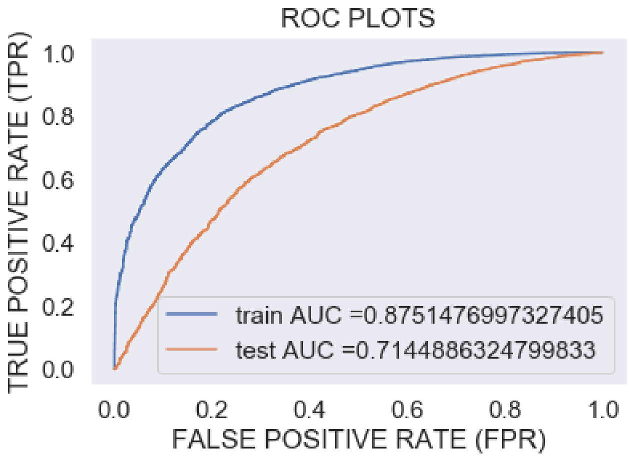
```
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=2,
              min_child_weight=1, missing=None, n_estimators=300, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
0.8751476997327405
0.7144886324799833
```

In [127]:

```python
n_estimators = 300
max_depth = 2
```

In [128]:

```
%%time

# Create ROC Plot for Test Set
parameters = {'max_depth':[max_depth], 'n_estimators':[n_estimators]}
xgb=GridSearchCV(XGBClassifier(class_weight='balanced', n_estimators=n_estimators,
max_depth=max_depth, n_jobs=-1), parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score
=True)
xgb.fit(X_train_avgw2v, y_train);

y_train_pred = clf.predict_proba(X_train_avgw2v)[:,1]
y_test_pred = clf.predict_proba(X_test_avgw2v)[:,1]

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```

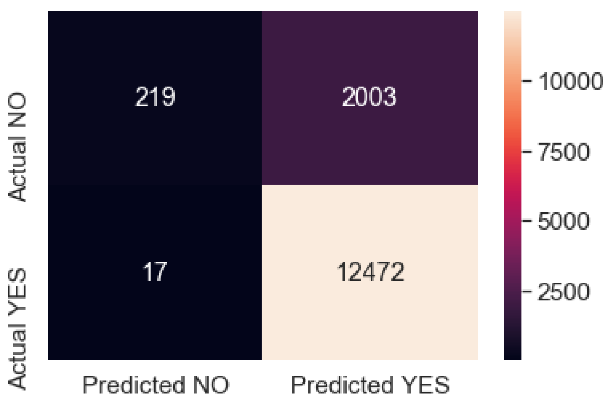



```
CPU times: user 3min 29s, sys: 1.19 s, total: 3min 30s
Wall time: 7min 30s
```

In [129]:

```
%%time
get_confusion_matrix(xgb,X_train_avgw2v,y_train)
```

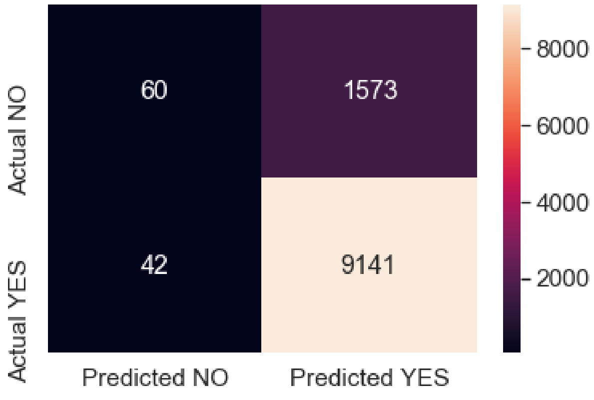
```
CPU times: user 3.23 s, sys: 210 ms, total: 3.44 s
Wall time: 3.49 s
```



In [130]:

```
%%time
get_confusion_matrix(xgb,X_test_avgw2v,y_test)
```

```
CPU times: user 2.53 s, sys: 135 ms, total: 2.66 s
Wall time: 3.08 s
```



2.5.4 Applying XGBOOST on TFIDF W2V, SET 4

In [139]:

```
%%time
# Please write all the code with proper documentation
# Prepare data for TFIDFW2V
X_train_tfidfw2v = hstack((X_train_school_state_pos, X_train_school_state_neg,  X_train_clean_categ
ories_pos, X_train_clean_categories_neg, X_train_clean_subcategories_pos,
X_train_clean_subcategories_neg, X_train_project_grade_category_pos,
X_train_project_grade_category_neg, X_train_teacher_prefix_pos, X_train_teacher_prefix_neg,
tfidf_w2v_vectors_text_train, tfidf_w2v_vectors_title_train,
tfidf_w2v_vectors_project_resource_summary_train, X_train_quantity_norm,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm)).tocsr()
X_cv_tfidfw2v = hstack((X_cv_school_state_pos, X_cv_school_state_neg, X_cv_clean_categories_pos,
X_cv_clean_categories_neg, X_cv_clean_subcategories_pos, X_cv_clean_subcategories_neg,
X_cv_project_grade_category_pos, X_cv_project_grade_category_neg, X_cv_teacher_prefix_pos,
X_cv_teacher_prefix_neg, tfidf_w2v_vectors_text_cv, tfidf_w2v_vectors_title_cv,
tfidf_w2v_vectors_project_resource_summary_cv, X_cv_quantity_norm,
X_cv_teacher_number_of_previously_posted_projects_norm, X_cv_price_norm)).tocsr()
X_test_tfidfw2v = hstack((X_test_school_state_pos, X_test_school_state_neg,
X_test_clean_categories_pos, X_test_clean_categories_neg, X_test_clean_subcategories_pos,
X_test_clean_subcategories_neg, X_test_project_grade_category_pos,
X_test_project_grade_category_neg, X_test_teacher_prefix_pos, X_test_teacher_prefix_neg,
tfidf_w2v_vectors_text_test, tfidf_w2v_vectors_title_test,
tfidf_w2v_vectors_project_resource_summary_test, X_test_quantity_norm,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm)).tocsr()


print(X_train_tfidfw2v.shape)
print(X_cv_tfidfw2v.shape)
print(X_test_tfidfw2v.shape)


import matplotlib.pyplot as plt
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV

xgb = XGBClassifier(class_weight='balanced', n_jobs=-1)
parameters = {'n_estimators':[10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth': [2, 3, 4, 5, 6, 7
, 8, 9, 10]}
clf=GridSearchCV(xgb, parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score=True)
clf.fit(X_train_tfidfw2v, y_train)
```

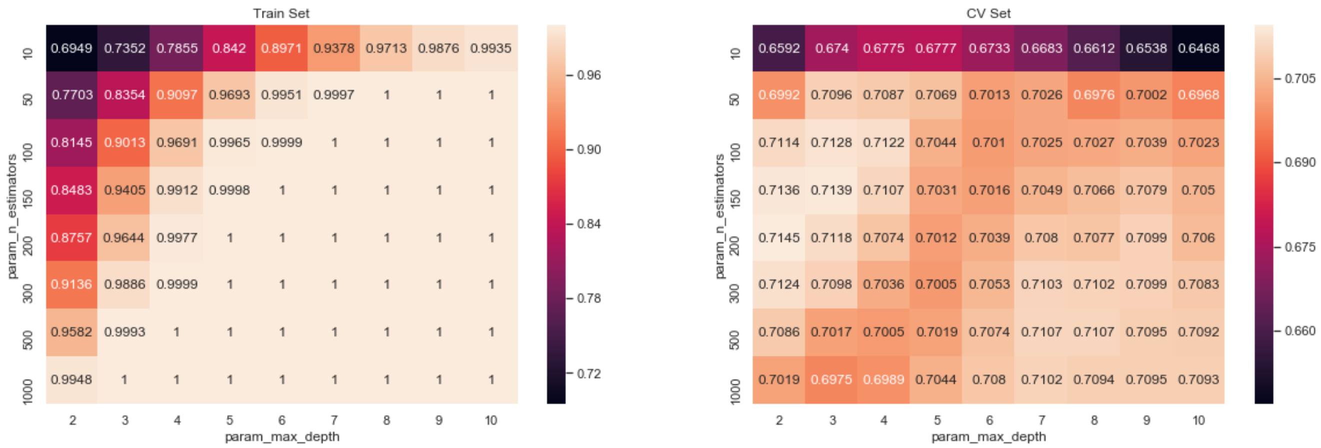
```
(14711, 913)
(7247, 913)
(10816, 913)
CPU times: user 2min 27s, sys: 1.97 s, total: 2min 29s
Wall time: 13h 31min 26s
```

Out[139]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     class_weight='balanced',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=-1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [10, 50, 100, 150, 200, 300, 500,
                                          1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [140]:

```python
# Find best hyper parameter max_depth and min_samples_split
import seaborn as sns; sns.set()
max_scores = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max(
).unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [141]:

```python
# Print params
print(clf.best_estimator_)
print(clf.score(X_train_tfidfw2v, y_train))
print(clf.score(X_test_tfidfw2v, y_test))
```

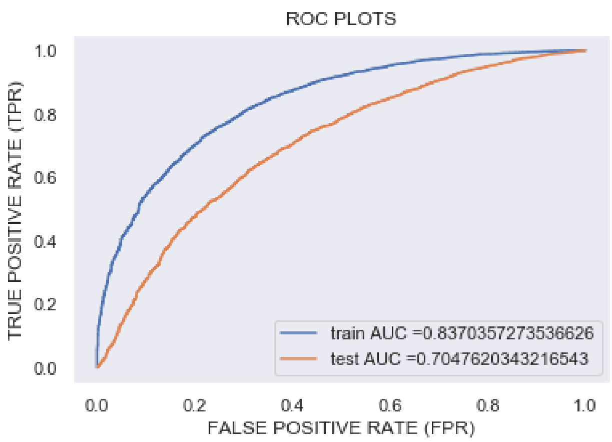
```
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=2,
              min_child_weight=1, missing=None, n_estimators=200, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
0.8370357273536626
0.7047620343216543
```

In [142]:

```python
n_estimators = 200
max_depth = 2
```

In [143]:

```
%%time

# Create ROC Plot for Test Set
parameters = {'max_depth':[max_depth], 'n_estimators':[n_estimators]}
xgb=GridSearchCV(XGBClassifier(class_weight='balanced', n_estimators=n_estimators,
max_depth=max_depth, n_jobs=-1), parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score
=True)
xgb.fit(X_train_tfidfw2v, y_train);

y_train_pred = clf.predict_proba(X_train_tfidfw2v)[:,1]
y_test_pred = clf.predict_proba(X_test_tfidfw2v)[:,1]

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```

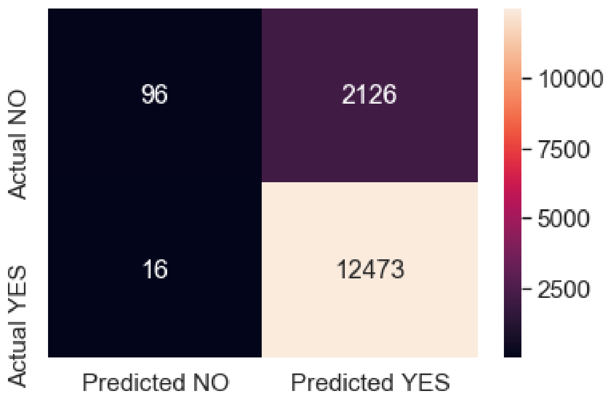



```
CPU times: user 2min 32s, sys: 1.55 s, total: 2min 34s
Wall time: 5min 49s
```

In [144]:

```
%%time
get_confusion_matrix(xgb,X_train_tfidfw2v,y_train)
```

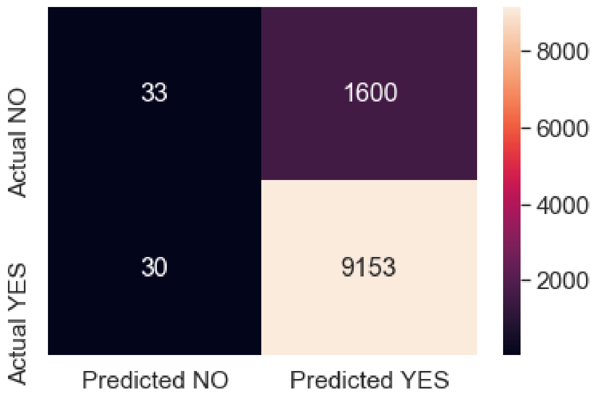
```
CPU times: user 3.37 s, sys: 226 ms, total: 3.6 s
Wall time: 3.89 s
```



In [145]:

```
%%time
get_confusion_matrix(xgb,X_test_tfidfw2v,y_test)
```

```
CPU times: user 2.49 s, sys: 129 ms, total: 2.62 s
Wall time: 2.69 s
```



3. Conclusion

In [348]:

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

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "N Estimators", "Max Depth", "AUC"]
x.add_row(["Bag of Words", "Random Forest", 500, 10, 0.70])
x.add_row(["TFIDF", "Random Forest", 1000, 10, .71])
x.add_row(["AVG W2V", "Random Forest", 1000, 7, .70])
x.add_row(["TFIDF W2V", "Random Forest", 1000, 6, .69])
x.add_row(["Bag of Words", "XGBOOST", 500, 2, .72])
x.add_row(["TFIDF", "XGBOOST", 200, 3, .72])
x.add_row(["AVG W2V", "XGBOOST", 300, 2, .71])
x.add_row(["TFIDF W2V", "XGBOOST", 200, 2, 0.70])


print(x)
```

```
+--------------+---------------+--------------+-----------+------+
|  Vectorizer  |     Model     | N Estimators | Max Depth | AUC  |
+--------------+---------------+--------------+-----------+------+
| Bag of Words | Random Forest |     500      |     10    | 0.7  |
|    TFIDF     | Random Forest |     1000     |     10    | 0.71 |
|   AVG W2V    | Random Forest |     1000     |     7     | 0.7  |
|  TFIDF W2V   | Random Forest |     1000     |     6     | 0.69 |
| Bag of Words |    XGBOOST    |     500      |     2     | 0.72 |
|    TFIDF     |    XGBOOST    |     200      |     3     | 0.72 |
|   AVG W2V    |    XGBOOST    |     300      |     2     | 0.71 |
|  TFIDF W2V   |    XGBOOST    |     200      |     2     | 0.7  |
+--------------+---------------+--------------+-----------+------+
```

In [ ]: