DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

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

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
warnings.filterwarnings("ignore")
```

```
C:\Users\Utkarsh Sri\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected
Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('E:\\Machine Learning\\Dataset\\train_data.csv')
resource_data = pd.read_csv('E:\\Machine Learning\\Dataset\\resources.csv')
```

In [3]:

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

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

In [4]:

```
# 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)
project_data.project_is_approved.value_counts()
```

Out[4]:

```
1    92706
0    16542
Name: project_is_approved, dtype: int64
```

In [5]:

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

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

Out[5]:

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

1.2 Preprocessing of `project_subject_categories`

In [6]:

```
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 [7]:

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

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

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

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

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

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

1.4 Text preprocessing

In [8]:

```
# 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 [9]:

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

Out[9]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_t
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Enginee STEAM the Prin Classro
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Sens Tools Fo

In [10]:

```
# 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)
```

```
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
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.
==================================================
```

In [11]:

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

```
def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [12]:

```
sent = decontracted(project_data['essay'].values[2000])
print(sent)
print("="*50)
```

```
\"Creativity is intelligence having fun.\" --Albert Einstein. Our elementary library at Greenville
Elementary is anything but a quiet, hushed space. It is a place for collaboration and research. It
is a place for incorporating technology. It is a place for innovation. And it is a place for creat
ing.Our school serves 350 third and fourth graders who primarily live in rural and poverty-stricke
n areas in our community. Being a Title I school, approximately 85% of them receive free or
reduced lunch. But they are inquisitive, creative, and eager to learn. They love visiting the libr
ary to check out books, hear \r\nstories, create digital stories, and use the computer lab for lea
rning and fun. We want to build our library is Makerspace with activities revolving around art and
literacy to provide more engaging, hands-on activities.We want to begin \"Makerspace Fridays!\"
Our school recently received a $1000 grant for books for our arts-integrated Makerspace. We have r
eceived titles such as \"Origami for Everyone,\" \"How to Make Stuff with Ducktape,\" and \"Cool E
ngineering Activities for Girls.\"  We now need supplies to correlate with these new informational
texts. By adding these art and craft supplies, students will be able to design and create
masterpieces related to their coursework. \r\n\r\nFor example, while studying Native Americans, st
udents can use the looms and yarn to recreate Navajo and/or Pueblo weaving. Weaving can also be in
tegrated with literacy through Greek mythology and the story of Arachne.\r\n\r\nCreating art with
perler beads has many possibilities! Students can design their own animals after studying their ch
aracteristics. They can use symmetry and patterning to create one-of-a-kind originals.
\r\n\r\nOrigami reinforces geometry, thinking skills, fractions, problem-solving, and just fun sci
ence!Our students need to be able to apply what they read and learn. If they read a how-to book, t
hey will apply that reading through a hands-on art activity and actually create a product. This is
a crucial skill in the real world. By creating and designing their own masterpieces, they are usin
g many critical thinking skills. Students will become more analytical thinkers.
==================================================
```

In [13]:

```
# \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)
```

```
 Creativity is intelligence having fun.  --Albert Einstein. Our elementary library at Greenville
Elementary is anything but a quiet, hushed space. It is a place for collaboration and research. It
is a place for incorporating technology. It is a place for innovation. And it is a place for creat
ing.Our school serves 350 third and fourth graders who primarily live in rural and poverty-stricke
n areas in our community. Being a Title I school, approximately 85% of them receive free or
reduced lunch. But they are inquisitive, creative, and eager to learn. They love visiting the libr
ary to check out books, hear   stories, create digital stories, and use the computer lab for learn
ing and fun. We want to build our library is Makerspace with activities revolving around art and l
iteracy to provide more engaging, hands-on activities.We want to begin  Makerspace Fridays!  Our s
chool recently received a $1000 grant for books for our arts-integrated Makerspace. We have receiv
ed titles such as  Origami for Everyone,   How to Make Stuff with Ducktape,  and  Cool Engineering
Activities for Girls.   We now need supplies to correlate with these new informational texts. By a
dding these art and craft supplies, students will be able to design and create masterpieces relate
d to their coursework.     For example, while studying Native Americans, students can use the loom
s and yarn to recreate Navajo and/or Pueblo weaving. Weaving can also be integrated with literacy
through Greek mythology and the story of Arachne.     Creating art with perler beads has many possi
bilities! Students can design their own animals after studying their characteristics. They can use
symmetry and patterning to create one-of-a-kind originals.     Origami reinforces geometry,
thinking skills, fractions, problem-solving, and just fun science!Our students need to be able to
apply what they read and learn. If they read a how-to book, they will apply that reading through a
hands-on art activity and actually create a product. This is a crucial skill in the real world. By
creating and designing their own masterpieces, they are using many critical thinking skills. Stude
nts will become more analytical thinkers.
```

In [14]:

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

```
 Creativity is intelligence having fun Albert Einstein Our elementary library at Greenville
Elementary is anything but a quiet hushed space It is a place for collaboration and research It is
a place for incorporating technology It is a place for innovation And it is a place for creating O
ur school serves 350 third and fourth graders who primarily live in rural and poverty stricken are
as in our community Being a Title I school approximately 85 of them receive free or reduced lunch
But they are inquisitive creative and eager to learn They love visiting the library to check out b
ooks hear stories create digital stories and use the computer lab for learning and fun We want to
build our library is Makerspace with activities revolving around art and literacy to provide more
engaging hands on activities We want to begin Makerspace Fridays Our school recently received a 10
00 grant for books for our arts integrated Makerspace We have received titles such as Origami for
Everyone How to Make Stuff with Ducktape and Cool Engineering Activities for Girls We now need sup
plies to correlate with these new informational texts By adding these art and craft supplies
students will be able to design and create masterpieces related to their coursework For example wh
ile studying Native Americans students can use the looms and yarn to recreate Navajo and or Pueblo
weaving Weaving can also be integrated with literacy through Greek mythology and the story of Arac
hne Creating art with perler beads has many possibilities Students can design their own animals af
ter studying their characteristics They can use symmetry and patterning to create one of a kind or
iginals Origami reinforces geometry thinking skills fractions problem solving and just fun science
Our students need to be able to apply what they read and learn If they read a how to book they wil
l apply that reading through a hands on art activity and actually create a product This is a cruci
al skill in the real world By creating and designing their own masterpieces they are using many cr
itical thinking skills Students will become more analytical thinkers
```

In [15]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', '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 [16]:

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

```
100%|██████████████████████████████████████████████████████████████████████████| 109248/109248
[01:14<00:00, 1464.74it/s]
```

In [17]:

```
# after preprocesing
print(preprocessed_essays[2000])
project_data['preprocessed_essays']=preprocessed_essays
```

```
creativity intelligence fun albert einstein elementary library greenville elementary anything quie
t hushed space place collaboration research place incorporating technology place innovation place c
reating school serves 350 third fourth graders primarily live rural poverty stricken areas
community title school approximately 85 receive free reduced lunch inquisitive creative eager
learn love visiting library check books hear stories create digital stories use computer lab learn
ing fun want build library makerspace activities revolving around art literacy provide engaging ha
nds activities want begin makerspace fridays school recently received 1000 grant books arts
integrated makerspace received titles origami everyone make stuff ducktape cool engineering activi
ties girls need supplies correlate new informational texts adding art craft supplies students able
design create masterpieces related coursework example studying native americans students use looms
yarn recreate navajo pueblo weaving weaving also integrated literacy greek mythology story arachne
creating art perler beads many possibilities students design animals studying characteristics use
symmetry patterning create one kind originals origami reinforces geometry thinking skills
fractions problem solving fun science students need able apply read learn read book apply reading
hands art activity actually create product crucial skill real world creating designing
masterpieces using many critical thinking skills students become analytical thinkers
```

1.4.1 Converting Essay to Number of Words

In [18]:

```
project_data['totalwords_essay'] = project_data['preprocessed_essays'].str.split().str.len()
```

1.5 Preprocessing of `project_title`

In [19]:

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

```
100%|█████████████████████████████████████████████████████████████████████████| 109248/109248
[00:03<00:00, 31213.77it/s]
```

In [20]:

```
print(preprocessed_project_title[2000])
print("="*50)
project_data['preprocessed_project_title']=preprocessed_project_title
project_data.head(5)
```

```
empowering students art makerspace
==================================================
```

Out[20]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_t
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Enginee STEAM the Prim Classro
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Sens Tools Fo
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Grades PreK-2	Mo Learn wi Mo Lister Ce
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	Flex Seating Flex Learn
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	Going De The Ar Ir Think

5 rows × 21 columns

1.5.1 Converting Title to Number of Words

In [21]:

```
project_data['totalwords_title'] =
project_data['preprocessed_project_title'].str.split().str.len()
```

1.6 Preprocessing Grades

In [22]:

```
#Preprocessing grades i.e removing all the spaces from the grades and replace - by _



pre_grades = []

for grade in tqdm(project_data['project_grade_category'].values):
    sent = re.sub('[^A-Za-z0-9]+', '_', grade)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    pre_grades.append(sent.lower().strip())


project_data['grade_category']=pre_grades
```

```
100%|██████████████████████████████████████████████████████████████████████| 109248/109248
[00:00<00:00, 154289.09it/s]
```

1.7 Preparing data for models

1.7 Preparing data for models

In [23]:

```
project_data.columns
```

Out[23]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', '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',
       'clean_categories', 'clean_subcategories', 'essay',
       'preprocessed_essays', 'totalwords_essay', 'preprocessed_project_title',
       'totalwords_title', 'grade_category'],
      dtype='object')
```

we are going to consider

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

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

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

1.8 Using Pretrained Models: Avg W2V

In [60]:

```
# 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('E:\Machine Learning\glove.42B.300d.txt')
```

```
Loading Glove Model
```

```
1917495it [06:51, 4664.81it/s]
```

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

In [61]:

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

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

```
all the words in the coupus 15568853
the unique words in the coupus 59501
The number of words that are present in both glove vectors and our coupus 51613 ( 86.743 %)
word 2 vec length 51613
```

In [62]:

```
# 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 [27]:

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

```
[nltk_data] Error loading vader_lexicon: <urlopen error [Errno 11001]
[nltk_data]     getaddrinfo failed>
```

Out[27]:

```
False
```

1.9 Calculating Sentiment score(Taking Compound in Consideration)

In [28]:

```
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\
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)

print(ss['compound'])

for k in ss:
    print(k)
    #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
```

```
0.9975
neg
neu
pos
compound
```

In [49]:

```
#https://programminghistorian.org/en/lessons/sentiment-analysis
#The “neg”, “neu”, and “pos” values describe the fraction of weighted scores that fall into each c
ategory. VADER also sums all weighted scores to calculate a “compound” value normalized between -1
and 1
#this value attempts to describe the overall affect of the entire text from strongly negative (-1)
to strongly positive (1).

sscore=[]
sid = SentimentIntensityAnalyzer()

for essay in tqdm(project_data['preprocessed_essays']):


    for_sentiment = essay
    ss = sid.polarity_scores(for_sentiment)
    sscore.append(ss['neu'])

project_data['sscore']=sscore
```

```
100%|██████████████████████████████████████████████████████████| 109248/109248
[03:55<00:00, 464.58it/s]
```

1.10 Combining project_data & resources

In [24]:

```
#Combining the data from the resources from the project data and resoure file for quantity and pri
ce

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

In [25]:

```
#replacing all the nan values from the teacher prefix to blank_space

project_data.teacher_prefix=project_data.teacher_prefix.fillna('')
```

In [26]:

```
#Seprating the values of approved projects from the whole data i.e removing the target value from
the data
X=project_data
```

In [27]:

```
y =X['project_is_approved'].values
#X.drop(['project_is_approved'], axis=1, inplace=True)
X.head(5)
y.shape
```

Out[27]:

```
(109248,)
```

Assignment 8: Apply RF & GBDT

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

    - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(BOW) + preprocessed_eassay (BOW)
    - Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
    - Set 3: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
    - Set 4: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

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

    - Find the best hyper parameter which will give the maximum AUC value
    - find the best hyper paramter using k-fold cross validation/simple cross validation data
    - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task

3. **Representation of results**

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

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

    or

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

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

4. **Conclusion**

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

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-leakag, 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.

2.0 Random Forest & GBDT

2.1 Spliting of data

In [28]:

```
#spliting of data using train_test_split

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.25, stratify=y_train)


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

```
(61452, 25) (61452,)
(20484, 25) (20484,)
(27312, 25) (27312,)
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', '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',
       'clean_categories', 'clean_subcategories', 'essay',
       'preprocessed_essays', 'totalwords_essay', 'preprocessed_project_title',
       'totalwords_title', 'grade_category', 'price', 'quantity'],
      dtype='object')
```

2.2 Vectorizing Numericals Features

2.2.1 Price Standarized

In [29]:

```
from sklearn.preprocessing import StandardScaler
price_scalar = StandardScaler()

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

X_train_price_norm = price_scalar.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_norm =price_scalar.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = price_scalar.transform(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
(61452, 1) (61452,)
(20484, 1) (20484,)
(27312, 1) (27312,)
====================================================================================================
----------------------------------------------------------------------------------------------------
```

2.2.2 Teacher_number_of_previously_posted_projects standardized

In [30]:

```
from sklearn.preprocessing import StandardScaler
price_scalar = StandardScaler()

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

X_train_pp_norm = price_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].v
alues.reshape(-1,1))
X_cv_pp_norm =price_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.r
eshape(-1,1))
X_test_pp_norm =
price_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))


print("After vectorizations")
print(X_train_pp_norm.shape, y_train.shape)
print(X_cv_pp_norm.shape, y_cv.shape)
print(X_test_pp_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(61452, 1) (61452,)
(20484, 1) (20484,)
(27312, 1) (27312,)
====================================================================================================
```

2.2.3 Quantity Standarized

In [31]:

```
from sklearn.preprocessing import StandardScaler
price_scalar = StandardScaler()

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

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

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)
```

```
(61452, 1) (61452,)
(20484, 1) (20484,)
(27312, 1) (27312,)
====================================================================================================
```

2.3 Vectorizing Categorical features

2.3.1 Vectorizing School_state

In [32]:

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

    value_count = X_train[feature].value_counts()
    donar_dict = dict()
    for i, denominator in value_count.items():

        vec = []
        for k in range(0,2):

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


            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 20*alpha))


        donar_dict[i]=vec
    return donar_dict


def get_donar_feature(alpha, feature, df):

    donar_dict = get_donar_fea_dict(alpha, feature, df)
    value_count = X_train[feature].value_counts()

    donar_fea = []
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            donar_fea.append(donar_dict[row[feature]])
        else:
            donar_fea.append([1/2,1/2])
    return donar_fea
```

In [33]:

```
# alpha is used for laplace smoothing
alpha = 1
X_train_school_res = np.array(get_donar_feature(alpha, "school_state",X_train))
X_test_school_res = np.array(get_donar_feature(alpha, "school_state", X_test))
X_cv_school_res = np.array(get_donar_feature(alpha, "school_state", X_cv))
```

In [34]:

```
print(X_train_school_res.shape, y_train.shape)
print(X_cv_school_res.shape, y_cv.shape)
print(X_test_school_res.shape, y_test.shape)
print("="*100)
```

```
(61452, 2) (61452,)
(20484, 2) (20484,)
(27312, 2) (27312,)
====================================================================================================
```

2.3.2 Vectorizing teacher_prefix

In [35]:

```
alpha = 1

X_train_pre_res = np.array(get_donar_feature(alpha, "teacher_prefix",X_train))
X_test_pre_res = np.array(get_donar_feature(alpha, "teacher_prefix", X_test))
X_cv_pre_res = np.array(get_donar_feature(alpha, "teacher_prefix", X_cv))
print(X_train_pre_res.shape, y_train.shape)
print(X_cv_pre_res.shape, y_cv.shape)
print(X_test_pre_res.shape, y_test.shape)
print("="*100)
```

```
(61452, 2) (61452,)
(20484, 2) (20484,)
(27312, 2) (27312,)
====================================================================================================
```

2.3.3 Vectorizing grade_category

In [36]:

```
alpha = 1

X_train_grade_res = np.array(get_donar_feature(alpha, "grade_category",X_train))
X_test_grade_res = np.array(get_donar_feature(alpha, "grade_category", X_test))
X_cv_grade_res = np.array(get_donar_feature(alpha, "grade_category", X_cv))
print(X_train_grade_res.shape, y_train.shape)
print(X_cv_grade_res.shape, y_cv.shape)
print(X_test_grade_res.shape, y_test.shape)
print("="*100)
```

```
(61452, 2) (61452,)
(20484, 2) (20484,)
(27312, 2) (27312,)
====================================================================================================
```

2.3.4 Vectorizing clean_categories

In [37]:

```
alpha = 1

X_train_cat_res = np.array(get_donar_feature(alpha, "clean_categories",X_train))
X_test_cat_res = np.array(get_donar_feature(alpha, "clean_categories", X_test))
X_cv_cat_res = np.array(get_donar_feature(alpha, "clean_categories", X_cv))
print(X_train_cat_res.shape, y_train.shape)
print(X_cv_cat_res.shape, y_cv.shape)
print(X_test_cat_res.shape, y_test.shape)
print("="*100)
```

```
(61452, 2) (61452,)
(20484, 2) (20484,)
(27312, 2) (27312,)
====================================================================================================
```

2.3.5 Vectorizing clean_subcategories

In [38]:

```
alpha = 1
X_train_scat_res= np.array(get_donar_feature(alpha, "clean_subcategories",X_train))
X_test_scat_res= np.array(get_donar_feature(alpha, "clean_subcategories", X_test))
X_cv_scat_res = np.array(get_donar_feature(alpha, "clean_subcategories", X_cv))
print(X_train_scat_res.shape, y_train.shape)
print(X_cv_scat_res.shape, y_cv.shape)
print(X_test_scat_res.shape, y_test.shape)
print("="*100)
```

```
(61452, 2) (61452,)
(20484, 2) (20484,)
(27312, 2) (27312,)
====================================================================================================
```

2.4 Encoding categorical & numerical features

In [39]:

```
#combining all the numerical and categorical values togeather

from scipy.sparse import hstack
```

```
X_tr_com = np.hstack((X_train_school_res,X_train_grade_res,X_train_pre_res,X_train_cat_res,X_train_scat_res,X_train_quantity_norm,X_train_price_norm,X_train_quantity_norm))
X_cr_com = np.hstack((X_cv_school_res,X_cv_grade_res,X_cv_pre_res,X_cv_cat_res,X_cv_scat_res,X_cv_quantity_norm,X_cv_price_norm,X_cv_quantity_norm))
X_te_com = np.hstack((X_test_school_res,X_test_grade_res,X_test_pre_res,X_test_cat_res,X_test_scat_res,X_test_quantity_norm,X_test_price_norm,X_test_quantity_norm))

print("Final Data matrix")
print(X_tr_com.shape, y_train.shape)
print(X_cr_com.shape, y_cv.shape)
print(X_te_com.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(61452, 13) (61452,)
(20484, 13) (20484,)
(27312, 13) (27312,)
====================================================================================================
```

2.4 Appling RF & GBDT on different kind of featurization as mentioned in the instructions

Apply RF & 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 instructions

In [40]:

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

2.4.1 Applying RF & GBDT on BOW, SET 1

2.4.1.1 Converting project_title & essay into BOW

In [41]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['preprocessed_essays'].values) # fit has to happen only on train data


X_train_essay_bow = vectorizer.transform(X_train['preprocessed_essays'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['preprocessed_essays'].values)
X_test_essay_bow = vectorizer.transform(X_test['preprocessed_essays'].values)

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("="*100)
```

```
essay_fea=vectorizer.get_feature_names()
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['preprocessed_project_title'].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['preprocessed_project_title'].values)
X_cv_title_bow = vectorizer.transform(X_cv['preprocessed_project_title'].values)
X_test_title_bow = vectorizer.transform(X_test['preprocessed_project_title'].values)

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("="*100)
```

```
After vectorizations
(61452, 5000) (61452,)
(20484, 5000) (20484,)
(27312, 5000) (27312,)
====================================================================================================

After vectorizations
(61452, 4582) (61452,)
(20484, 4582) (20484,)
(27312, 4582) (27312,)
====================================================================================================
```

2.4.1.2 Encoding numerical,categorical & BOW

In [42]:

```
from scipy.sparse import hstack
X_tr_bow = hstack((X_train_essay_bow, X_train_title_bow,X_tr_com)).tocsr()
X_cr_bow = hstack((X_cv_essay_bow, X_cv_title_bow,X_cr_com)).tocsr()
X_te_bow = hstack((X_test_essay_bow, X_test_title_bow,X_te_com)).tocsr()

print("Final Data matrix")
print(X_tr_bow.shape, y_train.shape)
print(X_cr_bow.shape, y_cv.shape)
print(X_te_bow.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(61452, 9595) (61452,)
(20484, 9595) (20484,)
(27312, 9595) (27312,)
====================================================================================================
```

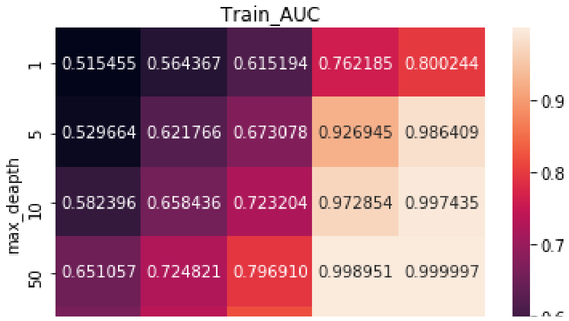
2.4.1.3 Applying RF

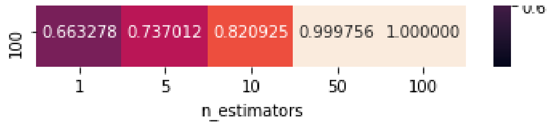
In [48]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
n_est=[1,5,10,50,100]
max_de=[1,5,10,50,100]
for i in n_est:
    for j in max_de:

        clf=  RandomForestClassifier(n_estimators=i,max_depth=j)
        clf.fit(X_tr_bow, y_train)

        y_train_pred = batch_predict(clf, X_tr_bow)
        y_cv_pred = batch_predict(clf, X_cr_bow)

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

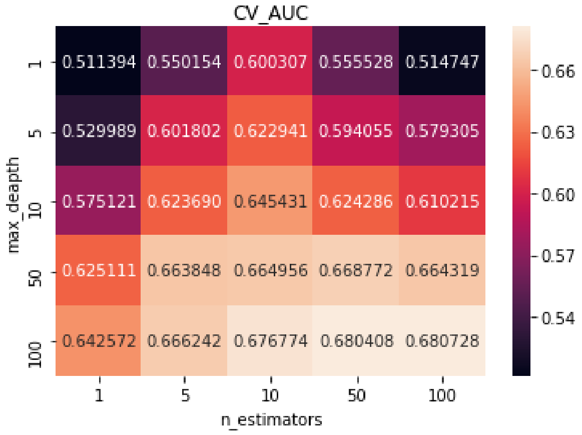
In [85]:

```
l=train_auc
ll=cv_auc
xx=[l[i:i+5] for i in range(0, len(l), 5)]
xxx=[ll[i:i+5] for i in range(0, len(ll), 5)]


import seaborn as sns

df_cm = pd.DataFrame(xx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("Train_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()

print("="*100)
df_cm = pd.DataFrame(xxx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("CV_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()
```





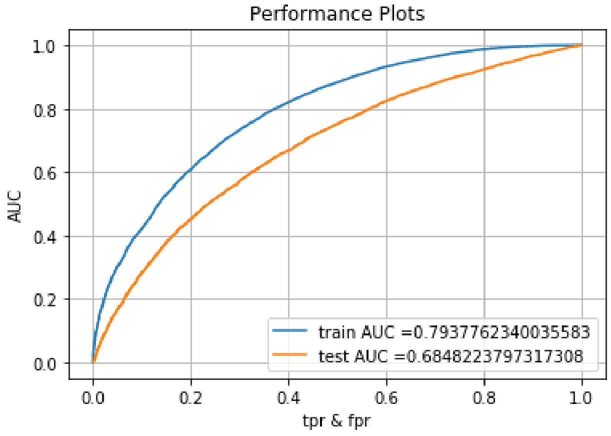
```
==========================================================================================
```



Considering the Hyperparameter max_depth=10 & n_estimators=50

In [44]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc


clf= RandomForestClassifier(max_depth=10,n_estimators=50)

clf.fit(X_tr_bow, y_train)


y_train_pred =batch_predict(clf,X_tr_bow)
y_test_pred = batch_predict(clf,X_te_bow)

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("tpr & fpr")
plt.ylabel("AUC")
plt.title("Performance Plots")
plt.grid()
plt.show()
```



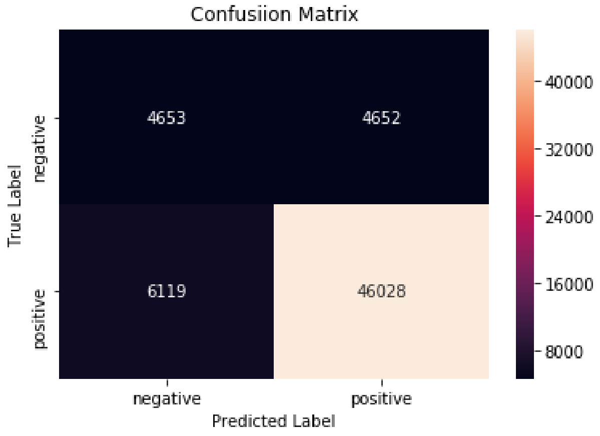
In [45]:

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

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

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

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

In [46]:

```
#https://www.kaggle.com/premvardhan/amazon-fine-food-reviews-analysis-using-knn

print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
cmtr=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmtr, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249999997112598 for threshold 0.831
```

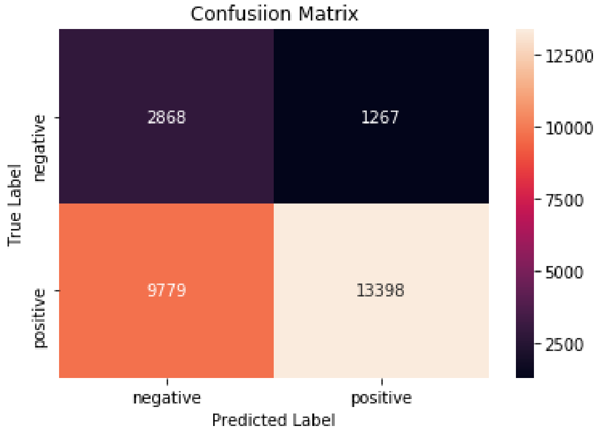


In [47]:

```
#https://www.kaggle.com/premvardhan/amazon-fine-food-reviews-analysis-using-knn

# plot confusion matrix to describe the performance of classifier.
import seaborn as sns
print("Test confusion matrix")

cmt=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmt, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Test confusion matrix
```

```
the maximum value of tpr*(1-fpr) 0.24999998537859927 for threshold 0.848
```



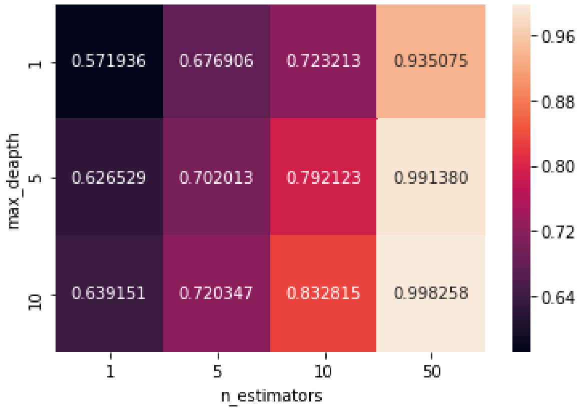
2.4.1.5 Applying GBDT

In [44]:

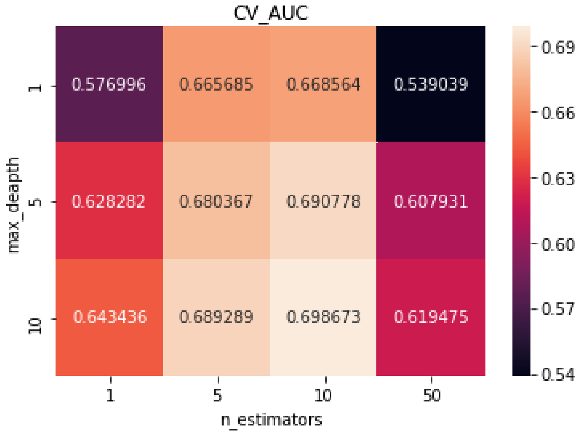
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
n_est=[1,5,10]
max_de=[1,5,10,50]
for i in n_est:
    for j in max_de:

        clf=  GradientBoostingClassifier(n_estimators=i,max_depth=j)
        clf.fit(X_tr_bow, y_train)

        y_train_pred = batch_predict(clf, X_tr_bow)
        y_cv_pred = batch_predict(clf, X_cr_bow)

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

In [50]:

```
l=train_auc
ll=cv_auc
xx=[l[i:i+4] for i in range(0, len(l), 4)]
xxx=[ll[i:i+4] for i in range(0, len(ll), 4)]


import seaborn as sns

df_cm = pd.DataFrame(xx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("Train_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()

print("="*100)
df_cm = pd.DataFrame(xxx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("CV_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()
```



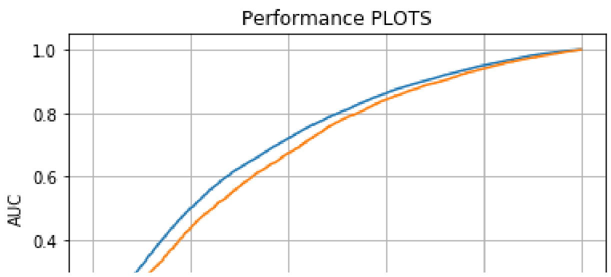
```
====================================================================================================
```

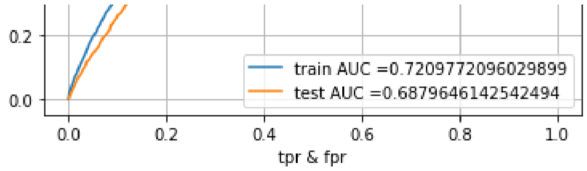

CV_AUC

Considering the Hyperparameter max_depth=5 & n_estimators=10

In [48]:

```
from sklearn.metrics import roc_curve, auc
from sklearn.ensemble import GradientBoostingClassifier

clf= GradientBoostingClassifier(max_depth=5,n_estimators=10)

clf.fit(X_tr_bow, y_train)


y_train_pred =batch_predict(clf,X_tr_bow)
y_test_pred = batch_predict(clf,X_te_bow)

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("tpr & fpr")
plt.ylabel("AUC")
plt.title("Performance PLOTS")
plt.grid()
plt.show()
```

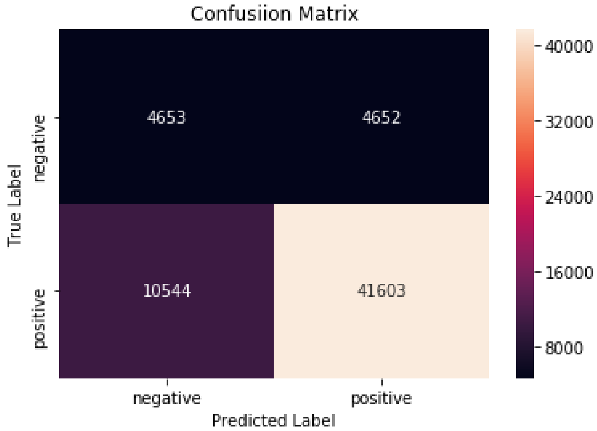

Performance PLOTS



In [49]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
cmtr=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmtr, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

print(cmtr)
```

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249999997112598 for threshold 0.815
```

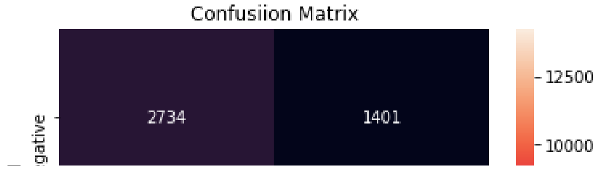


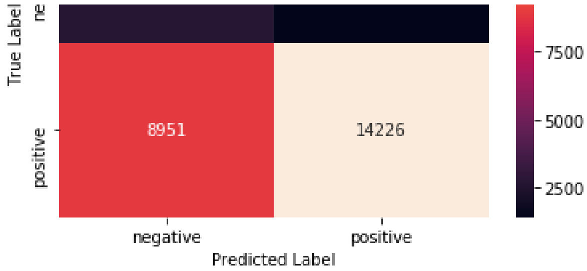
```
[[ 4653  4652]
 [10544 41603]]
```

In [50]:

```
print("Test confusion matrix")

cmt=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmt, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999998537859927 for threshold 0.853
```





2.4.2 Applying RF & GBDT on TFIDF, SET 2

2.4.2.1Converting Project_title & essay into tf-idf

In [51]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['preprocessed_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['preprocessed_essays'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['preprocessed_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['preprocessed_essays'].values)

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)


vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['preprocessed_project_title'].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['preprocessed_project_title'].values)
X_cv_title_tfidf = vectorizer.transform(X_cv['preprocessed_project_title'].values)
X_test_title_tfidf = vectorizer.transform(X_test['preprocessed_project_title'].values)

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

```
After vectorizations
(61452, 5000) (61452,)
(20484, 5000) (20484,)
(27312, 5000) (27312,)
====================================================================================================

After vectorizations
(61452, 5000) (61452,)
(20484, 5000) (20484,)
(27312, 5000) (27312,)
====================================================================================================
```

2.4.2.2 Encoding numerical,categorical & Tf-idf

In [52]:

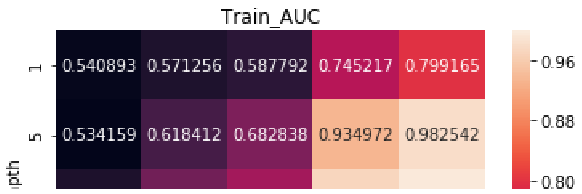
```
from scipy.sparse import hstack
X_tr_tfidf = hstack((X_train_essay_tfidf, X_train_title_tfidf,X_tr_com)).tocsr()
X_cr_tfidf = hstack((X_cv_essay_tfidf, X_cv_title_tfidf,X_cr_com)).tocsr()
X_te_tfidf = hstack((X_test_essay_tfidf, X_test_title_tfidf,X_te_com)).tocsr()

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

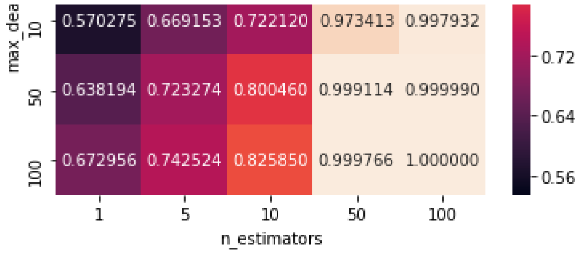
```
Final Data matrix
(61452, 10013) (61452,)
(20484, 10013) (20484,)
(27312, 10013) (27312,)
====================================================================================================
```

2.4.2.3 Applying RF

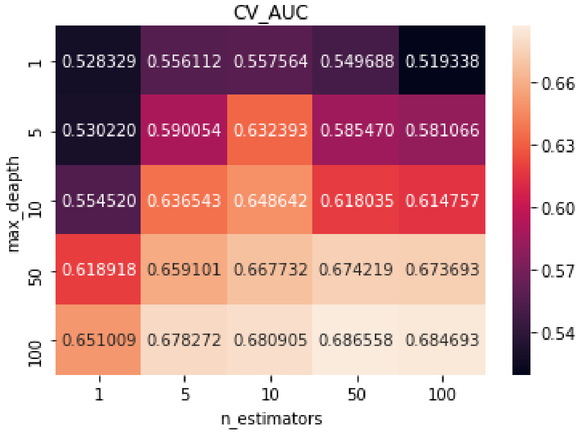
In [86]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []

max_de=[1,5,10,50,100]
n_est=[1,5,10,50,100]
for i in n_est:
    for j in max_de:

        clf=  RandomForestClassifier(n_estimators=i,max_depth=j)
        clf.fit(X_tr_tfidf, y_train)

        y_train_pred = batch_predict(clf, X_tr_tfidf)
        y_cv_pred = batch_predict(clf, X_cr_tfidf)

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






l=train_auc
ll=cv_auc
xx=[l[i:i+5] for i in range(0, len(l), 5)]
xxx=[ll[i:i+5] for i in range(0, len(ll), 5)]


import seaborn as sns

df_cm = pd.DataFrame(xx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("Train_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()

print("="*100)
df_cm = pd.DataFrame(xxx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("CV_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()
```





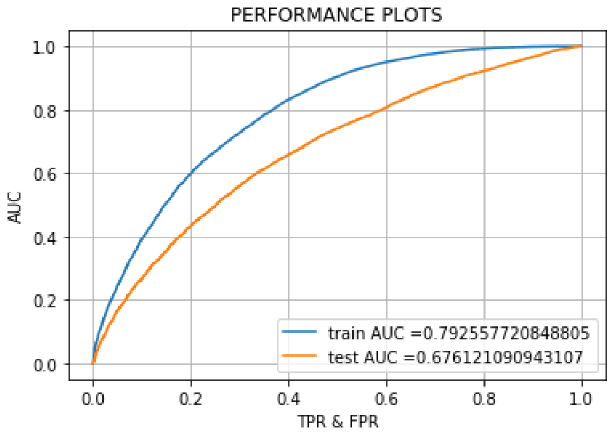
```
====================================================================================================================================
```



Considering the Hyperparameter max_depth=10 & n_estimators=50

In [53]:

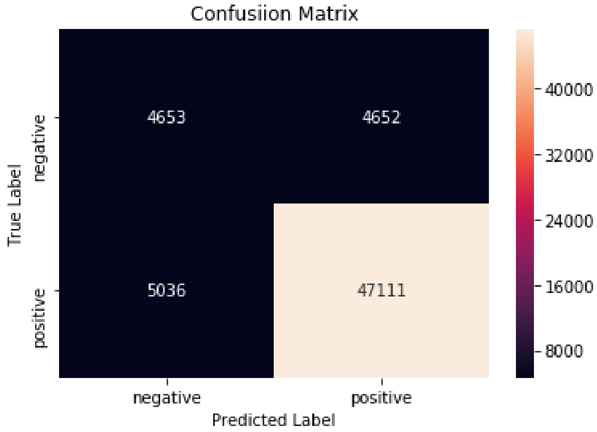
```
clf= RandomForestClassifier(max_depth=10,n_estimators=50)

clf.fit(X_tr_tfidf, y_train)


y_train_pred =batch_predict(clf,X_tr_tfidf)
y_test_pred = batch_predict(clf,X_te_tfidf)

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("TPR & FPR")
plt.ylabel("AUC")
plt.title("PERFORMANCE PLOTS")
plt.grid()
plt.show()
```



In [54]:

```
#https://www.kaggle.com/premvardhan/amazon-fine-food-reviews-analysis-using-knn
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
#print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
cmtr=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmtr, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

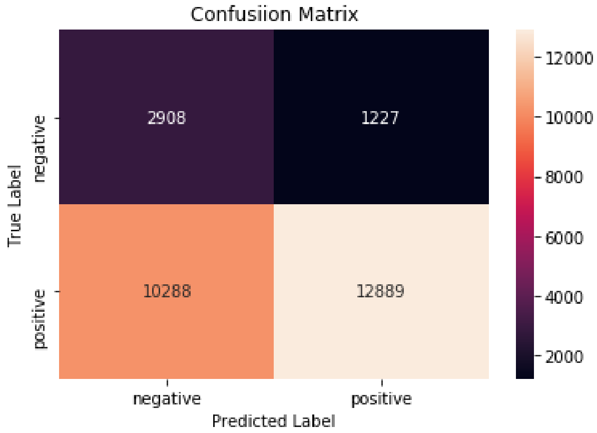
```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249999997112598 for threshold 0.83
```



In [55]:

```
#https://www.kaggle.com/premvardhan/amazon-fine-food-reviews-analysis-using-knn
print("Test confusion matrix")
cmt=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmt, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999998537859927 for threshold 0.851
```

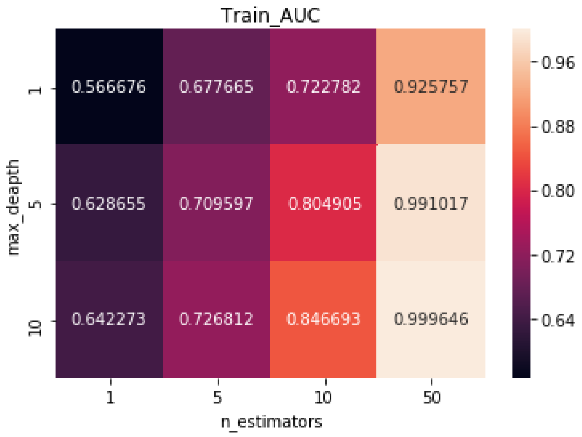


2.4.2.5 Applying GBDT

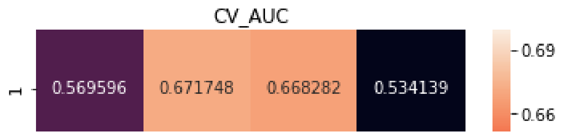
2.4.2.5 Applying GBDT

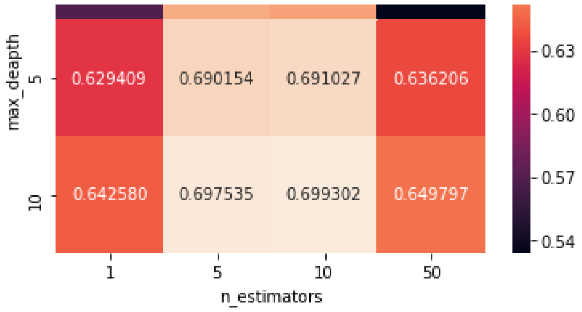
In [44]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
n_est=[1,5,10]
max_de=[1,5,10,50]
for i in n_est:
    for j in max_de:

        clf=  GradientBoostingClassifier(n_estimators=i,max_depth=j)
        clf.fit(X_tr_tfidf, y_train)

        y_train_pred = batch_predict(clf, X_tr_tfidf)
        y_cv_pred = batch_predict(clf, X_cr_tfidf)

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



l=train_auc
ll=cv_auc
xx=[l[i:i+4] for i in range(0, len(l), 4)]
xxx=[ll[i:i+4] for i in range(0, len(ll), 4)]


import seaborn as sns

df_cm = pd.DataFrame(xx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("Train_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()

print("="*100)
df_cm = pd.DataFrame(xxx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("CV_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()
```



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

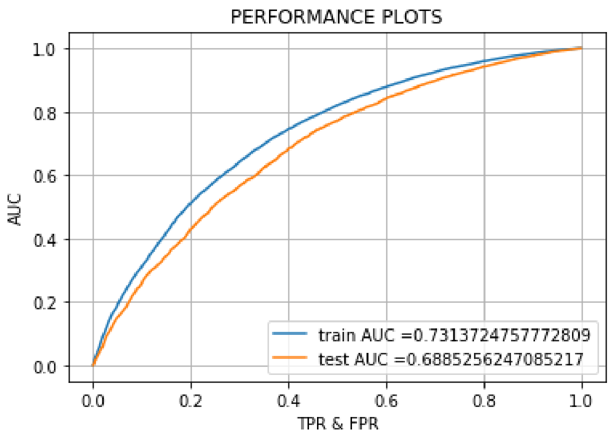




Considering the Hyperparameter max_depth=5 & n_estimators=10

In [59]:

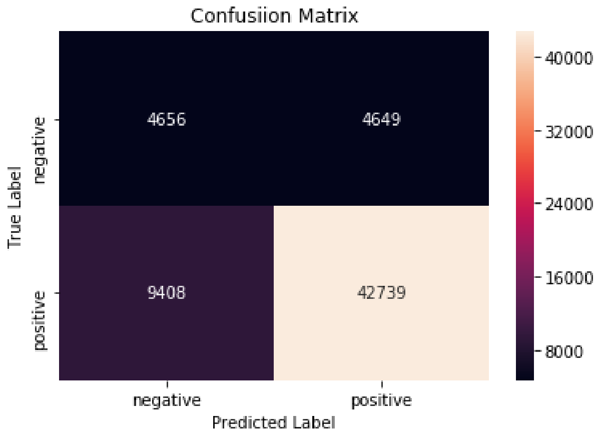
```
from sklearn.metrics import roc_curve, auc


clf= GradientBoostingClassifier(max_depth=5,n_estimators=10)

clf.fit(X_tr_tfidf, y_train)


y_train_pred =batch_predict(clf,X_tr_tfidf)
y_test_pred = batch_predict(clf,X_te_tfidf)

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("TPR & FPR")
plt.ylabel("AUC")
plt.title("PERFORMANCE PLOTS")
plt.grid()
plt.show()
```



In [57]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
cmtr=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmtr, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

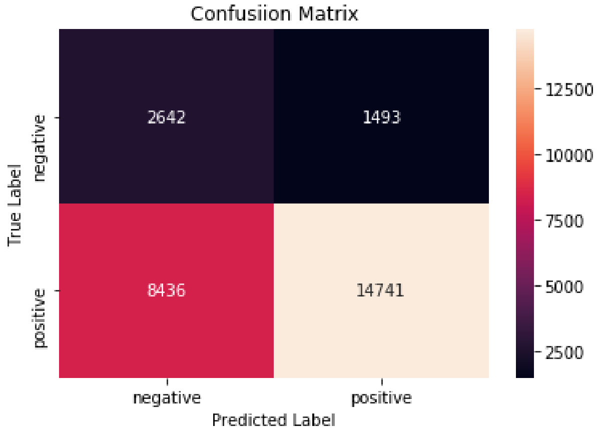
```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999985851730172 for threshold 0.815
```



In [58]:

```
print("Test confusion matrix")

cmt=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmt, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999998537859927 for threshold 0.852
```



2.4.3 Applying RF & GBDT on AVG W2V, SET 3

2.4.3.1 Converting Project_essay to Avg W2V

In [63]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
```

```
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train_essay.append(vector)

print(len(avg_w2v_vectors_train_essay))
print(len(avg_w2v_vectors_train_essay[0]))


avg_w2v_vectors_cv_essay = [];
for sentence in tqdm(X_cv['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv_essay.append(vector)

print(len(avg_w2v_vectors_cv_essay))
print(len(avg_w2v_vectors_cv_essay[0]))

avg_w2v_vectors_test_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test_essay.append(vector)

print(len(avg_w2v_vectors_test_essay))
print(len(avg_w2v_vectors_test_essay[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 61452/61452
[00:26<00:00, 2309.71it/s]
```

```
61452
300
```

```
100%|████████████████████████████████████████████████████████████████████████████| 20484/20484
[00:08<00:00, 2365.99it/s]
```

```
20484
300
```

```
100%|████████████████████████████████████████████████████████████████████████████| 27312/27312
[00:11<00:00, 2328.14it/s]
```

```
27312
300
```

2.4.3.2 Converting project_title to Avg W2V

In [64]:

```
avg_w2v_vectors_train_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train_title.append(vector)
```

```
print(len(avg_w2v_vectors_train_title))
print(len(avg_w2v_vectors_train_title[0]))


avg_w2v_vectors_cv_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv_title.append(vector)

print(len(avg_w2v_vectors_cv_title))
print(len(avg_w2v_vectors_cv_title[0]))

avg_w2v_vectors_test_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test_title.append(vector)

print(len(avg_w2v_vectors_test_title))
print(len(avg_w2v_vectors_test_title[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████████████| 61452/61452
[00:01<00:00, 41583.26it/s]
```

```
61452
300
```

```
100%|█████████████████████████████████████████████████████████████████████████████| 20484/20484
[00:00<00:00, 50774.55it/s]
```

```
20484
300
```

```
100%|█████████████████████████████████████████████████████████████████████████████| 27312/27312
[00:00<00:00, 41304.84it/s]
```

```
27312
300
```

2.4.3.3 Combing numerical,categorical & Avg W2V

In [65]:

```
from scipy.sparse import hstack
X_tr_w2v = np.hstack((avg_w2v_vectors_train_essay ,avg_w2v_vectors_train_title, X_tr_com))
X_cr_w2v = np.hstack((avg_w2v_vectors_cv_essay, avg_w2v_vectors_cv_title,X_cr_com))
X_te_w2v = np.hstack((avg_w2v_vectors_test_essay, avg_w2v_vectors_test_title,X_te_com))

print("Final Data matrix")
print(X_tr_w2v.shape, y_train.shape)
print(X_cr_w2v.shape, y_cv.shape)
print(X_te_w2v.shape, y_test.shape)
print("="*100)
```

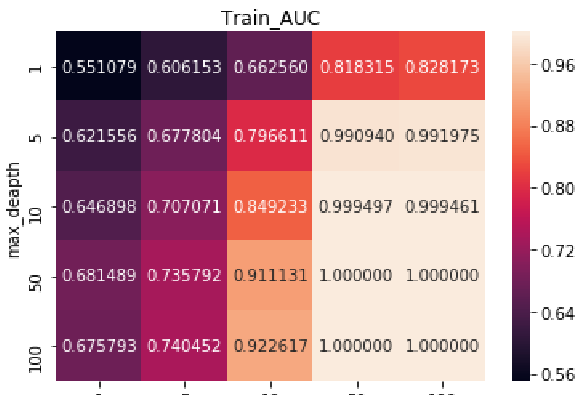
```
Final Data matrix
(61452, 613) (61452,)
(20484, 613) (20484,)
```

```
(20484, 613) (20484,)
(27312, 613) (27312,)
====================================================================================================
```

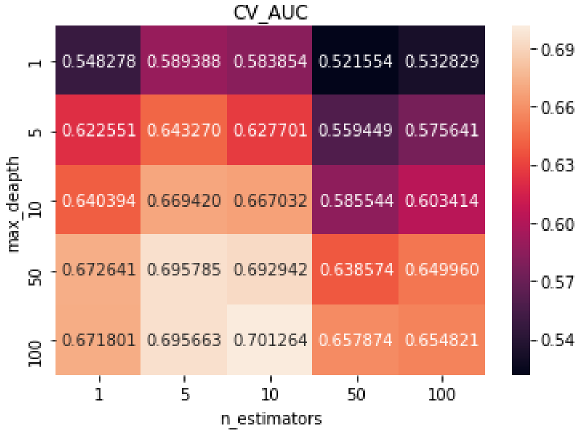
2.4.3.4 Applying RF

In [54]:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from mpl_toolkits.mplot3d import Axes3D

train_auc = []
cv_auc = []

max_de=[1,5,10,50,100]
n_est=[1,5,10,50,100]
for i in n_est:
    for j in max_de:

        clf=  RandomForestClassifier(n_estimators=i,max_depth=j)
        clf.fit(X_tr_w2v, y_train)

        y_train_pred = batch_predict(clf, X_tr_w2v)
        y_cv_pred = batch_predict(clf, X_cr_w2v)

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






l=train_auc
ll=cv_auc
xx=[l[i:i+5] for i in range(0, len(l), 5)]
xxx=[ll[i:i+5] for i in range(0, len(ll), 5)]


import seaborn as sns

df_cm = pd.DataFrame(xx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("Train_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()

print("="*100)
df_cm = pd.DataFrame(xxx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("CV_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()
```



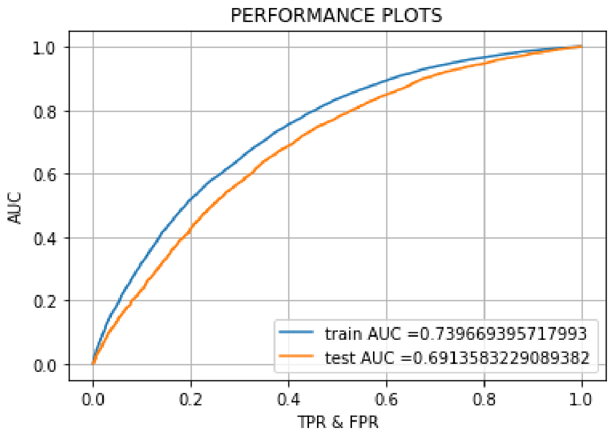
```
====================================================================================================
```



Considering the Hyperparameter max_depth=5 & n_estimators=50

In [66]:

```
clf= RandomForestClassifier(max_depth=5,n_estimators=50)

clf.fit(X_tr_w2v, y_train)


y_train_pred =batch_predict(clf,X_tr_w2v)
y_test_pred = batch_predict(clf,X_te_w2v)

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("TPR & FPR")
plt.ylabel("AUC")
plt.title("PERFORMANCE PLOTS")
plt.grid()
plt.show()
```

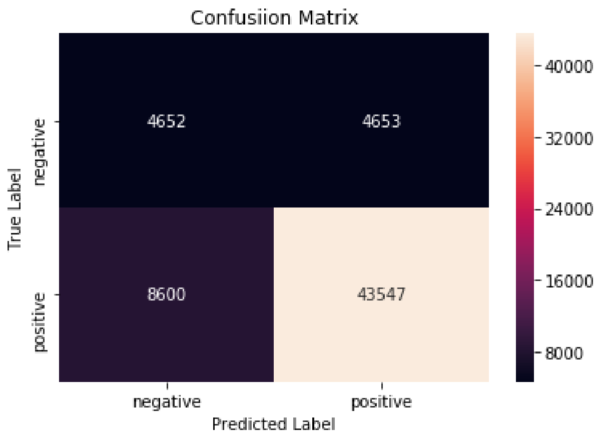


In [67]:

```
#https://www.kaggle.com/premvardhan/amazon-fine-food-reviews-analysis-using-knn
print("="*100)
from sklearn.metrics import confusion_matrix

print("Train confusion matrix")
```

```
cmtr=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmtr, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

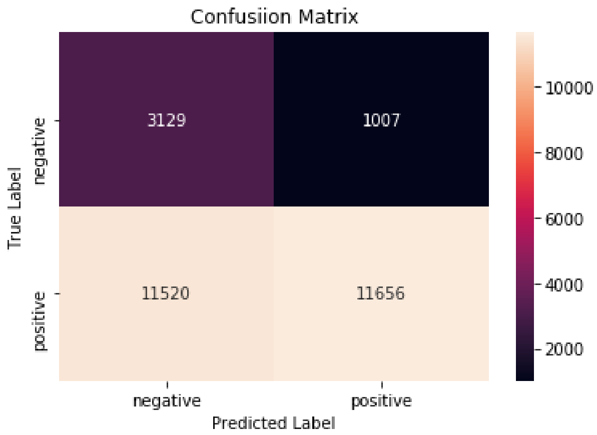
```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249999997112598 for threshold 0.825
```



In [62]:

```
#https://www.kaggle.com/premvardhan/amazon-fine-food-reviews-analysis-using-knn
import seaborn as sns
print("Test confusion matrix")

cmt=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmt, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

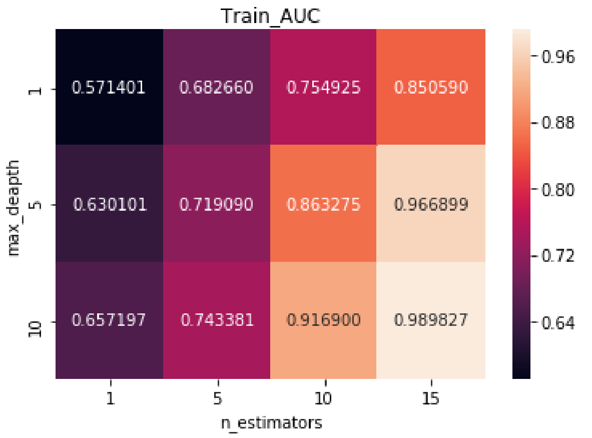
```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999994154267477 for threshold 0.855
```



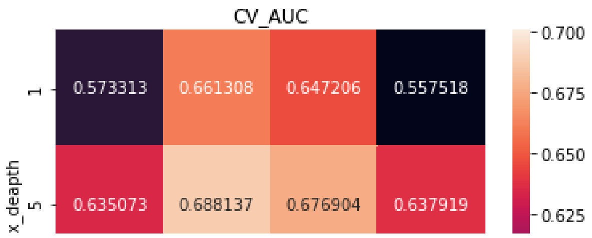
2.4.3.5 Applying GBDT

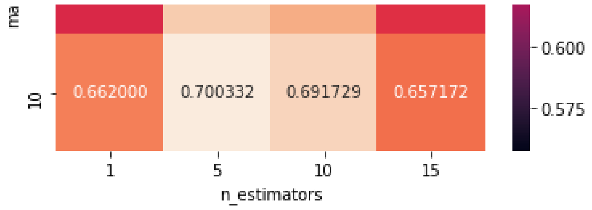
In [49]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
n_est=[1,5,10]
max_de=[1,5,10,15]
for i in n_est:
    for j in max_de:

        clf=  GradientBoostingClassifier(n_estimators=i,max_depth=j)
        clf.fit(X_tr_w2v, y_train)

        y_train_pred = batch_predict(clf, X_tr_w2v)
        y_cv_pred = batch_predict(clf, X_cr_w2v)

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



l=train_auc
ll=cv_auc
xx=[l[i:i+4] for i in range(0, len(l), 4)]
xxx=[ll[i:i+4] for i in range(0, len(ll), 4)]


import seaborn as sns

df_cm = pd.DataFrame(xx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("Train_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()

print("="*100)
df_cm = pd.DataFrame(xxx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("CV_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()
```



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

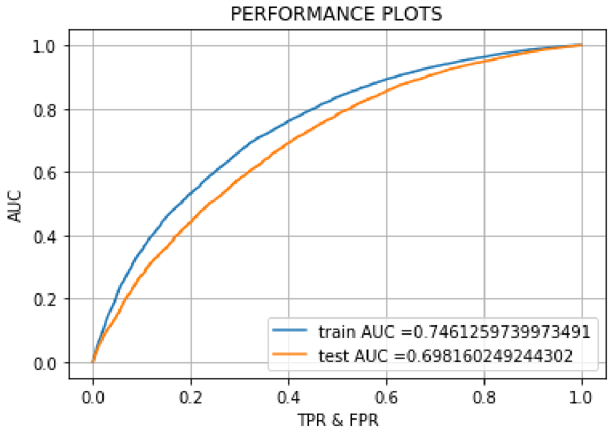




Considering the Hyperparameter max_depth=5 & n_estimators=10

In [68]:

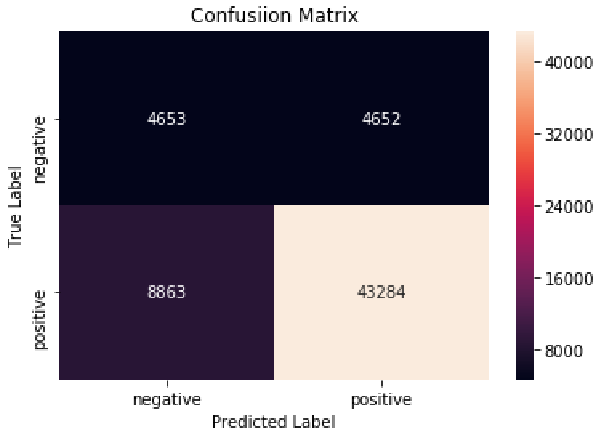
```
from sklearn.metrics import roc_curve, auc


clf= GradientBoostingClassifier(max_depth=5,n_estimators=10)

clf.fit(X_tr_w2v, y_train)


y_train_pred =batch_predict(clf,X_tr_w2v)
y_test_pred = batch_predict(clf,X_te_w2v)

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("TPR & FPR")
plt.ylabel("AUC")
plt.title("PERFORMANCE PLOTS")
plt.grid()
plt.show()
```



In [52]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
cmtr=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmtr, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

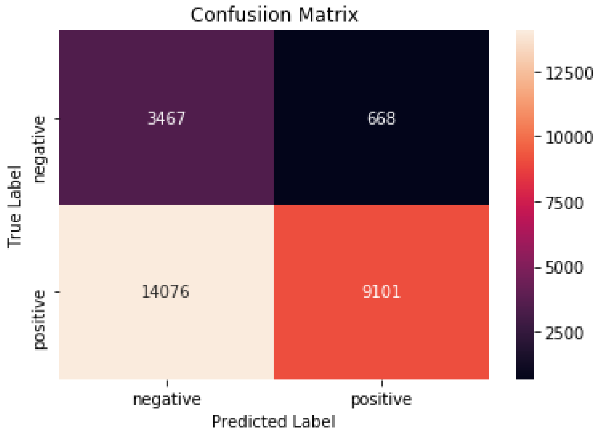
```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249999997112598 for threshold 0.819
```



In [53]:

```
print("Test confusion matrix")

cmt=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmt, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999998537859927 for threshold 0.876
```



2.4.4 Applying DT on TFIDF W2V, SET 4

2.4.4.1 Converting project_essay to TF-idf W2V

In [69]:

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


tfidf_w2v_vectors_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/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_essay.append(vector)

print(len(tfidf_w2v_vectors_essay))
print(len(tfidf_w2v_vectors_essay[0]))


tfidf_w2v_vectors_essay_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/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_essay_cv.append(vector)

print(len(tfidf_w2v_vectors_essay_cv))
print(len(tfidf_w2v_vectors_essay_cv[0]))



tfidf_w2v_vectors_essay_te = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/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_essay_te.append(vector)

print(len(tfidf_w2v_vectors_essay_te))
print(len(tfidf_w2v_vectors_essay_te[0]))
```

```
100%|███████████████████████████████████████████████████████████████████████████| 61452/61452 [02:
44<00:00, 374.06it/s]
```

```
61452
300
```

```
100%|███████████████████████████████████████████████████████████████████████████| 20484/20484 [00:
51<00:00, 398.88it/s]
```

```
20484
300
```

```
100%|███████████████████████████████████████████████████████████████████████████| 27312/27312 [01:
14<00:00, 367.55it/s]
```

```
27312
300
```

2.4.4.2 Converting project_title to TF-idf W2V

In [70]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_project_title'])
# 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())


tfidf_w2v_vectors_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_project_title']): # 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.append(vector)

print(len(tfidf_w2v_vectors_title))
print(len(tfidf_w2v_vectors_title[0]))



tfidf_w2v_vectors_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_project_title']): # 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]))


tfidf_w2v_vectors_title_te = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_project_title']): # 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_te.append(vector)

print(len(tfidf_w2v_vectors_title_te))
print(len(tfidf_w2v_vectors_title_te[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 61452/61452
[00:03<00:00, 18637.26it/s]
```

```
61452
300
```

```
100%|██████████████████████████████████████████████████████████████████████████| 20484/20484
[00:01<00:00, 18778.96it/s]
```

```
20484
300
```

```
100%|██████████████████████████████████████████████████████████████████████████| 27312/27312
[00:01<00:00, 22966.96it/s]
```

```
27312
300
```

2.4.4.3 Combing numerical,categorical features & tf-idf W2V

In [71]:

```
from scipy.sparse import hstack
X_tr_w2v_tfidf = np.hstack((tfidf_w2v_vectors_essay ,tfidf_w2v_vectors_title, X_tr_com))
X_cr_w2v_tfidf = np.hstack((tfidf_w2v_vectors_essay_cv, tfidf_w2v_vectors_title_cv,X_cr_com))
X_te_w2v_tfidf = np.hstack((tfidf_w2v_vectors_essay_te, tfidf_w2v_vectors_title_te,X_te_com))

print("Final Data matrix")
print(X_tr_w2v_tfidf.shape, y_train.shape)
print(X_cr_w2v_tfidf.shape, y_cv.shape)
print(X_te_w2v_tfidf.shape, y_test.shape)
print("="*100)
```

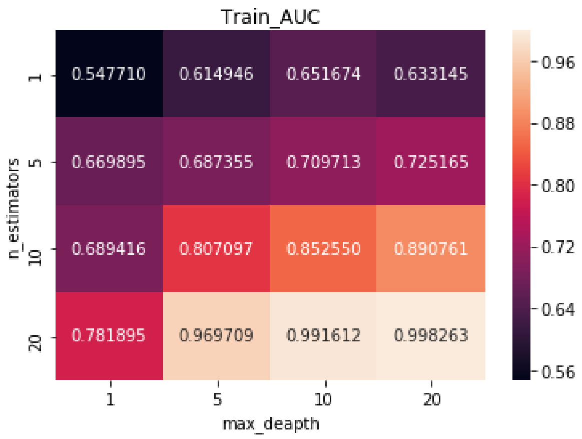
```
Final Data matrix
(61452, 613) (61452,)
(20484, 613) (20484,)
(27312, 613) (27312,)
====================================================================================================
```

2.4.4.4 Applying RF

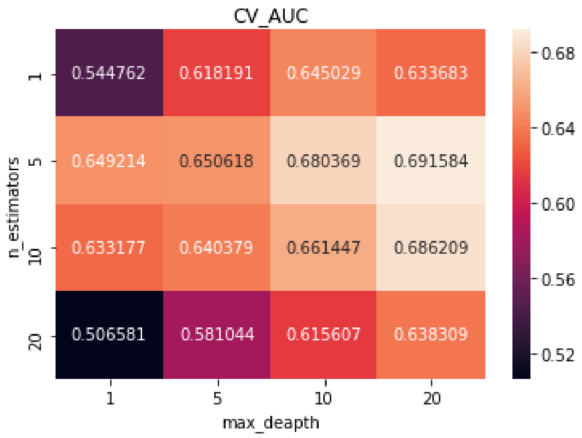
In [58]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from mpl_toolkits.mplot3d import Axes3D

train_auc = []
cv_auc = []

max_de=[1,5,10,20]
n_est=[1,5,10,20]
for i in max_de:
    for j in n_est:

        clf=  RandomForestClassifier(n_estimators=j,max_depth=i)
        clf.fit(X_tr_w2v_tfidf, y_train)

        y_train_pred = batch_predict(clf, X_tr_w2v_tfidf)
        y_cv_pred = batch_predict(clf, X_cr_w2v_tfidf)

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

```
l=train_auc
ll=cv_auc
xx=[l[i:i+4] for i in range(0, len(l), 4)]
xxx=[ll[i:i+4] for i in range(0, len(ll), 4)]


import seaborn as sns

df_cm = pd.DataFrame(xx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("Train_AUC")
plt.xlabel("max_deapth")
plt.ylabel("n_estimators")
plt.show()

print("="*100)
df_cm = pd.DataFrame(xxx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("CV_AUC")
plt.xlabel("max_deapth")
plt.ylabel("n_estimators")
plt.show()
```



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

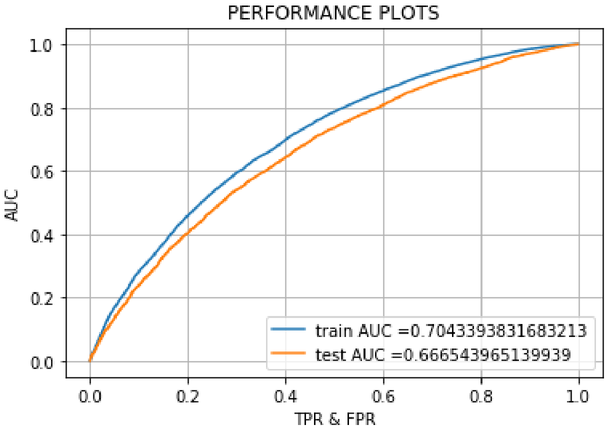


Considering the Hyperparameter max_depth=5 & n_estimators=10

In [72]:

```
from sklearn.ensemble import RandomForestClassifier
clf= RandomForestClassifier(max_depth=5,n_estimators=10)
clf.fit(X_tr_w2v_tfidf, y_train)
```

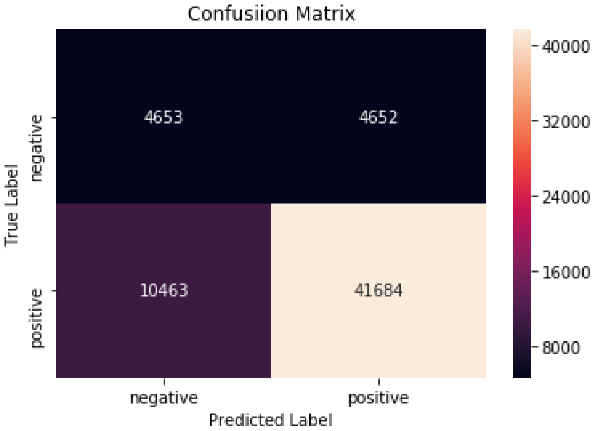
```
y_train_pred =batch_predict(clf,X_tr_w2v_tfidf)
y_test_pred = batch_predict(clf,X_te_w2v_tfidf)

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("TPR & FPR")
plt.ylabel("AUC")
plt.title("PERFORMANCE PLOTS")
plt.grid()
plt.show()
```



In [59]:

```
#https://www.kaggle.com/premvardhan/amazon-fine-food-reviews-analysis-using-knn
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")


cmtr=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmtr, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

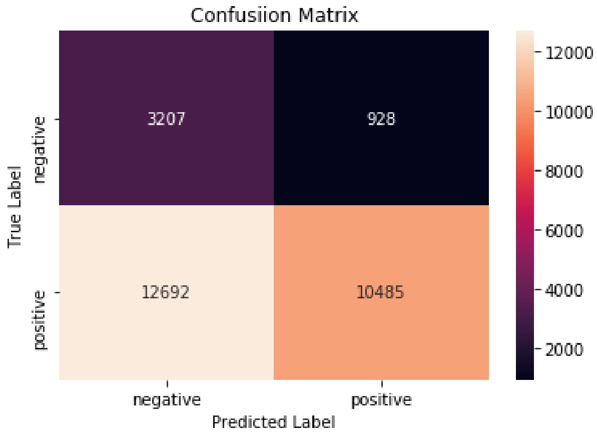
```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249999997112598 for threshold 0.826
```



In [60]:

```
#https://www.kaggle.com/premvardhan/amazon-fine-food-reviews-analysis-using-knn
print("="*100)
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")


cmt=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmt, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

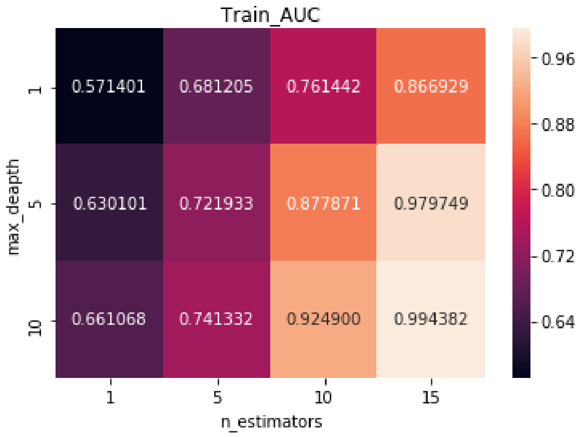
```
====================================================================================================

Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999998537859924 for threshold 0.861
```



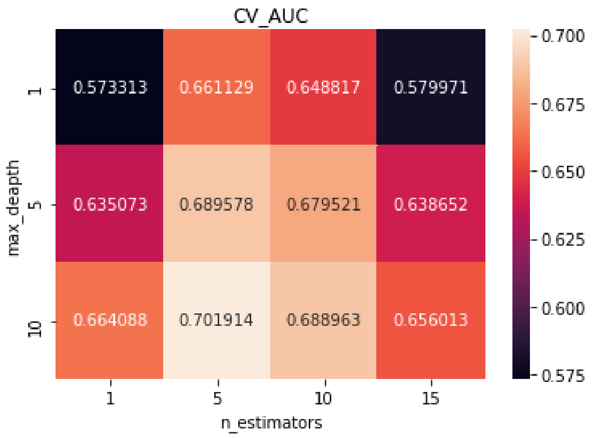
2.4.4.5 Applying GBDT

In [61]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
n_est=[1,5,10]
max_de=[1,5,10,15]
for i in n_est:
    for j in max_de:

        clf=  GradientBoostingClassifier(n_estimators=i,max_depth=j)
        clf.fit(X_tr_w2v_tfidf, y_train)

        y_train_pred = batch_predict(clf, X_tr_w2v_tfidf)
        y_cv_pred = batch_predict(clf, X_cr_w2v_tfidf)

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



l=train_auc
ll=cv_auc
xx=[l[i:i+4] for i in range(0, len(l), 4)]
xxx=[ll[i:i+4] for i in range(0, len(ll), 4)]


import seaborn as sns

df_cm = pd.DataFrame(xx, index =n_est, columns =max_de)
df_cm = pd.DataFrame(xx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("Train_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()

print("="*100)
df_cm = pd.DataFrame(xxx, index =n_est, columns =max_de)
sns.heatmap(df_cm, annot = True, fmt = "f")
plt.title("CV_AUC")
plt.xlabel("n_estimators")
plt.ylabel("max_deapth")
plt.show()
```



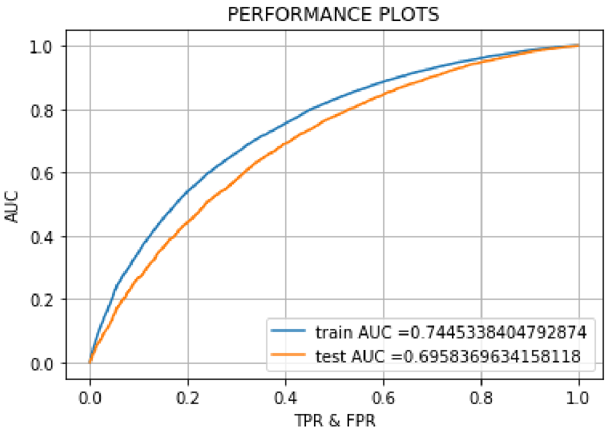
```
====================================================================================================
```



Considering the Hyperparameter max_depth=5 & n_estimators=10

In [73]:

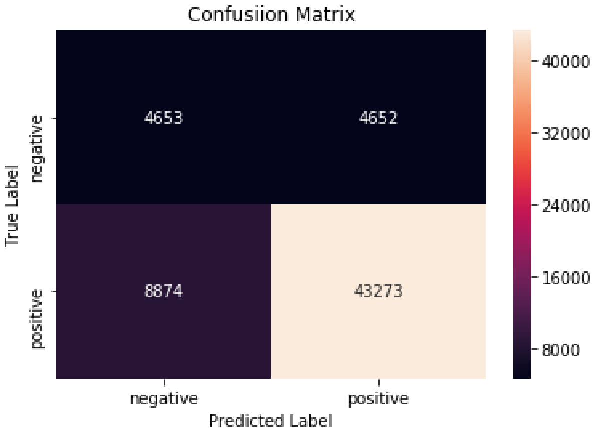
```
from sklearn.metrics import roc_curve, auc


clf= GradientBoostingClassifier(max_depth=5,n_estimators=10)

clf.fit(X_tr_w2v_tfidf, y_train)


y_train_pred =batch_predict(clf,X_tr_w2v_tfidf)
y_test_pred = batch_predict(clf,X_te_w2v_tfidf)

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("TPR & FPR")
plt.ylabel("AUC")
plt.title("PERFORMANCE PLOTS")
plt.grid()
plt.show()
```



In [74]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
cmtr=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
import seaborn as sns
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmtr, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

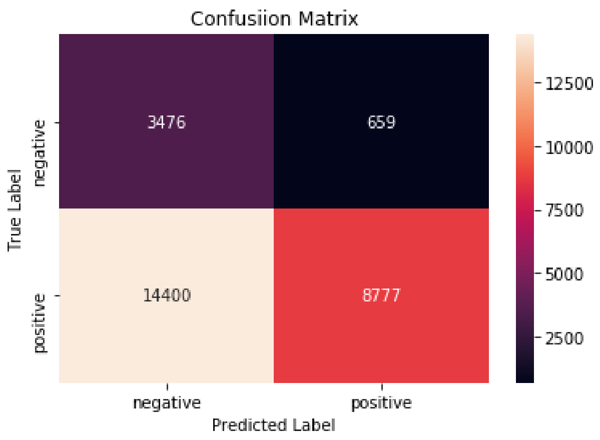
```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249999997112598 for threshold 0.814
```



In [75]:

```
print("Test confusion matrix")

cmt=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
class_label = ["negative", "positive"]
df_cm = pd.DataFrame(cmt, index = class_label, columns = class_label)
sns.heatmap(df_cm, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999998537859927 for threshold 0.879
```



3. Conclusions

In [68]:

```
#https://www.kaggle.com/premvardhan/amazon-fine-food-reviews-analysis-using-knn
models = pd.DataFrame({'Model': ['RF with Bow', "RF with TFIDF", "RF with Avg_w2v", "RF with
tfidf_w2v",'GBDT with Bow', "GBDT with TFIDF", "GBDT with Avg_w2v", "GBDT with tfidf_w2v"],
'Deapth':[10,5,10,5,5,5,5,5], 'n_estimators':[5,10,50,10,50,10,10,10],'Train AUC':[.79,.72,.79,.72,
.73,.74,.71,.74] , 'Test AUC': [.68,.69,.67,.68,.69,.70,.68,.71]}, columns = ["Model",
"Deapth","n_estimators", "Train AUC", "Test AUC"])
models#.sort_values(by='Test AUC', ascending=False)
```

Out[68]:

	Model	Deapth	n_estimators	Train AUC	Test AUC
0	RF with Bow	10	5	0.79	0.68
1	RF with TFIDF	5	10	0.72	0.69
2	RF with Avg_w2v	10	50	0.79	0.67
3	RF with tfidf_w2v	5	10	0.72	0.68
4	GBDT with Bow	5	50	0.73	0.69
5	GBDT with TFIDF	5	10	0.74	0.70
6	GBDT with Avg_w2v	5	10	0.71	0.68
7	GBDT with tfidf_w2v	5	10	0.74	0.71

Steps Taken

Step 1- We first Read the data from the file.
Step 2- Preprocessing all the data so that we can consider only information which has a value.
Step 3- Standardaried all the numerical data i.e Price,Quantity & Teacher number of previously posted projects.
Step 4- Response encoding of all the categorical data.
Step 5- Combining the data i.e Categorical and numerical.
Step 6- Converting the Text Data into BOW.
Step 7- Applying Random forest and GBDT to the data set
Step 8- Calculating the best hyperparameter
Step 9- Calculating the AUC-ROC value and plotting the heatmap
Step 10- Converting the Text Data into tfidf
Step 11- Applying Random forest and GBDT to the data set
Step 12- Calculating the best hyperparameter
Step 13- Calculating the AUC-ROC value and plotting the heatmap
Step 14- Converting the text data into word2vec
Step 15- Applying Random forest and GBDT to the data set

Step 16- Calculating the best hyperparameter
Step 17- Calculating the AUC-ROC value and plotting the heatmap
Step 18-Converting the text data into TFIDF word2vec
Step 19- Applying Random forest and GBDT to the data set
Step 20- Calculating the best hyperparameter
Step 21- Calculating the AUC-ROC value and plotting the heatmap
Step 22- Plotting Conclusions

In [ ]: