DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy`

Feature	Description
	• `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_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."

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

1.1 Reading Data

In [2]:

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

In [3]:

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

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

In [4]:

```
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]


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


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


project_data.head(2)
```

Out[4]:

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

In [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]))
```

1.3 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_cate
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5

In [10]:

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

In [11]:

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

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

int student work that is completed on the classroom Chromebooks.I want to try and remove all barri
ers for the students learning and create opportunities for learning. One of the biggest barriers i
s the students not having the resources to get pens, paper, and folders. My students will be able
to increase their literacy skills because of this project.
==========
\"Life moves pretty fast. If you don't stop and look around once in awhile, you could miss it.\"
from the movie, Ferris Bueller's Day Off. Think back...what do you remember about your
grandparents? How amazing would it be to be able to flip through a book to see a day in their
lives?My second graders are voracious readers! They love to read both fiction and nonfiction books
. Their favorite characters include Pete the Cat, Fly Guy, Piggie and Elephant, and Mercy Watson.
They also love to read about insects, space and plants. My students are hungry bookworms! My stude
nts are eager to learn and read about the world around them. My kids love to be at school and are
like little sponges absorbing everything around them. Their parents work long hours and usually do
not see their children. My students are usually cared for by their grandparents or a family
friend. Most of my students do not have someone who speaks English at home. Thus it is difficult f
or my students to acquire language.Now think forward... wouldn't it mean a lot to your kids,
nieces or nephews or grandchildren, to be able to see a day in your life today 30 years from now?
Memories are so precious to us and being able to share these memories with future generations will
be a rewarding experience. As part of our social studies curriculum, students will be learning ab
out changes over time. Students will be studying photos to learn about how their community has ch
anged over time. In particular, we will look at photos to study how the land, buildings,
clothing, and schools have changed over time. As a culminating activity, my students will capture
a slice of their history and preserve it through scrap booking. Key important events in their
young lives will be documented with the date, location, and names. Students will be using photos
from home and from school to create their second grade memories. Their scrap books will preserve
their unique stories for future generations to enjoy.Your donation to this project will provide my
second graders with an opportunity to learn about social studies in a fun and creative manner. Th
rough their scrapbooks, children will share their story with others and have a historical document
for the rest of their lives.
==========
\"A person's a person, no matter how small.\" (Dr.Seuss) I teach the smallest students with the bi
ggest enthusiasm for learning. My students learn in many different ways using all of our senses an
d multiple intelligences. I use a wide range of techniques to help all my students succeed. \r\nSt
udents in my class come from a variety of different backgrounds which makes for wonderful sharing
of experiences and cultures, including Native Americans.\r\nOur school is a caring community of su
ccessful learners which can be seen through collaborative student project based learning in and ou
t of the classroom. Kindergarteners in my class love to work with hands-on materials and have many
different opportunities to practice a skill before it is mastered. Having the social skills to wor
k cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is the
perfect place to learn about agriculture and nutrition. My students love to role play in our
pretend kitchen in the early childhood classroom. I have had several kids ask me, \"Can we try coo
king with REAL food?\" I will take their idea and create \"Common Core Cooking Lessons\" where we
learn important math and writing concepts while cooking delicious healthy food for snack time. My
students will have a grounded appreciation for the work that went into making the food and knowled
ge of where the ingredients came from as well as how it's healthy for their bodies. This project w
ould expand our learning of nutrition and agricultural cooking recipes by having us peel our own a
pples to make homemade applesauce, make our own bread, and mix up healthy plants from our classroo
m garden in the spring. We will also create our own cookbooks to be printed and shared with famili
es. \r\nStudents will gain math and literature skills as well as a life long enjoyment for healthy
cooking.nannan
==========
My classroom consists of twenty-two amazing sixth graders from different cultures and backgrounds.
They are a social bunch who enjoy working in partners and working with groups. They are hard-worki
ng and eager to head to middle school next year. My job is to get them ready to make this
transition and make it as smooth as possible. In order to do this, my students need to come to
school every day and feel safe and ready to learn. Because they are getting ready to head to
middle school, I give them lots of choice- choice on where to sit and work, the order to complete
assignments, choice of projects, etc. Part of the students feeling safe is the ability for them to
come into a welcoming, encouraging environment. My room is colorful and the atmosphere is casual.
I want them to take ownership of the classroom because we ALL share it together. Because my time w
ith them is limited, I want to ensure they get the most of this time and enjoy it to the best of t
heir abilities.Currently, we have twenty-two desks of differing sizes, yet the desks are similar t
o the ones the students will use in middle school. We also have a kidney table with crates for sea
ting. I allow my students to choose their own spots while they are working independently or in
groups. More often than not, most of them move out of their desks and onto the crates. Believe it
or not, this has proven to be more successful than making them stay at their desks! It is because
of this that I am looking toward the “Flexible Seating” option for my classroom.\r\n The students
look forward to their work time so they can move around the room. I would like to get rid of the c
onstricting desks and move toward more “fun” seating options. I am requesting various seating so m
y students have more options to sit. Currently, I have a stool and a papasan chair I inherited fro
m the previous sixth-grade teacher as well as five milk crate seats I made, but I would like to gi
ve them more options and reduce the competition for the “good seats”. I am also requesting two rug
s as not only more seating options but to make the classroom more welcoming and appealing. In orde
r for my students to be able to write and complete work without desks, I am requesting a class set
of clipboards. Finally, due to curriculum that requires groups to work together, I am requesting t
ables that we can fold up when we are not using them to leave more room for our flexible seating o
ptions.\r\nI know that with more seating options, they will be that much more excited about coming

to school! Thank you for your support in making my classroom one students will remember forever!nannan
==

In [12]:

```
# 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 [13]:

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

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

In [14]:

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

 A person is a person, no matter how small. (Dr.Seuss) I teach the smallest students with the big
gest enthusiasm for learning. My students learn in many different ways using all of our senses and
multiple intelligences. I use a wide range of techniques to help all my students succeed.
Students in my class come from a variety of different backgrounds which makes for wonderful
sharing of experiences and cultures, including Native Americans. Our school is a caring community
of successful learners which can be seen through collaborative student project based learning in a
nd out of the classroom. Kindergarteners in my class love to work with hands-on materials and have
many different opportunities to practice a skill before it is mastered. Having the social skills t
o work cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is
the perfect place to learn about agriculture and nutrition. My students love to role play in our p
retend kitchen in the early childhood classroom. I have had several kids ask me, Can we try cooki
ng with REAL food? I will take their idea and create Common Core Cooking Lessons where we learn

ng with REAL food. I will take their idea and create "Common Core Cooking Lessons" where we learn important math and writing concepts while cooking delicious healthy food for snack time. My students will have a grounded appreciation for the work that went into making the food and knowled ge of where the ingredients came from as well as how it is healthy for their bodies. This project would expand our learning of nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classro om garden in the spring. We will also create our own cookbooks to be printed and shared with famil ies. Students will gain math and literature skills as well as a life long enjoyment for healthy cooking.nannan

In [15]:

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

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

In [16]:

```
# 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', "d
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 [17]:

```
# Combining all the above stundents
from tqdm import tqdm
```

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

```
100%|██████████████████████████████████████████████████████████████████| 109248/109248
[00:59<00:00, 1838.42it/s]
```

In [18]:

```
# after preprocesing
project_data['preprocessed_essays']=preprocessed_essays
preprocessed_essays[20000]
```

Out[18]:

'a person person no matter small dr seuss i teach smallest students biggest enthusiasm learning my students learn many different ways using senses multiple intelligences i use wide range techniques help students succeed students class come variety different backgrounds makes wonderful sharing ex periences cultures including native americans our school caring community successful learners seen collaborative student project based learning classroom kindergarteners class love work hands materials many different opportunities practice skill mastered having social skills work cooperatively friends crucial aspect kindergarten curriculum montana perfect place learn agriculture nutrition my students love role play pretend kitchen early childhood classroom i several kids ask can try cooking real food i take idea create common core cooking lessons learn im portant math writing concepts cooking delicious healthy food snack time my students grounded appre ciation work went making food knowledge ingredients came well healthy bodies this project would ex pand learning nutrition agricultural cooking recipes us peel apples make homemade applesauce make bread mix healthy plants classroom garden spring we also create cookbooks printed shared families students gain math literature skills well life long enjoyment healthy cooking nannan'

1.4 Preprocessing of `project_title`

In [50]:

```
preprocessed_titles = []
for sentence in project_data['project_title'].values:
    snt= decontracted(sentence)
    snt= snt.replace('\\r', ' ')
    snt= snt.replace('\\"', ' ')
    snt= snt.replace('\\n', ' ')
    snt= re.sub('[^A-Za-z0-9]+', ' ', snt)

    # https://gist.github.com/sebleier/554280
    snt = ' '.join(e for e in snt.split() if e not in stopwords)
    preprocessed_titles.append(snt.lower().strip())

project_data['preprocessed_titles'] = preprocessed_titles
preprocessed_titles[1000]
```

Out[50]:

'empowering students through art learning about then now'

1.5 Preparing data for models

In [19]:

```
project_data.columns
```

Out[19]:

```
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'],
      dtype='object')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

In [0]:

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

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

In [0]:

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (109248, 30)
```

In [0]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [0]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_bow.shape)
```

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

In [0]:

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

1.5.2.2 TFIDF vectorizer

In [0]:

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

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

1.5.2.3 Using Pretrained Models: Avg W2V

In [0]:

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

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

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

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

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

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

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

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


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

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


'''
```

Out[0]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
============================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus",
len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\n
print("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [85]:

```
# 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 [0]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors.append(vector)

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

```
100%|██████████████████████████████████████████████████████████████████████████| 109248/109248
[00:32<00:00, 3369.33it/s]
```

```
109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [0]:

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

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/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.append(vector)

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

```
100%|██████████████████████████████████████████████████████████████████████████████| 109248/109248
[03:36<00:00, 503.77it/s]
```

```
109248
300
```

In [0]:

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

1.5.3 Vectorizing Numerical features

1.5.4 Merging all the above features

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

Computing Sentiment Scores

In [0]:

```
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\
```

```
t backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a carin
g community of successful \
learners which can be seen through collaborative student project based learning in and out of the
classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities to practice
a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect of the ki
ndergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role pla
y in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with real food
i will take their idea \
and create common core cooking lessons where we learn important math and writing concepts while co
oking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into maki
ng the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project woul
d expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade apple
sauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cook
books to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoymen
t for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

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

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

```
D:\installed\Anaconda3\lib\site-packages\nltk\twitter\__init__.py:20: UserWarning:

The twython library has not been installed. Some functionality from the twitter package will not b
e available.
```

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

Assignment 7: SVM

1. **[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) on these feature sets**
    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
    - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
    - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**
    - Find the best hyper parameter which will give the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning
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.
    - 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. Please visualize your confusion matrices using seaborn heatmaps.
4. **[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3**

- Consider these set of features Set 5 :
  - **school_state** : categorical data
  - **clean_categories** : categorical data
  - **clean_subcategories** : categorical data
  - **project_grade_category** :categorical data
  - **teacher_prefix** : categorical data
  - **quantity** : numerical data
  - **teacher_number_of_previously_posted_projects** : numerical data
  - **price** : numerical data
  - **sentiment score's of each of the essay** : numerical data
  - **number of words in the title** : numerical data
  - **number of words in the combine essays** : numerical data
  - **Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`) using elbow method** : numerical data

- **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-leakage, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

2. Support Vector Machines

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

In [0]:

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

In [20]:

```
project_data['teacher_prefix'].fillna(" ", inplace = True)
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 [51]:

```
from scipy import sparse
from scipy.sparse import csr_matrix
from scipy.sparse import lil_matrix

Y = project_data['project_is_approved'].values
X = project_data

Y = Y[0:60000]
X = X.head(60000)
```

In [52]:

```
teacher_prefix=[]
for s1 in X['teacher_prefix']:
    s1= s1.replace('.', '')
    teacher_prefix.append(s1.lower().strip())

X['teacher_prefix'] = teacher_prefix
X['teacher_prefix'].fillna(" ", inplace = True)
from collections import Counter
my_counter = Counter()
for word in X['teacher_prefix'].values:
    my_counter.update(word.split())
teacher_prefix = dict(my_counter)
teacher_prefix = dict(sorted(teacher_prefix.items(), key=lambda kv: kv[1]))
```

In [53]:

```
project_grade=[]
for s1 in X['project_grade_category']:
    s1= s1.replace('Grades', '')
    s1= s1.replace('-', '_')
    project_grade.append(s1.lower().strip())

X['project_grade_category'] = project_grade

from collections import Counter
my_counter = Counter()
for word in X['project_grade_category'].values:
    my_counter.update(word.split())
project_grade_category = dict(my_counter)
project_grade_category = dict(sorted(project_grade_category.items(), key=lambda kv: kv[1]))
```

In [54]:

```
l =[]
for i in X['essay']:
    words = len(i.split())
    l.append(words)
X['essay_len'] = l
```

In [55]:

```
l =[]
for i in X['project_title']:
    words = len(i.split())
    l.append(words)
X['title_len'] = l
```

In [56]:

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

import nltk
nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = X['preprocessed_essays']
ss =[]
for i in range(0,len(for_sentiment)):
    j= sid.polarity_scores(for_sentiment[i])
    ss.append(j)

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

neg = []
neu=[]
pos=[]
compound=[]
for k in range(0,len(ss)):
    neg.append(ss[k]['neg'])
```

```
    neu.append(ss[k]['neu'])
    pos.append(ss[k]['pos'])
    compound.append(ss[k]['compound'])

X['neg']= neg
X['neu']= neu
X['pos']= pos
X['compound']=compound

X.head()
```

```
[nltk_data] Downloading package vader_lexicon to C:\Users\Asus
[nltk_data]     PC\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[56]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	mrs	CA	2016-04-27 00:27:36	prek_2
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	ms	UT	2016-04-27 00:31:25	3_5
2	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	mrs	CA	2016-04-27 00:46:53	prek_2
3	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	mrs	GA	2016-04-27 00:53:00	prek_2
4	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	mrs	WA	2016-04-27 01:05:25	3_5

5 rows × 28 columns

In [57]:

```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle = False)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, shuffle = False)


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

```
(26934, 28) (26934,)
(13266, 28) (13266,)
(19800, 28) (19800,)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [0]:

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

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

In [58]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
vectorizer.fit_transform(X_train['clean_categories'].values)

X_train_cat= vectorizer.transform(X_train['clean_categories'].values)
X_test_cat= vectorizer.transform(X_test['clean_categories'].values)
X_cv_cat= vectorizer.transform(X_cv['clean_categories'].values)

print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding:")
print(X_train_cat.shape)
print(X_test_cat.shape)
print(X_cv_cat.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encoding:
(26934, 9)
(19800, 9)
(13266, 9)
```

In [59]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit_transform(project_data['clean_subcategories'].values)

X_train_subcat= vectorizer.transform(X_train['clean_subcategories'].values)
X_test_subcat= vectorizer.transform(X_test['clean_subcategories'].values)
X_cv_subcat= vectorizer.transform(X_cv['clean_subcategories'].values)

print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding:")
print(X_train_subcat.shape)
print(X_test_subcat.shape)
print(X_cv_subcat.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encoding:
(26934, 30)
(19800, 30)
(13266, 30)
```

In [60]:

```
from collections import Counter
my_counter = Counter()
for word in project_data['school_state'].values:
    my_counter.update(word.split())
school_state = dict(my_counter)
school_state = dict(sorted(school_state.items(), key=lambda kv: kv[1]))
```

```
vectorizer = CountVectorizer(vocabulary=list(school_state.keys()), lowercase=False, binary=True)
vectorizer.fit((project_data['school_state']).values)
print(vectorizer.get_feature_names())
X_train_school= vectorizer.transform(X_train['school_state'].values)
X_test_school= vectorizer.transform(X_test['school_state'].values)
X_cv_school= vectorizer.transform(X_cv['school_state'].values)


print("Shape of matrix after one hot encoding: ")
print(X_train_school.shape)
print(X_test_school.shape)
print(X_cv_school.shape)
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'WV', 'ME', 'HI', 'DC', 'NM', 'KS', 'I
A', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'NV', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ',
'NJ', 'OK', 'WA', 'MA', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX
', 'CA']
Shape of matrix after one hot encoding:
(26934, 51)
(19800, 51)
(13266, 51)
```

In [61]:

```
vectorizer = CountVectorizer(vocabulary= list(teacher_prefix.keys()), lowercase=False, binary=True
)
vectorizer.fit(X_train['teacher_prefix'])
print(vectorizer.get_feature_names())

X_train_teacher= vectorizer.transform(X_train['teacher_prefix'].values)
X_test_teacher= vectorizer.transform(X_test['teacher_prefix'].values)
X_cv_teacher= vectorizer.transform(X_cv['teacher_prefix'].values)

print("Shape of matrix after one hot encoding ")

print(X_train_teacher.shape)
print(X_test_teacher.shape)
print(X_cv_teacher.shape)

print(X_train_teacher[:5])
```

```
['dr', 'teacher', 'mr', 'ms', 'mrs']
Shape of matrix after one hot encoding
(26934, 5)
(19800, 5)
(13266, 5)
  (0, 4) 1
  (1, 3) 1
  (2, 4) 1
  (3, 4) 1
  (4, 4) 1
```

In [62]:

```
vectorizer = CountVectorizer(vocabulary=list(project_grade_category.keys()), lowercase=False, bina
ry=True)
vectorizer.fit(X_train['project_grade_category'])
print(vectorizer.get_feature_names())

X_train_grade= vectorizer.transform(X_train['project_grade_category'].values)
X_test_grade= vectorizer.transform(X_test['project_grade_category'].values)
X_cv_grade= vectorizer.transform(X_cv['project_grade_category'].values)

print("Shape of matrix after one hot encoding ")

print(X_train_grade.shape)
print(X_test_grade.shape)
print(X_cv_grade.shape)

print(X_train_grade[:5])
```

```
['9_12', '6_8', '3_5', 'prek_2']
```

```
Shape of matrix after one hot encoding
(26934, 4)
(19800, 4)
(13266, 4)
  (0, 3) 1
  (1, 2) 1
  (2, 3) 1
  (3, 3) 1
  (4, 2) 1
```

In [63]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['price'].values.reshape(1,-1))
X_train_price = normalizer.transform(X_train['price'].values.reshape(1,-1))
X_cv_price = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_price = normalizer.transform(X_test['price'].values.reshape(1,-1))

print("After vectorizations:")
print(X_train_price.shape, y_train.shape)
print(X_cv_price.shape, y_cv.shape)
print(X_test_price.shape, y_test.shape)
print("="*50)
```

```
After vectorizations:
(1, 26934) (26934,)
(1, 13266) (13266,)
(1, 19800) (19800,)
==================================================
```

In [64]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['quantity'].values.reshape(1,-1))
X_train_quantity = normalizer.transform(X_train['quantity'].values.reshape(1,-1))
X_cv_quantity = normalizer.transform(X_cv['quantity'].values.reshape(1,-1))
X_test_quantity = normalizer.transform(X_test['quantity'].values.reshape(1,-1))

print("After vectorizations:")
print(X_train_quantity.shape, y_train.shape)
print(X_cv_quantity.shape, y_cv.shape)
print(X_test_quantity.shape, y_test.shape)
print("="*50)
```

```
After vectorizations:
(1, 26934) (26934,)
(1, 13266) (13266,)
(1, 19800) (19800,)
==================================================
```

In [65]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_train_previous = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].va
lues.reshape(1,-1))
X_cv_previous = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.r
eshape(1,-1))
X_test_previous =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

print("After vectorizations:")
print(X_train_previous.shape, y_train.shape)
print(X_cv_previous.shape, y_cv.shape)
print(X_test_previous.shape, y_test.shape)
print("="*50)
```

```
After vectorizations:
(1, 26934) (26934,)
(1, 13266) (13266,)
```

```
(1, 19800) (19800,)
==================================================
```

In [66]:

```
X_train_num_bow = np.hstack((X_train_price.reshape(-1,1),X_train_quantity.reshape(-1,1),X_train_pre
vious.reshape(-1,1)))
X_test_num_bow = np.hstack((X_test_price.reshape(-1,1),X_test_quantity.reshape(-1,1),X_test_previou
s.reshape(-1,1)))
X_cv_num_bow = np.hstack((X_cv_price.reshape(-1,1),X_cv_quantity.reshape(-1,1),X_cv_previous.reshap
e(-1,1)))
```

In [67]:

```
from scipy.sparse import hstack

X_train_features = hstack((X_train_cat, X_train_subcat,X_train_school,X_train_teacher,X_train_grade
,X_train_num_bow)).tocsr()
X_test_features = hstack((X_test_cat,
X_test_subcat,X_test_school,X_test_teacher,X_test_grade,X_test_num_bow)).tocsr()
X_cv_features = hstack((X_cv_cat,
X_cv_subcat,X_cv_school,X_cv_teacher,X_cv_grade,X_cv_num_bow)).tocsr()

#X_features= X_features.tocsr()[0:10000,]


print("Final Data matrix")
print(X_train_features.shape, y_train.shape)
print(X_cv_features.shape, y_cv.shape)
print(X_test_features.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(26934, 102) (26934,)
(13266, 102) (13266,)
(19800, 102) (19800,)
====================================================================================================
```

In [68]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['essay_len'].values.reshape(1,-1))
X_train_essay_len = normalizer.transform(X_train['essay_len'].values.reshape(1,-1))
X_cv_essay_len = normalizer.transform(X_cv['essay_len'].values.reshape(1,-1))
X_test_essay_len = normalizer.transform(X_test['essay_len'].values.reshape(1,-1))

print("After vectorizations:")
print(X_train_essay_len.shape, y_train.shape)
print(X_cv_essay_len.shape, y_cv.shape)
print(X_test_essay_len.shape, y_test.shape)
print("="*50)
```

```
After vectorizations:
(1, 26934) (26934,)
(1, 13266) (13266,)
(1, 19800) (19800,)
==================================================
```

In [69]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['title_len'].values.reshape(1,-1))
X_train_title_len = normalizer.transform(X_train['title_len'].values.reshape(1,-1))
X_cv_title_len = normalizer.transform(X_cv['title_len'].values.reshape(1,-1))
X_test_title_len = normalizer.transform(X_test['title_len'].values.reshape(1,-1))

print("After vectorizations:")
print(X_train_title_len.shape, y_train.shape)
print(X_cv_title_len.shape, y_cv.shape)
```

```
print(X_test_title_len.shape, y_test.shape)
print("="*50)
```

```
After vectorizations:
(1, 26934) (26934,)
(1, 13266) (13266,)
(1, 19800) (19800,)
==================================================
```

In [70]:

```
X_train_features_set5 = hstack((X_train_features,X_train['neg'].values.reshape(-1,1),X_train['neu']
.values.reshape(-1,1),X_train['pos'].values.reshape(-1,1),X_train['compound'].values.reshape(-1,1),
X_train_essay_len.reshape(-1,1),X_train_title_len.reshape(-1,1)))
X_test_features_set5 = hstack((X_test_features,X_test['neg'].values.reshape(-1,1),X_test['neu'].val
ues.reshape(-1,1),X_test['pos'].values.reshape(-1,1),X_test['compound'].values.reshape(-1,1),X_test
_essay_len.reshape(-1,1),X_test_title_len.reshape(-1,1)))
X_cv_features_set5 = hstack((X_cv_features,X_cv['neg'].values.reshape(-1,1),X_cv['neu'].values.resh
ape(-1,1),X_cv['pos'].values.reshape(-1,1),X_cv['compound'].values.reshape(-1,1),X_cv_essay_len.res
hape(-1,1),X_cv_title_len.reshape(-1,1)))

print("Final Data matrix")
print(X_train_features_set5.shape, y_train.shape)
print(X_cv_features_set5.shape, y_cv.shape)
print(X_test_features_set5.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(26934, 108) (26934,)
(13266, 108) (13266,)
(19800, 108) (19800,)
====================================================================================================
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [71]:

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

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

X_train_text = X_train['preprocessed_essays']
X_cv_text = X_cv['preprocessed_essays']
X_test_text = X_test['preprocessed_essays']
```

In [72]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df =10)
vectorizer.fit(X_train_text) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.fit_transform(X_train_text)
X_cv_bow = vectorizer.transform(X_cv_text)
X_test_bow = vectorizer.transform(X_test_text)

print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(26934, 9437) (26934,)
(13266, 9437) (13266,)
(19800, 9437) (19800,)
====================================================================================================
```

In [73]:

```
X_train_text = X_train['preprocessed_titles']
X_cv_text = X_cv['preprocessed_titles']
X_test_text = X_test['preprocessed_titles']
```

In [75]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df =10)
vectorizer.fit(X_train_text) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow2 = vectorizer.fit_transform(X_train_text)
X_cv_bow2 = vectorizer.transform(X_cv_text)
X_test_bow2 = vectorizer.transform(X_test_text)

print("After vectorizations")
print(X_train_bow2.shape, y_train.shape)
print(X_cv_bow2.shape, y_cv.shape)
print(X_test_bow2.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(26934, 1349) (26934,)
(13266, 1349) (13266,)
(19800, 1349) (19800,)
====================================================================================================
```

In [76]:

```
X_train_features_bow = hstack((X_train_features, X_train_bow, X_train_bow2))
X_cv_features_bow = hstack((X_cv_features, X_cv_bow, X_cv_bow2))
X_test_features_bow = hstack((X_test_features, X_test_bow, X_test_bow2))

print(X_train_features_bow.shape, y_train.shape)
print(X_cv_features_bow.shape, y_cv.shape)
print(X_test_features_bow.shape, y_test.shape)
print("="*100)
```

```
(26934, 10888) (26934,)
(13266, 10888) (13266,)
(19800, 10888) (19800,)
====================================================================================================
```

TFIDF

In [79]:

```
X_train_text = X_train['preprocessed_essays']
X_cv_text = X_cv['preprocessed_essays']
X_test_text = X_test['preprocessed_essays']

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train_text) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tfidf = vectorizer.transform(X_train_text)
X_cv_tfidf = vectorizer.transform(X_cv_text)
X_test_tfidf = vectorizer.transform(X_test_text)
```

```
print("After vectorizations")
print(X_train_tfidf.shape, y_train.shape)
print(X_cv_tfidf.shape, y_cv.shape)
print(X_test_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(26934, 9437) (26934,)
(13266, 9437) (13266,)
(19800, 9437) (19800,)
====================================================================================================
```

In [80]:

```
X_train_text = X_train['preprocessed_titles']
X_cv_text = X_cv['preprocessed_titles']
X_test_text = X_test['preprocessed_titles']

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train_text) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tfidf2 = vectorizer.transform(X_train_text)
X_cv_tfidf2 = vectorizer.transform(X_cv_text)
X_test_tfidf2 = vectorizer.transform(X_test_text)

print("After vectorizations")
print(X_train_tfidf2.shape, y_train.shape)
print(X_cv_tfidf2.shape, y_cv.shape)
print(X_test_tfidf2.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(26934, 1349) (26934,)
(13266, 1349) (13266,)
(19800, 1349) (19800,)
====================================================================================================
```

In [81]:

```
X_train_features_tfidf = hstack((X_train_features, X_train_tfidf, X_train_tfidf2))
X_cv_features_tfidf = hstack((X_cv_features, X_cv_tfidf, X_cv_tfidf2))
X_test_features_tfidf = hstack((X_test_features, X_test_tfidf, X_test_tfidf2))
```

AvgW2V

In [82]:

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
```

In [83]:

```
X_train_text = X_train['preprocessed_essays']
X_cv_text = X_cv['preprocessed_essays']
X_test_text = X_test['preprocessed_essays']

i=0
list_of_sentence_train=[]
for sentence in X_train_text:
    list_of_sentence_train.append(sentence.split())
```

In [86]:

```
# average Word2Vec
```

```
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sent in X_train_text: # 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
    sent_vectors_train.append(vector)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
```

```
(26934, 300)
```

In [87]:

```
i=0
list_of_sentence_cv=[]
for sentence in X_cv_text:
    list_of_sentence_cv.append(sentence.split())

sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sentence_cv: # 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
    sent_vectors_cv.append(vector)
sent_vectors_cv = np.array(sent_vectors_cv)
print(sent_vectors_cv.shape)
```

```
(13266, 300)
```

In [88]:

```
i=0
list_of_sentence_test=[]
for sentence in X_test_text:
    list_of_sentence_test.append(sentence.split())


sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sentence_test: # 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
    sent_vectors_test.append(vector)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
```

```
(19800, 300)
```

In [89]:

```
X_train_features_avgw2v = hstack((X_train_features, sent_vectors_train))
X_cv_features_avgw2v = hstack((X_cv_features, sent_vectors_cv))
X_test_features_avgw2v = hstack((X_test_features, sent_vectors_test))
```

In [90]:

```
X_train_text = X_train['preprocessed_titles']
X_cv_text = X_cv['preprocessed_titles']
X_test_text = X_test['preprocessed_titles']

i=0
list_of_sentence_train=[]
for sentence in X_train_text:
    list_of_sentence_train.append(sentence.split())

# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sent in X_train_text: # 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
    sent_vectors_train.append(vector)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
```

```
(26934, 300)
```

In [91]:

```
i=0
list_of_sentence_cv=[]
for sentence in X_cv_text:
    list_of_sentence_cv.append(sentence.split())

sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sentence_cv: # 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
    sent_vectors_cv.append(vector)
sent_vectors_cv = np.array(sent_vectors_cv)
print(sent_vectors_cv.shape)
```

```
(13266, 300)
```

In [92]:

```
i=0
list_of_sentence_test=[]
for sentence in X_test_text:
    list_of_sentence_test.append(sentence.split())


sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sentence_test: # 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
    sent_vectors_test.append(vector)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
```

```
(19800, 300)
```

(19000, 300)

In [93]:

```
X_train_features_avgw2v = hstack((X_train_features, sent_vectors_train))
X_cv_features_avgw2v = hstack((X_cv_features, sent_vectors_cv))
X_test_features_avgw2v = hstack((X_test_features, sent_vectors_test))
```

TFIDF Word2Vec

In [94]:

```
X_train_text = X_train['preprocessed_essays']
X_cv_text = X_cv['preprocessed_essays']
X_test_text = X_test['preprocessed_essays']
```

In [95]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train_text)
tfidf_model.transform(X_train_text)
# 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 = [];
for sentence in tqdm(X_train_text): # 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.append(vector)

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

X_train_features_tfidfw2v = hstack((X_train_features, tfidf_w2v_vectors))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 26934/26934 [00:
59<00:00, 452.07it/s]
```

```
26934
300
```

In [96]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train_text)
tfidf_model.transform(X_test_text)
# 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 = [];
for sentence in tqdm(X_test_text): # 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.append(vector)

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

X_test_features_tfidfw2v = hstack((X_test_features, tfidf_w2v_vectors))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 19800/19800 [00:
42<00:00, 462.15it/s]
```

```
19800
300
```

In [97]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train_text)
tfidf_model.transform(X_cv_text)
# 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 = [];
for sentence in tqdm(X_cv_text): # 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.append(vector)

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

X_cv_features_tfidfw2v = hstack((X_cv_features, tfidf_w2v_vectors))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 13266/13266 [00:
28<00:00, 463.30it/s]
```

```
13266
300
```

In [98]:

```
X_train_text = X_train['preprocessed_titles']
X_cv_text = X_cv['preprocessed_titles']
X_test_text = X_test['preprocessed_titles']
```

In [99]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train_text)
tfidf_model.transform(X_train_text)
# 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 = [];
for sentence in tqdm(X_train_text): # 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.append(vector)

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

X_train_features_tfidfw2v = hstack((X_train_features, tfidf_w2v_vectors))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 26934/26934
[00:01<00:00, 26844.89it/s]
```

```
26934
300
```

In [100]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train_text)
tfidf_model.transform(X_test_text)
# 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 = [];
for sentence in tqdm(X_test_text): # 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.append(vector)

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

X_test_features_tfidfw2v = hstack((X_test_features, tfidf_w2v_vectors))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 19800/19800
[00:00<00:00, 29764.64it/s]
```

```
19800
300
```

In [102]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
```

```
tfidf_model.fit(X_train_text)
tfidf_model.transform(X_cv_text)
# 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 = [];
for sentence in tqdm(X_cv_text): # 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.append(vector)

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

X_cv_features_tfidfw2v = hstack((X_cv_features, tfidf_w2v_vectors))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 13266/13266
[00:00<00:00, 31150.95it/s]
```

```
13266
300
```

All the features have been encoded and vectorized.

2.4 Appling Support Vector Machines on different kind of featurization as mentioned in the instructions

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

In [0]:

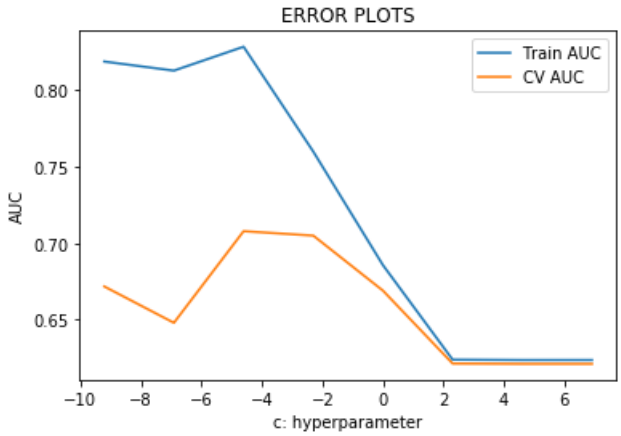
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging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

In [162]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

svm = SGDClassifier(loss='hinge', class_weight = 'balanced')
c = [0.0001,0.001,0.01,0.1,1,10,100,1000]
penalty = ['l1', 'l2']
parameters = {'alpha': c, 'penalty':penalty}

clf = GridSearchCV(svm, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_features_bow, y_train)

best_penalty = clf.best_estimator_.get_params()['penalty']
```

```
best_penalty = clf.best_estimator_.get_params()['penalty']
print('Best Penalty:', best_penalty)

train_auc = []
cv_auc = []
for i in c:
    svm = SGDClassifier(loss='hinge',alpha=i, penalty= best_penalty, class_weight = 'balanced')
    svm.fit(X_train_features_bow, y_train)
    y_train_pred_bow =  svm.decision_function(X_train_features_bow)
    y_cv_pred_bow =  svm.decision_function(X_cv_features_bow)

    train_auc.append(roc_auc_score(y_train,y_train_pred_bow))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred_bow))

plt.plot(np.log(c), train_auc, label='Train AUC')
plt.plot(np.log(c), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

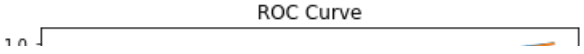
```
Best Penalty: l2
```

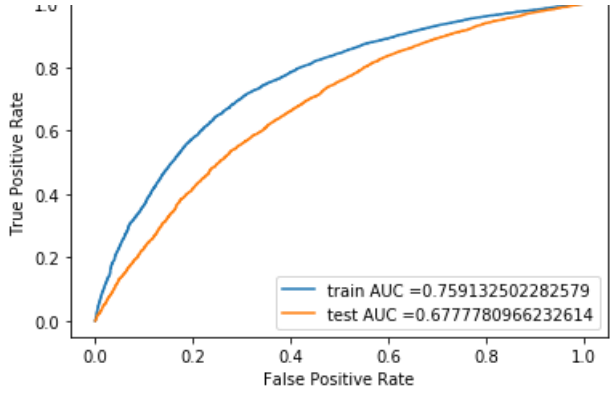


In [163]:

```
best_alpha = clf.best_estimator_.get_params()['alpha']
print("best alpha: ", best_alpha)

from sklearn.metrics import roc_curve, auc


svm = SGDClassifier(loss='hinge',alpha=best_alpha, penalty = best_penalty, class_weight = 'balanced
')
svm.fit(X_train_features_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred_bow =  svm.decision_function(X_train_features_bow)
y_test_pred_bow = svm.decision_function(X_test_features_bow)
train_fpr, train_tpr, thresholds = roc_curve(y_train, svm.decision_function(X_train_features_bow))
test_fpr, test_tpr, thresholds = roc_curve(y_test, svm.decision_function(X_test_features_bow))

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()

print("="*100)
```

```
best alpha:  0.1
```





==========

In [164]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

Confusion matrix for BOW

In [165]:

```
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
```

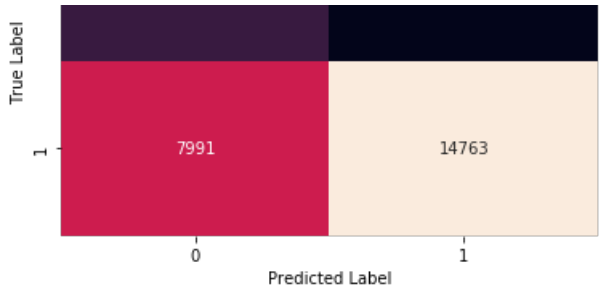
In [166]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds, train_fpr, train_tpr)

print("Train confusion matrix")
matrix_train= confusion_matrix(y_train, predict_with_best_t(y_train_pred_bow, best_t))
print(matrix_train)
sns.heatmap(matrix_train,annot=True,cbar=False,fmt='d')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.4935028406593846 for threshold -0.009
Train confusion matrix
[[ 3127  1053]
 [ 7991 14763]]
```

Out[166]:

```
Text(0.5,1,'Train Confusion Matrix')
```



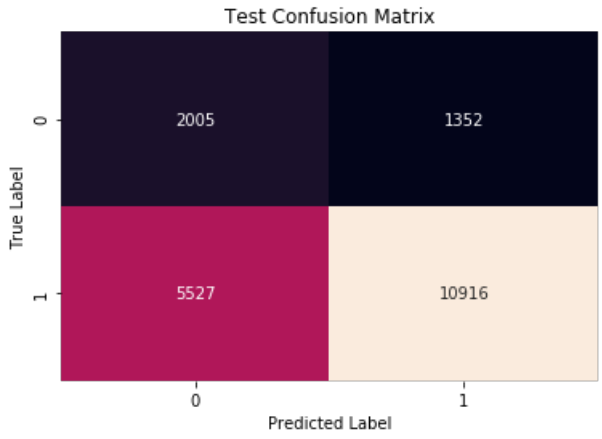
In [167]:

```
print("Test confusion matrix")
matrix_test= confusion_matrix(y_test, predict_with_best_t(y_test_pred_bow, best_t))
print(matrix_test)
sns.heatmap(matrix_test,annot=True,cbar=False,fmt='d')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Test Confusion Matrix')
```

```
Test confusion matrix
[[ 2005  1352]
 [ 5527 10916]]
```

Out[167]:

```
Text(0.5,1,'Test Confusion Matrix')
```



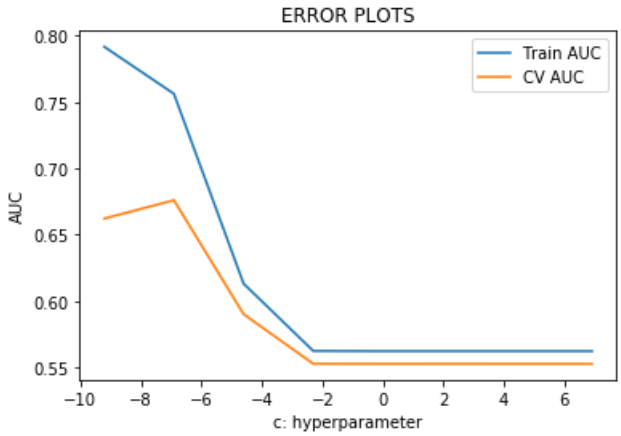
TFIDF

In [169]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

c = [0.0001,0.001,0.01,0.1,1,10,100,1000]
penalty = ['l1', 'l2']
parameters = {'alpha': c, 'penalty':penalty}
svm = SGDClassifier(loss='hinge',class_weight = 'balanced')
clf = GridSearchCV(svm, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_features_tfidf, y_train)
best_penalty = clf.best_estimator_.get_params()['penalty']
print('Best Penalty:', best_penalty)

train_auc = []
cv_auc = []
for i in c:
    svm = SGDClassifier(loss='hinge',alpha=i, penalty = best_penalty,class_weight = 'balanced')
    svm.fit(X_train_features_tfidf, y_train)
    y_train_pred_tfidf =  svm.decision_function(X_train_features_tfidf)
```

```
    y_cv_pred_tfidf =  svm.decision_function(X_cv_features_tfidf)

    train_auc.append(roc_auc_score(y_train,y_train_pred_tfidf))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred_tfidf))

plt.plot(np.log(c), train_auc, label='Train AUC')
plt.plot(np.log(c), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

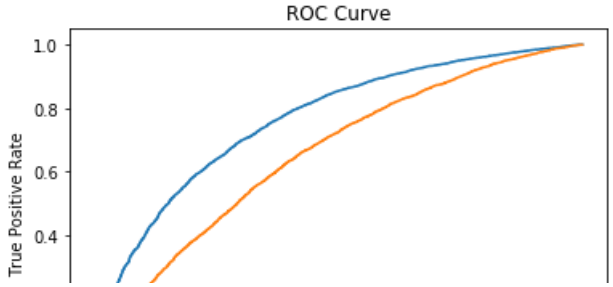
```
Best Penalty: l2
```

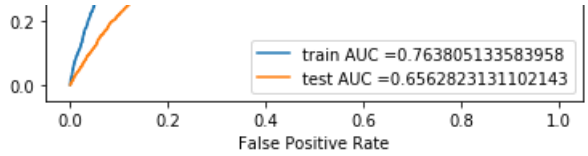


In [170]:

```
best_alpha = clf.best_estimator_.get_params()['alpha']
print("best alpha: ", best_alpha)
from sklearn.metrics import roc_curve, auc


svm = SGDClassifier(loss='hinge',alpha=best_alpha, penalty = best_penalty,class_weight = 'balanced'
)
svm.fit(X_train_features_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred_tfidf =  svm.decision_function(X_train_features_tfidf)
y_test_pred_tfidf = svm.decision_function(X_test_features_tfidf)
train_fpr, train_tpr, thresholds = roc_curve(y_train, svm.decision_function(X_train_features_tfidf
))
test_fpr, test_tpr, thresholds = roc_curve(y_test, svm.decision_function(X_test_features_tfidf))

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()

print("="*100)
```

```
best alpha:  0.001
```





==========

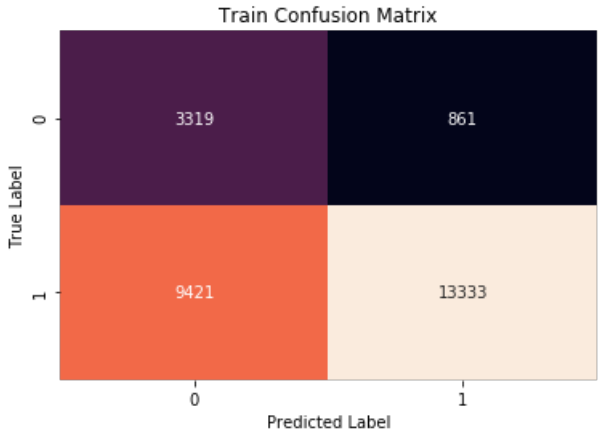
In [171]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds, train_fpr, train_tpr)

print("Train confusion matrix")
matrix_train= confusion_matrix(y_train, predict_with_best_t(y_train_pred_tfidf, best_t))
print(matrix_train)
sns.heatmap(matrix_train,annot=True,cbar=False,fmt='d')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.4881247863039382 for threshold -0.08
Train confusion matrix
[[ 3319   861]
 [ 9421 13333]]
```

Out[171]:

```
Text(0.5,1,'Train Confusion Matrix')
```



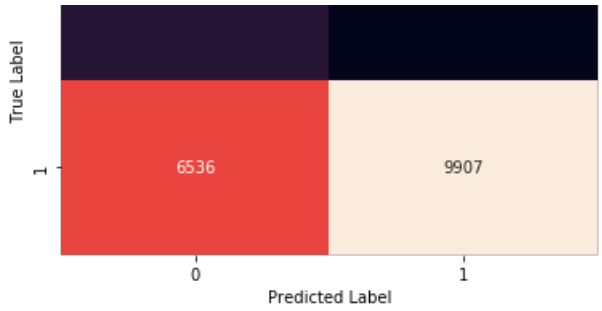
In [172]:

```
print("Test confusion matrix")
matrix_test= confusion_matrix(y_test, predict_with_best_t(y_test_pred_tfidf, best_t))
print(matrix_test)
sns.heatmap(matrix_test,annot=True,cbar=False,fmt='d')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Test Confusion Matrix')
```

```
Test confusion matrix
[[2102 1255]
 [6536 9907]]
```

Out[172]:

```
Text(0.5,1,'Test Confusion Matrix')
```

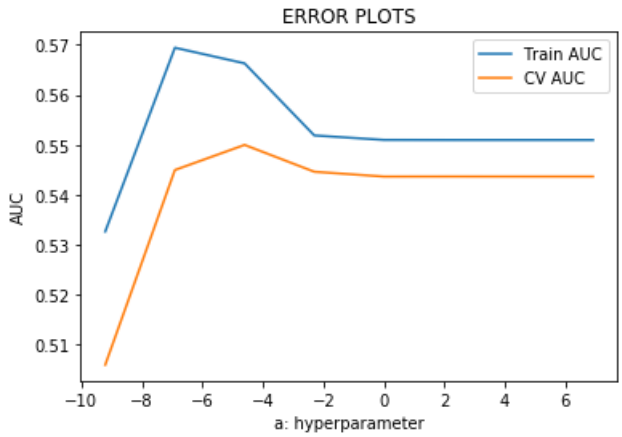


AVGw2v

In [173]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

c = [0.0001,0.001,0.01,0.1,1,10,100,1000]
penalty = ['l1', 'l2']
parameters = {'alpha': c, 'penalty':penalty}
svm = SGDClassifier(loss='hinge',class_weight = 'balanced')
clf = GridSearchCV(svm, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_features_avgw2v, y_train)
best_penalty = clf.best_estimator_.get_params()['penalty']
print('Best Penalty:', best_penalty)

train_auc = []
cv_auc = []
for i in c:
    svm = SGDClassifier(loss='hinge',alpha=i, penalty = best_penalty,class_weight = 'balanced')
    svm.fit(X_train_features_avgw2v, y_train)
    y_train_pred_avgw2v =  svm.decision_function(X_train_features_avgw2v)
    y_cv_pred_avgw2v =  svm.decision_function(X_cv_features_avgw2v)

    train_auc.append(roc_auc_score(y_train,y_train_pred_avgw2v))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred_avgw2v))

plt.plot(np.log(c), train_auc, label='Train AUC')
plt.plot(np.log(c), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("a: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
Best Penalty: l2
```

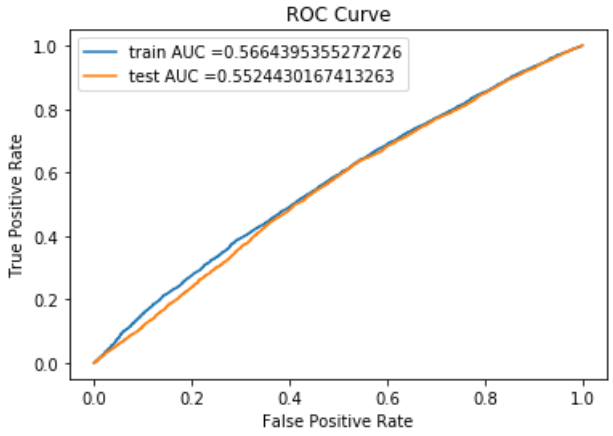


In [174]:

```
best_alpha = clf.best_estimator_.get_params()['alpha']
print("best_alpha: ", best_alpha)
```

```
from sklearn.metrics import roc_curve, auc


svm = SGDClassifier(loss='hinge',alpha=best_alpha, penalty=best_penalty,class_weight = 'balanced')
svm.fit(X_train_features_avgw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred_avgw2v= svm.decision_function(X_train_features_avgw2v)
y_test_pred_avgw2v = svm.decision_function(X_test_features_avgw2v)
train_fpr, train_tpr, thresholds = roc_curve(y_train, y_train_pred_avgw2v)
test_fpr, test_tpr, thresholds = roc_curve(y_test, y_test_pred_avgw2v)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()

print("="*100)
```

```
best alpha:  0.01
```




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

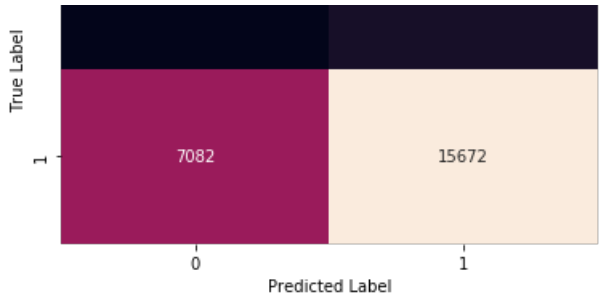
In [175]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds, train_fpr, train_tpr)

print("Train confusion matrix")
matrix_train= confusion_matrix(y_train, predict_with_best_t(y_train_pred_avgw2v, best_t))
print(matrix_train)
sns.heatmap(matrix_train,annot=True,cbar=False,fmt='d')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.3001933936217324 for threshold 0.072
Train confusion matrix
[[ 1672  2508]
 [ 7082 15672]]
```

Out[175]:

```
Text(0.5,1,'Train Confusion Matrix')
```



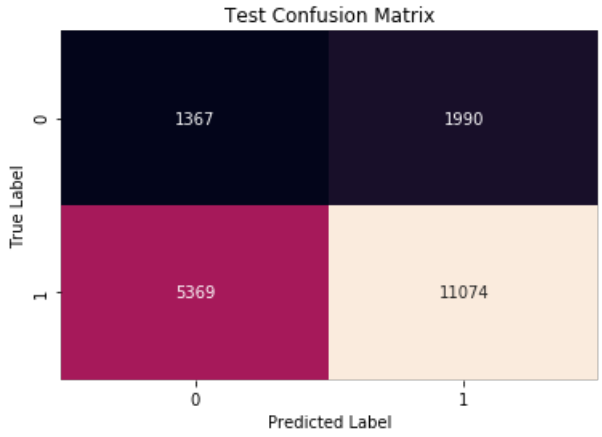
In [176]:

```
print("Test confusion matrix")
matrix_test= confusion_matrix(y_test, predict_with_best_t(y_test_pred_avgw2v, best_t))
print(matrix_test)
sns.heatmap(matrix_test,annot=True,cbar=False,fmt='d')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Test Confusion Matrix')
```

```
Test confusion matrix
[[ 1367  1990]
 [ 5369 11074]]
```

Out[176]:

```
Text(0.5,1,'Test Confusion Matrix')
```



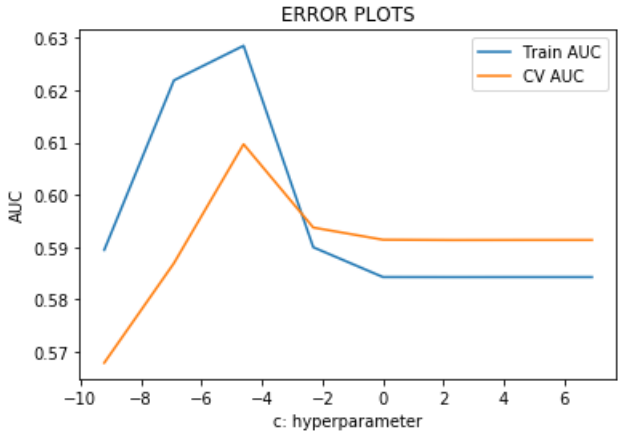
TFIDFw2v

In [177]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

c = [0.0001,0.001,0.01,0.1,1,10,100,1000]
penalty = ['l1', 'l2']
parameters = {'alpha': c, 'penalty':penalty}
svm = SGDClassifier(loss='hinge',class_weight = 'balanced')
clf = GridSearchCV(svm, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_features_tfidfw2v, y_train)
best_penalty = clf.best_estimator_.get_params()['penalty']
print('Best Penalty:', best_penalty)

train_auc = []
cv_auc = []

for i in c:
    svm = SGDClassifier(loss='hinge',alpha=i, penalty = best_penalty,class_weight = 'balanced')
    svm.fit(X_train_features_tfidfw2v, y_train)
    y_train_pred_tfidfw2v =  svm.decision_function(X_train_features_tfidfw2v)
```

```
    y_cv_pred_tfidfw2v =  svm.decision_function(X_cv_features_tfidfw2v)

    train_auc.append(roc_auc_score(y_train,y_train_pred_tfidfw2v))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred_tfidfw2v))

plt.plot(np.log(c), train_auc, label='Train AUC')
plt.plot(np.log(c), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

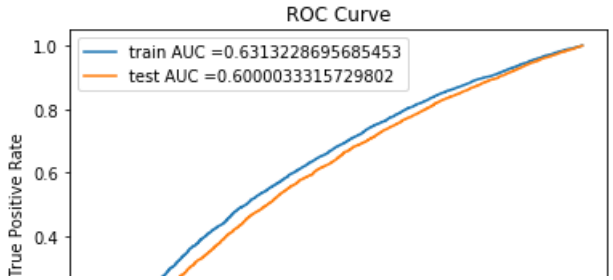
```
Best Penalty: l2
```

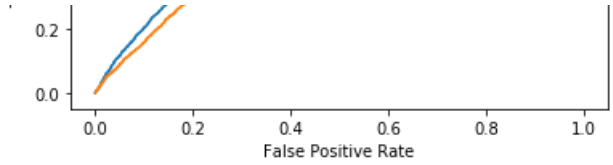


In [178]:

```
best_alpha = clf.best_estimator_.get_params()['alpha']
print("best alpha: ", best_alpha)
from sklearn.metrics import roc_curve, auc


svm = SGDClassifier(loss='hinge',alpha=best_alpha, penalty=best_penalty,class_weight = 'balanced')
svm.fit(X_train_features_tfidfw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred_tfidfw2v =  svm.decision_function(X_train_features_tfidfw2v)
y_test_pred_tfidfw2v = svm.decision_function(X_test_features_tfidfw2v)
train_fpr, train_tpr, thresholds = roc_curve(y_train,
svm.decision_function(X_train_features_tfidfw2v))
test_fpr, test_tpr, thresholds = roc_curve(y_test, svm.decision_function(X_test_features_tfidfw2v))

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()

print("="*100)
```

```
best alpha:  0.01
```





==========

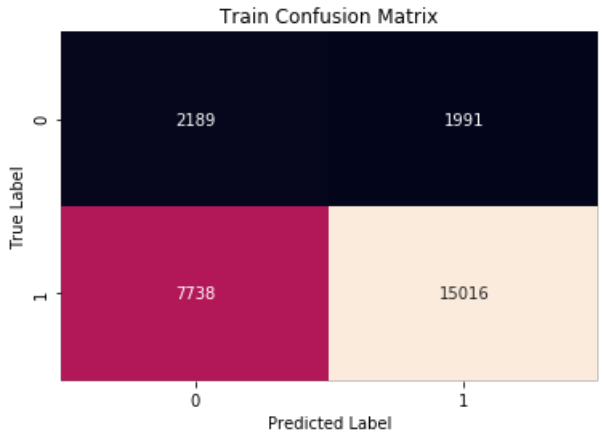
In [179]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds, train_fpr, train_tpr)

print("Train confusion matrix")
matrix_train= confusion_matrix(y_train, predict_with_best_t(y_train_pred_tfidfw2v, best_t))
print(matrix_train)
sns.heatmap(matrix_train,annot=True,cbar=False,fmt='d')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.3561986262050565 for threshold -0.306
Train confusion matrix
[[ 2189  1991]
 [ 7738 15016]]
```

Out[179]:

```
Text(0.5,1,'Train Confusion Matrix')
```



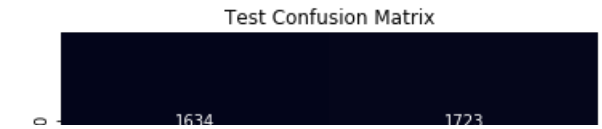
In [180]:

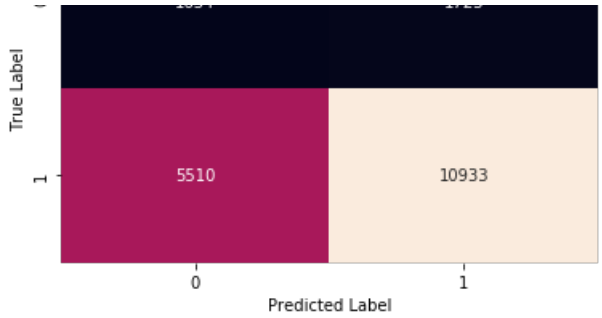
```
print("Test confusion matrix")
matrix_test= confusion_matrix(y_test, predict_with_best_t(y_test_pred_tfidfw2v, best_t))
print(matrix_test)
sns.heatmap(matrix_test,annot=True,cbar=False,fmt='d')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Test Confusion Matrix')
```

```
Test confusion matrix
[[ 1634  1723]
 [ 5510 10933]]
```

Out[180]:

```
Text(0.5,1,'Test Confusion Matrix')
```





2.5 Support Vector Machines with added Features `Set 5`

In [0]:

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

In [145]:

```
#https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_of_components_in_ts

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import TruncatedSVD
from scipy.sparse import csr_matrix

X_sparse = csr_matrix(X_train_tfidf)

tsvd = TruncatedSVD(n_components=X_sparse.shape[1]-1)
X_tsvd = tsvd.fit(X_train_tfidf)

tsvd_var_ratios = tsvd.explained_variance_ratio_

def select_n_components(var_ratio, goal_var: float) -> int:
    # Set initial variance explained so far
    total_variance = 0.0

    # Set initial number of features
    n_components = 0

    # For the explained variance of each feature:
    for explained_variance in var_ratio:

        # Add the explained variance to the total
        total_variance += explained_variance

        # Add one to the number of components
        n_components += 1

        # If we reach our goal level of explained variance
        if total_variance >= goal_var:
            # End the loop
            break

    # Return the number of components
    return n_components

select_n_components(tsvd_var_ratios, 0.95)
```

Out[145]:

```
5151
```

In [149]:

```
n= select_n_components(tsvd_var_ratios, 0.95)
```

In [153]:

```
tsvd = TruncatedSVD(n_components=n)
tsvd.fit(X_train_tfidf)
X_tsvd_train = tsvd.transform(X_train_tfidf)
X_tsvd_test = tsvd.transform(X_test_tfidf)
X_tsvd_cv = tsvd.transform(X_cv_tfidf)

print(X_tsvd_train.shape)
print(X_tsvd_test.shape)
print(X_tsvd_cv.shape)
```

```
(26934, 5151)
(19800, 5151)
(13266, 5151)
```

In [154]:

```
X_train_features_set5 =hstack((X_train_features_set5, X_tsvd_train))
X_test_features_set5 =hstack((X_test_features_set5, X_tsvd_test))
X_cv_features_set5= hstack((X_cv_features_set5, X_tsvd_cv))

print("Final Data matrix set 5:")
print(X_train_features_set5.shape, y_train.shape)
print(X_cv_features_set5.shape, y_cv.shape)
print(X_test_features_set5.shape, y_test.shape)
print("="*100)
```

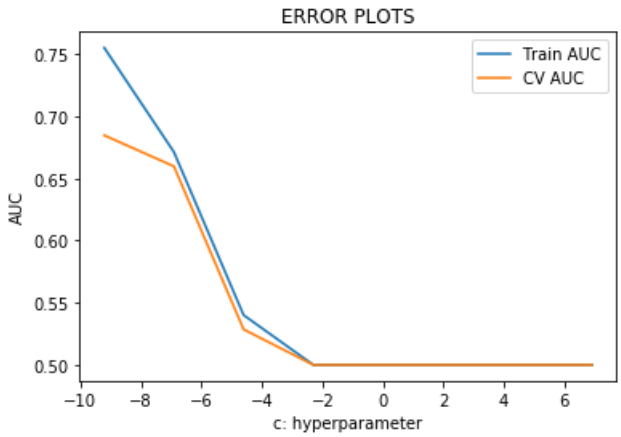
```
Final Data matrix set 5:
(26934, 5259) (26934,)
(13266, 5259) (13266,)
(19800, 5259) (19800,)
====================================================================================================
```

In [181]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt


c = [0.0001,0.001,0.01,0.1,1,10,100,1000]
penalty = ['l1', 'l2']
parameters = {'alpha': c, 'penalty':penalty}
svm = SGDClassifier(loss='hinge',class_weight = 'balanced')
clf = GridSearchCV(svm, parameters, cv=3, scoring='roc_auc')
clf.fit(X_train_features_set5, y_train)
best_penalty = clf.best_estimator_.get_params()['penalty']
print('Best Penalty:', best_penalty)

train_auc = []
cv_auc = []
for i in c:
    svm = SGDClassifier(loss='hinge',alpha=i, penalty= best_penalty,class_weight = 'balanced')
    svm.fit(X_train_features_set5, y_train)
    y_train_pred_set5 =  svm.decision_function(X_train_features_set5)
    y_cv_pred_set5 =  svm.decision_function(X_cv_features_set5)

    train_auc.append(roc_auc_score(y_train,y_train_pred_set5))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred_set5))

plt.plot(np.log(c), train_auc, label='Train AUC')
plt.plot(np.log(c), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("c: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
plt.show()
```

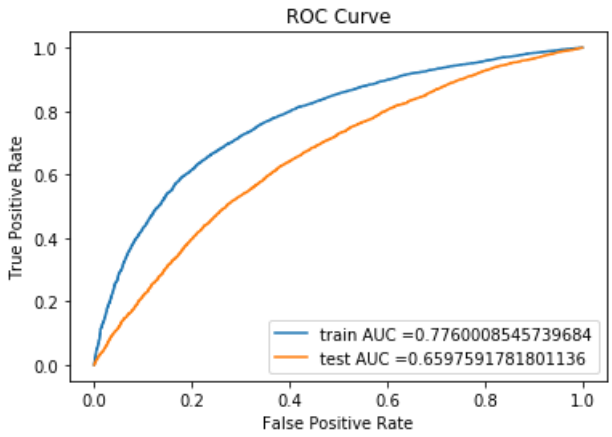
```
Best Penalty: l1
```



In [182]:

```
best_alpha = clf.best_estimator_.get_params()['alpha']
print("best alpha: ", best_alpha)
from sklearn.metrics import roc_curve, auc


svm = SGDClassifier(loss='hinge',alpha=best_alpha, penalty=best_penalty,class_weight = 'balanced')

svm.fit(X_train_features_set5, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs
y_train_pred_set5 =  svm.decision_function(X_train_features_set5)
y_test_pred_set5 = svm.decision_function(X_test_features_set5)
train_fpr, train_tpr, thresholds = roc_curve(y_train, svm.decision_function(X_train_features_set5)
)
test_fpr, test_tpr, thresholds = roc_curve(y_test, svm.decision_function(X_test_features_set5))

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()

print("="*100)
```

```
best alpha:  0.0001
```




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

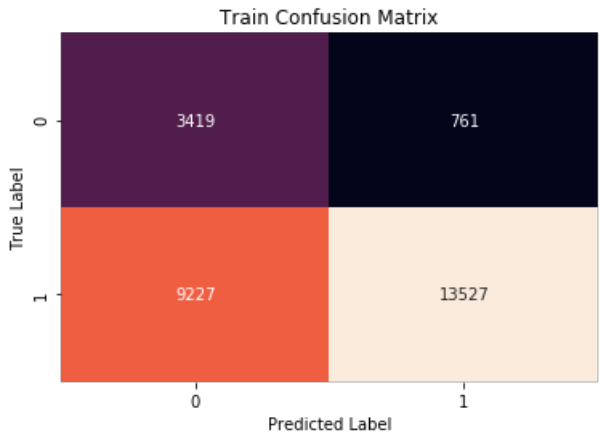
In [184]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds, train_fpr, train_tpr)

print("Train confusion matrix")
matrix_train= confusion_matrix(y_train, predict_with_best_t(y_train_pred_set5, best_t))
print(matrix_train)
sns.heatmap(matrix_train,annot=True,cbar=False,fmt='d')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.5065021955233278 for threshold 1.551
Train confusion matrix
[[ 3419   761]
 [ 9227 13527]]
```

Out[184]:

```
Text(0.5,1,'Train Confusion Matrix')
```



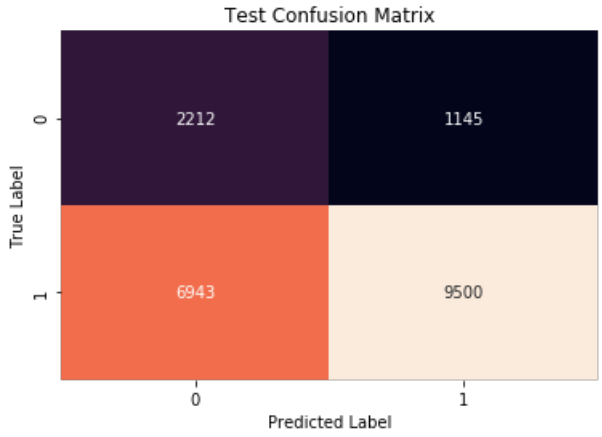
In [185]:

```
print("Test confusion matrix")
matrix_test= confusion_matrix(y_test, predict_with_best_t(y_test_pred_set5, best_t))
print(matrix_test)
sns.heatmap(matrix_test,annot=True,cbar=False,fmt='d')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Test Confusion Matrix')
```

```
Test confusion matrix
[[2212 1145]
 [6943 9500]]
```

Out[185]:

```
Text(0.5,1,'Test Confusion Matrix')
```



3. Conclusion

In [187]:

```
# Please compare all your models using Prettytable library

from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyperparameter","Penalty", "AUC"]

x.add_row(["BOW", "SVM", 0.1,"l2", 0.67])
x.add_row(["TFIDF", "SVM", 0.001,"l1", 0.65])
x.add_row(["AVGW2V", "SVM", 0.01,"l1", 0.55])
x.add_row(["TFIDFW2V", "SVM", 0.01,"l2", 0.60])
x.add_row(["TFIDF with TruncateSVD", "SVM", 0.0001,"l1", 0.65])


print(x)
```

```
+------------------------+-------+----------------+---------+------+
|       Vectorizer       | Model | Hyperparameter | Penalty | AUC  |
+------------------------+-------+----------------+---------+------+
|          BOW           |  SVM  |      0.1       |    l2   | 0.67 |
|         TFIDF          |  SVM  |     0.001      |    l1   | 0.65 |
|         AVGW2V         |  SVM  |      0.01      |    l1   | 0.55 |
|        TFIDFW2V        |  SVM  |      0.01      |    l2   | 0.6  |
| TFIDF with TruncateSVD |  SVM  |     0.0001     |    l1   | 0.65 |
+------------------------+-------+----------------+---------+------+
```

We observe that BOW and TFIDF have similarly good performances.

After adding the class_weights = 'balanced' parameter, the AUC scored of the models have increased as compared to before.