DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

- `Literature & Writing, Social Sciences`

Feature	Description
`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 [4]:

```
%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 [5]:

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

In [6]:

```
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 [7]:

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


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


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


project_data.head(2)
```

Out[7]:

	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 [8]:

```
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[8]:

	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 [9]:

```
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 [10]:

```
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 [11]:

```
# 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 [12]:

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

Out[12]:

	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 [13]:

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

In [14]:

```
# 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)

'''
```

Out[14]:

```
'\nprint(project_data[\'essay\'].values[0])\nprint("="*50)\nprint(project_data[\'essay\'].values[15
nprint("="*50)\nprint(project_data[\'essay\'].values[1000])\nprint("="*50)\nprint(project_data[\'es
'].values[20000])\nprint("="*50)\nprint(project_data[\'essay\'].values[99999])\nprint("="*50)\n\n'
```

In [15]:

```
# 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 [16]:

```
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 [17]:

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

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

In [18]:

```
#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
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 [19]:

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

In [20]:

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

In [21]:

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

Out[21]:

```
'person person no matter small dr seuss teach smallest students biggest enthusiasm learning
students learn many different ways using senses multiple intelligences use wide range techniques h
elp students succeed students class come variety different backgrounds makes wonderful sharing exp
eriences cultures including native americans school caring community successful learners seen coll
aborative student project based learning classroom kindergarteners class love work hands materials
many different opportunities practice skill mastered social skills work cooperatively friends cruc
ial aspect kindergarten curriculum montana perfect place learn agriculture nutrition students love
```

```
role play pretend kitchen early childhood classroom several kids ask try cooking real food take id
ea create common core cooking lessons learn important math writing concepts cooking delicious heal
thy food snack time students grounded appreciation work went making food knowledge ingredients cam
e well healthy bodies project would expand learning nutrition agricultural cooking recipes us peel
apples make homemade applesauce make bread mix healthy plants classroom garden spring also create
cookbooks printed shared families students gain math literature skills well life long enjoyment he
althy cooking nannan'
```

In [22]:

```
project_data['preprocessed_essays'] = preprocessed_essays
```

1.4 Preprocessing of `project_title`

In [23]:

```
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())

preprocessed_titles[1000]
```

Out[23]:

```
'empowering students through art learning about then now'
```

In [24]:

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

In [25]:

```
project_data.head()
```

Out[25]:

	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
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA	2016-04-27 00:46:53	Grades PreK-2

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

1.5 Preparing data for models

In [26]:

```
project_data.columns
```

Out[26]:

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

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

In [235]:

```
project_data['teacher_prefix'].fillna(" ", inplace = True)
from collections import Counter
my_counter = Counter()
for word in project_data['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]))



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

```
print(vectorizer.get_feature_names())


teacher_prefix_one_hot = vectorizer.transform(project_data['teacher_prefix'].values)
print("Shape of matrix after one hot encoding ",teacher_prefix_one_hot.shape)
```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encoding  (109248, 5)
```

In [236]:

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

project_data['project_grade_category'] = project_grade

from collections import Counter
my_counter = Counter()
for word in project_data['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]))


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


project_grade_category_one_hot =
vectorizer.transform(project_data['project_grade_category'].values)
print("Shape of matrix after one hot encoding ",project_grade_category_one_hot.shape)
```

```
['9-12', '6-8', '3-5', 'prek-2']
Shape of matrix after one hot encoding  (109248, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [237]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10, max_features=500)
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, 500)
```

In [286]:

```
text_bow= 0
```

In [239]:

```
vectorizer = CountVectorizer(min_df=5, max_features=500)
text_bow2 = vectorizer.fit_transform(preprocessed_titles)
print("Shape of matrix after one hot encodig ",text_bow2.shape)
```

```
Shape of matrix after one hot encodig  (109248, 500)
```

In [287]:

```
text_bow2=text_bow2[0:5000]
text_bow2 = 0
```

```
text_bow2 = 0
```

1.5.2.2 TFIDF vectorizer

In [60]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10, max_features = 4000)
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, 4000)
```

In [30]:

```
vectorizer = TfidfVectorizer(min_df=5, max_features = 1000)
text_tfidf2 = vectorizer.fit_transform(preprocessed_titles)
print("Shape of matrix after one hot encodig ",text_tfidf2.shape)
```

```
Shape of matrix after one hot encodig  (109248, 1000)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [61]:

```
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in 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')
```

```
Loading Glove Model
Done. 1917495  words loaded!
```

In [122]:

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

for i in preprocessed_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:
```

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

```
all the words in the coupus 15565238
the unique words in the coupus 58959
The number of words that are present in both glove vectors and our coupus 51494 ( 87.339 %)
word 2 vec length 51494
```

In [183]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-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 [63]:

```
# 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 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]))
```

```
109248
300
```

In [64]:

```
avg_w2v_vectors2 = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors2.append(vector)

print(len(avg_w2v_vectors2))
print(len(avg_w2v_vectors2[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████| 109248/109248
[00:02<00:00, 49903.45it/s]
```

```
109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [65]:

```
# 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 [66]:

```
tfidf_w2v_vectors = [];
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
[03:32<00:00, 513.87it/s]
```

```
109248
300
```

In [38]:

```
tfidf_w2v_vectors2 = [];
for sentence in tqdm(preprocessed_titles): # for each title
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the title
    for word in sentence.split(): # for each word in a title
        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_vectors2.append(vector)

print(len(tfidf_w2v_vectors2))
print(len(tfidf_w2v_vectors2[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████| 109248/109248
[00:03<00:00, 32934.56it/s]
```

```
109248
300
```

1.5.3 Vectorizing Numerical features

In [31]:

```
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 [32]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-
```

```
learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

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

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

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

```
Mean : 298.1193425966608, Standard deviation : 367.49634838483496
```

In [33]:

```
price_standardized
```

Out[33]:

```
array([[ 1.16172762],
       [-0.23153793],
       [ 0.08402983],
       ...,
       [ 0.27450792],
       [-0.0282706 ],
       [-0.79625102]])
```

1.5.4 Merging all the above features

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

In [245]:

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
```

```
(109248, 9)
(109248, 30)
(5000, 500)
(109248, 1)
```

In [246]:

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

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**

    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
    - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
    - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

2. **Hyper paramter tuning to find best K**

    - Find the best hyper parameter which results in 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 write your own for loops to do this task

3. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
   - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

4. **[Task-2]**

   - Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features

     -
       ```
       from sklearn.datasets import load_digits
       from sklearn.feature_selection import SelectKBest, chi2
       X, y = load_digits(return_X_y=True)
       X.shape
       X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
       X_new.shape
       ========
       output:
       (1797, 64)
       (1797, 20)
       ```
   - Repeat the steps 2 and 3 on the data matrix after feature selection

5. **Conclusion**

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

Note: Data Leakage

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

2. K Nearest Neighbor

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

In [155]:

```
from scipy import sparse
from scipy.sparse import csr_matrix
from scipy.sparse import lil_matrix
project_data['teacher_prefix'].fillna(" ", inplace = True)
Y = project_data['project_is_approved'].values
X = project_data

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

In [156]:

```
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 [157]:

```
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 [158]:

```
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, 22) (26934,)
(13266, 22) (13266,)
(19800, 22) (19800,)
```

In [159]:

```
X.head()
```

Out[159]:

	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	prek_2

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
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 × 22 columns

2.2 Make Data Model Ready: encoding numerical, categorical features

In [45]:

```
# 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 [160]:

```
# 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 [161]:

```
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 [162]:

```
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 [163]:

```
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)
```

```
(26934, 5)
(19800, 5)
(13266, 5)
  (0, 4)	1
  (1, 3)	1
  (2, 4)	1
  (3, 4)	1
  (4, 4)	1
```

In [164]:

```
vectorizer = CountVectorizer(vocabulary=list(project_grade_category.keys()), lowercase=False, binary=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 [165]:

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

In [166]:

```
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)
print("="*50)
```

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

In [167]:

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

In [168]:

```
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_price,X_train_quantity,X_train_previous)).tocsr()
X_test_features = hstack((X_test_cat, X_test_subcat,X_test_school,X_test_teacher,X_test_grade,
X_test_price,X_test_quantity,X_test_previous)).tocsr()
X_cv_features = hstack((X_cv_cat, X_cv_subcat,X_cv_school,X_cv_teacher,X_cv_grade, X_cv_price,X_cv_
quantity,X_cv_previous)).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,)
====================================================================================================
```

2.3 Make Data Model Ready: encoding eassay, and project_title

Bag of words

In [169]:

```
# 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 [170]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(max_features = 500)
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, 500) (26934,)
(13266, 500) (13266,)
(19800, 500) (19800,)
====================================================================================================
```

In [171]:

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

In [172]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(max_features = 100)
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, 100) (26934,)
(13266, 100) (13266,)
(19800, 100) (19800,)
====================================================================================================
```

In [173]:

```
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_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, 702) (26934,)
(13266, 702) (13266,)
(19800, 702) (19800,)
====================================================================================================
```

TFIDF

In [174]:

```
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, 9336) (26934,)
(13266, 9336) (13266,)
(19800, 9336) (19800,)
====================================================================================================
```

In [175]:

```
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 [176]:

```
X_train_features_tfidf = hstack((X_train_features, X_train_tfidf, X_train_tfidf2))
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 [180]:

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

In [183]:

```
# 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 [181]:

```
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 [184]:

```
# 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 [185]:

```
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 [186]:

```
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 [187]:

```
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 [188]:

```
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 [189]:

```
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
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 [190]:

```
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 [191]:

```
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 [192]:

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

In [193]:

```
# 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 [00:58<00:00, 462.21it/s]
```

```
26934
300
```

In [194]:

```
# 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 [00:40<00:00, 494.36it/s]
```

```
19800
300
```

In [195]:

```
# 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:
29<00:00, 450.10it/s]
```

```
13266
300
```

In [196]:

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

In [197]:

```
# 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 
[00:00<00:00, 28821.81it/s]
```

```
26934
300
```

In [198]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train_text)
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, 29720.05it/s]
```

```
19800
300
```

In [199]:

```
# 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, 26081.13it/s]
```

```
13266
300
```

All the features have been encoded and vectorized.

2.4 Appling KNN on different kind of featurization as mentioned in the instructions

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

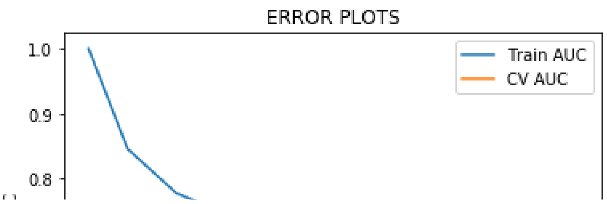
In [597]:

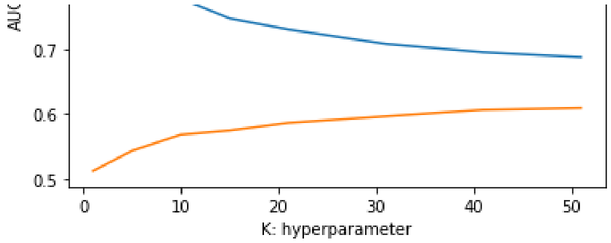
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging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

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

2.4.1 Applying KNN brute force on BOW, SET 1

In [215]:

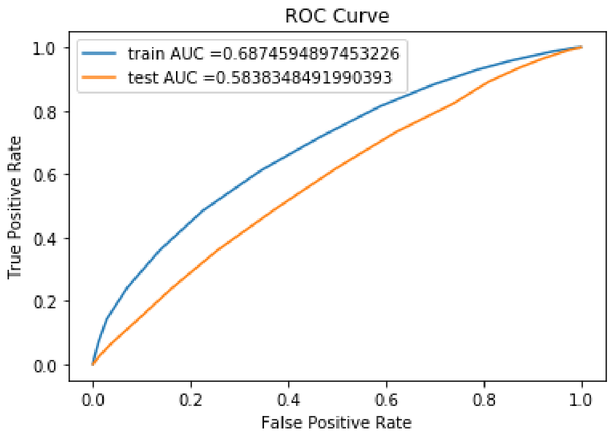
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_features_bow, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    y_train_pred_bow =  neigh.predict_proba(X_train_features_bow)[:,1]
    y_cv_pred_bow =  neigh.predict_proba(X_cv_features_bow)[:,1]

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





In [216]:

```
best_k = 51

from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.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_test_pred_bow = neigh.predict_proba(X_test_features_bow)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_features_bow)[:,
1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_features_bow)[:,1])

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)
```



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

In [217]:

```
# 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:
```

```
        else:
            predictions.append(0)
    return predictions
```

Confusion matrix for BOW

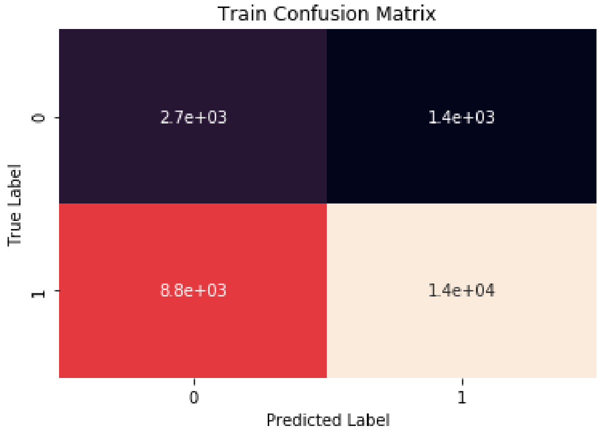
In [218]:

```
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)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.4011469879842358 for threshold 0.843
Train confusion matrix
[[ 2737  1443]
 [ 8814 13940]]
```

Out[218]:

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



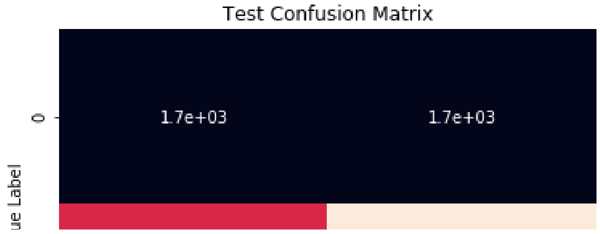
In [219]:

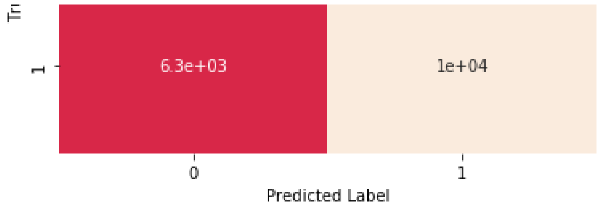
```
print("Test confusion matrix")
matrix_test= confusion_matrix(y_test, predict_with_best_t(y_test_pred_bow, best_t))
print(matrix_test)
sns.heatmap(matrix_test,annot=True,cbar=False)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Test Confusion Matrix')
```

```
Test confusion matrix
[[ 1684  1673]
 [ 6277 10166]]
```

Out[219]:

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

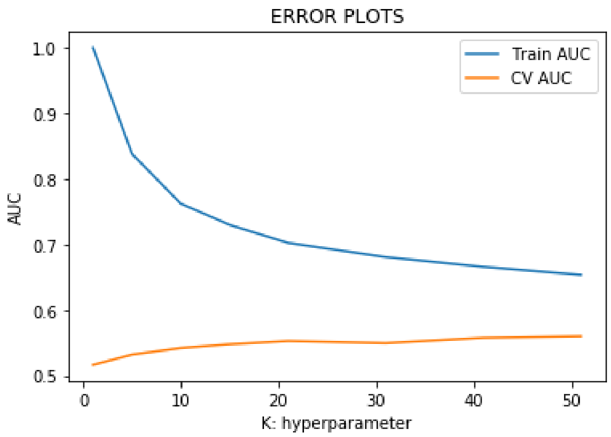




2.4.2 Applying KNN brute force on TFIDF, SET 2

In [220]:

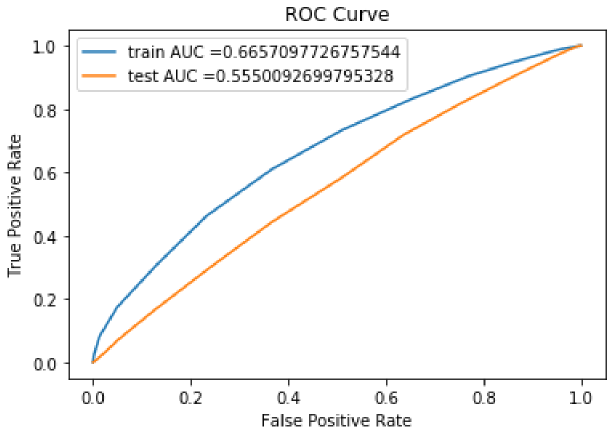
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_features_tfidf, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    y_train_pred_tfidf =  neigh.predict_proba(X_train_features_tfidf)[:,1]
    y_cv_pred_tfidf =  neigh.predict_proba(X_cv_features_tfidf)[:,1]

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



In [221]:

```
best_k = 41
```

```
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.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_test_pred_tfidf = neigh.predict_proba(X_test_features_tfidf)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_features_tfidf)[
:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_features_tfidf)[:,1])

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)
```



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

Confusion matrix for TFIDF

In [222]:

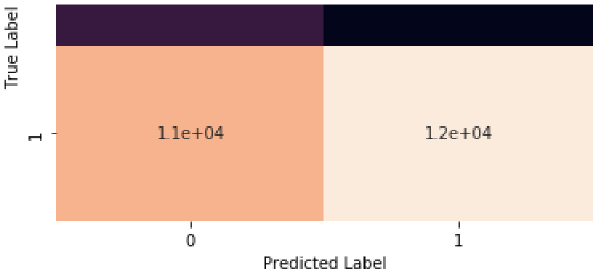
```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
matrix_train=confusion_matrix(y_train, predict_with_best_t(y_train_pred_tfidf, best_t))
print(matrix_train)
sns.heatmap(matrix_train,annot=True,cbar=False)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.3863683886696613 for threshold 0.854
Train confusion matrix
[[ 2884  1296]
 [10581 12173]]
```

Out[222]:

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

Train Confusion Matrix
2.9e+03
1.3e+03
0



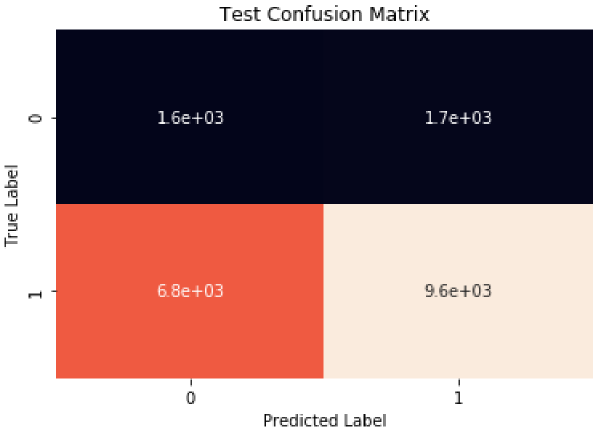
In [223]:

```python
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)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Test Confusion Matrix')
```

```
Test confusion matrix
[[1643 1714]
 [6840 9603]]
```

Out[223]:

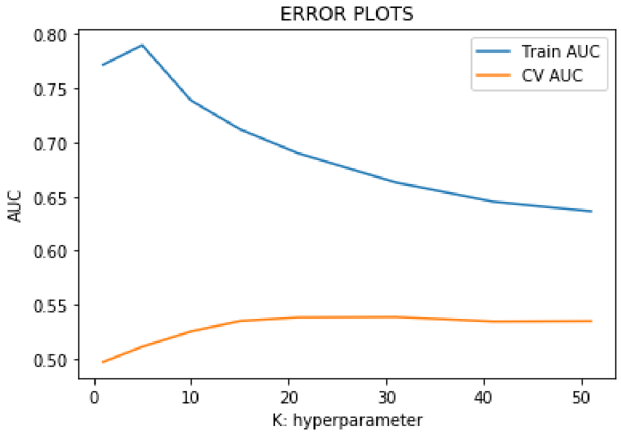
```
Text(0.5,1,'Test Confusion Matrix')
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

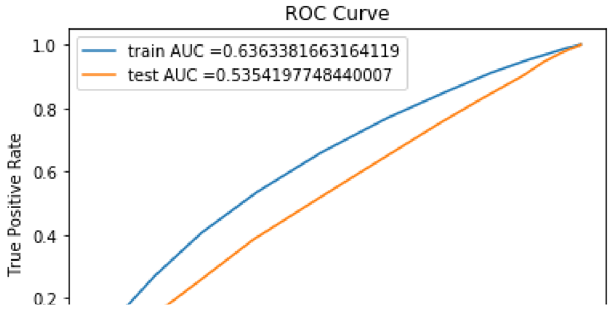
In [224]:

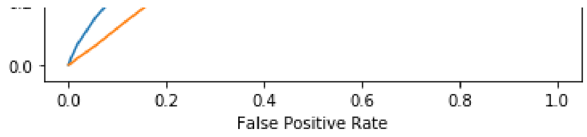
```python
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_features_avgw2v, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    y_train_pred_avgw2v =  neigh.predict_proba(X_train_features_avgw2v)[:,1]
    y_cv_pred_avgw2v =  neigh.predict_proba(X_cv_features_avgw2v)[:,1]

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



In [225]:

```
best_k = 51

from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.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_test_pred_avgw2v = neigh.predict_proba(X_test_features_avgw2v)[:,1]

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_features_avgw2v)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_features_avgw2v)[:,1]
)

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)
```





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

Confusion matrix for AVGw2v

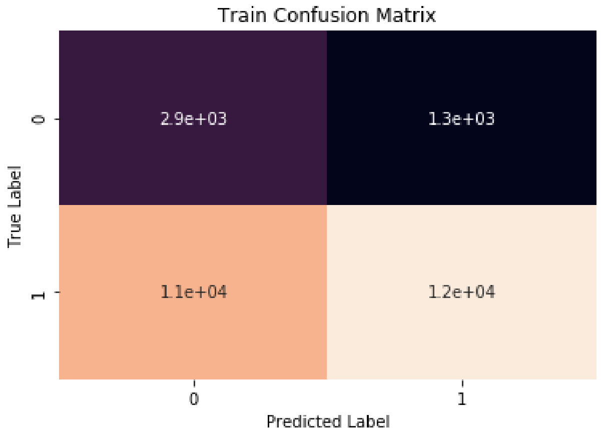
In [226]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(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)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.3552910724356578 for threshold 0.863
Train confusion matrix
[[ 2771  1409]
 [10559 12195]]
[[ 2884  1296]
 [10581 12173]]
```

Out[226]:

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



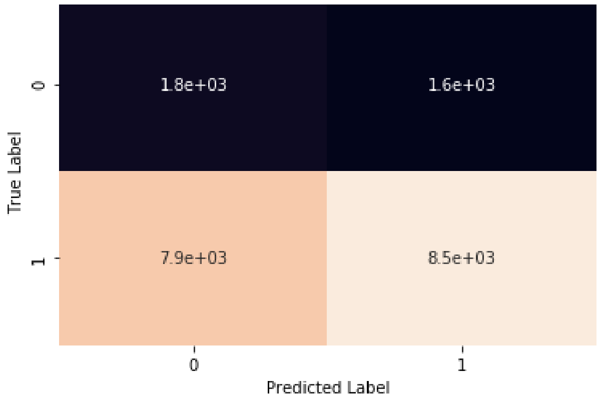
In [227]:

```
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)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Test Confusion Matrix')
```

```
Test confusion matrix
[[1792 1565]
 [7912 8531]]
```

Out[227]:

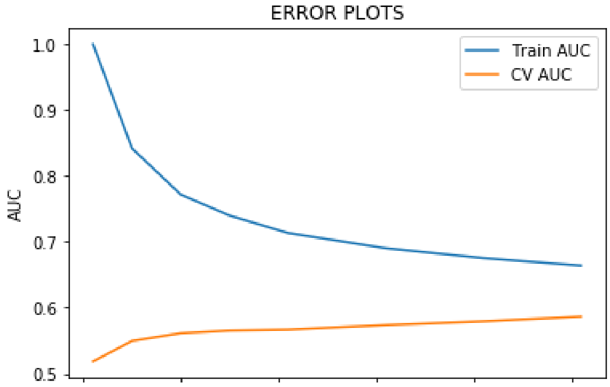
```
Text(0.5,1,'Test Confusion Matrix')
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

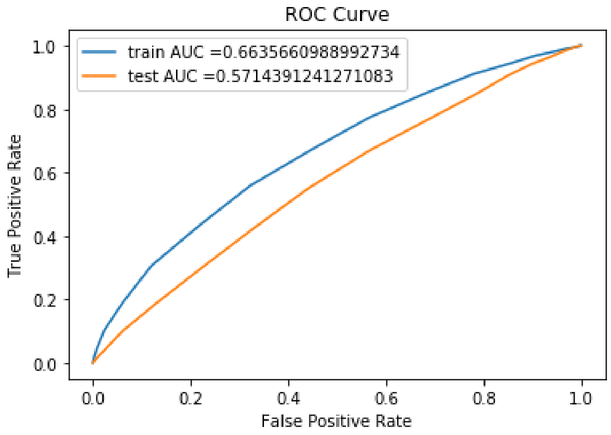
In [228]:

```
# Please write all the code with proper documentation
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.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 =  neigh.predict_proba(X_train_features_tfidfw2v)[:,1]
    y_cv_pred_tfidfw2v =  neigh.predict_proba(X_cv_features_tfidfw2v)[:,1]

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



In [229]:

```
best_k =51

from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.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_test_pred_tfidfw2v = neigh.predict_proba(X_test_features_tfidfw2v)[:,1]

train_fpr, train_tpr, thresholds = roc_curve(y_train,
neigh.predict_proba(X_train_features_tfidfw2v)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_features_tfidfw2v)[:,
1])

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)
```



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

Confusion matrix for TFIDFw2V

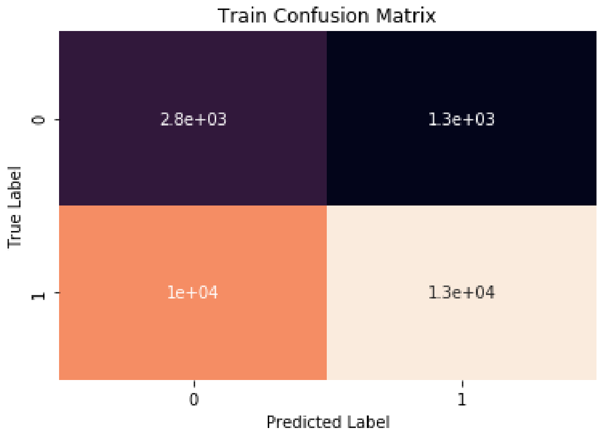
In [230]:

```
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)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.3787298137390429 for threshold 0.863
Train confusion matrix
[[ 2831  1349]
 [10030 12724]]
```

Out[230]:

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



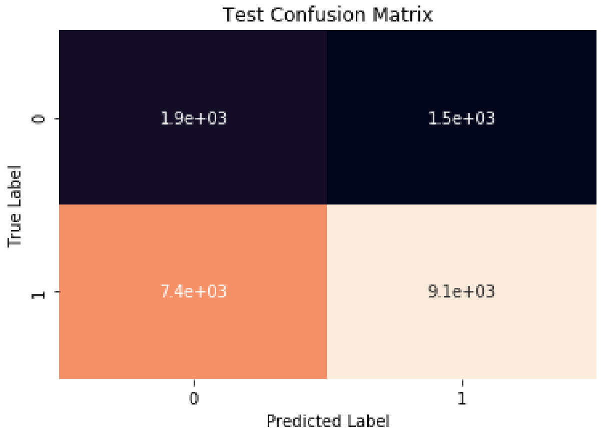
In [231]:

```
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)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Test Confusion Matrix')
```

```
Test confusion matrix
[[1869 1488]
 [7361 9082]]
```

Out[231]:

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



2.5 Feature selection with `SelectKBest`

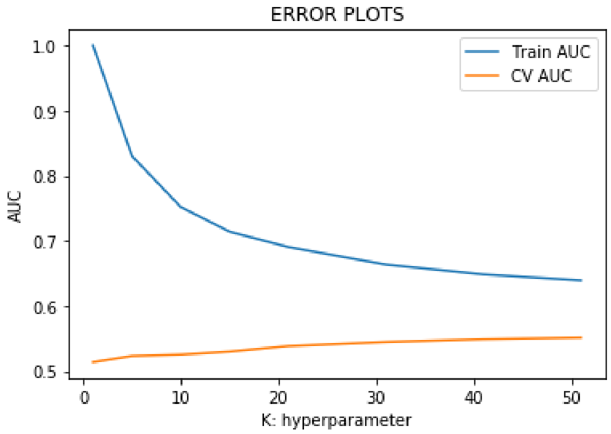
In [232]:

```
# 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

from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import SelectKBest, chi2
select=SelectKBest(chi2, k=2000)
select.fit(X_train_features_tfidf, y_train)
X_train_new_tfidf = select.transform(X_train_features_tfidf)
print(X_train_new_tfidf.shape)
X_test_new_tfidf = select.transform(X_test_features_tfidf)
print(X_test_new_tfidf.shape)
X_cv_new_tfidf = select.transform(X_cv_features_tfidf)
print(X_cv_new_tfidf.shape)
```

```
(26934, 2000)
(19800, 2000)
(13266, 2000)
```

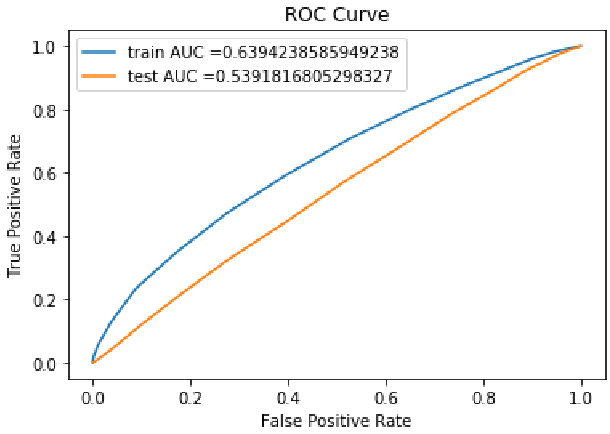
In [233]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_new_tfidf, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    y_train_pred_tfidf =  neigh.predict_proba(X_train_new_tfidf)[:,1]
    y_cv_pred_tfidf =  neigh.predict_proba(X_cv_new_tfidf)[:,1]

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



In [234]:

```
best_k = 51

from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_new_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_test_pred_tfidf = neigh.predict_proba(X_test_new_tfidf)[:,1]
train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_new_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_new_tfidf)[:,1])

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)
```



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

Confusion matrix for TFIDF with SelectKBest

In [235]:

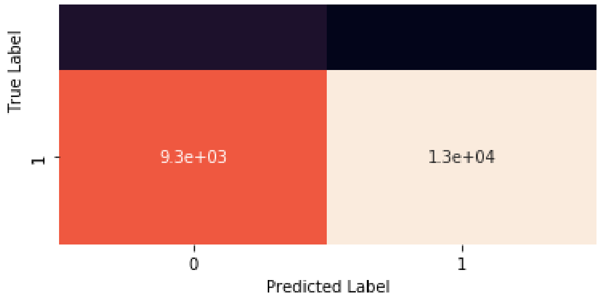
```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
matrix_train=(confusion_matrix(y_train, predict_with_best_t(y_train_pred_tfidf, best_t)))
print(matrix_train)
sns.heatmap(matrix_train,annot=True,cbar=False)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Train Confusion Matrix')
```

```
the maximum value of tpr*(1-fpr) 0.3577906066676115 for threshold 0.843
Train confusion matrix
[[ 2532  1648]
 [ 9314 13440]]
```

Out[235]:

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

Train Confusion Matrix
0
2.5e+03
1.6e+03



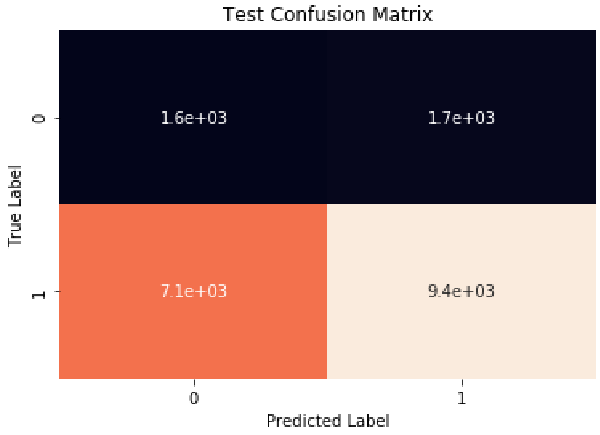
In [236]:

```
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)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Test Confusion Matrix')
```

```
Test confusion matrix
[[1631 1726]
 [7063 9380]]
```

Out[236]:

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



3. Conclusions

In [238]:

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

from prettytable import PrettyTable

x = PrettyTable()

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

x.add_row(["BOW", "Brute", 51, 0.58])
x.add_row(["TFIDF", "Brute", 41, 0.55])
x.add_row(["AVGW2V", "Brute", 51, 0.53])
x.add_row(["TFIDFW2V", "Brute", 51, 0.57])
x.add_row(["TFIDF", "SelectKBest", 51, 0.53])


print(x)
```

```
+------------+-------------+----------------+------+
| Vectorizer |    Model    | Hyperparameter | AUC  |
+------------+-------------+----------------+------+
|    BOW     |    Brute    |       51       | 0.58 |
|   TFIDF    |    Brute    |       41       | 0.55 |
|   AVGW2V   |    Brute    |       51       | 0.53 |
|  TFIDFW2V  |    Brute    |       51       | 0.57 |
|   TFIDF    | SelectKBest |       51       | 0.53 |
+------------+-------------+----------------+------+
```

We can see that the Brute BOW model with total max_features 600 is the best model for KNN