DonorsChoose

1. Importing All necessary LIbs to Work

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

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

```
/anaconda3/lib/python3.6/site-packages/smart_open/ssh.py:34: UserWarning: paramiko missing, openin
g SSH/SCP/SFTP paths will be disabled.  `pip install paramiko` to suppress
  warnings.warn('paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install
paramiko` to suppress')
```

2. Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv', nrows=30000)
resource_data = pd.read_csv('resources.csv', nrows=30000)
```

In [3]:

```
data = project_data[['id', 'teacher_prefix', 'school_state', 'project_grade_category',
                     'project_subject_categories', 'project_subject_subcategories', 'project_title',
                     'project_essay_1', 'project_essay_2', 'project_essay_3', 'project_essay_4',
                     'teacher_number_of_previously_posted_projects', 'project_is_approved']].copy()
```

```
#data = resource_data[['quantity','price']].copy()
```

In [4]:

```
# I have to add to data frames to spilt it properly so i seached as below
# gSearch Key: df add column from other df
# https://stackoverflow.com/a/20603020/6000190
data = data.join(resource_data[['quantity','price']])
```

In [5]:

```
# Lets preprocess a little bit.. like creating one eassy and removing 4 parts of it.
# Dropping ID too.. Cause We might dont need id for anything.


data["essay"] = data["project_essay_1"].map(str) +\
                        data["project_essay_2"].map(str) + \
                        data["project_essay_3"].map(str) + \
                        data["project_essay_4"].map(str)


data = data.drop(['id', 'project_essay_1', 'project_essay_2', 'project_essay_3', 'project_essay_4'
], axis=1)
```

In [6]:

```
data.head(2)
```

Out[6]:

	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcategories	project_title	teacher_
0	Mrs.	IN	Grades PreK-2	Literacy & Language	ESL, Literacy	Educational Support for English Learners at Home	
1	Mr.	FL	Grades 6-8	History & Civics, Health & Sports	Civics & Government, Team Sports	Wanted: Projector for Hungry Learners	

In [7]:

```
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 (30000, 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']
```

3. Text preprocessing

1.3.1 Essay Text

In [8]:

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

```
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 [9]:

```
sent = decontracted(data['essay'].values[100])
print(sent)
print("="*50)
```

```
I teach in a dual immersion 4th grade classroom. We teach 50% of the day in English and 50% in Spa
nish. My classroom is the English model for two classrooms of 30 students. Half of the 4th grade s
tudents at my school come to me to learn science, writing, and math in English in the morning, and
the other 1/2 come to me in the afternoon to learn the same subjects.  Most of my students are Eng
lish Learners; however, many come to this school speaking English only. This school is a Title I s
chool and is located in a high poverty area where many of the families are farm workers.
\r\n\r\nWe continually discuss how to implement the 4 Cs into our lessons: collaboration, communic
ation, critical thinking, and creativity. We also never forget about the 5th and most important \"
C,\" which is CARING! I have a great bunch of students who are enthusiastic about learning and rea
ding.  Most importantly, my students are so grateful for any help that we may receive. I really ha
ve a great group of kids who are so sweet.My students are becoming better researchers, project bui
lders and writers each day.  Currently, we do not have the ability for students to print and publi
sh their work.  All of my students have small student laptops, but they do not have the ability to
print and our school printer is rather far away and is overworked! We currently have plenty of pap
er but nothing to do with it! I would like my students to freely print their work in order to
display and share with other students and their families.   We need printers, ink and computers th
at can connect to the printer.  \r\nI would like my students to have a printing station within the
classroom so that they can create colorful displays and start to love to use the resources availab
le to them.   This is a 21st-century skill and I would like my students to be able to learn a
skill that is needed in the real world!nannan
==================================================
```

In [10]:

```
# \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)
```

```
I teach in a dual immersion 4th grade classroom. We teach 50% of the day in English and 50% in Spa
nish. My classroom is the English model for two classrooms of 30 students. Half of the 4th grade s
tudents at my school come to me to learn science, writing, and math in English in the morning, and
the other 1/2 come to me in the afternoon to learn the same subjects.  Most of my students are Eng
lish Learners; however, many come to this school speaking English only. This school is a Title I s
chool and is located in a high poverty area where many of the families are farm workers.     We co
ntinually discuss how to implement the 4 Cs into our lessons: collaboration, communication, critic
al thinking, and creativity. We also never forget about the 5th and most important  C,  which is C
ARING! I have a great bunch of students who are enthusiastic about learning and reading.  Most imp
ortantly, my students are so grateful for any help that we may receive. I really have a great grou
p of kids who are so sweet.My students are becoming better researchers, project builders and write
rs each day.  Currently, we do not have the ability for students to print and publish their work.
All of my students have small student laptops, but they do not have the ability to print and our s
chool printer is rather far away and is overworked! We currently have plenty of paper but nothing
to do with it! I would like my students to freely print their work in order to display and share w
ith other students and their families.   We need printers, ink and computers that can connect to t
he printer.    I would like my students to have a printing station within the classroom so that th
ey can create colorful displays and start to love to use the resources available to them.   This i
s a 21st-century skill and I would like my students to be able to learn a skill that is needed in
the real world!nannan
```

In [11]:

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

```
I teach in a dual immersion 4th grade classroom We teach 50 of the day in English and 50 in
Spanish My classroom is the English model for two classrooms of 30 students Half of the 4th grade
students at my school come to me to learn science writing and math in English in the morning and t
he other 1 2 come to me in the afternoon to learn the same subjects Most of my students are Englis
h Learners however many come to this school speaking English only This school is a Title I school
and is located in a high poverty area where many of the families are farm workers We continually d
iscuss how to implement the 4 Cs into our lessons collaboration communication critical thinking an
d creativity We also never forget about the 5th and most important C which is CARING I have a grea
t bunch of students who are enthusiastic about learning and reading Most importantly my students a
re so grateful for any help that we may receive I really have a great group of kids who are so swe
et My students are becoming better researchers project builders and writers each day Currently we
do not have the ability for students to print and publish their work All of my students have small
student laptops but they do not have the ability to print and our school printer is rather far awa
y and is overworked We currently have plenty of paper but nothing to do with it I would like my
students to freely print their work in order to display and share with other students and their
families We need printers ink and computers that can connect to the printer I would like my studen
ts to have a printing station within the classroom so that they can create colorful displays and s
tart to love to use the resources available to them This is a 21st century skill and I would like
my students to be able to learn a skill that is needed in the real world nannan
```

In [12]:

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

In [13]:

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

```
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████| 30000/30000 [00:26<00:00, 1139.50it/s]
```

In [14]:

```
# after preprocesing
preprocessed_essays[191]
```

Out[14]:

```
'our school large elementary school warm inviting most students come variety different economic ba
ckgrounds 70 students receiving free reduced price lunches harvesters food bank provides backpack
snacks one hundred students many students arrive school little no school supplies my students exci
ted learning i commitment whatever takes ensure talents developed atmosphere high expectations per
sonal support students school responsible respectful always ready learn flexible classrooms give s
tudents choice kind learning space works best help work collaboratively communicate engage
critical thinking the couch table set give students opportunities i want classroom represent real
world seating arrangements classrooms 70 years ago students need seating confront gives
alternative desk i believe providing flexible soft alternative seating students would able move re
lease energy happier comfortable work nannan'
```

In [15]:

```
data['essay']=preprocessed_essays
```

1.3.2 Project title Text

In [16]:

```
# similarly you can preprocess the titles also
# Processing steps for project title
# 1. Clean pharase
# 2. Remove String patterns
# 3. Remove Special charcter
# 4. Remove Stop words


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

```
100%|██████████| 30000/30000 [00:01<00:00, 25381.27it/s]
```

In [17]:

```
# after preprocesing
preprocessed_project_title[0]
```

Out[17]:

```
'educational support english learners home'
```

In [18]:

```
data['project_title'][0]
```

Out[18]:

'Educational Support for English Learners at Home'

In [19]:

```
# for i in tqdm(range(len(preprocessed_project_title))):
#     data['project_title'][i] = preprocessed_project_title[i]

data['project_title']=preprocessed_project_title
```

Preprocessing for project_grade_category data.

In [20]:

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

```
100%|██████████| 30000/30000 [00:00<00:00, 32323.31it/s]
```

In [21]:

```
print(data['project_grade_category'][0])
print(preprocessed_project_grade_category[0])
```

```
Grades PreK-2
grades prek 2
```

In [22]:

```
data['project_grade_category']=preprocessed_project_grade_category
```

Preprocessing for project_subject_categories data.

In [23]:

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

```
100%|██████████| 30000/30000 [00:00<00:00, 182757.28it/s]
```

In [24]:

```
print(data['project_subject_categories'][0])
print(sub_cat_list[0])
```

```
Literacy & Language
ESL Literacy
```

In [25]:

```
data['project_subject_categories']=sub_cat_list
```

Preprocessing for project_subject_subcategories data.

In [26]:

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

```
100%|██████████| 30000/30000 [00:00<00:00, 229599.68it/s]
```

In [27]:

```
data['project_subject_subcategories'] = sub_cat_sub_list
```

In [28]:

```
print(data['project_subject_subcategories'][0])
print(sub_cat_sub_list[0])
```

```
ESL Literacy
ESL Literacy
```

4. Spliting Data

In [29]:

In [29]:

```
data.head(1)
```

Out[29]:

	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcategories	project_title	teacher_
0	Mrs.	IN	grades prek 2	ESL Literacy	ESL Literacy	educational support english learners home	

In [30]:

```
# Just Checking for Nan values .. before splitting them up
# Google search :- check which column has nan pandas
# https://stackoverflow.com/questions/36226083/how-to-find-which-columns-contain-any-nan-value-in-pandas-dataframe-python/36226137

data.isna().any()
```

Out[30]:

```
teacher_prefix                                 True
school_state                                  False
project_grade_category                        False
project_subject_categories                    False
project_subject_subcategories                 False
project_title                                 False
teacher_number_of_previously_posted_projects  False
project_is_approved                           False
quantity                                      False
price                                         False
essay                                         False
dtype: bool
```

In [31]:

```
data['teacher_prefix'].unique()
```

Out[31]:

```
array(['Mrs.', 'Mr.', 'Ms.', 'Teacher', nan], dtype=object)
```

In [32]:

```
data.fillna("", inplace=True)
print("Data Cleaned from nan")
```

```
Data Cleaned from nan
```

In [33]:

```
data.isna().any()
```

Out[33]:

```
teacher_prefix                                False
school_state                                  False
project_grade_category                        False
project_subject_categories                    False
project_subject_subcategories                 False
project_title                                 False
teacher_number_of_previously_posted_projects  False
project_is_approved                           False
quantity                                      False
price                                         False
essay                                         False
dtype: bool
```

In [34]:

```
y = data['project_is_approved'].values
data.drop(['project_is_approved'], axis=1, inplace=True)
data.head(1)
```

Out[34]:

	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcategories	project_title	teacher_
0	Mrs.	IN	grades prek 2	ESL Literacy	ESL Literacy	educational support english learners home	

In [35]:

```
X = data
```

In [36]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

5. Vectorizing Data

CountVectorizer:

Convert a collection of text documents to a matrix of token counts. Like as expalined in Video.. [0 ,1(if present) , 0(if not present)]

- vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4))

it will initiate the CountVectorizer with given parameters. parameters: min_dif= 10, float in range ngram_range= (1, 4), count range tuple of combinatio of word

- vectorizer.fit(X_train['essay'].values)

It will create a vocabulary dictionary from all tokens in the raw documents. and fit has to apply on train data cause it should learn of the train data itself. it should not learn from test data .

- X_train_essay_bow = vectorizer.transform(X_train['essay'].values)

.Learn the vocabulary dictionary from data.. and output with bow ..

Difference Between fit(), transform() and fit_transform()

- Difference Between fit() n transform() and fit_transform()

fit() - fit caluclates the value of 🅰️🅱️ and saves internally then transform() - Apply the transformation to given list of data points. fit_tranform() - this method not only calculate then applies on the data points and returns the transformed dataset too. we can say like fit_tranform is the combine form of fit() and tranform()

In [ ]:

```

```

In [ ]:

```

```

*** Vectorizing Essay**

In [37]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4)) # , max_features=5000
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

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

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*50)
```

```
After vectorizations
(13467, 42008) (13467,)
(6633, 42008) (6633,)
(9900, 42008) (9900,)
==================================================
```

In [38]:

```
print(X_train_essay_bow.shape)
# print(X_train_essay_bow[0:3,0:3])
print(X_train_essay_bow.toarray()[:5,:5])
print(X_cv_essay_bow.toarray()[:5,:5])
print(X_test_essay_bow.toarray()[:5,:5])
```

```
(13467, 42008)
[[0 0 0 0 0]
 [0 0 0 1 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
[[0 0 0 1 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

*** Vectorizing Title**

In [39]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4)) # , max_features=5000
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on train data

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

print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print("="*50)
```

```
After vectorizations
(13467, 1223) (13467,)
(6633, 1223) (6633,)
(9900, 1223) (9900,)
```

==

In [40]:

```
print(X_train_title_bow.shape)
# print(X_train_essay_bow[0:3,0:3])
print(X_train_title_bow.toarray()[:5,:5])
print(X_cv_title_bow.toarray()[:5,:5])
print(X_test_title_bow.toarray()[:5,:5])
```

```
(13467, 1223)
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

*** Vectorizing State**

In [41]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(13467, 51) (13467,)
(6633, 51) (6633,)
(9900, 51) (9900,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv
', 'wy']
====================================================================================================
```

In [42]:

```
print(X_train_state_ohe.shape)
# print(X_train_essay_bow[0:3,0:3])
print(X_train_state_ohe.toarray()[:5,:5])
print(X_cv_state_ohe.toarray()[:5,:5])
print(X_test_state_ohe.toarray()[:5,:5])
```

```
(13467, 51)
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

```
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 1]
 [0 0 0 0 0]
 [0 0 0 0 0]]
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

*** Vectorizing Teacher_prefix**

In [43]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(13467, 4) (13467,)
(6633, 4) (6633,)
(9900, 4) (9900,)
['mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

In [44]:

```
print(X_train_teacher_ohe.shape)
# print(X_train_essay_bow[0:3,0:3])
print(X_train_teacher_ohe.toarray()[:5,:5])
print(X_cv_teacher_ohe.toarray()[:5,:5])
print(X_test_teacher_ohe.toarray()[:5,:5])
```

```
(13467, 4)
[[0 0 1 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 1 0 0]
 [0 0 1 0]]
[[0 0 1 0]
 [0 1 0 0]
 [1 0 0 0]
 [0 0 1 0]
 [0 1 0 0]]
[[0 1 0 0]
 [0 1 0 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 1 0 0]]
```

*** Vectorizing Project Grade Category**

In [45]:

```
my_counter_project_grade = Counter()
for word in X_train['project_grade_category'].values:
        my_counter_project_grade.update(word.split(","))
```

```
cat_dict_procat = dict(my_counter_project_grade)
sorted_procat = dict(sorted(cat_dict_procat.items(), key=lambda kv: kv[1]))

# we use count vectorizer to convert the values into one hot encoded features
vectorizer_teacher = CountVectorizer(vocabulary=list(sorted_procat.keys()), lowercase=False,
binary=True)
vectorizer_teacher.fit(X_train['project_grade_category'].values )



# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer_teacher.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer_teacher.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer_teacher.transform(X_test['project_grade_category'].values)




print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer_teacher.get_feature_names())
print("="*100)
```

```
After vectorizations
(13467, 4) (13467,)
(6633, 4) (6633,)
(9900, 4) (9900,)
['grades 9 12', 'grades 6 8', 'grades 3 5', 'grades prek 2']
====================================================================================================
```

In [46]:

```
print(X_train_grade_ohe.shape)
# print(X_train_essay_bow[0:3,0:3])
print(X_train_grade_ohe.toarray()[:5,:5])
print(X_cv_grade_ohe.toarray()[:5,:5])
print(X_test_grade_ohe.toarray()[:5,:5])
```

```
(13467, 4)
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

In [ ]:

```

```

*** Vectorizing Project Subject Category**

In [47]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_subject_categories'].values) # fit has to happen only on train
data

# we use the fitted CountVectorizer to convert the text to vector
X_train_subject_cat = vectorizer.transform(X_train['project_subject_categories'].values)
```

```
X_cv_subject_cat = vectorizer.transform(X_cv['project_subject_categories'].values)
X_test_subject_cat = vectorizer.transform(X_test['project_subject_categories'].values)

print("After vectorizations")
print(X_train_subject_cat.shape, y_train.shape)
print(X_cv_subject_cat.shape, y_cv.shape)
print(X_test_subject_cat.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(13467, 30) (13467,)
(6633, 30) (6633,)
(9900, 30) (9900,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================================
```

In [48]:

```
print(X_train_subject_cat.shape)
# print(X_train_essay_bow[0:3,0:3])
print(X_train_subject_cat.toarray()[:5,:5])
print(X_cv_subject_cat.toarray()[:5,:5])
print(X_test_subject_cat.toarray()[:5,:5])
```

```
(13467, 30)
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 1]
 [1 0 0 0 0]
 [0 0 0 0 0]]
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

In [ ]:

```
```

*** Vectorizing Project Subject Subcategories**

In [49]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_subject_subcategories'].values) # fit has to happen only on train
data

# we use the fitted CountVectorizer to convert the text to vector
X_train_sub_cat = vectorizer.transform(X_train['project_subject_subcategories'].values)
X_cv_sub_cat = vectorizer.transform(X_cv['project_subject_subcategories'].values)
X_test_sub_cat = vectorizer.transform(X_test['project_subject_subcategories'].values)

print("After vectorizations")
print(X_train_sub_cat.shape, y_train.shape)
print(X_cv_sub_cat.shape, y_cv.shape)
print(X_test_sub_cat.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(13467, 30) (13467,)
(6633, 30) (6633,)
(9900, 30) (9900,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
==========================================================================================================
```

In [50]:

```
print(X_train_sub_cat.shape)
# print(X_train_essay_bow[0:3,0:3])
print(X_train_sub_cat.toarray()[:5,:5])
print(X_cv_sub_cat.toarray()[:5,:5])
print(X_test_sub_cat.toarray()[:5,:5])
```

```
(13467, 30)
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 1]
 [1 0 0 0 0]
 [0 0 0 0 0]]
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]]
```

*** Vectorizing Teacher Number Of Previously Posted Projects**

In [51]:

```
# vectorizer = CountVectorizer()
# vectorizer.fit(X_train['teacher_number_of_previously_posted_projects'].values) # fit has to happ
en only on train data

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

X_train_prPos_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].
values.reshape(-1,1))
X_cv_prPos_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values
.reshape(-1,1))
X_test_prPos_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].va
lues.reshape(-1,1))

print("After vectorizations")
print(X_train_prPos_norm.shape, y_train.shape)
print(X_cv_prPos_norm.shape, y_cv.shape)
print(X_test_prPos_norm.shape, y_test.shape)
print(X_cv_prPos_norm)
print("="*100)
```

```
After vectorizations
(13467, 1) (13467,)
(6633, 1) (6633,)
(9900, 1) (9900,)
[[1.]
 [0.]
 [1.]
```

```
 ...
 [1.]
 [1.]
 [0.]]
==================================================================================================
```

*** Vectorizing Quantity**

In [52]:

```
# vectorizer = CountVectorizer()
# vectorizer.fit(X_train['teacher_number_of_previously_posted_projects'].values) # fit has to happ
en only on train data


normalizer = Normalizer()

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

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

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

```
After vectorizations
(13467, 1) (13467,)
(6633, 1) (6633,)
(9900, 1) (9900,)
[[1.]
 [1.]
 [1.]
 ...
 [1.]
 [1.]
 [1.]]
==================================================================================================
```

*** Standardising Price**

In [53]:

```
# vectorizer = CountVectorizer()
# vectorizer.fit(X_train['teacher_number_of_previously_posted_projects'].values) # fit has to happ
en only on train data


normalizer = Normalizer()

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

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

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

```
After vectorizations
(13467, 1) (13467,)
```

```
(6633, 1) (6633,)
(9900, 1) (9900,)
[[1.]
 [1.]
 [1.]
 ...
 [1.]
 [1.]
 [1.]]
==========================================================================================================
```

In [ ]:

```
```

In [ ]:

```
```

In [54]:

```
X_train_essay_bow.shape
```

Out[54]:

```
(13467, 42008)
```

In [55]:

```
X_train_price_norm.shape
```

Out[55]:

```
(13467, 1)
```

In [56]:

```
X_cv_essay_bow.shape
```

Out[56]:

```
(6633, 42008)
```

In [57]:

```
X_cv_price_norm.shape
```

Out[57]:

```
(6633, 1)
```

In [58]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack
X_tr = hstack(
    (X_train_essay_bow,
     X_train_title_bow,
     X_train_state_ohe,
     X_train_teacher_ohe,
     X_train_sub_cat,
     X_train_prPos_norm,
     X_train_grade_ohe,
     X_train_quantity_norm,
     X_train_price_norm)).tocsr()


X_cr = hstack(
```

```
    (X_cv_essay_bow,
     X_cv_title_bow,
     X_cv_state_ohe,
     X_cv_teacher_ohe,
     X_cv_sub_cat,
     X_cv_prPos_norm,
     X_cv_grade_ohe,
     X_cv_quantity_norm,
     X_cv_price_norm)).tocsr()



X_te = hstack(
    (X_test_essay_bow,
     X_test_title_bow,
     X_test_state_ohe,
     X_test_teacher_ohe,
     X_test_sub_cat,
     X_test_prPos_norm,
     X_test_grade_ohe,
     X_test_quantity_norm,
     X_test_price_norm)).tocsr()



print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(13467, 43323) (13467,)
(6633, 43323) (6633,)
(9900, 43323) (9900,)
====================================================================================================
```

6. Let's Build a model with above data

In [59]:

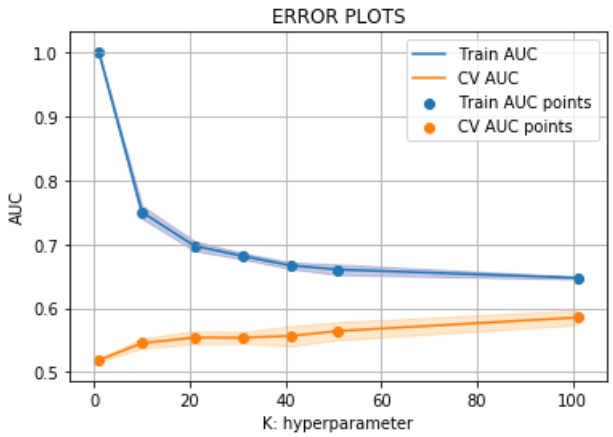
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

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

    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

Applying KNN brute force on BOW, SET 1

In [60]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 10, 21, 31, 41, 51, 101]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
```

```
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



- As per the error Plot The minimum gap represents the best k value.
- And in this plot 71 is having less space between Train and Test Auc Graphs

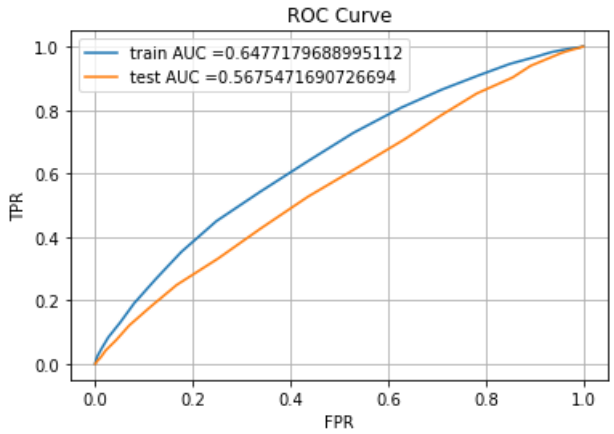
In [ ]:

```
```

In [61]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

best_k_m1 =71
# for i in tqdm(parameters):
neigh = KNeighborsClassifier(n_neighbors=best_k_m1)
neigh.fit(X_tr, 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 = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

```
m1_Auc = str(auc(train_fpr, train_tpr))


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



- Train AUC values is 0.64
- Test AUC value is 0.56
- As i tried with Diffrent K Values it representing with diffrent values

In [62]:

```
# # For Ploting Code I got from officaial website of sklearn
# # https://scikit-learn.org/0.16/auto_examples/model_selection/plot_confusion_matrix.html

# def plot_confusion_matrix(cm, title='Confusion matrix', cmap=plt.cm.Blues, targetname=[0,1]):
#     plt.imshow(cm, interpolation='nearest', cmap=cmap)
#     plt.title(title)
#     plt.colorbar()
#     tick_marks = np.arange(len(targetname))
#     plt.xticks(tick_marks, targetname, rotation=45)
#     plt.yticks(tick_marks, targetname)
#     plt.tight_layout()
#     plt.ylabel('True label')
#     plt.xlabel('Predicted label')
```

In [63]:

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

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

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

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

In [64]:

```
from sklearn.metrics import confusion_matrix
```

```
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24916307797602497 for threshold 0.775
[[ 977 1097]
 [3094 8299]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24847772104273044 for threshold 0.775
[[ 559  966]
 [2447 5928]]
```

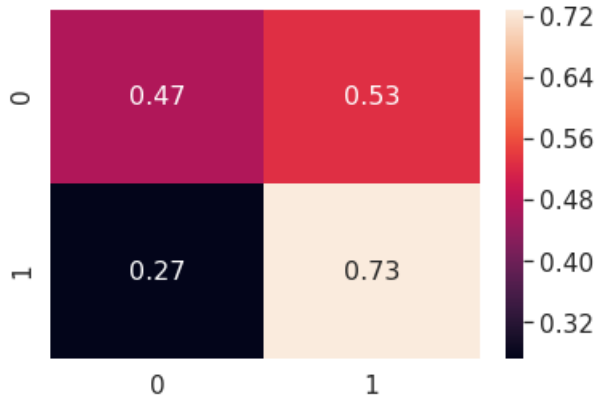
In [65]:

```
# https://stackoverflow.com/a/42265865/6000190
def plotConfusionMatrix(cm):
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(cm_normalized, annot=True,annot_kws={"size": 16})# font size
```

In [66]:

```
# Normalize the confusion matrix by row (i.e by the number of samples
# in each class)
# https://scikit-learn.org/0.16/auto_examples/model_selection/plot_confusion_matrix.html

cm_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))

plotConfusionMatrix(cm_train)
```

```
the maximum value of tpr*(1-fpr) 0.24916307797602497 for threshold 0.775
```

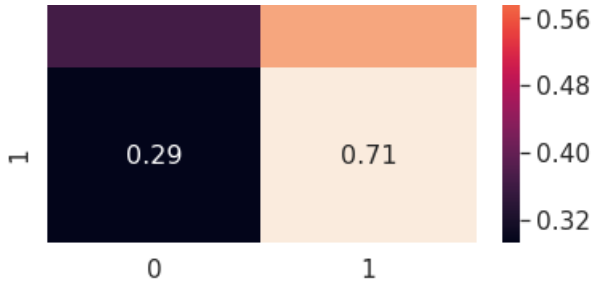


- TP = 0.47
- TN = 0.53
- FP = 0.27
- FN = 0.73
- FN has a good value of closer to 1 but TP Should have higher value as well. Means.. we can try with more data.

In [67]:

```
cm_test = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))

plotConfusionMatrix(cm_test)
```

```
the maximum value of tpr*(1-fpr) 0.24847772104273044 for threshold 0.775
```



- TP = 0.37
- TN = 0.63
- FP = 0.29
- FN = 0.71
- FN has higher values comparing to TP.

Applying KNN brute force on AVG W2V, SET 3

FEaturization to Text for W2v

In [68]:

```
# 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

#https://drive.google.com/open?id=14nf-h6aYdhL_01I8DVg9CFZ5aMqAXeTi

with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [69]:

```
# THis is w2v vectorization type.

def makew2vList(xtrainEassyArray):
    avg_w2v_vector_eassy = []; # the avg-w2v for each sentence/review is stored in this list
    print(len(xtrainEassyArray))
    for sentence in tqdm(xtrainEassyArray): # 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_vector_eassy.append(vector)

#     print(len(avg_w2v_vector_eassy))
#     print(len(avg_w2v_vectors[0]))
    return avg_w2v_vector_eassy
```

In [70]:

```
# avg_w2v_vector_eassy = []; # the avg-w2v for each sentence/review is stored in this list
# for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
#     vector = np.zeros(300) # as word vectors are of zero length
#     cnt_words =0; # num of words with a valid vector in the sentence/review
#     for word in sentence.split(): # for each word in a review/sentence
#         if word in glove_words:
#             vector += model[word]
#             cnt_words += 1
#     if cnt_words != 0:
#         vector /= cnt_words
#     avg_w2v_vector_eassy.append(vector)
```

```
# print(len(avg_w2v_vector_eassy))
# print(len(avg_w2v_vector_eassy[0]))
# print(np.array(avg_w2v_vector_eassy).shape)
```

- Now Combining all of the feature.. to put into the model

In [71]:

```
x_tr_w2v_eassy = makew2vList(X_train['essay'].values)
print(len(x_tr_w2v_eassy))
```

```
  1%|          | 181/13467 [00:00<00:07, 1800.49it/s]
```

```
13467
```

```
100%|██████████| 13467/13467 [00:07<00:00, 1793.69it/s]
```

```
13467
```

In [72]:

```
x_cv_w2v_eassy = makew2vList(X_cv['essay'].values)
print(len(x_cv_w2v_eassy))
print(np.array(x_cv_w2v_eassy).shape)
```

```
  2%|          | 156/6633 [00:00<00:04, 1559.93it/s]
```

```
6633
```

```
100%|██████████| 6633/6633 [00:03<00:00, 1919.50it/s]
```

```
6633
(6633, 300)
```

In [73]:

```
x_tes_w2v_eassy = makew2vList(X_test['essay'].values)
print(len(x_tes_w2v_eassy))
```

```
  2%|          | 166/9900 [00:00<00:05, 1648.10it/s]
```

```
9900
```

```
100%|██████████| 9900/9900 [00:05<00:00, 1765.46it/s]
```

```
9900
```

- For Titles

In [74]:

```
x_tr_w2v_title = makew2vList(X_train['project_title'].values)
print(len(x_cv_w2v_eassy))
```

```
 15%|▌         | 2059/13467 [00:00<00:00, 20588.15it/s]
```

```
13467
```

```
100%|██████████| 13467/13467 [00:00<00:00, 22772.05it/s]
```

```
6633
```

In [75]:

```
x_cv_w2v_title = makew2vList(X_cv['project_title'].values)
print(len(x_cv_w2v_eassy))
```

```
 36%|███▌      | 2368/6633 [00:00<00:00, 23674.60it/s]
```

```
6633
```

```
100%|██████████| 6633/6633 [00:00<00:00, 24482.61it/s]
```

```
6633
```

In [76]:

```
x_tes_w2v_title = makew2vList(X_test['project_title'].values)
print(len(x_cv_w2v_eassy))
```

```
 23%|██▎       | 2290/9900 [00:00<00:00, 22899.75it/s]
```

```
9900
```

```
100%|██████████| 9900/9900 [00:00<00:00, 20957.00it/s]
```

```
6633
```

In [77]:

```
X_cv_price_norm.shape
```

Out[77]:

```
(6633, 1)
```

In [78]:

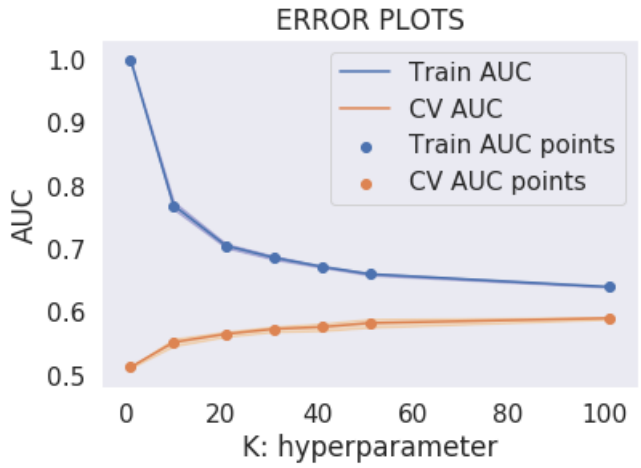
```
# combining all features

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

# from scipy.sparse import hstack
X_tr = hstack(
    (np.array(x_tr_w2v_eassy),
     np.array(x_tr_w2v_title),
     X_train_state_ohe,
     X_train_teacher_ohe,
     X_train_sub_cat,
     X_train_prPos_norm,
     X_train_grade_ohe,
     X_train_quantity_norm,
     X_train_price_norm)).tocsr()

#np.array()

X_cv_eassy_w2v = np.array(x_cv_w2v_eassy)
X_cv_title_w2v = np.array(x_cv_w2v_title)
```

```
# print(X_cv_eassy_w2v.reshape(1106, 1))

X_cr = hstack(
    (X_cv_eassy_w2v,
     X_cv_title_w2v,
     X_cv_state_ohe,
     X_cv_teacher_ohe,
     X_cv_sub_cat,
     X_cv_prPos_norm,
     X_cv_grade_ohe,
     X_cv_quantity_norm,
     X_cv_price_norm)).tocsr()



X_te = hstack(
    (np.array(x_tes_w2v_eassy),
     np.array(x_tes_w2v_title),
     X_test_state_ohe,
     X_test_teacher_ohe,
     X_test_sub_cat,
     X_test_prPos_norm,
     X_test_grade_ohe,
     X_test_quantity_norm,
     X_test_price_norm)).tocsr()



print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(13467, 692) (13467,)
(6633, 692) (6633,)
(9900, 692) (9900,)
====================================================================================================
```

In [79]:

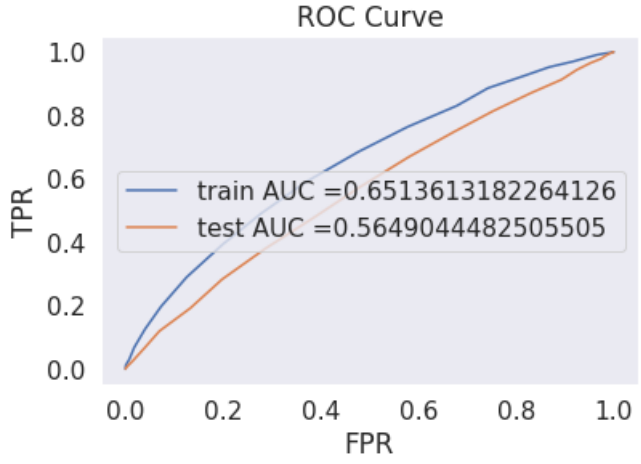
```
neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 10, 21, 31, 41, 51, 101]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')



plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



- As per the error Plot The minimum gap represents the best k value.
- And in this plot 61 is having less space between Train and Test Auc Graphs

In [80]:

```
best_k_m3 = 61


neigh = KNeighborsClassifier(n_neighbors=best_k_m1)
neigh.fit(X_tr, 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 = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

m3_Auc = str(auc(train_fpr, train_tpr))

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



- Train AUC values is 0.65
- Test AUC value is 0.56
- Comparing to Raw model this model is not performing in the test data sets

- Comparing to Bow model this model is not performing in the test data sets.

In [81]:

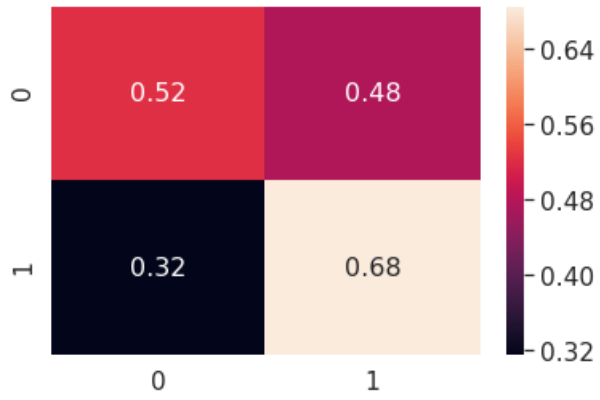
```
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.249464369904656 for threshold 0.845
[[1085  989]
 [3591 7802]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2496 for threshold 0.859
[[ 793  732]
 [3571 4804]]
```

In [82]:

```
# Normalize the confusion matrix by row (i.e by the number of samples
# in each class)
# https://scikit-learn.org/0.16/auto_examples/model_selection/plot_confusion_matrix.html

cm_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))

plotConfusionMatrix(cm_train)
```

```
the maximum value of tpr*(1-fpr) 0.249464369904656 for threshold 0.845
```

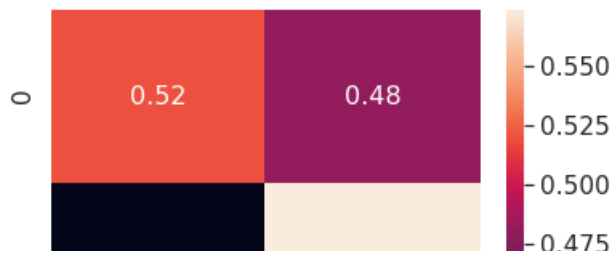


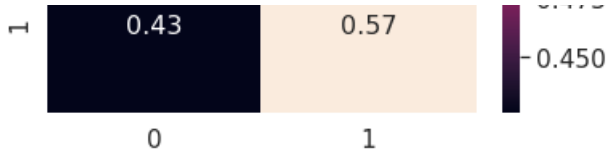
- TP = 0.52
- TN = 0.48
- FP = 0.32
- FN = 0.68
- FN and TP has higher value from other which shows model is good for Train sets.

In [83]:

```
cm_test = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))

plotConfusionMatrix(cm_test)
```

```
the maximum value of tpr*(1-fpr) 0.2496 for threshold 0.859
```





- TP = 0.52
- TN = 0.48
- FP = 0.43
- FN = 0.57
- FN and TP has both heigher value which is a good result comparing to Bow output.

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

Applying KNN brute force on TFIDF W2V, SET 4

In [84]:

```
# 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 [85]:

```
# Similarly you can vectorize for title also

def maketfidfw2vList(xEassyArray):
    tfidf_w2v_vectors_project_title = []; # the avg-w2v for each sentence/review is stored in this
list
    for sentence in tqdm(xEassyArray): # 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 tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors_project_title.append(vector)

#     print(len(tfidf_w2v_vectors_project_title))
```

```
#     print(len(tfidf_w2v_vectors_project_title))
#     print(len(tfidf_w2v_vectors_project_title[0]))
    return tfidf_w2v_vectors_project_title
```

In [86]:

```
x_tr_tfidfw2v_eassy = maketfidfw2vList(X_train['essay'].values)
print(len(x_tr_tfidfw2v_eassy))
```

```
100%|██████████| 13467/13467 [00:52<00:00, 257.74it/s]
```

```
13467
```

In [87]:

```
x_cv_tfidfw2v_eassy = maketfidfw2vList(X_cv['essay'].values)
print(len(x_cv_tfidfw2v_eassy))
```

```
100%|██████████| 6633/6633 [00:26<00:00, 253.21it/s]
```

```
6633
```

In [88]:

```
x_tes_tfidfw2v_eassy = makew2vList(X_test['essay'].values)
print(len(x_tes_tfidfw2v_eassy))
```

```
  1%|          | 141/9900 [00:00<00:06, 1406.01it/s]
```

```
9900
```

```
100%|██████████| 9900/9900 [00:05<00:00, 1675.09it/s]
```

```
9900
```

- tfidf weighted W2v for titles

In [89]:

```
x_tr_tfidfw2v_title = maketfidfw2vList(X_train['project_title'].values)
print(len(x_tr_tfidfw2v_title))
```

```
100%|██████████| 13467/13467 [00:01<00:00, 11904.57it/s]
```

```
13467
```

In [90]:

```
x_cv_tfidfw2v_title = maketfidfw2vList(X_cv['project_title'].values)
print(len(x_cv_tfidfw2v_title))
```

```
100%|██████████| 6633/6633 [00:00<00:00, 12322.49it/s]
```

```
6633
```

In [91]:

```
x_tes_tfidfw2v_title = makew2vList(X_test['project_title'].values)
print(len(x_tes_tfidfw2v_title))
```

```
 26%|██▌       | 2595/9900 [00:00<00:00, 25946.44it/s]
```

```
9900
```

```
100%|██████████| 9900/9900 [00:00<00:00, 25201.61it/s]
```

```
9900
```

- Combining All Vecotrizers to train cv and test to put into model

In [92]:

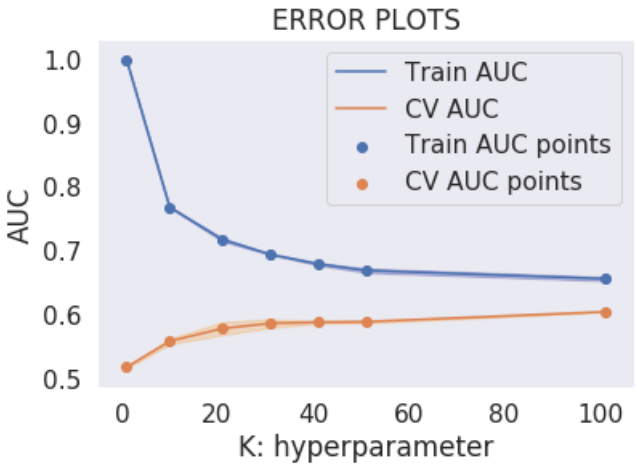
```
# combining all features

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

# from scipy.sparse import hstack
X_tr = hstack(
    (np.array(x_tr_tfidfw2v_eassy),
     np.array(x_tr_tfidfw2v_title),
     X_train_state_ohe,
     X_train_teacher_ohe,
     X_train_sub_cat,
     X_train_prPos_norm,
     X_train_grade_ohe,
     X_train_quantity_norm,
     X_train_price_norm)).tocsr()

#np.array()

X_cv_eassy_tfidfw2v = np.array(x_cv_tfidfw2v_eassy)
X_cv_title_tfidfw2v = np.array(x_cv_tfidfw2v_title)

# print(X_cv_eassy_w2v.reshape(1106, 1))

X_cr = hstack(
    (X_cv_eassy_tfidfw2v,
     X_cv_title_tfidfw2v,
     X_cv_state_ohe,
     X_cv_teacher_ohe,
     X_cv_sub_cat,
     X_cv_prPos_norm,
     X_cv_grade_ohe,
     X_cv_quantity_norm,
     X_cv_price_norm)).tocsr()



X_te = hstack(
    (np.array(x_tes_tfidfw2v_eassy),
     np.array(x_tes_tfidfw2v_title),
     X_test_state_ohe,
     X_test_teacher_ohe,
     X_test_sub_cat,
     X_test_prPos_norm,
     X_test_grade_ohe,
     X_test_quantity_norm,
     X_test_price_norm)).tocsr()
```

```
print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(13467, 692) (13467,)
(6633, 692) (6633,)
(9900, 692) (9900,)
====================================================================================================
```

In [93]:

```
neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 10, 21, 31, 41, 51, 101]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

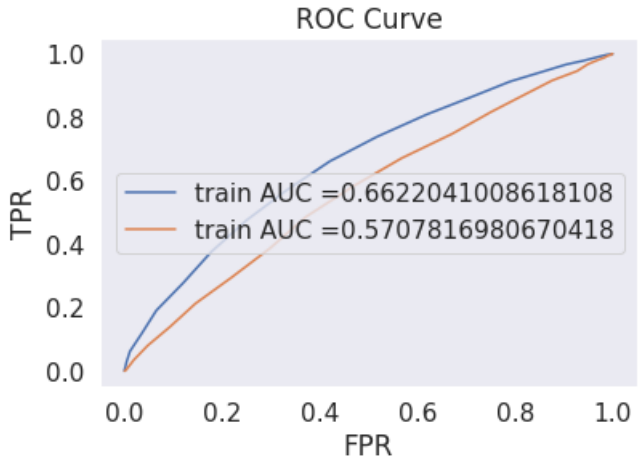


- As per the error Plot The minimum gap represents the best k value.
- And in this plot 61 is having less space between Train and Test Auc Graphs

In [94]:

```
best_k_m4 = 61
neigh = KNeighborsClassifier(n_neighbors=best_k_m1)
neigh.fit(X_tr, 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 = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

m4_Auc = str(auc(train_fpr, train_tpr))

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



- Train AUC values is 0.66
- Test AUC value is 0.57
- Comparing to Bow model this model is not performing in the test data sets.

In [95]:

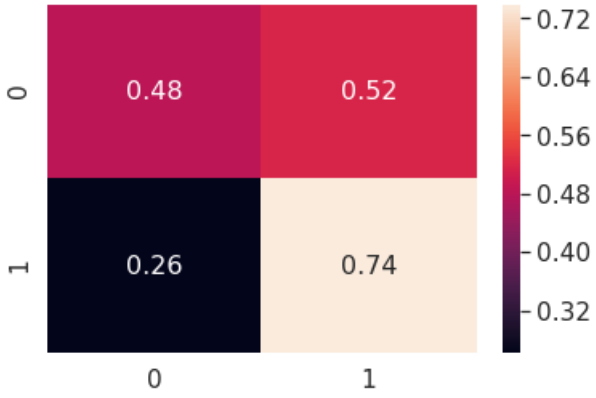
```
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24969870807136899 for threshold 0.831
[[1001 1073]
 [2975 8418]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24902983069067458 for threshold 0.845
[[ 810  715]
 [3493 4882]]
```

In [96]:

```
# Normalize the confusion matrix by row (i.e by the number of samples
# in each class)
# https://scikit-learn.org/0.16/auto_examples/model_selection/plot_confusion_matrix.html

cm_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))

plotConfusionMatrix(cm_train)
```

```
the maximum value of tpr*(1-fpr) 0.24969870807136899 for threshold 0.831
```

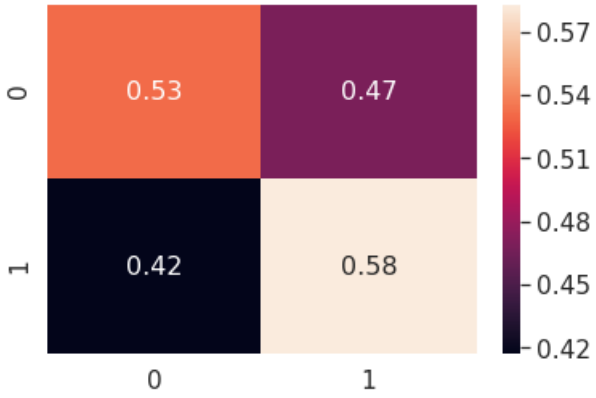


- TP = 0.48
- TN = 0.52
- FP = 0.26
- FN = 0.74
- FN has Hiher value then other values which is good but with that.. TP should have higher value too.

In [97]:

```
cm_test = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))

plotConfusionMatrix(cm_test)
```

```
the maximum value of tpr*(1-fpr) 0.24902983069067458 for threshold 0.845
```



- TP = 0.53
- TN = 0.47
- FP = 0.42
- FN = 0.58
- Both TP and FN has light Color .. it means test data set got more good results then train one.

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
```

Applying KNN brute force on TFIDF, SET 2 With Best 20 Features

In [98]:

```
# Making DataMatrix with eassay and title with TFIDF

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizerTfIDF = TfidfVectorizer(min_df=10)
vectorizerTfIDF.fit(X_train['essay'].values)

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

print("Shape of matrix Shape of matrix TFIDFg ",X_train_essay_tfidf.shape)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*50)
```

```
Shape of matrix Shape of matrix TFIDFg  (13467, 7125)
After vectorizations
(13467, 7125) (13467,)
(6633, 7125) (6633,)
(9900, 7125) (9900,)
==================================================
```

In [ ]:

```
```

In [99]:

```
# Similarly you can vectorize for title also with TFIDF
vectorizerTfIDF = TfidfVectorizer(min_df=10)
vectorizerTfIDF.fit(X_train['project_title'].values)

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


print("Shape of matrix TFIDF ",X_train_title_tfidf.shape)


print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
print("="*50)
```

```
Shape of matrix TFIDF  (13467, 851)
After vectorizations
(13467, 851) (13467,)
(6633, 851) (6633,)
(9900, 851) (9900,)
==================================================
```

In [100]:

```
```

```
X_cv_essay_tfidf.shape
```

Out[100]:

```
(6633, 7125)
```

In [101]:

```
X_train_essay_tfidf.shape
```

Out[101]:

```
(13467, 7125)
```

In [102]:

```
X_cv_price_norm.shape
```

Out[102]:

```
(6633, 1)
```

In [103]:

```
# combining all features

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack
X_tr = hstack(
    (X_train_essay_tfidf,
     X_train_title_tfidf,
     X_train_state_ohe,
     X_train_teacher_ohe,
     X_train_sub_cat,
     X_train_prPos_norm,
     X_train_grade_ohe,
     X_train_quantity_norm,
     X_train_price_norm)).tocsr()


X_cr = hstack(
    (X_cv_essay_tfidf,
     X_cv_title_tfidf,
     X_cv_state_ohe,
     X_cv_teacher_ohe,
     X_cv_sub_cat,
     X_cv_prPos_norm,
     X_cv_grade_ohe,
     X_cv_quantity_norm,
     X_cv_price_norm)).tocsr()




X_te = hstack(
    (X_test_essay_tfidf,
     X_test_title_tfidf,
     X_test_state_ohe,
     X_test_teacher_ohe,
     X_test_sub_cat,
     X_test_prPos_norm,
     X_test_grade_ohe,
     X_test_quantity_norm,
     X_test_price_norm)).tocsr()




print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(13467, 8068) (13467,)
(6633, 8068) (6633,)
(9900, 8068) (9900,)
====================================================================================================
```

In [105]:

```
# Selecting Best 20 features for this set of Train Data
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, chi2

selectKBest = SelectKBest(chi2, k=2000).fit(X_tr, y_train)


X_new_train = selectKBest.transform(X_tr)
X_new_cv = selectKBest.transform(X_cr)
X_new_test = selectKBest.transform(X_te)


print("Shape of Best 20 feature matrix")
print(X_new_train.shape, y_train.shape)
print(X_new_cv.shape, y_cv.shape)
print(X_new_test.shape, y_test.shape)
print("="*100)
```

```
Shape of Best 20 feature matrix
(13467, 2000) (13467,)
(6633, 2000) (6633,)
(9900, 2000) (9900,)
====================================================================================================
```

In [106]:

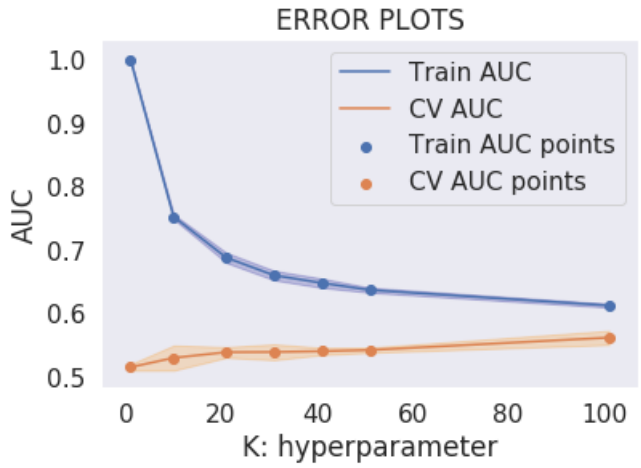
```
# # New best 20 feature for x test too
# X_new_test = SelectKBest(chi2, k=20).fit_transform(X_te, y_test)
# X_new_test.shape
```

Finding Best K with Top 20 features

In [107]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# from sklearn.model_selection import GridSearchCV
# from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 10, 21, 31, 41, 51, 101]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_new_train, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')
```

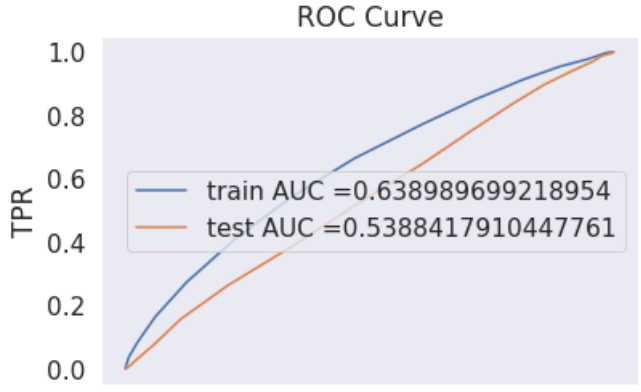
```
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



- As per the error Plot The minimum gap represents the best k value.
- And in this plot 31 is having less space between Train and Test Auc Graphs

In [114]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
# from sklearn.metrics import roc_curve, auc

best_k_m2 = 51
neigh = KNeighborsClassifier(n_neighbors=best_k_m2)
neigh.fit(X_new_train, 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 = batch_predict(neigh, X_new_train)
y_test_pred = batch_predict(neigh, X_new_test)

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

m2_Auc = str(auc(train_fpr, train_tpr))

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



- Train AUC values is 0.63
- Test AUC value is 0.47
- After Selecting Best 20 feature too Auc Doesn't show any good result. Best guess would be for this behiviour other feature also matter too.

In [115]:

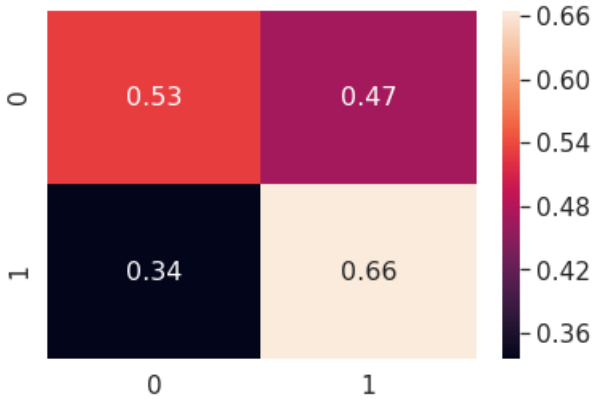
```
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24904776871938838 for threshold 0.843
[[1101  973]
 [3822 7571]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24957334049986563 for threshold 0.863
[[ 794  731]
 [3977 4398]]
```

In [116]:

```
cm_train = confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))

plotConfusionMatrix(cm_train)
```

```
the maximum value of tpr*(1-fpr) 0.24904776871938838 for threshold 0.843
```

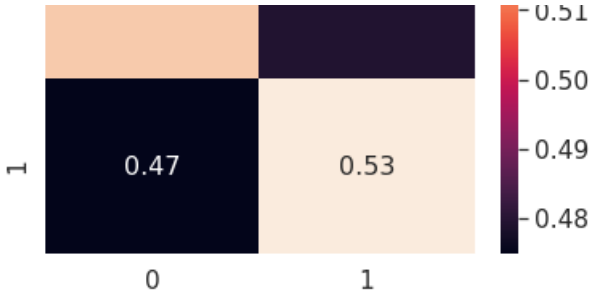


- TP = 0.53
- TN = 0.47
- FP = 0.34
- FN = 0.66
- FN has Hiher value then other values which is good & TP should have higher value too.

In [117]:

```
cm_test = confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))

plotConfusionMatrix(cm_test)
```

```
the maximum value of tpr*(1-fpr) 0.24957334049986563 for threshold 0.863
```



- TP = 0.52
- TN = 0.48
- FP = 0.47
- FN = 0.53
- TP has Heigher value . But Fn Should have heigher value too.

Final Table

In [118]:

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

In [119]:

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

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]


x.add_row(["BOW", "Brute", best_k_m1, m1_Auc])
x.add_row([" ", " ", " ", " "])
x.add_row(["TFIDF", "Brute", best_k_m2, m2_Auc])
x.add_row([" ", " ", " ", " "])
x.add_row(["W2V", "Brute", best_k_m3, m3_Auc])
x.add_row([" ", " ", " ", " "])
x.add_row(["TFIDFW2V", "Brute", best_k_m4, m4_Auc])


print(x)
```

```
+------------+-------+-----------------+--------------------+
| Vectorizer | Model | Hyper Parameter |        AUC         |
+------------+-------+-----------------+--------------------+
|    BOW     | Brute |        71       | 0.6477179688995112 |
|            |       |                 |                    |
|   TFIDF    | Brute |        51       | 0.638989699218954  |
|            |       |                 |                    |
|    W2V     | Brute |        61       | 0.6513613182264126 |
|            |       |                 |                    |
|  TFIDFW2V  | Brute |        61       | 0.6622041008618108 |
+------------+-------+-----------------+--------------------+
```

In [ ]:

```

```

Observations

- 1. We Have applied Knn For this Dataset.
- 1. We have taken 30k Data points for this computation
- 1. TFIDF weighted Model (Set 4) is doing good.
- 1. TFIDF vectorized Data set is not performing Good.
- 1. Finding top 2k features didn't shown any best performance.
- 1. But in Confusion matrix TFIDF graphs shows better result then Bow model

- 1. But in Confusion matrix TFIDF graphs shows better result then Bow model.

In [ ]:

In [ ]:

In [ ]:

In [ ]: