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	Desc
`project_id`	A unique identifier for the proposed project. **Example:** `p0`
`project_title`	Title of the project. **Exa** • `Art Will Make You H` • `First Grad`
`project_grade_category`	Grade level of students for which the project is targeted. One of the fo enumerated v • `Grades P` • `Grade` • `Grade` • `Grades`
`project_subject_categories`	One or more (comma-separated) subject categories for the project fr following enumerated list of v • `Applied Lea` • `Care & H` • `Health & S` • `History & C` • `Literacy & Lan` • `Math & Sc` • `Music & The` • `Special` • `W` **Exar** • `Music & The` • `Literacy & Language, Math & Sc`

Feature	Desc
`school_state`	State where school is located (Two-letter U.S. posta (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_c **Exampl**
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the p **Exan** • Lit • Literature & Writing, Social Sci
`project_resource_summary`	An explanation of the resources needed for the project. **Exa** • My students need hands on literacy materials to ma sensory needs!<
`project_essay_1`	First application
`project_essay_2`	Second application
`project_essay_3`	Third application
`project_essay_4`	Fourth application
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** 2016-0 12:43:5
`teacher_id`	A unique identifier for the teacher of the proposed project. **Exa** bdf8baa8fedef6bfeec7ae4ff1c
`teacher_prefix`	Teacher's title. One of the following enumerated v • • • • • • Tea
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same te **Exam**

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

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

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

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv', nrows=25000)
resource_data = pd.read_csv('resources.csv')
```

In [3]:

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

```
Number of data points in train data (25000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 's
chool_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

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


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


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


project_data.head(2)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2 0 00:5
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2 0 02:0

In [5]:

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

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

Out[5]:

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

1.2 preprocessing of project_subject_categories

In [6]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

# 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", "
        if 'The' in j.split(): # this will split each of the catogory based on space "Math
            j=j.replace('The','') # if we have the words "The" we are going to replace it w
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.3 preprocessing of project_subject_subcategories

In [7]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

# 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", "
        if 'The' in j.split(): # this will split each of the catogory based on space "Math
            j=j.replace('The','') # if we have the words "The" we are going to replace it w
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        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]))
```

Preprocessing of project_grade_category

In [8]:

```
#reference link: https://stackoverflow.com/questions/28986489/python-pandas-how-to-replace-

project_data['project_grade_category'] = project_data['project_grade_category'].str.replace
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace
```

1.3 Text preprocessing

In [9]:

```
# 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 [10]:

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

Out[10]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2 0 00:5
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2 0 02:0

In [11]:

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

In [12]:

```
# 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)
```

```
I recently read an article about giving students a choice about how they lea
rn. We already set goals; why not let them choose where to sit, and give the
m options of what to sit on?I teach at a low-income (Title 1) school. Every
year, I have a class with a range of abilities, yet they are all the same ag
e. They learn differently, and they have different interests. Some have ADH
D, and some are fast learners. Yet they are eager and active learners that w
ant and need to be able to move around the room, yet have a place that they
can be comfortable to complete their work.We need a classroom rug that we ca
n use as a class for reading time, and students can use during other learnin
g times. I have also requested four Kore Kids wobble chairs and four Back Ja
ck padded portable chairs so that students can still move during whole group
lessons without disrupting the class. Having these areas will provide these
little ones with a way to wiggle while working.Benjamin Franklin once said,
\"Tell me and I forget, teach me and I may remember, involve me and I lear
n.\" I want these children to be involved in their learning by having a choi
ce on where to sit and how to learn, all by giving them options for comforta
ble flexible seating.
==================================================
Who remembers middle school- the chaos and panic as you rocketed through pub
erty (or worse- hadn't started it yet)? In my classroom, we try to immerse o
urselves in literature, our own writing, or heated debate. Though sometimes
we're (intentionally) chaotic, our class is a respite from the madness.My st
udents want to be heard. They constantly want to read out loud, tell me thei
r opinions, and spill out ALL their ideas in one run-on sentence. My student
s are an extremely diverse group of kids, and they're proud of who they are.
Each kid has an astonishing backstory and they aren't afraid to share it.  M
y school is in an area that typically has families of low SES. Sometimes I n
eed to give my students a pencil, a binder, or maybe even a shirt. The great
thing about these kids is that they don't let that bring them down. Every si
ngle child I teach is opinionated and in the process of learning how to shap
e and share those opinions. They deserve every opportunity I can give them.M
y kids just need to hold books in their hands. They've expressed the desire
to perform plays (they love attention), because they zone out a little when
just one of us reads a novel aloud.  Our school doesn't have any class sets
of plays. I'd take any play, but we only have novels. I want literature that
can involve as many kids in the experience as possible. With a play, I can h
ave 6-8 readers at once without the dreaded 'popcorn reading' of my youth. O
ur district, in order to support differentiation, has a limit on how many bo
oks can be ordered through the school. Because of this, our library has a dw
indling number of class sets. While I understand that and can work around i
t, my students don't get why I can't just find them 30 copies of a play to r
ead. This is a direct request from my kids: \"We just need some books!\"Over
half of my school (and an even larger percentage of my students) are new to
English as a spoken language. Conquering words penned by Shakespeare can sha
pe their understanding of English as a language and increase their confidenc
e. This will help them increase their speaking and listening skills, but mor
e importantly- push them out of their comfort zones. Children who can gather
the courage read a line in a play (when they USED to be afraid to speak in c
```

lass) can do anything.
==
My students have become artists! This school year they dove head first into a new art program developed for student in special day programs who are clas sified as Emotionally Disturbed. These kids tend to work better in small gr oups and have projects that allow them to get up, move around, listen to mus ic, work with their hands and freely be themselves. Often times, out in reg ular education, the classrooms have too many kids, the rigor of work can be overwhelming and the kids sometimes don't know how to work through their emo tions. To help them out, we created an art program focusing on 3D Art and p ottery that allows them to access their creative juices, but do so in an env ironment that feels safe and welcoming.\r\nAs the art program has been such a success, the school has asked me to teach a 2nd class of art to students i n our special day classes who are classified as intellectually disabled. I am so excited to be able to expand our pottery project and allow other stude nts to experience art on a daily basis.This school year we launched our Pott ery Project and introduced pottery to the students each week on Thursdays an d Fridays. The students had so much fun. They learned how to throw on the wheels and they learned how to hand build with clay. The rest of the week they spent doing 2D and 3D art projects, which included making 3D conversati onal heart, 3D words, trees created out of wire and paper mache, masks, boxe s and used wood burning tools.\r\nNext school year multiple students in the SDC program on our campus will be able to take an art class designed specifi cally with the in mind, focusing in on their own personal learning styles. It is so very exciting to create an art program that will challenge the stud ents to empower art.nannan
==
My fifth grade classroom is one of 6 on a campus of over 1400 students (K- 6). We are the largest school in our district. Our population is extremely d iverse, with 24 languages spoken and 188 students living below the poverty l ine. Like any classroom, my students have a wide variety of needs. \r\nOf th e 28 students in my class, one is full-inclusion (Down's Syndrome), two have individual education plans (IEPs), three have special modification plans (50 4s for ADHD), two are English language learners, and five are identified as gifted and talented. This group of students requires diverse instructional strategies that allow me to be up and moving among them. Proximity is key to keeping them on task and engaged.I currently use a document camera during di rect instruction. It is stationary on my desk, which ties me to that locatio n. The iPad Pro allows me to be mobile during direct instruction. I can walk anywhere in the classroom, while still projecting the document/image that is on my iPad onto the large screen. \r\nUsing the iPad Pro with the iPen, in c onjunction with the Notability app, will allow me to be interactive and mobi le with my lessons, projecting them on the big screen and using the pen to w rite answers/notes. I can then email that exact document with written notes to an absent student or parent needing extra support. It is saved in my clou d, and it allows me to be completely paperless.nannan
==

In [13]:

```
# 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 [14]:

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

```
My fifth grade classroom is one of 6 on a campus of over 1400 students (K-
6). We are the largest school in our district. Our population is extremely d
iverse, with 24 languages spoken and 188 students living below the poverty l
ine. Like any classroom, my students have a wide variety of needs. \r\nOf th
e 28 students in my class, one is full-inclusion (Down is Syndrome), two hav
e individual education plans (IEPs), three have special modification plans
(504s for ADHD ), two are English language learners, and five are identified
as gifted and talented.  This group of students requires diverse instruction
al strategies that allow me to be up and moving among them. Proximity is key
to keeping them on task and engaged.I currently use a document camera during
direct instruction. It is stationary on my desk, which ties me to that locat
ion. The iPad Pro allows me to be mobile during direct instruction. I can wa
lk anywhere in the classroom, while still projecting the document/image that
is on my iPad onto the large screen. \r\nUsing the iPad Pro with the iPen, i
n conjunction with the Notability app, will allow me to be interactive and m
obile with my lessons, projecting them on the big screen and using the pen t
o write answers/notes. I can then email that exact document with written not
es to an absent student or parent needing extra support. It is saved in my c
loud, and it allows me to be completely paperless.nannan
==================================================
```

In [15]:

```
# \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)
```

```
My fifth grade classroom is one of 6 on a campus of over 1400 students (K-
6). We are the largest school in our district. Our population is extremely d
iverse, with 24 languages spoken and 188 students living below the poverty l
ine. Like any classroom, my students have a wide variety of needs.   Of the
28 students in my class, one is full-inclusion (Down is Syndrome), two have
individual education plans (IEPs), three have special modification plans (50
4s for ADHD ), two are English language learners, and five are identified as
gifted and talented.  This group of students requires diverse instructional
strategies that allow me to be up and moving among them. Proximity is key to
keeping them on task and engaged.I currently use a document camera during di
rect instruction. It is stationary on my desk, which ties me to that locatio
n. The iPad Pro allows me to be mobile during direct instruction. I can walk
anywhere in the classroom, while still projecting the document/image that is
on my iPad onto the large screen.   Using the iPad Pro with the iPen, in con
junction with the Notability app, will allow me to be interactive and mobile
with my lessons, projecting them on the big screen and using the pen to writ
e answers/notes. I can then email that exact document with written notes to
an absent student or parent needing extra support. It is saved in my cloud,
and it allows me to be completely paperless.nannan
```

In [16]:

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

```
My fifth grade classroom is one of 6 on a campus of over 1400 students K 6 W
e are the largest school in our district Our population is extremely diverse
with 24 languages spoken and 188 students living below the poverty line Like
any classroom my students have a wide variety of needs Of the 28 students in
my class one is full inclusion Down is Syndrome two have individual educatio
n plans IEPs three have special modification plans 504s for ADHD two are Eng
lish language learners and five are identified as gifted and talented This g
roup of students requires diverse instructional strategies that allow me to
be up and moving among them Proximity is key to keeping them on task and eng
aged I currently use a document camera during direct instruction It is stati
onary on my desk which ties me to that location The iPad Pro allows me to be
mobile during direct instruction I can walk anywhere in the classroom while
still projecting the document image that is on my iPad onto the large screen
Using the iPad Pro with the iPen in conjunction with the Notability app will
allow me to be interactive and mobile with my lessons projecting them on the
big screen and using the pen to write answers notes I can then email that ex
act document with written notes to an absent student or parent needing extra
support It is saved in my cloud and it allows me to be completely paperless
nannan
```

In [17]:

```
# 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'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"]
```

In [18]:

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

```
100%|████████████████████████████████████████████████████████████████
████| 25000/25000 [00:26<00:00, 942.33it/s]
```

In [19]:

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

Out[19]:

```
'fifth grade classroom one 6 campus 1400 students k 6 largest school distric
t population extremely diverse 24 languages spoken 188 students living pover
ty line like classroom students wide variety needs 28 students class one ful
l inclusion syndrome two individual education plans ieps three special modif
ication plans 504s adhd two english language learners five identified gifted
talented group students requires diverse instructional strategies allow movi
ng among proximity key keeping task engaged currently use document camera di
rect instruction stationary desk ties location ipad pro allows mobile direct
instruction walk anywhere classroom still projecting document image ipad ont
o large screen using ipad pro ipen conjunction notability app allow interact
ive mobile lessons projecting big screen using pen write answers notes email
exact document written notes absent student parent needing extra support sav
ed cloud allows completely paperless nannan'
```

1.4 Preprocessing of `project_title`

In [20]:

```
# similarly you can preprocess the titles also
```

In [21]:

```
from tqdm import tqdm
preprocessed_titles = []
for sentance in tqdm(project_data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\n', ' ')
    sent = sent.replace('\\"', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████
██| 25000/25000 [00:01<00:00, 19039.76it/s]
```

1.5 Preparing data for models

In [22]:

```
project_data.columns
```

Out[22]:

```
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_approve
d',
       'clean_categories', 'clean_subcategories', 'essay'],
      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/ (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)

In [23]:

```
# 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, bina
categories_one_hot = vectorizer.fit_transform(project_data['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning',
'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (25000, 9)
```

In [24]:

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement',
'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducat
ion', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'Characte
rEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_
Geography', 'Health_LifeScience', 'ESL', 'EarlyDevelopment', 'Gym_Fitness',
'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (25000, 30)
```

In [25]:

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

In [26]:

```
#school state
#Using CountVectorizer to convert values into one hot encoded
vectorizer = CountVectorizer(lowercase=False , binary=True)
vectorizer.fit(project_data['school_state'].values)
print(vectorizer.get_feature_names())

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

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'I
A', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO',
'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY', 'OH', 'OK', 'O
R', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV',
'WY']
Shape of matrix after one hot encoding (25000, 51)
```

In [27]:

```
#project_grade_category
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(project_data['project_grade_category'].values.astype('U'))
print(vectorizer.get_feature_names())


project_grade_category_one_hot = vectorizer.fit_transform(project_data['project_grade_categ
print('Shape of matrix of one hot encoding', project_grade_category_one_hot.shape)
```

```
['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
Shape of matrix of one hot encoding (25000, 4)
```

In [28]:

```
#teacher_prefix
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(project_data['teacher_prefix'].values.astype('U'))
#While running this i got an error:np.nan is an invalid document, expected byte or unicode
#I fixed it by using stackoverflow.com
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np
print(vectorizer.get_feature_names())


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

```
['Mr', 'Mrs', 'Ms', 'Teacher', 'nan']
Shape of matrix of one hot encoding (25000, 5)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [29]:

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

```
Shape of matrix after one hot encodig  (25000, 9218)
```

In [30]:

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

In [31]:

```
# Similarly you can vectorize for title also
vectorizer = CountVectorizer(min_df=3) #here i used min_df=3 because titles are much smalle
bow_titles = vectorizer.fit_transform(preprocessed_titles)
print('Shape of matrix after one hot encoding', text_bow.shape)
```

```
Shape of matrix after one hot encoding (25000, 9218)
```

1.5.2.2 TFIDF vectorizer

In [32]:

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

```
Shape of matrix after one hot encodig  (25000, 9218)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [33]:

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

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

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

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

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

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

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

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


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickl

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


'''
```

Out[33]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4
```

```
084039\ndef (https://stackoverflow.com/a/38230349/4084039\ndef) loadGloveMod
el(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFil
e,\'r\', encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n
splitLine = line.split()\n        word = splitLine[0]\n        embedding = n
p.array([float(val) for val in splitLine[1:]])\n        model[word] = embedd
ing\n    print ("Done.",len(model)," words loaded!")\n    return model\nmode
l = loadGloveModel(\'glove.42B.300d.txt\')\n\n# ==========================
\nOutput:\n    \nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\nDone.
 1917495  words loaded!\n\n# ============================\n\nwords = []\nfor
i in preproced_texts:\n    words.extend(i.split(\' \'))\n\nfor i in preproce
d_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the cou
pus", len(words))\nwords = set(words)\nprint("the unique words in the coupu
s", len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprin
t("The number of words that are present in both glove vectors and our coupu
s",       len(inter_words),"(",np.round(len(inter_words)/len(words)*100,
3),"%)")\n\nwords_courpus = {}\nwords_glove = set(model.keys())\nfor i in wo
rds:\n    if i in words_glove:\n        words_courpus[i] = model[i]\nprint
("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pi
ckle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-
load-variables-in-python/\n\nimport (http://www.jessicayung.com/how-to-use-p
ickle-to-save-and-load-variables-in-python/\n\nimport) pickle\nwith open(\'g
love_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [34]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickl
# 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 [35]:

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

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

```
100%|████████████████████████████████████████████████████████████████
███| 25000/25000 [00:13<00:00, 1916.59it/s]

25000
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [36]:

```
# 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 [37]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_essays = []; # the avg-w2v for each sentence/review is stored in this lis
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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            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_essays.append(vector)

print(len(tfidf_w2v_vectors_essays))
print(len(tfidf_w2v_vectors_essays[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████| 25000/25000 [01:29<00:00, 278.63it/s]

25000
300
```

In [38]:

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

In [39]:

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

In [40]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_titles = []; # the avg-w2v for each sentence/review is stored in this lis
for sentence in tqdm(preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            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_titles.append(vector)

print(len(tfidf_w2v_vectors_titles))
print(len(tfidf_w2v_vectors_titles[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██| 25000/25000 [00:01<00:00, 16437.26it/s]

25000
300
```

1.5.3 Vectorizing Numerical features

In [41]:

```
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 [42]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.prepro
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.
# 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 standar
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 : 298.7497976, Standard deviation : 374.8142029086185
```

In [43]:

```
price_standardized
```

Out[43]:

```
array([[ 0.48634817],
       [ 0.00330885],
       [-0.66110034],
       ...,
       [ 0.20599594],
       [-0.31242092],
       [-0.08051935]])
```

1.5.4 Merging all the above features

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

In [44]:

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

```
(25000, 9)
(25000, 30)
(25000, 9218)
(25000, 1)
```

In [45]:

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

Out[45]:

```
(25000, 9258)
```

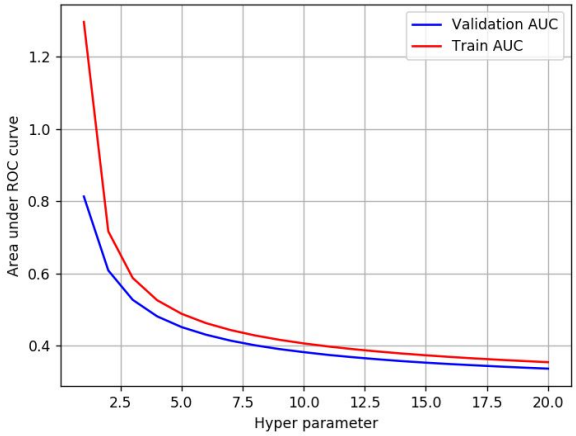
Assignment 3: Apply KNN

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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-

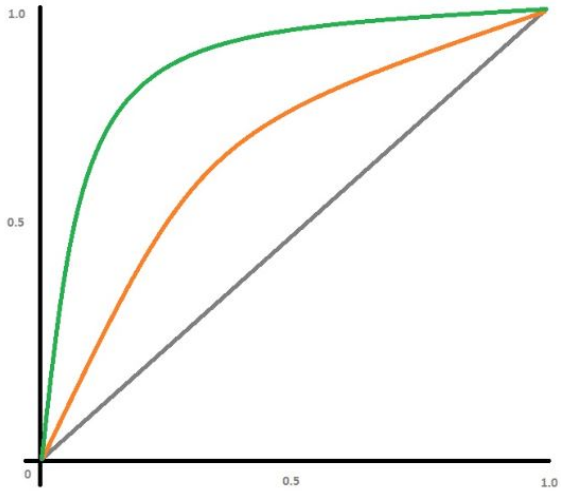
characteristic-curve-roc-curve-and-auc-1/) value
- Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
- Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task

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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

4. **[Task-2]**

- Select top 2000 features from feature Set 2 using `SelectKBest` (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) 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 (http://zetcode.com/python/prettytable/)

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

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. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

2. K Nearest Neighbor

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

In [46]:

```
# please write all the code with proper documentation, and proper titles for each subsectio
# 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 cod
# 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 [47]:

```
y = project_data['project_is_approved'].values
x = project_data.drop(['project_is_approved'], axis=1)
```

In [48]:

```
#splitting Data into Train, Test, Cv

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=
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [49]:

```
# please write all the code with proper documentation, and proper titles for each subsectio
# 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 cod
# 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 [ ]:

```

```

In [50]:

```
#encoding numerical features


#Price
from sklearn.preprocessing import Normalizer
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_test_price_norm = normalizer.transform(x_test['price'].values.reshape(1, -1))
x_cv_price_norm = normalizer.transform(x_cv['price'].values.reshape(1, -1))



print('AFter vectorizations:')
print(x_train_price_norm.shape, y_train.shape)
print(x_test_price_norm.shape, y_test.shape)
print(x_cv_price_norm.shape, y_cv.shape)
```

```
AFter vectorizations:
(1, 11222) (11222,)
(1, 8250) (8250,)
(1, 5528) (5528,)
```

In [51]:

```
#Teacher_number_of_previously_posted_projects

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

x_train_previous_projects_norm = normalizer.transform(x_train['teacher_number_of_previously
x_test_previous_projects_norm = normalizer.transform(x_test['teacher_number_of_previously_p
x_cv_previous_projects_norm = normalizer.transform(x_cv['teacher_number_of_previously_poste

print('After Vectorizations:')
print(x_train_previous_projects_norm.shape, y_train.shape)
print(x_test_previous_projects_norm.shape, y_test.shape)
print(x_cv_previous_projects_norm.shape, y_cv.shape)
```

```
After Vectorizations:
(1, 11222) (11222,)
(1, 8250) (8250,)
(1, 5528) (5528,)
```

In [ ]:

In [52]:

```
#Encoding Categorical Features

#school_state
vectorizer = CountVectorizer()
vectorizer.fit(x_train['school_state'].values)

x_train_state_one_hot = vectorizer.transform(x_train['school_state'].values)
x_test_state_one_hot = vectorizer.transform(x_test['school_state'].values)
x_cv_state_one_hot = vectorizer.transform(x_cv['school_state'].values)

print("After Vectorizations:")
print(x_train_state_one_hot.shape, y_train.shape)
print(x_test_state_one_hot.shape, y_test.shape)
print(x_cv_state_one_hot.shape, y_cv.shape)
print(vectorizer.get_feature_names())
```

```
After Vectorizations:
(11222, 51) (11222,)
(8250, 51) (8250,)
(5528, 51) (5528,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'i
a', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo',
'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'o
r', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv',
'wy']
```

In [53]:

```
#teacher_prefix

vectorizer = CountVectorizer()
vectorizer.fit(x_train['teacher_prefix'].values.astype('U'))

x_train_teacher_one_hot = vectorizer.transform(x_train['teacher_prefix'].values.astype('U'))
x_test_teacher_one_hot = vectorizer.transform(x_test['teacher_prefix'].values.astype('U'))
x_cv_teacher_one_hot = vectorizer.transform(x_cv['teacher_prefix'].values.astype('U'))

print("After Vectorizations:")
print(x_train_teacher_one_hot.shape, y_train.shape)
print(x_test_teacher_one_hot.shape, y_test.shape)
print(x_cv_teacher_one_hot.shape, y_cv.shape)
print(vectorizer.get_feature_names())
```

```
After Vectorizations:
(11222, 5) (11222,)
(8250, 5) (8250,)
(5528, 5) (5528,)
['mr', 'mrs', 'ms', 'nan', 'teacher']
```

In [54]:

```
#Project_grade_category

vectorizer = CountVectorizer()
vectorizer.fit(x_train['project_grade_category'].values)

x_train_grade_one_hot = vectorizer.transform(x_train['project_grade_category'].values)
x_test_grade_one_hot = vectorizer.transform(x_test['project_grade_category'].values)
x_cv_grade_one_hot = vectorizer.transform(x_cv['project_grade_category'].values)

print("After Vectorizations:")
print(x_train_grade_one_hot.shape, y_train.shape)
print(x_test_grade_one_hot.shape, y_test.shape)
print(x_cv_grade_one_hot.shape, y_cv.shape)
print(vectorizer.get_feature_names())
```

```
After Vectorizations:
(11222, 4) (11222,)
(8250, 4) (8250,)
(5528, 4) (5528,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

In [55]:

```
#project_subject_categories


vectorizer = CountVectorizer()
vectorizer.fit(x_train['clean_categories'].values)

x_train_categories_one_hot = vectorizer.transform(x_train['clean_categories'].values)
x_test_categories_one_hot = vectorizer.transform(x_test['clean_categories'].values)
x_cv_categories_one_hot = vectorizer.transform(x_cv['clean_categories'].values)

print("After Vectorizations:")
print(x_train_categories_one_hot.shape, y_train.shape)
print(x_test_categories_one_hot.shape, y_test.shape)
print(x_cv_categories_one_hot.shape, y_cv.shape)
print(vectorizer.get_feature_names())
```

```
After Vectorizations:
(11222, 9) (11222,)
(8250, 9) (8250,)
(5528, 9) (5528,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'liter
acy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
```

In [56]:

```
#project_subject_subcategories


vectorizer = CountVectorizer()
vectorizer.fit(x_train['clean_subcategories'].values)

x_train_subcategories_one_hot = vectorizer.transform(x_train['clean_subcategories'].values)
x_test_subcategories_one_hot = vectorizer.transform(x_test['clean_subcategories'].values)
x_cv_subcategories_one_hot = vectorizer.transform(x_cv['clean_subcategories'].values)

print("After Vectorizations:")
print(x_train_subcategories_one_hot.shape, y_train.shape)
print(x_test_subcategories_one_hot.shape, y_test.shape)
print(x_cv_subcategories_one_hot.shape, y_cv.shape)
print(vectorizer.get_feature_names())
```

```
After Vectorizations:
(11222, 30) (11222,)
(8250, 30) (8250,)
(5528, 30) (5528,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_governmen
t', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economic
s', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy',
'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness',
'history_geography', 'literacy', 'literature_writing', 'mathematics', 'musi
c', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 's
ocialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [57]:

```
# please write all the code with proper documentation, and proper titles for each subsectio
# 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 cod
# 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
```

BOW: essay, project_title

In [58]:

```
vectorizer = CountVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer.fit(x_train['essay'].values)

x_train_essay_bow = vectorizer.transform(x_train['essay'].values)
x_test_essay_bow = vectorizer.transform(x_test['essay'].values)
x_cv_essay_bow = vectorizer.transform(x_cv['essay'].values)

print("After Vectorizations:")
print(x_train_essay_bow.shape, y_train.shape)
print(x_test_essay_bow.shape, y_test.shape)
print(x_cv_essay_bow.shape, y_cv.shape)
```

```
After Vectorizations:
(11222, 5000) (11222,)
(8250, 5000) (8250,)
(5528, 5000) (5528,)
```

In [59]:

```
vectorizer = CountVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer.fit(x_train['project_title'].values)

x_train_title_bow = vectorizer.transform(x_train['project_title'].values)
x_test_title_bow = vectorizer.transform(x_test['project_title'].values)
x_cv_title_bow = vectorizer.transform(x_cv['project_title'].values)

print("After Vectorizations:")
print(x_train_title_bow.shape, y_train.shape)
print(x_test_title_bow.shape, y_test.shape)
print(x_cv_title_bow.shape, y_cv.shape)
```

```
After Vectorizations:
(11222, 1295) (11222,)
(8250, 1295) (8250,)
(5528, 1295) (5528,)
```

TFIDf: essay, project_title

In [60]:

```
#essay

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer.fit(x_train['essay'].values)

x_train_essay_tfidf = vectorizer.transform(x_train['essay'].values)
x_test_essay_tfidf = vectorizer.transform(x_test['essay'].values)
x_cv_essay_tfidf = vectorizer.transform(x_cv['essay'].values)

print("After Vectorizations:")
print(x_train_essay_tfidf.shape, y_train.shape)
print(x_test_essay_tfidf.shape, y_test.shape)
print(x_cv_essay_tfidf.shape, y_cv.shape)
```

```
After Vectorizations:
(11222, 5000) (11222,)
(8250, 5000) (8250,)
(5528, 5000) (5528,)
```

In [61]:

```
#project_title

vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer.fit(x_train['project_title'].values)

x_train_title_tfidf = vectorizer.transform(x_train['project_title'].values)
x_test_title_tfidf = vectorizer.transform(x_test['project_title'].values)
x_cv_title_tfidf = vectorizer.transform(x_cv['project_title'].values)

print("After Vectorizations:")
print(x_train_title_tfidf.shape, y_train.shape)
print(x_test_title_tfidf.shape, y_test.shape)
print(x_cv_title_tfidf.shape, y_cv.shape)
```

```
After Vectorizations:
(11222, 1295) (11222,)
(8250, 1295) (8250,)
(5528, 1295) (5528,)
```

Average word2vec: essay, project_title

In [62]:

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

Essay

In [63]:

```
# average Word2Vec
# compute average word2vec for each review.
x_train_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train['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
    x_train_essay_avg_w2v.append(vector)

print(len(x_train_essay_avg_w2v))
print(len(x_train_essay_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████
███| 11222/11222 [00:08<00:00, 1310.83it/s]

11222
300
```

In [64]:

```
x_cv_essay_avg_w2v = []; # 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
    x_cv_essay_avg_w2v.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████
█████| 5528/5528 [00:04<00:00, 1237.89it/s]
```

In [65]:

```
x_test_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_test['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
    x_test_essay_avg_w2v.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████
█████| 8250/8250 [00:06<00:00, 1318.57it/s]
```

project_title

In [66]:

```
x_train_title_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train['project_title'].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
    x_train_title_avg_w2v.append(vector)

print(len(x_train_title_avg_w2v))
print(len(x_train_title_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██| 11222/11222 [00:00<00:00, 51218.23it/s]

11222
300
```

In [67]:

```
x_test_title_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_test['project_title'].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
    x_test_title_avg_w2v.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████
████| 8250/8250 [00:00<00:00, 73622.79it/s]
```

In [68]:

```
x_cv_title_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_cv['project_title'].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
    x_cv_title_avg_w2v.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████
████| 5528/5528 [00:00<00:00, 65743.19it/s]
```

Tfidf w2v

In [69]:

```
#re-using some code from previous assignment


#essay

tfidf_model = TfidfVectorizer()
tfidf_model.fit(x_train['essay'])
# 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 [70]:

```
x_train_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train['essay']): # 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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_train_essay_tfidf_w2v.append(vector)

print(len(x_train_essay_tfidf_w2v))
```

```
100%|████████████████████████████████████████████████████████████████████████████
████| 11222/11222 [01:27<00:00, 128.70it/s]

11222
```

In [71]:

```
x_test_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_test['essay']): # 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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_test_essay_tfidf_w2v.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████████
██████| 8250/8250 [01:03<00:00, 129.10it/s]
```

In [72]:

```
x_cv_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_cv['essay']): # 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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_cv_essay_tfidf_w2v.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████████
██████| 5528/5528 [00:42<00:00, 130.86it/s]
```

In [73]:

```
#project_title

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

In [74]:

```
x_train_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_train_title_tfidf_w2v.append(vector)

print(len(x_train_title_tfidf_w2v))
```

```
100%|████████████████████████████████████████████████████████████████████████████
██| 11222/11222 [00:00<00:00, 42578.53it/s]

11222
```

In [75]:

```
x_test_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_test['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_test_title_tfidf_w2v.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████
████| 8250/8250 [00:00<00:00, 39067.44it/s]
```

In [76]:

```
x_cv_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_cv['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_cv_title_tfidf_w2v.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████
████| 5528/5528 [00:00<00:00, 47806.61it/s]
```

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 [77]:

```
# please write all the code with proper documentation, and proper titles for each subsectio
# 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 cod

# 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 [78]:

```
#Before merging, re-shape some features. if we dont reshape, we'll get error
#Re-shaping
x_train_price_norm = x_train_price_norm.reshape(-1,1)
x_train_previous_projects_norm = x_train_previous_projects_norm.reshape(-1,1)

x_test_price_norm = x_test_price_norm.reshape(-1,1)
x_test_previous_projects_norm = x_test_previous_projects_norm.reshape(-1,1)

x_cv_price_norm = x_cv_price_norm.reshape(-1,1)
x_cv_previous_projects_norm = x_cv_previous_projects_norm.reshape(-1,1)
```

2.4.1 Applying KNN brute force on BOW, SET 1

In [79]:

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

In [80]:

```
#Merging features

from scipy.sparse import hstack
x_train_knn_bow = hstack((x_train_price_norm, x_train_previous_projects_norm, x_train_state
                      x_train_grade_one_hot, x_train_categories_one_hot, x_train_subcategori
                      x_train_title_bow)).tocsr()

x_test_knn_bow = hstack((x_test_price_norm, x_test_previous_projects_norm, x_test_state_one
                      x_test_grade_one_hot, x_test_categories_one_hot, x_test_subcategories_
                      x_test_title_bow)).tocsr()

x_cv_knn_bow = hstack((x_cv_price_norm, x_cv_previous_projects_norm, x_cv_state_one_hot, x_
                      x_cv_grade_one_hot, x_cv_categories_one_hot, x_cv_subcategories_one_ho
                      x_cv_title_bow)).tocsr()
```

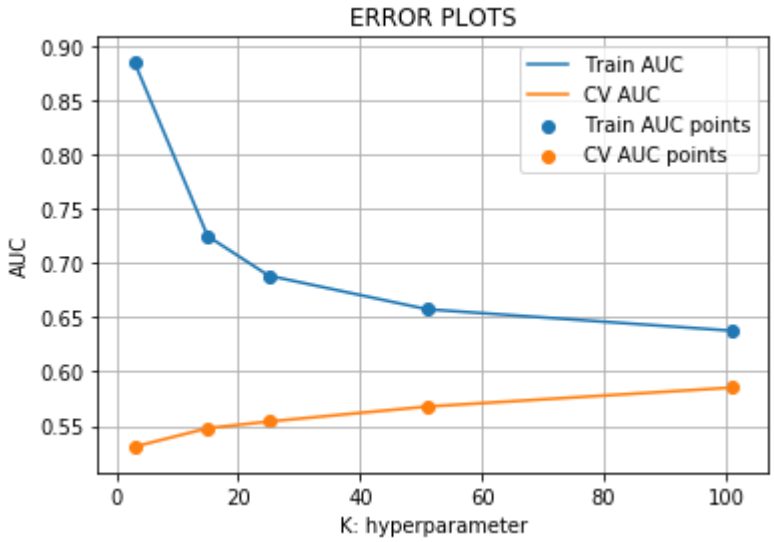
In [81]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # 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 tr_loop will be 49041 - 49041%1000 = 4900
    # 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
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [82]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
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
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 = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(x_train_knn_bow, y_train)

    y_train_pred = batch_predict(neigh, x_train_knn_bow)
    y_cv_pred = batch_predict(neigh, x_cv_knn_bow)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
100%|██████████| 5/5 [02:38<00:00, 31.85s/it]
```

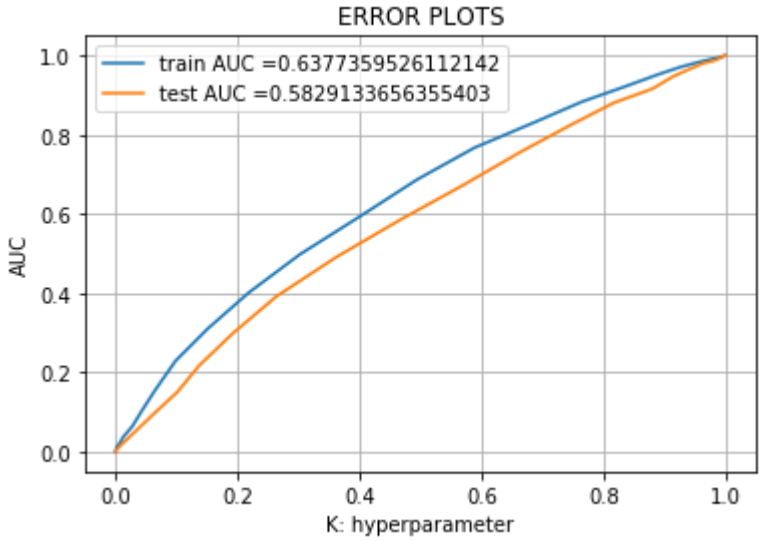


In [83]:

```
#here we are choosing the best_k based on forloop results
#We choose best_k such that we'll have maximum auc on cv and gap b/w test and train is less
best_k = 101
```

In [84]:

```
#Hyper parameter tuning with best_k
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(x_train_knn_bow, y_train)

y_train_pred = batch_predict(neigh, x_train_knn_bow)
y_test_pred = batch_predict(neigh, x_test_knn_bow)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [85]:

```
# 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
    return t

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

In [112]:

```
#Confusion matrix

from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.32107641042065593 for threshold 0.864
Train confusion matrix
[[1113  600]
 [4810 4699]]
Test confusion matrix
[[ 706  553]
 [3717 3274]]
```

In [113]:

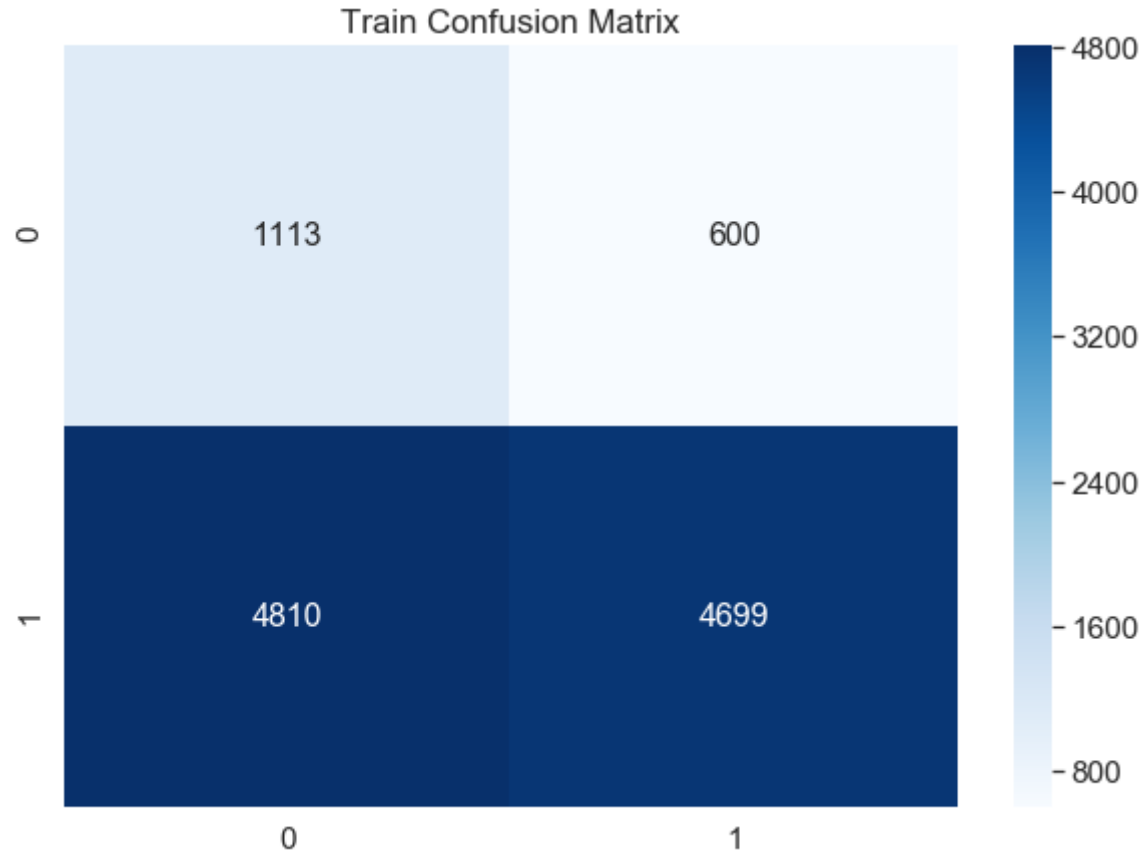
```
import seaborn as sns
```

In [114]:

```
#Train Confusion matrix

#Reference- https://www.kaggle.com/agungor2/various-confusion-matrix-plots

df_cm = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)),

plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
plt.title('Train Confusion Matrix')
sns.heatmap(df_cm, cmap="Blues", annot=True, fmt='g', annot_kws={"size": 16})
```

Out[114]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x26401a6cf98>
```

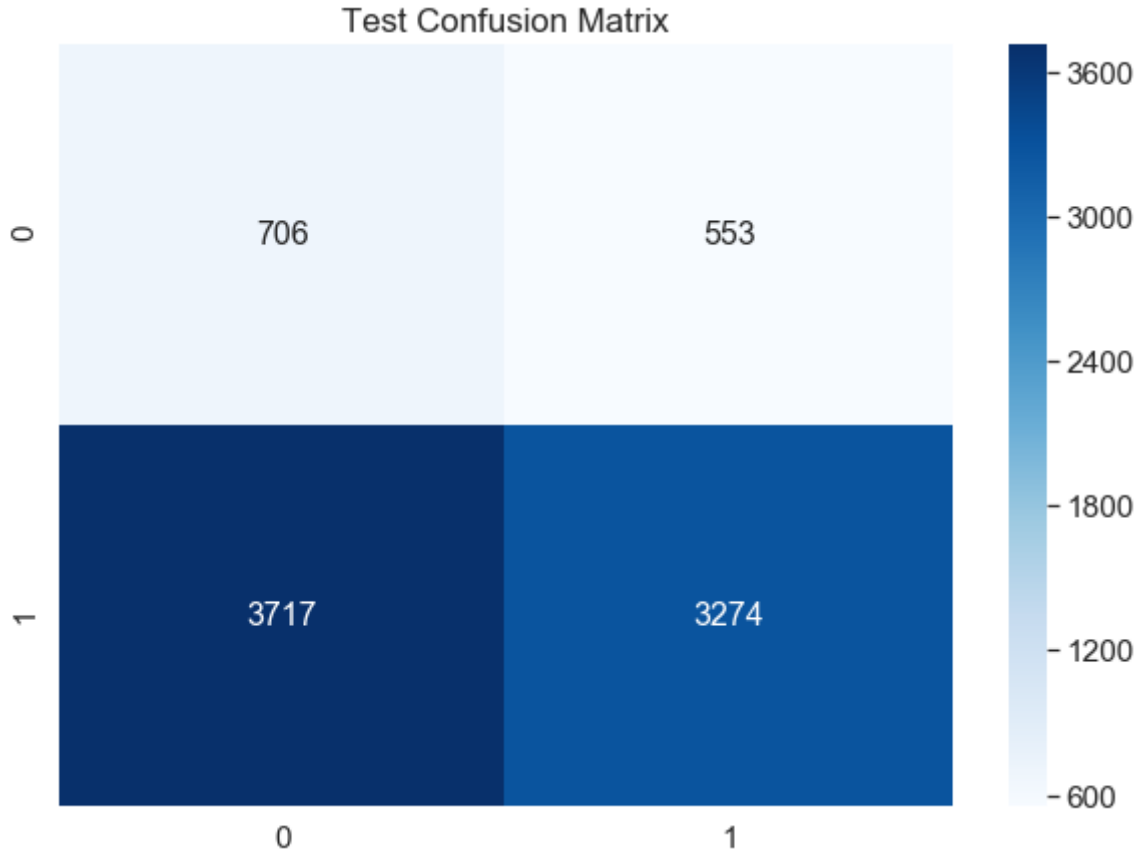


In [115]:

```
#Test Confusion matrix
df_cm = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), cc

plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
plt.title('Test Confusion Matrix')
sns.heatmap(df_cm, cmap="Blues", annot=True, fmt='g', annot_kws={"size": 16})
```

Out[115]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x26419159320>
```



2.4.2 Applying KNN brute force on TFIDF, SET 2

In [91]:

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

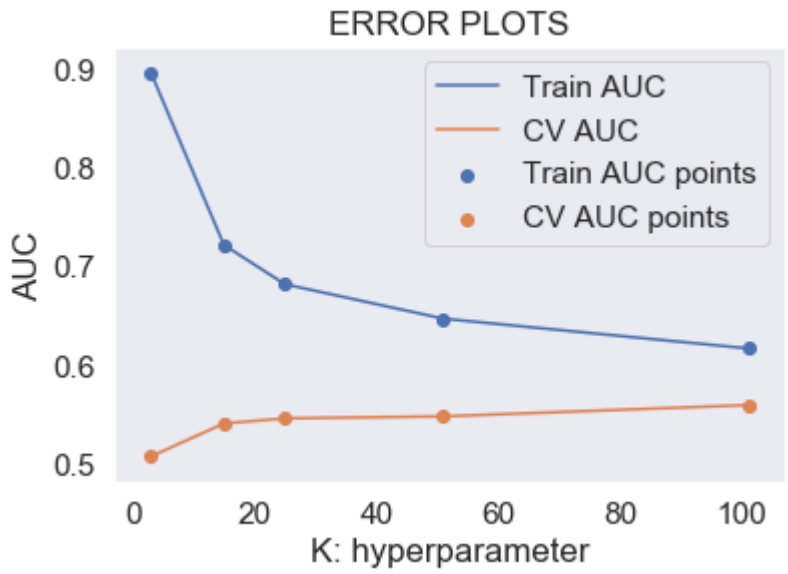
In [92]:

```
#Merging Features

x_train_knn_tfidf = hstack((x_train_price_norm, x_train_previous_projects_norm, x_train_sta
                    x_train_grade_one_hot, x_train_categories_one_hot, x_train_subcategori
                    x_train_title_tfidf)).tocsr()

x_test_knn_tfidf = hstack((x_test_price_norm, x_test_previous_projects_norm, x_test_state_c
                    x_test_grade_one_hot, x_test_categories_one_hot, x_test_subcategories_
                    x_test_title_tfidf)).tocsr()

x_cv_knn_tfidf = hstack((x_cv_price_norm, x_cv_previous_projects_norm, x_cv_state_one_hot,
                    x_cv_grade_one_hot, x_cv_categories_one_hot, x_cv_subcategories_one_hc
                    x_cv_title_tfidf)).tocsr()
```

In [93]:

```
#Hyper parameter tuning

import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
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
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 = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(x_train_knn_tfidf, y_train)

    y_train_pred = batch_predict(neigh, x_train_knn_tfidf)
    y_cv_pred = batch_predict(neigh, x_cv_knn_tfidf)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
100%|██████████| 5/5 [02:36<00:00, 31.25s/it]
```



In [ ]:

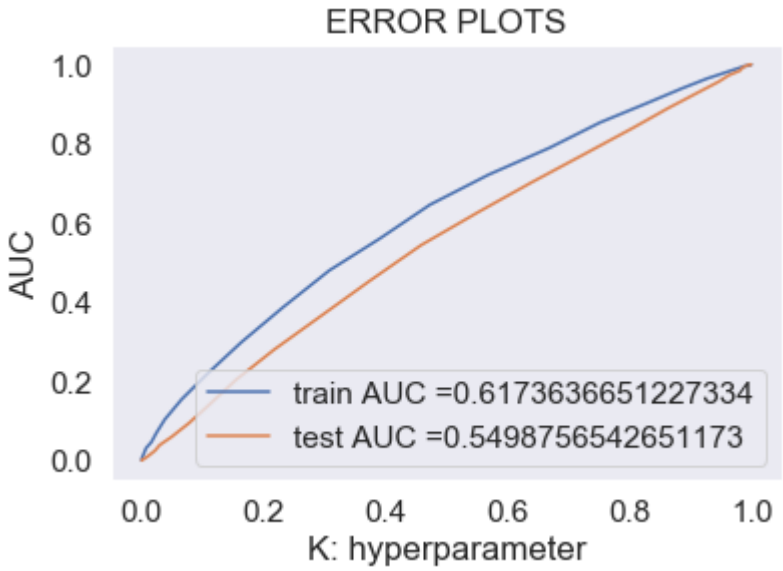
```
```

In [94]:

```
#here we are choosing the best_k based on forloop results
best_k = 103
```

In [95]:

```
#Hyper parameter tuning with best_k
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(x_train_knn_tfidf, y_train)

y_train_pred = batch_predict(neigh, x_train_knn_tfidf)
y_test_pred = batch_predict(neigh, x_test_knn_tfidf)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [116]:

```
# 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
    return t

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

In [117]:

```
#Confusion matrix

from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

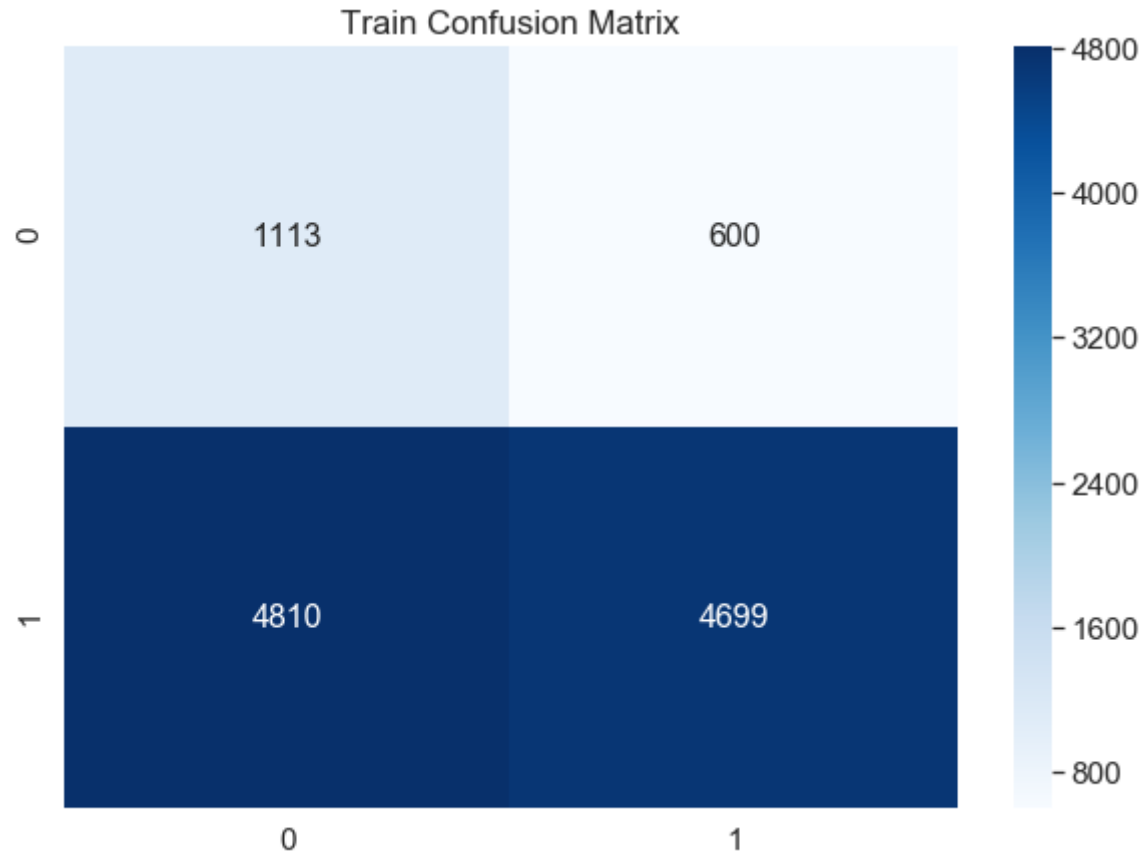
```
the maximum value of tpr*(1-fpr) 0.32107641042065593 for threshold 0.864
Train confusion matrix
[[1113  600]
 [4810 4699]]
Test confusion matrix
[[ 706  553]
 [3717 3274]]
```

In [118]:

```
#Train Confusion matrix
df_cm = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)),

plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
plt.title('Train Confusion Matrix')
sns.heatmap(df_cm, cmap="Blues", annot=True, fmt='g', annot_kws={"size": 16})
```

Out[118]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x264007037f0>
```

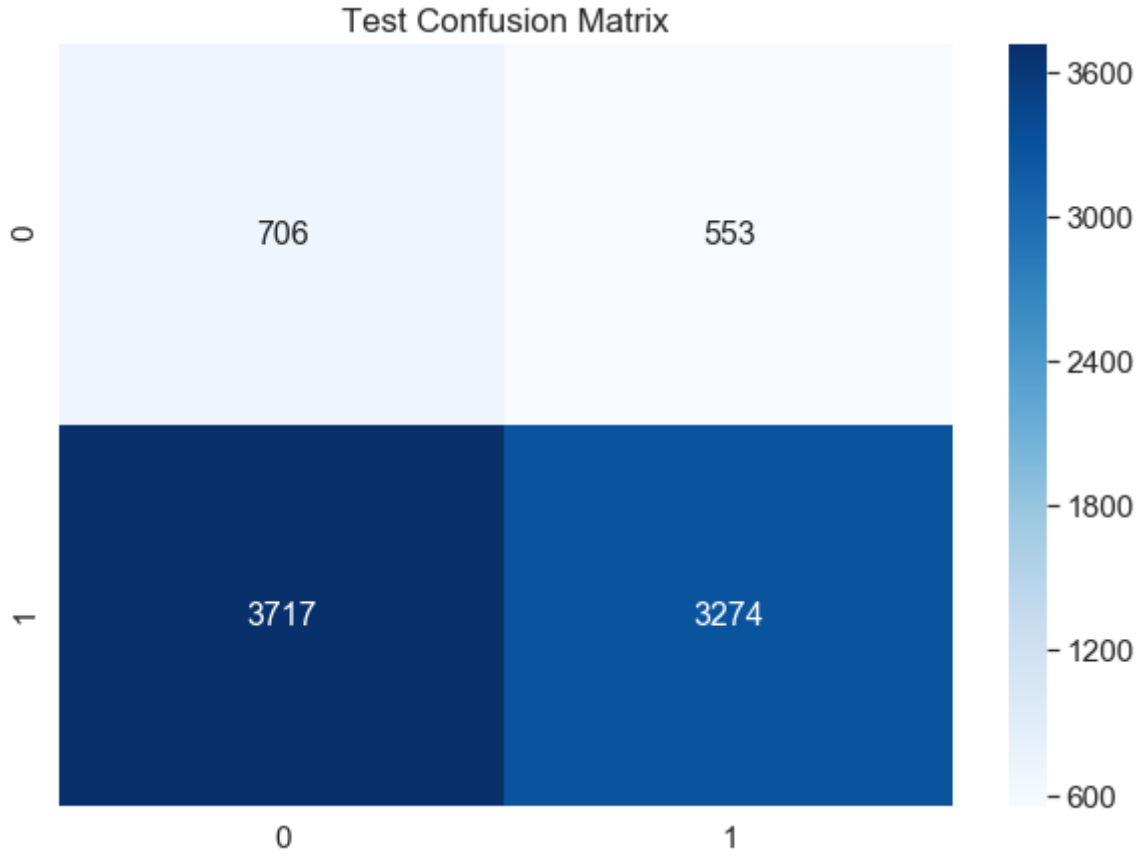


In [119]:

```
#Test Confusion matrix
df_cm = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), cc

plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
plt.title('Test Confusion Matrix')
sns.heatmap(df_cm, cmap="Blues", annot=True, fmt='g', annot_kws={"size": 16})
```

Out[119]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2641915f278>
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [100]:

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

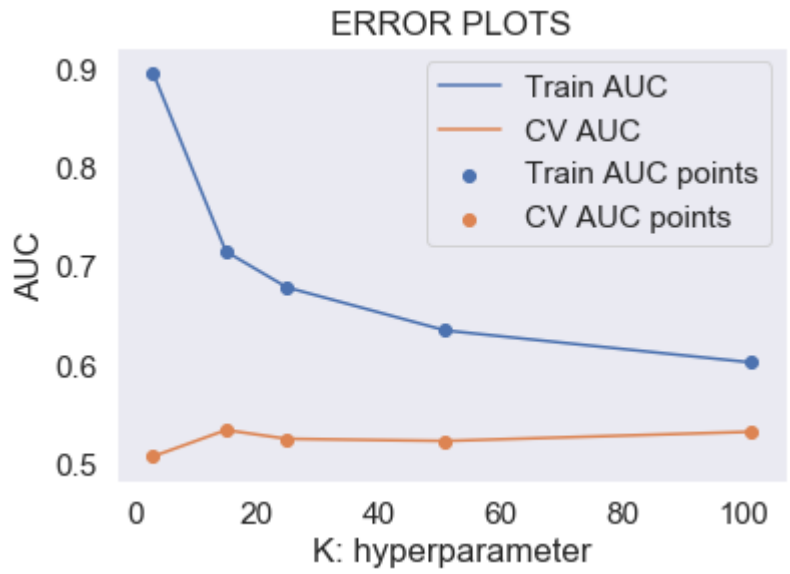
In [101]:

```
#Merging features

x_train_knn_avg_w2v = hstack((x_train_price_norm, x_train_previous_projects_norm, x_train_s
                      x_train_grade_one_hot, x_train_categories_one_hot, x_train_subcategori
                      x_train_title_avg_w2v)).tocsr()

x_test_knn_avg_w2v = hstack((x_test_price_norm, x_test_previous_projects_norm, x_test_state
                      x_test_grade_one_hot, x_test_categories_one_hot, x_test_subcategories_
                      x_test_title_avg_w2v)).tocsr()

x_cv_knn_avg_w2v = hstack((x_cv_price_norm, x_cv_previous_projects_norm, x_cv_state_one_hot
                      x_cv_grade_one_hot, x_cv_categories_one_hot, x_cv_subcategories_one_ho
                      x_cv_title_avg_w2v)).tocsr()
```

In [102]:

```
#Hyperparameter tuning

import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
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
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 = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(x_train_knn_avg_w2v, y_train)

    y_train_pred = batch_predict(neigh, x_train_knn_avg_w2v)
    y_cv_pred = batch_predict(neigh, x_cv_knn_avg_w2v)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
100%|██████████| 5/5 [09:45<00:00, 116.90s/it]
```



In [ ]:

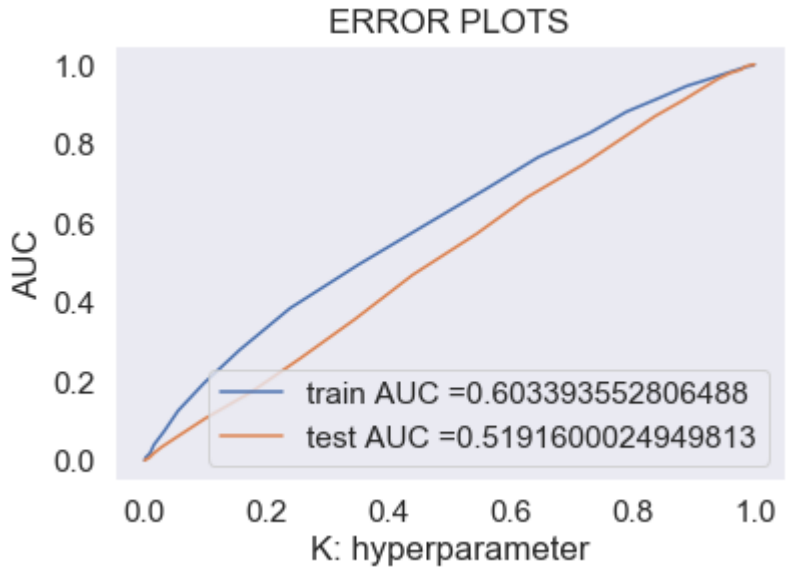
```
```

In [103]:

```
#here we are choosing the best_k based on forloop results
best_k = 103
```

In [104]:

```
#Hyper parameter tuning with best_k
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(x_train_knn_avg_w2v, y_train)

y_train_pred = batch_predict(neigh, x_train_knn_avg_w2v)
y_test_pred = batch_predict(neigh, x_test_knn_avg_w2v)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [120]:

```
# 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
    return t

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

In [121]:

```
#Confusion matrix

from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

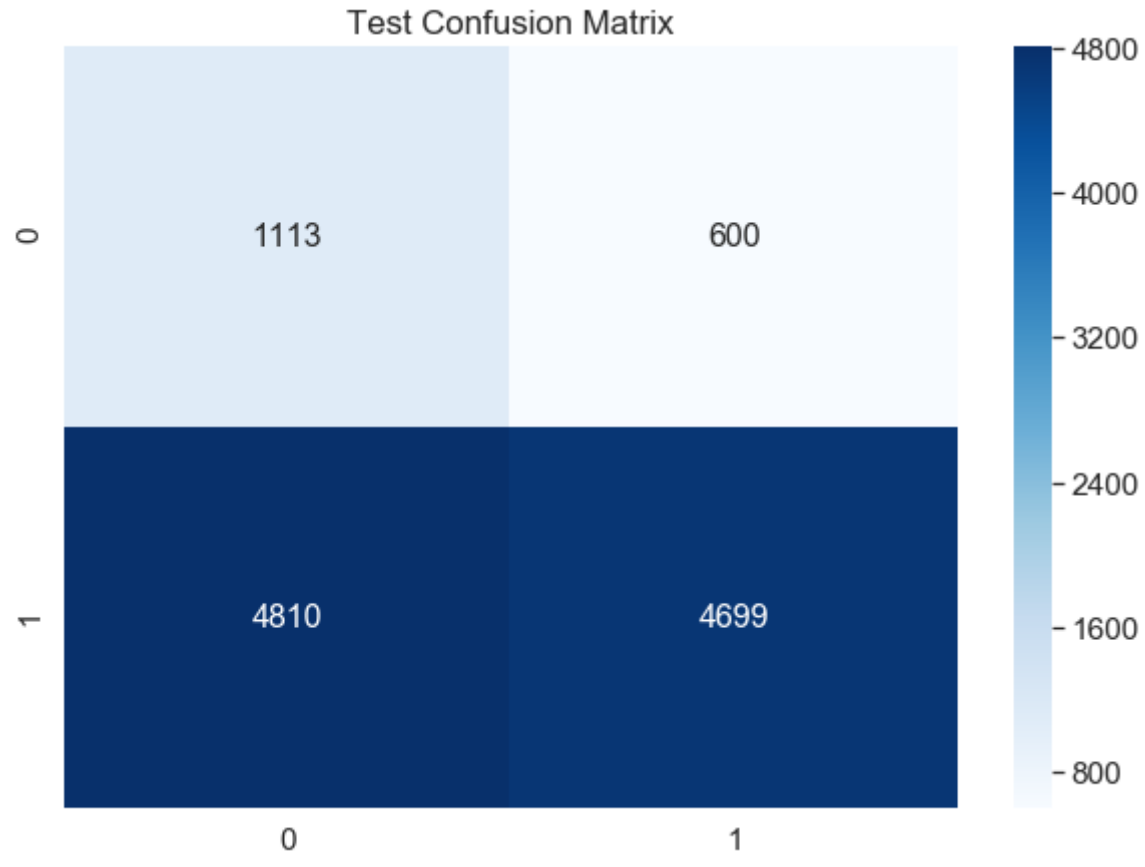
```
the maximum value of tpr*(1-fpr) 0.32107641042065593 for threshold 0.864
Train confusion matrix
[[1113  600]
 [4810 4699]]
Test confusion matrix
[[ 706  553]
 [3717 3274]]
```

In [122]:

```
#Train Confusion matrix
df_cm = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)),

plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
plt.title('Test Confusion Matrix')
sns.heatmap(df_cm, cmap="Blues", annot=True, fmt='g', annot_kws={"size": 16})
```

Out[122]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x264007820b8>
```

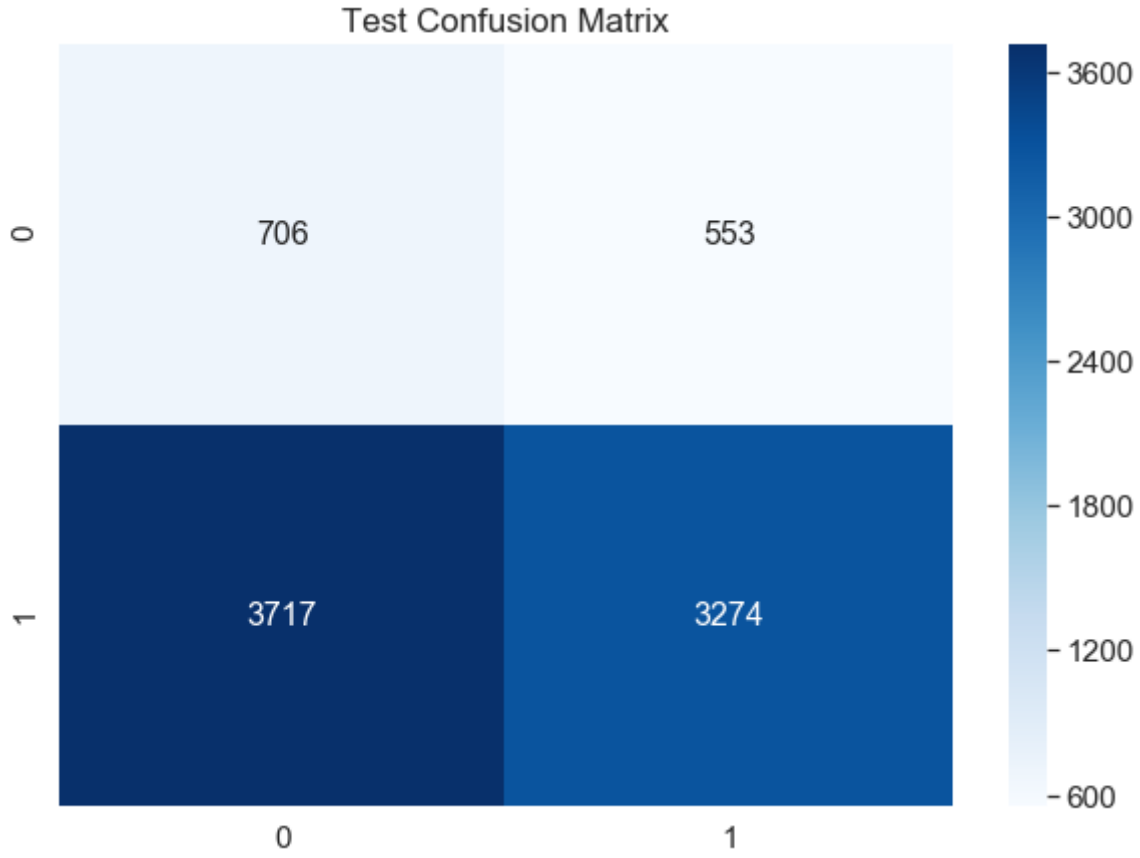


In [123]:

```
#Test Confusion matrix
df_cm = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), cc

plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
plt.title('Test Confusion Matrix')
sns.heatmap(df_cm, cmap="Blues", annot=True, fmt='g', annot_kws={"size": 16})
```

Out[123]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x264008f5518>
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [124]:

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

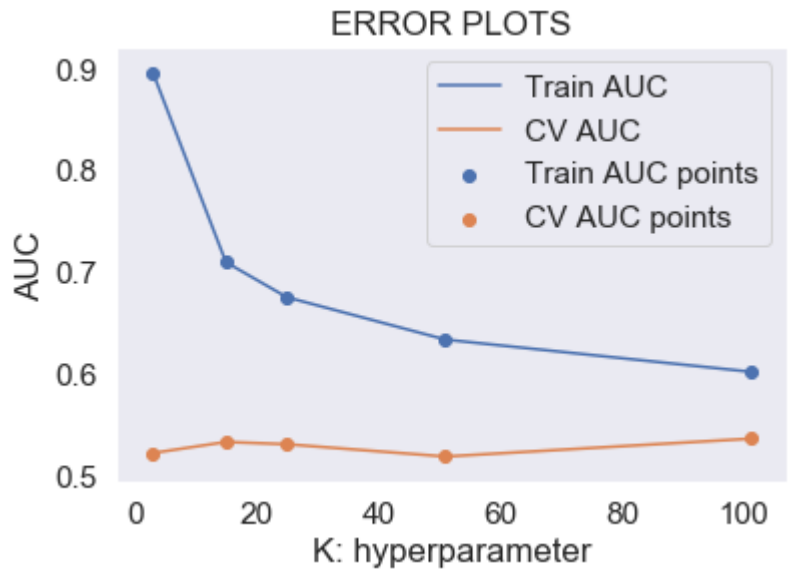
In [125]:

```
#Merging features

x_train_knn_tfidf_w2v = hstack((x_train_price_norm, x_train_previous_projects_norm, x_trair
                    x_train_grade_one_hot, x_train_categories_one_hot, x_train_subcategori
                    x_train_title_tfidf_w2v)).tocsr()

x_test_knn_tfidf_w2v = hstack((x_test_price_norm, x_test_previous_projects_norm, x_test_sta
                    x_test_grade_one_hot, x_test_categories_one_hot, x_test_subcategories_
                    x_test_title_tfidf_w2v)).tocsr()

x_cv_knn_tfidf_w2v = hstack((x_cv_price_norm, x_cv_previous_projects_norm, x_cv_state_one_h
                    x_cv_grade_one_hot, x_cv_categories_one_hot, x_cv_subcategories_one_ho
                    x_cv_title_tfidf_w2v)).tocsr()
```

In [128]:

```
#Hyper parameter tuning

import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
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
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 = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(x_train_knn_tfidf_w2v, y_train)

    y_train_pred = batch_predict(neigh, x_train_knn_tfidf_w2v)
    y_cv_pred = batch_predict(neigh, x_cv_knn_tfidf_w2v)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
  0%|
| 0/5 [00:00<?, ?it/s]
 20%|████████████████
| 1/5 [01:56<07:44, 116.05s/it]
 40%|████████████████████████████████
| 2/5 [03:51<05:47, 115.86s/it]
 60%|████████████████████████████████████████████████
| 3/5 [05:47<03:51, 115.84s/it]
 80%|████████████████████████████████████████████████████████████████
| 4/5 [07:42<01:55, 115.75s/it]
100%|████████████████████████████████████████████████████████████████
████████████| 5/5 [09:38<00:00, 115.60s/it]
```



In [ ]:

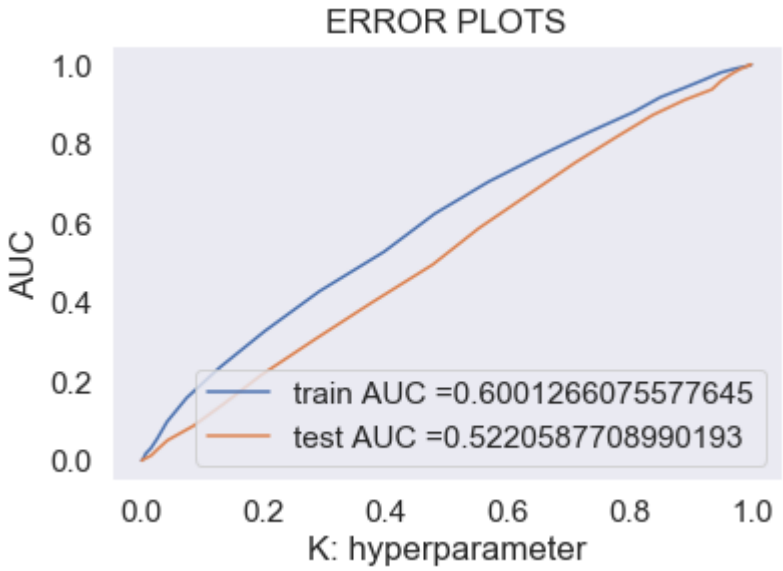
```
```

In [129]:

```
#here we are choosing the best_k based on forloop results
best_k = 105
```

In [130]:

```
#Hyper parameter tuning with best_k
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(x_train_knn_tfidf_w2v, y_train)

y_train_pred = batch_predict(neigh, x_train_knn_tfidf_w2v)
y_test_pred = batch_predict(neigh, x_test_knn_tfidf_w2v)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [135]:

```
# 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
    return t

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

In [136]:

```
#Confusion matrix

from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

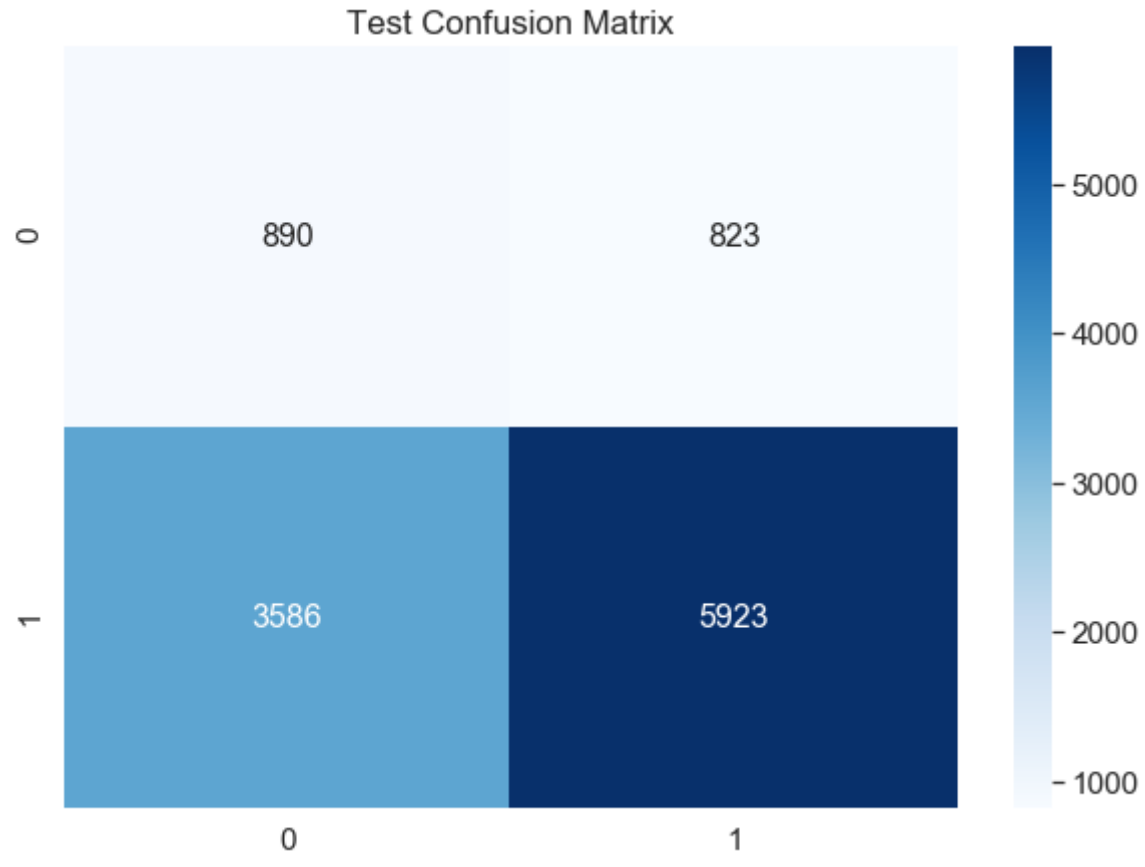
```
the maximum value of tpr*(1-fpr) 0.32362311134619937 for threshold 0.848
Train confusion matrix
[[ 890  823]
 [3586 5923]]
Test confusion matrix
[[ 563  696]
 [2893 4098]]
```

In [137]:

```
#Train Confusion matrix
df_cm = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)),

plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
plt.title('Test Confusion Matrix')
sns.heatmap(df_cm, cmap="Blues", annot=True, fmt='g', annot_kws={"size": 16})
```

Out[137]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x264007f1f28>
```

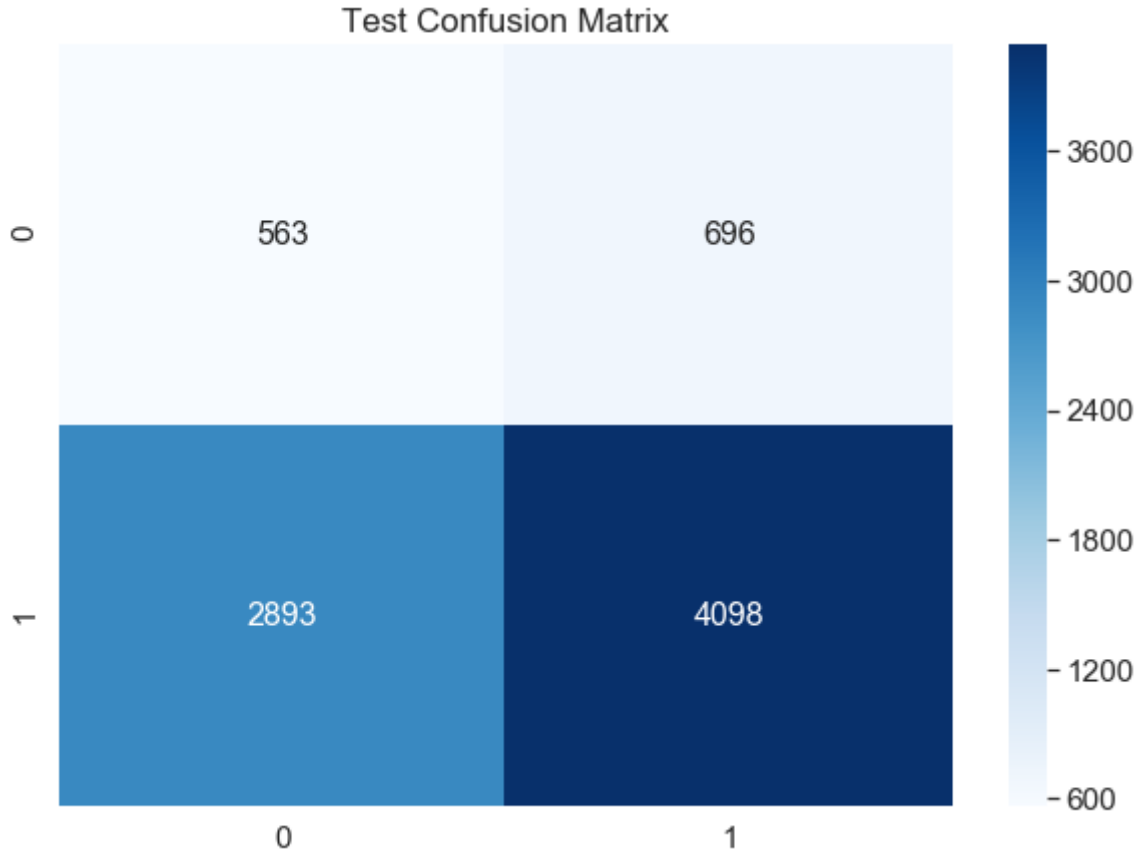


In [138]:

```
#Test Confusion matrix
df_cm = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), cc

plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
plt.title('Test Confusion Matrix')
sns.heatmap(df_cm, cmap="Blues", annot=True, fmt='g', annot_kws={"size": 16})
```

Out[138]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x264002af5c0>
```



2.5 Feature selection with `SelectKBest`

In [139]:

```
# please write all the code with proper documentation, and proper titles for each subsectio
# 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 cod

# 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 [140]:

```
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, chi2, f_classif
```

In [141]:

```
#Merging Features

x_train_knn_tfidf = hstack((x_train_price_norm, x_train_previous_projects_norm, x_train_sta
                     x_train_grade_one_hot, x_train_categories_one_hot, x_train_subcategori
                     x_train_title_tfidf)).tocsr()

x_test_knn_tfidf = hstack((x_test_price_norm, x_test_previous_projects_norm, x_test_state_c
                     x_test_grade_one_hot, x_test_categories_one_hot, x_test_subcategories_
                     x_test_title_tfidf)).tocsr()

x_cv_knn_tfidf = hstack((x_cv_price_norm, x_cv_previous_projects_norm, x_cv_state_one_hot,
                     x_cv_grade_one_hot, x_cv_categories_one_hot, x_cv_subcategories_one_hc
                     x_cv_title_tfidf)).tocsr()
```

In [142]:

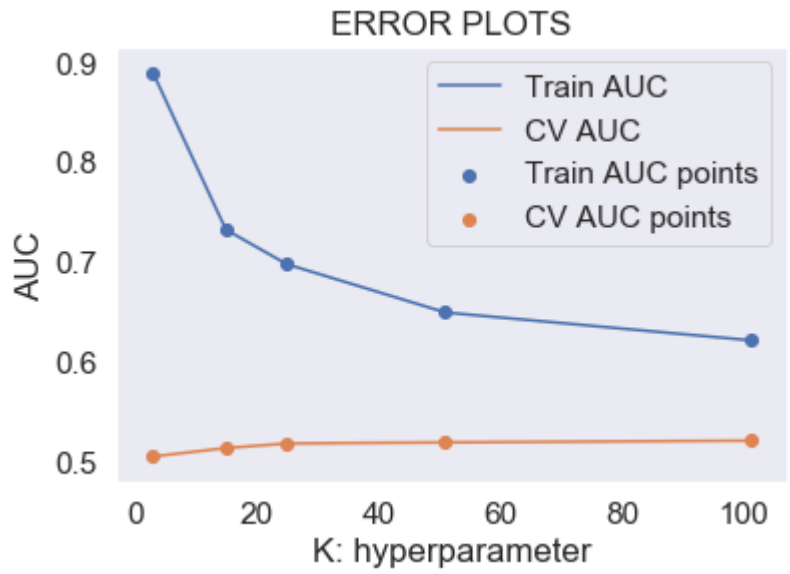
```
# selecting the best 2000 features using SelectKBest from TFIDF model
best_feature = SelectKBest(f_classif, k = 2000)
best_feature.fit(x_train_knn_tfidf, y_train)
# selecting the best 2000 features for train, test and cross validation
x_train_knn_tfidf_new = best_feature.transform(x_train_knn_tfidf)
x_cv_knn_tfidf_new = best_feature.transform(x_cv_knn_tfidf)
x_test_knn_tfidf_new = best_feature.transform(x_test_knn_tfidf)


print(x_train_knn_tfidf_new.shape)
print(x_cv_knn_tfidf_new.shape)
print(x_test_knn_tfidf_new.shape)
```

```
(11222, 2000)
(5528, 2000)
(8250, 2000)
```

In [143]:

```
#Hyper parameter tuning

import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
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
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 = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(x_train_knn_tfidf_new, y_train)

    y_train_pred = batch_predict(neigh, x_train_knn_tfidf_new)
    y_cv_pred = batch_predict(neigh, x_cv_knn_tfidf_new)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
  0%|
| 0/5 [00:00<?, ?it/s]
 20%|████████████████▍
| 1/5 [00:22<01:28, 22.11s/it]
 40%|████████████████████████████████
| 2/5 [00:45<01:07, 22.53s/it]
 60%|████████████████████████████████████████████████▏
| 3/5 [01:09<00:45, 22.79s/it]
 80%|████████████████████████████████████████████████████████████████▏
| 4/5 [01:33<00:23, 23.25s/it]
100%|████████████████████████████████████████████████████████████████████
████████████| 5/5 [01:57<00:00, 23.51s/it]
```



In [ ]:

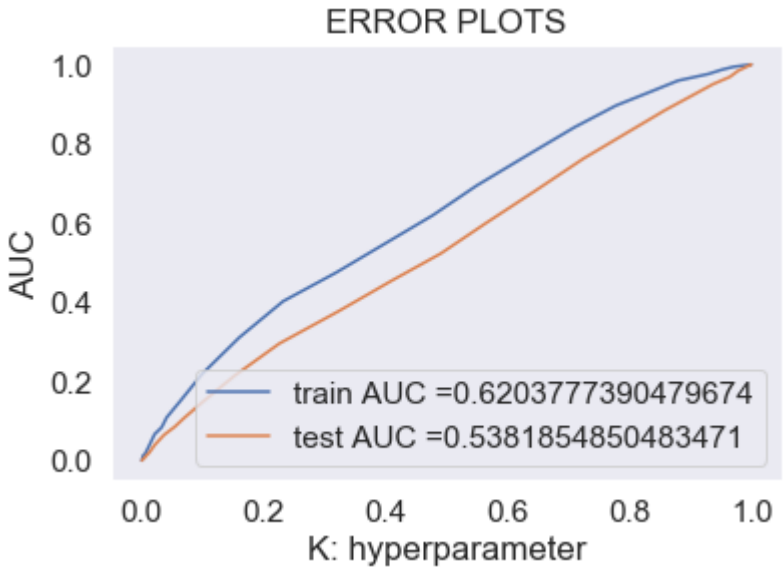
```
```

In [144]:

```
#here we are choosing the best_k based on forloop results
#We choose best_k such that we'll have maximum auc on cv and gap b/w test and train is less
#from above plot the best_k we are choosing is 103
best_k = 103
```

In [145]:

```
#Hyper parameter tuning with best_k
from sklearn.metrics import roc_curve, auc

neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(x_train_knn_tfidf_new, y_train)

y_train_pred = batch_predict(neigh, x_train_knn_tfidf_new)
y_test_pred = batch_predict(neigh, x_test_knn_tfidf_new)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [146]:

```
# 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
    return t

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

In [147]:

```
#Confusion matrix

from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

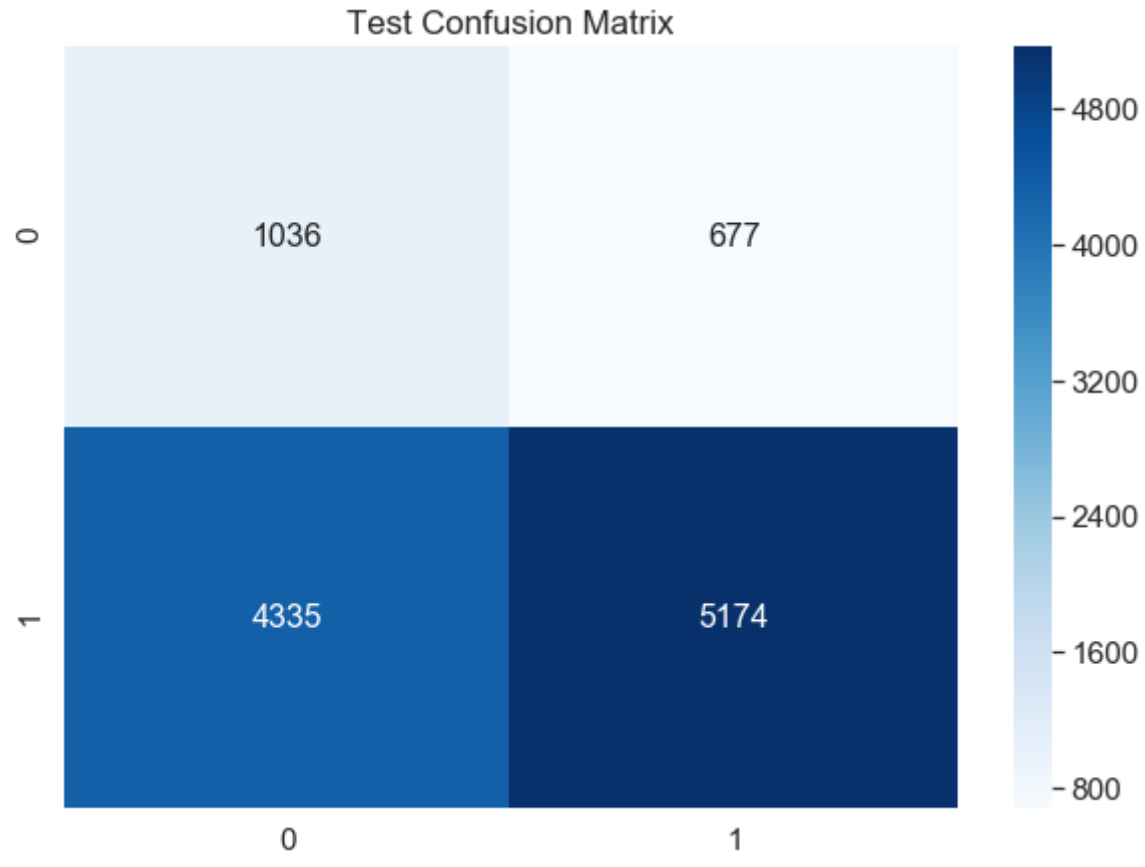
```
the maximum value of tpr*(1-fpr) 0.329074302484321 for threshold 0.845
Train confusion matrix
[[1036  677]
 [4335 5174]]
Test confusion matrix
[[ 641  618]
 [3336 3655]]
```

In [148]:

```
#Train Confusion matrix
df_cm = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)),

plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
plt.title('Test Confusion Matrix')
sns.heatmap(df_cm, cmap="Blues", annot=True, fmt='g', annot_kws={"size": 16})
```

Out[148]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x264002ec828>
```

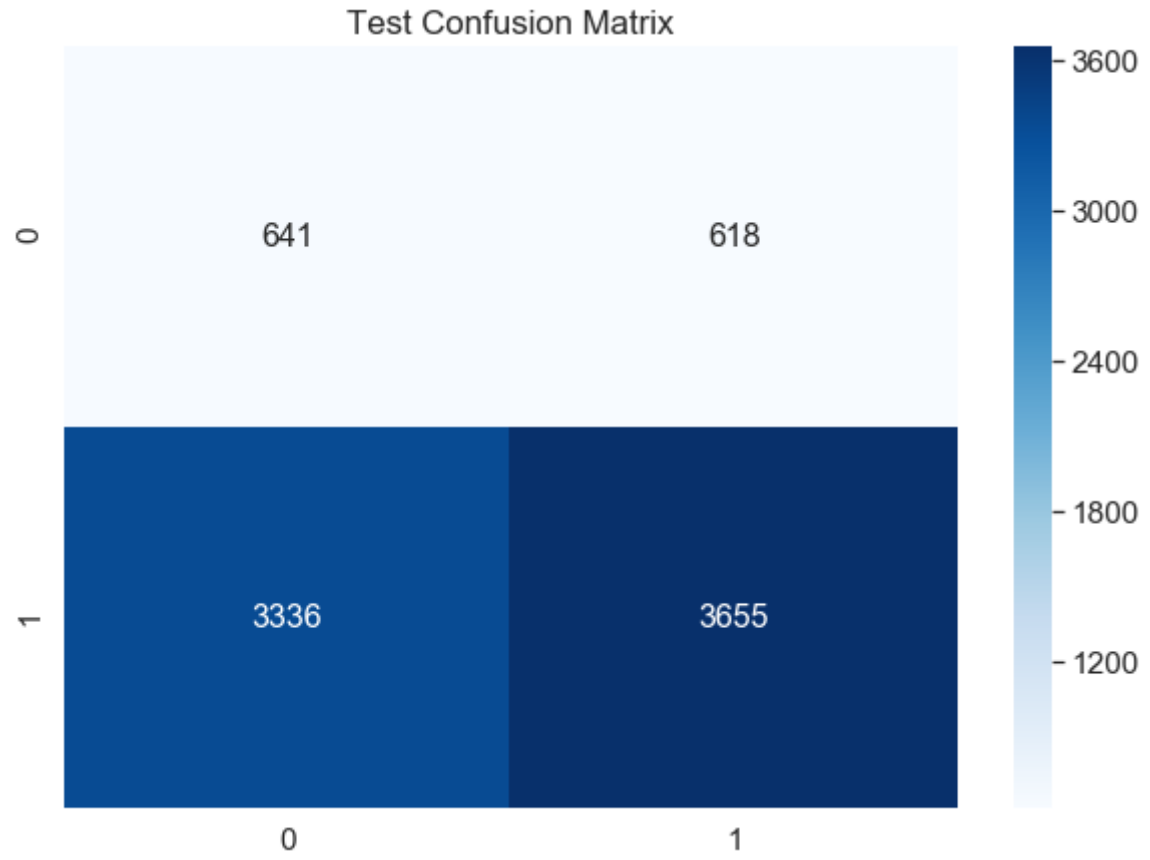


In [149]:

```
#Test Confusion matrix
df_cm = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), cc

plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
plt.title('Test Confusion Matrix')
sns.heatmap(df_cm, cmap="Blues", annot=True, fmt='g', annot_kws={"size": 16})
```

Out[149]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x26401a7ed30>
```



3. Conclusions

In [75]:

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

In [127]:

```
from prettytable import PrettyTable

x = PrettyTable()

x. field_names = ['Vectorizer', 'Model', 'HyperParameter', 'AUC']

x.add_row(['BOW', 'Brute', '101','0.64'])
x.add_row(['TFIDF', 'Brute', '103', '0.55'])
x.add_row(['AVG W2V', 'Brute', '103', '0.54'])
x.add_row(['TFIDF W2V', 'Brute', '105', '0.59'])
x.add_row(['TFIDF with 2k points', 'Brute', '103', '0.53'])

print(x)
```

```
+----------------------+-------+----------------+------+
|      Vectorizer      | Model | HyperParameter | AUC  |
+----------------------+-------+----------------+------+
|         BOW          | Brute |      101       | 0.64 |
|        TFIDF         | Brute |      103       | 0.55 |
|       AVG W2V        | Brute |      103       | 0.54 |
|      TFIDF W2V       | Brute |      105       | 0.59 |
| TFIDF with 2k points | Brute |      103       | 0.53 |
+----------------------+-------+----------------+------+
```

In [ ]: