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`

Feature	Desc
`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`
`school_state`	State where school is located (Two-letter U.S. post (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 **Exar** • `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-( 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 **Examp**

[*] 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`

Feature	Description
`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 [24]:

```
%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
```

1.1 Reading Data

In [25]:

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

In [26]:

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

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' '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 [27]:

```
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[27]:

	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

In [28]:

```

project_grade_category = []

for i in range(len(project_data)):
    a = project_data["project_grade_category"][i].replace(" ", "_")
    project_grade_category.append(a)


```

In [29]:

```
project_grade_category[0:5]
```

Out[29]:

```
['Grades_PreK-2', 'Grades_6-8', 'Grades_6-8', 'Grades_PreK-2', 'Grades_PreK-
2']
```

In [30]:

```

project_data.drop(['project_grade_category'], axis=1, inplace=True)
project_data["project_grade_category"] = project_grade_category
```

1.1 Sorted by time

In [31]:

```
#https://stats.stackexchange.com/questions/341312/train-test-split-with-time-and-person
# 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.co


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/40840
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[31]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	00:
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	00:

In [ ]:

```

```

1.2 Adding resource data in dataframe

In [32]:

```
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[32]:

	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

In [33]:

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

In [34]:

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

Out[34]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Dat
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-2 00:27:3
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-2 00:31:2

In [35]:

```
project_data = project_data.sample(n=50000)
#project_data=project_data.tail(1000)
project_data.shape
```

Out[35]:

```
(50000, 19)
```

In [36]:

```
project_data.shape
```

Out[36]:

```
(50000, 19)
```

1.2 preprocessing of `project_subject_categories`

In [37]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/4

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-str
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-pyt
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", "Warmt
        if 'The' in j.split(): # this will split each of the catogory based on space "
            j=j.replace('The','') # if we have the words "The" we are going to replace
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing sp
        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 [38]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/4

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

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", "Warmt
        if 'The' in j.split(): # this will split each of the catogory based on space "
            j=j.replace('The','') # if we have the words "The" we are going to replace
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing sp
        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/22898595/408403
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]))
```

In [39]:

```
project_data.columns
```

Out[39]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', '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',
       'project_grade_category', 'price', 'quantity', 'clean_categories',
       'clean_subcategories'],
      dtype='object')
```

In [ ]:

```

```

1.3 Text preprocessing

In [40]:

```python
# 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 [41]:

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

Out[41]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
60490	163252	p069033	f7ac240660515b986030743e83db1766	Mrs.	LA	13
42105	3358	p224685	e1df7b66db29045e36fbbc6f38a69b33	Ms.	CT	19

In [42]:

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

In [43]:

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

Out[43]:

```
'# printing some random reviews\nprint(project_data[\'essay\'].values[0])\np
rint("="*50)\nprint(project_data[\'essay\'].values[150])\nprint("="*50)\npri
nt(project_data[\'essay\'].values[1000])\nprint("="*50)\nprint(project_data
[\'essay\'].values[20000])\nprint("="*50)\nprint(project_data[\'essay\'].val
ues[99999])\nprint("="*50)'
```

In [44]:

```
# 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 [45]:

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

Out[45]:

```
'sent = decontracted(project_data[\'essay\'].values[20000])\nprint(sent)\npr
int("="*50)'
```

In [47]:

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

Out[47]:

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

In [ ]:

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

In [48]:

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

1.3.1Preprocess of Preprocessing of `essay`

In [50]:

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

```
100%|███████████████████████████████████████████████████████████████████
███| 50000/50000 [00:21<00:00, 2311.59it/s]
```

In [51]:

```
# after preprocesing
#preprocessed_essays[10:]
```

In [52]:

```
project_data['preprocessed_essays'] = preprocessed_essays
project_data.drop(['essay'], axis=1, inplace=True)
```

###

1.3.2Preprocessing of `project_title`

In [53]:

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

In [54]:

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

```
100%|████████████████████████████████████████
█| 50000/50000 [00:00<00:00, 51576.55it/s]
```

In [55]:

```
# after preprocesing
#preprocessed_project_title[1000]
```

In [56]:

```
#https://stackoverflow.com/questions/26666919/add-column-in-dataframe-from-list/384907
project_data['preprocessed_project_title'] = preprocessed_project_title
project_data.drop(['project_title'], axis=1, inplace=True)
```

In [57]:

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

Out[57]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
60490	163252	p069033	f7ac240660515b986030743e83db1766	Mrs.	LA	13
42105	3358	p224685	e1df7b66db29045e36fbbc6f38a69b33	Ms.	CT	19

In [58]:

```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 60490 to 106517
Data columns (total 20 columns):
Unnamed: 0                                      50000 non-null int64
id                                              50000 non-null object
teacher_id                                      50000 non-null object
teacher_prefix                                  49998 non-null object
school_state                                    50000 non-null object
Date                                            50000 non-null datetime64[n
s]
project_essay_1                                 50000 non-null object
project_essay_2                                 50000 non-null object
project_essay_3                                 1706 non-null object
project_essay_4                                 1706 non-null object
project_resource_summary                        50000 non-null object
teacher_number_of_previously_posted_projects    50000 non-null int64
project_is_approved                             50000 non-null int64
project_grade_category                          50000 non-null object
price                                           50000 non-null float64
quantity                                        50000 non-null int64
clean_categories                                50000 non-null object
clean_subcategories                             50000 non-null object
preprocessed_essays                             50000 non-null object
preprocessed_project_title                      50000 non-null object
dtypes: datetime64[ns](1), float64(1), int64(4), object(14)
memory usage: 8.0+ MB
```

Apply KNN

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

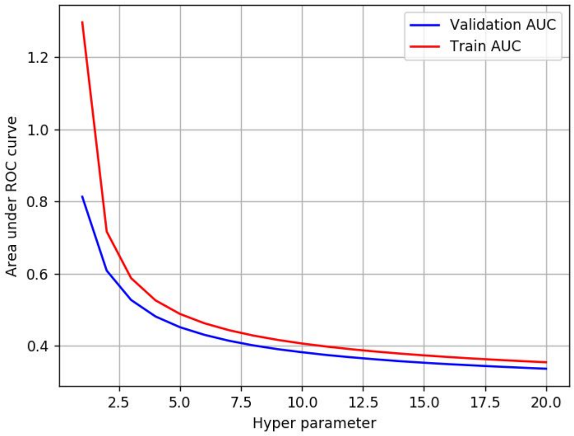
    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
    - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
    - Set 4: categorical,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

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

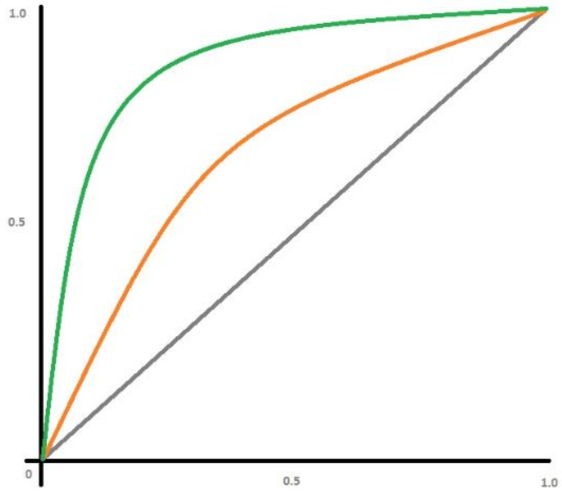
    - Find the best hyper parameter which results in the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
    - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
    - Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task

3. **Representation of results**

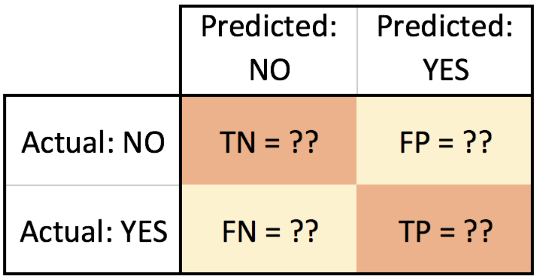
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure



- 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]**

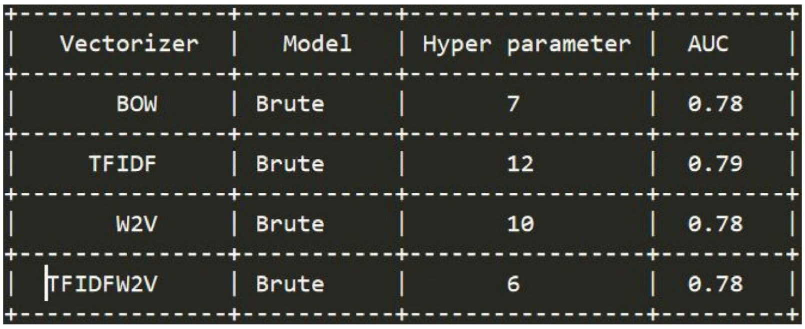
- Select top 2000 features from feature Set 2 using `SelectKBest` (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) and then apply KNN on top of these features

-
  ```
  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/)](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)](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 [59]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

from collections import Counter
from sklearn.metrics import accuracy_score

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
```

In [60]:

```
y=project_data['project_is_approved']
y.shape
```

Out[60]:

```
(50000,)
```

In [61]:

```
#replace NAN to space https ://stackoverflow.com/questions/49259305/raise-valueerrornp
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(' ')
```

In [62]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_

#split the data into train and test fo bag of words

x_t,x_test,y_t,y_test=model_selection.train_test_split(project_data,y,test_size=0.3,ra
#split train into cross val train and cross val test
x_train,x_cv,y_train,y_cv=model_selection.train_test_split(x_t,y_t,test_size=0.3,rando
```

spliting train_data into train and cross validation in ratio of 7/3

In [63]:

```
# please write all the code with proper documentation, and proper titles for each subse
# 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 you
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

#2.2 Make Data Model Ready: encoding numerical, categorical features

2.2.1 encoding categorical features

In [64]:

```
x_train.head(2)
```

Out[64]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
39295	76565	p155169	aa5da503348fa16bc4c92658f4a28422	Mr.	MI	11
26606	102370	p131104	6babe90a39e59f61ae6b9bd7297c41e4	Mrs.	PA	16

In [65]:

```
#one hot encoding for clean_categories
#__________________________
# 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,
vectorizer.fit(x_train['clean_categories'].values)
x_train_categories_one_hot = vectorizer.transform(x_train['clean_categories'].values)
x_cv_categories_one_hot = vectorizer.transform(x_cv['clean_categories'].values)
x_test_categories_one_hot = vectorizer.transform(x_test['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",x_train_categories_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_cv_categories_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_test_categories_one_hot.shape)
print("*"*50)

print(x_train_categories_one_hot[0:5])
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning',
'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (24500, 9)
Shape of matrix after one hot encodig  (10500, 9)
Shape of matrix after one hot encodig  (15000, 9)
**************************************************
  (0, 5)	1
  (0, 6)	1
  (1, 6)	1
  (2, 2)	1
  (2, 7)	1
  (3, 7)	1
  (4, 7)	1
```

In [66]:

```
#one hot encoding for clean_subcategories
#_________________________
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=Fa
vectorizer.fit(x_train['clean_subcategories'].values)
x_train_subcategories_one_hot = vectorizer.transform(x_train['clean_subcategories'].va
x_cv_subcategories_one_hot = vectorizer.transform(x_cv['clean_subcategories'].values)
x_test_subcategories_one_hot = vectorizer.transform(x_test['clean_subcategories'].valu
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",x_train_subcategories_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_cv_subcategories_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_test_subcategories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement',
'Civics_Government', 'Extracurricular', 'ForeignLanguages', 'Warmth', 'Care_
Hunger', 'NutritionEducation', 'PerformingArts', 'SocialSciences', 'Characte
rEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_
Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness',
'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (24500, 30)
Shape of matrix after one hot encodig  (10500, 30)
Shape of matrix after one hot encodig  (15000, 30)
```

In [67]:

```
#one hot encoding for school_state


my_counter = Counter()
for state in project_data['school_state'].values:
    my_counter.update(state.split())

school_state_cat_dict = dict(my_counter)
sorted_school_state_cat_dict = dict(sorted(school_state_cat_dict.items(), key=lambda k

#_________________________
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_school_state_cat_dict.keys()), low
vectorizer.fit(x_train['school_state'].values)
x_train_school_state_one_hot = vectorizer.transform(x_train['school_state'].values)
x_cv_school_state_one_hot = vectorizer.transform(x_cv['school_state'].values)
x_test_school_state_one_hot = vectorizer.transform(x_test['school_state'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",x_train_school_state_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_cv_school_state_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_test_school_state_one_hot.shape)
```

```
['WY', 'VT', 'ND', 'MT', 'AK', 'RI', 'SD', 'NE', 'DE', 'NH', 'ME', 'HI', 'D
C', 'WV', 'NM', 'IA', 'KS', 'ID', 'AR', 'CO', 'MN', 'OR', 'NV', 'KY', 'MS',
'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ', 'NJ', 'OK', 'LA', 'MA', 'W
A', 'OH', 'MO', 'IN', 'MI', 'PA', 'GA', 'SC', 'IL', 'NC', 'FL', 'NY', 'TX',
'CA']
Shape of matrix after one hot encodig  (24500, 51)
Shape of matrix after one hot encodig  (10500, 51)
Shape of matrix after one hot encodig  (15000, 51)
```

In [68]:

```
#one hot encoding for project_grade_category

my_counter = Counter()
for project_grade in project_data['project_grade_category'].values:
    my_counter.update(project_grade.split())


project_grade_cat_dict = dict(my_counter)
sorted_project_grade_cat_dict = dict(sorted(project_grade_cat_dict.items(), key=lambda
#_________________________
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_cat_dict.keys()), lo
vectorizer.fit(x_train['project_grade_category'].values)
x_train_grade_category_one_hot = vectorizer.transform(x_train['project_grade_category'
x_cv_grade_category_one_hot = vectorizer.transform(x_cv['project_grade_category'].valu
x_test_grade_category_one_hot = vectorizer.transform(x_test['project_grade_category'].
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",x_train_grade_category_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_cv_grade_category_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_test_grade_category_one_hot.shape)
```

```
['Grades_9-12', 'Grades_6-8', 'Grades_3-5', 'Grades_PreK-2']
Shape of matrix after one hot encodig  (24500, 4)
Shape of matrix after one hot encodig  (10500, 4)
Shape of matrix after one hot encodig  (15000, 4)
```

In [69]:

```
#one hot encoding for prefix_category

my_counter = Counter()
for teacher_prefix in project_data['teacher_prefix'].values:
    teacher_prefix = str(teacher_prefix)
    my_counter.update(teacher_prefix.split())

teacher_prefix_cat_dict = dict(my_counter)
sorted_teacher_prefix_cat_dict = dict(sorted(teacher_prefix_cat_dict.items(), key=lamb



#_________________________
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_cat_dict.keys()), l
vectorizer.fit(x_train['teacher_prefix'].values)
x_train_prefix_one_hot = vectorizer.transform(x_train['teacher_prefix'].values)
x_cv_prefix_one_hot = vectorizer.transform(x_cv['teacher_prefix'].values)
x_test_prefix_one_hot = vectorizer.transform(x_test['teacher_prefix'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",x_train_prefix_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_cv_prefix_one_hot.shape)
print("Shape of matrix after one hot encodig ",x_test_prefix_one_hot.shape)
```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encodig  (24500, 5)
Shape of matrix after one hot encodig  (10500, 5)
Shape of matrix after one hot encodig  (15000, 5)
```

In [70]:

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

2.2.2 encoding numerical features

In [71]:

```
x_train.head(2)
```

Out[71]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
39295	76565	p155169	aa5da503348fa16bc4c92658f4a28422	Mr.	MI	11
26606	102370	p131104	6babe90a39e59f61ae6b9bd7297c41e4	Mrs.	PA	16

In [72]:

```
#price standardization of x_train data
#---------------------------------------
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.pr
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.   ...
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()
price_scalar.fit(x_train['price'].values.reshape(-1,1)) # finding the mean and standar
x_train_price_standardized=price_scalar.transform(x_train['price'].values.reshape(-1,1
x_cv_price_standardized = price_scalar.transform(x_cv['price'].values.reshape(-1, 1))
x_test_price_standardized = price_scalar.transform(x_test['price'].values.reshape(-1, 

print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_

print("After standardization")
print(x_train_price_standardized.shape, y_train.shape)
print(x_cv_price_standardized.shape, y_cv.shape)
print(x_test_price_standardized.shape, y_test.shape)

# Now standardize the data with above maen and variance.
#x_train_price_standardized = price_scalar.transform(x_train['price'].values.reshape(-
```

```
Mean : 297.4391379591837, Standard deviation : 355.84486777404317
After standardization
(24500, 1) (24500,)
(10500, 1) (10500,)
(15000, 1) (15000,)
```

2.2.3 merge numerical and categorical data

In [73]:

```
# 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 matri
x_train_ohe = hstack((x_train_categories_one_hot, x_train_subcategories_one_hot, x_tra
x_cv_ohe = hstack((x_cv_categories_one_hot, x_cv_subcategories_one_hot, x_cv_school_st
x_test_ohe = hstack((x_test_categories_one_hot, x_test_subcategories_one_hot, x_test_s

print(x_train_ohe.shape)
print(x_cv_ohe.shape)
print(x_test_ohe.shape)
```

```
(24500, 100)
(10500, 100)
(15000, 100)
```

In [74]:

```
type(x_train_ohe)
```

Out[74]:

```
scipy.sparse.csr.csr_matrix
```

2.3 Make Data Model Ready: encoding eassay, and project_title

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

In [75]:

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

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

2.4.1 Applying KNN brute force on BOW, SET 1

vectorize the essay and title data, SET 1

In [76]:

```
#you can vectorize the essay
#______________________________
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.Cou

# We are considering only the words which appeared in at least 10 documents(rows or pro
vectorizer = CountVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer.fit(x_train['preprocessed_essays'].values)# fit has to apply only on train d

# we use fitted CountVectorizer to convert the text to vector
x_train_bow_essays = vectorizer.transform(x_train['preprocessed_essays'].values)
x_cv_bow_essays = vectorizer.transform(x_cv['preprocessed_essays'].values)
x_test_bow_essays = vectorizer.transform(x_test['preprocessed_essays'].values)

print("Shape of matrix after one hot encodig ",x_train_bow_essays.shape, y_train.shape
print("Shape of matrix after one hot encodig ",x_cv_bow_essays.shape)
print("Shape of matrix after one hot encodig ",x_test_bow_essays.shape)
```

```
Shape of matrix after one hot encodig  (24500, 5000) (24500,)
Shape of matrix after one hot encodig  (10500, 5000)
Shape of matrix after one hot encodig  (15000, 5000)
```

In [77]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.Cour
#you can vectorize the title
# We are considering only the words which appeared in at least 10 documents(rows or pro
vectorizer = CountVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer.fit(x_train['preprocessed_project_title'].values)# fit has to apply only on

# we use fitted CountVectorizer to convert the text to vector
x_train_bow_title = vectorizer.transform(x_train['preprocessed_project_title'].values)
x_cv_bow_title = vectorizer.transform(x_cv['preprocessed_project_title'].values)
x_test_bow_title = vectorizer.transform(x_test['preprocessed_project_title'].values)

print("Shape of matrix after one hot encodig ",x_train_bow_title.shape)
print("Shape of matrix after one hot encodig ",x_cv_bow_title.shape)
print("Shape of matrix after one hot encodig ",x_test_bow_title.shape)
```

```
Shape of matrix after one hot encodig  (24500, 2142)
Shape of matrix after one hot encodig  (10500, 2142)
Shape of matrix after one hot encodig  (15000, 2142)
```

In [78]:

```
x_train.columns
```

Out[78]:

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

In [79]:

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

merge dataset, SET 1

In [80]:

```python
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
#https://stackoverflow.com/questions/30163830/accessing-elements-in-coo-matrix
from scipy.sparse import hstack
from scipy.sparse import coo_matrix
# with the same hstack function we are concatinating a sparse matrix and a dense matir
x_train_bow = hstack((x_train_ohe, x_train_bow_essays, x_train_bow_title)).tocsr()
x_cv_bow = hstack((x_cv_ohe, x_cv_bow_essays, x_cv_bow_title)).tocsr()
x_test_bow = hstack((x_test_ohe, x_test_bow_essays, x_test_bow_title)).tocsr()

print(x_train_bow.shape, y_train.shape)
print(x_cv_bow.shape)
print(x_test_bow.shape)
```

```
(24500, 7242) (24500,)
(10500, 7242)
(15000, 7242)
```

In [ ]:

```python
type(x_train_bow)
```

Hyperparameter tuning by AUC plot for cv and train dataset, SET 1

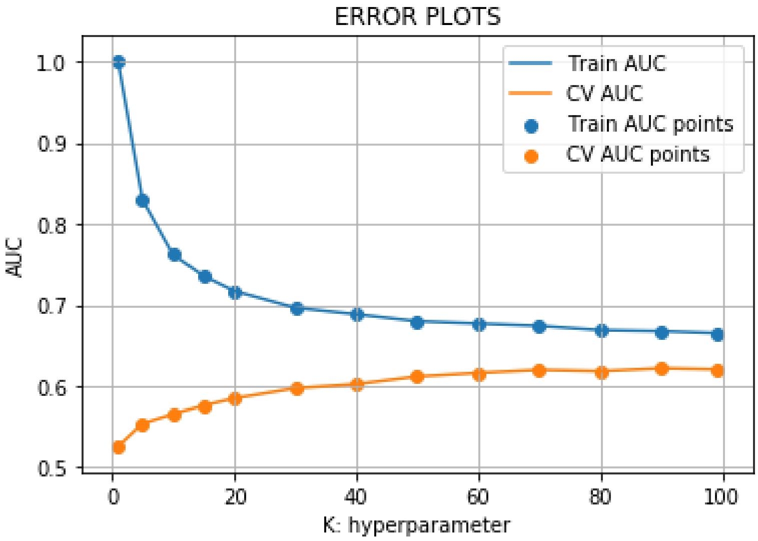
In [81]:

```python
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
    # 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 =
    # 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
from scipy.sparse import coo_matrix
"""
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 va
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 99]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(x_train_bow, y_train)


    y_train_pred = batch_predict(neigh, x_train_bow)
    y_cv_pred = batch_predict(neigh, x_cv_bow)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
    # 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%|████████████████████████████████████████
█████| 13/13 [52:58<00:00, 245.86s/it]
```



PARAMETER TUNING USING GRID SEARCH

In [ ]:

```
'''# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSear
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier()

grid_val = {'n_neighbors':[1, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 99]}

clf = GridSearchCV(neigh, grid_val, cv= 5, scoring='roc_auc')
clf.fit(x_train_bow, y_train)

results_grid_bow = pd.DataFrame.from_dict(clf.cv_results_).sort_values(['param_n_neighb


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

plt.plot(grid_val['n_neighbors'], train_auc, label='Train AUC')
# code reference: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(grid_val['n_neighbors'],train_auc - train_auc_std,train_auc + tr

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

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

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

results_grid_bow.head()'''
```

In [ ]:

```

```

PARAMETER TUNING USING Random search

In [ ]:

```
'''# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSear
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

neigh = KNeighborsClassifier(n_jobs=-1)
parameters = {'n_neighbors':sp_randint(40, 100)}
clf = RandomizedSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(x_train_bow, y_train)

results_rand_bow = pd.DataFrame.from_dict(clf.cv_results_).sort_values(['param_n_neighb


train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_n_neighbors']

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0

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

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("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results_rand_bow.head()'''
```

In [128]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and go
# Note: based on the method you use you might get different hyperparameter values as be
# so, you choose according to the method you choose, you use gridsearch if you are hav
# if you increase the cv values in the GridSearchCV you will get more rebust results.

#here we are choosing the best_k based on forloop results
opt_k_bow=90
```

Apply best hyperparameter on test dataset, SET 1

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

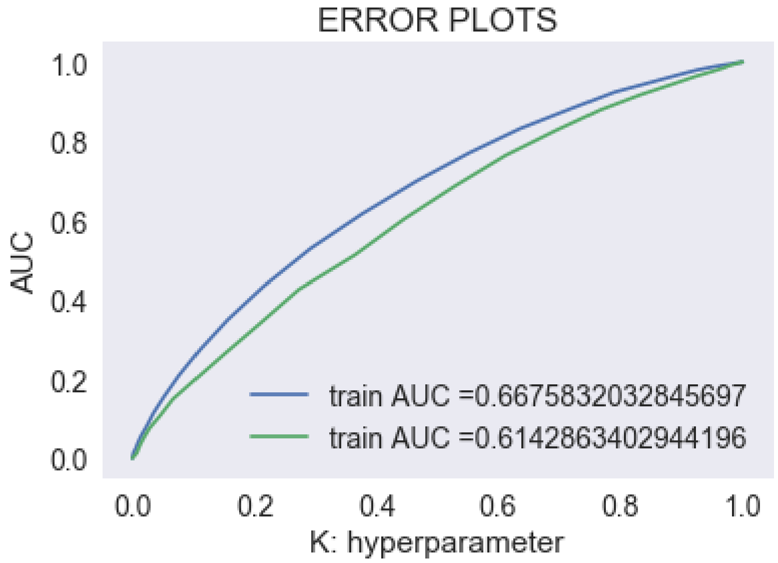
Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find

the AUC on test data and plot the ROC curve on both train and test using model-M.
Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

IF Your system is getting stuck when you are working with the Knn: YOU NEED TO USE BATCH WISE PREDICTION

In [129]:

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


neigh = KNeighborsClassifier(n_neighbors=opt_k_bow, n_jobs=-1)
neigh.fit(x_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
# not the predicted outputs

y_train_pred = batch_predict(neigh, x_train_bow)
y_test_pred = batch_predict(neigh, x_test_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)
tain_auc_bow=auc(train_fpr, train_tpr)
test_auc_bow=auc(test_fpr, test_tpr)

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



In [ ]:

```

```

In [ ]:

```
'''knn1 = KNeighborsClassifier(n_neighbors=opt_k_bow,algorithm='brute',weights='unifor
knn1.fit(x_train_bow,y_train)

pred_prob_test = knn1.predict_proba(x_test_bow)


#AUC of train dataset
pred_prob_train = knn1.predict_proba(x_train_bow)
fpr1, tpr1, thresholds = roc_curve(y_train,pred_prob_train[:, 1])
bow_roc_auc_train = auc(fpr1, tpr1)
print("Best AUC of train: ",bow_roc_auc_train)

#AUC of test dataset
pred_prob_test = knn1.predict_proba(x_test_bow)
fpr2, tpr2, thresholds = roc_curve(y_test,pred_prob_test[:, 1])
bow_roc_auc_test = auc(fpr2, tpr2)
print("Best AUC of test: ",bow_roc_auc_test)

#value taken from from GridsearchCV section
plt.title('Receiver Operating Characteristic')
plt.plot(fpr1, tpr1, 'r',label='AUC_train = %0.2f'% bow_roc_auc_train)
plt.plot(fpr2, tpr2, 'g',label='AUC_test = %0.2f'% bow_roc_auc_test)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

print("Best AUC of train: ",bow_roc_auc_train)
print("Best AUC of test: ",bow_roc_auc_test)'''
```

In [130]:

```
# 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.rou
    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 [131]:

```
print("="*100)
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) 0.3844324967040373 for threshold 0.811
Train confusion matrix
[[ 2309  1393]
 [ 7979 12819]]
Test confusion matrix
[[1275 1025]
 [5033 7667]]
```

In [132]:

```
#CONFUSION MATRIX
def predict(proba, threshould, fpr, tpr):

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

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

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




```

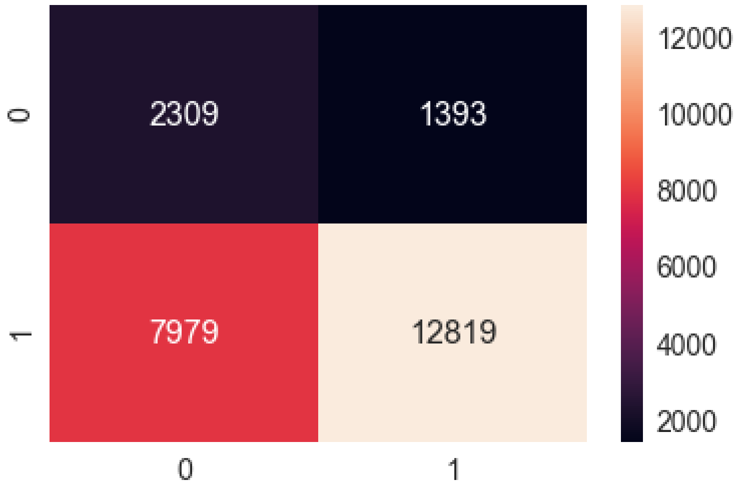
In [133]:

```
print("="*100)
print("TRAIN confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_


conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
============================================================================
========================
TRAIN confusion matrix
the maximum value of tpr*(1-fpr) 0.3844324967040373 for threshold 0.811
[[ 2309  1393]
 [ 7979 12819]]
the maximum value of tpr*(1-fpr) 0.3844324967040373 for threshold 0.811
```

Out[133]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea2ad41860>
```



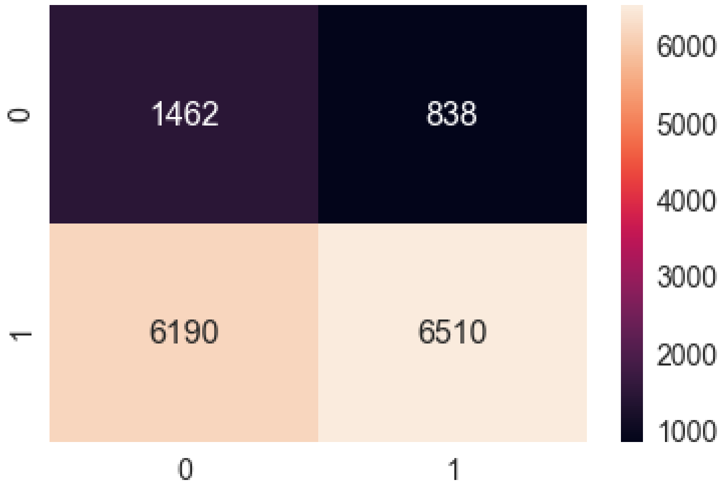
In [134]:

```

print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr)


conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, te_th

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
============================================================================
========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.33466021910304694 for threshold 0.822
[[1462  838]
 [6190 6510]]
the maximum value of tpr*(1-fpr) 0.33466021910304694 for threshold 0.822
```

Out[134]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea20f59c18>
```



In [ ]:

```

```

Feature selection with `SelectKBest` (top 2000) , SET 1

In [135]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif
#https://stackoverflow.com/questions/49300193/feature-selection-f-classif-scikit-learn
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import f_classif

x_train_bow_2000 = SelectKBest(f_classif, k=2000).fit_transform(x_train_bow, y_train)
x_cv_bow_2000 = SelectKBest(f_classif, k=2000).fit_transform(x_cv_bow, y_cv)
x_test_bow_2000 = SelectKBest(f_classif, k=2000).fit_transform(x_test_bow, y_test)
print(x_train_bow_2000.shape)
print(x_cv_bow_2000.shape)
print(x_test_bow_2000.shape)


```

```
(24500, 2000)
(10500, 2000)
(15000, 2000)
```

In [136]:

```
type(x_train_bow_2000)
```

Out[136]:

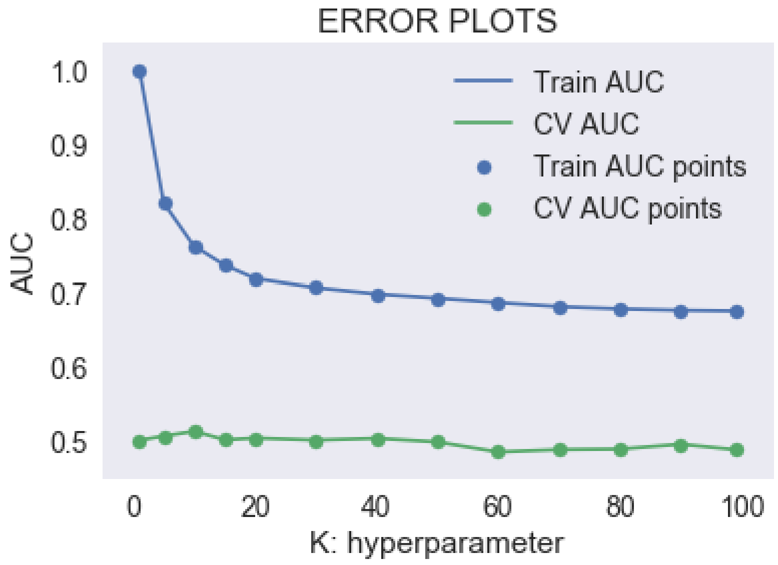
```
scipy.sparse.csr.csr_matrix
```

Parameter tuning using Gridsearch

In [137]:

```

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 99]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(x_train_bow_2000, y_train)


    y_train_pred = batch_predict(neigh, x_train_bow_2000)
    y_cv_pred = batch_predict(neigh, x_cv_bow_2000)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
    # 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%|█████████████████████████████████████████████████████████████████
█████████| 13/13 [54:14<00:00, 256.43s/it]
```



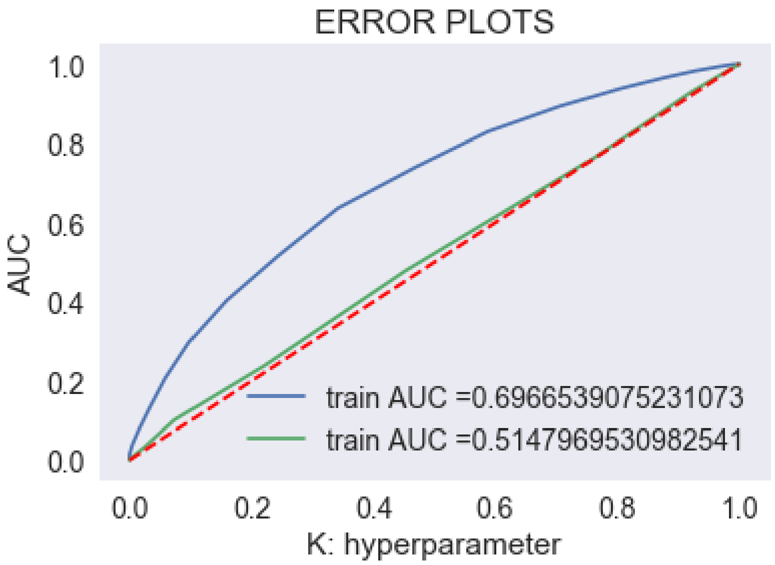
In [138]:

```
opt_k_bow_2000=40
```

Applying on Test data and ROC

In [139]:

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


neigh = KNeighborsClassifier(n_neighbors=opt_k_bow_2000, n_jobs=-1)
neigh.fit(x_train_bow_2000, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
# not the predicted outputs

y_train_pred = batch_predict(neigh, x_train_bow_2000)
y_test_pred = batch_predict(neigh, x_test_bow_2000)

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)
tain_auc_bow_2000=auc(train_fpr, train_tpr)
test_auc_bow_2000=auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.plot([0,1],[0,1],'r--')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [140]:

```
#CONFUSION MATRIX
def predict(proba, threshould, fpr, tpr):

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

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

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



```

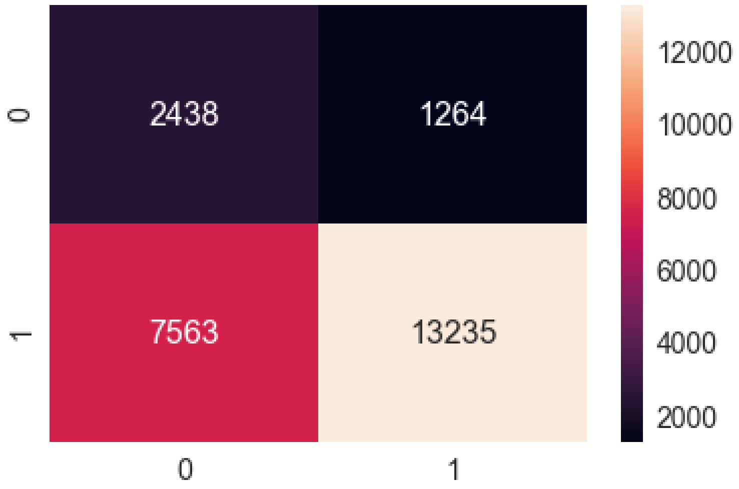
In [141]:

```
print("="*100)
print("TRAIN confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_


conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
============================================================================
========================
TRAIN confusion matrix
the maximum value of tpr*(1-fpr) 0.41908262799445295 for threshold 0.775
[[ 2438  1264]
 [ 7563 13235]]
the maximum value of tpr*(1-fpr) 0.41908262799445295 for threshold 0.775
```

Out[141]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea2ad39ba8>
```



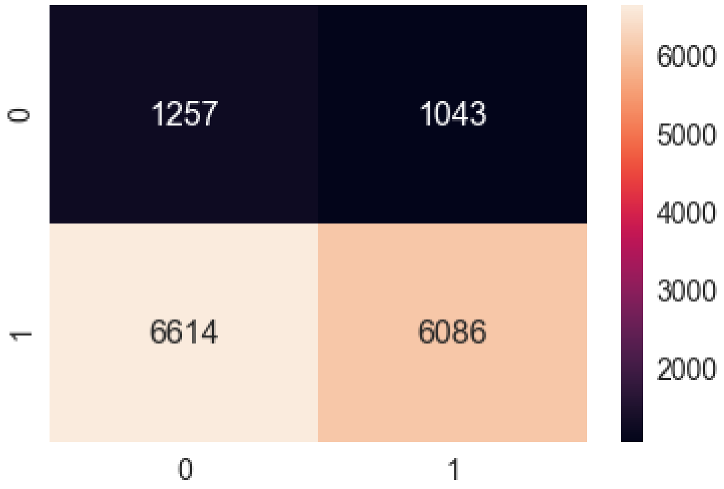
In [142]:

```

print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr)


conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, te_th

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
============================================================================
========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2619001027045532 for threshold 0.675
[[1257 1043]
 [6614 6086]]
the maximum value of tpr*(1-fpr) 0.2619001027045532 for threshold 0.675
```

Out[142]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea21a7fc18>
```



2.4.2 Applying KNN brute force on TFIDF, SET 2

TFIDF Vectorizing essy and title variable, SET 2

In [ ]:

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

In [83]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf = TfidfVectorizer(min_df=10,  ngram_range=(1,4), max_features=5000)
vectorizer_tfidf.fit(x_train['preprocessed_essays'].values)# fit has to apply only on 

# we use fitted CountVectorizer to convert the text to vector
x_train_tfidf_essays = vectorizer.transform(x_train['preprocessed_essays'].values)
x_cv_tfidf_essays = vectorizer.transform(x_cv['preprocessed_essays'].values)
x_test_tfidf_essays = vectorizer.transform(x_test['preprocessed_essays'].values)

print("Shape of matrix after one hot encodig ",x_train_tfidf_essays.shape, y_train.sha
print("Shape of matrix after one hot encodig ",x_cv_tfidf_essays.shape)
print("Shape of matrix after one hot encodig ",x_test_tfidf_essays.shape)
```

```
Shape of matrix after one hot encodig  (24500, 2142) (24500,)
Shape of matrix after one hot encodig  (10500, 2142)
Shape of matrix after one hot encodig  (15000, 2142)
```

In [84]:

```
#TFIDF Vectorizer on `project_title`

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer_tfidf.fit(x_train['preprocessed_project_title'].values)# fit has to apply o

# we use fitted CountVectorizer to convert the text to vector
x_train_tfidf_title = vectorizer.transform(x_train['preprocessed_project_title'].value
x_cv_tfidf_title = vectorizer.transform(x_cv['preprocessed_project_title'].values)
x_test_tfidf_title = vectorizer.transform(x_test['preprocessed_project_title'].values)

print("Shape of matrix after one hot encodig ",x_train_tfidf_title.shape)
print("Shape of matrix after one hot encodig ",x_cv_tfidf_title.shape)
print("Shape of matrix after one hot encodig ",x_test_tfidf_title.shape)


```

```
Shape of matrix after one hot encodig  (24500, 2142)
Shape of matrix after one hot encodig  (10500, 2142)
Shape of matrix after one hot encodig  (15000, 2142)
```

merge all sparse data, SET 2

In [85]:

```
# 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 matir
x_train_tfidf = hstack((x_train_ohe, x_train_tfidf_essays, x_train_tfidf_title)).tocsr
x_cv_tfidf = hstack((x_cv_ohe, x_cv_tfidf_essays, x_cv_tfidf_title)).tocsr()
x_test_tfidf = hstack((x_test_ohe, x_test_tfidf_essays, x_test_tfidf_title)).tocsr()

print(x_train_tfidf.shape)
print(x_cv_tfidf.shape)
print(x_test_tfidf.shape)
```

```
(24500, 4384)
(10500, 4384)
(15000, 4384)
```

In [86]:

```
type(x_train_tfidf)
```

Out[86]:

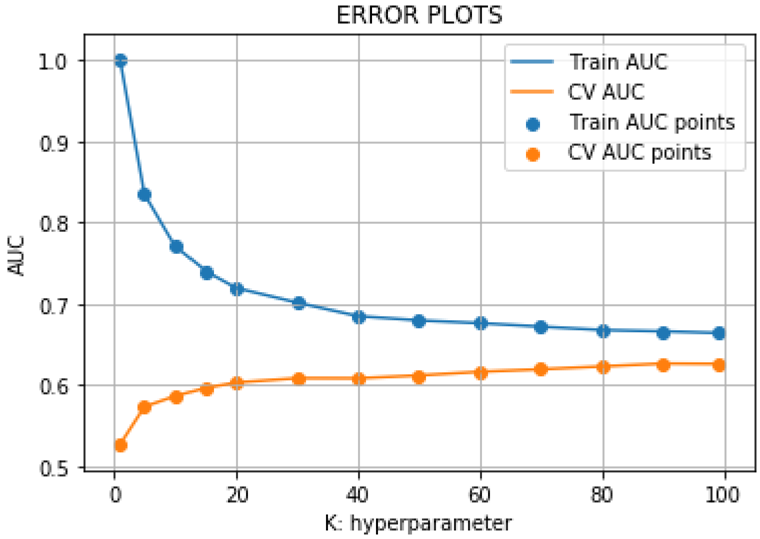
```
scipy.sparse.csr.csr_matrix
```

Hyperparameter tuning by AUC plot for cv and train dataset, SET 2

In [87]:

```python

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 99]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(x_train_tfidf, y_train)


    y_train_pred = batch_predict(neigh, x_train_tfidf)
    y_cv_pred = batch_predict(neigh, x_cv_tfidf)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
    # 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%|██████████████████████████████████████████████████████
████████| 13/13 [49:12<00:00, 222.23s/it]
```



In [123]:

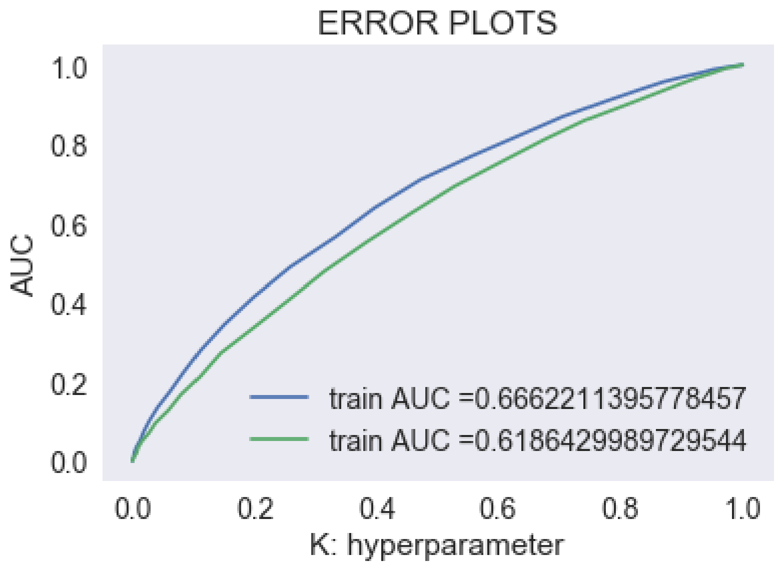
```python
opt_k_tfidf=90
```

Apply best hyperparameter on test dataset, SET 2

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

In [124]:

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

neigh = KNeighborsClassifier(n_neighbors=opt_k_tfidf, n_jobs=-1)
neigh.fit(x_train_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
# not the predicted outputs

y_train_pred = batch_predict(neigh, x_train_tfidf)
y_test_pred = batch_predict(neigh, x_test_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)
tain_auc_tfidf=auc(train_fpr, train_tpr)
test_auc_tfidf=auc(test_fpr, test_tpr)

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



CONFUSION MATRIX

In [125]:

```
#CONFUSION MATRIX
def predict(proba, threshould, fpr, tpr):

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

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

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




```

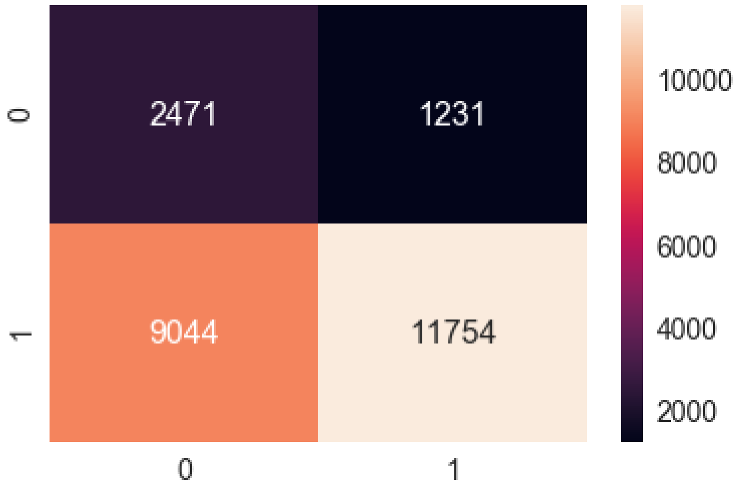
In [126]:

```
print("="*100)
print("TRAIN confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_


conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
============================================================================
========================
TRAIN confusion matrix
the maximum value of tpr*(1-fpr) 0.3852374534828573 for threshold 0.811
[[ 2471  1231]
 [ 9044 11754]]
the maximum value of tpr*(1-fpr) 0.3852374534828573 for threshold 0.811
```

Out[126]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea218deb00>
```



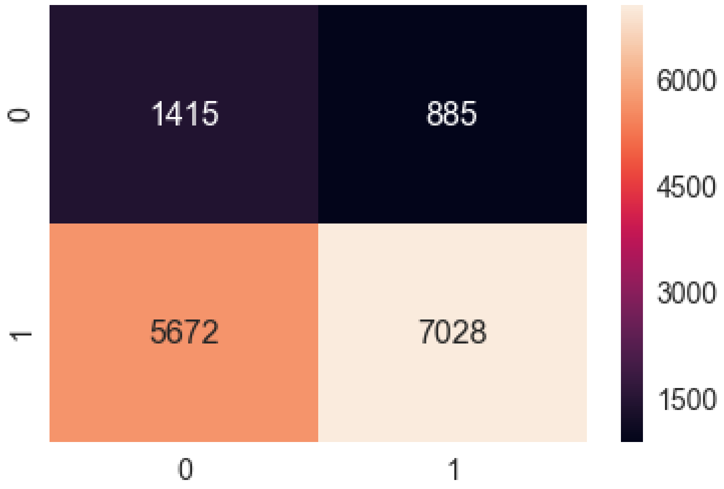
In [127]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr)


conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, te_th

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
============================================================================
========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3404525847312564 for threshold 0.811
[[1415  885]
 [5672 7028]]
the maximum value of tpr*(1-fpr) 0.3404525847312564 for threshold 0.811
```

Out[127]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea211326d8>
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

vectorize using AVG W2V, SET 3

In [ ]:

```
'''
# 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-

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


'''
```

In [ ]:

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

In [88]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-
# 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 [89]:

```
# Using Pretrained Models: AVG W2V on `essay`
#_______________________________________________

# -----average Word2Vec on train
# compute average word2vec for each review.
avg_w2v_vectors_essays_train = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(x_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_essays_train.append(vector)

print(len(avg_w2v_vectors_essays_train))
print(len(avg_w2v_vectors_essays_train[0]))








```

```
100%|███████████████████████████████████████████████████████████████████████
███| 24500/24500 [00:05<00:00, 4697.52it/s]

24500
300
```

In [90]:

```
# -----average Word2Vec on CV
# compute average word2vec for each review.
avg_w2v_vectors_essays_cv = []; # the avg-w2v for each sentence/review is stored in th
for sentence in tqdm(x_cv['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_essays_cv.append(vector)

print(len(avg_w2v_vectors_essays_cv))
print(len(avg_w2v_vectors_essays_cv[0]))
```

```
100%|███████████████████████████████████████████████████████████████████████
███| 10500/10500 [00:02<00:00, 4718.00it/s]

10500
300
```

In [91]:

```
# -----average Word2Vec on test
# compute average word2vec for each review.
avg_w2v_vectors_essays_test = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(x_test['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_essays_test.append(vector)

print(len(avg_w2v_vectors_essays_test))
print(len(avg_w2v_vectors_essays_test[0]))
```

```
100%|███████████████████████████████████████████████████████████████████████
███| 15000/15000 [00:03<00:00, 4661.69it/s]

15000
300
```

In [92]:

```
# Using Pretrained Models: AVG W2V on `project_title`
#_____________________________________________________

# ------average Word2Vec on train
# compute average word2vec for each review.
avg_w2v_vectors_project_title_train = []; # the avg-w2v for each sentence/review is st
for sentence in tqdm(x_train['preprocessed_project_title']): # for each review/sentenc
    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_project_title_train.append(vector)

print(len(avg_w2v_vectors_project_title_train))
print(len(avg_w2v_vectors_project_title_train[0]))
```

```
100%|████████████████████████████████████████
█| 24500/24500 [00:00<00:00, 87814.06it/s]

24500
300
```

In [93]:

```
# ------average Word2Vec on cv
# compute average word2vec for each review.
avg_w2v_vectors_project_title_cv = []; # the avg-w2v for each sentence/review is store
for sentence in tqdm(x_cv['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_project_title_cv.append(vector)

print(len(avg_w2v_vectors_project_title_cv))
print(len(avg_w2v_vectors_project_title_cv[0]))
```

```
100%|████████████████████████████████████████
█| 10500/10500 [00:00<00:00, 84210.73it/s]

10500
300
```

In [94]:

```
# ------average Word2Vec on test
# compute average word2vec for each review.
avg_w2v_vectors_project_title_test = []; # the avg-w2v for each sentence/review is sto
for sentence in tqdm(x_test['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_project_title_test.append(vector)

print(len(avg_w2v_vectors_project_title_test))
print(len(avg_w2v_vectors_project_title_test[0]))
```

```
100%|████████████████████████████████████████████████
█| 15000/15000 [00:00<00:00, 86899.88it/s]

15000
300
```

merge all sparse data, SET 3

In [95]:

```
# 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 matir
x_train_AVGW2V = hstack((x_train_ohe, avg_w2v_vectors_essays_train, avg_w2v_vectors_pr
x_cv_AVGW2V = hstack((x_cv_ohe, avg_w2v_vectors_essays_cv, avg_w2v_vectors_project_tit
x_test_AVGW2V = hstack((x_test_ohe, avg_w2v_vectors_essays_test, avg_w2v_vectors_proje

print(x_train_AVGW2V.shape)
print(x_cv_AVGW2V.shape)
print(x_test_AVGW2V.shape)
```

```
(24500, 700)
(10500, 700)
(15000, 700)
```

In [96]:

```
type(x_train_AVGW2V)
```

Out[96]:

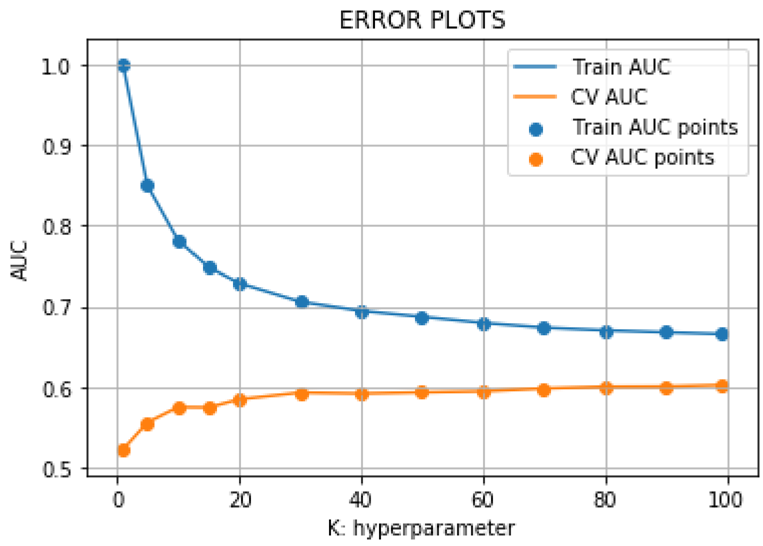
```
scipy.sparse.csr.csr_matrix
```

Hyperparameter tuning by AUC plot for cv and train dataset, SET 3

In [97]:

```python

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 99]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(x_train_AVGW2V, y_train)


    y_train_pred = batch_predict(neigh, x_train_AVGW2V)
    y_cv_pred = batch_predict(neigh, x_cv_AVGW2V)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
    # 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%|████████████████████████████████████████████████
█████| 13/13 [2:10:49<00:00, 609.83s/it]
```



In [118]:

```python
opt_k_AVGW2V=90
```

In [ ]:

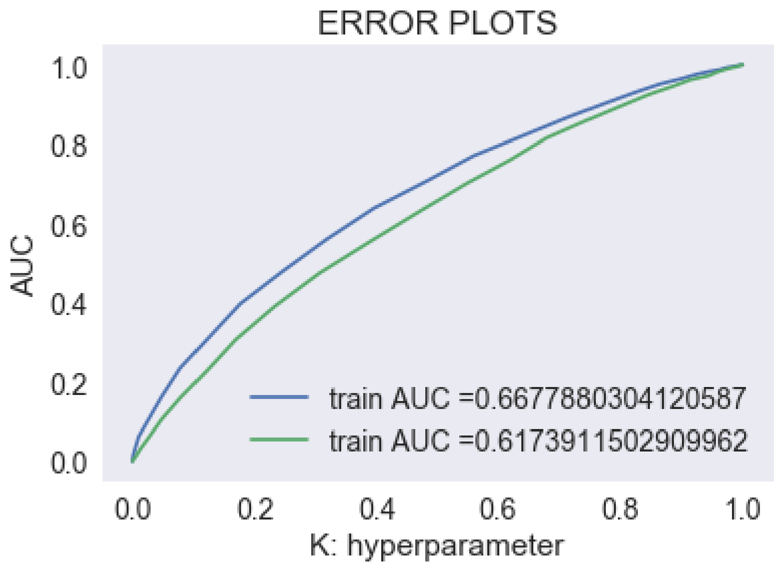
```python

```

Apply best hyperparameter on test dataset, SET 3

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

In [119]:

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

neigh = KNeighborsClassifier(n_neighbors=opt_k_AVGW2V, n_jobs=-1)
neigh.fit(x_train_AVGW2V, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
# not the predicted outputs

y_train_pred = batch_predict(neigh, x_train_AVGW2V)
y_test_pred = batch_predict(neigh, x_test_AVGW2V)

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)
tain_auc_AVGW2V=auc(train_fpr, train_tpr)
test_auc_AVGW2V=auc(test_fpr, test_tpr)

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



CONFUSION MATRIX, SET 3

In [120]:

```
#CONFUSION MATRIX
def predict(proba, threshould, fpr, tpr):

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

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

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



```

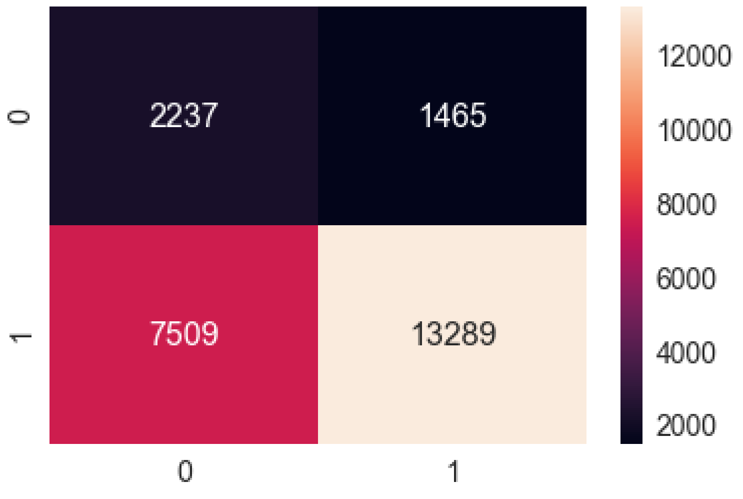
In [121]:

```
print("="*100)
print("TRAIN confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_


conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
============================================================================
========================
TRAIN confusion matrix
the maximum value of tpr*(1-fpr) 0.386100440609835 for threshold 0.856
[[ 2237  1465]
 [ 7509 13289]]
the maximum value of tpr*(1-fpr) 0.386100440609835 for threshold 0.856
```

Out[121]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea218f6cf8>
```



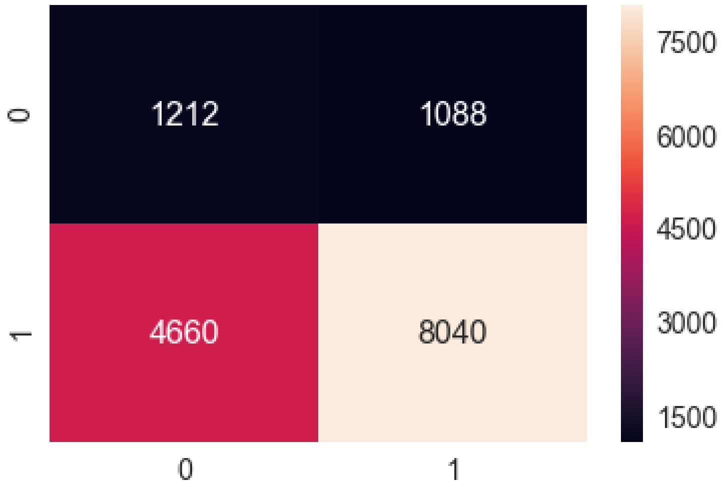
In [122]:

```

print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr)


conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, te_th

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
============================================================================
========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3386686751112633 for threshold 0.856
[[1212 1088]
 [4660 8040]]
the maximum value of tpr*(1-fpr) 0.3386686751112633 for threshold 0.856
```

Out[122]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea2191c470>
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

Vectorize using TFIDF W2V, SET 4

In [ ]:

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

In [143]:

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

In [144]:

```
#Using Pretrained Models: TFIDFW weighted W2V on `essay
#________________________________________

# average Word2Vec---train
# compute average word2vec for each review.
tfidf_w2v_vectors_essays_train = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(x_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((se
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # g
            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_train.append(vector)

print(len(tfidf_w2v_vectors_essays_train))
print(len(tfidf_w2v_vectors_essays_train[0]))
```

```
100%|████████████████████████████████████████
██| 24500/24500 [00:42<00:00, 579.65it/s]

24500
300
```

In [145]:

```
# average Word2Vec---cv
# compute average word2vec for each review.
tfidf_w2v_vectors_essays_cv = []; # the avg-w2v for each sentence/review is stored in 
for sentence in tqdm(x_cv['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((se
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # g
            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_cv.append(vector)

print(len(tfidf_w2v_vectors_essays_cv))
print(len(tfidf_w2v_vectors_essays_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
████| 10500/10500 [00:18<00:00, 582.98it/s]

10500
300
```

In [146]:

```
# average Word2Vec---test
# compute average word2vec for each review.
tfidf_w2v_vectors_essays_test = []; # the avg-w2v for each sentence/review is stored i
for sentence in tqdm(x_test['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((se
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # g
            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_test.append(vector)

print(len(tfidf_w2v_vectors_essays_test))
print(len(tfidf_w2v_vectors_essays_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
████| 15000/15000 [00:25<00:00, 592.62it/s]

15000
300
```

In [147]:

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

In [148]:

```
# average Word2Vec--train
# compute average word2vec for each review.
tfidf_w2v_vectors_project_title_train = []; # the avg-w2v for each sentence/review is :
for sentence in tqdm(x_train['preprocessed_project_title']): # for each review/sentenc
    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((sen
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # g
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_project_title_train.append(vector)

print(len(tfidf_w2v_vectors_project_title_train))
print(len(tfidf_w2v_vectors_project_title_train[0]))
```

```
100%|████████████████████████████████████████
█| 24500/24500 [00:00<00:00, 40602.14it/s]

24500
300
```

In [149]:

```
# average Word2Vec--cv
# compute average word2vec for each review.
tfidf_w2v_vectors_project_title_cv = []; # the avg-w2v for each sentence/review is sto
for sentence in tqdm(x_cv['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((se
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # g
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_project_title_cv.append(vector)

print(len(tfidf_w2v_vectors_project_title_cv))
print(len(tfidf_w2v_vectors_project_title_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█| 10500/10500 [00:00<00:00, 40179.61it/s]

10500
300
```

In [150]:

```
# average Word2Vec--test
# compute average word2vec for each review.
tfidf_w2v_vectors_project_title_test = []; # the avg-w2v for each sentence/review is s
for sentence in tqdm(x_test['preprocessed_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((se
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # g
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_project_title_test.append(vector)

print(len(tfidf_w2v_vectors_project_title_test))
print(len(tfidf_w2v_vectors_project_title_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█| 15000/15000 [00:00<00:00, 41544.08it/s]

15000
300
```

merge all aparse data, SET 4

In [151]:

```
# 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 matir
x_train_TFIDFW2V = hstack((x_train_ohe, tfidf_w2v_vectors_essays_train, tfidf_w2v_vect
x_cv_TFIDFW2V = hstack((x_cv_ohe, tfidf_w2v_vectors_essays_cv, tfidf_w2v_vectors_proje
x_test_TFIDFW2V = hstack((x_test_ohe, tfidf_w2v_vectors_essays_test, tfidf_w2v_vectors_

print(x_train_TFIDFW2V.shape)
print(x_cv_TFIDFW2V.shape)
print(x_test_TFIDFW2V.shape)
```

```
(24500, 700)
(10500, 700)
(15000, 700)
```

In [152]:

```
type(x_train_TFIDFW2V)
```

Out[152]:

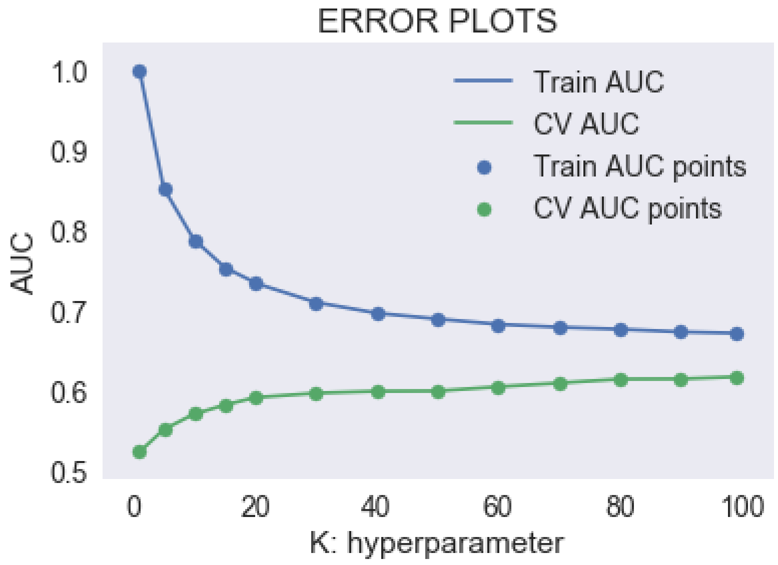
```
scipy.sparse.csr.csr_matrix
```

Hyperparameter tuning by AUC plot for cv and train dataset, SET 4

In [153]:

```

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 99]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(x_train_TFIDFW2V, y_train)


    y_train_pred = batch_predict(neigh, x_train_TFIDFW2V)
    y_cv_pred = batch_predict(neigh, x_cv_TFIDFW2V)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
    # 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%|██████████████████████████████████████████████████
█████| 13/13 [2:15:41<00:00, 629.95s/it]
```



In [154]:

```
opt_k_TFIDFW2V=80
```

Apply best hyperparameter on test dataset, SET 4

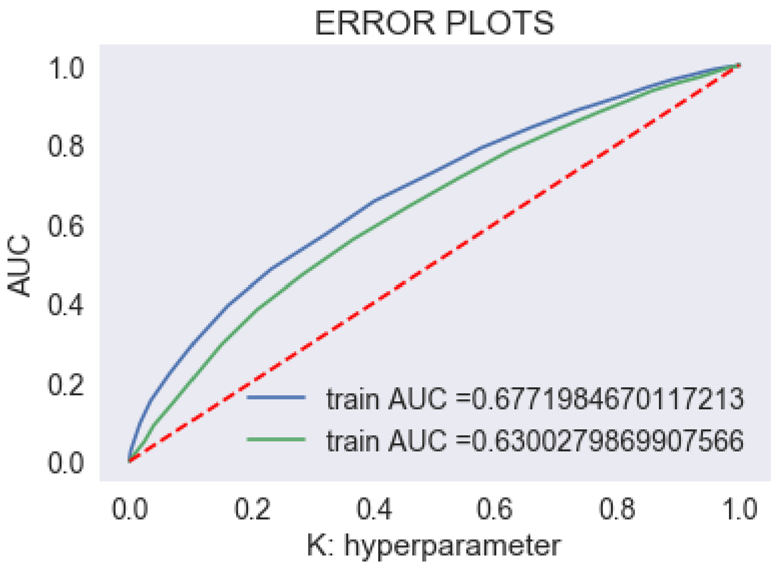
You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure

Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.

Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

In [155]:

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

neigh = KNeighborsClassifier(n_neighbors=opt_k_TFIDFW2V, n_jobs=-1)
neigh.fit(x_train_TFIDFW2V, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
# not the predicted outputs

y_train_pred = batch_predict(neigh, x_train_TFIDFW2V)
y_test_pred = batch_predict(neigh, x_test_TFIDFW2V)

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)
tain_auc_TFIDFW2V=auc(train_fpr, train_tpr)
test_auc_TFIDFW2V=auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.plot([0,1],[0,1],'r--')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



CONFUSION MATRIX, SET 4

In [156]:

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

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

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

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




```

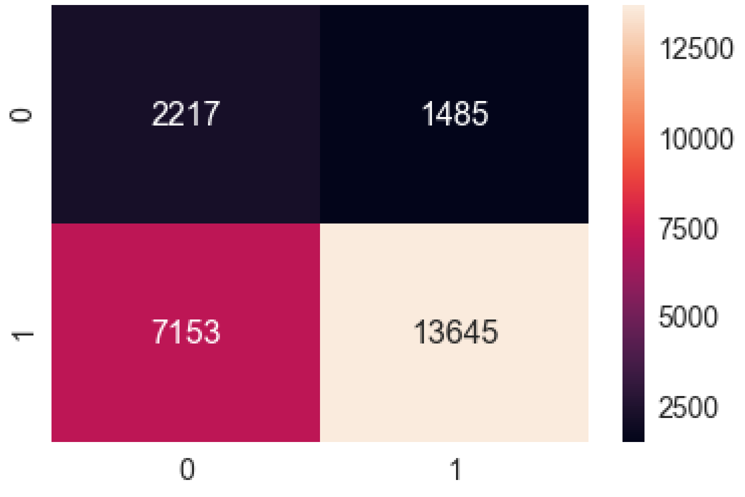
In [157]:

```
print("="*100)
print("TRAIN confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_


conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
============================================================================
========================
TRAIN confusion matrix
the maximum value of tpr*(1-fpr) 0.39289929074653884 for threshold 0.85
[[ 2217  1485]
 [ 7153 13645]]
the maximum value of tpr*(1-fpr) 0.39289929074653884 for threshold 0.85
```

Out[157]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea2ad3bd30>
```



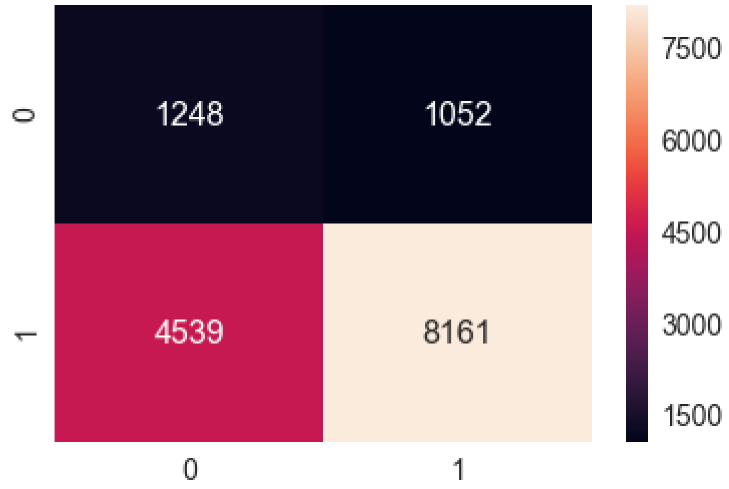
In [158]:

```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_tpr)


conf_matr_df_test_1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, te_th

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test_1, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
============================================================================
========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.35439233139335846 for threshold 0.85
[[1248 1052]
 [4539 8161]]
the maximum value of tpr*(1-fpr) 0.35439233139335846 for threshold 0.85
```

Out[158]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea2042c0b8>
```



Conclusion

In [ ]:

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

In [161]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper parameter k","AUC_train", "AUC_test"]
x.add_row(["BOW", "Brute", opt_k_bow, tain_auc_bow,test_auc_bow])
x.add_row(["BOW_top2000", "Brute", opt_k_bow_2000, tain_auc_bow_2000,test_auc_bow_2000
x.add_row(["TFIDF", "Brute", opt_k_tfidf, tain_auc_tfidf,tain_auc_tfidf])
x.add_row(["AVG W2V", "Brute", opt_k_AVGW2V, tain_auc_AVGW2V,tain_auc_AVGW2V])
x.add_row(["TFIDF W2V", "Brute", opt_k_TFIDFW2V, tain_auc_TFIDFW2V,test_auc_TFIDFW2V])
print(x)
with open('KNN_Result.txt','w') as file:
    file.write(str(x))

```

```
+-------------+-------+-------------------+--------------------+------------
--------+
|  Vectorizer | Model | Hyper parameter k |     AUC_train      |       AUC_te
st      |
+-------------+-------+-------------------+--------------------+------------
--------+
|     BOW     | Brute |         90        | 0.6675832032845697 | 0.614286340
2944196 |
| BOW_top2000 | Brute |         40        | 0.6966539075231073 | 0.514796953
0982541 |
|    TFIDF    | Brute |         90        | 0.6662211395778457 | 0.666221139
5778457 |
|   AVG W2V   | Brute |         90        | 0.6677880304120587 | 0.667788030
4120587 |
|  TFIDF W2V  | Brute |         80        | 0.6771984670117213 | 0.630027986
9907566 |
+-------------+-------+-------------------+--------------------+------------
--------+
```

Observation: (Above analysis done on 50000 dataset dut to memory issue)

1)Hyper parameter value k is same i.e 3 for all cases.

2)there is not much difference in AUC value of train and test value and AUC value not much low that shows that model is neither underfit nor overfit.

3) most of error occuring in FN box of confusion where actual value is 1 but predicted 0.

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```