Assignment 3: Apply KNN on Donors Choose dataset

This exercise is to apply KNN on Donors Choose dataset and predict approval of a new project proposal.

Relevant Information : The dataset is divided into two files -

1. train.csv file which contains information regarding projects, schools and teachers who submitted the projects.
2. resources.csv which provides information about the resources required for each project.
3. test.csv file which contains information regarding projects, schools and teachers who submitted the projects. We will be testing our model on this data.

OBJECTIVE : The goal 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 using KNN.

In [2]:
```
# Importing the required libraries
# Warning reference : https://stackoverflow.com/questions/41658568/chunkize-warning-while-installing-gensim

%matplotlib inline
import warnings
warnings.filterwarnings(action='ignore', category = UserWarning , module = 'gensim')

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
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. Reading the data

In [3]:
```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')

print("\nNumber of data points in train data", project_data.shape)
print('-'*120)
print("The attributes of data :", project_data.columns.values)
```

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

In [4]:
```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109248 entries, 0 to 109247
Data columns (total 17 columns):
Unnamed: 0                                      109248 non-null int64
id                                              109248 non-null object
teacher_id                                      109248 non-null object
teacher_prefix                                  109245 non-null object
school_state                                    109248 non-null object
project_submitted_datetime                      109248 non-null object
project_grade_category                          109248 non-null object
project_subject_categories                      109248 non-null object
project_subject_subcategories                   109248 non-null object
project_title                                   109248 non-null object
project_essay_1                                 109248 non-null object
project_essay_2                                 109248 non-null object
project_essay_3                                 3758 non-null object
project_essay_4                                 3758 non-null object
project_resource_summary                        109248 non-null object
teacher_number_of_previously_posted_projects    109248 non-null int64
project_is_approved                             109248 non-null int64
dtypes: int64(3), object(14)
memory usage: 14.2+ MB
```

NOTE:

1. We have a total of 109248 datapoints and 17 columns.
2. Now we have to sort the data according to date and time so as to have a better prediction on the future data (Test data).
3. As we can see there are null points for project_essay_3 and project_essay_4. Only 3758 points are not null.
4. In teacher_prefix there are 109245 points which means 3 points are null.

Sorting according to date

In [5]:
```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]


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


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

In [6]:
```
project_data.drop('Unnamed: 0', axis=1, inplace=True)
```

Adding the price and quantity column from resource_data to the project_data

In [7]:
```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in-one-step
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()

# join two dataframes in python:
project_data = pd.merge(project_data, price_data, on='id', how='left')

del resource_data
```

Now we have to separate the approved and the not approved data points

In [8]:
```
# Creating the approved and rejected data frames

project_approved = project_data[project_data['project_is_approved'] == 1]
project_reject = project_data[project_data['project_is_approved'] == 0]

print('Number of projects that got approval : ', project_approved.shape)
print("Numer of projects that didn't get approval : ", project_reject.shape)
```

```
Number of projects that got approval :  (92706, 18)
Numer of projects that didn't get approval :  (16542, 18)
```

NOTE:

1. We have a imbalanced approved (+ve) and not approved (-ve) points.
2. We have to sample our data as number of approved (+ve) points are more than not approved (-ve) points.

In [9]:
```
# Sampling 16500 points for not approved (-ve) and 25500 points for approved (+ve) points

# Reference: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sample.html

project_approved = project_approved[project_approved['project_is_approved'] == 1].sample(n = 25500, random_state=1)
project_reject = project_reject[project_reject['project_is_approved'] == 0].sample(n = 16500, random_state=1)

# Reference: https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html
# Concatinating the two data frames

# Reference: https://stackoverflow.com/questions/51835369/combine-two-dataframes-one-row-from-each-one-at-a-time-python-
project_data = pd.concat([project_approved, project_reject]).sort_index(kind='merge')
```

In [10]:
```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 42000 entries, 1 to 109245
Data columns (total 18 columns):
id                                              42000 non-null object
teacher_id                                      42000 non-null object
teacher_prefix                                  42000 non-null object
school_state                                    42000 non-null object
Date                                            42000 non-null datetime64[ns]
project_grade_category                          42000 non-null object
project_subject_categories                      42000 non-null object
project_subject_subcategories                   42000 non-null object
project_title                                   42000 non-null object
project_essay_1                                 42000 non-null object
project_essay_2                                 42000 non-null object
project_essay_3                                 1376 non-null object
project_essay_4                                 1376 non-null object
project_resource_summary                        42000 non-null object
teacher_number_of_previously_posted_projects    42000 non-null int64
project_is_approved                             42000 non-null int64
price                                           42000 non-null float64
quantity                                        42000 non-null int64
dtypes: datetime64[ns](1), float64(1), int64(3), object(13)
memory usage: 6.1+ MB
```

NOTE:

1. As we can see that the price and the quantity column has been added to the project_data
2. This is where the preprocessing will start.
3. we are going to consider

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

- project_title : text data
- essay : text data

- quantity : numerical
- teacher_number_of_previously_posted_projects : numerical
- price : numerical
```

2. Preprocessing Data

project_subject_categories

In [11]:
```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Sci
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())


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

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

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

project_subject_subcategories

In [12]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Sci
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

Text Preprocessing

Essay

In [13]:
```
# Combining all the essay
# 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 [14]:
```
project_data['essay'].describe()
```

Out[14]:
```
count                                                42000
unique                                               41966
top       Throughout the day, our playground is filled w...
freq                                                     3
Name: essay, dtype: object
```

In [15]:
```
# Making the decontracted function

# 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


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


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

```
100%|████████████████████████████████████████████████████████████████████████████| 42000/42000 [00:29<00:00, 1412.90it/s]
```

`project_title`

In [16]:
```
project_data['project_title'].describe()
```

Out[16]:
```
count                42000
unique               39982
top       Flexible Seating
freq                   101
Name: project_title, dtype: object
```

In [17]:
```
# Project_title

# Combining all the above statements
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['project_title'].values):
    sent = decontracted(sentence)
    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_titles.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████████| 42000/42000 [00:01<00:00, 29155.33it/s]
```

Replacing the columns with new cleaned columns - Project_title and Essay

In [18]:
```
# Adding processed essay columns in place of previous essays columns and dropping the previous columns

## ESSAY

project_data['clean_essays'] = preprocessed_essays
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
project_data.drop(['essay'], axis=1, inplace=True)
```

In [19]:
```
## Project_title

# Adding processed project_title columns in place of previous project_title column and dropping the previous column

project_data['clean_titles'] = preprocessed_titles

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

In [20]:
```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 42000 entries, 1 to 109245
Data columns (total 15 columns):
id                                              42000 non-null object
teacher_id                                      42000 non-null object
teacher_prefix                                  42000 non-null object
school_state                                    42000 non-null object
Date                                            42000 non-null datetime64[ns]
project_grade_category                          42000 non-null object
project_resource_summary                        42000 non-null object
teacher_number_of_previously_posted_projects    42000 non-null int64
project_is_approved                             42000 non-null int64
price                                           42000 non-null float64
quantity                                        42000 non-null int64
clean_categories                                42000 non-null object
clean_subcategories                             42000 non-null object
clean_essays                                    42000 non-null object
clean_titles                                    42000 non-null object
dtypes: datetime64[ns](1), float64(1), int64(3), object(10)
memory usage: 5.1+ MB
```

NOTE:

- Till now we have preprocessed the data.
- Now we have to split the data and vectorize the data for BOW, TF-IDF, Avg W2V and TFIDF weighted W2Vec

**********ASSIGNMENT**********

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

In [21]:
```
# Creating label and feature data frame : Label- y, Features- X

y = project_data['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
X = project_data

print(y.shape)
print(X.shape)
```

```
(42000,)
(42000, 14)
```

In [22]:
```
## train test cross-validation split
# Referance : https://stackoverflow.com/questions/34842405/parameter-stratify-from-method-train-test-split-scikit-learn

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.30, stratify=y_train)
```

NOTE:

- This `stratify` parameter makes a split so that the proportion of values in the sample produced will be the same as the proportion of values provided to parameter stratify.

- For example, if variable y is a binary categorical variable with values 0 and 1 and there are 25% of zeros and 75% of ones, stratify=y will make sure that your random split has 25% of 0's and 75% of 1's.

In [23]:
```
## Shape of the matrices

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

```
(20580, 14) (20580,)
(8820, 14) (8820,)
(12600, 14) (12600,)
```

NOTE:

1. We will now use the train data for training our model, cv data to validate the model and perform testing on the test data

4. Make Data Model Ready: encoding numerical, categorical features

Vectorizing Categorical Features

1. clean_categories

In [24]:
```
# We use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()

# We will fit the train data only
vectorizer.fit(X_train['clean_categories'].values)

# We use the fitted CountVectorizer to convert the text to vector
X_train_clean_category = vectorizer.transform(X_train['clean_categories'].values)
X_cv_clean_category = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_category = vectorizer.transform(X_test['clean_categories'].values)

print("Clean categories are vectorized")
print(X_train_clean_category.shape, y_train.shape)
print(X_cv_clean_category.shape, y_cv.shape)
print(X_test_clean_category.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
Clean categories are vectorized
(20580, 9) (20580,)
(8820, 9) (8820,)
(12600, 9) (12600,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
```

2. clean_subcategories

In [25]:
```
vectorizer = CountVectorizer()

# We will fit the train data only
vectorizer.fit(X_train['clean_subcategories'].values)


# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategories = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_clean_subcategories = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_clean_subcategories = vectorizer.transform(X_test['clean_subcategories'].values)

print("clean_subcategories are vectorized")
print(X_train_clean_subcategories.shape, y_train.shape)
print(X_cv_clean_subcategories.shape, y_cv.shape)
print(X_test_clean_subcategories.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
clean_subcategories are vectorized
(20580, 30) (20580,)
(8820, 30) (8820,)
(12600, 30) (12600,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice',
'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlanguag
es', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'ma
thematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialsciences', 'specialne
eds', 'teamsports', 'visualarts', 'warmth']
```

3. teacher_prefix

In [26]:
```
vectorizer = CountVectorizer()

# We will fit the train data only
vectorizer.fit(X_train['teacher_prefix'].values)

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

print("teacher_prefix are vectorized")
print(X_train_teacher_prefix.shape, y_train.shape)
print(X_cv_teacher_prefix.shape, y_cv.shape)
print(X_test_teacher_prefix.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
teacher_prefix are vectorized
(20580, 5) (20580,)
(8820, 5) (8820,)
(12600, 5) (12600,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

4. school_state

In [27]:
```
vectorizer = CountVectorizer()

# We will fit the train data only
vectorizer.fit(X_train['school_state'].values)

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

print("school_state are vectorized")
print(X_train_school_state.shape, y_train.shape)
print(X_cv_school_state.shape, y_cv.shape)
print(X_test_school_state.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
school_state are vectorized
(20580, 51) (20580,)
(8820, 51) (8820,)
(12600, 51) (12600,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'm
a', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa',
'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
```

5. project_grade_category

In [28]:
```
#This step is to intialize a vectorizer with vocab from train data

# Creating the list of grades
grades = list(set(project_data['project_grade_category'].values))

#*************************************************************************************************************

# we use count vectorizer to convert the values into one hot encoded features
# We will fit the train data only
vectorizer = CountVectorizer(vocabulary = grades, lowercase=False, binary=True)
vectorizer.fit(X_train['project_grade_category'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_project_grade = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_project_grade = vectorizer.transform(X_test['project_grade_category'].values)


print("project_grade_category are vectorized")
print(X_train_project_grade.shape, y_train.shape)
print(X_cv_project_grade.shape, y_cv.shape)
print(X_test_project_grade.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
project_grade_category are vectorized
(20580, 4) (20580,)
(8820, 4) (8820,)
(12600, 4) (12600,)
['Grades PreK-2', 'Grades 6-8', 'Grades 3-5', 'Grades 9-12']
```

Standardizing Numerical features

1. price

In [29]:
```
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

price_scalar = StandardScaler()

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

# Now standardize the data with above maen and variance.
X_train_price = price_scalar.transform(X_train['price'].values.reshape(-1,1))
X_cv_price = price_scalar.transform(X_cv['price'].values.reshape(-1,1))
X_test_price = price_scalar.transform(X_test['price'].values.reshape(-1,1))

print("price is standardized")
print(X_train_price.shape, y_train.shape)
print(X_cv_price.shape, y_cv.shape)
print(X_test_price.shape, y_test.shape)
```

```
Mean : 318.66180077745383, Standard deviation : 393.2699336857306
price is standardized
(20580, 1) (20580,)
(8820, 1) (8820,)
(12600, 1) (12600,)
```

2. teacher_number_of_previously _posted_projects

In [30]:
```
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

# https://stackoverflow.com/questions/29086398/sklearn-turning-off-warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

previous_post_scalar = StandardScaler()

# We will fit the train data only
# finding the mean and standard deviation of this data
previous_post_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
print(f"Mean : {previous_post_scalar.mean_[0]}, Standard deviation : {np.sqrt(previous_post_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
X_train_previous_projects = previous_post_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].value
X_cv_previous_projects = previous_post_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.resh
X_test_previous_projects = previous_post_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.

print("teacher_number_of_previously_posted_projects is standardized")
print(X_train_previous_projects.shape, y_train.shape)
print(X_cv_previous_projects.shape, y_cv.shape)
print(X_test_previous_projects.shape, y_test.shape)
```

```
Mean : 9.863702623906706, Standard deviation : 24.63539834459144
teacher_number_of_previously_posted_projects is standardized
(20580, 1) (20580,)
(8820, 1) (8820,)
(12600, 1) (12600,)
```

3. quantity

In [31]:
```
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

# https://stackoverflow.com/questions/29086398/sklearn-turning-off-warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

quantity_scalar = StandardScaler()

# We will fit the train data only
# finding the mean and standard deviation of this data
quantity_scalar.fit(X_train['quantity'].values.reshape(-1,1))
print(f"Mean : {quantity_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
X_train_quantity = quantity_scalar.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_quantity = quantity_scalar.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity = quantity_scalar.transform(X_test['quantity'].values.reshape(-1,1))

print("quantity is standardized")
print(X_train_quantity.shape, y_train.shape)
print(X_cv_quantity.shape, y_cv.shape)
print(X_test_quantity.shape, y_test.shape)
```

```
Mean : 18.044752186588923, Standard deviation : 28.895300245031923
quantity is standardized
(20580, 1) (20580,)
(8820, 1) (8820,)
(12600, 1) (12600,)
```

5. Make Data Model Ready: encoding eassay, and project_title

BOW

1. clean_essay

In [31]:
```
%%time
# Vectorizing the essay column

from sklearn.feature_extraction.text import CountVectorizer

# We are considering only the words which appeared in at least 10 documents(rows or projects).
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
# Taking top 5000 features

vectorizer = CountVectorizer(min_df=10, max_features=5000)

# We will fit the train data only
vectorizer.fit(X_train['clean_essays'].values)

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

print("Essay vectorized")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
Essay vectorized
(20580, 5000) (20580,)
(8820, 5000) (8820,)
(12600, 5000) (12600,)
Wall time: 7.97 s
```

2. clean_titles

In [32]:
```
# Vectorizing the project_title column

# We are considering only the words which appeared in at least 10 documents(rows or projects).
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
# Taking top 5000 features

vectorizer = CountVectorizer(min_df=10, max_features=5000)

# We will fit the train data only
vectorizer.fit(X_train['clean_titles'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_bow = vectorizer.transform(X_train['clean_titles'].values)
X_cv_titles_bow = vectorizer.transform(X_cv['clean_titles'].values)
X_test_titles_bow = vectorizer.transform(X_test['clean_titles'].values)

print("Project Titles vectorized")
print(X_train_titles_bow.shape, y_train.shape)
print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)
```

```
Project Titles vectorized
(20580, 1159) (20580,)
(8820, 1159) (8820,)
(12600, 1159) (12600,)
```

TF-IDF

1. clean_essay

In [31]:
```
%%time
# Vectorizing the essay column

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2

vectorizer = TfidfVectorizer(min_df=10, max_features=5000)

# We will fit the train data only
vectorizer.fit(X_train['clean_essays'].values)

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


print("Essay vectorized")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
Essay vectorized
(20580, 5000) (20580,)
(8820, 5000) (8820,)
(12600, 5000) (12600,)
Wall time: 8.27 s
```

2. clean_titles

In [32]:
```
%%time
# Vectorizing the project_title column

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2

vectorizer = TfidfVectorizer(min_df=5, max_features=5000)

# We will fit the train data only
vectorizer.fit(X_train['clean_titles'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_tfidf = vectorizer.transform(X_train['clean_titles'].values)
X_cv_titles_tfidf = vectorizer.transform(X_cv['clean_titles'].values)
X_test_titles_tfidf = vectorizer.transform(X_test['clean_titles'].values)


print("Titles vectorized")
print(X_train_titles_tfidf.shape, y_train.shape)
print(X_cv_titles_tfidf.shape, y_cv.shape)
print(X_test_titles_tfidf.shape, y_test.shape)
```

```
Titles vectorized
(20580, 1975) (20580,)
(8820, 1975) (8820,)
(12600, 1975) (12600,)
Wall time: 500 ms
```

Average W2V

1. clean_essay

In [31]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-
# 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 [32]:
```
%%time
# average Word2Vec
# compute average word2vec
train_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train['clean_essays'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_w2v_vectors_essays.append(vector)
print("Train  vector")
print(len(train_w2v_vectors_essays))
print(len(train_w2v_vectors_essays[0]))
print('='*120)


# average Word2Vec
# compute average word2vec
test_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_test['clean_essays'].values): # for each essay in test data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_w2v_vectors_essays.append(vector)

print("Test vec")
print(len(test_w2v_vectors_essays))
print(len(test_w2v_vectors_essays[0]))
print('='*120)


# average Word2Vec
# compute average word2vec
cv_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_cv['clean_essays'].values): # for each essay in cv data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_w2v_vectors_essays.append(vector)

print("CV vec")
print(len(cv_w2v_vectors_essays))
print(len(cv_w2v_vectors_essays[0]))
print('='*120)
```

```
100%|████████████████████████████████████████████████████████████████| 20580/20580 [00:07<00:00, 2632.87it/s]

Train  vector
20580
50
============================================================================================================
=

100%|████████████████████████████████████████████████████████████████| 12600/12600 [00:05<00:00, 2391.55it/s]

Test vec
12600
50
============================================================================================================
=

100%|████████████████████████████████████████████████████████████████| 8820/8820 [00:03<00:00, 2537.06it/s]

CV vec
8820
50
============================================================================================================
=
Wall time: 16.6 s
```

In [33]:
```
# Changing list to numpy arrays
train_w2v_vectors_essays = np.array(train_w2v_vectors_essays)
test_w2v_vectors_essays = np.array(test_w2v_vectors_essays)
cv_w2v_vectors_essays = np.array(cv_w2v_vectors_essays)

print("Essay vectorized")
print(train_w2v_vectors_essays.shape, y_train.shape)
print(cv_w2v_vectors_essays.shape, y_cv.shape)
print(test_w2v_vectors_essays.shape, y_test.shape)
```

```
Essay vectorized
(20580, 50) (20580,)
(8820, 50) (8820,)
(12600, 50) (12600,)
```

2. clean_titles

In [34]:
```
%%time
# average Word2Vec
# compute average word2vec
train_w2v_vectors_titles = []; # the avg-w2v for each title is stored in this list
for sentence in tqdm(X_train['clean_titles'].values): # for each title in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the title
    for word in sentence.split(): # for each word in a title
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_w2v_vectors_titles.append(vector)
print("Train  vector")
print(len(train_w2v_vectors_titles))
print(len(train_w2v_vectors_titles[0]))
print('='*120)

# average Word2Vec
# compute average word2vec
test_w2v_vectors_titles = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_test['clean_titles'].values): # for each essay in test data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the title
    for word in sentence.split(): # for each word in a title
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_w2v_vectors_titles.append(vector)

print("Test vec")
print(len(test_w2v_vectors_titles))
print(len(test_w2v_vectors_titles[0]))
print('='*120)

# average Word2Vec
# compute average word2vec
cv_w2v_vectors_titles = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_cv['clean_titles'].values): # for each essay in cv data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the title
    for word in sentence.split(): # for each word in a title
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_w2v_vectors_titles.append(vector)

print("CV vec")
print(len(cv_w2v_vectors_titles))
print(len(cv_w2v_vectors_titles[0]))
print('='*120)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 20580/20580 [00:00<00:00, 52661.50it/s]

Train  vector
20580
50
============================================================================================================================
=

100%|████████████████████████████████████████████████████████████████████████████| 12600/12600 [00:00<00:00, 50836.23it/s]

Test vec
12600
50
============================================================================================================================
=

100%|████████████████████████████████████████████████████████████████████████████| 8820/8820 [00:00<00:00, 51080.34it/s]

CV vec
8820
50
============================================================================================================================
=
Wall time: 861 ms
```

In [35]:
```
# Changing list to numpy arrays
train_w2v_vectors_titles = np.array(train_w2v_vectors_titles)
test_w2v_vectors_titles = np.array(test_w2v_vectors_titles)
cv_w2v_vectors_titles = np.array(cv_w2v_vectors_titles)

print("Title vectorized")
print(train_w2v_vectors_titles.shape, y_train.shape)
print(cv_w2v_vectors_titles.shape, y_cv.shape)
print(test_w2v_vectors_titles.shape, y_test.shape)
```

```
Title vectorized
(20580, 50) (20580,)
(8820, 50) (8820,)
(12600, 50) (12600,)
```

TF-IDF weighted W2V

1. clean_essay

In [31]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-
# 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 [32]:
```
%%time
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['clean_essays'].values)
# 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())


# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_essays'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_tfidf_w2v_essays.append(vector)

print("Train matrix:")
print(len(train_tfidf_w2v_essays))
print(len(train_tfidf_w2v_essays[0]))
print('='*120)


cv_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_essays'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_tfidf_w2v_essays.append(vector)

print("CV matrix:")
print(len(cv_tfidf_w2v_essays))
print(len(cv_tfidf_w2v_essays[0]))
print('='*120)


test_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_essays'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_essays.append(vector)

print("Test matrix:")
print(len(test_tfidf_w2v_essays))
print(len(test_tfidf_w2v_essays[0]))
print('='*120)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 20580/20580 [00:50<00:00, 406.89it/s]

Train matrix:
20580
50
========================================================================================================================
=

100%|████████████████████████████████████████████████████████████████████████████| 8820/8820 [00:22<00:00, 397.37it/s]

CV matrix:
8820
50
========================================================================================================================
=
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 12600/12600 [00:30<00:00, 413.88it/s]

Test matrix:
12600
50
==========================================================================================================================
=
Wall time: 1min 46s
```

In [33]:
```
# Changing list to numpy arrays
train_tfidf_w2v_essays = np.array(train_tfidf_w2v_essays)
test_tfidf_w2v_essays = np.array(test_tfidf_w2v_essays)
cv_tfidf_w2v_essays = np.array(cv_tfidf_w2v_essays)

print("Essay vectorized")
print(train_tfidf_w2v_essays.shape, y_train.shape)
print(cv_tfidf_w2v_essays.shape, y_cv.shape)
print(test_tfidf_w2v_essays.shape, y_test.shape)
```

```
Essay vectorized
(20580, 50) (20580,)
(8820, 50) (8820,)
(12600, 50) (12600,)
```

2. clean_titles

In [34]:
```
%%time
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['clean_titles'].values)
# 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())


# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_tfidf_w2v_titles.append(vector)

print("Train matrix:")
print(len(train_tfidf_w2v_titles))
print(len(train_tfidf_w2v_titles[0]))
print('='*120)


cv_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_tfidf_w2v_titles.append(vector)

print("CV matrix:")
print(len(cv_tfidf_w2v_titles))
print(len(cv_tfidf_w2v_titles[0]))
print('='*120)


test_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles'].values): # for each review/sentence
    vector = np.zeros(50) # 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][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_titles.append(vector)

print("Test matrix:")
print(len(test_tfidf_w2v_titles))
print(len(test_tfidf_w2v_titles[0]))
print('='*120)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 20580/20580 [00:00<00:00, 25582.13it/s]

Train matrix:
20580
50
============================================================================================================================
=

100%|████████████████████████████████████████████████████████████████████████████| 8820/8820 [00:00<00:00, 26305.40it/s]

CV matrix:
8820
50
============================================================================================================================
=
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 12600/12600 [00:00<00:00, 26827.52it/s]

Test matrix:
12600
50
==========================================================================================================================
=
Wall time: 1.87 s
```

In [35]:
```
# Changing list to numpy arrays
train_tfidf_w2v_titles = np.array(train_tfidf_w2v_titles)
test_tfidf_w2v_titles = np.array(test_tfidf_w2v_titles)
cv_tfidf_w2v_titles = np.array(cv_tfidf_w2v_titles)

print("Title vectorized")
print(train_tfidf_w2v_titles.shape, y_train.shape)
print(cv_tfidf_w2v_titles.shape, y_cv.shape)
print(test_tfidf_w2v_titles.shape, y_test.shape)
```

```
Title vectorized
(20580, 50) (20580,)
(8820, 50) (8820,)
(12600, 50) (12600,)
```

1. Applying KNN brute force on BOW, SET 1

Merging the categorical, numerical and text features

In [45]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# https://stackoverflow.com/questions/54226138/constructing-sparse-csr-matrix-directly-vs-using-coo-tocsr-scipy
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)

from scipy.sparse import hstack

# Training data
X_tr = hstack((X_train_essay_bow, X_train_titles_bow, X_train_clean_category, X_train_clean_subcategories,
               X_train_project_grade, X_train_school_state, X_train_teacher_prefix,
               X_train_previous_projects, X_train_price, X_train_quantity)).tocsr()
# CV data
X_cr = hstack((X_cv_essay_bow, X_cv_titles_bow, X_cv_clean_category, X_cv_clean_subcategories,
               X_cv_project_grade, X_cv_school_state, X_cv_teacher_prefix,
               X_cv_previous_projects, X_cv_price, X_cv_quantity)).tocsr()
# Test data
X_te = hstack((X_test_essay_bow, X_test_titles_bow, X_test_clean_category, X_test_clean_subcategories,
               X_test_project_grade, X_test_school_state, X_test_teacher_prefix,
               X_test_previous_projects, X_test_price, X_test_quantity)).tocsr()
```

In [46]:
```
## Print the final data matrix

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

```
Final Data matrix
(20580, 6261) (20580,)
(8820, 6261) (8820,)
(12600, 6261) (12600,)
```

Function to predict the probability scores in batches

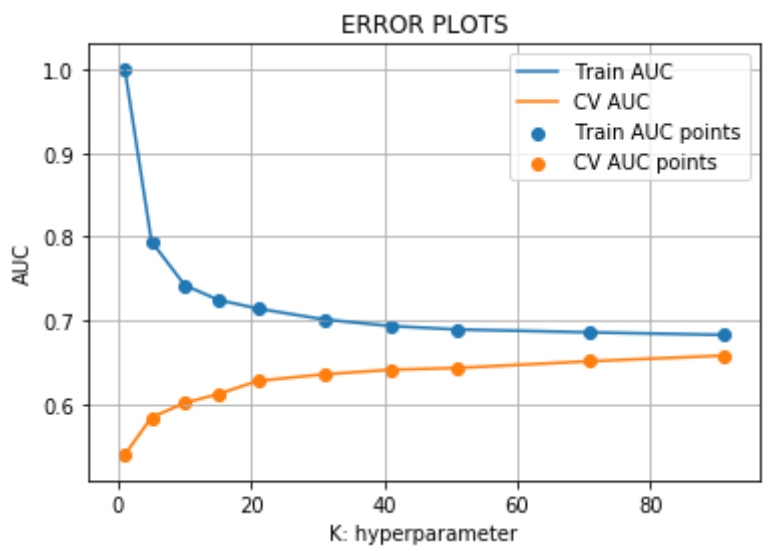
In [47]:
```
# Reference : Assignment_SAMPLE_SOLUTION

def batch_predict(clf, data):

    """
    This function returns the predicted probability scores
    """

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000

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

    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

Hyper paramter tuning to find best K (Using Simple CV)

Applying KNN for multiple K values

In [48]:
```
%%time
# Reference : https://stackoverflow.com/questions/32565829/simple-way-to-measure-cell-execution-time-in-ipython-notebook

# Trying various K values

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

# To store the roc_auc_scores of train and test data we need the two lists
train_auc = []
cv_auc = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 71, 91]

for i in tqdm(K):

    # Creating the classifier
    classifier = KNeighborsClassifier(n_neighbors = i, n_jobs = -1)
    # Fitting the train data
    classifier.fit(X_tr, y_train)

    # Getting the probability scores
    y_train_pred = batch_predict(classifier, X_tr)
    y_cv_pred = batch_predict(classifier, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # 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))


# Plotting the Error
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%|██████████| 10/10 [10:21<00:00, 61.99s/it]
```

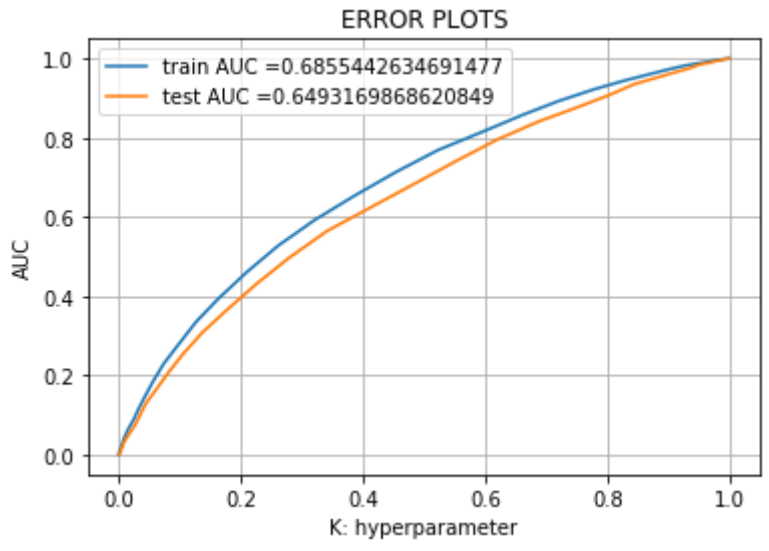



```
Wall time: 10min 22s
```

Now creating the model with best K

In [49]:
```
# From the error plot we choose K such that, we will have maximum AUC on cv data and gap between the train and cv is les

# Here we are choosing the best_k based on GridSearchCV results
best_k = 71
```

In [50]:
```
%%time

# Using the best_K

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

# Classifier
neigh = KNeighborsClassifier(n_neighbors = best_k)
# Fitting train data
neigh.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)


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


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




```
Wall time: 1min 15s
```

NOTE:

- As we can see from the graph. We are not doing good with the test set. The AUC curve is lower for the test set than
the train set.
- The AUC scores for the Train and Test data are : 68% and 65% respectively
- We need more data to train the model so as to make it more accurate.

In [51]:
```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

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

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

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

In [52]:
```
from sklearn.metrics import confusion_matrix

print("="*120)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("="*120)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
========================================================================================================================
=
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24947924590649212 for threshold 0.521
[[3858 4227]
 [2886 9609]]
========================================================================================================================
=
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24975169880624426 for threshold 0.549
[[2906 2044]
 [2872 4778]]
```

NOTE:

1. As we can see from the confusion matrix that:

   ON THE TRAIN SET:
   - The number of correct predictions : 13467
   - The number of incorrect predictions : 7113

   ON THE TEST SET:
   - The number of correct predictions : 7684
   - The number of incorrect predictions : 4916

In [53]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

from sklearn.metrics import classification_report

y_pred_new = neigh.predict(X_te)
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred_new, target_names = target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.55      0.38      0.45      4950
     class 1       0.66      0.80      0.72      7650

   micro avg       0.63      0.63      0.63     12600
   macro avg       0.61      0.59      0.59     12600
weighted avg       0.62      0.63      0.62     12600
```

In [54]:
```
%%time
# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = neigh.predict(X_te)
print("Accuracy on test set: {}".format(accuracy_score(y_test, y_pred_new)))
print("Precision on test set: {}".format(precision_score(y_test, y_pred_new)))
print("Recall on test set: {}".format(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: {}".format(f1_score(y_test, y_pred_new)))
```

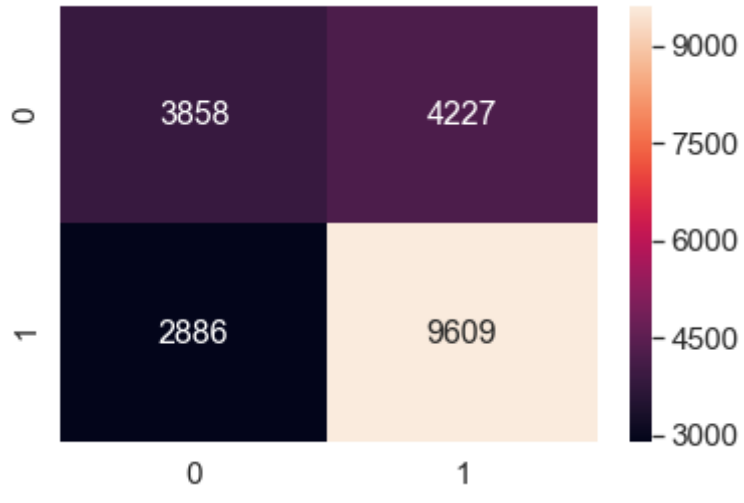
```
Accuracy on test set: 0.6323015873015873
Precision on test set: 0.6648093521249864
Recall on test set: 0.7954248366013071
F1-Score on test set: 0.724275427007082
Wall time: 31.4 s
```

Confusion Matrix on train data

In [55]:
```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

df_cm = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)), range(2),ran
sns.set(font_scale=1.4) #for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24947924590649212 for threshold 0.521
```

Out[55]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ffee79b860>
```

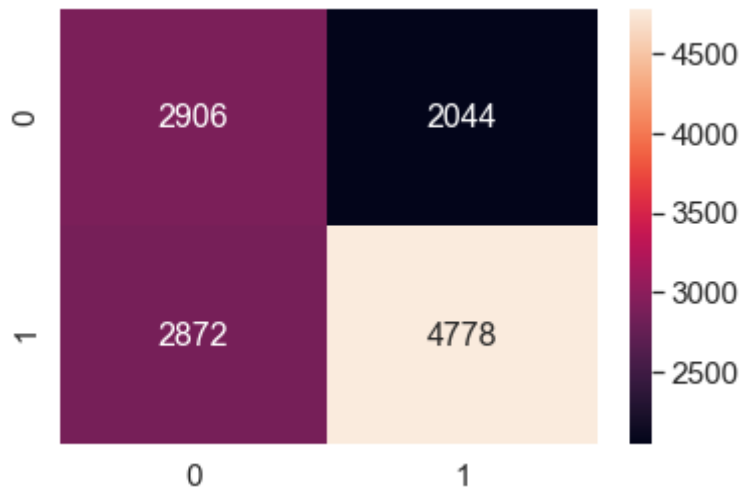


Confusion Matrix on test data

In [56]:
```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

df_cm = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)), range(2),range(2
sns.set(font_scale=1.4) #for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24975169880624426 for threshold 0.549
```

Out[56]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ffee7ee4a8>
```



NOTE:

```
- STEPS INVOLVED:
1. To apply brute force we 1st created a function called batch predict which will take a bacth of 1000 data to p
redict
the values
2. Then we applied KNN algorithm using brute force algorithm
3. Using the GridSearchCV we found the optimal K value and using that K value we again plotted the error plot.
4. After finding the best K we used it to train our KNN model and got the probability predictions.

- Evaluation:
1. Model managed to predict the test set results with an AUC score of just 65%
2. Model is not robust as we need more data to train our model and make it more accurate
```

2. Applying KNN brute force on TFIDF, SET 2

Merging the categorical, numerical and text features

In [33]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# https://stackoverflow.com/questions/54226138/constructing-sparse-csr-matrix-directly-vs-using-coo-tocsr-scipy
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)

from scipy.sparse import hstack

# Training data
X_tr = hstack((X_train_essay_tfidf, X_train_titles_tfidf, X_train_clean_category, X_train_clean_subcategories,
               X_train_project_grade, X_train_school_state, X_train_teacher_prefix,
               X_train_previous_projects, X_train_price, X_train_quantity)).tocsr()
# CV data
X_cr = hstack((X_cv_essay_tfidf, X_cv_titles_tfidf, X_cv_clean_category, X_cv_clean_subcategories,
               X_cv_project_grade, X_cv_school_state, X_cv_teacher_prefix,
               X_cv_previous_projects, X_cv_price, X_cv_quantity)).tocsr()
# Test data
X_te = hstack((X_test_essay_tfidf, X_test_titles_tfidf, X_test_clean_category, X_test_clean_subcategories,
               X_test_project_grade, X_test_school_state, X_test_teacher_prefix,
               X_test_previous_projects, X_test_price, X_test_quantity)).tocsr()
```

In [34]:
```
## Print the final data matrix

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

```
Final Data matrix
(20580, 7077) (20580,)
(8820, 7077) (8820,)
(12600, 7077) (12600,)
```

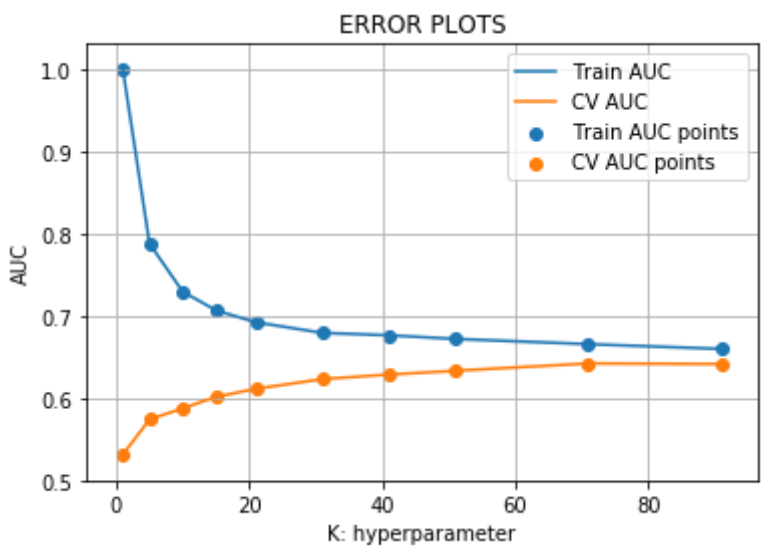
Hyper paramter tuning to find best K (Using Simple CV)

Applying KNN for multiple K values

In [35]:
```
# Reference : Assignment_SAMPLE_SOLUTION

def batch_predict(clf, data):

    """
    This function returns the predicted probability scores
    """

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000

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

    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [36]:
```
%%time
# Reference : https://stackoverflow.com/questions/32565829/simple-way-to-measure-cell-execution-time-in-ipython-notebook

# Trying various K values

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

# To store the roc_auc_scores of train and test data we need the two lists
train_auc = []
cv_auc = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 71, 91]

for i in tqdm(K):

    # Creating the classifier
    classifier = KNeighborsClassifier(n_neighbors = i, n_jobs = -1)
    # Fitting the train data
    classifier.fit(X_tr, y_train)

    # Getting the probability scores
    y_train_pred = batch_predict(classifier, X_tr)
    y_cv_pred = batch_predict(classifier, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # 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))


# Plotting the Error
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%|██████████| 10/10 [10:53<00:00, 67.00s/it]
```

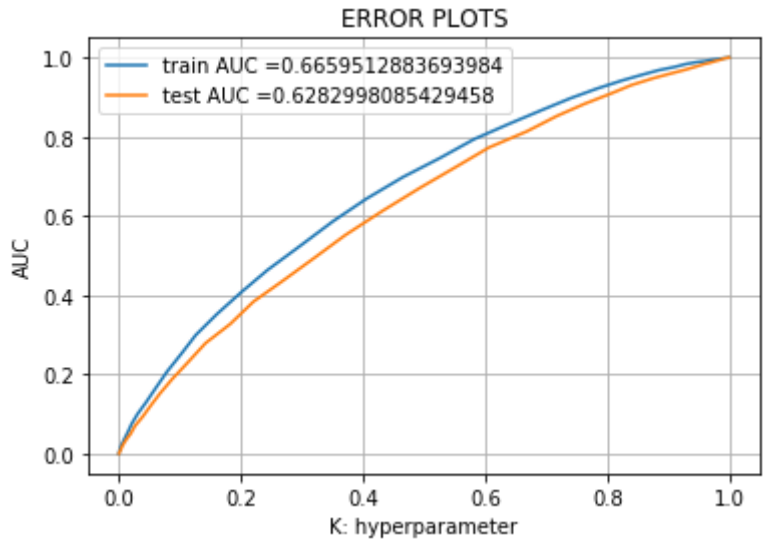



```
Wall time: 10min 54s
```

Now creating the model with best K

In [37]:
```
# From the error plot we choose K such that, we will have maximum AUC on cv data and gap between the train and cv is les

# Here we are choosing the best_k based on forloop results
best_k = 71
```

In [38]:
```
%%time

# Using the best_K

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

# Classifier
neigh = KNeighborsClassifier(n_neighbors = best_k)
# Fitting train data
neigh.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)


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


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




```
Wall time: 1min 19s
```

NOTE:

- As we can see from the graph the test AUC is very close to the Train AUC
- The AUC scores for the Train and Test data are : 66% and 62% respectively
- We need more data to train the model so as to make it more accurate.

In [39]:
```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

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

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

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

In [40]:
```
from sklearn.metrics import confusion_matrix

print("="*120)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("="*120)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
========================================================================================================================
=
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24932855876932208 for threshold 0.577
[[3833 4252]
 [3170 9325]]
========================================================================================================================
=
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24995000510152024 for threshold 0.62
[[3113 1837]
 [3427 4223]]
```

NOTE:

1. As we can see from the confusion matrix that:

   ON THE TRAIN SET:
   - The number of correct predictions : 13158
   - The number of incorrect predictions : 7422

   ON THE TEST SET:
   - The number of correct predictions : 7336
   - The number of incorrect predictions : 5264

In [41]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

from sklearn.metrics import classification_report

y_pred_new = neigh.predict(X_te)
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred_new, target_names = target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.58      0.19      0.29      4950
     class 1       0.64      0.91      0.75      7650

   micro avg       0.63      0.63      0.63     12600
   macro avg       0.61      0.55      0.52     12600
weighted avg       0.61      0.63      0.57     12600
```

In [42]:
```
%%time
# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = neigh.predict(X_te)
print("Accuracy on test set: {}".format(accuracy_score(y_test, y_pred_new)))
print("Precision on test set: {}".format(precision_score(y_test, y_pred_new)))
print("Recall on test set: {}".format(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: {}".format(f1_score(y_test, y_pred_new)))
```

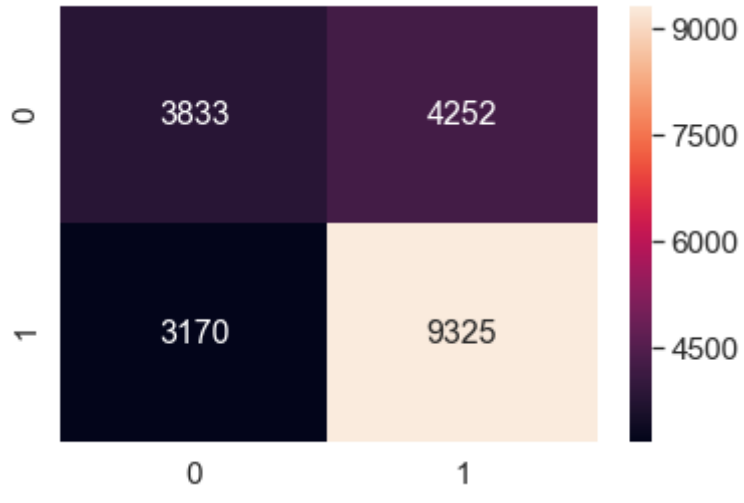
```
Accuracy on test set: 0.6281746031746032
Precision on test set: 0.6354747327058393
Recall on test set: 0.9090196078431373
F1-Score on test set: 0.7480234496853654
Wall time: 32.7 s
```

Confusion Matrix on train data

In [43]:
```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

df_cm = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)), range(2),ran
sns.set(font_scale=1.4) #for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24932855876932208 for threshold 0.577
```

Out[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1dd7b589da0>
```

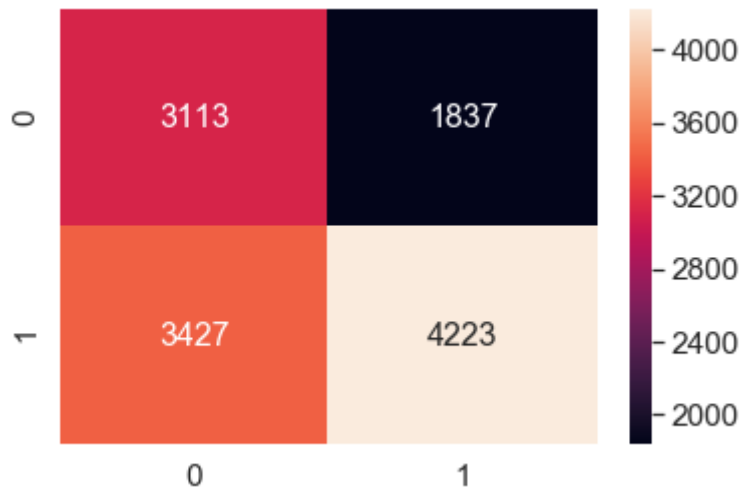


Confusion Matrix on test data

In [44]:
```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

df_cm = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)), range(2),range(2
sns.set(font_scale=1.4) #for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24995000510152024 for threshold 0.62
```

Out[44]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1dd6cd05eb8>
```



NOTE:

- Evaluation:
  1. Model managed to predict the test set results with an AUC of 62.8%
  2. Model is not robust as we need more data to train our model and make it more accurate

3. Applying KNN brute force on AVG W2V, SET 3

Merging the categorical, numerical and text features

In [36]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# https://stackoverflow.com/questions/54226138/constructing-sparse-csr-matrix-directly-vs-using-coo-tocsr-scipy
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)

from scipy.sparse import hstack

# Training data
X_tr = hstack((train_w2v_vectors_essays, train_w2v_vectors_titles, X_train_clean_category, X_train_clean_subcategories,
               X_train_project_grade, X_train_school_state, X_train_teacher_prefix,
               X_train_previous_projects, X_train_price, X_train_quantity)).tocsr()
# CV data
X_cr = hstack((cv_w2v_vectors_essays, cv_w2v_vectors_titles, X_cv_clean_category, X_cv_clean_subcategories,
               X_cv_project_grade, X_cv_school_state, X_cv_teacher_prefix,
               X_cv_previous_projects, X_cv_price, X_cv_quantity)).tocsr()
# Test data
X_te = hstack((test_w2v_vectors_essays, test_w2v_vectors_titles, X_test_clean_category, X_test_clean_subcategories,
               X_test_project_grade, X_test_school_state, X_test_teacher_prefix,
               X_test_previous_projects, X_test_price, X_test_quantity)).tocsr()
```

In [37]:
```
## Print the final data matrix

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

```
Final Data matrix
(20580, 202) (20580,)
(8820, 202) (8820,)
(12600, 202) (12600,)
```

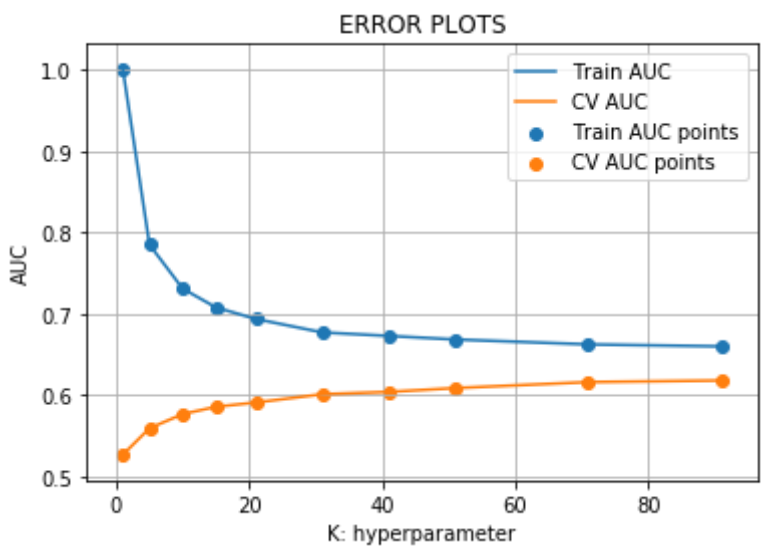
Hyper paramter tuning to find best K (Using Simple CV)

Applying KNN for multiple K values

In [38]:
```
# Reference : Assignment_SAMPLE_SOLUTION

def batch_predict(clf, data):

    """
    This function returns the predicted probability scores
    """

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000

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

    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [40]:
```
%%time
# Reference : https://stackoverflow.com/questions/32565829/simple-way-to-measure-cell-execution-time-in-ipython-notebook

# Trying various K values

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

# To store the roc_auc_scores of train and test data we need the two lists
train_auc = []
cv_auc = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 71, 91]

for i in tqdm(K):

    # Creating the classifier
    classifier = KNeighborsClassifier(n_neighbors = i, n_jobs = -1)
    # Fitting the train data
    classifier.fit(X_tr, y_train)

    # Getting the probability scores
    y_train_pred = batch_predict(classifier, X_tr)
    y_cv_pred = batch_predict(classifier, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # 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))


# Plotting the Error
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%|██████████| 10/10 [19:35<00:00, 117.46s/it]
```

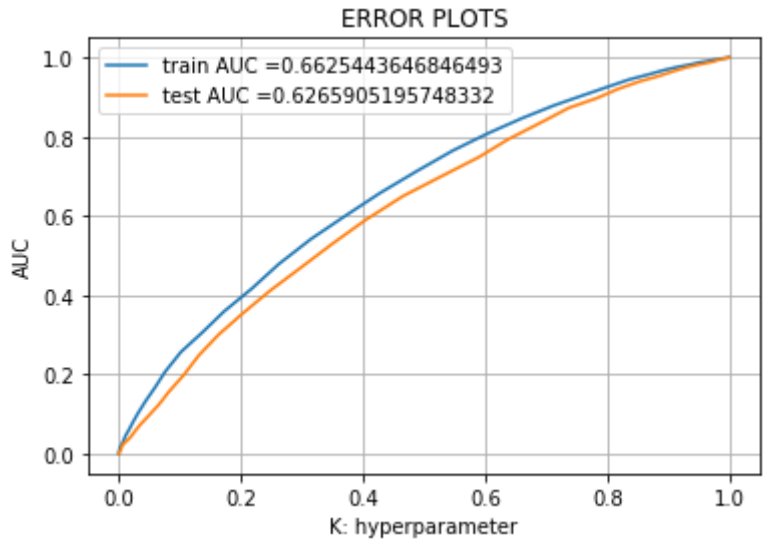



```
Wall time: 19min 35s
```

Now creating the model with best K

In [43]:
```
# From the error plot we choose K such that, we will have maximum AUC on cv data and gap between the train and cv is les

# Here we are choosing the best_k based on forloop results
best_k = 71
```

In [44]:
```
%%time

# Using the best_K

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

# Classifier
neigh = KNeighborsClassifier(n_neighbors = best_k)
# Fitting train data
neigh.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)


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


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




```
Wall time: 3min 27s
```

NOTE:

- As we can see from the graph the test AUC is close to the Train AUC
- The AUC scores for the Train and Test data are : 66 % and 62 % respectively
- We need more data to train the model so as to make it more accurate.

In [45]:
```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

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

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

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

In [46]:
```
from sklearn.metrics import confusion_matrix

print("="*120)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("="*120)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
========================================================================================================================
=
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2498545716144444 for threshold 0.592
[[4140 3945]
 [3583 8912]]
========================================================================================================================
=
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24922277318640956 for threshold 0.606
[[2645 2305]
 [2674 4976]]
```

NOTE:

1. As we can see from the confusion matrix that:

   ON THE TRAIN SET:
   - The number of correct predictions : 13052
   - The number of incorrect predictions : 7528

   ON THE TEST SET:
   - The number of correct predictions : 7621
   - The number of incorrect predictions : 4979

In [47]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

from sklearn.metrics import classification_report

y_pred_new = neigh.predict(X_te)
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred_new, target_names = target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.60      0.18      0.28      4950
     class 1       0.64      0.92      0.75      7650

   micro avg       0.63      0.63      0.63     12600
   macro avg       0.62      0.55      0.52     12600
weighted avg       0.62      0.63      0.57     12600
```

In [48]:
```
%%time
# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = neigh.predict(X_te)
print("Accuracy on test set: {}".format(accuracy_score(y_test, y_pred_new)))
print("Precision on test set: {}".format(precision_score(y_test, y_pred_new)))
print("Recall on test set: {}".format(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: {}".format(f1_score(y_test, y_pred_new)))
```

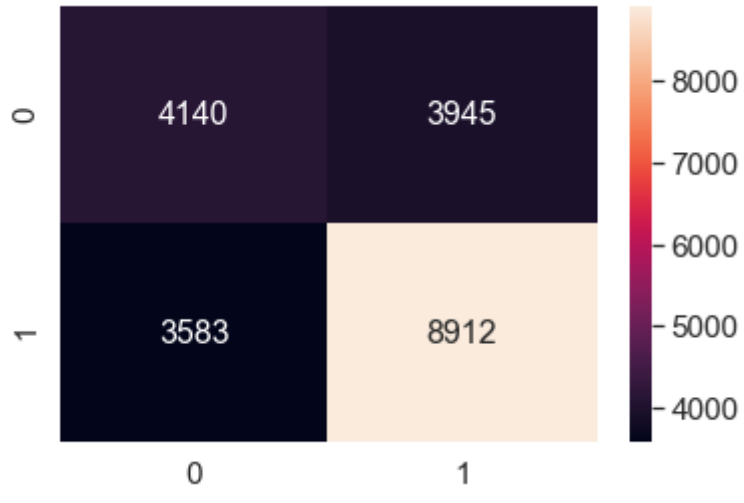
```
Accuracy on test set: 0.6306349206349207
Precision on test set: 0.63510101010101
Recall on test set: 0.9205228758169934
F1-Score on test set: 0.7516277084000427
Wall time: 1min 17s
```

Confusion Matrix on train data

In [49]:
```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

df_cm = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)), range(2),ran
sns.set(font_scale=1.4) #for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.2498545716144444 for threshold 0.592
```

Out[49]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22ee06a3f98>
```

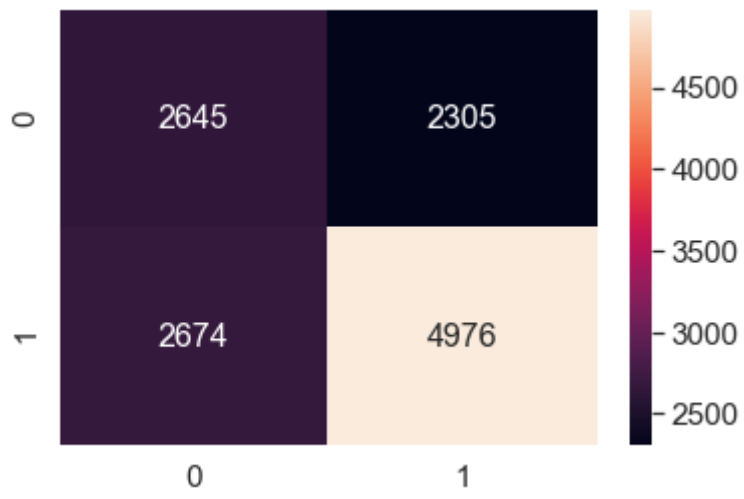


Confusion Matrix on test data

In [50]:
```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

df_cm = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)), range(2),range(2
sns.set(font_scale=1.4) #for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24922277318640956 for threshold 0.606
```

Out[50]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22ee078f518>
```



NOTE:

- Evaluation:
  1. Model managed to predict the test set results with an AUC of 62%
  2. Model is not robust as we need more data to train our model and make it more accurate

4. Applying KNN brute force on TFIDF W2V, SET 4

Merging the categorical, numerical and text features

In [36]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# https://stackoverflow.com/questions/54226138/constructing-sparse-csr-matrix-directly-vs-using-coo-tocsr-scipy
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)

from scipy.sparse import hstack

# Training data
X_tr = hstack((train_tfidf_w2v_essays, train_tfidf_w2v_titles, X_train_clean_category, X_train_clean_subcategories,
               X_train_project_grade, X_train_school_state, X_train_teacher_prefix,
               X_train_previous_projects, X_train_price, X_train_quantity)).tocsr()
# CV data
X_cr = hstack((cv_tfidf_w2v_essays, cv_tfidf_w2v_titles, X_cv_clean_category, X_cv_clean_subcategories,
               X_cv_project_grade, X_cv_school_state, X_cv_teacher_prefix,
               X_cv_previous_projects, X_cv_price, X_cv_quantity)).tocsr()
# Test data
X_te = hstack((test_tfidf_w2v_essays, test_tfidf_w2v_titles, X_test_clean_category, X_test_clean_subcategories,
               X_test_project_grade, X_test_school_state, X_test_teacher_prefix,
               X_test_previous_projects, X_test_price, X_test_quantity)).tocsr()
```

In [37]:
```
## Print the final data matrix

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

```
Final Data matrix
(20580, 202) (20580,)
(8820, 202) (8820,)
(12600, 202) (12600,)
```

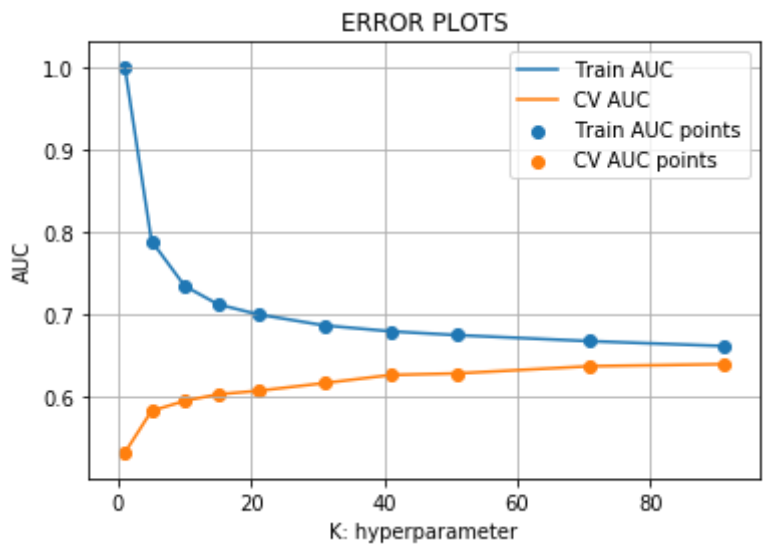
Hyper paramter tuning to find best K (Using Simple CV)

Applying KNN for multiple K values

In [38]:
```
# Reference : Assignment_SAMPLE_SOLUTION

def batch_predict(clf, data):

    """
    This function returns the predicted probability scores
    """

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000

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

    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [39]:
```
%%time
# Reference : https://stackoverflow.com/questions/32565829/simple-way-to-measure-cell-execution-time-in-ipython-notebook

# Trying various K values

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

# To store the roc_auc_scores of train and test data we need the two lists
train_auc = []
cv_auc = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 71, 91]

for i in tqdm(K):

    # Creating the classifier
    classifier = KNeighborsClassifier(n_neighbors = i, n_jobs = -1)
    # Fitting the train data
    classifier.fit(X_tr, y_train)

    # Getting the probability scores
    y_train_pred = batch_predict(classifier, X_tr)
    y_cv_pred = batch_predict(classifier, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # 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))


# Plotting the Error
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%|██████████| 10/10 [20:19<00:00, 118.17s/it]
```

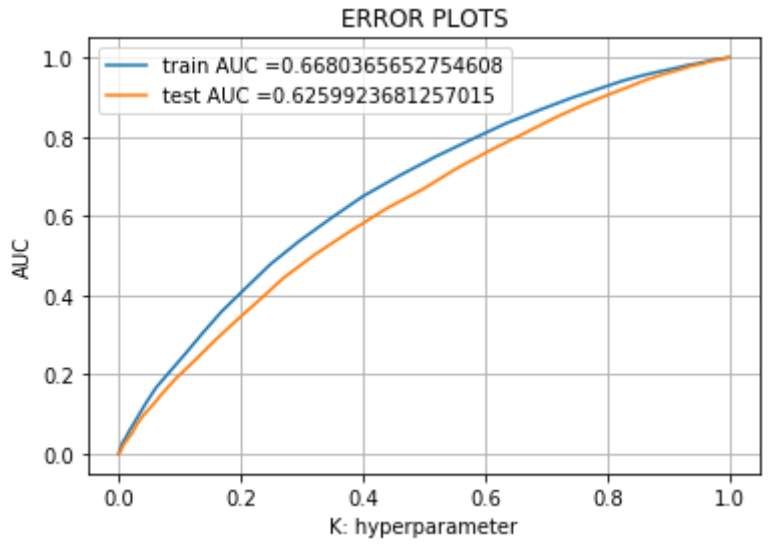



```
Wall time: 20min 19s
```

Now creating the model with best K

In [40]:
```
# From the error plot we choose K such that, we will have maximum AUC on cv data and gap between the train and cv is les

# Here we are choosing the best_k based on forloop results
best_k = 71
```

In [41]:
```
%%time

# Using the best_K

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

# Classifier
neigh = KNeighborsClassifier(n_neighbors = best_k)
# Fitting train data
neigh.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)


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


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




```
Wall time: 3min 12s
```

NOTE:

- As we can see from the graph the test AUC is close to the Train AUC
- The AUC scores for the Train and Test data are : 67 % and 62 % respectively
- We need more data to train the model so as to make it more accurate.

In [42]:
```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

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

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

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

In [43]:
```
from sklearn.metrics import confusion_matrix

print("="*120)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("="*120)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
========================================================================================================================
=
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24965808170684928 for threshold 0.577
[[3893 4192]
 [3128 9367]]
========================================================================================================================
=
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999591878379757 for threshold 0.606
[[2777 2173]
 [2911 4739]]
```

NOTE:

1. As we can see from the confusion matrix that:

   ON THE TRAIN SET:
   - The number of correct predictions : 13260
   - The number of incorrect predictions : 7320

   ON THE TEST SET:
   - The number of correct predictions : 7516
   - The number of incorrect predictions : 5084

In [44]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

from sklearn.metrics import classification_report

y_pred_new = neigh.predict(X_te)
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred_new, target_names = target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.58      0.20      0.30      4950
     class 1       0.64      0.90      0.75      7650

   micro avg       0.63      0.63      0.63     12600
   macro avg       0.61      0.55      0.52     12600
weighted avg       0.61      0.63      0.57     12600
```

In [45]:
```
%%time
# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = neigh.predict(X_te)
print("Accuracy on test set: {}".format(accuracy_score(y_test, y_pred_new)))
print("Precision on test set: {}".format(precision_score(y_test, y_pred_new)))
print("Recall on test set: {}".format(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: {}".format(f1_score(y_test, y_pred_new)))
```

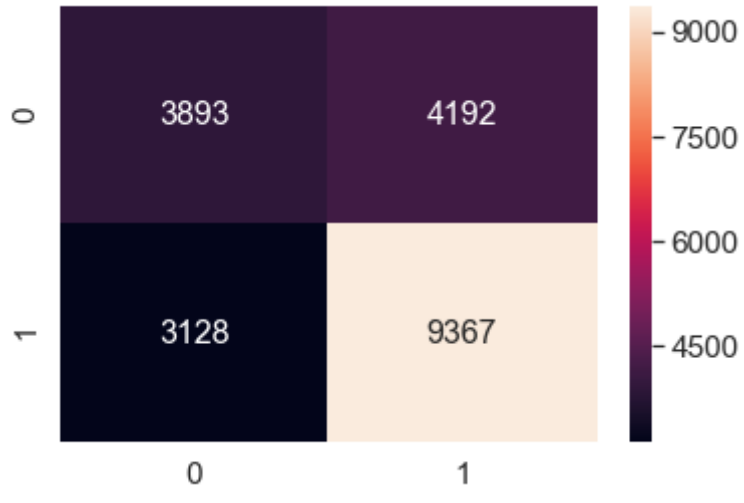
```
Accuracy on test set: 0.628095238095238
Precision on test set: 0.6364138438880707
Recall on test set: 0.9037908496732027
F1-Score on test set: 0.7468942421950956
Wall time: 1min 16s
```

Confusion Matrix on train data

In [46]:
```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

df_cm = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)), range(2),ran
sns.set(font_scale=1.4) #for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24965808170684928 for threshold 0.577
```

Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fcb6d767f0>
```

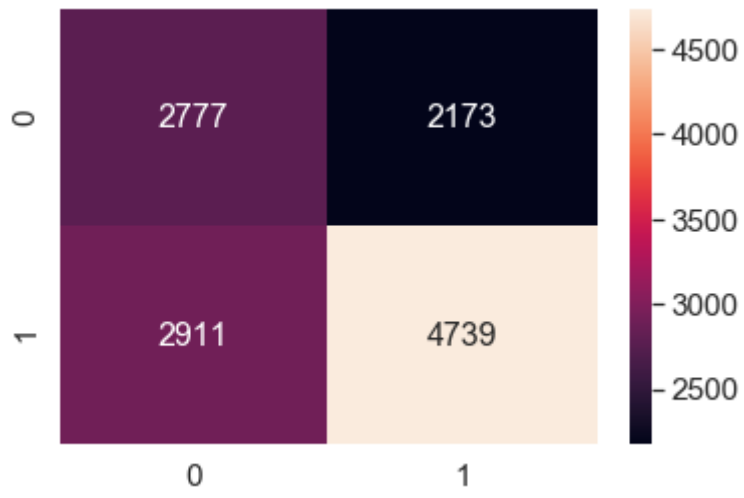


Confusion Matrix on test data

In [47]:
```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

df_cm = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)), range(2),range(2
sns.set(font_scale=1.4) #for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24999591878379757 for threshold 0.606
```

Out[47]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fcc6ba9710>
```



NOTE:

- Evaluation:
  1. Model managed to predict the test set results with an AUC of 62%
  2. Model is not robust as we need more data to train our model and make it more accurate

7. [TASK - 2] Selecting top 2000 features

1. clean_essay

In [32]:
```
%%time
# Vectorizing the essay column

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2

vectorizer = TfidfVectorizer(min_df=10)

# We will fit the train data only
vectorizer.fit(X_train['clean_essays'].values)

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


#Selecting top 2000 best features from the generated tfidf features
selector = SelectKBest(chi2, k = 2000 )
selector.fit(X_train_essay_tfidf, y_train)

X_train_essay_2000 = selector.transform(X_train_essay_tfidf)
X_cv_essay_2000 = selector.transform(X_cv_essay_tfidf)
X_test_essay_2000 = selector.transform(X_test_essay_tfidf)


print("Essay vectorized")
print(X_train_essay_2000.shape, y_train.shape)
print(X_cv_essay_2000.shape, y_cv.shape)
print(X_test_essay_2000.shape, y_test.shape)
```

```
Essay vectorized
(20580, 2000) (20580,)
(8820, 2000) (8820,)
(12600, 2000) (12600,)
Wall time: 8.61 s
```

2. clean_title

In [33]:
```
%%time
# Vectorizing the project_title column

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2

vectorizer = TfidfVectorizer(min_df=3)

# We will fit the train data only
vectorizer.fit(X_train['clean_titles'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_tfidf = vectorizer.transform(X_train['clean_titles'].values)
X_cv_titles_tfidf = vectorizer.transform(X_cv['clean_titles'].values)
X_test_titles_tfidf = vectorizer.transform(X_test['clean_titles'].values)


#Selecting top 2000 best features from the generated tfidf features
selector = SelectKBest(chi2, k = 2000)
selector.fit(X_train_titles_tfidf, y_train)

X_train_titles_2000 = selector.transform(X_train_titles_tfidf)
X_cv_titles_2000 = selector.transform(X_cv_titles_tfidf)
X_test_titles_2000 = selector.transform(X_test_titles_tfidf)


print("Titles vectorized")
print(X_train_titles_2000.shape, y_train.shape)
print(X_cv_titles_2000.shape, y_cv.shape)
print(X_test_titles_2000.shape, y_test.shape)
```

```
Titles vectorized
(20580, 2000) (20580,)
(8820, 2000) (8820,)
(12600, 2000) (12600,)
Wall time: 489 ms
```

Merging the categorical, numerical and text features

In [34]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# https://stackoverflow.com/questions/54226138/constructing-sparse-csr-matrix-directly-vs-using-coo-tocsr-scipy
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)

from scipy.sparse import hstack

# Training data
X_tr = hstack((X_train_essay_2000, X_train_titles_2000, X_train_clean_category, X_train_clean_subcategories,
               X_train_project_grade, X_train_school_state, X_train_teacher_prefix,
               X_train_previous_projects, X_train_price, X_train_quantity)).tocsr()
# CV data
X_cr = hstack((X_cv_essay_2000, X_cv_titles_2000, X_cv_clean_category, X_cv_clean_subcategories,
               X_cv_project_grade, X_cv_school_state, X_cv_teacher_prefix,
               X_cv_previous_projects, X_cv_price, X_cv_quantity)).tocsr()
# Test data
X_te = hstack((X_test_essay_2000, X_test_titles_2000, X_test_clean_category, X_test_clean_subcategories,
               X_test_project_grade, X_test_school_state, X_test_teacher_prefix,
               X_test_previous_projects, X_test_price, X_test_quantity)).tocsr()
```

In [35]:
```
## Print the final data matrix

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

```
Final Data matrix
(20580, 4102) (20580,)
(8820, 4102) (8820,)
(12600, 4102) (12600,)
```

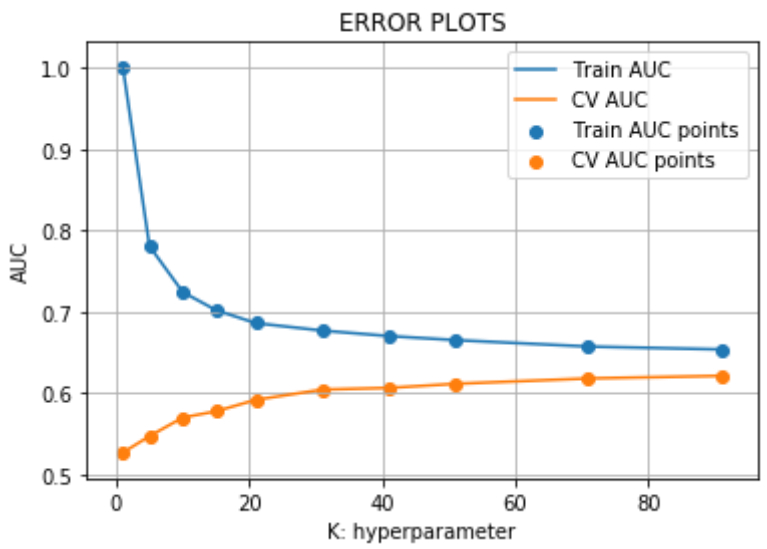
Hyper paramter tuning to find best K (Using Simple CV)

Applying KNN for multiple K values

In [36]:
```
# Reference : Assignment_SAMPLE_SOLUTION

def batch_predict(clf, data):

    """
    This function returns the predicted probability scores
    """

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000

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

    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [37]:
```
%%time
# Reference : https://stackoverflow.com/questions/32565829/simple-way-to-measure-cell-execution-time-in-ipython-notebook

# Trying various K values

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

# To store the roc_auc_scores of train and test data we need the two lists
train_auc = []
cv_auc = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 71, 91]

for i in tqdm(K):

    # Creating the classifier
    classifier = KNeighborsClassifier(n_neighbors = i, n_jobs = -1)
    # Fitting the train data
    classifier.fit(X_tr, y_train)

    # Getting the probability scores
    y_train_pred = batch_predict(classifier, X_tr)
    y_cv_pred = batch_predict(classifier, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # 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))


# Plotting the Error
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%|██████████| 10/10 [09:09<00:00, 55.33s/it]
```

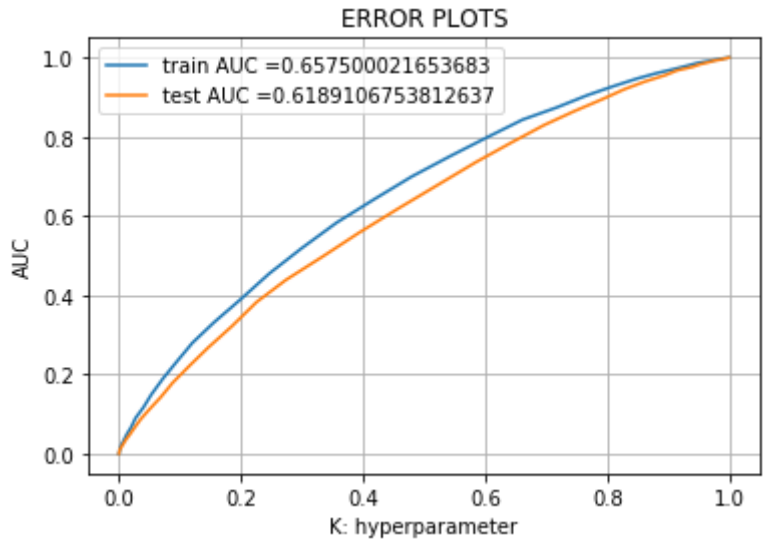



```
Wall time: 9min 9s
```

Now creating the model with best K

In [38]:
```
# From the error plot we choose K such that, we will have maximum AUC on cv data and gap between the train and cv is les

# Here we are choosing the best_k based on forloop results
best_k = 71
```

In [39]:
```
%%time

# Using the best_K

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

# Classifier
neigh = KNeighborsClassifier(n_neighbors = best_k)
# Fitting train data
neigh.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)


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


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




```
Wall time: 1min
```

NOTE:

- As we can see from the graph the test AUC is close to the Train AUC
- The AUC scores for the Train and Test data are : 65% and 62% respectively
- We need more data to train the model so as to make it more accurate.

In [40]:
```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

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

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

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

In [41]:
```
from sklearn.metrics import confusion_matrix

print("="*120)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("="*120)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
========================================================================================================================
=
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24958097884681507 for threshold 0.577
[[4208 3877]
 [3756 8739]]
========================================================================================================================
=
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24936230996837058 for threshold 0.606
[[3006 1944]
 [3388 4262]]
```

NOTE:

1. As we can see from the confusion matrix that:

   ON THE TRAIN SET:
   - The number of correct predictions : 12947
   - The number of incorrect predictions : 7633

   ON THE TEST SET:
   - The number of correct predictions : 7268
   - The number of incorrect predictions : 5332

In [43]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

from sklearn.metrics import classification_report

y_pred_new = neigh.predict(X_te)
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred_new, target_names = target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.56      0.21      0.30      4950
     class 1       0.64      0.90      0.74      7650

   micro avg       0.62      0.62      0.62     12600
   macro avg       0.60      0.55      0.52     12600
weighted avg       0.61      0.62      0.57     12600
```

In [44]:
```
%%time
# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = neigh.predict(X_te)
print("Accuracy on test set: {}".format(accuracy_score(y_test, y_pred_new)))
print("Precision on test set: {}".format(precision_score(y_test, y_pred_new)))
print("Recall on test set: {}".format(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: {}".format(f1_score(y_test, y_pred_new)))
```

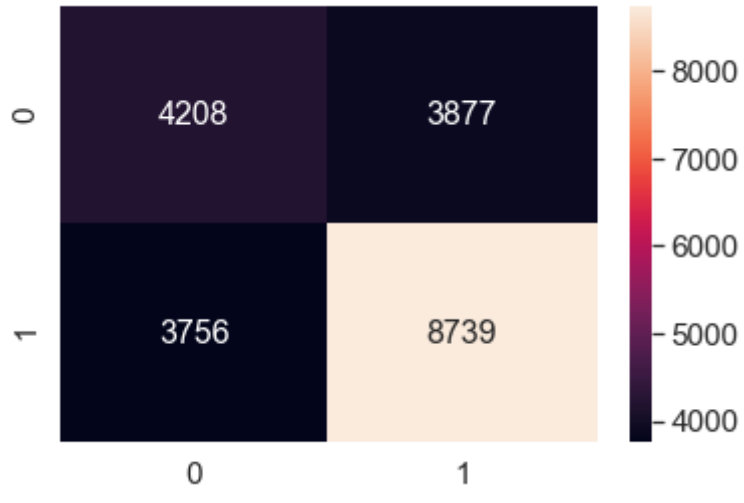
```
Accuracy on test set: 0.625
Precision on test set: 0.6357812645065454
Recall on test set: 0.8951633986928105
F1-Score on test set: 0.7434992671407633
Wall time: 23.7 s
```

Confusion Matrix on train data

In [45]:
```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

df_cm = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)), range(2),ran
sns.set(font_scale=1.4) #for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24958097884681507 for threshold 0.577
```

Out[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf981b77b8>
```

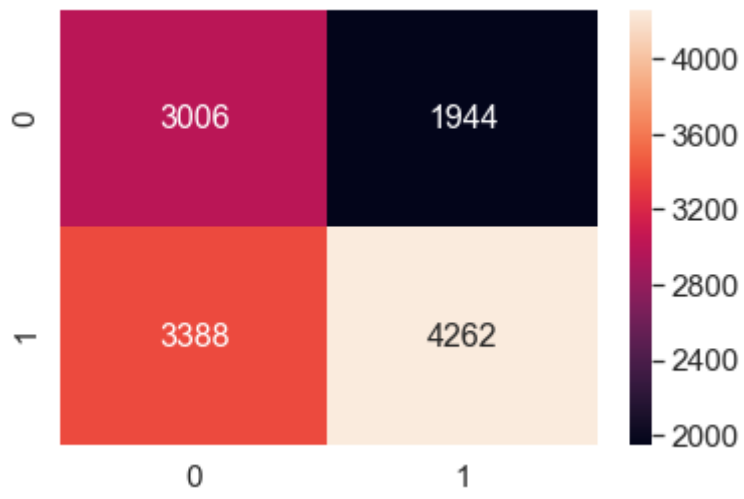


Confusion Matrix on test data

In [46]:
```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

df_cm = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)), range(2),range(2
sns.set(font_scale=1.4) #for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
the maximum value of tpr*(1-fpr) 0.24936230996837058 for threshold 0.606
```

Out[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bf843bc320>
```



NOTE:

- Evaluation:
1. Model managed to predict the test set results with an AUC of 62%
2. Model is not robust as we need more data to train our model and make it more accurate

CONCLUSION

In [48]:
```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "KNN Model [Algorithm]" ,"Hyper parameter [K-value]", "AUC Score"]
x.add_row(["Bag of Words", "Auto", 71, 0.65])
x.add_row(["TFIDF", "Auto", 71, 0.63])
x.add_row(["AVG W2V", "Auto", 71, 0.626])
x.add_row(["TFIDF weighted W2V", "Auto", 71, 0.625])
x.add_row(["TFIDF [2000 features]", "Auto", 71, 0.62])

print(x)
```

```
+----------------------+----------------------+---------------------------+-----------+
|      Vectorizer      | KNN Model [Algorithm] | Hyper parameter [K-value] | AUC Score |
+----------------------+----------------------+---------------------------+-----------+
|     Bag of Words     |         Auto         |             71            |    0.65   |
|        TFIDF         |         Auto         |             71            |    0.63   |
|       AVG W2V        |         Auto         |             71            |   0.626   |
|  TFIDF weighted W2V  |         Auto         |             71            |   0.625   |
| TFIDF [2000 features] |         Auto         |             71            |    0.62   |
+----------------------+----------------------+---------------------------+-----------+
```

NOTE:

1. As we can see that there is no significant decrease in the AUC score of TFIDF vectorizer when we select the top 2000 features
2. The hyper parameter i.e. the K-value in KNN is 61 for all the different vectorizers because on increase or decreasing this value the AUC value changes.
3. The GridSearchCV has been used to find the best K value in the KNN algorithm.
4. As we can see the BOW model has more AUC score than any of the other vectorizers
5. KNN for higher dimensionality data is not so usefull because it makes a lot of incorrect predictions. Also it doesn't give a high accuracy score.

In [ ]:

In [ ]: