Donors Choose - KNN

Importing packages

In [2]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore", category = UserWarning, mod
ule = '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
# Tutorial about Python regular expressions: https://pymotw.c
om/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
```

```
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

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

```
C:\Users\z004286m\AppData\Local\Continuum\anaconda3\lib\site-packages\smart_open\ssh.py:34: UserWarning: paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install paramiko` to suppress
  warnings.warn('paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install paramiko` to suppress')
C:\Users\z004286m\AppData\Local\Continuum\anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

Reading data

In [3]:

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

Project data

In [4]:

```
# Sorting data based on date

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

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

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

project_data.head(2)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	sch
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	

In [5]:

```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109248 entries, 55660 to 78306
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
Date
  109248 non-null datetime64[ns]
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: datetime64[ns](1), int64(3), object(13
)
memory usage: 15.0+ MB
```

In [6]:

```
approved_project = project_data[project_data["project_is_appr
oved"] == 1]
rejected_project = project_data[project_data["project_is_appr
oved"] == 0]

print('Number of approved projects :', approved_project.shape
)
print('Number of rejected projects :', rejected_project.shape
)
```

```
Number of approved projects : (92706, 17)
Number of rejected projects : (16542, 17)
```

Resource data

In [7]:

```

```

```
resource_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1541272 entries, 0 to 1541271
Data columns (total 4 columns):
id              1541272 non-null object
description     1540980 non-null object
quantity        1541272 non-null int64
price           1541272 non-null float64
dtypes: float64(1), int64(1), object(2)
memory usage: 47.0+ MB
```

Considering 5K Points (Random Sampling)

In [10]:

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

approved_project = project_data[project_data["project_is_approved"] == 1].sample(n = 35000, random_state = 1)
rejected_project = project_data[project_data["project_is_approved"] == 0].sample(n = 15000, random_state = 1)
project_data = pd.concat([approved_project, rejected_project])
)
```

Data Preprocessing

Preprocessing project subject categories

In [11]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-pyth
on/
# https://stackoverflow.com/questions/23669024/how-to-strip-a
-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whit
espace-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","&", "Scien
ce"
            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 placing all the ' '(sp
ace) 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())

# Adding a new column "clean_categories" and dropping the col
umn "project_subject_categories"
project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inp
lace=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]))
```

Preprocessing of project subject sub categories

In [12]:

```
sub_catogories = list(project_data['project_subject_subcatego
ries'].values)
# remove special characters from list of strings python: http
s://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-pyth
on/
# https://stackoverflow.com/questions/23669024/how-to-strip-a
-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whit
espace-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","&", "Scien
ce"
            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 ' '(s
pace) 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://stackoverf
low.com/a/22898595/4084039
from collections import Counter
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/6132
18/4084039
sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=l
ambda kv: kv[1]))
```

Price

In [13]:

```
# reference : 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()
price_data.head(2)
```

Out[13]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [14]:

```
# join two dataframes(project_data and price_data) in python
# reference : https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.merge.html
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

Preprocessing project essays

In [15]:

```
# merge two column text dataframe:
# reference : https://stackoverflow.com/questions/19377969/combine-two-columns-of-text-in-dataframe-in-pandas-python
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(s
```

```
tr) + \
                        project_data["project_essay_4"].map(s
tr)
```

In [16]:

```
# 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

# 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', 'b
etween', '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', "co
uldn'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"]


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

```
100%|
```

```
|████████████████████████████████| 50000/50000
 [00:36<00:00, 1381.51it/s]
```

In [17]:

```
# placing the preprocessed essay into the dataframe
project_data['clean_essays'] = preprocessed_essays
project_data.drop(['essay'], axis=1, inplace=True)
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)
```

Preprocessing project titles

In [18]:

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

```
100%|████████████████████████████████████████████████
|████████████████████████████████| 50000/50000
[00:01<00:00, 29655.96it/s]
```

In [19]:

```
# placing the preprocessed essay into the dataframe
project_data['clean_titles'] = preprocessed_titles
project_data.drop(['project_title'], axis=1, inplace=True)
```

Splitting the data and Starifying the samplimgs

In [20]:

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

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

```
(50000, 15)
(50000,)
```

In [21]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
#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=0.33, shuffle=Flase)# this is for time series split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify = y) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify = y_train) # this is random splitting

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

```
print(X_test.shape, y_test.shape)
```

```
(22445, 15) (22445,)
(11055, 15) (11055,)
(16500, 15) (16500,)
```

One-Hot encoding of 'Categorical features'

Clean categories

In [22]:

```
# we use count vectorizer to convert the values into one hot
encoded features

# Vectorizing "clean_categories"
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.
keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_categories'].values)

X_train_categories_one_hot = vectorizer.transform(X_train['cl
ean_categories'].values)
X_cv_categories_one_hot = vectorizer.transform(X_cv['clean_ca
tegories'].values)
X_test_categories_one_hot = vectorizer.transform(X_test['clea
n_categories'].values)

print("After verctorizing")
print(X_train_categories_one_hot.shape, y_train.shape)
print(X_cv_categories_one_hot.shape, y_cv.shape)
print(X_test_categories_one_hot.shape, y_test.shape)

print(vectorizer.get_feature_names())
```

```
After verctorizing
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'M
usic_Arts', 'AppliedLearning', 'SpecialNeeds',
 'Health_Sports', 'Math_Science', 'Literacy_La
nguage']
```

Clean sub categories

In [23]:

```
# Vectorizing "clean_subcategories"
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_d
ict.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)

X_train_sub_categories_one_hot = vectorizer.transform(X_train
['clean_subcategories'].values)
X_cv_sub_categories_one_hot = vectorizer.transform(X_cv['clea
n_subcategories'].values)
X_test_sub_categories_one_hot = vectorizer.transform(X_test['
clean_subcategories'].values)

print("After verctorizing")
print(X_train_sub_categories_one_hot.shape, y_train.shape)
print(X_cv_sub_categories_one_hot.shape, y_cv.shape)
print(X_test_sub_categories_one_hot.shape, y_test.shape)

print(vectorizer.get_feature_names())
```

```
After verctorizing
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['Economics', 'CommunityService', 'FinancialLi
teracy', 'ParentInvolvement', 'Civics_Governme
nt', 'Extracurricular', 'ForeignLanguages', 'W
armth', 'Care_Hunger', 'NutritionEducation', '
```

```
SocialSciences', 'PerformingArts', 'CharacterE
ducation', 'TeamSports', 'Other', 'College_Car
eerPrep', 'Music', 'History_Geography', 'ESL',
 'EarlyDevelopment', 'Health_LifeScience', 'Gy
m_Fitness', 'EnvironmentalScience', 'VisualArt
s', 'Health_Wellness', 'AppliedSciences', 'Spe
cialNeeds', 'Literature_Writing', 'Mathematics
', 'Literacy']
```

Teacher Prefixes

In [24]:

```
# Vectorizing "teacher_prefix"
prefix = list(set(X_train['teacher_prefix'].values))

vectorizer = CountVectorizer(vocabulary=prefix, lowercase=Fal
se, binary=True)
vectorizer.fit(X_train['teacher_prefix'].values)

X_train_prefix_one_hot = vectorizer.transform(X_train['teache
r_prefix'])
X_cv_prefix_one_hot = vectorizer.transform(X_cv['teacher_pref
ix'])
X_test_prefix_one_hot = vectorizer.transform(X_test['teacher_
prefix'])

print("After verctorizing")
print(X_train_prefix_one_hot.shape, y_train.shape)
print(X_cv_prefix_one_hot.shape, y_cv.shape)
print(X_test_prefix_one_hot.shape, y_test.shape)

print(vectorizer.get_feature_names())
```

```
After verctorizing
(22445, 5) (22445,)
```

```
(11055, 5) (11055,)
(16500, 5) (16500,)
['Ms.', 'Mr.', 'Teacher', 'Mrs.', 'Dr.']
```

School state

In [25]:

```
# Vectorizing "school_state"
from collections import Counter
my_counter = Counter()
for word in X_train['school_state'].values:
    my_counter.update(word.split())

state_dict = dict(my_counter)
sorted_state_dict = dict(sorted(state_dict.items(), key=lambd
a kv: kv[1]))

vectorizer = CountVectorizer(vocabulary=list(sorted_state_dic
t.keys()), lowercase=False, binary=True)
vectorizer.fit(X_train['school_state'].values)

X_train_state_one_hot = vectorizer.transform(X_train['school_
state'].values)
X_cv_state_one_hot = vectorizer.transform(X_cv['school_state'
].values)
X_test_state_one_hot = vectorizer.transform(X_test['school_st
ate'].values)

print("After verctorizing")
print(X_train_state_one_hot.shape, y_train.shape)
print(X_cv_state_one_hot.shape, y_cv.shape)
print(X_test_state_one_hot.shape, y_test.shape)

print(vectorizer.get_feature_names())
```

```
After verctorizing
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['VT', 'WY', 'ND', 'MT', 'RI', 'NE', 'NH', 'DE
', 'AK', 'SD', 'WV', 'ME', 'HI', 'NM', 'DC', '
IA', 'KS', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY',
 'MS', 'NV', 'MD', 'CT', 'AL', 'TN', 'UT', 'WI
', 'WA', 'VA', 'NJ', 'AZ', 'MA', 'OK', 'OH', '
IN', 'MO', 'LA', 'PA', 'MI', 'SC', 'IL', 'GA',
 'NC', 'FL', 'NY', 'TX', 'CA']
```

Project grade category

In [26]:

```
# Vectorizing "project_grade_category"
prefix = list(set(X_train["project_grade_category"].values))

vectorizer = CountVectorizer(vocabulary=prefix, lowercase=Fal
se, binary=True)
vectorizer.fit(X_train['project_grade_category'])

X_train_grade_one_hot = vectorizer.transform(X_train['project
_grade_category'])
X_cv_grade_one_hot = vectorizer.transform(X_cv['project_grade
_category'])
X_test_grade_one_hot = vectorizer.transform(X_test['project_g
rade_category'])

print("After verctorizing")
print(X_train_grade_one_hot.shape, y_train.shape)
print(X_cv_grade_one_hot.shape, y_cv.shape)
print(X_test_grade_one_hot.shape, y_test.shape)
```

```
print(vectorizer.get_feature_names())
```

```
After verctorizing
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['Grades PreK-2', 'Grades 9-12', 'Grades 3-5',
 'Grades 6-8']
```

Normalizing numerical features

Price

In [27]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4
&t=530s
# standardization sklearn: https://scikit-learn.org/stable/mo
dules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price
'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=
[725.05 213.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()
price_scalar.fit(X_train['price'].values.reshape(-1,1)) # fin
ding 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.
train_price_standardized = price_scalar.transform(X_train['pr
ice'].values.reshape(-1, 1))
cv_price_standardized = price_scalar.transform(X_cv['price'].
values.reshape(-1, 1))
test_price_standardized = price_scalar.transform(X_test['pric
e'].values.reshape(-1, 1))
```

```
print('After Vectorizing')
print(train_price_standardized.shape, y_train.shape)
print(cv_price_standardized.shape, y_cv.shape)
print(test_price_standardized.shape, y_test.shape)
```

```
Mean : 312.6973811539319, Standard deviation :
 389.3271092984677
After Vectorizing
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Number of previously posted assignments

In [28]:

```
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

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

# Now standardize the data with above maen and variance.
train_teacher_number_of_previously_posted_projects_standardized =teacher_number_of_previously_posted_projects_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].
```

```
values.reshape(-1, 1))
cv_teacher_number_of_previously_posted_projects_standardized
=teacher_number_of_previously_posted_projects_scalar.transform
(X_cv['teacher_number_of_previously_posted_projects'].values.
reshape(-1, 1))
test_teacher_number_of_previously_posted_projects_standardize
d =teacher_number_of_previously_posted_projects_scalar.transf
orm(X_test['teacher_number_of_previously_posted_projects'].va
lues.reshape(-1, 1))

print('After Vectorizing')
print(train_teacher_number_of_previously_posted_projects_stan
dardized.shape, y_train.shape)
print(cv_teacher_number_of_previously_posted_projects_standar
dized.shape, y_cv.shape)
print(test_teacher_number_of_previously_posted_projects_stand
ardized.shape, y_test.shape)
```

```
Mean : 10.561728670082424, Standard deviation
: 26.514559461562783
After Vectorizing
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Resource quantity

In [29]:

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

```
# Now standardize the data with above maen and variance.
train_quantity_standardized = quantity_scalar.transform(X_tra
in['quantity'].values.reshape(-1, 1))
cv_quantity_standardized = quantity_scalar.transform(X_cv['qu
antity'].values.reshape(-1, 1))
test_quantity_standardized = quantity_scalar.transform(X_test
['quantity'].values.reshape(-1, 1))

print('After Vectorizing')
print(train_quantity_standardized.shape, y_train.shape)
print(cv_quantity_standardized.shape, y_cv.shape)
print(test_quantity_standardized.shape, y_test.shape)
```

```
Mean : 17.801113833815993, Standard deviation
: 27.602493076997202
After Vectorizing
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Functions:

In [30]:

```
def batch_predict(clf, data):
    # 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 wil
l be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 mul
tiplier
    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]:

```
def get_confusion_matrix(clf,X_te,y_test):
    y_pred = clf.predict(X_te)
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), ra
nge(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt
```

```
='g')
```

KNN on Set 1: categorical, numerical features + Text(BOW)

Text vectorization:BoW

Bow on essays

In [31]:

```
# We are considering only the words which appeared in at leas
t 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10, max_features = 6000)
vectorizer.fit(X_train['clean_essays'].values)

X_train_essay_bow = vectorizer.transform(X_train['clean_essay
s'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['clean_essays'].va
lues)
X_test_essay_bow = vectorizer.transform(X_test['clean_essays'
].values)

print("After vectorizing")
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)
```

```
After vectorizing
(22445, 6000) (22445,)
(11055, 6000) (11055,)
(16500, 6000) (16500,)
```

Bow on titles

In [32]:

```
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train['clean_titles'].values)

X_train_titles_bow = vectorizer.transform(X_train['clean_titl
es'].values)
X_cv_titles_bow = vectorizer.transform(X_cv['clean_titles'].v
alues)
X_test_titles_bow = vectorizer.transform(X_test['clean_titles
'].values)

print("After vectorizing")
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)
```

```
After vectorizing
(22445, 1212) (22445,)
(11055, 1212) (11055,)
(16500, 1212) (16500,)
```

BoW on Project resource summary

In [33]:

```
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train['project_resource_summary'].values) #
fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to ve
ctor
X_train_summary_bow = vectorizer.transform(X_train['project_r
```

```
esource_summary'].values)
X_cv_summary_bow = vectorizer.transform(X_cv['project_resourc
e_summary'].values)
X_test_summary_bow = vectorizer.transform(X_test['project_res
ource_summary'].values)

print("After vectorizations")
print(X_train_summary_bow.shape, y_train.shape)
print(X_cv_summary_bow.shape, y_cv.shape)
print(X_test_summary_bow.shape, y_test.shape)
```

```
After vectorizations
(22445, 2565) (22445,)
(11055, 2565) (11055,)
(16500, 2565) (16500,)
```

Concatinating all the features

In [34]:

```
# merge two sparse matrices: https://stackoverflow.com/a/1971
0648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse
 matrix and a dense matirx :)
X_train_bow = hstack((X_train_categories_one_hot, X_train_sub
_categories_one_hot, X_train_essay_bow, train_quantity_standa
rdized, X_train_state_one_hot, X_train_prefix_one_hot, X_trai
n_grade_one_hot, X_train_titles_bow, train_price_standardized
, train_teacher_number_of_previously_posted_projects_standard
ized, X_train_grade_one_hot,X_train_summary_bow)).tocsr()
X_cv_bow = hstack((X_cv_categories_one_hot, X_cv_sub_categori
es_one_hot, X_cv_essay_bow, cv_quantity_standardized, X_cv_st
ate_one_hot, X_cv_prefix_one_hot, X_cv_grade_one_hot, X_cv_ti
tles_bow, cv_price_standardized, cv_teacher_number_of_previou
sly_posted_projects_standardized, X_cv_grade_one_hot,X_cv_sum
```

```
mary_bow)).tocsr()
X_test_bow = hstack((X_test_categories_one_hot, X_test_sub_ca
tegories_one_hot, X_test_essay_bow, test_quantity_standardize
d, X_test_state_one_hot, X_test_prefix_one_hot, X_test_grade_
one_hot, X_test_titles_bow, test_price_standardized, test_tea
cher_number_of_previously_posted_projects_standardized, X_tes
t_grade_one_hot,X_test_summary_bow)).tocsr()

print('Final matrix')
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
```

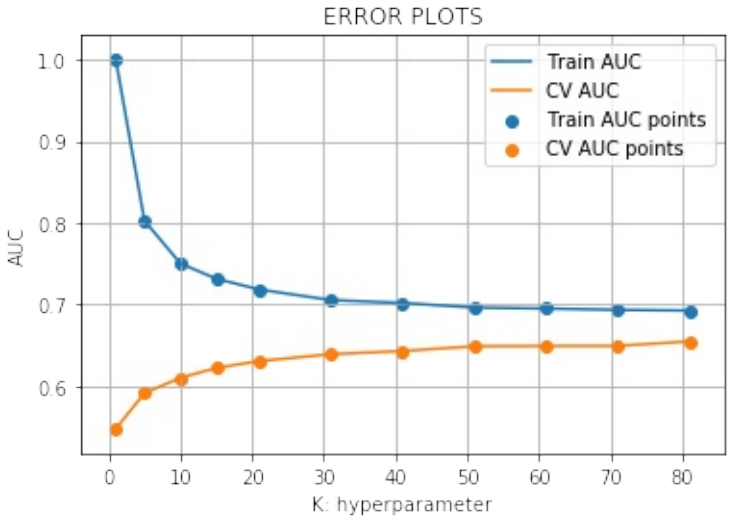
```
Final matrix
(22445, 9883) (22445,)
(11055, 9883) (11055,)
(16500, 9883) (16500,)
```

Hyper Parameter tuning using simple For loop

In [35]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

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

```
 class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 61, 71, 81]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs = -1)
    neigh.fit(X_train_bow, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should
 be probability estimates of the positive class
    # not the predicted outputs
    y_train_pred = batch_predict(neigh, X_train_bow)
    y_cv_pred = batch_predict(neigh, X_cv_bow)

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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

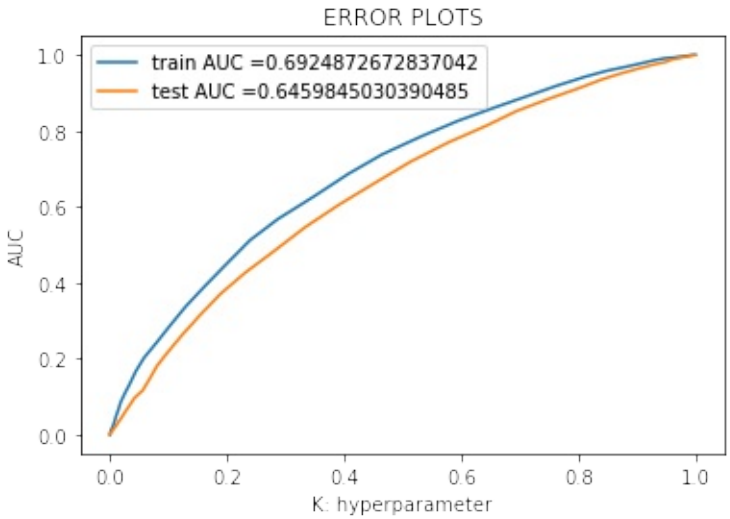


Plotting error plots: Testing on Test data

In [36]:

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


neigh = KNeighborsClassifier(n_neighbors = 81)
neigh.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, batch_predict(neigh, X_train_bow))
test_fpr, test_tpr, thresholds = roc_curve(y_test, batch_predict(neigh, X_test_bow))

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.show()
```



Confusion matrix

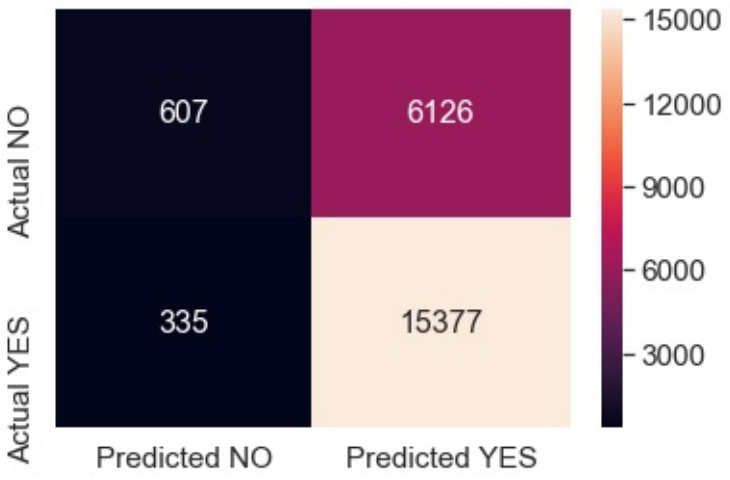
In [37]:

```
# Plotting confusion matrix of train and test set
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_bow)))
```

```
Train confusion matrix
[[  607  6126]
 [  335 15377]]
Test confusion matrix
[[  348  4602]
 [  288 11262]]
```

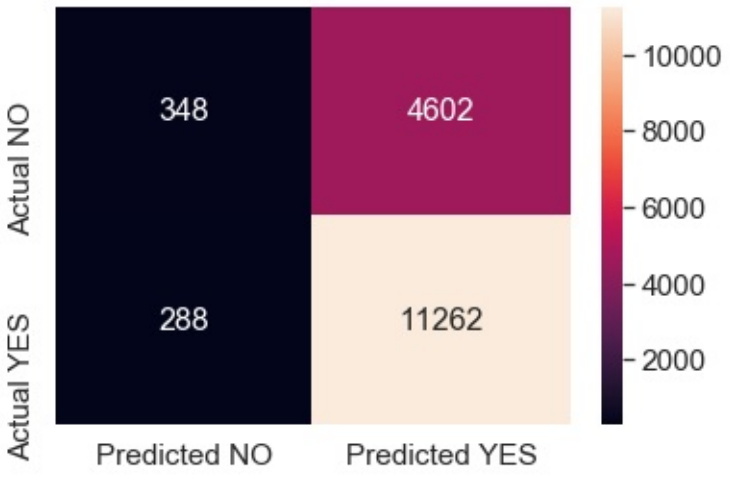
In [40]:

```
get_confusion_matrix(neigh,X_train_bow,y_train)
```



In [41]:

```
get_confusion_matrix(neigh,X_test_bow,y_test)
```



Evaluating Model Performance

In [42]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = neigh.predict(X_test_bow)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test,
 y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test,
 y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pre
d_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_
new)))
```

```
Accuracy on test set: 70.364%
Precision on test set: 0.710
Recall on test set: 0.975
F1-Score on test set: 0.822
```

KNN on Set 2: categorical, numerical features + Text(TFIDF)

Text vectorization:tfidf

TFIDF on essays

In [43]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(min_df=10, max_features = 6000)
vectorizer.fit(X_train['clean_essays'].values)

X_train_essay_tfidf = vectorizer.transform(X_train['clean_ess
ays'])
X_cv_essay_tfidf = vectorizer.transform(X_cv['clean_essays'])
X_test_essay_tfidf = vectorizer.transform(X_test['clean_essay
s'])

print("After vectorizing")
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)
```

```
After vectorizing
(22445, 6000) (22445,)
(11055, 6000) (11055,)
(16500, 6000) (16500,)
```

TFIDF on titles

In [44]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['clean_titles'].values)

X_train_title_tfidf = vectorizer.transform(X_train['clean_tit
les'])
X_cv_title_tfidf = vectorizer.transform(X_cv['clean_titles'])
X_test_title_tfidf = vectorizer.transform(X_test['clean_title
s'])

print("After vectorizing")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
```

```
After vectorizing
(22445, 1212) (22445,)
(11055, 1212) (11055,)
(16500, 1212) (16500,)
```

TFIDF on Summary

In [45]:

```
vectorizer = TfidfVectorizer(min_df=5)
vectorizer.fit(X_train['project_resource_summary'].values) #
fit has to happen only on train datadata

# we use the fitted CountVectorizer to convert the text to ve
ctor
X_train_summary_tfidf = vectorizer.transform(X_train['project
```

```
_resource_summary'].values)
X_cv_summary_tfidf = vectorizer.transform(X_cv['project_resou
rce_summary'].values)
X_test_summary_tfidf = vectorizer.transform(X_test['project_r
esource_summary'].values)

print("After vectorizations")
print(X_train_summary_tfidf.shape, y_train.shape)
print(X_cv_summary_tfidf.shape, y_cv.shape)
print(X_test_summary_tfidf.shape, y_test.shape)
```

```
After vectorizations
(22445, 3952) (22445,)
(11055, 3952) (11055,)
(16500, 3952) (16500,)
```

Concatinating all the features

In [46]:

```
# merge two sparse matrices: https://stackoverflow.com/a/1971
0648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse
 matrix and a dense matirx :)
X_train_tfidf = hstack((X_train_categories_one_hot, X_train_s
ub_categories_one_hot, X_train_essay_tfidf, train_quantity_st
andardized, X_train_state_one_hot, X_train_prefix_one_hot, X_
train_grade_one_hot, X_train_title_tfidf, train_price_standar
dized, train_teacher_number_of_previously_posted_projects_sta
ndardized, X_train_grade_one_hot,X_train_summary_tfidf)).tocs
r()
X_cv_tfidf = hstack((X_cv_categories_one_hot, X_cv_sub_catego
ries_one_hot, X_cv_essay_tfidf, cv_quantity_standardized, X_c
v_state_one_hot, X_cv_prefix_one_hot, X_cv_grade_one_hot, X_c
v_title_tfidf, cv_price_standardized, cv_teacher_number_of_pr
```

```
eviously_posted_projects_standardized, X_cv_grade_one_hot,X_c
v_summary_tfidf)).tocsr()
X_test_tfidf = hstack((X_test_categories_one_hot, X_test_sub_
categories_one_hot, X_test_essay_tfidf, test_quantity_standar
dized, X_test_state_one_hot, X_test_prefix_one_hot, X_test_gr
ade_one_hot, X_test_title_tfidf, test_price_standardized, tes
t_teacher_number_of_previously_posted_projects_standardized,
X_test_grade_one_hot,X_test_summary_tfidf)).tocsr()

print('Final matrix')
print(X_train_tfidf.shape, y_train.shape)
print(X_cv_tfidf.shape, y_cv.shape)
print(X_test_tfidf.shape, y_test.shape)
```

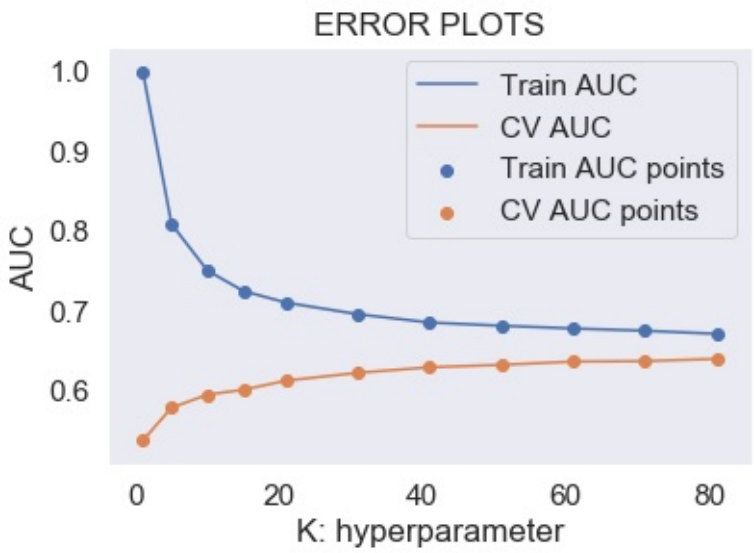
```
Final matrix
(22445, 11270) (22445,)
(11055, 11270) (11055,)
(16500, 11270) (16500,)
```

Hyperparameter tunning using simple For loop

In [47]:

```
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 61, 71, 81]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs = -1)
    neigh.fit(X_train_tfidf, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should
 be probability estimates of the positive class
    # not the predicted outputs
    y_train_pred = batch_predict(neigh, X_train_tfidf)
    y_cv_pred = batch_predict(neigh, X_cv_tfidf)

    train_auc.append(roc_auc_score(y_train,y_train_pred))
```

```
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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



Plotting error plot:Testing on Test set

In [48]:

```
neigh = KNeighborsClassifier(n_neighbors = 81)
neigh.fit(X_train_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be
probability estimates of the positive class
```

```
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, batch_p
redict(neigh, X_train_tfidf))
test_fpr, test_tpr, thresholds = roc_curve(y_test, batch_pred
ict(neigh, X_test_tfidf))

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(tr
ain_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.show()
```

Confusion matrix

In [49]:

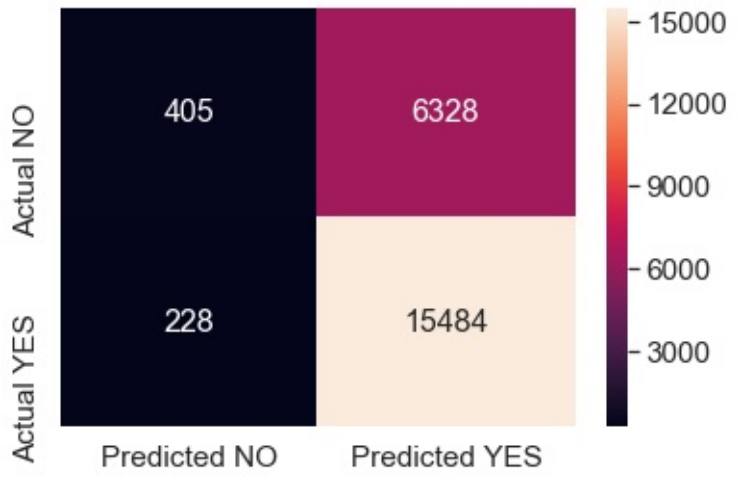
```
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_tfidf))
```

```
)
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tfidf)))
```

```
Train confusion matrix
[[  405  6328]
 [  228 15484]]
Test confusion matrix
[[  264  4686]
 [  213 11337]]
```

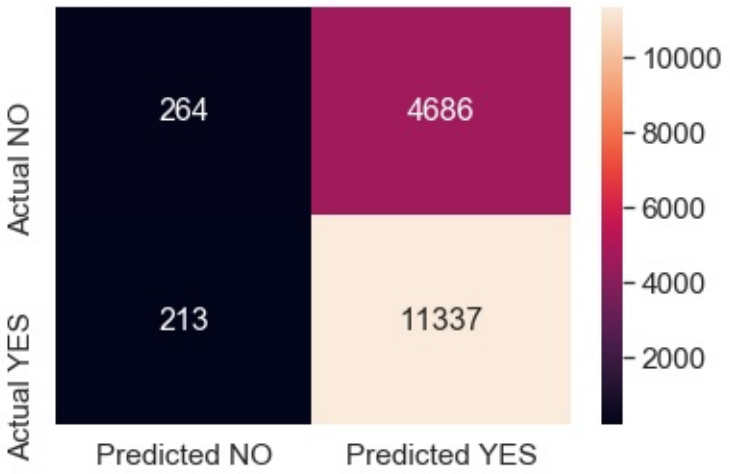
In [51]:

```
get_confusion_matrix(neigh,X_train_tfidf,y_train)
```



In [52]:

```
get_confusion_matrix(neigh,X_test_tfidf,y_test)
```



Evaluating model performance

In [53]:

```
y_pred_new = neigh.predict(X_test_tfidf)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test,
 y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test,
 y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pre
d_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_
new)))
```

```
Accuracy on test set: 70.309%
Precision on test set: 0.708
Recall on test set: 0.982
F1-Score on test set: 0.822
```

KNN on Set 3: categorical, numerical features + Text(AVG W2V)

Text Vectorization:Avg w2v

In [54]:

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


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

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

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

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


# stronging variables into pickle files python: http://www.je
ssicayung.com/how-to-use-pickle-to-save-and-load-variables-in
-python/

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

```
Loading Glove Model
```

```
1917494it [07:45, 4116.59it/s]
```

```
Done. 1917494  words loaded!
all the words in the coupus 7721384
the unique words in the coupus 43510
The number of words that are present in both g
love vectors and our coupus 39582 ( 90.972 %)
word 2 vec length 39582
```

In [55]:

```
# stronging variables into pickle files python: http://www.je
ssicayung.com/how-to-use-pickle-to-save-and-load-variables-in
-python/
```

```
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

Avg w2v on essays

In [56]:

```
train_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in words_glove:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_essay_avg_w2v.append(vector)

print(len(train_essay_avg_w2v))
print(len(train_essay_avg_w2v[0]))


cv_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
```

```
    cnt_words =0; # num of words with a valid vector in the s
entence/review
    for word in sentence.split(): # for each word in a review
/sentence
        if word in words_glove:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_essay_avg_w2v.append(vector)

print(len(cv_essay_avg_w2v))
print(len(cv_essay_avg_w2v[0]))


test_essay_avg_w2v = []; # the avg-w2v for each sentence/revi
ew is stored in this list
for sentence in tqdm(X_test['clean_essays']): # for each revi
ew/sentence
    vector = np.zeros(300) # as word vectors are of zero leng
th
    cnt_words =0; # num of words with a valid vector in the s
entence/review
    for word in sentence.split(): # for each word in a review
/sentence
        if word in words_glove:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_essay_avg_w2v.append(vector)

print(len(test_essay_avg_w2v))
print(len(test_essay_avg_w2v[0]))
```

```
100%|██████████████████████████████████████████████████
██████████████████████████████| 22445/22445
```

```
 [00:11<00:00, 1945.96it/s]
```

```
22445
300
```

```
100%|██████████████████████████████████████████████████| 11055/11055
 [00:05<00:00, 2062.07it/s]
```

```
11055
300
```

```
100%|██████████████████████████████████████████████████| 16500/16500
 [00:07<00:00, 2122.92it/s]
```

```
16500
300
```

In [57]:

```
# Changing list to numpy arrays
train_essay_avg_w2v = np.array(train_essay_avg_w2v)
cv_essay_avg_w2v = np.array(cv_essay_avg_w2v)
test_essay_avg_w2v = np.array(test_essay_avg_w2v)
```

Avg w2v on titles

In [58]:

```
train_title_avg_w2v = []; # the avg-w2v for each sentence/rev
iew is stored in this list
for sentence in tqdm(X_train['clean_titles']): # for each rev
iew/sentence
    vector = np.zeros(300) # as word vectors are of zero leng
th
    cnt_words =0; # num of words with a valid vector in the s
```

```
entence/review
    for word in sentence.split(): # for each word in a review
/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_title_avg_w2v.append(vector)

print(len(train_title_avg_w2v))
print(len(train_title_avg_w2v[0]))


cv_title_avg_w2v = []; # the avg-w2v for each sentence/review
 is stored in this list
for sentence in tqdm(X_cv['clean_titles']): # for each review
/sentence
    vector = np.zeros(300) # as word vectors are of zero leng
th
    cnt_words =0; # num of words with a valid vector in the s
entence/review
    for word in sentence.split(): # for each word in a review
/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_title_avg_w2v.append(vector)

print(len(cv_title_avg_w2v))
print(len(cv_title_avg_w2v[0]))


test_title_avg_w2v = []; # the avg-w2v for each sentence/revi
ew is stored in this list
```

```
for sentence in tqdm(X_test['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_title_avg_w2v.append(vector)

print(len(test_title_avg_w2v))
print(len(test_title_avg_w2v[0]))
```

```
100%|██████████████████████████████████████████████████| 22445/22445 [00:00<00:00, 32123.18it/s]
```

```
22445
300
```

```
100%|██████████████████████████████████████████████████| 11055/11055 [00:00<00:00, 37729.95it/s]
```

```
11055
300
```

```
100%|██████████████████████████████████████████████████| 16500/16500 [00:00<00:00, 31309.51it/s]
```

```
16500
300
```

In [59]:

```
# Changing list to numpy arrays
train_title_avg_w2v = np.array(train_title_avg_w2v)
cv_title_avg_w2v = np.array(cv_title_avg_w2v)
test_title_avg_w2v = np.array(test_title_avg_w2v)
```

In [60]:

```
# average Word2Vec
# compute average word2vec for each review.
train_w2v_vectors_summary = []; # the avg-w2v for each essay
is stored in this list
for sentence in tqdm(X_train['project_resource_summary'].valu
es): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero lengt
h
    cnt_words =0; # num of words with a valid vector in the e
ssay
    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_summary.append(vector)
print("train  vector")
print(len(train_w2v_vectors_summary))
print(len(train_w2v_vectors_summary[0]))


# average Word2Vec
# compute average word2vec for each review.
test_w2v_vectors_summary = []; # the avg-w2v for each essay i
s stored in this list
for sentence in tqdm(X_test['project_resource_summary'].value
s): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero lengt
```

```
h
    cnt_words =0; # num of words with a valid vector in the e
ssay
    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_summary.append(vector)

print("Test vec")
print(len(test_w2v_vectors_summary))
print(len(test_w2v_vectors_summary[0]))


# average Word2Vec
# compute average word2vec for each review.
cv_w2v_vectors_summary = []; # the avg-w2v for each essay is
stored in this list
for sentence in tqdm(X_cv['project_resource_summary'].values)
: # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero lengt
h
    cnt_words =0; # num of words with a valid vector in the e
ssay
    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_summary.append(vector)

print("CV vec")
print(len(cv_w2v_vectors_summary))
print(len(cv_w2v_vectors_summary[0]))
```

```
100%|██████████████████████████████████████████████████| 22445/22445
 [00:06<00:00, 3586.29it/s]
```

```
train  vector
22445
50
```

```
100%|██████████████████████████████████████████████████| 16500/16500
 [00:01<00:00, 8535.16it/s]
```

```
Test vec
16500
50
```

```
100%|██████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 11293.22it/s]
```

```
CV vec
11055
50
```

In [61]:

```
# Changing list to numpy arrays
train_w2v_vectors_summary = np.array(train_w2v_vectors_summar
y)
test_w2v_vectors_summary = np.array(test_w2v_vectors_summary)
cv_w2v_vectors_summary = np.array(cv_w2v_vectors_summary)
```

Concatinating all the features

In [63]:

```
# merge two sparse matrices: https://stackoverflow.com/a/1971
```

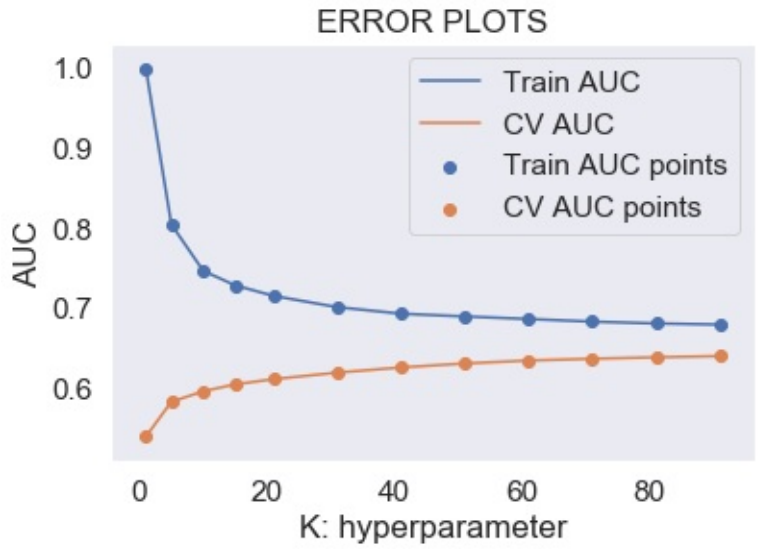
```
0648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse
 matrix and a dense matirx :)
X_train_avg_w2v = hstack((X_train_categories_one_hot, X_train
_sub_categories_one_hot, train_essay_avg_w2v, train_quantity_
standardized, X_train_state_one_hot, X_train_prefix_one_hot,
X_train_grade_one_hot, train_title_avg_w2v, train_price_stand
ardized, train_teacher_number_of_previously_posted_projects_s
tandardized, X_train_grade_one_hot,train_w2v_vectors_summary)
).tocsr()
X_cv_avg_w2v = hstack((X_cv_categories_one_hot, X_cv_sub_cate
gories_one_hot, cv_essay_avg_w2v, cv_quantity_standardized, X
_cv_state_one_hot, X_cv_prefix_one_hot, X_cv_grade_one_hot, c
v_title_avg_w2v, cv_price_standardized, cv_teacher_number_of_
previously_posted_projects_standardized, X_cv_grade_one_hot,c
v_w2v_vectors_summary)).tocsr()
X_test_avg_w2v = hstack((X_test_categories_one_hot, X_test_su
b_categories_one_hot, test_essay_avg_w2v, test_quantity_stand
ardized, X_test_state_one_hot, X_test_prefix_one_hot, X_test_
grade_one_hot, test_title_avg_w2v, test_price_standardized, t
est_teacher_number_of_previously_posted_projects_standardized
, X_test_grade_one_hot,test_w2v_vectors_summary)).tocsr()

print('Final matrix')
print(X_train_avg_w2v.shape, y_train.shape)
print(X_cv_avg_w2v.shape, y_cv.shape)
print(X_test_avg_w2v.shape, y_test.shape)
```

```
Final matrix
(22445, 756) (22445,)
(11055, 756) (11055,)
(16500, 756) (16500,)
```

Hyperparameter tuning using simple for loop

In [64]:

```
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 61, 71, 81, 91]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs = -1)
    neigh.fit(X_train_avg_w2v, 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_train_avg_w2v)
    y_cv_pred = batch_predict(neigh, X_cv_avg_w2v)

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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

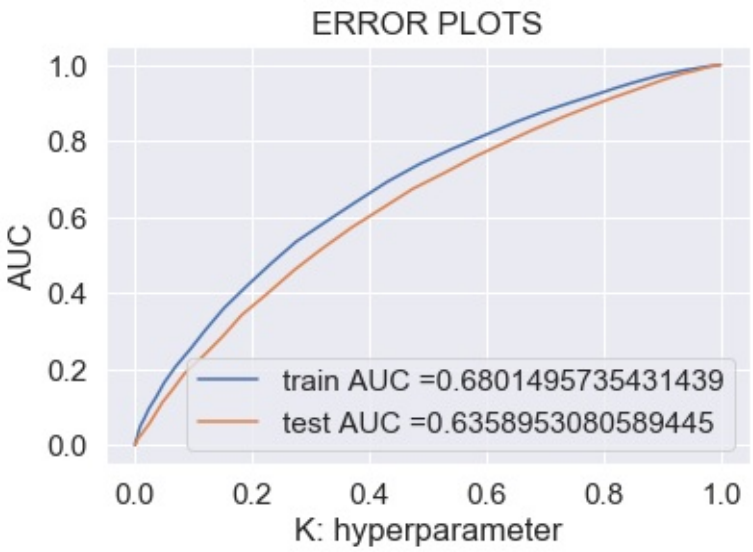


Plotting error plots

In [65]:

```
neigh = KNeighborsClassifier(n_neighbors = 81)
neigh.fit(X_train_avg_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be
probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, batch_p
redict(neigh, X_train_avg_w2v))
test_fpr, test_tpr, thresholds = roc_curve(y_test, batch_pred
ict(neigh, X_test_avg_w2v))

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(tr
ain_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.show()
```



Confusion matrix

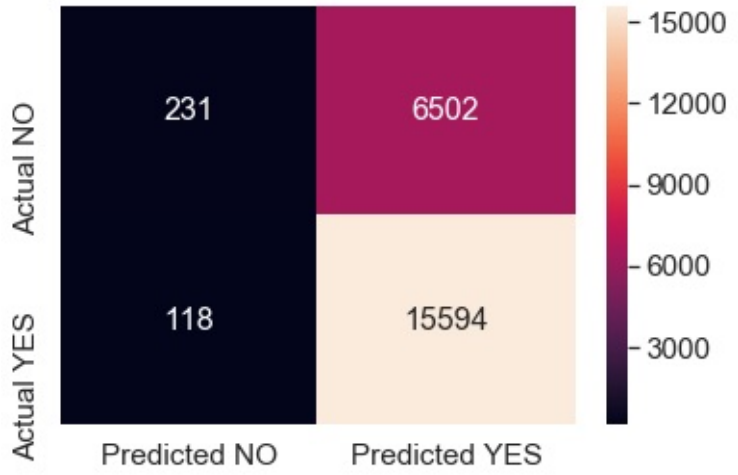
In [66]:

```
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_avg_w2v
)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_avg_w2v))
)
```

```
Train confusion matrix
[[  231  6502]
 [  118 15594]]
Test confusion matrix
[[  131  4819]
 [   96 11454]]
```

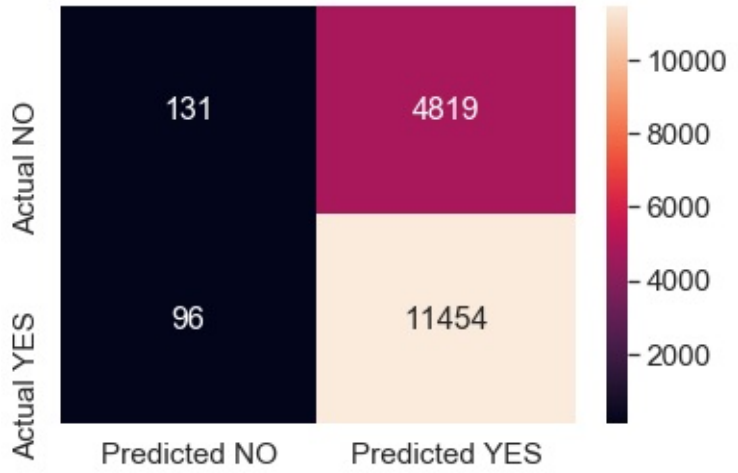
In [68]:

```
get_confusion_matrix(neigh,X_train_avg_w2v,y_train)
```



In [70]:

```
get_confusion_matrix(neigh,X_test_avg_w2v,y_test)
```



Evaluating Model Performance

In [71]:

```
y_pred_new = neigh.predict(X_test_avg_w2v)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test,
 y_pred_new)*100))
```

```
print("Precision on test set: %0.3f"%(precision_score(y_test,
 y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pre
d_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_
new)))
```

```
Accuracy on test set: 70.212%
Precision on test set: 0.704
Recall on test set: 0.992
F1-Score on test set: 0.823
```

KNN on Set 4: categorical, numerical features + Text(TFIDF W2V)

Text Vectorization:TFIDF W2V

Tfidf W2V on essay

In [72]:

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

In [73]:

```
train_essay_tfidf_w2v = []; # the avg-w2v for each sentence/r
eview is stored in this list
for sentence in tqdm(X_train['clean_essays']): # for each rev
iew/sentence
    vector = np.zeros(300) # as word vectors are of zero leng
th
    tf_idf_weight =0; # num of words with a valid vector in t
he sentence/review
    for word in sentence.split(): # for each word in a review
/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each w
```

```
ord
            # here we are multiplying idf value(dictionary[wo
rd]) and the tf value((sentence.count(word)/len(sentence.spli
t())))
            tf_idf = dictionary[word]*(sentence.count(word)/l
en(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weig
hted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_essay_tfidf_w2v.append(vector)

print(len(train_essay_tfidf_w2v))
print(len(train_essay_tfidf_w2v[0]))


cv_essay_tfidf_w2v = []; # the avg-w2v for each sentence/revi
ew is stored in this list
for sentence in tqdm(X_cv['clean_essays']): # for each review
/sentence
    vector = np.zeros(300) # as word vectors are of zero leng
th
    tf_idf_weight =0; # num of words with a valid vector in t
he sentence/review
    for word in sentence.split(): # for each word in a review
/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each w
ord
            # here we are multiplying idf value(dictionary[wo
rd]) and the tf value((sentence.count(word)/len(sentence.spli
t())))
            tf_idf = dictionary[word]*(sentence.count(word)/l
en(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weig
hted w2v
```

```
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_essay_tfidf_w2v.append(vector)

print(len(cv_essay_tfidf_w2v))
print(len(cv_essay_tfidf_w2v[0]))


test_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_essay_tfidf_w2v.append(vector)

print(len(test_essay_tfidf_w2v))
print(len(test_essay_tfidf_w2v[0]))
```

```
100%|██████████████████████████████████████████████████| 22445/22445 [01:38<00:00, 227.61it/s]

22445
300
```

```
100%|██████████████████████████████████████████████████| 11055/11055 [00:41<00:00, 264.32it/s]

11055
300
```

```
100%|██████████████████████████████████████████████████| 16500/16500 [01:01<00:00, 270.46it/s]

16500
300
```

In [74]:

```
# Changing list to numpy arrays
train_essay_tfidf_w2v = np.array(train_essay_tfidf_w2v)
cv_essay_tfidf_w2v = np.array(cv_essay_tfidf_w2v)
test_essay_tfidf_w2v = np.array(test_essay_tfidf_w2v)
```

TFIDF W2V on titles

In [75]:

```
train_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero leng
```

```
th
    tf_idf_weight =0; # num of words with a valid vector in t
he sentence/review
    for word in sentence.split(): # for each word in a review
/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each w
ord
            # here we are multiplying idf value(dictionary[wo
rd]) and the tf value((sentence.count(word)/len(sentence.spli
t())))
            tf_idf = dictionary[word]*(sentence.count(word)/l
en(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weig
hted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_title_tfidf_w2v.append(vector)

print(len(train_title_tfidf_w2v))
print(len(train_title_tfidf_w2v[0]))


cv_title_tfidf_w2v = []; # the avg-w2v for each sentence/revi
ew is stored in this list
for sentence in tqdm(X_cv['clean_titles']): # for each review
/sentence
    vector = np.zeros(300) # as word vectors are of zero leng
th
    tf_idf_weight =0; # num of words with a valid vector in t
he sentence/review
    for word in sentence.split(): # for each word in a review
/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each w
ord
```

```
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_title_tfidf_w2v.append(vector)

print(len(cv_title_tfidf_w2v))
print(len(cv_title_tfidf_w2v[0]))



test_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
```

```
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_title_tfidf_w2v.append(vector)

print(len(test_title_tfidf_w2v))
print(len(test_title_tfidf_w2v[0]))
```

```
100%|██████████████████████████████████████
████████████████████████████████| 22445/22445
[00:01<00:00, 11248.46it/s]
```

```
22445
300
```

```
100%|██████████████████████████████████████
████████████████████████████████| 11055/11055
[00:00<00:00, 16673.60it/s]
```

```
11055
300
```

```
100%|██████████████████████████████████████
████████████████████████████████| 16500/16500
[00:01<00:00, 16144.26it/s]
```

```
16500
300
```

In [76]:

```
# Changing list to numpy arrays
train_title_tfidf_w2v = np.array(train_title_tfidf_w2v)
cv_title_tfidf_w2v = np.array(cv_title_tfidf_w2v)
test_title_tfidf_w2v = np.array(test_title_tfidf_w2v)
```

TFIDF on Project resource summary

In [77]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['project_resource_summary'].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(t
fidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())


# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_summary = []; # the avg-w2v for each sentence
/review is stored in this list
for sentence in tqdm(X_train['project_resource_summary'].valu
es): # for each review/sentence
    vector = np.zeros(50) # as word vectors are of zero lengt
h
    tf_idf_weight =0; # num of words with a valid vector in t
he 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 e
ach word
            # here we are multiplying idf value(dictionary[wo
rd]) and the tf value((sentence.count(word)/len(sentence.spli
t())))
            tf_idf = dictionary[word]*(sentence.count(word)/l
en(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weig
hted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_tfidf_w2v_summary.append(vector)
```

```
print("Train matrix:")
print(len(train_tfidf_w2v_summary))
print(len(train_tfidf_w2v_summary[0]))
print('='*50)

cv_tfidf_w2v_summary = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_resource_summary'].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.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_tfidf_w2v_summary.append(vector)

print("CV matrix:")
print(len(cv_tfidf_w2v_summary))
print(len(cv_tfidf_w2v_summary[0]))
print('='*50)

test_tfidf_w2v_summary = []; # the avg-w2v for each sentence/
```

```
review is stored in this list
for sentence in tqdm(X_test['project_resource_summary'].value
s): # for each review/sentence
    vector = np.zeros(50) # as word vectors are of zero lengt
h
    tf_idf_weight =0; # num of words with a valid vector in t
he 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 e
ach word
            # here we are multiplying idf value(dictionary[wo
rd]) and the tf value((sentence.count(word)/len(sentence.spli
t())))
            tf_idf = dictionary[word]*(sentence.count(word)/l
en(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weig
hted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_summary.append(vector)

print(len(test_tfidf_w2v_summary))
print(len(test_tfidf_w2v_summary[0]))
```

```
100%|██████████████████████████████████████████████████
██████████████████████████████████| 22445/22445
 [00:04<00:00, 4514.37it/s]

Train matrix:
22445
50
==========================================
====

100%|██████████████████████████████████████████
```

```
100%|██████████████████████████████████| 11055/11055
 [00:02<00:00, 4087.92it/s]
```

```
CV matrix:
11055
50
================================================
====
```

```
100%|██████████████████████████████████████████
██████████████████████████████| 16500/16500
 [00:03<00:00, 4348.52it/s]
```

```
16500
50
```

In [78]:

```
# Changing list to numpy arrays
train_tfidf_w2v_summary = np.array(train_tfidf_w2v_summary)
test_tfidf_w2v_summary = np.array(test_tfidf_w2v_summary)
cv_tfidf_w2v_summary = np.array(cv_tfidf_w2v_summary)
```

Concatinating all the features

In [79]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_tfidf_w2v = hstack((X_train_categories_one_hot, X_train_sub_categories_one_hot, train_essay_tfidf_w2v, train_quantity_standardized, X_train_state_one_hot, X_train_prefix_one_hot, X_train_grade_one_hot, train_title_tfidf_w2v, train_price_standardized, train_teacher_number_of_previously_posted_proj
```

```
ects_standardized, X_train_grade_one_hot,train_tfidf_w2v_summ
ary)).tocsr()
X_cv_tfidf_w2v = hstack((X_cv_categories_one_hot, X_cv_sub_ca
tegories_one_hot, cv_essay_tfidf_w2v, cv_quantity_standardize
d, X_cv_state_one_hot, X_cv_prefix_one_hot, X_cv_grade_one_ho
t, cv_title_avg_w2v, cv_price_standardized, cv_teacher_number
_of_previously_posted_projects_standardized, X_cv_grade_one_h
ot,cv_tfidf_w2v_summary)).tocsr()
X_test_tfidf_w2v = hstack((X_test_categories_one_hot, X_test_
sub_categories_one_hot, test_essay_tfidf_w2v, test_quantity_s
tandardized, X_test_state_one_hot, X_test_prefix_one_hot, X_t
est_grade_one_hot, test_title_tfidf_w2v, test_price_standardi
zed, test_teacher_number_of_previously_posted_projects_standa
rdized, X_test_grade_one_hot,test_tfidf_w2v_summary)).tocsr()

print('Final matrix')
print(X_train_tfidf_w2v.shape, y_train.shape)
print(X_cv_tfidf_w2v.shape, y_cv.shape)
print(X_test_tfidf_w2v.shape, y_test.shape)
```

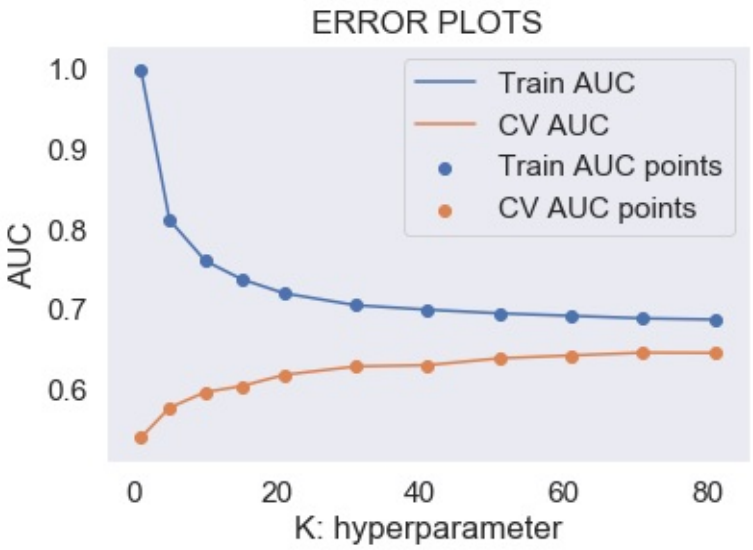
```
Final matrix
(22445, 756) (22445,)
(11055, 756) (11055,)
(16500, 756) (16500,)
```

Hyper parameter tuning using simple for loop

In [80]:

```
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 61, 71, 81]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs = -1)
    neigh.fit(X_train_tfidf_w2v, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should
```

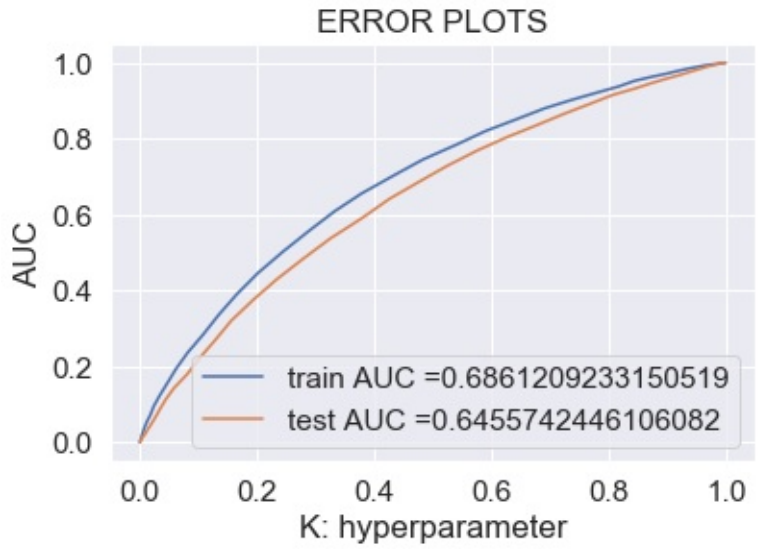
```
 be probability estimates of the positive class
    # not the predicted outputs
    y_train_pred = batch_predict(neigh, X_train_tfidf_w2v)

    y_cv_pred = batch_predict(neigh, X_cv_tfidf_w2v)

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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



Plotting error plot

In [81]:

```
neigh = KNeighborsClassifier(n_neighbors = 81)
neigh.fit(X_train_tfidf_w2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be
probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, batch_p
redict(neigh, X_train_tfidf_w2v))
test_fpr, test_tpr, thresholds = roc_curve(y_test, batch_pred
ict(neigh, X_test_tfidf_w2v))

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(tr
ain_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.show()
```



Confusion matrix

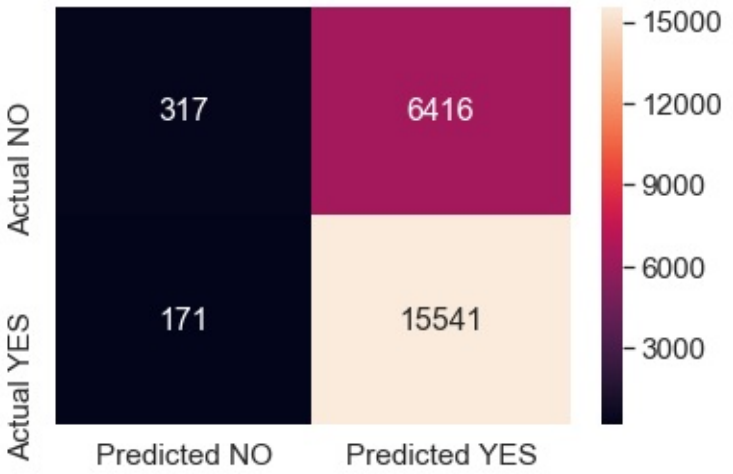
In [82]:

```
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_tfidf_w
2v)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tfidf_w2v
)))
```

```
Train confusion matrix
[[  317  6416]
 [  171 15541]]
Test confusion matrix
[[  186  4764]
 [  160 11390]]
```

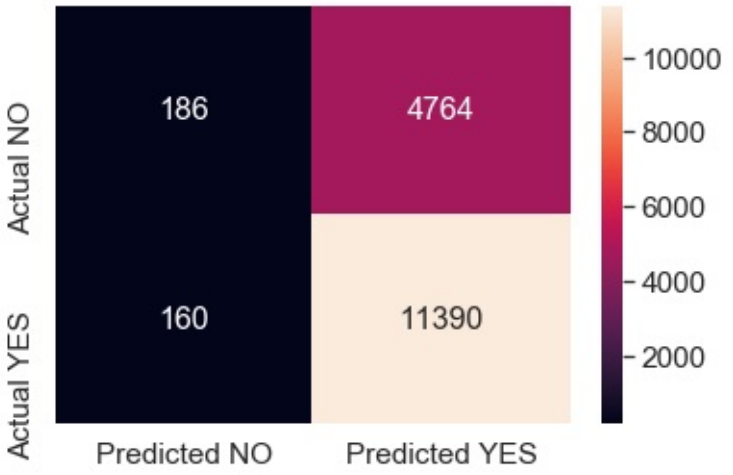
In [85]:

```
get_confusion_matrix(neigh,X_train_tfidf_w2v,y_train)
```



In [87]:

```
get_confusion_matrix(neigh,X_test_tfidf_w2v,y_test)
```



Evaluating Model Performance

In [88]:

```
y_pred_new = neigh.predict(X_test_tfidf_w2v)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test,
 y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test,
```

```
 y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pre
d_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_
new)))
```

```
Accuracy on test set: 70.158%
Precision on test set: 0.705
Recall on test set: 0.986
F1-Score on test set: 0.822
```

Set 5: categorical, numerical features + Text(TFIDF) with top 2K best features

Selecting top best 2000 features

In [89]:

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

vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['clean_essays'].values)

X_train_essay_tfidf = vectorizer.transform(X_train['clean_ess
ays'])
X_cv_essay_tfidf = vectorizer.transform(X_cv['clean_essays'])
X_test_essay_tfidf = vectorizer.transform(X_test['clean_essay
s'])

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

select = SelectKBest(chi2, k = 2000)
select.fit(X_train_essay_tfidf, y_train)

X_train_essay_tfidf_2000 = select.transform(X_train_essay_tfi
df)
X_cv_essay_tfidf_2000 = select.transform(X_cv_essay_tfidf)
X_test_essay_tfidf_2000 = select.transform(X_test_essay_tfidf
```

```
)

print("After selecting top 2000 features")
print(X_train_essay_tfidf_2000.shape, y_train.shape)
print(X_cv_essay_tfidf_2000.shape, y_cv.shape)
print(X_test_essay_tfidf_2000.shape, y_test.shape)
```

```
After vectorizing
(22445, 8813) (22445,)
(11055, 8813) (11055,)
(16500, 8813) (16500,)
================================================
====
After selecting top 2000 features
(22445, 2000) (22445,)
(11055, 2000) (11055,)
(16500, 2000) (16500,)
```

In [90]:

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

vectorizer = TfidfVectorizer(min_df=2)
vectorizer.fit(X_train['clean_titles'].values)

X_train_title_tfidf = vectorizer.transform(X_train['clean_tit
les'])
X_cv_title_tfidf = vectorizer.transform(X_cv['clean_titles'])
X_test_title_tfidf = vectorizer.transform(X_test['clean_title
s'])

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

```
select = SelectKBest(chi2, k = 2000)
select.fit(X_train_title_tfidf, y_train)

X_train_title_tfidf_2000 = select.transform(X_train_title_tfi
df)
X_cv_title_tfidf_2000 = select.transform(X_cv_title_tfidf)
X_test_title_tfidf_2000 = select.transform(X_test_title_tfidf
)

print("After selecting top 2000 features")
print(X_train_title_tfidf_2000.shape, y_train.shape)
print(X_cv_title_tfidf_2000.shape, y_cv.shape)
print(X_test_title_tfidf_2000.shape, y_test.shape)
```

```
After vectorizing
(22445, 4068) (22445,)
(11055, 4068) (11055,)
(16500, 4068) (16500,)
================================================
====
After selecting top 2000 features
(22445, 2000) (22445,)
(11055, 2000) (11055,)
(16500, 2000) (16500,)
```

Concatinating all the features

In [91]:

```
# merge two sparse matrices: https://stackoverflow.com/a/1971
0648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse
 matrix and a dense matirx :)
X_train_tfidf_2000 = hstack((X_train_categories_one_hot, X_tr
ain_sub_categories_one_hot, X_train_essay_tfidf_2000, train_q
```

```
uantity_standardized, X_train_state_one_hot, X_train_prefix_o
ne_hot, X_train_grade_one_hot, X_train_title_tfidf_2000, trai
n_price_standardized, train_teacher_number_of_previously_post
ed_projects_standardized, X_train_grade_one_hot)).tocsr()
X_cv_tfidf_2000 = hstack((X_cv_categories_one_hot, X_cv_sub_c
ategories_one_hot, X_cv_essay_tfidf_2000, cv_quantity_standar
dized, X_cv_state_one_hot, X_cv_prefix_one_hot, X_cv_grade_on
e_hot, X_cv_title_tfidf_2000, cv_price_standardized, cv_teach
er_number_of_previously_posted_projects_standardized, X_cv_gr
ade_one_hot)).tocsr()
X_test_tfidf_2000 = hstack((X_test_categories_one_hot, X_test
_sub_categories_one_hot, X_test_essay_tfidf_2000, test_quanti
ty_standardized, X_test_state_one_hot, X_test_prefix_one_hot,
 X_test_grade_one_hot, X_test_title_tfidf_2000, test_price_st
andardized, test_teacher_number_of_previously_posted_projects
_standardized, X_test_grade_one_hot)).tocsr()

print('Final matrix')
print(X_train_tfidf_2000.shape, y_train.shape)
print(X_cv_tfidf_2000.shape, y_cv.shape)
print(X_test_tfidf_2000.shape, y_test.shape)
```

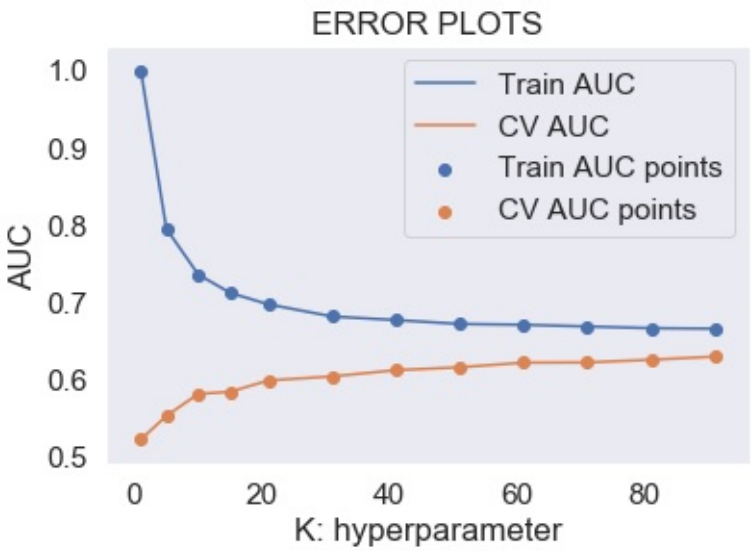
```
Final matrix
(22445, 4106) (22445,)
(11055, 4106) (11055,)
(16500, 4106) (16500,)
```

Hyperparameter tuning using For Loop

In [92]:

```
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 61, 71, 81, 91]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs = -1)
```

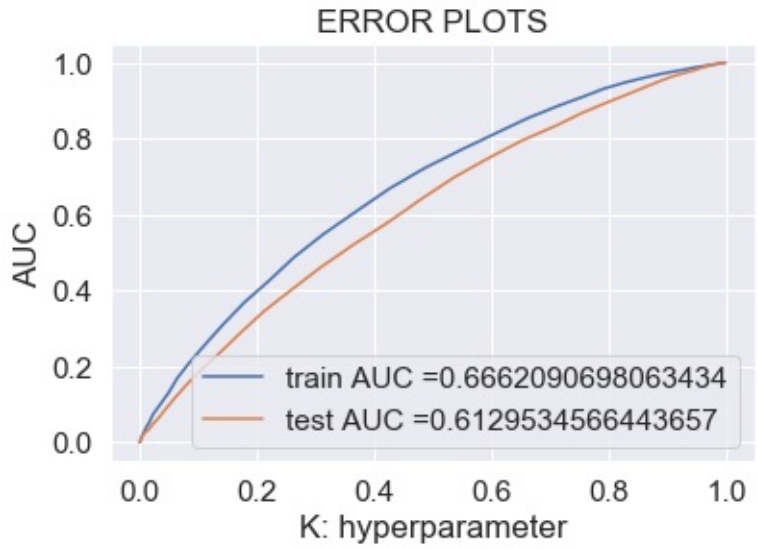
```
    neigh.fit(X_train_tfidf_2000, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should
 be probability estimates of the positive class
    # not the predicted outputs
    y_train_pred = batch_predict(neigh, X_train_tfidf_2000)

    y_cv_pred = batch_predict(neigh, X_cv_tfidf_2000)

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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



Plotting error plot

In [93]:

```
neigh = KNeighborsClassifier(n_neighbors = 81)
neigh.fit(X_train_tfidf_2000, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be
probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, batch_p
redict(neigh, X_train_tfidf_2000))
test_fpr, test_tpr, thresholds = roc_curve(y_test, batch_pred
ict(neigh, X_test_tfidf_2000))

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(tr
ain_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.show()
```



Confusion matrix

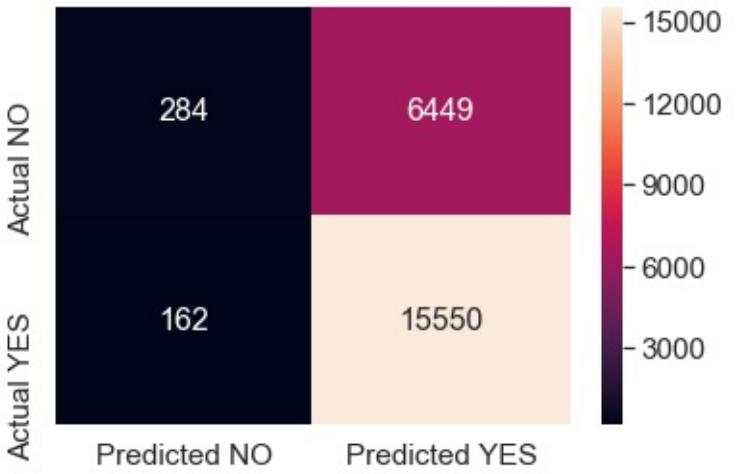
In [94]:

```
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_tfidf_2
000)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tfidf_200
0)))
```

```
Train confusion matrix
[[  284  6449]
 [  162 15550]]
Test confusion matrix
[[  195  4755]
 [  148 11402]]
```

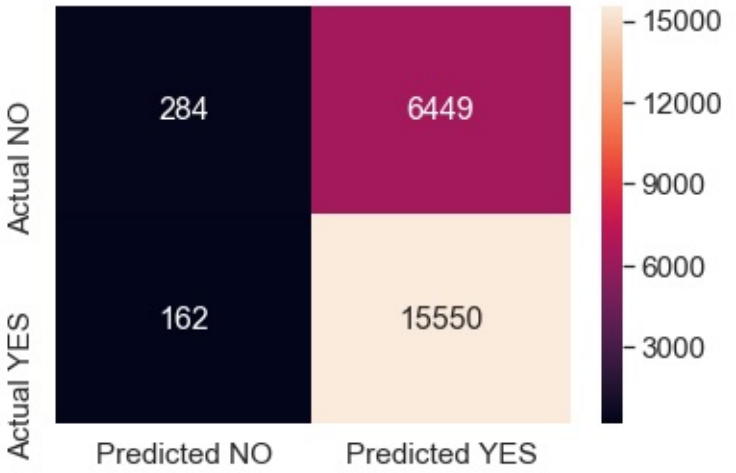
In [95]:

```
get_confusion_matrix(neigh,X_train_tfidf_2000,y_train)
```



In [96]:

```
get_confusion_matrix(neigh,X_train_tfidf_2000,y_train)
```



Evaluating Model performance

In [97]:

```
y_pred_new = neigh.predict(X_test_tfidf_2000)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test,
 y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test,
```

```
 y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pre
d_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_
new)))
```

```
Accuracy on test set: 70.285%
Precision on test set: 0.706
Recall on test set: 0.987
F1-Score on test set: 0.823
```

Conclusion:

In [101]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyperparameter", "Train AUC","Test AUC","F1 Score","Accuracy on test set"]

x.add_row(["BoW", "Brute",81, 0.69, 0.64,0.82,"70.1%"])
x.add_row(["TFIDF", "Brute",81, 0.67, 0.62,0.82,"70.3%"])
x.add_row(["Avg W2V", "Brute",81,0.68, 0.63,0.82,"70.2%"])
x.add_row(["TFIDF W2V", "Brute",71, 0.68, 0.64,0.82,"70.1%"])
x.add_row(["TFIDF (2000 features)","Brute",71, 0.66, 0.61,0.82,"70.28%"])


print(x)
```

```
+----------------------+-------+-------------
---+-----------+----------+----------+--------
--------------+
|       Vectorizer      | Model | Hyperparamet
er | Train AUC | Test AUC | F1 Score | Accurac
y on test set |
+----------------------+-------+-------------
---+-----------+----------+----------+--------
--------------+
|          BoW          | Brute |       81
   |    0.69   |   0.64   |   0.82   |
70.1%         |
|         TFIDF         | Brute |       81
```

```
   |    0.67   |   0.62   |   0.82   |
70.3%         |
|        Avg W2V        | Brute |       81
   |    0.68   |   0.63   |   0.82   |
70.2%         |
|       TFIDF W2V       | Brute |       71
   |    0.68   |   0.64   |   0.82   |
70.1%         |
| TFIDF (2000 features) | Brute |       71
   |    0.66   |   0.61   |   0.82   |
70.28%        |
+-----------------------+-------+-------------
---+-----------+----------+----------+--------
--------------+
```

1. BoW, Avg w2v and Tfidf have perfomed well on the basis of Train AUC score.
2. Accuracy on taking 5000 features and 2000 doesnot have any significant difference.
3. We haven't got any model with more than 69% accuracy while considering 50K points.