DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located ([Two-letter U.S. postal code](https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*

Feature	Description
`project_essay_4`	Fourth application essay
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [5]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os

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

In [ ]:

1.1 Reading Data

In [6]:

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

In [7]:

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

```
Number of data points in train data (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 [8]:

```
# 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[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	

In [9]:

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

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

Out[9]:

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

1.2 preprocessing of `project_subject_categories`

In [10]:

```
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 & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & 
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # 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]))
```

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

1.3 preprocessing of `project_subject_subcategories`

In [11]:

```
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 & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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]))
```

1.3 Text preprocessing

In [12]:

```
# 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 [13]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**
    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
    - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
    - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

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

- Find the best hyper parameter which results in the maximum AUC value
- Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
- Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task

3. **Representation of results**

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

4. **[Task-2]**

   - Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features
     - ```
       from sklearn.datasets import load_digits
       from sklearn.feature_selection import SelectKBest, chi2
       X, y = load_digits(return_X_y=True)
       X.shape
       X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
       X_new.shape
       ========
       output:
       (1797, 64)
       (1797, 20)
       ```
   - Repeat the steps 2 and 3 on the data matrix after feature selection

5. **Conclusion**

   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

2. K Nearest Neighbor

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

Sampling of data

I took a sample of 15000 data points and stored it in sample_data

In [14]:

```
sample_data=project_data.sample(50000)
```

In [15]:

```
sample_data.shape
```

Out[15]:

(50000, 20)
```

```
(50000, 20)
```

Splitting the data

The sample data is splitted into 3 parts , train data, test data, and crossvalidation data.

In [16]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
from sklearn.model_selection import train_test_split
data=sample_data
data.head()
y=data.project_is_approved
x=data.drop('project_is_approved',axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.8,test_size=0.2,stratify=y)
x_train,x_cv,y_train,y_cv=train_test_split(x_train,y_train,train_size=0.8,test_size=0.2,stratify=y
_train)


print("shape of train data ")
print(x_train.shape)
print(y_train.shape)
print("shape of test data ")
print(x_test.shape)
print(y_test.shape)
print("shape of crossvalidation data ")
print(x_cv.shape)
print(y_cv.shape)
```

```
shape of train data
(32000, 19)
(32000,)
shape of test data
(10000, 19)
(10000,)
shape of crossvalidation data
(8000, 19)
(8000,)
```

In [17]:

```
train=pd.concat([x_train,y_train],axis=1)
train.shape
```

Out[17]:

```
(32000, 20)
```

In [18]:

```
count_class_1, count_class_0 = train.project_is_approved.value_counts()

# Divide by class
class_0 = train[train['project_is_approved'] == 0]# Counting the positive and negative data
class_1 = train[train['project_is_approved'] == 1]
print(count_class_0)
print(count_class_1)
# As the negative data is higher in the dataset, so it unbalanced and I am balancing the data by r
andom oversampling .
class_0_over = class_0.sample(count_class_1, replace=True)
train_over = pd.concat([class_1, class_0_over], axis=0)

print('Random over-sampling:')
print(train_over.project_is_approved.value_counts())
```

```
4844
27156
Random over-sampling:
1    27156
0    27156
Name: project_is_approved, dtype: int64
```

In [19]:

```
y_sample = train_over.project_is_approved
x_sample = train_over.drop('project_is_approved',axis=1)
print(x_sample.shape)
print(y_sample.shape)
```

```
(54312, 19)
(54312,)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [20]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

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

x_sample.columns
```

Out[20]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'essay', 'price', 'quantity'],
      dtype='object')
```

Vectorizing numerical and categorical data

In [32]:

```
def veccat(x):
    from sklearn.feature_extraction.text import CountVectorizer
    from sklearn.preprocessing import StandardScaler
    from scipy.sparse import hstack
    vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=T
rue)
    vectorizer.fit(x['clean_categories'].values)

    print(vectorizer.get_feature_names())


    categories_one_hot = vectorizer.transform(x['clean_categories'].values)
    print("Shape of matrix after one hot encodig ",categories_one_hot.shape)

    vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False,
binary=True)
    vectorizer.fit(x['clean_subcategories'].values)
    print(vectorizer.get_feature_names())
```

```
    subcategories_one_hot = vectorizer.transform(x['clean_subcategories'].values)
    print("Shape of matrix after one hot encodig ",subcategories_one_hot.shape)


    vectorizer = CountVectorizer(lowercase=False, binary=True)
    vectorizer.fit(x['school_state'].values)
    print(vectorizer.get_feature_names())


    state_one_hot = vectorizer.transform(x['school_state'].values)
    print("Shape of matrix after one hot encodig ",state_one_hot.shape)

    x = x.replace(np.nan, '', regex=True)
    vectorizer = CountVectorizer( vocabulary=['Mrs.', 'Ms.','Mr.','Teacher','Dr.'],lowercase=False)
    vectorizer.fit(x['teacher_prefix'].values)
    print(vectorizer.get_feature_names())


    prefix_one_hot = vectorizer.transform(x['teacher_prefix'].values)
    print("Shape of matrix after one hot encodig ",prefix_one_hot.shape)

    vocab=[]
    for i in x['project_grade_category'].values:
        vocab.append(i)
    v_set=set(vocab)
    vocab=list(v_set)
    vectorizer = CountVectorizer(vocabulary=vocab,lowercase=False)
    vectorizer.fit(x['project_grade_category'].values)
    print(vectorizer.get_feature_names())


    grade_one_hot = vectorizer.transform(x['project_grade_category'].values)
    print("Shape of matrix after one hot encodig ",grade_one_hot.shape)

    price_scalar = StandardScaler()
    price_scalar.fit(x['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.
    price_standardized = price_scalar.transform(x['price'].values.reshape(-1, 1))

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

    # Now standardize the data with above maen and variance.
    projects_standardized =
projects_scalar.transform(x['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))


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

    # Now standardize the data with above maen and variance.
    qty_standardized = qty_scalar.transform(x['quantity'].values.reshape(-1, 1))

    X1 = hstack((state_one_hot,categories_one_hot, subcategories_one_hot,prefix_one_hot, price_stan
dardized,projects_standardized,qty_standardized))
    print(X1.shape)
    return(X1)
```

In [ ]:

In [33]:

```
x=veccat(x_sample)
t=veccat(x_test)
cv=veccat(x_cv)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (54312, 9)
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (54312, 30)
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'K
S', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV
', 'WY']
Shape of matrix after one hot encodig  (54312, 51)
['Mrs.', 'Ms.', 'Mr.', 'Teacher', 'Dr.']
Shape of matrix after one hot encodig  (54312, 5)
['Grades PreK-2', 'Grades 6-8', 'Grades 3-5', 'Grades 9-12']
Shape of matrix after one hot encodig  (54312, 4)
Mean : 322.6913503461482, Standard deviation : 373.1902961503055
Mean : 9.257309618500516, Standard deviation : 24.062630440535916
Mean : 18.71137133598468, Standard deviation : 27.3612534710272
(54312, 98)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (10000, 9)
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (10000, 30)
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'K
S', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV
', 'WY']
Shape of matrix after one hot encodig  (10000, 51)
['Mrs.', 'Ms.', 'Mr.', 'Teacher', 'Dr.']
Shape of matrix after one hot encodig  (10000, 5)
['Grades 6-8', 'Grades 9-12', 'Grades PreK-2', 'Grades 3-5']
Shape of matrix after one hot encodig  (10000, 4)
Mean : 299.132584, Standard deviation : 369.84268496938387
Mean : 11.2904, Standard deviation : 28.2100419680652
Mean : 16.2243, Standard deviation : 25.06227422860902
(10000, 98)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (8000, 9)
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (8000, 30)
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'K
S', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV
', 'WY']
Shape of matrix after one hot encodig  (8000, 51)
['Mrs.', 'Ms.', 'Mr.', 'Teacher', 'Dr.']
Shape of matrix after one hot encodig  (8000, 5)
['Grades 3-5', 'Grades 6-8', 'Grades PreK-2', 'Grades 9-12']
Shape of matrix after one hot encodig  (8000, 4)
Mean : 306.6500025, Standard deviation : 390.7150112636767
Mean : 11.03425, Standard deviation : 27.207670369539176
Mean : 16.64875, Standard deviation : 25.28692099559573
(8000, 98)
```

In [34]:

```
print(x.todense())
```

```
[[ 0.          0.          0.         ... -0.7768191  -0.09380976
   0.41257718]
 [ 0.          0.          0.         ...  0.84506123 -0.34315906
  -0.42802759]
 [ 0.          0.          0.         ...  0.99541348 -0.09380976
  -0.42802759]
 ...
 [ 0.          0.          0.         ...  0.08828914 -0.38471728
  -0.02599922]
 [ 0.          0.          0.         ...  0.07164883 -0.38471728
  -0.31838349]
 [ 0.          0.          1.         ...  0.20573592 -0.26004263
  -0.53767169]]
```

2.3 Make Data Model Ready: encoding eassay, and project_title

Vectorizing essay and project_title

In [19]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

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

import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [20]:

```
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
```

```
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [21]:

```
## Preprocessing is done
def preprocessing(x):
    import nltk
    nltk.download('stopwords')
    from tqdm import tqdm
    preprocessed_essays = []
    # tqdm is for printing the status bar
    for sentence in tqdm(x.values):
        sent = decontracted(sentence)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\"', ' ')
        sent = sent.replace('\\n', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        sent = re.sub(r"[-()\"#/@;:<>{}`+=~|.!?,]", "", sent)
        # https://gist.github.com/sebleier/554280
        sent=' '.join(e.lower() for e in sent.split() if e.lower() not in stopwords)
        preprocessed_essays.append(sent.strip())
    return preprocessed_essays
```

In [22]:

```
train_essay=[]
test_essay=[]
cv_essay=[]
train_title=[]
test_title=[]
cv_title=[]
train_essay=preprocessing(x_sample['essay'])
test_essay=preprocessing(x_test['essay'])
cv_essay=preprocessing(x_cv['essay'])

train_title=preprocessing(x_sample['project_title'])
test_title=preprocessing(x_test['project_title'])
cv_title=preprocessing(x_cv['project_title'])
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|██████████████████████████████████| 54340/54340 [00:49<00:00, 1088.57it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|██████████████████████████████████| 10000/10000 [00:08<00:00, 1149.10it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|████████████████████████████████████| 8000/8000 [00:07<00:00, 1129.56it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|█████████████████████████████████| 54340/54340 [00:02<00:00, 21708.70it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|█████████████████████████████████| 10000/10000 [00:00<00:00, 24329.50it/s]
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
100%|███████████████████████████████████| 8000/8000 [00:00<00:00, 24314.72it/s]
```

Encoding using BOW

In [23]:

```
from scipy.sparse import hstack
def bow_text(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x1,t1,cv1):
    from scipy.sparse import hstack
    vectorizer = CountVectorizer()
    vectorizer.fit(train_essay)
    text_bow=vectorizer.transform(train_essay)
    text_bow1 = vectorizer.transform(test_essay)
    text_bow2 = vectorizer.transform(cv_essay)
    vectorizer.fit(train_title)
    title_bow=vectorizer.transform(train_title)
    title_bow1 = vectorizer.transform(test_title)
    title_bow2 = vectorizer.transform(cv_title)
    x1 = hstack((x1,text_bow,title_bow )).tocsr()
    t1= hstack((t1,text_bow1,title_bow1 )).tocsr()
    cv1 = hstack((cv1,text_bow2,title_bow2 )).tocsr()



    return x1,t1,cv1
```

In [24]:

```
# Data matrix using BOW
x1,t1,cv1=bow_text(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x,t,cv)
```

In [25]:

```
print(x1.shape)
print(t1.shape)
print(cv1.shape)
```

```
(54340, 44344)
(10000, 44344)
(8000, 44344)
```

Encoding using TFIDF

In [26]:

```
def tfidf_text(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x1,t1,cv1):
    from sklearn.feature_extraction.text import TfidfVectorizer
    vectorizer = TfidfVectorizer()
    vectorizer.fit(train_essay)
    text_tfidf=vectorizer.transform(train_essay)
    text_tfidf1 = vectorizer.transform(test_essay)
    text_tfidf2 = vectorizer.transform(cv_essay)
    vectorizer.fit(train_title)
    title_tfidf=vectorizer.transform(train_title)
    title_tfidf1 = vectorizer.transform(test_title)
    title_tfidf2 = vectorizer.transform(cv_title)
    x1 = hstack((x1,text_tfidf,title_tfidf )).tocsr()
    t1= hstack((t1,text_tfidf1,title_tfidf1 )).tocsr()
    cv1 = hstack((cv1,text_tfidf2,title_tfidf2 )).tocsr()



    return x1,t1,cv1
```

In [27]:

```
# Data matrix using TFIDF
x2,t2,cv2=tfidf_text(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x,t,cv)
print(x2.shape)
print(t2.shape)
print(cv2.shape)
```

```
(54340, 44344)
(10000, 44344)
(8000, 44344)
```

Encoding using AVG W2V

In [28]:

```
def avgword2vec(text):

    with open('glove_vectors', 'rb') as f:
        model = pickle.load(f)
        glove_words =  set(model.keys())
    avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(text):# for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        cnt_words =0; # num of words with a valid vector in the sentence/review
        for word in sentence.split(): # for each word in a review/sentence
            if word in glove_words:
                vector += model[word]
                cnt_words += 1
        if cnt_words != 0:
            vector /= cnt_words
        avg_w2v_vectors.append(vector)
    return avg_w2v_vectors
```

In [29]:

```
def w2v(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x,t,cv):

    text_w2v=avgword2vec(train_essay)
    text_w2v1 = avgword2vec(test_essay)
    text_w2v2 = avgword2vec(cv_essay)

    title_w2v=avgword2vec(train_title)
    title_w2v1 = avgword2vec(test_title)
    title_w2v2 = avgword2vec(cv_title)
    x1 = hstack((x,text_w2v,title_w2v )).tocsr()
    t1= hstack((t,text_w2v1,title_w2v1 )).tocsr()
    cv1 = hstack((cv,text_w2v2,title_w2v2 )).tocsr()
    return x1,t1,cv1
```

In [30]:

```
# Final Data matrix using Avg W2V
x3,t3,cv3=w2v(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x,t,cv)
print(x3.shape)
print(t3.shape)
print(cv3.shape)
```

```
100%|██████████████████████████████████| 54340/54340 [00:22<00:00, 2438.27it/s]
100%|██████████████████████████████████| 10000/10000 [00:03<00:00, 2518.75it/s]
100%|████████████████████████████████████| 8000/8000 [00:03<00:00, 2369.53it/s]
100%|█████████████████████████████████| 54340/54340 [00:01<00:00, 40309.27it/s]
100%|█████████████████████████████████| 10000/10000 [00:00<00:00, 42916.03it/s]
100%|███████████████████████████████████| 8000/8000 [00:00<00:00, 40401.71it/s]
```

```
(54340, 698)
(10000, 698)
(8000, 698)
```

Encoding using TFIDFW2V

In [31]:

```
def tfidfw2v(text,dictionary,tfidf_words):
```

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

    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(text): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence/review
        for word in sentence.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words):
                vec = model[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting
the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)
    return tfidf_w2v_vectors
```

In [32]:

```
def tfiw2v(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x,t,cv):

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

    text_tfw2v=tfidfw2v(train_essay,dictionary,tfidf_words)
    text_tfw2v1=tfidfw2v(test_essay,dictionary,tfidf_words)
    text_tfw2v2=tfidfw2v(cv_essay,dictionary,tfidf_words)

    tfidf_model.fit(train_title)
    # we are converting a dictionary with word as a key, and the idf as a value
    dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
    tfidf_words = set(tfidf_model.get_feature_names())
    title_tfw2v=tfidfw2v(train_title,dictionary,tfidf_words)
    title_tfw2v1=tfidfw2v(test_title,dictionary,tfidf_words)
    title_tfw2v2=tfidfw2v(cv_title,dictionary,tfidf_words)
    x1 = hstack((x,text_tfw2v,title_tfw2v )).tocsr()
    t1= hstack((t,text_tfw2v1,title_tfw2v1 )).tocsr()
    cv1 = hstack((cv,text_tfw2v2,title_tfw2v2 )).tocsr()
    return x1,t1,cv1
```

In [33]:

```
x4,t4,cv4=tfiw2v(train_essay, test_essay, cv_essay,train_title,test_title,cv_title,x,t,cv)
print(x4.shape)
print(t4.shape)
print(cv4.shape)
```

```
100%|████████████████████████████████████| 54340/54340 [02:48<00:00, 322.38it/s]
100%|████████████████████████████████████| 10000/10000 [00:32<00:00, 307.18it/s]
100%|██████████████████████████████████████| 8000/8000 [00:26<00:00, 297.04it/s]
100%|██████████████████████████████████| 54340/54340 [00:02<00:00, 19559.72it/s]
100%|██████████████████████████████████| 10000/10000 [00:00<00:00, 19010.32it/s]
100%|████████████████████████████████████| 8000/8000 [00:00<00:00, 20252.02it/s]
```

```
(54340, 698)
(10000, 698)
(8000, 698)
```

In [34]:

```
cv4.ndim
```

Out[34]:

2

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

Apply KNN on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instructions

In [35]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code

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

In [36]:

```
 # Function for finding the optimal K in BOW and TFIDF

def k_from_roc(x_train,y_train,x_cv,y_cv):
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.model_selection import cross_val_score
    from sklearn.metrics import roc_curve
    from sklearn.metrics import roc_auc_score
    k_range=[1,40,100,150,170,200,250,300]
    scores_auc=[]

    for k in k_range:
        knn=KNeighborsClassifier(n_neighbors=k,algorithm='brute')
        knn.fit(x_train,y_train)
        prob= knn.predict_proba(x_cv)
        prob=prob[:,1]
        scores_auc.append(roc_auc_score(y_cv,prob))
    # Chosen optimal k with the highest AUC score
    optimal_k = k_range[scores_auc.index(max(scores_auc))]

    model = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='brute')
    model.fit(x_train, y_train)
    # predict probabilities
    prob = model.predict_proba(x_cv)
    # keep probabilities for the positive outcome only
    prob = prob[:, 1]
    prob1=model.predict_proba(x_train)
    prob1=prob1[:, 1]
    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(y_cv, prob)
    fpr1, tpr1, thresholds1 = roc_curve(y_train, prob1)
    # plot no skill

    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr, tpr, marker='.',label="CrossValidated or test ROC" )
    plt.plot(fpr1, tpr1, marker='.' , label='Train ROC')
    # show the plot
    plt.title("Receiver Operating Characteristics")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    plt.show()
    print("Optimal K ",optimal_k)
    print("AUC: ", max(scores_auc))
    plt.plot(k_range, scores_auc, label='CV or TEST  AUC')
    plt.scatter(k_range, scores_auc, label='CV or TEST AUC points')
    plt.xlabel("K: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
```

```
    plt.title('ERROR PLOTS')
    plt.legend()

    plt.show()
```

In [37]:

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

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

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


    return y_data_pred
```

In [59]:

```
# Function for finding the optimal K in AVGW2V and TFIDFAVGW2V

def k_from_roc1(x_train,y_train,x_cv,y_cv):
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.model_selection import cross_val_score
    from sklearn.metrics import roc_curve
    from sklearn.metrics import roc_auc_score
    k_range=[1,100,250,300,350,400]
    scores_auc=[]

    for i in k_range:
        neigh = KNeighborsClassifier(n_neighbors=i)
        neigh.fit(x_train, y_train)
        prob=batch_predict(neigh,x_cv)
        scores_auc.append(roc_auc_score(y_cv,prob))
    # Chosen optimal k with the highest AUC score
    optimal_k = k_range[scores_auc.index(max(scores_auc))]
    model = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='brute')
    model.fit(x_train, y_train)
    # predict probabilities
    prob = batch_predict(model,x_cv)
    prob1=batch_predict(model,x_train)

    # keep probabilities for the positive outcome only
    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(y_cv, prob)
    fpr1, tpr1, thresholds1 = roc_curve(y_train, prob1)
    # plot no skill
    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr, tpr, marker='.',label="CrossValidated or test ROC")
    plt.plot(fpr1, tpr1, marker='.' , label='Train ROC')
    plt.legend()
    # show the plot
    plt.title("Receiver Operating Characteristics")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.show()
    print("Optimal K ",optimal_k)
    print("AUC: ", max(scores_auc))
    plt.plot(k_range, scores_auc, label='CV or Test AUC')
    plt.scatter(k_range, scores_auc, label='CV or Test AUC points')
    plt.xlabel("K: hyperparameter")
    plt.ylabel("AUC")
```

```
    plt.ylabel('AUC')
    plt.title("ERROR PLOTS")
    plt.legend()
    plt.show()
```

In [50]:

```
# Predictions  on test data in BOW and TFIDF

def predict(x_train,y_train,x_test,y_test,k):

    knn=KNeighborsClassifier(n_neighbors=k,algorithm='brute')
    knn.fit(x_train,y_train)
    prob1=knn.predict_proba(x_train)
    prob1=prob1[:,1]
    fpr, tpr, threshould = roc_curve(y_train, prob1)
    t = threshould[np.argmax(tpr*(1-fpr))]
    prob= knn.predict_proba(x_test)
    prob=prob[:,1]

    # (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))
    print("TRAIN AUC =",str(auc(fpr, tpr)))
    predictions = []
    for i in prob:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [49]:

```
# Predictions  on test data in AVGW2V and TFIDFAVGW2V

def predict1(x_train,y_train,x_test,y_test,k):

    knn=KNeighborsClassifier(n_neighbors=k,algorithm='brute')
    knn.fit(x_train,y_train)
    prob1=batch_predict(knn,x_train)

    fpr, tpr, threshould = roc_curve(y_train, prob1)
    t = threshould[np.argmax(tpr*(1-fpr))]

    prob= batch_predict(knn,x_test)


    # (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))
    print("TRAIN AUC =",str(auc(fpr, tpr)))
    predictions = []
    for i in prob:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [40]:

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

def grid_search(x_train,y_train,x_cv,y_cv):
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.model_selection import GridSearchCV
    from sklearn.model_selection import RandomizedSearchCV
    from sklearn.metrics import roc_curve
    from sklearn.metrics import roc_auc_score
    from sklearn import preprocessing
```

```
k_range=[1,4,70,100,150]
neigh=KNeighborsClassifier(n_neighbors=k_range,algorithm='brute')
parameters = dict(n_neighbors=k_range)
# Fitted the model on train data
clf = GridSearchCV(neigh, parameters, cv=3,verbose=5, scoring='roc_auc')
clf.fit(x_train, y_train)
train_auc= clf.cv_results_['mean_train_score']
cv_auc = clf.cv_results_['mean_test_score']
max_score= clf.best_score_

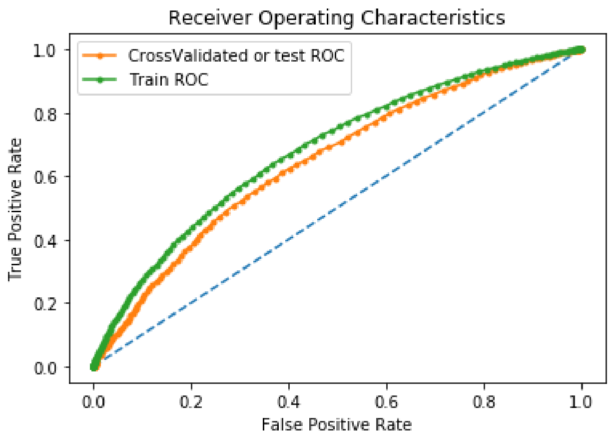
opt_par=clf.best_params_
print("Optimal K ",opt_par)
print("AUC: ", max_score)
plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
plt.plot(parameters['n_neighbors'], train_auc, label='TRAIN AUC')
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.legend()
plt.show()
# Found out the score for crossvalidated data
print("Accuracy on crossvalidated data: " , clf.score(x_cv,y_cv))
```

2.4.1 Applying KNN brute force on BOW, SET 1

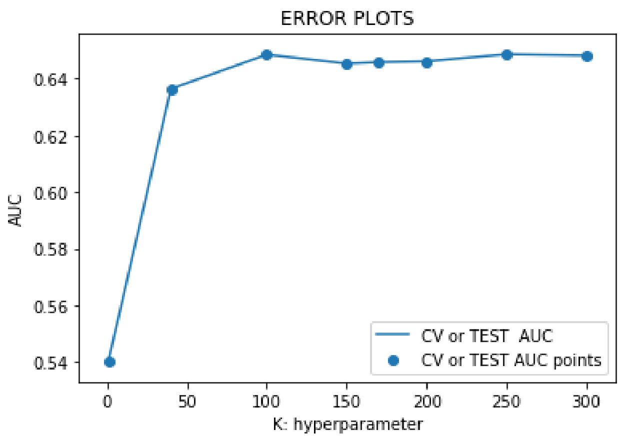
ROC AUC Score

In [41]:

```
k_from_roc(x1,y_sample,cv1,y_cv)
```




```
Optimal K  250
AUC:  0.6486733199571517
```



GRIDSEARCHCV

In [82]:

```
grid_search(x1,y_sample,cv1,y_cv)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV] n_neighbors=1 ...................................................
[CV] ......... n_neighbors=1, score=0.8692723859997792, total= 1.2min
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  3.4min remaining:    0.0s
```

```
[CV] n_neighbors=1 ...................................................
[CV] ......... n_neighbors=1, score=0.8734680357734348, total= 1.2min
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  6.7min remaining:    0.0s
```

```
[CV] n_neighbors=1 ...................................................
[CV] ......... n_neighbors=1, score=0.8674359540636042, total= 1.1min
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed: 10.2min remaining:    0.0s
```

```
[CV] n_neighbors=4 ...................................................
[CV] ......... n_neighbors=4, score=0.8229019931658766, total= 1.2min
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed: 13.8min remaining:    0.0s
```

```
[CV] n_neighbors=4 ...................................................
[CV] ......... n_neighbors=4, score=0.8361344561377125, total= 1.3min
[CV] n_neighbors=4 ...................................................
[CV] ......... n_neighbors=4, score=0.8216591402904113, total= 1.3min
[CV] n_neighbors=70 ..................................................
[CV] ......... n_neighbors=70, score=0.677969592285308, total= 1.2min
[CV] n_neighbors=70 ..................................................
[CV] ........ n_neighbors=70, score=0.6789853458246338, total= 1.3min
[CV] n_neighbors=70 ..................................................
[CV] ........ n_neighbors=70, score=0.6785523127005582, total= 1.2min
[CV] n_neighbors=100 .................................................
[CV] ........ n_neighbors=100, score=0.671690618549976, total= 1.3min
[CV] n_neighbors=100 .................................................
[CV] ....... n_neighbors=100, score=0.6731777100629068, total= 1.3min
[CV] n_neighbors=100 .................................................
[CV] ....... n_neighbors=100, score=0.6720967015991585, total= 1.2min
[CV] n_neighbors=150 .................................................
[CV] ....... n_neighbors=150, score=0.6669946167616383, total= 1.2min
[CV] n_neighbors=150 .................................................
[CV] ...... n_neighbors=150, score=0.6718591072313731, total=487.1min
[CV] n_neighbors=150 .................................................
[CV] ........ n_neighbors=150, score=0.667479176242602, total= 1.1min
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed: 539.7min finished
```

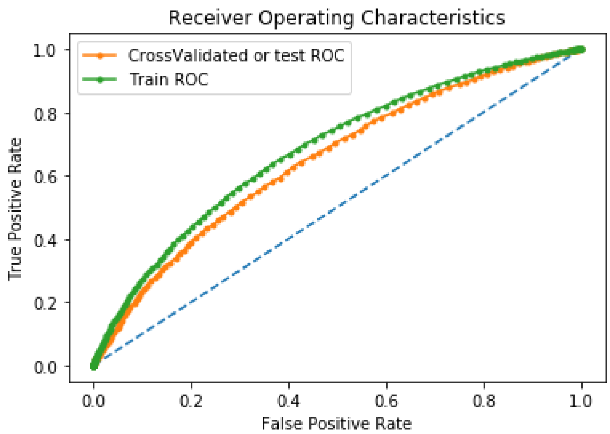
```
Optimal K  {'n_neighbors': 1}
AUC:  0.8700588884799411
```


```
Accuracy on crossvalidated data:  0.5401187269267269
```

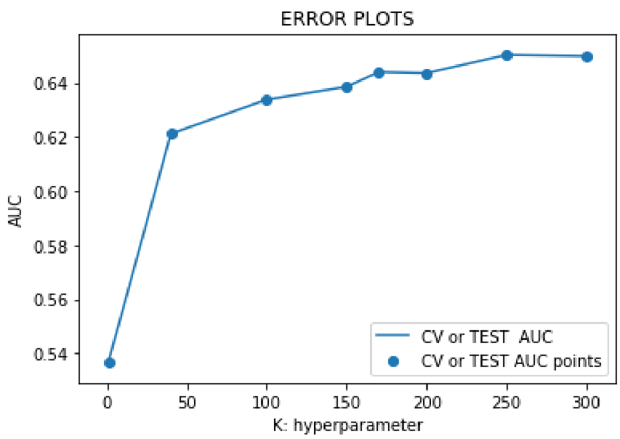
ROC Score of Test Data

In [42]:

```
k_from_roc(x1,y_sample,t1,y_test)
```




```
Optimal K  250
AUC:  0.650349814901663
```



CONFUSION MATRIX FOR TRAIN AND TEST DATA

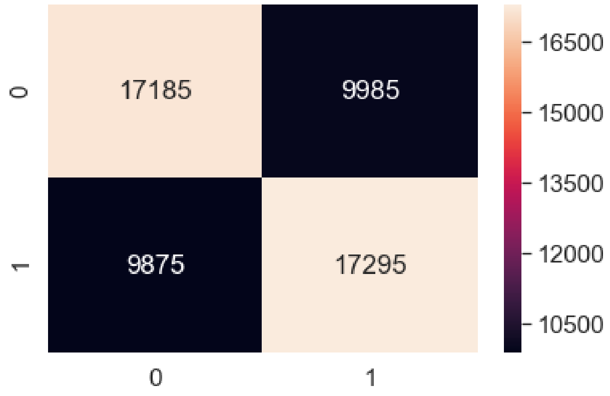
In [51]:

```
# AUC score is higher for k =250 on test data . So I use that to calculate the confusion matrix on
train data.
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_sample, predict(x1, y_sample, x1, y_sample,250),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4026158110529418 for threshold 0.4
TRAIN AUC = 0.6825105820317258
```

Out[51]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cbd52b0>
```



In [52]:

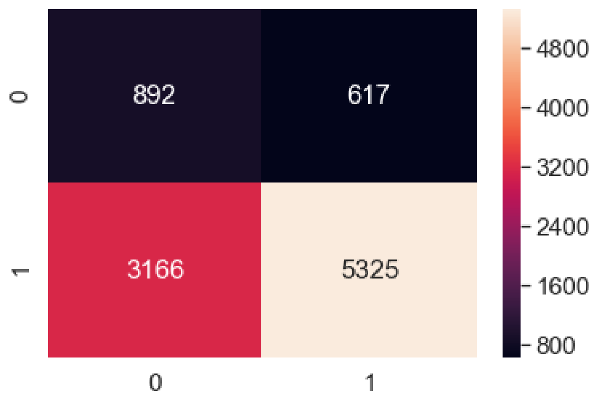
```
# AUC score is higher for k =250 on test data . So I use that to calculate the confusion matrix on test data.


from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict(x1, y_sample, t1, y_test,250),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4026158110529418 for threshold 0.4
TRAIN AUC = 0.6825105820317258
```

Out[52]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x310df588>
```

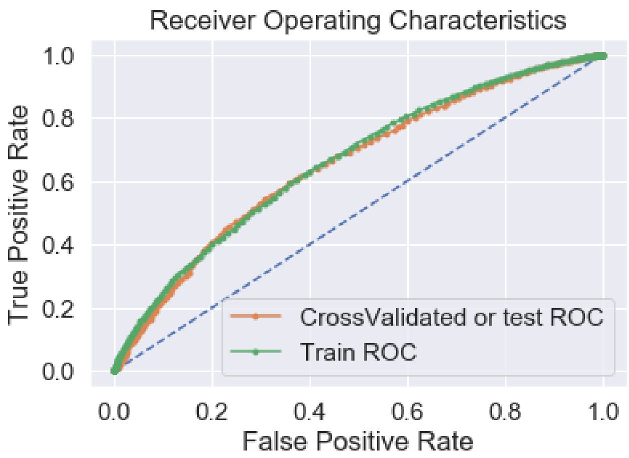


2.4.2 Applying KNN brute force on TFIDF, SET 2

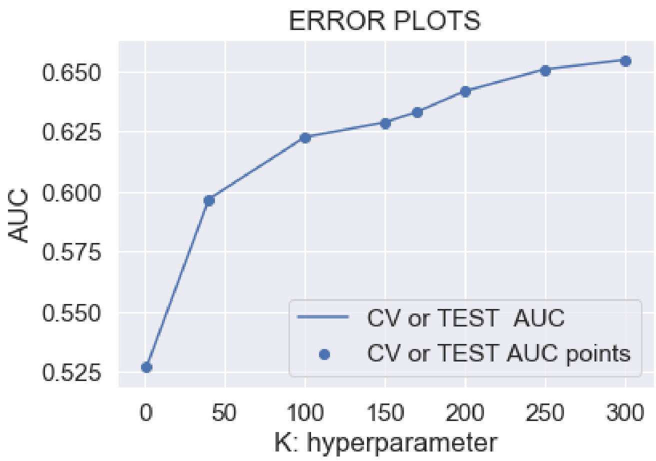
ROC AUC Score

In [46]:

```
k_from_roc(x2,y_sample,cv2,y_cv)
```




```
Optimal K  300
AUC:  0.6548102968221954
```



GRIDSEARCHCV

In [83]:

```
grid_search(x2,y_sample,cv2,y_cv)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV] n_neighbors=1 ....................................................
[CV] ......... n_neighbors=1, score=0.8932317544440763, total=  58.9s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  2.9min remaining:    0.0s
```

```
[CV] n_neighbors=1 ....................................................
[CV] ......... n_neighbors=1, score=0.8979794633984762, total=  55.9s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  5.7min remaining:    0.0s
```

```
[CV] n_neighbors=1 ....................................................
[CV] ......... n_neighbors=1, score=0.8946002650176678, total=  55.7s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  8.5min remaining:    0.0s
```

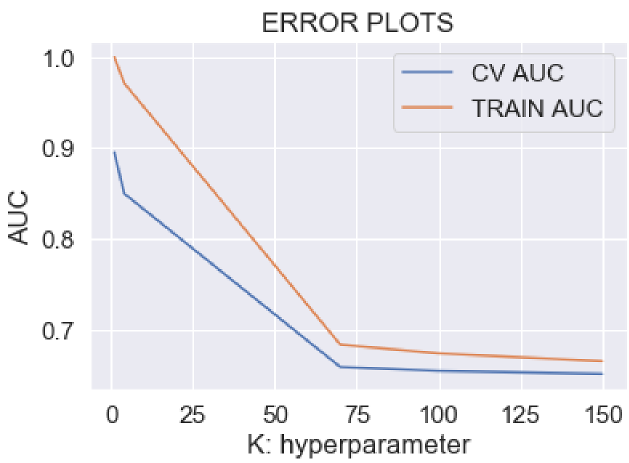
```
[CV] n_neighbors=4 ....................................................
[CV] ......... n_neighbors=4, score=0.8449300992625204, total= 1.0min
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed: 11.5min remaining:    0.0s
```

```
[CV] n_neighbors=4 ..................................................
[CV] .......... n_neighbors=4, score=0.8565942435971832, total= 1.0min
[CV] n_neighbors=4 ..................................................
[CV] .......... n_neighbors=4, score=0.8486061514865494, total= 1.0min
[CV] n_neighbors=70 .................................................
[CV] ......... n_neighbors=70, score=0.6598010850983167, total= 1.0min
[CV] n_neighbors=70 .................................................
[CV] ......... n_neighbors=70, score=0.6599012018261925, total= 1.0min
[CV] n_neighbors=70 .................................................
[CV] ......... n_neighbors=70, score=0.6583211394608655, total= 1.0min
[CV] n_neighbors=100 ................................................
[CV] ......... n_neighbors=100, score=0.655596225195235, total= 1.0min
[CV] n_neighbors=100 ................................................
[CV] ........ n_neighbors=100, score=0.6553872448594527, total= 1.0min
[CV] n_neighbors=100 ................................................
[CV] ......... n_neighbors=100, score=0.653978162185194, total= 1.0min
[CV] n_neighbors=150 ................................................
[CV] ........ n_neighbors=150, score=0.6526526410110131, total= 1.0min
[CV] n_neighbors=150 ................................................
[CV] ........ n_neighbors=150, score=0.6535113408145428, total= 1.1min
[CV] n_neighbors=150 ................................................
[CV] ........ n_neighbors=150, score=0.6491298023234309, total= 1.1min
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed: 45.9min finished
```

```
Optimal K  {'n_neighbors': 1}
AUC:  0.8952705189547293
```

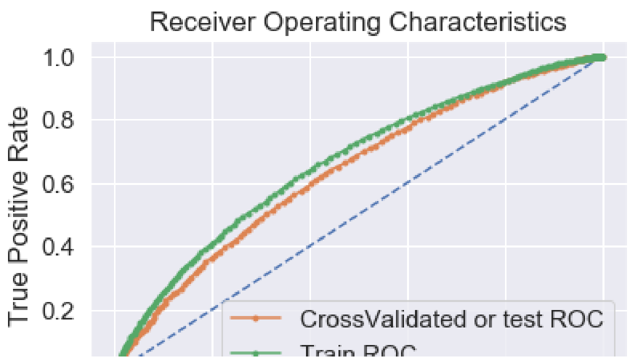


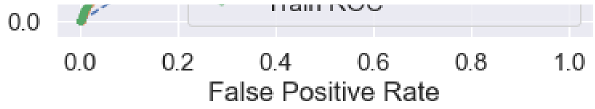

```
Accuracy on crossvalidated data:  0.5269257146258193
```

ROC Score on Test Data

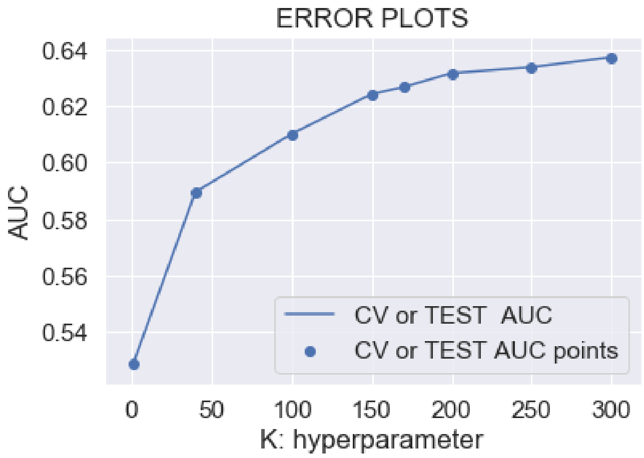
In [169]:

```
k_from_roc(x2,y_sample,t2,y_test)
```






```
Optimal K  300
AUC:  0.6373484561184461
```



CONFUSION MATRIX FOR TRAIN AND TEST DATA

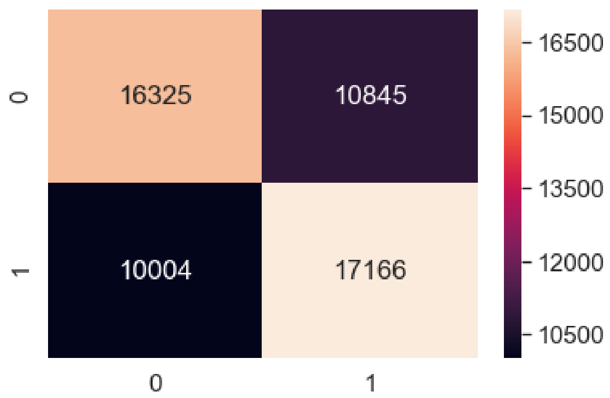
In [54]:

```python
# AUC score is higher for k =300 on test data . So I use that to calculate the confusion matrix on train data.



from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

print("TRAIN confusion matrix")
labels=[0,1]
cm=confusion_matrix(y_sample, predict(x2, y_sample, x2, y_sample,300),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
TRAIN confusion matrix
the maximum value of tpr*(1-fpr) 0.3796146998498663 for threshold 0.49
TRAIN AUC = 0.6624029058712243
```

Out[54]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1cbee4e0>
```



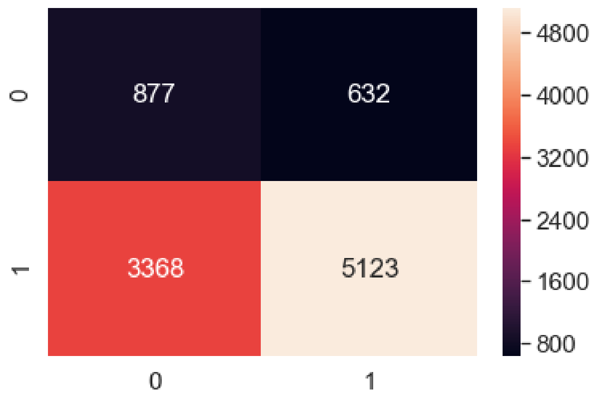
In [57]:

```
# AUC score is higher for k =300 on test data . So I use that to calculate the confusion matrix on test data.
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

print("Test confusion matrix")
labels=[0,1]
cm=confusion_matrix(y_test, predict(x2, y_sample, t2, y_test,300),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3796146998498663 for threshold 0.49
TRAIN AUC = 0.6624029058712243
```

Out[57]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x34185240>
```

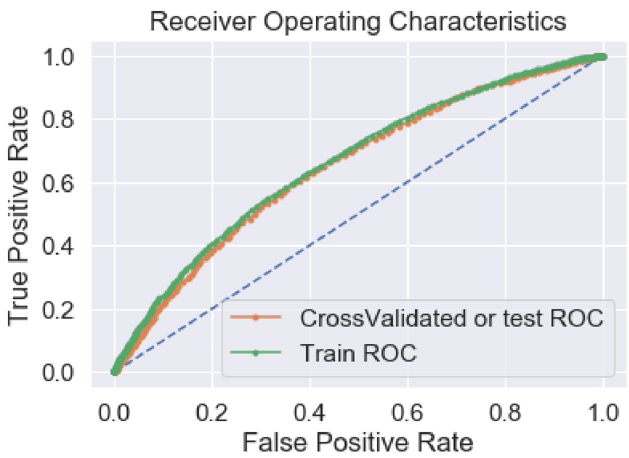


2.4.3 Applying KNN brute force on AVG W2V, SET 3

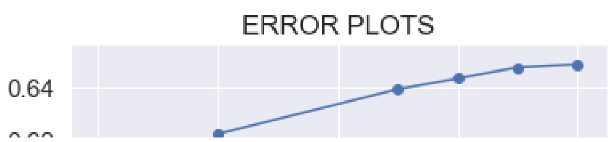
ROC AUC Score

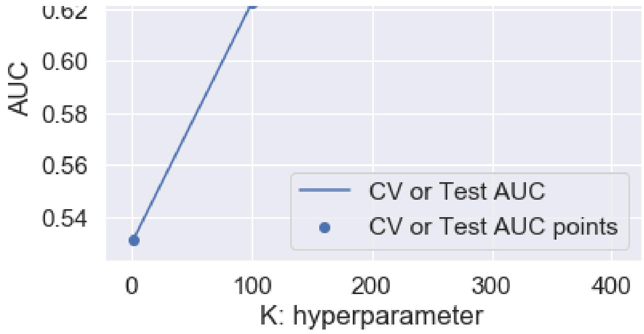
In [60]:

```
k_from_roc1(x3,y_sample,cv3,y_cv)
```




```
Optimal K  400
AUC:  0.6489971339715539
```





GRIDSEARCHCV

In [84]:

```
grid_search(x3,y_sample,cv3,y_cv)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV] n_neighbors=1 ...................................................
[CV] .......... n_neighbors=1, score=0.8990283758418902, total=13.6min
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed: 40.7min remaining:    0.0s
```

```
[CV] n_neighbors=1 ...................................................
[CV] .......... n_neighbors=1, score=0.9006845533841229, total=13.7min
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed: 81.7min remaining:    0.0s
```

```
[CV] n_neighbors=1 ...................................................
[CV] .......... n_neighbors=1, score=0.8986307420494699, total=13.6min
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed: 123.2min remaining:    0.0s
```

```
[CV] n_neighbors=4 ...................................................
[CV] .......... n_neighbors=4, score=0.8541302322541072, total=14.1min
```

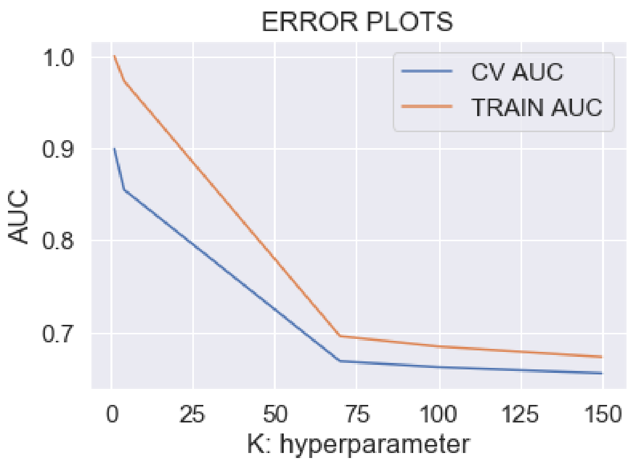
```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed: 166.7min remaining:    0.0s
```

```
[CV] n_neighbors=4 ...................................................
[CV] .......... n_neighbors=4, score=0.8580327487820839, total=14.4min
[CV] n_neighbors=4 ...................................................
[CV] ......... n_neighbors=4, score=0.8527258846896109, total=338.7min
[CV] n_neighbors=70 ..................................................
[CV] ......... n_neighbors=70, score=0.6708610546952588, total=14.4min
[CV] n_neighbors=70 ..................................................
[CV] ......... n_neighbors=70, score=0.6708651020320813, total=14.2min
[CV] n_neighbors=70 ..................................................
[CV] ......... n_neighbors=70, score=0.6645175845387632, total=14.4min
[CV] n_neighbors=100 .................................................
[CV] ........ n_neighbors=100, score=0.6644203520624699, total=14.8min
[CV] n_neighbors=100 .................................................
[CV] ........ n_neighbors=100, score=0.6653119730500032, total=13.1min
[CV] n_neighbors=100 .................................................
[CV] ........ n_neighbors=100, score=0.6571998588094183, total=13.9min
[CV] n_neighbors=150 .................................................
[CV] ........ n_neighbors=150, score=0.6590419534378523, total=14.4min
[CV] n_neighbors=150 .................................................
[CV] ....... n_neighbors=150, score=0.6576678825866124, total=461.0min
[CV] n_neighbors=150 .................................................
[CV] ........ n_neighbors=150, score=0.6502399625338684, total=13.9min
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed: 1434.8min finished
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed: [illegible]min finished
```

```
Optimal K  {'n_neighbors': 1}
AUC:  0.8994479205005521
```

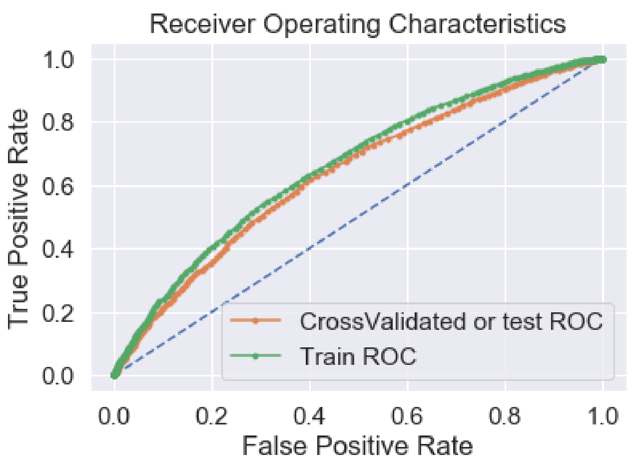



```
Accuracy on crossvalidated data:  0.5311041350500803
```

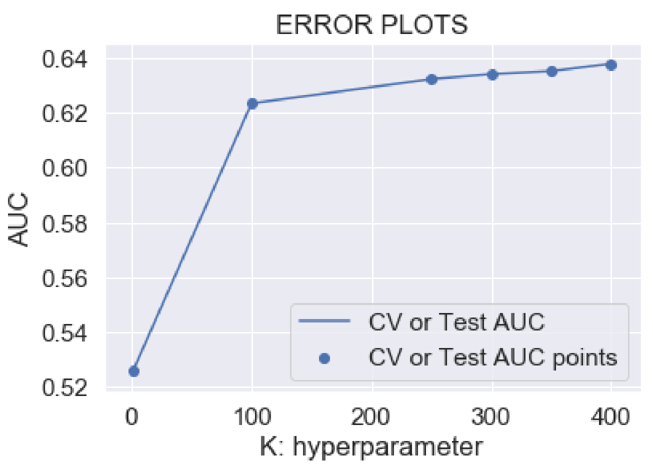
ROC Score of Test Data

In [86]:

```
k_from_roc1(x3,y_sample,t3,y_test)
```




```
Optimal K  400
AUC:  0.6380341981401739
```



CONFUSION MATRIX FOR TRAIN AND TEST DATA

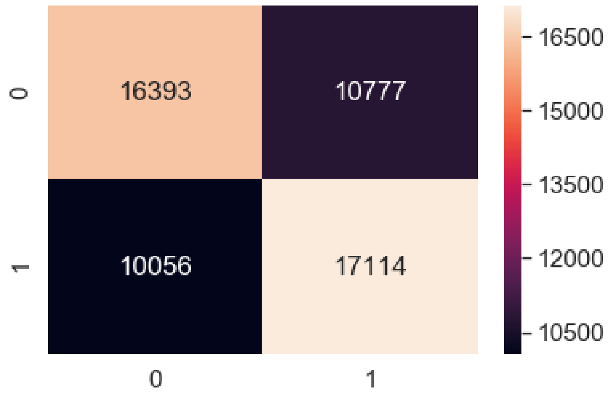
In [63]:

```
# AUC score is higher for k =400 on test data . So I use that to calculate the confusion matrix on train data.


from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

print("Test confusion matrix")
labels=[0,1]
cm=confusion_matrix(y_sample, predict(x3, y_sample, x3, y_sample,400),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.38004120784780565 for threshold 0.495
TRAIN AUC = 0.6621359360473709
```

Out[63]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2f808c18>
```



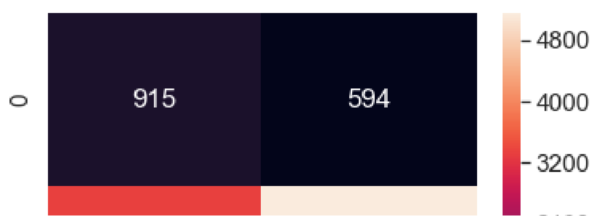
In [64]:

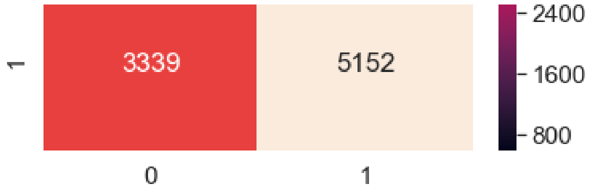
```
# AUC score is higher for k =400 on test data . So I use that to calculate the confusion matrix on test data.
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

print("Test confusion matrix")
labels=[0,1]
cm=confusion_matrix(y_test, predict(x3, y_sample, t3, y_test,400),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.38004120784780565 for threshold 0.495
TRAIN AUC = 0.6621359360473709
```

Out[64]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1b9b00>
```



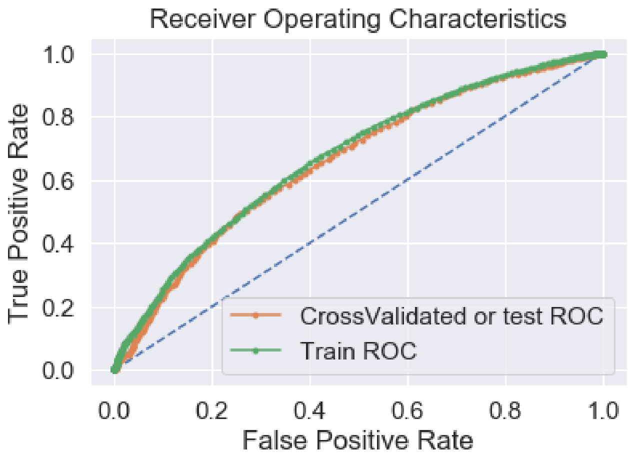


2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

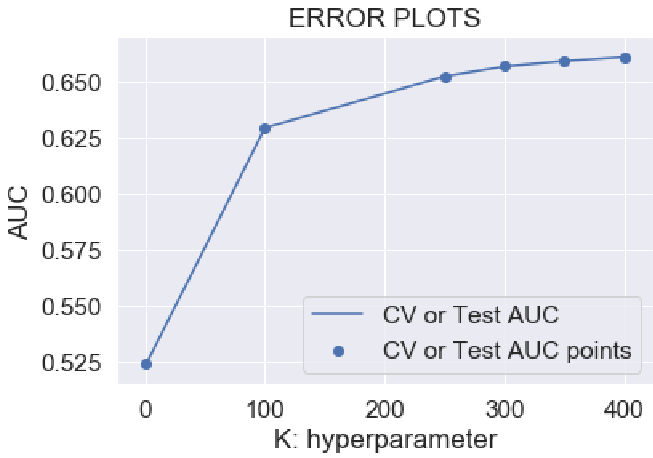
ROC AUC Score

In [65]:

```
k_from_roc1(x4,y_sample,cv4,y_cv)
```




```
Optimal K  400
AUC:  0.6612427311071597
```



GRIDSEARCHCV

In [85]:

```
grid_search(x4,y_sample,cv4,y_cv)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV] n_neighbors=1 ....................................................
[CV] ......... n_neighbors=1, score=0.8959920503477974, total=13.8min
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed: 41.6min remaining:    0.0s
```

```
[CV] n_neighbors=1 ...................................................
[CV] .......... n_neighbors=1, score=0.8990283758418903, total=13.8min
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed: 82.8min remaining:    0.0s
```

```
[CV] n_neighbors=1 ...................................................
[CV] .......... n_neighbors=1, score=0.8951523851590105, total=13.6min
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed: 123.4min remaining:    0.0s
```

```
[CV] n_neighbors=4 ...................................................
[CV] .......... n_neighbors=4, score=0.8523026036725023, total=13.5min
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed: 165.2min remaining:    0.0s
```

```
[CV] n_neighbors=4 ...................................................
[CV] .......... n_neighbors=4, score=0.8573818102369802, total=14.2min
[CV] n_neighbors=4 ...................................................
[CV] .......... n_neighbors=4, score=0.852022669214093l, total=13.0min
[CV] n_neighbors=70 ..................................................
[CV] ......... n_neighbors=70, score=0.6718357070415213, total=14.7min
[CV] n_neighbors=70 ..................................................
[CV] .......... n_neighbors=70, score=0.674191488696916, total=13.5min
[CV] n_neighbors=70 ..................................................
[CV] ......... n_neighbors=70, score=0.6678731227427456, total=15.9min
[CV] n_neighbors=100 .................................................
[CV] ........ n_neighbors=100, score=0.6672775097575256, total=13.8min
[CV] n_neighbors=100 .................................................
[CV] ........ n_neighbors=100, score=0.6705324950616091, total=13.3min
[CV] n_neighbors=100 .................................................
[CV] ........ n_neighbors=100, score=0.6617525697973504, total=13.7min
[CV] n_neighbors=150 .................................................
[CV] ........ n_neighbors=150, score=0.6678239550870446, total=13.0min
[CV] n_neighbors=150 .................................................
[CV] ........ n_neighbors=150, score=0.6702034356062434, total=14.4min
[CV] n_neighbors=150 .................................................
[CV] ........ n_neighbors=150, score=0.6608199891292812, total=15.8min
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed: 1421.3min finished
```

```
Optimal K  {'n_neighbors': 1}
AUC:  0.8967243283032756
```

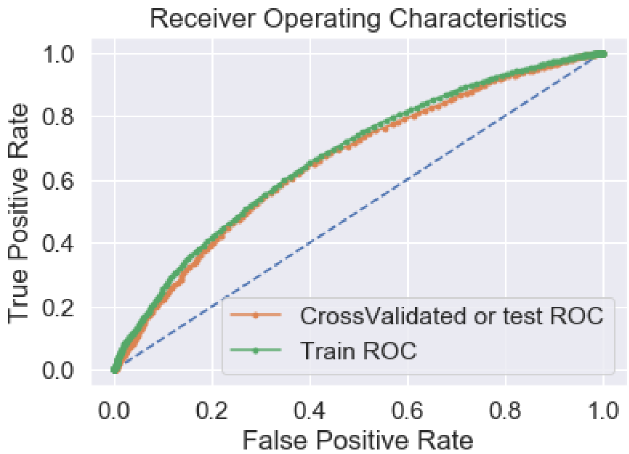

```
Accuracy on crossvalidated data:  0.5239335145797412
```

ROC Score of Test Data

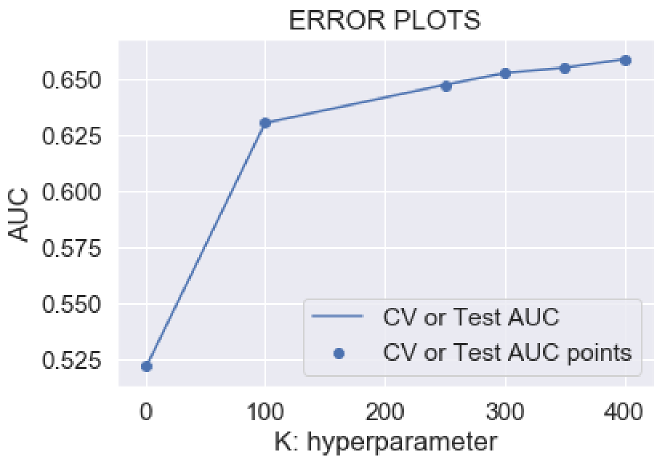
ROC Score of Test Data

In [67]:

```
k_from_roc1(x4,y_sample,t4,y_test)
```




```
Optimal K  400
AUC:  0.6586882348979183
```



CONFUSION MATRIX FOR TEST DATA

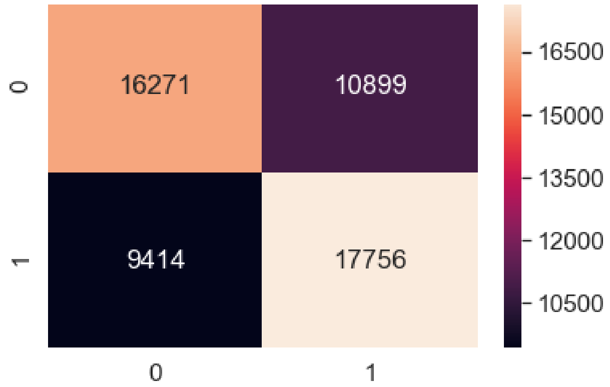
In [66]:

```
# AUC score is higher for k =250 on crossvalidated data . So I use that to calculate the confusion matrix on test data.


from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

print("Test confusion matrix")
labels=[0,1]
cm=confusion_matrix(y_sample, predict(x4, y_sample, x4, y_sample,400))
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3913633065112057 for threshold 0.48
TRAIN AUC = 0.6734379922810468
```

Out[66]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x311e7080>
```



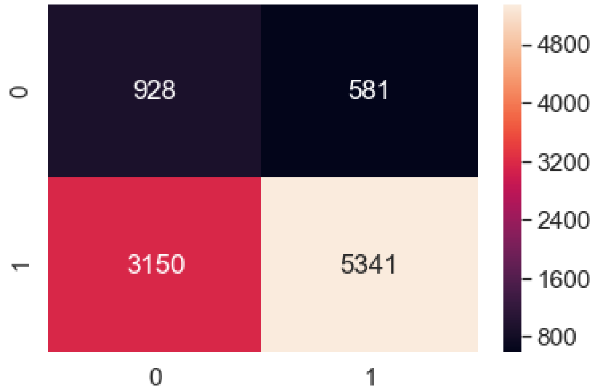
In [68]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

print("Test confusion matrix")
labels=[0,1]
cm=confusion_matrix(y_test, predict(x4, y_sample, t4, y_test,400))
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3913633065112057 for threshold 0.48
TRAIN AUC = 0.6734379922810468
```

Out[68]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x33c9ec18>
```



2.5 Feature selection with `SelectKBest`

In [95]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code

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

from sklearn.feature_selection import SelectKBest, f_classif
x1_k = SelectKBest(f_classif, k=2000).fit(x1, y_sample)
x2_k = SelectKBest(f_classif, k=2000).fit(x2, y_sample)
x1_best=x1_k.transform(x1)
x2_best=x2_k.transform(x2)
cv1_best = x1_k.transform(cv1)
```

```
cv2_best = x2_k.transform(cv2)
t1_best = x1_k.transform(t1)
t2_best = x2_k.transform(t2)
```

```
C:\Users\ADMIN\Anaconda3\lib\site-packages\sklearn\feature_selection\univariate_selection.py:114:
UserWarning:

Features [0 0 0 0] are constant.

C:\Users\ADMIN\Anaconda3\lib\site-packages\sklearn\feature_selection\univariate_selection.py:114:
UserWarning:

Features [0 0 0 0] are constant.
```

In [96]:

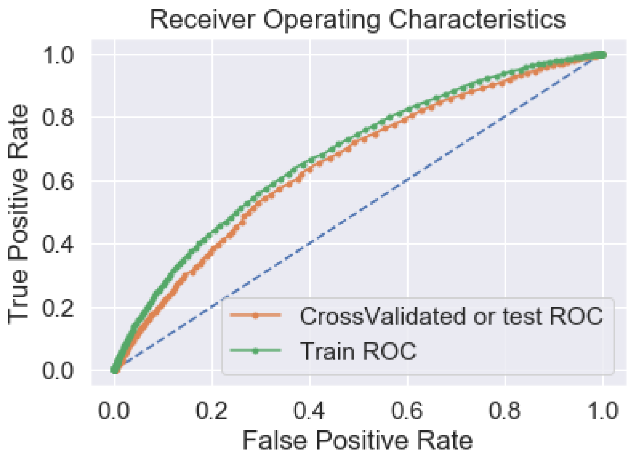
```
print(x1_best.shape)
print(x2_best.shape)
print(cv1_best.shape)
print(cv2_best.shape)
print(t1_best.shape)
print(t2_best.shape)
```

```
(54340, 2000)
(54340, 2000)
(8000, 2000)
(8000, 2000)
(10000, 2000)
(10000, 2000)
```

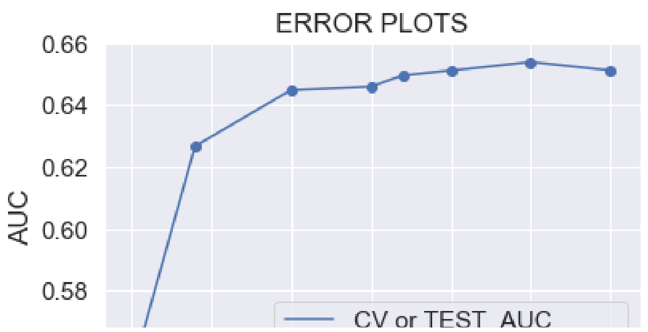
ROC AUC Score

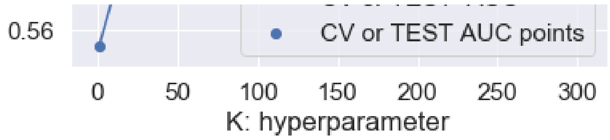
In [97]:

```
k_from_roc(x1_best,y_sample,cv1_best,y_cv)
```




```
Optimal K  250
AUC:  0.65389520207641
```





GRIDSEARCHCV

In [98]:

```
grid_search(x1_best,y_sample,cv1_best,y_cv)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV] n_neighbors=1 ...................................................
[CV] .......... n_neighbors=1, score=0.8589488793198631, total=  39.0s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.9min remaining:    0.0s
```

```
[CV] n_neighbors=1 ...................................................
[CV] .......... n_neighbors=1, score=0.8513304626255934, total=  39.2s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  3.8min remaining:    0.0s
```

```
[CV] n_neighbors=1 ...................................................
[CV] ........... n_neighbors=1, score=0.846731448763251, total=  37.7s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  5.8min remaining:    0.0s
```

```
[CV] n_neighbors=4 ...................................................
[CV] .......... n_neighbors=4, score=0.8167681262082502, total=  44.5s
```

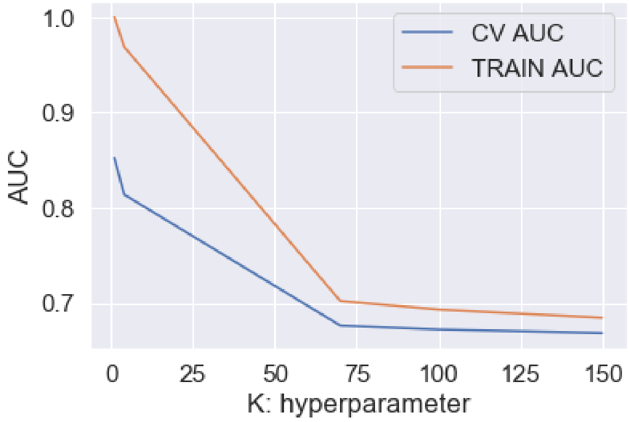
```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:  8.0min remaining:    0.0s
```

```
[CV] n_neighbors=4 ...................................................
[CV] .......... n_neighbors=4, score=0.8121870102699589, total=  44.7s
[CV] n_neighbors=4 ...................................................
[CV] .......... n_neighbors=4, score=0.8123921670832605, total=  44.9s
[CV] n_neighbors=70 ..................................................
[CV] ......... n_neighbors=70, score=0.6736838405042571, total=  44.8s
[CV] n_neighbors=70 ..................................................
[CV] ......... n_neighbors=70, score=0.6781139554258263, total=  45.3s
[CV] n_neighbors=70 ..................................................
[CV] ......... n_neighbors=70, score=0.6772433685103447, total=  44.6s
[CV] n_neighbors=100 .................................................
[CV] ........ n_neighbors=100, score=0.6678828852864421, total=  44.8s
[CV] n_neighbors=100 .................................................
[CV] ........ n_neighbors=100, score=0.6733668986778119, total=  45.1s
[CV] n_neighbors=100 .................................................
[CV] ........ n_neighbors=100, score=0.6753981373944143, total=  45.2s
[CV] n_neighbors=150 .................................................
[CV] ........ n_neighbors=150, score=0.6630554364821749, total=  43.5s
[CV] n_neighbors=150 .................................................
[CV] ........ n_neighbors=150, score=0.6688560808352639, total=  43.1s
[CV] n_neighbors=150 .................................................
[CV] ........ n_neighbors=150, score=0.6734423968959532, total=  42.1s
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed: 32.5min finished
```

```
Optimal K  {'n_neighbors': 1}
AUC:  0.8523371365476629
```

ERROR PLOTS




```
Accuracy on crossvalidated data:  0.5546828567982222
```

CONFUSION MATRIX FOR TRAIN DATA

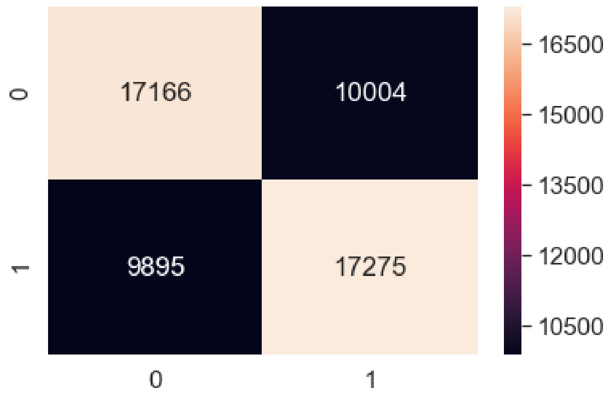
In [99]:

```
# AUC score is higher for k =170 on crossvalidated data . So I use that to calculate the confusion matrix on test data.


from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

print("Test confusion matrix")
labels=[0,1]
cm=confusion_matrix(y_sample, predict(x1_best, y_sample, x1_best, y_sample,250),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4017056012193838 for threshold 0.364
TRAIN AUC = 0.6830418015821809
```

Out[99]:

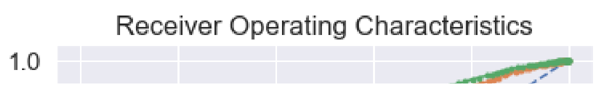
```
<matplotlib.axes._subplots.AxesSubplot at 0x30b1f4e0>
```

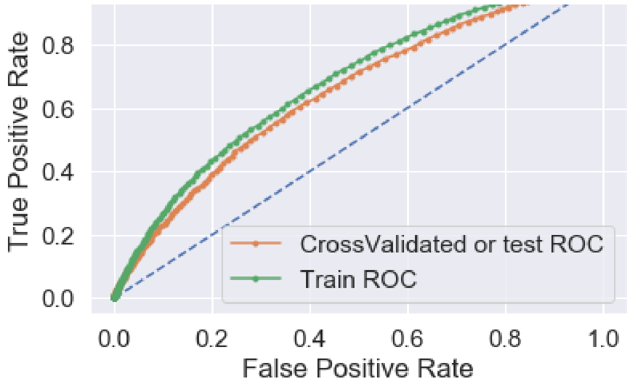


ROC Score of Test Data

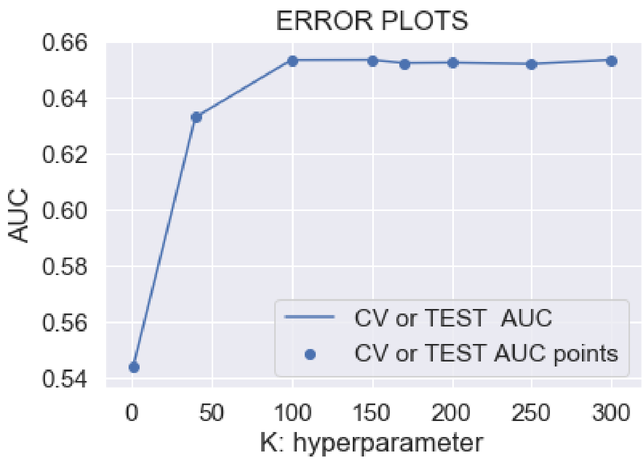
In [100]:

```
k_from_roc(x1_best,y_sample,t1_best,y_test)
```






```
Optimal K  300
AUC:  0.6535266866199654
```



Confusion Matrix of Test Data

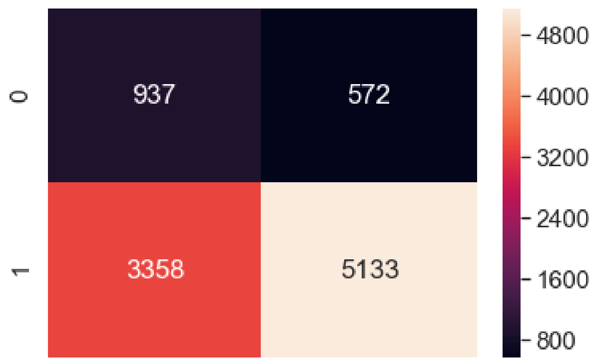
In [101]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

print("Test confusion matrix")
labels=[0,1]
cm=confusion_matrix(y_test, predict(x1_best, y_sample, t1_best, y_test,300),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3953380486201128 for threshold 0.367
TRAIN AUC = 0.6819411619122989
```

Out[101]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e6d1940>
```

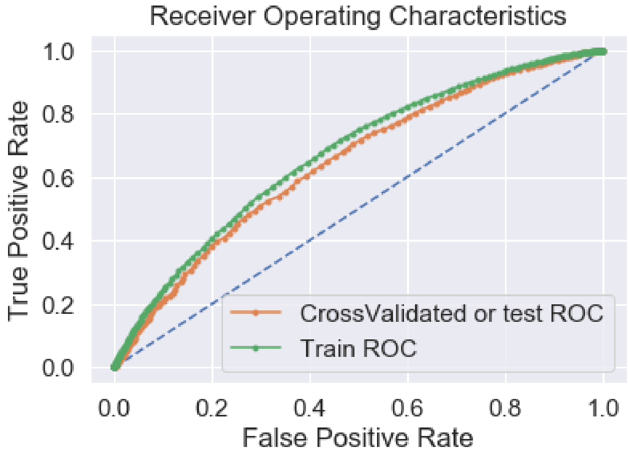


0 1

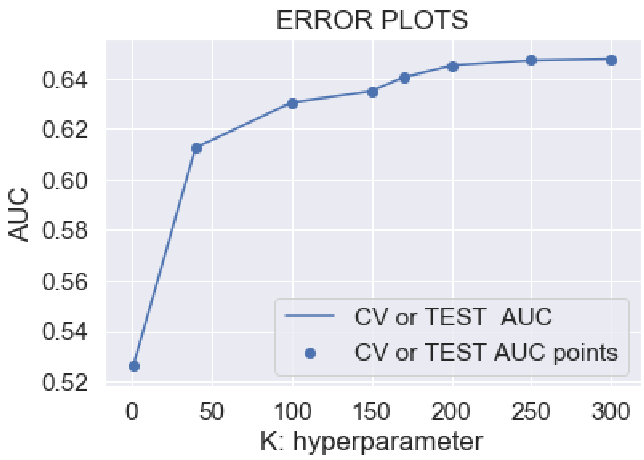
ROC AUC Score

In [102]:

```
k_from_roc(x2_best,y_sample,cv2_best,y_cv)
```




```
Optimal K  300
AUC:  0.6476075998600342
```



GRIDSEARCHCV

In [103]:

```
grid_search(x2_best,y_sample,cv2_best,y_cv)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[CV] n_neighbors=1 ...................................................
[CV] ......... n_neighbors=1, score=0.8924036656729601, total=  40.0s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  2.0min remaining:    0.0s
```

```
[CV] n_neighbors=1 ...................................................
[CV] ......... n_neighbors=1, score=0.8908578999668766, total=  43.1s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  4.0min remaining:    0.0s
```

```
[CV] n_neighbors=1 ..................................................
[CV] .......... n_neighbors=1, score=0.8901833038869258, total=  39.7s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  6.0min remaining:    0.0s
```

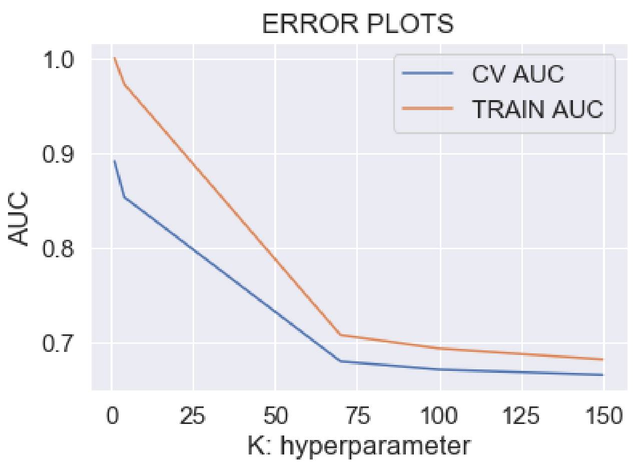
```
[CV] n_neighbors=4 ..................................................
[CV] .......... n_neighbors=4, score=0.8556318307875767, total=  44.4s
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:  8.3min remaining:    0.0s
```

```
[CV] n_neighbors=4 ..................................................
[CV] .......... n_neighbors=4, score=0.8530765593136175, total=  46.6s
[CV] n_neighbors=4 ..................................................
[CV] .......... n_neighbors=4, score=0.8507262050363502, total=  47.9s
[CV] n_neighbors=70 .................................................
[CV] .......... n_neighbors=70, score=0.679136084739725, total=  47.0s
[CV] n_neighbors=70 .................................................
[CV] ......... n_neighbors=70, score=0.6761252501531496, total=  48.4s
[CV] n_neighbors=70 .................................................
[CV] ......... n_neighbors=70, score=0.6839040615167189, total=  47.5s
[CV] n_neighbors=100 ................................................
[CV] ........ n_neighbors=100, score=0.6692701343639023, total=  50.6s
[CV] n_neighbors=100 ................................................
[CV] ........ n_neighbors=100, score=0.6682704604549043, total=  49.1s
[CV] n_neighbors=100 ................................................
[CV] ........ n_neighbors=100, score=0.6757785186636118, total=  48.0s
[CV] n_neighbors=150 ................................................
[CV] ........ n_neighbors=150, score=0.6662075926112647, total=  47.2s
[CV] n_neighbors=150 ................................................
[CV] ........ n_neighbors=150, score=0.6608636633988932, total=  47.9s
[CV] n_neighbors=150 ................................................
[CV] ........ n_neighbors=150, score=0.6694011286467244, total=  45.5s
```

```
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed: 34.6min finished
```

```
Optimal K  {'n_neighbors': 1}
AUC:  0.8911483253588517
```




```
Accuracy on crossvalidated data:  0.5261555129305462
```

CONFUSION MATRIX FOR TRAIN DATA

In [79]:

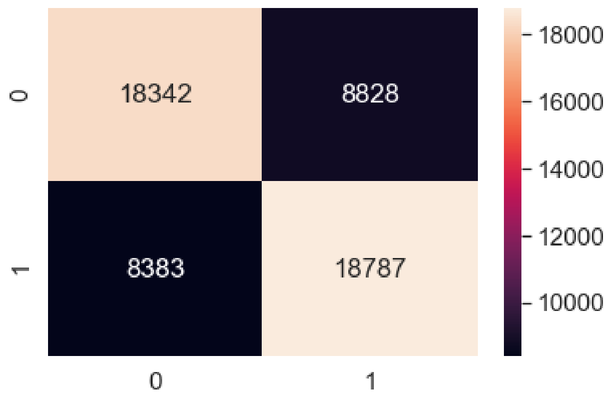
```
# AUC score is more or less same for the above two methods on crossvalidated data . So I use k=2 (
lower value of k)o calculate the confusion matrix on test data.


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

```
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

print("Test confusion matrix")
labels=[0,1]
cm=confusion_matrix(y_sample, predict(x2_best, y_sample, x2_best, y_sample,300))
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4667935512562908 for threshold 0.475
TRAIN AUC = 0.7516143018595415
```

Out[79]:

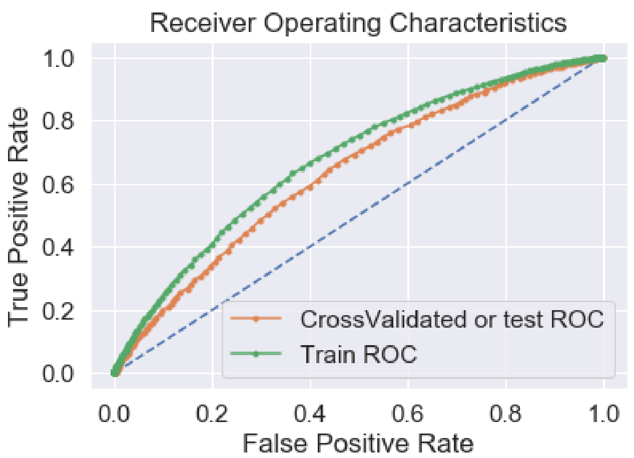
```
<matplotlib.axes._subplots.AxesSubplot at 0x33ddb2b0>
```



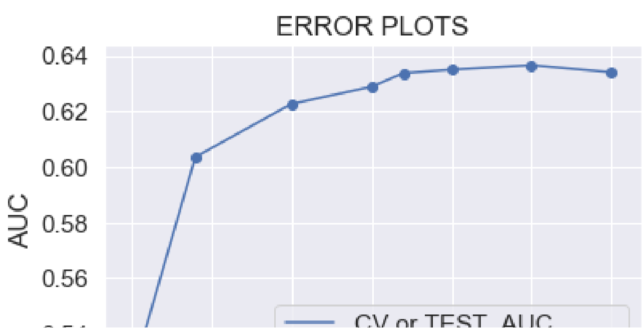
ROC Score of Test Data

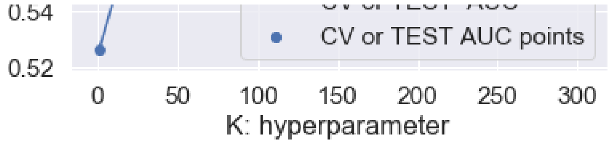
In [104]:

```
k_from_roc(x2_best,y_sample,t2_best,y_test)
```




```
Optimal K  250
AUC:  0.6365334862415036
```





Confusion Matrix of Test Data

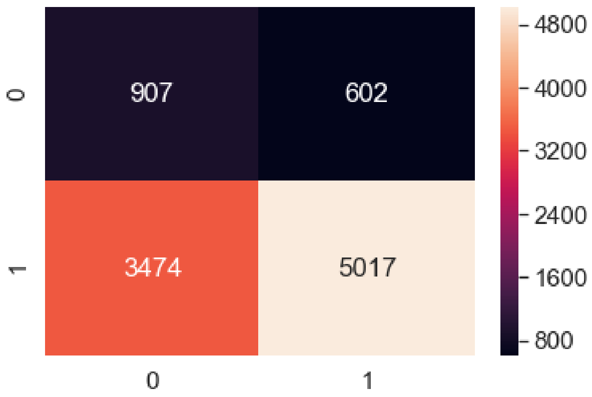
In [105]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

print("Test confusion matrix")
labels=[0,1]
cm=confusion_matrix(y_test, predict(x2_best, y_sample, t2_best, y_test,250))
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4011947810436856 for threshold 0.484
TRAIN AUC = 0.6776296411490026
```

Out[105]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2e54b7b8>
```



3. Conclusions

In [108]:

```
# Please compare all your models using Prettytable library
# I choose k which gives the highest AUC score.

from prettytable import PrettyTable
x=PrettyTable()
x.field_names=["Vectorizer","Model","Features","Hyperparameter","Test AUC"]
x.add_row(["BOW","Brute","2000","300","0.653"])
x.add_row(["TFIDF","Brute","2000","250","0.636"])
x.add_row(["W2V","Brute","All","400","0.638"])
x.add_row(["TFIDF W2V","Brute","All","400","0.658"])
```

In [109]:

```
print(x)
```

```
+------------+-------+----------+----------------+----------+
| Vectorizer | Model | Features | Hyperparameter | Test AUC |
+------------+-------+----------+----------------+----------+
|    BOW     | Brute |   2000   |      300       |  0.653   |
|   TFIDF    | Brute |   2000   |      250       |  0.636   |
|    W2V     | Brute |   All    |      400       |  0.638   |
```

```
|    W2V     | Brute |   All    |      400       |  0.658   |
| TFIDF W2V  | Brute |   All    |      400       |  0.658   |
+------------+-------+----------+----------------+----------+
```

In [ ]:

```

```