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). **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 [10]:

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [11]:

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

In [12]:

```
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']
```

1.2 preprocessing of project_subject_categories

In [13]:

```
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]))
```

1.3 preprocessing of project_subject_subcategories¶

In [14]:

```
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 [15]:

```
# 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 [16]:

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

Out[16]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_cate
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades P
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grade

In [17]:

```
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 7: SVM

1. **[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) on these feature sets**

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

2. **The hyper paramter tuning (best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**

   - Find the best hyper parameter which will give 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 you can also write your own for loops to do this task of hyperparameter tuning

3. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

4. **[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3**

   - Consider these set of features Set 5 :
     - **school_state** : categorical data
     - **clean_categories** : categorical data
     - **clean_subcategories** : categorical data
     - **project_grade_category** :categorical data
     - **teacher_prefix** : categorical data
     - **quantity** : numerical data
     - **teacher_number_of_previously_posted_projects** : numerical data
     - **price** : numerical data
     - **sentiment score's of each of the essay** : numerical data
     - **number of words in the title** : numerical data
     - **number of words in the combine essays** : numerical data

- **Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`) using elbow method** : numerical data

- **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-leakage, 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. Support Vector Machines

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

In [18]:

```
# 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
sample_data=project_data.sample(50000)
sample_data.shape
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,)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

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
def veccat(x):
    from sklearn.feature_extraction.text import CountVectorizer
    from sklearn.preprocessing import Normalizer
    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)




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

    price_scalar =  Normalizer(copy=False,norm='l2')
    price_scalar.fit(x['price'].values.reshape(1,-1)) # finding the mean and standard deviation of
this data



    # Now standardize the data with above maen and variance.
    price_standardized = price_scalar.transform(x['price'].values.reshape(1, -1))
    price_standardized = price_scalar.transform(x['price'].values.reshape(1, -1))
    price_standardized=np.transpose(price_standardized)

    projects_scalar = Normalizer(copy=False,norm='l2')
    projects_scalar.fit(x['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)) # f
inding the mean and standard deviation of this data

    # 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))
    projects_standardized =np.transpose(projects_standardized)

    qty_scalar= Normalizer(copy=False,norm='l2')
    qty_scalar.fit(x['quantity'].values.reshape(1,-1)) # finding the mean and standard deviation of
this data


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

    X1 = hstack((categories_one_hot, subcategories_one_hot,state_one_hot,prefix_one_hot,grade_one_h
ot, price_standardized,projects_standardized,qty_standardized))
    print(X1.shape)
    return(X1)
```

In [20]:

```
x=veccat(x_train)
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  (32000, 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  (32000, 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  (32000, 51)
['Mrs.', 'Ms.', 'Mr.', 'Teacher', 'Dr.']
Shape of matrix after one hot encodig  (32000, 5)
Shape of matrix after one hot encodig  (32000, 4)
(32000, 102)
['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)
Shape of matrix after one hot encodig  (10000, 4)
(10000, 102)
['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)
Shape of matrix after one hot encodig  (8000, 4)
(8000, 102)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

Vectorizing essay and project_title

In [21]:

```
# 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 [22]:

```
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 [23]:

```
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)
        # 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 [24]:

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

train_title=preprocessing(x_train['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%|███████████████████████████████████| 32000/32000 [00:25<00:00, 1278.85it/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, 1231.91it/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 [00:06<00:00, 1304.34it/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%|██████████████████████████████████| 32000/32000 [00:01<00:00, 27071.21it/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, 26453.51it/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, 24538.48it/s]
```

Encoding using BOW

In [29]:

```
from scipy.sparse import hstack
from sklearn.preprocessing import StandardScaler
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(ngram_range=(1,2),min_df=10,max_features=5000)

    vectorizer.fit(train_essay)

    text_bow=vectorizer.transform(train_essay)

    text_bow1 = vectorizer.transform(test_essay)

    text_bow2 = vectorizer.transform(cv_essay)

    vectorizer= CountVectorizer()
    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 [30]:

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

```
(32000, 14524)
(10000, 14524)
(8000, 14524)
```

In [33]:

```
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(ngram_range=(1,2),min_df=10,max_features=5000)
    vectorizer.fit(train_essay)

    text_tfidf=vectorizer.transform(train_essay)
    text_tfidf1 = vectorizer.transform(test_essay)
    text_tfidf2 = vectorizer.transform(cv_essay)
    vectorizer = TfidfVectorizer()
    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 ))
    t1= hstack((t1,text_tfidf1,title_tfidf1 ))
    cv1 = hstack((cv1,text_tfidf2,title_tfidf2 ))



    return x1,t1,cv1
```

In [34]:

```
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)
```

```
(32000, 14524)
(10000, 14524)
(8000, 14524)
```

In [35]:

```
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 [36]:

```
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 [37]:

```
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%|██████████████████████████████████| 32000/32000 [00:12<00:00, 2523.12it/s]
100%|██████████████████████████████████| 10000/10000 [00:03<00:00, 2675.07it/s]
100%|████████████████████████████████████| 8000/8000 [00:03<00:00, 2509.27it/s]
100%|█████████████████████████████████| 32000/32000 [00:00<00:00, 46644.56it/s]
100%|█████████████████████████████████| 10000/10000 [00:00<00:00, 45659.49it/s]
100%|███████████████████████████████████| 8000/8000 [00:00<00:00, 49076.92it/s]
```

```
(32000, 702)
(10000, 702)
(8000, 702)
```

Encoding using TFIDFW2V

In [38]:

```
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 [39]:

```
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 [40]:

```
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%|███████████████████████████████████| 32000/32000 [01:35<00:00, 336.04it/s]
100%|███████████████████████████████████| 10000/10000 [00:36<00:00, 270.36it/s]
100%|█████████████████████████████████████| 8000/8000 [00:24<00:00, 321.24it/s]
100%|█████████████████████████████████| 32000/32000 [00:01<00:00, 17826.28it/s]
100%|█████████████████████████████████| 10000/10000 [00:00<00:00, 15336.54it/s]
100%|███████████████████████████████████| 8000/8000 [00:00<00:00, 18098.52it/s]
```

```
(32000, 702)
(10000, 702)
(8000, 702)
```

2.4 Appling Support Vector Machines on different kind of featurization as mentioned in the instructions

Apply Support Vector Machines 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 instrucations

In [48]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# 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
def alpha_from_roc(x_train,y_train,x_cv,y_cv,penalty):
    from sklearn.linear_model import SGDClassifier
    from sklearn.model_selection import cross_val_score
    from sklearn.metrics import roc_curve
    from sklearn.metrics import roc_auc_score


    a_range=[0.001,0.01,0.1,1,10,100,1000,10000]
    scores_auc=[]

    for a in a_range:
        clf=SGDClassifier(loss='hinge',alpha=a,class_weight='balanced',penalty=penalty)
        clf.fit(x_train,y_train)
        score_roc = clf.decision_function(x_cv)
        fpr, tpr, thresholds = metrics.roc_curve(y_cv, score_roc)
        roc_auc = auc(fpr, tpr)
        scores_auc.append(roc_auc)
    # Chosen optimal k with the highest AUC score
    optimal_alpha = a_range[scores_auc.index(max(scores_auc))]

    clf =SGDClassifier(loss='hinge',alpha=optimal_alpha,class_weight='balanced',penalty=penalty)
    clf.fit(x_train,y_train)

    score_roc = clf.decision_function(x_cv)
    fpr, tpr, thresholds = metrics.roc_curve(y_cv, score_roc)


    score_roc = clf.decision_function(x_train)
    fpr1, tpr1, thresholds = metrics.roc_curve(y_train, score_roc)

    # 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 alpha ",optimal_alpha)
    print("AUC: ", max(scores_auc))
    plt.plot(np.log10(a_range), scores_auc, label='CV or TEST AUC')
    plt.scatter(np.log10(a_range), scores_auc, label='CV or TEST AUC points')
    plt.xlabel("LOG(alpha): hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.legend()

    plt.show()
```

In [49]:

```
def kfold_cv(x_train,y_train,penalty):
    from sklearn.model_selection import  cross_val_score
    from sklearn.linear_model import SGDClassifier
    a_range=[0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
    scores_cv=[]
    for a in a_range:
        clf=SGDClassifier(loss='hinge',alpha=a,class_weight='balanced',penalty=penalty)
        scores_cv.append(np.mean(cross_val_score(clf, x_train, y_train, cv=3, n_jobs=1)))
    for i in range(len(scores_cv)):
        if scores_cv[i]==max(scores_cv):
            print("Optimal alpha :",a_range[i])
            print("CV SCORE ",max(scores_cv))
```

In [51]:

```
# Gridsearchcv function

def grid_search(x_train,y_train,x_cv,y_cv,penalty):
    from sklearn.linear_model import SGDClassifier
    from sklearn.model_selection import GridSearchCV
    from sklearn.metrics import roc_curve
    from sklearn.metrics import roc_auc_score
    from sklearn import preprocessing

    a_range=[0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

    parameters = dict(alpha=a_range)
    # Fitted the model on train data
    clf1 = GridSearchCV(SGDClassifier(loss='hinge',class_weight='balanced',penalty=penalty),
parameters, cv=3,verbose=15, scoring='roc_auc')
    clf1.fit(x_train, y_train)
    train_auc= clf1.cv_results_['mean_train_score']
    cv_auc = clf1.cv_results_['mean_test_score']
    max_score= clf1.best_score_

    opt_par=clf1.best_params_
    print("Optimal alpha ",opt_par)
    print("AUC: ", max_score)
    plt.plot(parameters['alpha'], cv_auc, label='CV AUC')
    plt.plot(parameters['alpha'], train_auc, label='TRAIN AUC')
    plt.xlabel("Alpha: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.legend()
    plt.show()
    # Found out the score for crossvalidated data
    print("Accuracy on crossvalidated data: " , clf1.score(x_cv,y_cv))
```

In [77]:

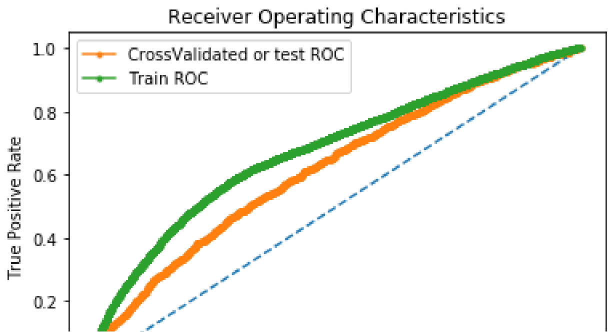
```
def predict(x_train,y_train,x_test,y_test,a,penalty):
    from sklearn.linear_model import SGDClassifier
    clf=SGDClassifier(loss='hinge',alpha=a,class_weight='balanced',penalty=penalty)
    clf.fit(x_train,y_train)
    score_roc = clf.decision_function(x_test)
    fpr, tpr, thresholds = metrics.roc_curve(y_test, score_roc)
    t = thresholds[np.argmax(tpr*(1-fpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    print("train or test AUC =",str(auc(fpr, tpr)))
    predictions=[]
    predictions=clf.predict(x_test)
    return predictions
```

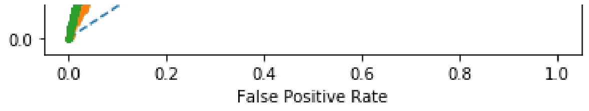
Applying SVM on BOW

ROC SCORE on CV data

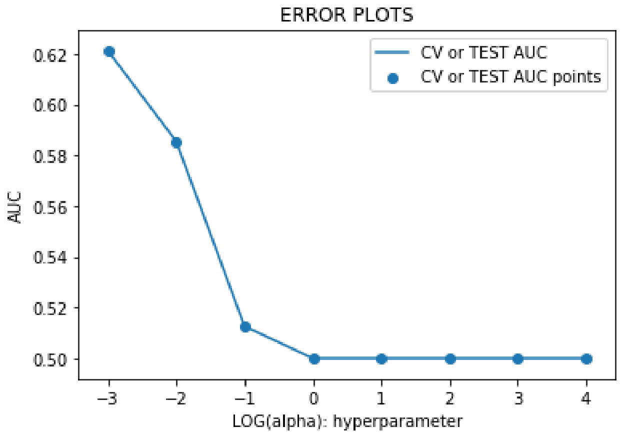
In [53]:

```
alpha_from_roc(x1,y_train,cv1,y_cv,'l1')
```



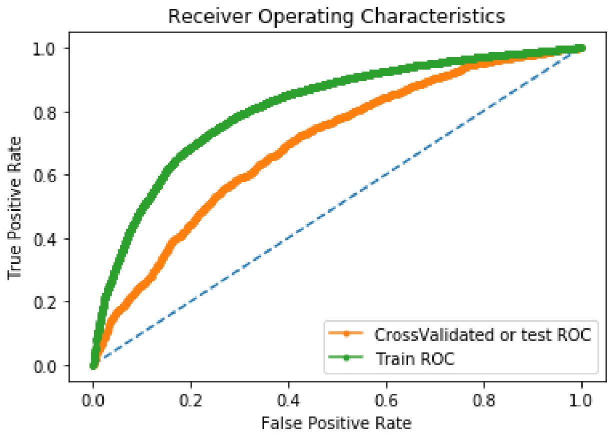


```
Optimal alpha  0.001
AUC:  0.6210476414528805
```

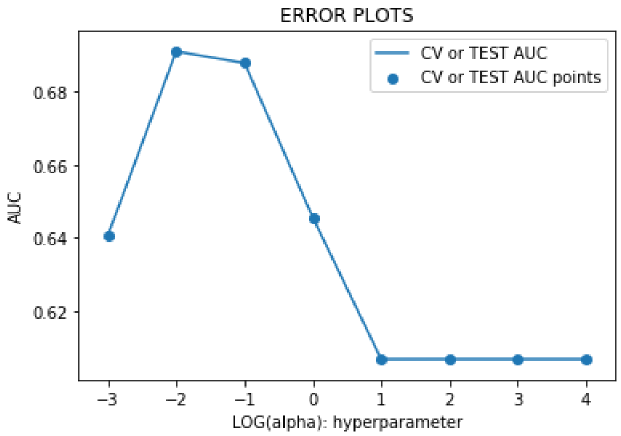


In [54]:

```
alpha_from_roc(x1,y_train,cv1,y_cv,'l2')
```



```
Optimal alpha  0.01
AUC:  0.6909148979762538
```



K FOLD Cross Validation

In [55]:

```
kfold_cv(x1,y_train,'l1')
```

```
Optimal alpha : 10
CV SCORE  0.8498750027332277
Optimal alpha : 1000
CV SCORE  0.8498750027332277
Optimal alpha : 10000
CV SCORE  0.8498750027332277
```

In [56]:

```
kfold_cv(x1,y_train,'l2')
```

```
Optimal alpha : 1000
CV SCORE  0.8498750027332277
```

GridsearchCV

In [57]:

```
grid_search(x1,y_train,cv1,y_cv,'l1')
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
```

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

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6252378176628431, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6182360514474372, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.5s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6146769824292801, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.8s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6222479525234974, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    1.2s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............. alpha=0.001, score=0.607492662645692, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    1.5s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6135752854410836, total=   0.2s
```

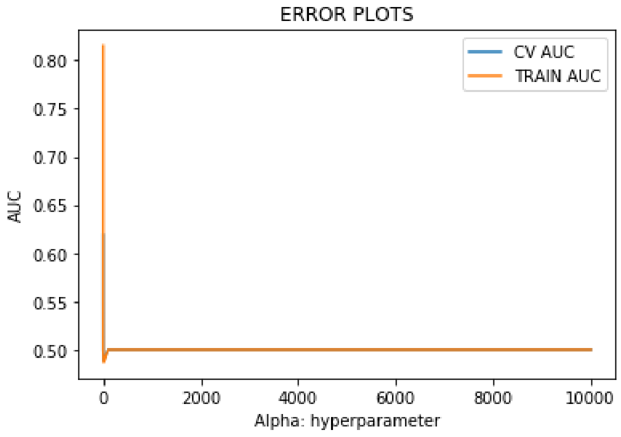
```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    1.8s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.6021105319211343, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    2.0s remaining:    0.0s
```

```
[CV] alpha=0.01
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5676290983331226, total=   0.1s

[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    2.3s remaining:    0.0s

[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5659477856676036, total=   0.1s

[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    2.5s remaining:    0.0s

[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.5422803863359638, total=   0.1s

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    2.8s remaining:    0.0s

[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.5315767206995902, total=   0.1s

[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    3.0s remaining:    0.0s

[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.5636719397315453, total=   0.1s

[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    3.2s remaining:    0.0s

[CV] alpha=1 .........................................................
[CV] ............................... alpha=1, score=0.5, total=   0.1s

[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    3.5s remaining:    0.0s

[CV] alpha=1 .........................................................
[CV] ............................... alpha=1, score=0.5, total=   0.1s

[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    3.7s remaining:    0.0s

[CV] alpha=1 .........................................................
[CV] ............................... alpha=1, score=0.5, total=   0.1s
[CV] alpha=10 ........................................................
[CV] .............................. alpha=10, score=0.5, total=   0.1s
[CV] alpha=10 ........................................................
[CV] .............. alpha=10, score=0.46632510086601286, total=   0.1s
[CV] alpha=10 ........................................................
[CV] .............................. alpha=10, score=0.5, total=   0.1s
[CV] alpha=100 .......................................................
[CV] ............................. alpha=100, score=0.5, total=   0.1s
[CV] alpha=100 .......................................................
[CV] ............................. alpha=100, score=0.5, total=   0.1s
[CV] alpha=100 .......................................................
[CV] ............................. alpha=100, score=0.5, total=   0.1s
[CV] alpha=1000 ......................................................
[CV] ............................ alpha=1000, score=0.5, total=   0.1s
[CV] alpha=1000 ......................................................
[CV] ............................ alpha=1000, score=0.5, total=   0.1s
[CV] alpha=1000 ......................................................
[CV] ............................ alpha=1000, score=0.5, total=   0.1s
[CV] alpha=10000 .....................................................
[CV] ........................... alpha=10000, score=0.5, total=   0.1s
[CV] alpha=10000 .....................................................
[CV] ........................... alpha=10000, score=0.5, total=   0.1s
[CV] alpha=10000 .....................................................
[CV] ........................... alpha=10000, score=0.5, total=   0.1s

[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:    6.6s finished

Optimal alpha  {'alpha': 0.0001}
AUC:  0.6193839830673836
```



```
Accuracy on crossvalidated data:  0.6391042469756352
```

In [58]:

```
grid_search(x1,y_train,cv1,y_cv,'l2')
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
```

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

```
[CV] alpha=0.0001 ....................................................
[CV] .......... alpha=0.0001, score=0.6021925012111212, total=   0.0s
```

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

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.638869046614206, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] .......... alpha=0.0001, score=0.6172248246666021, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.4s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ........... alpha=0.001, score=0.6296381329537064, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.5s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ........... alpha=0.001, score=0.6379963157334443, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.7s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ........... alpha=0.001, score=0.6175140812778004, total=   0.0s
```

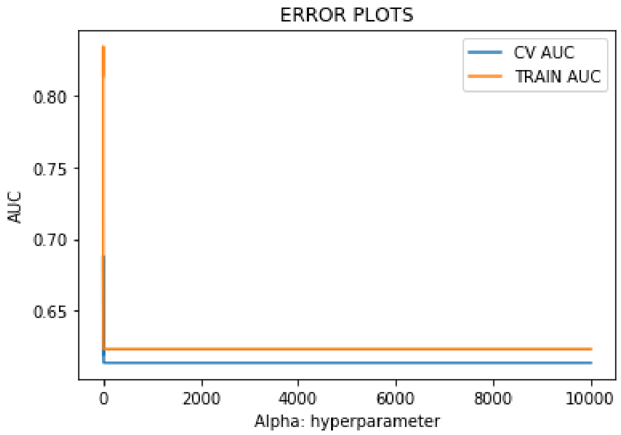
```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.9s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............ alpha=0.01, score=0.6754502561738264, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    1.0s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............ alpha=0.01, score=0.6722961690003584, total=   0.0s
```

```
[CV] .............. alpha=0.01, score=0.672296169005584, total=   0.0s

[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    1.2s remaining:    0.0s

[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.6628016893743671, total=   0.0s

[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    1.4s remaining:    0.0s

[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.6901345673412316, total=   0.0s

[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    1.6s remaining:    0.0s

[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.6930017194851673, total=   0.1s

[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    1.7s remaining:    0.0s

[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.6810907275616832, total=   0.1s

[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    1.9s remaining:    0.0s

[CV] alpha=1 .........................................................
[CV] ................. alpha=1, score=0.659740244449567, total=   0.0s

[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    2.1s remaining:    0.0s

[CV] alpha=1 .........................................................
[CV] ................ alpha=1, score=0.6572254034554383, total=   0.0s

[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    2.3s remaining:    0.0s

[CV] alpha=1 .........................................................
[CV] ................ alpha=1, score=0.6513289232839513, total=   0.0s
[CV] alpha=10 ........................................................
[CV] ................ alpha=10, score=0.619184552565415, total=   0.0s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.6098465761711946, total=   0.0s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.6117522384141463, total=   0.0s
[CV] alpha=100 .......................................................
[CV] .............. alpha=100, score=0.6192124379601607, total=   0.0s
[CV] alpha=100 .......................................................
[CV] .............. alpha=100, score=0.6094062832351401, total=   0.0s
[CV] alpha=100 .......................................................
[CV] ............... alpha=100, score=0.611788481619837, total=   0.0s
[CV] alpha=1000 ......................................................
[CV] ............. alpha=1000, score=0.6192117494318954, total=   0.0s
[CV] alpha=1000 ......................................................
[CV] ............. alpha=1000, score=0.6094062832351401, total=   0.0s
[CV] alpha=1000 ......................................................
[CV] ............. alpha=1000, score=0.6117950274459599, total=   0.1s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.6192082379377422, total=   0.0s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.6094062832351401, total=   0.1s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.6117952341562586, total=   0.0s

[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:    4.5s finished

Optimal alpha  {'alpha': 0.1}
AUC:  0.6880758001436864
```



```
Accuracy on crossvalidated data:  0.6884505349821856
```

Train Confusion matrix

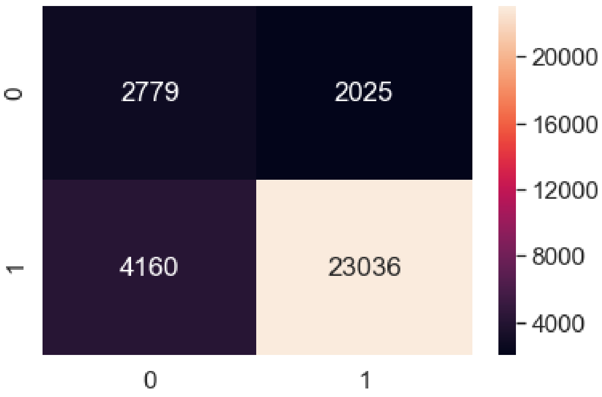
In [80]:

```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
labels=[0,1]
print("Train confusion matrix")
cm=confusion_matrix(y_train, predict(x1, y_train, x1, y_train,0.01,'l2'),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.5412132425924907 for threshold 0.397
train or test AUC = 0.8026756173980624
```

Out[80]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x6842cfd0>
```



Test Confusion matrix

In [81]:

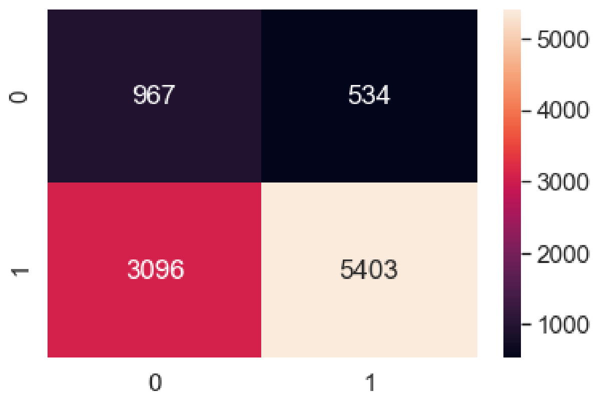
```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict(x1, y_train, t1, y_test,0.01,'l2'),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
```

```
the maximum value of tpr*(1-fpr) 0.4131520273694464 for threshold 0.02
train or test AUC = 0.6936661200647583
```

Out[81]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2d94ffd0>
```

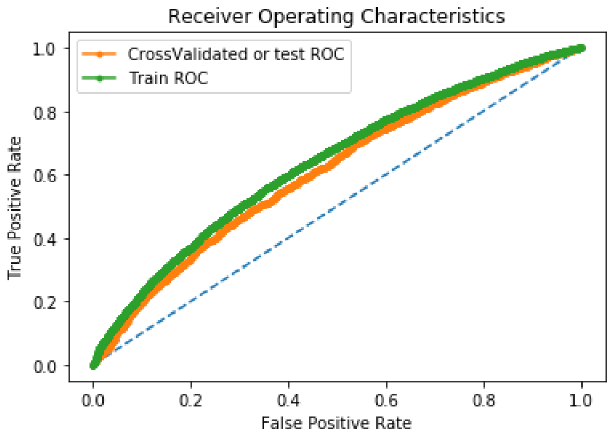


Encoding in TFIDF

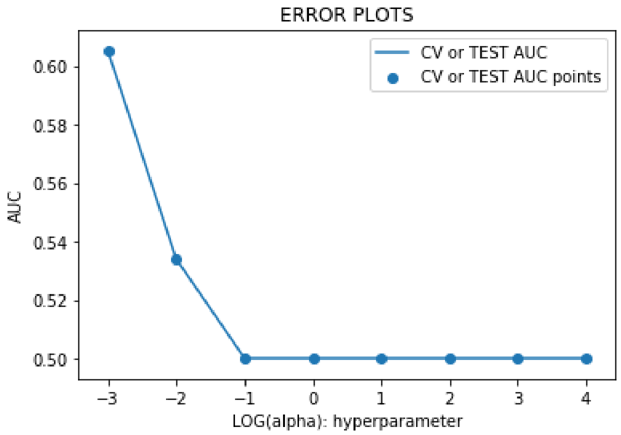
ROC SCORE on CV data

In [59]:

```
alpha_from_roc(x2,y_train,cv2,y_cv,'l1')
```

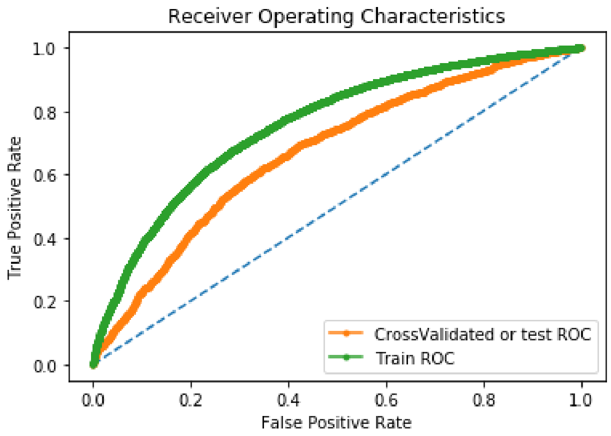


```
Optimal alpha  0.001
AUC:  0.6052039782996936
```

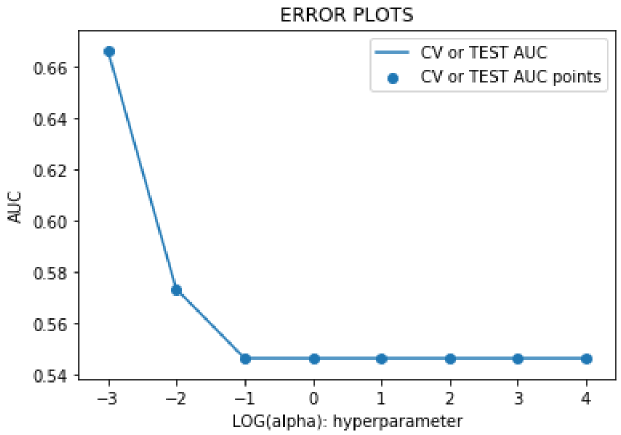


In [60]:

```
alpha_from_roc(x2,y_train,cv2,y_cv,'l2')
```



```
Optimal alpha  0.001
AUC:  0.6660378497646039
```



K FOLD Cross Validation

In [61]:

```
kfold_cv(x2,y_train,'l1')
```

```
Optimal alpha : 10
CV SCORE  0.8498750027332277
Optimal alpha : 100
CV SCORE  0.8498750027332277
Optimal alpha : 10000
CV SCORE  0.8498750027332277
```

In [62]:

```
kfold_cv(x2,y_train,'l2')
```

```
Optimal alpha : 100
CV SCORE  0.8498750027332277
Optimal alpha : 10000
CV SCORE  0.8498750027332277
```

GridsearchCV

In [63]:

```
grid_search(x2,y_train,cv2,y_cv,'l1')
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
```

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

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6542334642363272, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6583494940593182, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.5s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6527116084714013, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.8s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6130487329289744, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    1.1s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6170438153484463, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    1.3s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6249891390963934, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    1.6s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5291692589755856, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    1.8s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5277544405678607, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    2.0s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5420183124653545, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    2.3s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] ............................. alpha=0.1, score=0.5, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    2.5s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] ............................. alpha=0.1, score=0.5, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    2.7s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] ............................. alpha=0.1, score=0.5, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    2.9s remaining:    0.0s
```

```
[CV] alpha=1 ........................................................
[CV] ............................... alpha=1, score=0.5, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    3.2s remaining:    0.0s
```

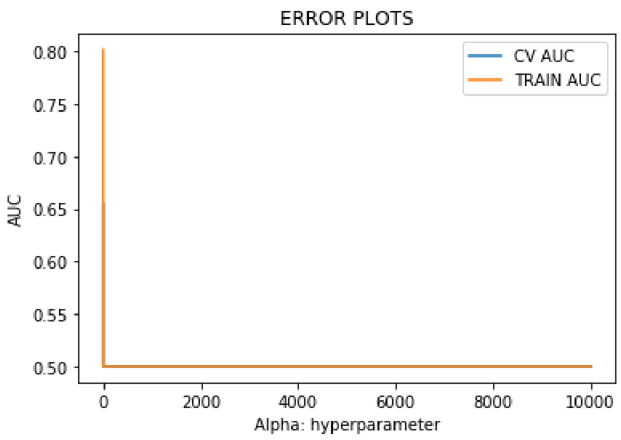
```
[CV] alpha=1 ........................................................
[CV] ............................... alpha=1, score=0.5, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    3.4s remaining:    0.0s
```

```
[CV] alpha=1 ........................................................
[CV] ............................... alpha=1, score=0.5, total=   0.1s
[CV] alpha=10 .......................................................
[CV] .............................. alpha=10, score=0.5, total=   0.1s
[CV] alpha=10 .......................................................
[CV] .............................. alpha=10, score=0.5, total=   0.1s
[CV] alpha=10 .......................................................
[CV] .............................. alpha=10, score=0.5, total=   0.1s
[CV] alpha=100 ......................................................
[CV] ............................. alpha=100, score=0.5, total=   0.1s
[CV] alpha=100 ......................................................
[CV] ............................. alpha=100, score=0.5, total=   0.1s
[CV] alpha=100 ......................................................
[CV] ............................. alpha=100, score=0.5, total=   0.1s
[CV] alpha=1000 .....................................................
[CV] ............................ alpha=1000, score=0.5, total=   0.1s
[CV] alpha=1000 .....................................................
[CV] ............................ alpha=1000, score=0.5, total=   0.1s
[CV] alpha=1000 .....................................................
[CV] ............................ alpha=1000, score=0.5, total=   0.1s
[CV] alpha=10000 ....................................................
[CV] ........................... alpha=10000, score=0.5, total=   0.1s
[CV] alpha=10000 ....................................................
[CV] ........................... alpha=10000, score=0.5, total=   0.1s
[CV] alpha=10000 ....................................................
[CV] ........................... alpha=10000, score=0.5, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:    6.4s finished
```

```
Optimal alpha  {'alpha': 0.0001}
AUC:  0.655098134877056
```



```
Accuracy on crossvalidated data:  0.6776775347405621
```

In [64]:

```
grid_search(x2,y_train,cv2,y_cv,'l2')
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
Fitting 3 folds for each of 9 candidates, totalling 27 fits
```

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

```
[CV] alpha=0.0001 ....................................................
[CV] ............ alpha=0.0001, score=0.647141967367616, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6354703847877756, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6457105373675375, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.3s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6535166374592977, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.5s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6699683354274235, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.6s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6664247007782298, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.8s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] .............. alpha=0.01, score=0.584364920806856, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.9s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5879284630779232, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    1.1s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5999315099877249, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    1.3s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.5540417917378261, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    1.4s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.5551092067733453, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    1.6s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.5674596647916894, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    1.8s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ................ alpha=1, score=0.5541103003002259, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    1.9s remaining:    0.0s
```

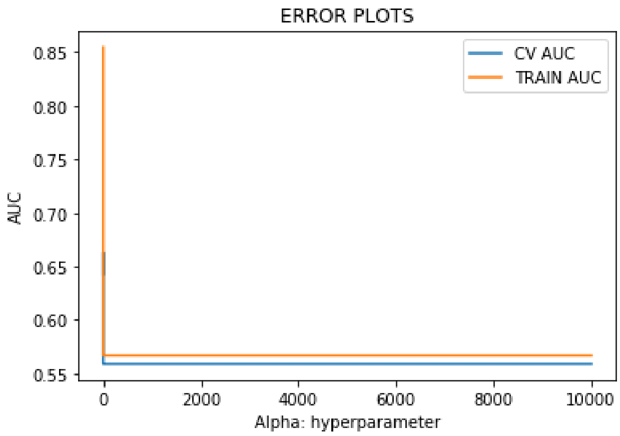
```
[CV] alpha=1 .........................................................
[CV] ................ alpha=1, score=0.5550678647136219, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    2.1s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ................ alpha=1, score=0.5673614084964134, total=   0.0s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.5541103003002259, total=   0.0s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.5550678647136219, total=   0.1s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.5673614084964134, total=   0.0s
[CV] alpha=100 .......................................................
[CV] .............. alpha=100, score=0.5541103003002259, total=   0.0s
[CV] alpha=100 .......................................................
[CV] .............. alpha=100, score=0.5550678647136219, total=   0.1s
[CV] alpha=100 .......................................................
[CV] .............. alpha=100, score=0.5673614084964134, total=   0.1s
[CV] alpha=1000 ......................................................
[CV] ............. alpha=1000, score=0.5541103003002259, total=   0.0s
[CV] alpha=1000 ......................................................
[CV] ............. alpha=1000, score=0.5550678647136219, total=   0.0s
[CV] alpha=1000 ......................................................
[CV] ............. alpha=1000, score=0.5673614084964134, total=   0.1s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.5541103003002259, total=   0.1s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.5550678647136219, total=   0.1s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.5673614084964134, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:    4.4s finished
```

```
Optimal alpha  {'alpha': 0.001}
AUC:  0.6633026128932902
```



```
Accuracy on crossvalidated data:  0.6714376985693273
```

Train Confusion matrix

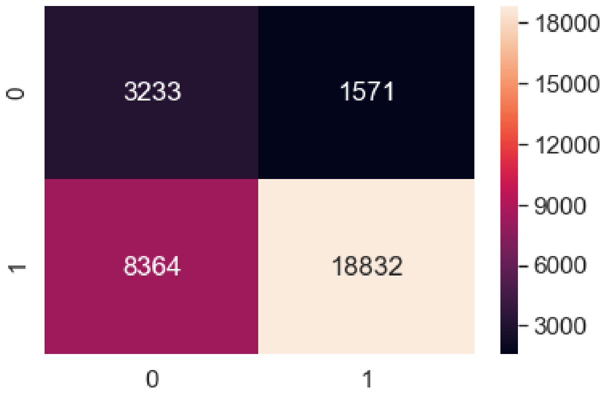
In [90]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_train, predict(x2, y_train, x2, y_train,0.001,'l2'),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4673222687031289 for threshold -0.041
train or test AUC = 0.7445750343912308
```

Out[90]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x66a79e10>
```



Test Confusion matrix

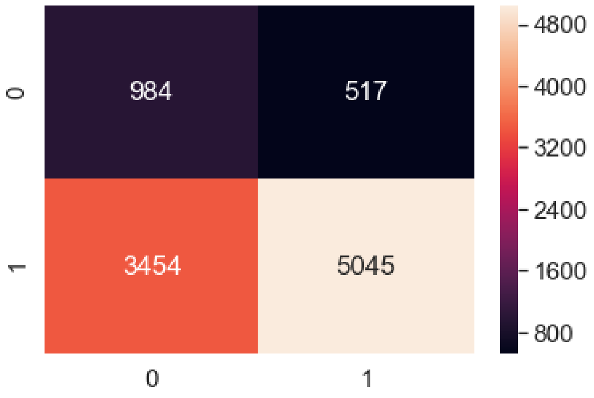
In [91]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict(x2, y_train, t2, y_test,0.001,'l2'),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.39946479575643146 for threshold -0.073
train or test AUC = 0.675082752612899
```

Out[91]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x636f9160>
```



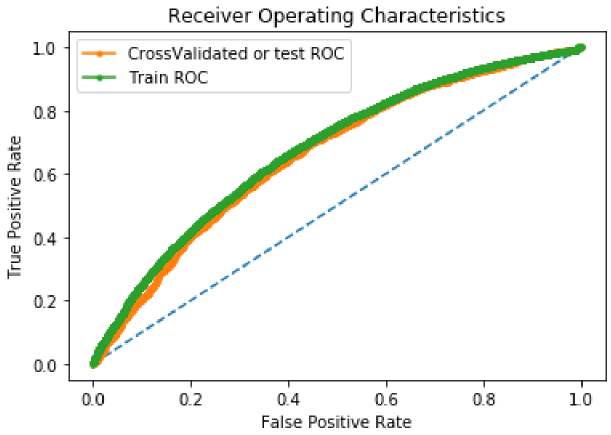
Encoding in TFIDF W2V

Encoding in TFIDF W2V

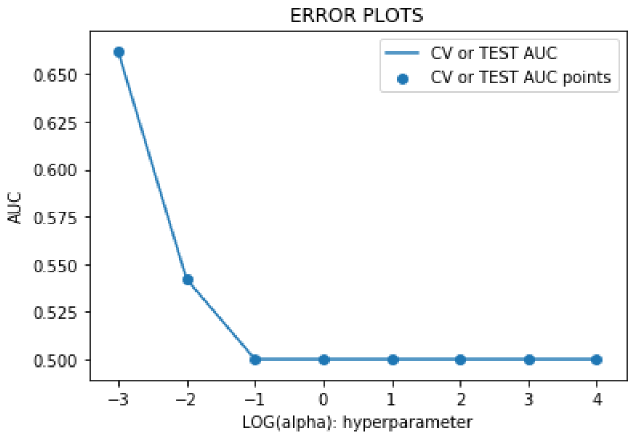
ROC SCORE on CV data

In [65]:

```
alpha_from_roc(x3,y_train,cv3,y_cv,'l1')
```

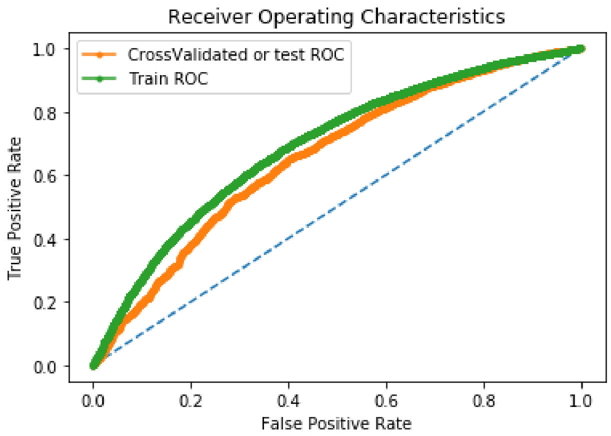


```
Optimal alpha  0.001
AUC:  0.6620581049840923
```

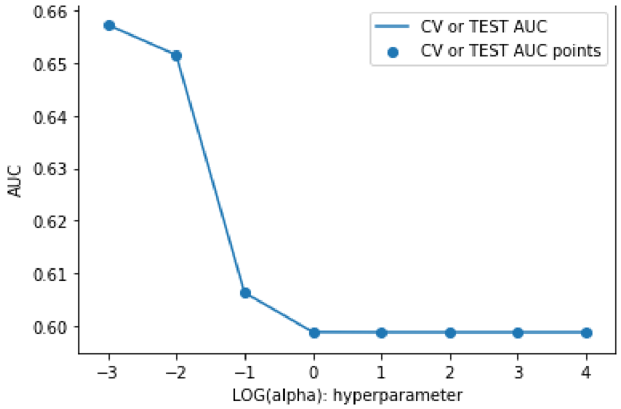


In [66]:

```
alpha_from_roc(x3,y_train,cv3,y_cv,'l2')
```



```
Optimal alpha  0.001
AUC:  0.6572629883000622
```



K FOLD Cross Validation

In [67]:

```
kfold_cv(x3,y_train,'l1')
```

```
Optimal alpha : 10
CV SCORE  0.8498750027332277
Optimal alpha : 100
CV SCORE  0.8498750027332277
Optimal alpha : 1000
CV SCORE  0.8498750027332277
Optimal alpha : 10000
CV SCORE  0.8498750027332277
```

In [68]:

```
kfold_cv(x3,y_train,'l2')
```

```
Optimal alpha : 1000
CV SCORE  0.8498750027332277
```

GridsearchCV

In [69]:

```
grid_search(x3,y_train,cv3,y_cv,'l1')
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
```

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

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6430594422976132, total=   0.9s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.0s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6485488075744166, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    1.9s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6503227264537161, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    2.8s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6479067501383253, total=   0.8s
```

```
[CV] ............ alpha=0.001, score=0.6479067501585255, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    3.8s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6555719966802326, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    4.8s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6489382497770113, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    5.7s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5587718087885399, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    6.7s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5521223807651933, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    7.6s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5729411395869859, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    8.6s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] ............................. alpha=0.1, score=0.5, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    9.5s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] ............................. alpha=0.1, score=0.5, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:   10.5s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] ............................. alpha=0.1, score=0.5, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:   11.5s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ............................... alpha=1, score=0.5, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:   12.4s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ............................... alpha=1, score=0.5, total=   0.8s
```

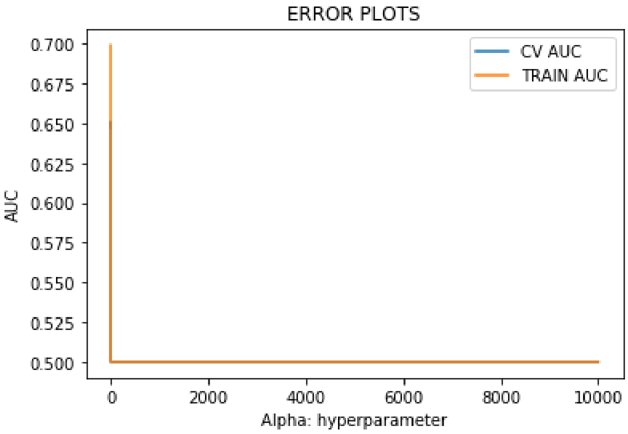
```
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:   13.4s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ............................... alpha=1, score=0.5, total=   0.9s
[CV] alpha=10 ........................................................
[CV] .............................. alpha=10, score=0.5, total=   0.9s
[CV] alpha=10 ........................................................
[CV] .............................. alpha=10, score=0.5, total=   0.9s
[CV] alpha=10 ........................................................
[CV] .............................. alpha=10, score=0.5, total=   0.9s
[CV] alpha=100 .......................................................
[CV] ............................. alpha=100, score=0.5, total=   0.9s
```

```
[CV] .............................. alpha=100, score=0.5, total=   0.8s
[CV] alpha=100 ...........................................................
[CV] .............................. alpha=100, score=0.5, total=   0.9s
[CV] alpha=100 ...........................................................
[CV] .............................. alpha=100, score=0.5, total=   0.8s
[CV] alpha=1000 ..........................................................
[CV] ............................. alpha=1000, score=0.5, total=   0.9s
[CV] alpha=1000 ..........................................................
[CV] ............................. alpha=1000, score=0.5, total=   0.8s
[CV] alpha=1000 ..........................................................
[CV] ............................. alpha=1000, score=0.5, total=   0.9s
[CV] alpha=10000 .........................................................
[CV] ............................ alpha=10000, score=0.5, total=   0.8s
[CV] alpha=10000 .........................................................
[CV] ............................ alpha=10000, score=0.5, total=   0.8s
[CV] alpha=10000 .........................................................
[CV] ............................ alpha=10000, score=0.5, total=   0.9s
```

```
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:   26.6s finished
```

```
Optimal alpha  {'alpha': 0.001}
AUC:  0.6508054843496442
```



```
Accuracy on crossvalidated data:  0.6651770433497897
```

In [70]:

```
grid_search(x3,y_train,cv3,y_cv,'l2')
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
```

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

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6130841577082254, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6145242924220349, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.7s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6173066819448545, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    1.0s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6424107109660244, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    1.4s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6340797757055453, total=   0.3s
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    1.9s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6566532982522989, total=   0.3s
```

```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    2.3s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.6399433010744071, total=   0.3s
```

```
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    2.8s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.6384606559675713, total=   0.3s
```

```
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    3.2s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.6575516612100889, total=   0.3s
```

```
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    3.7s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.5934306003443193, total=   0.3s
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    4.1s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.5991743301638903, total=   0.3s
```

```
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    4.5s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.6169158616736024, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    4.9s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ................ alpha=1, score=0.5848561512977518, total=   0.3s
```

```
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    5.4s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ................ alpha=1, score=0.5906717843543042, total=   0.3s
```

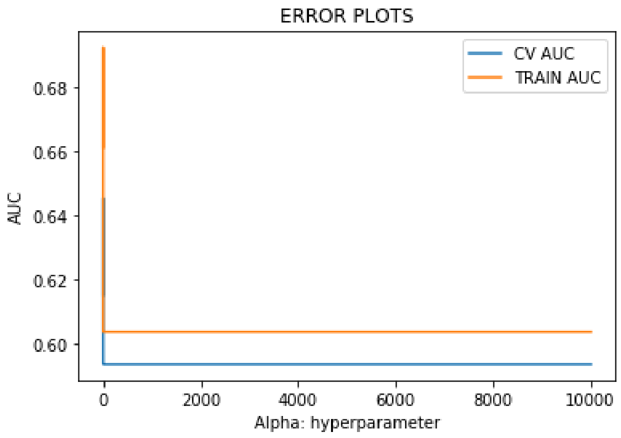
```
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    5.8s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ................ alpha=1, score=0.6048031204986679, total=   0.4s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.5848633808445377, total=   0.3s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.5905383184048304, total=   0.3s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.6047105831883204, total=   0.2s
[CV] alpha=100 .......................................................
[CV] .............. alpha=100, score=0.5848673743084767, total=   0.3s
```

```
[CV] alpha=100 .......................................................
[CV] ............... alpha=100, score=0.590551478960509, total=   0.2s
[CV] alpha=100 .......................................................
[CV] .............. alpha=100, score=0.6046847444009933, total=   0.3s
[CV] alpha=1000 ......................................................
[CV] ............. alpha=1000, score=0.5848651710180277, total=   0.3s
[CV] alpha=1000 ......................................................
[CV] ............. alpha=1000, score=0.5905513411536434, total=   0.3s
[CV] alpha=1000 ......................................................
[CV] ............. alpha=1000, score=0.6046863980833821, total=   0.3s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.5848609709956092, total=   0.3s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.5905500319884187, total=   0.3s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.6046897743515929, total=   0.3s
```

```
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:   11.5s finished
```

```
Optimal alpha  {'alpha': 0.01}
AUC:  0.6453182034649593
```



```
Accuracy on crossvalidated data:  0.6515239114729978
```

Train Confusion matrix

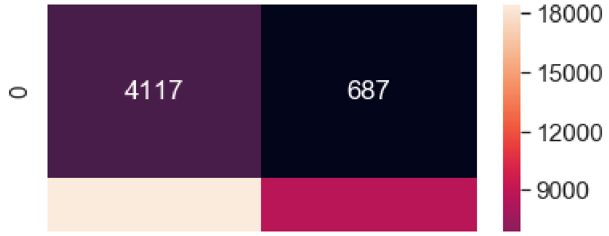
In [87]:

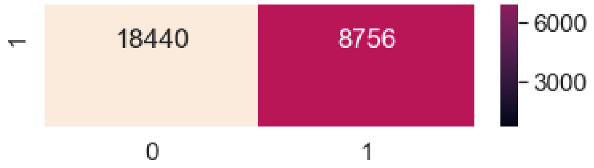
```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_train, predict(x3, y_train, x3, y_train,0.001,'l1'),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3977153880566509 for threshold -0.537
train or test AUC = 0.6783087001639438
```

Out[87]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x34a574e0>
```





Test Confusion matrix

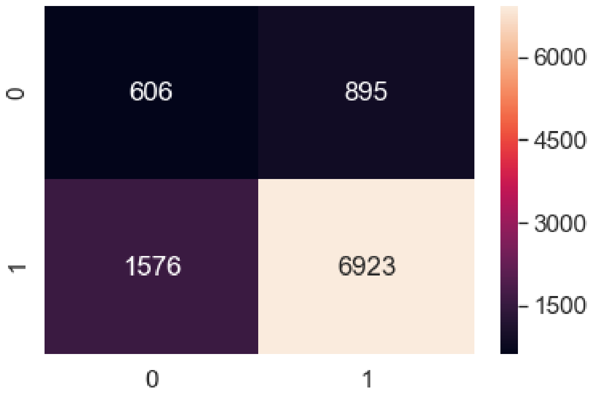
In [88]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict(x3, y_train, t3, y_test,0.001,'l1'),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.38545868036832176 for threshold 0.429
train or test AUC = 0.6668130177011067
```

Out[88]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x63e6d588>
```

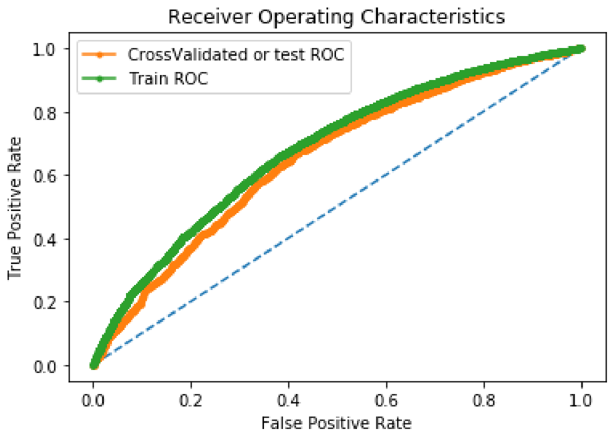


Encoding in AVG W2V

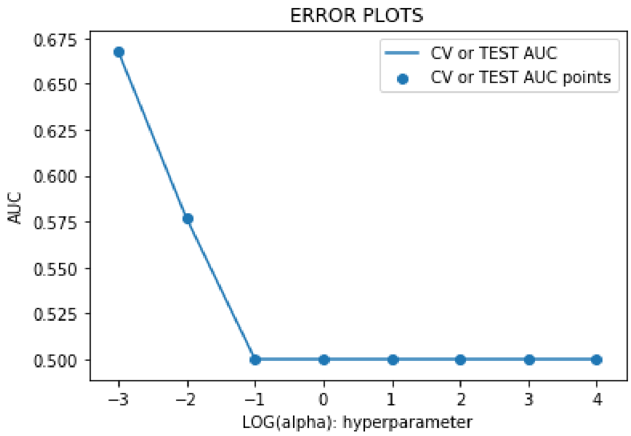
ROC SCORE on CV data

In [71]:

```
alpha_from_roc(x4,y_train,cv4,y_cv,'l1')
```

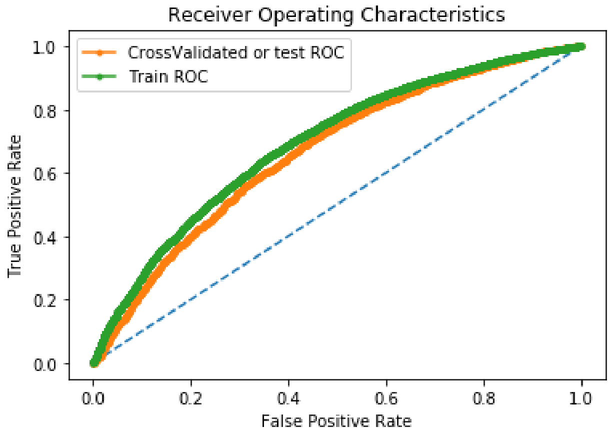


```
Optimal alpha  0.001
AUC:  0.6675836028685709
```

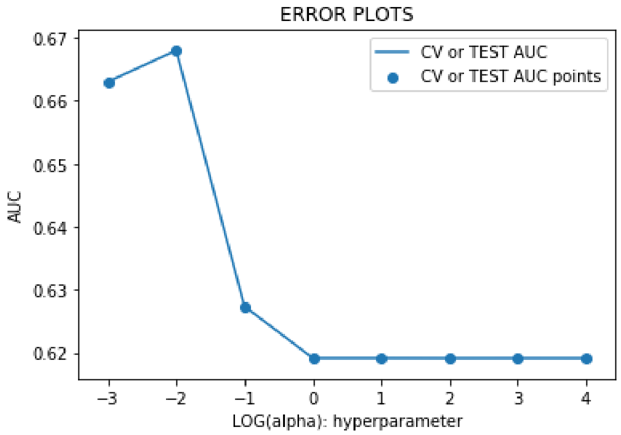


In [72]:

```
alpha_from_roc(x4,y_train,cv4,y_cv,'l2')
```



```
Optimal alpha  0.01
AUC:  0.6679669182873174
```



K FOLD Cross Validation

In [73]:

```
kfold_cv(x4,y_train,'l1')
```

```
Optimal alpha : 100
CV SCORE  0.8498750027332277
CV SCORE  0.8498750027332277
```

In [74]:

```
kfold_cv(x4,y_train,'l2')
```

```
Optimal alpha : 1000
CV SCORE  0.8498750027332277
Optimal alpha : 10000
CV SCORE  0.8498750027332277
```

GridsearchCV

In [75]:

```
grid_search(x4,y_train,cv4,y_cv,'l1')
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
```

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

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6327802661189287, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.8s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6355858669412698, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    1.8s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6428493912209448, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    2.7s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6568045664847025, total=   0.9s
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    3.8s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6702825350813215, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    4.8s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6642667141640998, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    5.7s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5699963687019287, total=   0.9s
```

```
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    6.8s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5803421262152413, total=   0.9s
```

```
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    7.8s remaining:    0.0s
```

```
[CV] alpha=0.01
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5833021487880058, total=   0.9s
```

```
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    8.8s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] ............................. alpha=0.1, score=0.5, total=   0.9s
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    9.9s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] ............................. alpha=0.1, score=0.5, total=   0.9s
```

```
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:   10.9s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] ............................. alpha=0.1, score=0.5, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:   11.9s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ............................... alpha=1, score=0.5, total=   0.9s
```

```
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:   13.0s remaining:    0.0s
```

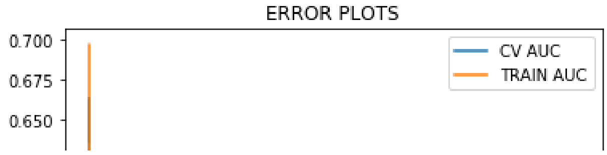
```
[CV] alpha=1 .........................................................
[CV] ............................... alpha=1, score=0.5, total=   0.8s
```

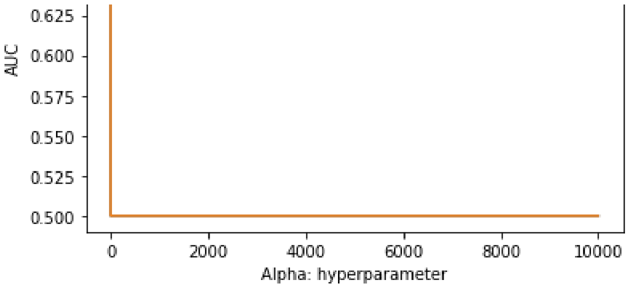
```
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:   13.9s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ............................... alpha=1, score=0.5, total=   0.8s
[CV] alpha=10 ........................................................
[CV] .............................. alpha=10, score=0.5, total=   0.8s
[CV] alpha=10 ........................................................
[CV] .............................. alpha=10, score=0.5, total=   0.8s
[CV] alpha=10 ........................................................
[CV] .............................. alpha=10, score=0.5, total=   0.8s
[CV] alpha=100 .......................................................
[CV] ............................. alpha=100, score=0.5, total=   0.8s
[CV] alpha=100 .......................................................
[CV] ............................. alpha=100, score=0.5, total=   0.8s
[CV] alpha=100 .......................................................
[CV] ............................. alpha=100, score=0.5, total=   0.8s
[CV] alpha=1000 ......................................................
[CV] ............................ alpha=1000, score=0.5, total=   0.9s
[CV] alpha=1000 ......................................................
[CV] ............................ alpha=1000, score=0.5, total=   0.9s
[CV] alpha=1000 ......................................................
[CV] ............................ alpha=1000, score=0.5, total=   0.9s
[CV] alpha=10000 .....................................................
[CV] ........................... alpha=10000, score=0.5, total=   1.0s
[CV] alpha=10000 .....................................................
[CV] ........................... alpha=10000, score=0.5, total=   0.9s
[CV] alpha=10000 .....................................................
[CV] ........................... alpha=10000, score=0.5, total=   0.8s
```

```
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:   27.1s finished
```

```
Optimal alpha  {'alpha': 0.001}
AUC:  0.6637841689909522
```





```
Accuracy on crossvalidated data:  0.664559452405145
```

In [76]:

```python
grid_search(x4,y_train,cv4,y_cv,'l2')
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
```

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

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6367511807571222, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.3s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6154602077507404, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.6s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6268068116555668, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    1.0s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6407136264976523, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    1.4s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6584219115672671, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    1.8s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6505392899432338, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    2.2s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.6528163697870493, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    2.6s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.6646921239586538, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    3.0s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.673480205594063, total=   0.2s
```

```
[CV] .............. alpha=0.01, score=0.675480205594065, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    3.4s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.6124792856271377, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    3.8s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] ............... alpha=0.1, score=0.621637055990585, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    4.2s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.6354850612189775, total=   0.3s
```

```
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    4.6s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ................ alpha=1, score=0.6013785575222677, total=   0.3s
```

```
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    5.0s remaining:    0.0s
```

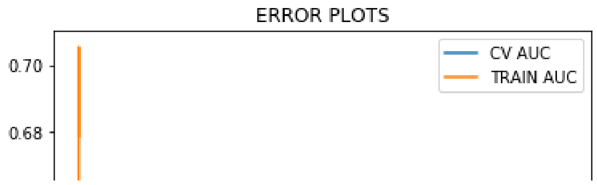
```
[CV] alpha=1 .........................................................
[CV] ................ alpha=1, score=0.6086971291040176, total=   0.3s
```

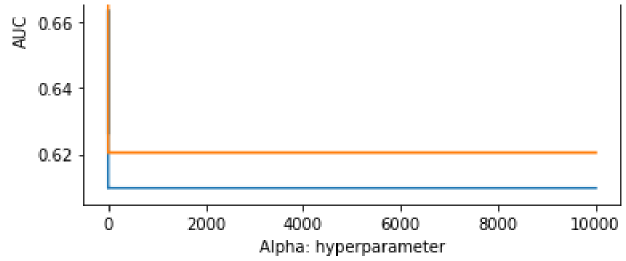
```
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    5.5s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ................. alpha=1, score=0.618998536835603, total=   0.2s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.6013320818643584, total=   0.3s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.6088089593755694, total=   0.2s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.6193836381219268, total=   0.3s
[CV] alpha=100 .......................................................
[CV] .............. alpha=100, score=0.6013231654233224, total=   0.3s
[CV] alpha=100 .......................................................
[CV] .............. alpha=100, score=0.6088159186222897, total=   0.3s
[CV] alpha=100 .......................................................
[CV] .............. alpha=100, score=0.6192878623502341, total=   0.3s
[CV] alpha=1000 ......................................................
[CV] .............. alpha=1000, score=0.601322476895057, total=   0.2s
[CV] ............. alpha=1000 ........................................
[CV] ............. alpha=1000, score=0.6088159186222897, total=   0.3s
[CV] alpha=1000 ......................................................
[CV] ............. alpha=1000, score=0.6192942014660583, total=   0.3s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.6013242670685469, total=   0.3s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.6088159186222897, total=   0.3s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.6192928233974009, total=   0.3s
```

```
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:   10.8s finished
```

```
Optimal alpha  {'alpha': 0.01}
AUC:  0.6636622218717975
```





```
Accuracy on crossvalidated data:  0.6688743838633271
```

Train Confusion matrix

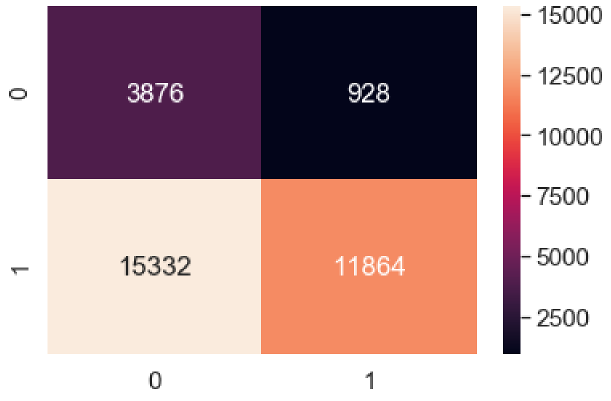
In [83]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_train, predict(x4, y_train, x4, y_train,0.01,'l2'),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4180209483100995 for threshold -0.42
train or test AUC = 0.6962105290744744
```

Out[83]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x323f8e48>
```



Test Confusion matrix

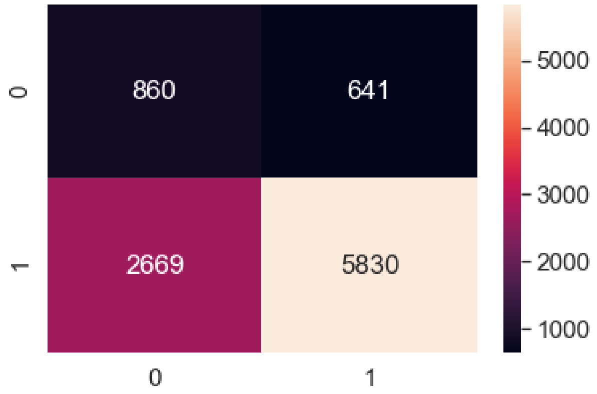
In [86]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict(x4, y_train, t4, y_test,0.01,'l2'),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.39957610720201514 for threshold 0.113
train or test AUC = 0.6786567122878978
```

Out[86]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x61387eb8>
```



2.5 Support Vector Machines with added Features `Set 5`

In [99]:

```
vectorizer = TfidfVectorizer(ngram_range=(1,2),min_df=10,max_features=5000)
vectorizer.fit(train_essay)
feature_tfidf=[]
text_tfidf=vectorizer.transform(train_essay)
text_tfidf1 = vectorizer.transform(test_essay)
text_tfidf2 = vectorizer.transform(cv_essay)
feature_tfidf.extend(vectorizer.get_feature_names())

print(text_tfidf.shape)
print(len(feature_tfidf))
```

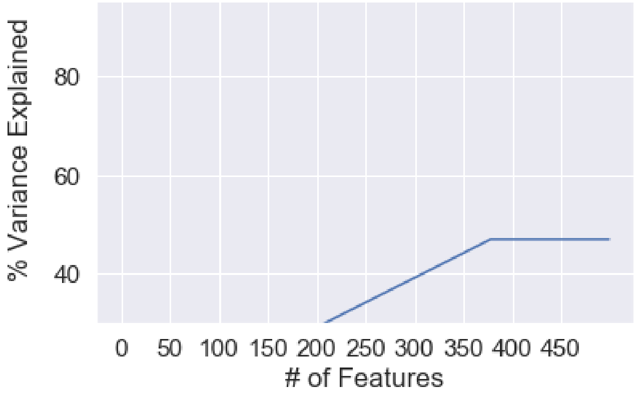
```
(32000, 5000)
5000
```

In [112]:

```
# 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.decomposition import TruncatedSVD


svd = TruncatedSVD(n_components=500)
svd.fit(text_tfidf)
variance = svd.explained_variance_ratio_ #calculate variance ratios

var=np.cumsum(np.round(svd.explained_variance_ratio_, decimals=3)*100)

# plot
plt.ylabel('% Variance Explained')
plt.xlabel('# of Features')
plt.title('SVD Analysis')
plt.ylim(30,100.5)
plt.style.context('seaborn-whitegrid')
plt.xticks(range(0,500,50))

plt.plot(var)
```

Out[112]:

```
[<matplotlib.lines.Line2D at 0x19094860>]
```



In [113]:

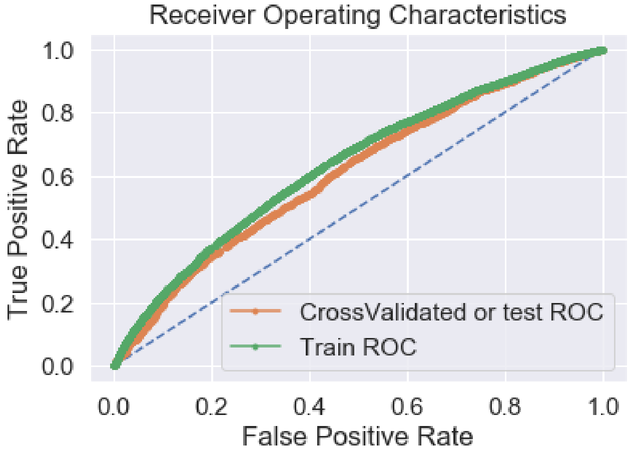
```
svd = TruncatedSVD(n_components=375)
svd.fit(text_tfidf)
svd.transform(text_tfidf)
svd.transform(text_tfidf1)
svd.transform(text_tfidf2)
x5 = hstack((x,text_tfidf ))
t5= hstack((t,text_tfidf1))
cv5 = hstack((cv,text_tfidf2))
print(x5.shape)
print(t5.shape)
print(cv5.shape)
```

```
(32000, 5102)
(10000, 5102)
(8000, 5102)
```

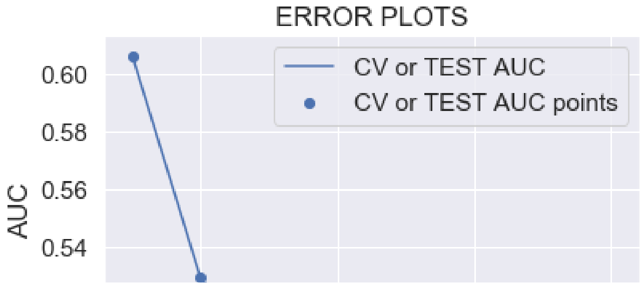
ROC SCORE on CV data

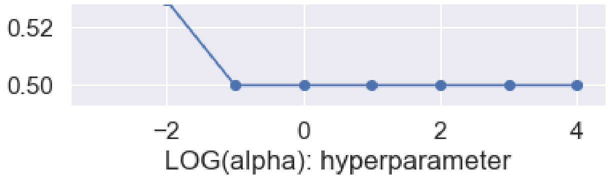
In [114]:

```
alpha_from_roc(x5,y_train,cv5,y_cv,'l1')
```



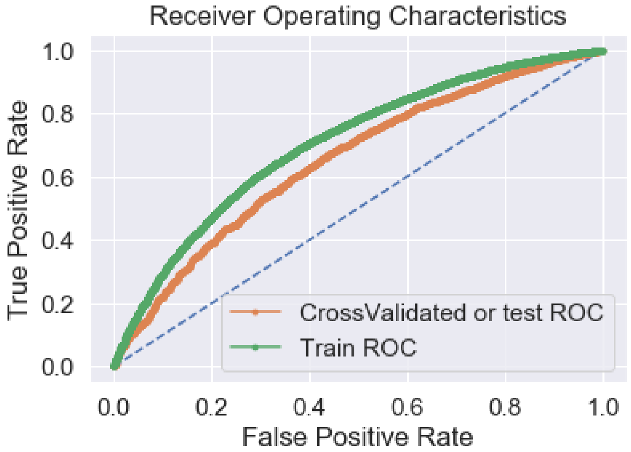
```
Optimal alpha  0.001
AUC:  0.6062228257841219
```



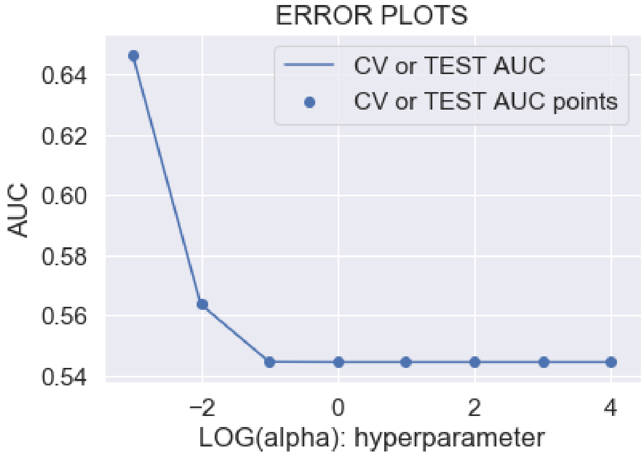


In [115]:

```
alpha_from_roc(x5,y_train,cv5,y_cv,'l2')
```



```
Optimal alpha  0.001
AUC:  0.6468958370353479
```



K FOLD Cross Validation

In [116]:

```
kfold_cv(x5,y_train,'l1')
```

```
Optimal alpha : 0.1
CV SCORE  0.8498750027332277
Optimal alpha : 10
CV SCORE  0.8498750027332277
Optimal alpha : 100
CV SCORE  0.8498750027332277
Optimal alpha : 10000
CV SCORE  0.8498750027332277
```

In [117]:

```
kfold_cv(x5,y_train,'l2')
```

```
Optimal alpha : 100
CV SCORE  0.8498750027332277
Optimal alpha : 1000
CV SCORE  0.8498750027332277
Optimal alpha : 10000
CV SCORE  0.8498750027332277
```

GridsearchCV

In [118]:

```
grid_search(x5,y_train,cv5,y_cv,'l1')
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
```

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

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6684690615332202, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6693181970865562, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.5s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6652051789198216, total=   0.3s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.9s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6174810992105886, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    1.2s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6151119698010035, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    1.5s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6263604552174196, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    1.7s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5359286786619306, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    2.0s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5237202479283322, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    2.2s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5452278688202665, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    2.5s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] ............................ alpha=0.1, score=0.5, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    2.8s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] ............................ alpha=0.1, score=0.5, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    3.0s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] ............................ alpha=0.1, score=0.5, total=   0.2s
```

```
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    3.3s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] .............................. alpha=1, score=0.5, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    3.6s remaining:    0.0s
```

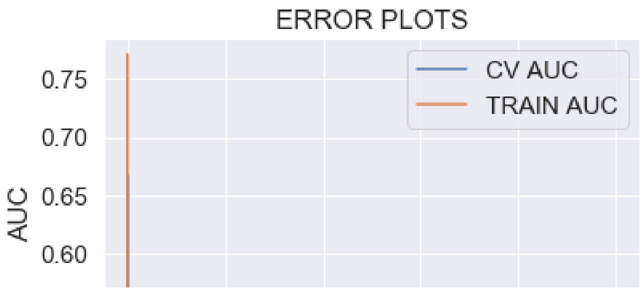
```
[CV] alpha=1 .........................................................
[CV] .............................. alpha=1, score=0.5, total=   0.2s
```

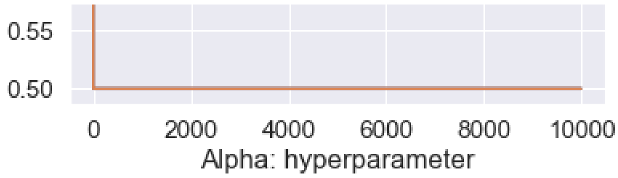
```
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    3.9s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] .............................. alpha=1, score=0.5, total=   0.1s
[CV] alpha=10 ........................................................
[CV] ............................. alpha=10, score=0.5, total=   0.1s
[CV] alpha=10 ........................................................
[CV] ............................. alpha=10, score=0.5, total=   0.1s
[CV] alpha=10 ........................................................
[CV] ............................. alpha=10, score=0.5, total=   0.1s
[CV] alpha=100 .......................................................
[CV] ............................ alpha=100, score=0.5, total=   0.1s
[CV] alpha=100 .......................................................
[CV] ............................ alpha=100, score=0.5, total=   0.1s
[CV] alpha=100 .......................................................
[CV] ............................ alpha=100, score=0.5, total=   0.1s
[CV] alpha=1000 ......................................................
[CV] ........................... alpha=1000, score=0.5, total=   0.1s
[CV] alpha=1000 ......................................................
[CV] ........................... alpha=1000, score=0.5, total=   0.2s
[CV] alpha=1000 ......................................................
[CV] ........................... alpha=1000, score=0.5, total=   0.1s
[CV] alpha=10000 .....................................................
[CV] .......................... alpha=10000, score=0.5, total=   0.1s
[CV] alpha=10000 .....................................................
[CV] .......................... alpha=10000, score=0.5, total=   0.1s
[CV] alpha=10000 .....................................................
[CV] .......................... alpha=10000, score=0.5, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:    6.9s finished
```

```
Optimal alpha  {'alpha': 0.0001}
AUC:  0.667664196153763
```





```
Accuracy on crossvalidated data:  0.6952925315093235
```

In [119]:

```
grid_search(x5,y_train,cv5,y_cv,'l2')
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
```

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

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6366300686352515, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ............... alpha=0.0001, score=0.656765231879, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.2s remaining:    0.0s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ........... alpha=0.0001, score=0.6533736326544393, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.4s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6532751017438218, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.5s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6724047263620745, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.7s remaining:    0.0s
```

```
[CV] alpha=0.001 .....................................................
[CV] ............ alpha=0.001, score=0.6616827665279525, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.9s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5757972881901153, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    1.1s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5816059874327029, total=   0.1s
```

```
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    1.3s remaining:    0.0s
```

```
[CV] alpha=0.01 ......................................................
[CV] ............. alpha=0.01, score=0.5912602196710344, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    1.4s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.5518835310373394, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    1.6s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.5527289032330525, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:    1.8s remaining:    0.0s
```

```
[CV] alpha=0.1 .......................................................
[CV] .............. alpha=0.1, score=0.5650155222208403, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    2.0s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ................ alpha=1, score=0.5518327176513584, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    2.1s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ................ alpha=1, score=0.5529378873449544, total=   0.0s
```

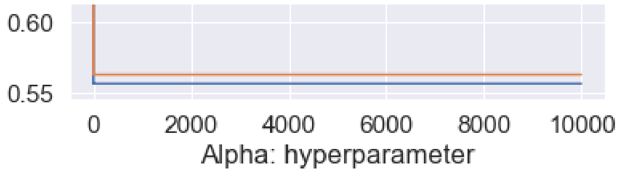
```
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    2.3s remaining:    0.0s
```

```
[CV] alpha=1 .........................................................
[CV] ................ alpha=1, score=0.5651298330159755, total=   0.0s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.5518327176513584, total=   0.0s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.5529462246603318, total=   0.0s
[CV] alpha=10 ........................................................
[CV] ............... alpha=10, score=0.5651298330159755, total=   0.0s
[CV] alpha=100 .......................................................
[CV] .............. alpha=100, score=0.5518327176513584, total=   0.0s
[CV] alpha=100 .......................................................
[CV] .............. alpha=100, score=0.5529462246603318, total=   0.0s
[CV] alpha=100 .......................................................
[CV] .............. alpha=100, score=0.5651298330159755, total=   0.0s
[CV] alpha=1000 ......................................................
[CV] ............. alpha=1000, score=0.5518327176513584, total=   0.0s
[CV] alpha=1000 ......................................................
[CV] ............. alpha=1000, score=0.5529462246603318, total=   0.0s
[CV] alpha=1000 ......................................................
[CV] ............. alpha=1000, score=0.5651298330159755, total=   0.0s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.5518327176513584, total=   0.1s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.5529462246603318, total=   0.0s
[CV] alpha=10000 .....................................................
[CV] ............ alpha=10000, score=0.5651298330159755, total=   0.0s
```

```
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:    4.5s finished
```

```
Optimal alpha  {'alpha': 0.001}
AUC:  0.6624536245177537
```

ERROR PLOTS
CV AUC
TRAIN AUC
0.75
0.70
0.65
AUC



```
Accuracy on crossvalidated data:  0.6450662100845266
```

Train Confusion matrix

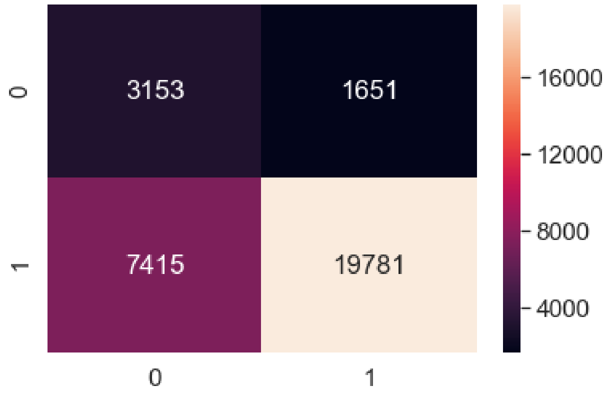
In [120]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_train, predict(x5, y_train, x5, y_train,0.0001,'l2'),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4838085056589235 for threshold 0.145
train or test AUC = 0.7630434935024363
```

Out[120]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1de4dc50>
```



Test Confusion matrix

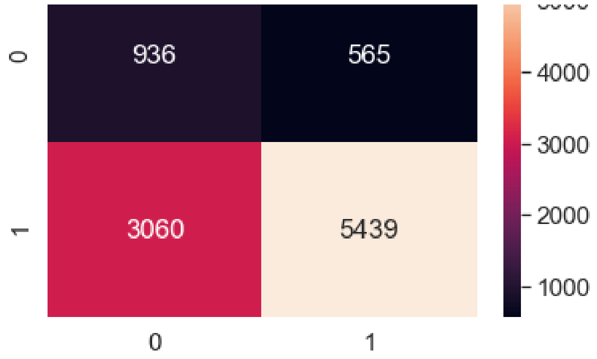
In [121]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
labels=[0,1]
print("Test confusion matrix")
cm=confusion_matrix(y_test, predict(x5, y_train, t5, y_test,0.0001,'l2'),labels)
sns.set(font_scale=1.4)
sns.heatmap(cm,fmt='d',annot=True)
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3998150348683103 for threshold 0.004
train or test AUC = 0.6782025302345794
```

Out[121]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x4e8c13c8>
```



3. Conclusion

In [1]:

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

from prettytable import PrettyTable
x=PrettyTable()
x.field_names=["Vectorizer","Model","Features","Hyperparameter","Test AUC"]
x.add_row(["BOW","Brute","All","0.01","0.693"])
x.add_row(["TFIDF","Brute","All","0.001","0.675"])
x.add_row(["W2V","Brute","All","0.001","0.666"])
x.add_row(["TFIDF W2V","Brute","All","0.01","0.678"])
x.add_row(["TFIDF with Truncated SVD","Brute","All","0.0001","0.678"])
print(x)
```

```
+--------------------------+-------+----------+----------------+----------+
|        Vectorizer        | Model | Features | Hyperparameter | Test AUC |
+--------------------------+-------+----------+----------------+----------+
|           BOW            | Brute |   All    |      0.01      |  0.693   |
|          TFIDF           | Brute |   All    |     0.001      |  0.675   |
|           W2V            | Brute |   All    |     0.001      |  0.666   |
|        TFIDF W2V         | Brute |   All    |      0.01      |  0.678   |
| TFIDF with Truncated SVD | Brute |   All    |     0.0001     |  0.678   |
+--------------------------+-------+----------+----------------+----------+
```

In [ ]: