DonorsChoose

In [1]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

import time
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
```

In [2]:

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

In [3]:

```
print(len(project_data))
```

```
109248
```

In [4]:

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


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

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

print(cols)
project_data.head(2)
```

```
['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state', 'Date',
'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']
```

Out[4]:

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

In [5]:

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

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

In [6]:

```
#project_data = project_data.sample(frac=0.5)
```

Preprocessing data

1.2 preprocessing of `project_subject_categories`

In [7]:

```
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]))
```

In [8]:

```
preprocessed_grade=project_data['project_grade_category']
```

In [9]:

```
new=[i.replace("-","_") for i in preprocessed_grade]
new=[i.replace(" ","_") for i in new]
```

In [10]:

```
project_data['preprocessed_grade']=new
```

In [11]:

```
print(project_data['preprocessed_grade'])
```

```
0          Grades_PreK_2
1             Grades_3_5
2          Grades_PreK_2
3          Grades_PreK_2
4             Grades_3_5
5             Grades_3_5
6             Grades_3_5
7             Grades_3_5
8          Grades_PreK_2
9             Grades_3_5
10         Grades_PreK_2
11           Grades_9_12
12         Grades_PreK_2
13            Grades_3_5
14            Grades_3_5
15         Grades_PreK_2
16            Grades_3_5
17            Grades_3_5
18            Grades_3_5
19           Grades_9_12
20         Grades_PreK_2
21            Grades_3_5
22           Grades_9_12
23            Grades_3_5
24            Grades_3_5
25         Grades_PreK_2
26         Grades_PreK_2
27         Grades_PreK_2
28            Grades_3_5
29         Grades_PreK_2
              ...
109218     Grades_PreK_2
109219     Grades_PreK_2
109220        Grades_3_5
109221     Grades_PreK_2
109222     Grades_PreK_2
109223       Grades_9_12
109224     Grades_PreK_2
109225        Grades_3_5
109226     Grades_PreK_2
109227        Grades_3_5
109228     Grades_PreK_2
109229     Grades_PreK_2
109230        Grades_3_5
```

```
109231       Grades_6_8
109232       Grades_3_5
109233    Grades_PreK_2
109234    Grades_PreK_2
109235       Grades_3_5
109236      Grades_9_12
109237    Grades_PreK_2
109238    Grades_PreK_2
109239       Grades_6_8
109240      Grades_9_12
109241       Grades_3_5
109242    Grades_PreK_2
109243      Grades_9_12
109244    Grades_PreK_2
109245       Grades_3_5
109246      Grades_9_12
109247    Grades_PreK_2
Name: preprocessed_grade, Length: 109248, dtype: object
```

In [12]:

```
print(project_data['clean_categories'].unique())
```

```
['Math_Science' 'SpecialNeeds' 'Literacy_Language' 'AppliedLearning'
 'Math_Science History_Civics' 'Literacy_Language Math_Science'
 'AppliedLearning Music_Arts' 'Math_Science AppliedLearning'
 'Math_Science Literacy_Language' 'History_Civics Literacy_Language'
 'AppliedLearning Health_Sports' 'Math_Science Music_Arts'
 'AppliedLearning Literacy_Language' 'Music_Arts' 'Health_Sports'
 'Literacy_Language SpecialNeeds' 'Math_Science SpecialNeeds'
 'AppliedLearning History_Civics' 'AppliedLearning SpecialNeeds'
 'Health_Sports Literacy_Language' 'Literacy_Language Music_Arts'
 'History_Civics Math_Science' 'SpecialNeeds Health_Sports'
 'Literacy_Language History_Civics' 'Health_Sports SpecialNeeds'
 'History_Civics Music_Arts' 'Math_Science Health_Sports'
 'Music_Arts SpecialNeeds' 'SpecialNeeds Music_Arts'
 'Health_Sports History_Civics' 'History_Civics'
 'Health_Sports AppliedLearning' 'History_Civics SpecialNeeds'
 'AppliedLearning Math_Science' 'Health_Sports Music_Arts'
 'Literacy_Language Health_Sports' 'Literacy_Language AppliedLearning'
 'Music_Arts Health_Sports' 'Music_Arts AppliedLearning'
 'Music_Arts History_Civics' 'Health_Sports Math_Science'
 'History_Civics AppliedLearning' 'History_Civics Health_Sports'
 'Health_Sports Warmth Care_Hunger' 'History_Civics Warmth Care_Hunger'
 'Math_Science Warmth Care_Hunger' 'SpecialNeeds Warmth Care_Hunger'
 'Warmth Care_Hunger' 'Literacy_Language Warmth Care_Hunger'
 'Music_Arts Warmth Care_Hunger' 'AppliedLearning Warmth Care_Hunger']
```

1.3 preprocessing of `project_subject_subcategories`

In [13]:

```
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]))
```

1.4 Preprocessing of project_grade_category

1.3 Text preprocessing

In [14]:

```
# 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 [15]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [16]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', '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 [17]:

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

```
100%|██████████████████████████████████████████████████████████████████████████████| 109248/109248
[1:02:23<00:00, 29.18it/s]
```

In [18]:

```
project_data['project_resource_summary']
preprocessed_resource_summary=[]
for sentence in tqdm(project_data['project_resource_summary'].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 not in stopwords)
    preprocessed_resource_summary.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 109248/109248
[20:29<00:00, 88.87it/s]
```

In [19]:

```
project_data['clean_resource_summary'] = preprocessed_resource_summary
```

1.4 Preprocessing of `project_title`

In [20]:

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

```
100%|██████████████████████████████████████████████████████████████████████████████| 109248/109248
[05:35<00:00, 325.88it/s]
```

In [21]:

```
#Adding processed columns at place of original columns
project_data['clean_essays'] = preprocessed_essays
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [22]:

```
project_data['clean_titles'] = preprocessed_titles
```

In [23]:

```
# we cannot remove rows where teacher prefix is not available therefore we are replacing 'nan' val
ue with
# 'null'(string)
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-split
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')
```

In [24]:

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

Out[24]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5

In [25]:

```
filtered_negative = project_data.loc[project_data['project_is_approved'] == 0]
#print(filtered_negative.count())

filtered_positive = project_data.loc[project_data['project_is_approved'] == 1]
sample_positive = filtered_positive.take(np.random.permutation(len(filtered_positive))[:50000])
```

In [26]:

```
project_data = pd.concat([filtered_negative, sample_positive]).sort_index(kind='merge')
```

In [27]:

```
project_data.count()
```

Out[27]:

```
Unnamed: 0                                      66542
id                                              66542
teacher_id                                      66542
teacher_prefix                                  66542
```

```
school_state                                    66542
Date                                            66542
project_grade_category                          66542
project_title                                   66542
project_resource_summary                        66542
teacher_number_of_previously_posted_projects    66542
project_is_approved                             66542
price                                           66542
quantity                                        66542
clean_categories                                66542
preprocessed_grade                              66542
clean_subcategories                             66542
essay                                           66542
clean_resource_summary                          66542
clean_essays                                    66542
clean_titles                                    66542
dtype: int64
```

So far we have preprocessed the data. Next is to split and vectorize data for BoW,TFIDF,Avg W2Vec and TFIDF weighted W2Vec

1.Splitting data

In [28]:

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

In [29]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

In [30]:

```
x = np.count_nonzero(y_test)
print(len(y_test) - x)
```

```
5459
```

In [31]:

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

print("="*100)
```

```
(29870, 19) (29870,)
(14713, 19) (14713,)
(21959, 19) (21959,)
====================================================================================================
```

2.Vectorizing data

BoW

2.1 Text data

In [32]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, max_features=5000)
vectorizer.fit(X_train['clean_essays'].values) # fit has to happen only on train data

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

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(29870, 5000) (29870,)
(14713, 5000) (14713,)
(21959, 5000) (21959,)
====================================================================================================
```

In [33]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, max_features=5000)
vectorizer.fit(X_train['clean_titles'].values) # fit has to happen only on train data

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

print("After vectorizations")
print(X_train_titles_bow.shape, y_train.shape)
print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(29870, 1505) (29870,)
(14713, 1505) (14713,)
(21959, 1505) (21959,)
====================================================================================================
```

In [34]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10, max_features=5000)
vectorizer.fit(X_train['clean_resource_summary'].values) # fit has to happen only on train data
#Bow_FeatureList=Bow_FeatureList + (vectorizer.get_feature_names())
# we use the fitted CountVectorizer to convert the text to vector
X_train_summary_bow = vectorizer.transform(X_train['clean_resource_summary'].values)
X_cv_summary_bow = vectorizer.transform(X_cv['clean_resource_summary'].values)
X_test_summary_bow = vectorizer.transform(X_test['clean_resource_summary'].values)

print("After vectorizations")
print(X_train_summary_bow.shape, y_train.shape)
print(X_cv_summary_bow.shape, y_cv.shape)
print(X_test_summary_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(29870, 2952) (29870,)
(14713, 2952) (14713,)
(21959, 2952) (21959,)
====================================================================================================
```

In [35]:

```
X_train_summary_bow.shape
```

Out[35]:

```
(29870, 2952)
```

2.2 one hot encoding the catogorical features: clean_categories

In [36]:

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


# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_cat_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_clean_cat_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_cat_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_cat_ohe.shape, y_train.shape)
print(X_cv_clean_cat_ohe.shape, y_cv.shape)
print(X_test_clean_cat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(29870, 9) (29870,)
(14713, 9) (14713,)
(21959, 9) (21959,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================================
```

2.3 one hot encoding the catogorical features: clean_subcategories

In [37]:

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


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

print("After vectorizations")
print(X_train_clean_subcat_ohe.shape, y_train.shape)
print(X_cv_clean_subcat_ohe.shape, y_cv.shape)
print(X_test_clean_subcat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(29870, 30) (29870,)
(14713, 30) (14713,)
(21959, 30) (21959,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================================
```

2.3 one hot encoding the catogorical features: teacher_prefix

In [38]:

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

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

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(29870, 6) (29870,)
(14713, 6) (14713,)
(21959, 6) (21959,)
['dr', 'mr', 'mrs', 'ms', 'null', 'teacher']
====================================================================================================
```

2.4 one hot encoding the catogorical features: school_state

In [39]:

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

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

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(29870, 51) (29870,)
(14713, 51) (14713,)
(21959, 51) (21959,)
['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']
====================================================================================================
```

2.4 one hot encoding the catogorical features: project_grade_category

In [40]:

```
X_train.head(2)
```

Out[40]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
						2016-	

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
39765	169616	p219272	b94a387c93dfcccec1a93c72931f2f3f	Ms.	FL	08-31 23:57:37	Grades PreK-2
68257	107753	p017066	b03ea1aee38240e5e38e9fe33876b7fa	Mrs.	OH	2016-11-17 15:27:34	Grades PreK-2

In [41]:

```
#This step is to intialize a vectorizer with vocab from train data
from collections import Counter
my_counter4 = Counter()
for word in X_train['preprocessed_grade'].values:
    my_counter4.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
project_grade_category_dict = dict(my_counter4)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lambda
kv: kv[1]))
```

In [42]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()), lowercase
=False, binary=True)
vectorizer.fit(X_train['preprocessed_grade'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['preprocessed_grade'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['preprocessed_grade'].values)
X_test_grade_ohe = vectorizer.transform(X_test['preprocessed_grade'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(29870, 4) (29870,)
(14713, 4) (14713,)
(21959, 4) (21959,)
['Grades_9_12', 'Grades_6_8', 'Grades_3_5', 'Grades_PreK_2']
====================================================================================================
```

2.5 Normalizing the numerical features: Price

In [43]:

```
X_train.head(2)
```

Out[43]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
39765	169616	p219272	b94a387c93dfcccec1a93c72931f2f3f	Ms.	FL	2016-08-31	Grades PreK-2

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	23:57:37 Date	project_grade_cate
68257	107753	p017066	b03ea1aee38240e5e38e9fe33876b7fa	Mrs.	OH	2016-11-17 15:27:34	Grades PreK-2

In [44]:

```
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
standard_vec.fit(X_train['price'].values.reshape(-1,1))

X_train_price_std = standard_vec.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_std = standard_vec.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_std = standard_vec.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_std.shape, y_train.shape)
print(X_cv_price_std.shape, y_cv.shape)
print(X_test_price_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(29870, 1) (29870,)
(14713, 1) (14713,)
(21959, 1) (21959,)
====================================================================================================
```

2.6 Vectorizing numerical features: teacher_number_of_previously _posted_projects"

In [45]:

```
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
standard_vec.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_projects_std =
standard_vec.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1
))
X_cv_projects_std = standard_vec.transform(X_cv['teacher_number_of_previously_posted_projects'].va
lues.reshape(-1,1))
X_test_projects_std = standard_vec.transform(X_test['teacher_number_of_previously_posted_projects'
].values.reshape(-1,1))

print("After vectorizations")
print(X_train_projects_std.shape, y_train.shape)
print(X_cv_projects_std.shape, y_cv.shape)
print(X_test_projects_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(29870, 1) (29870,)
(14713, 1) (14713,)
```

```
(21959, 1) (21959,)
====================================================================================================
```

In [46]:

```
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
standard_vec.fit(X_train['quantity'].values.reshape(-1,1))

X_train_qty_std = standard_vec.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_qty_std = standard_vec.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_qty_std = standard_vec.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_qty_std.shape, y_train.shape)
print(X_cv_qty_std.shape, y_cv.shape)
print(X_test_qty_std.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(29870, 1) (29870,)
(14713, 1) (14713,)
(21959, 1) (21959,)
====================================================================================================
```

In [47]:

```
#function to get heatmap confusion matrix for obtaining colur encoded matrix
#Reference link https://seaborn.pydata.org/generated/seaborn.heatmap.html
def get_confusion_matrix(clf,X_te,y_test):
    y_pred = clf.predict(X_te)
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

2.7 Concatinating all the features

In [48]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr =
hstack((X_train_essay_bow,X_train_titles_bow,X_train_summary_bow,X_train_clean_cat_ohe,X_train_clea
n_subcat_ohe, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
X_train_price_std,X_train_projects_std,X_train_qty_std)).tocsr()
X_cr =
hstack((X_cv_essay_bow,X_cv_titles_bow,X_cv_summary_bow,X_cv_clean_cat_ohe,X_cv_clean_subcat_ohe,
X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_std,X_cv_projects_std,X_cv_qty_std)).t
ocsr()
X_te =
hstack((X_test_essay_bow,X_test_titles_bow,X_test_summary_bow,X_test_clean_cat_ohe,X_test_clean_sub
cat_ohe, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
X_test_price_std,X_test_projects_std,X_test_qty_std)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(29870, 9560) (29870,)
```

```
(14713, 9560) (14713,)
(21959, 9560) (21959,)
==========================================================================================================
```

Applying Logistic Regression

In [49]:

```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import math

def predAUC(X_tr,y_train):
    Cs = [10**i for i in range(-5,4)]
    parameters = {'C':Cs}
    LoR = LogisticRegression(class_weight='balanced')
    clf = GridSearchCV(LoR, parameters, cv=3, scoring='roc_auc')
    clf.fit(X_tr, y_train)
    auc_tr = clf.cv_results_['mean_train_score']
    auc_tr_std = clf.cv_results_['std_train_score']
    auc_cv = clf.cv_results_['mean_test_score']
    auc_cv_std= clf.cv_results_['std_test_score']
    print(clf.best_estimator_)
    print(clf.score(X_te, y_test))
    return Cs, auc_tr, auc_cv

def plotPerformance(Cs, auc_tr, auc_cv, plt_title):
    #plt.plot(Cs, auc_tr, label='AUC_Train')
    #plt.plot(Cs, auc_cv, label='AUC_Validation')

    #plt.scatter(Cs, auc_tr, label='Coordinates')
    #plt.scatter(Cs, auc_cv, label='Coordinates')
    newmylist=[math.log10(i) for i in Cs]
    plt.plot(newmylist,list(auc_tr),color='blue', linestyle='dashed',
marker='o',markerfacecolor='red', markersize=10)
    plt.plot(newmylist,list(auc_cv),color='orange', linestyle='dashed',
marker='o',markerfacecolor='red', markersize=10)
    plt.xlabel('Hyperparameter - C')
    plt.ylabel('AUC')
    plt.title("AUC on various Cs using %s on text features"%plt_title)

    plt.legend()
```

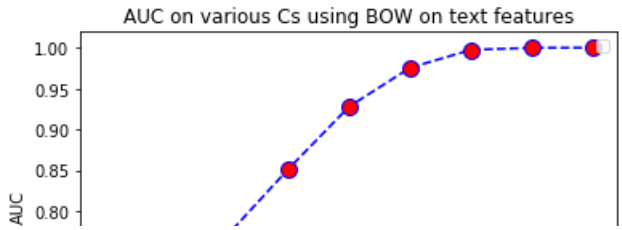
```
Wall time: 0 ns
```

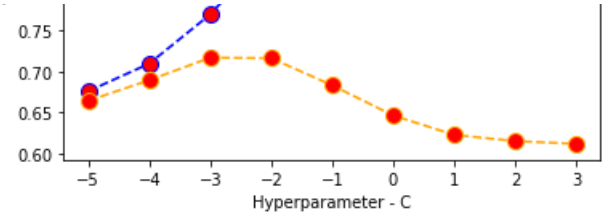
In [50]:

```
Cs, auc_tr, auc_cv=predAUC(X_tr,y_train)
plt_title1 = 'BOW'
plotPerformance(Cs, auc_tr, auc_cv, plt_title1)
```

```
LogisticRegression(C=0.001, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
0.7244445591655704
```

```
No handles with labels found to put in legend.
```





In [51]:

```
newmylist=[math.log10(i) for i in Cs]
print(Cs)
print(newmylist)
```

```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
[-5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0]
```

Logistic Regression with optimal alpha

In [52]:

```
def Get_LOR_OptimalC(optimal_c):
    LoR = LogisticRegression(C=optimal_c,class_weight = 'balanced')
    LoR.fit(X_tr, y_train)
    return LoR
```

In [53]:

```
def getROC_Data(data_pnts_test, y_test, data_pnts_tr, y_tr, LoR):
    predicted_y_test = LoR.predict_proba(data_pnts_test)[:, 1]
    predicted_y_tr = LoR.predict_proba(data_pnts_tr)[:, 1]

    fpr_test, tpr_test, thres_test = roc_curve(y_test, predicted_y_test)
    fpr_tr, tpr_tr, thres_tr = roc_curve(y_tr, predicted_y_tr)

    return [fpr_test, tpr_test, thres_test], [fpr_tr, tpr_tr, thres_tr]

def makeROC(test_data, train_data, plt_title):
    fpr_tr, tpr_tr, _ = train_data
    fpr_test, tpr_test, _ = test_data

    plt.plot(fpr_tr, tpr_tr, label='AUC_Train')
    plt.plot(fpr_test, tpr_test, label='AUC_Test')
    plt.title("ROC Curve using %s on text features"%plt_title)

    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.legend()
```

In [54]:

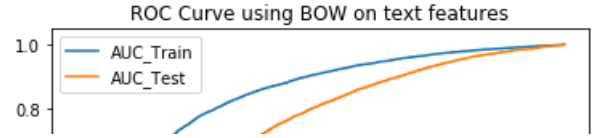
```
optimal_c=0.01
LOR=Get_LOR_OptimalC(optimal_c)
```

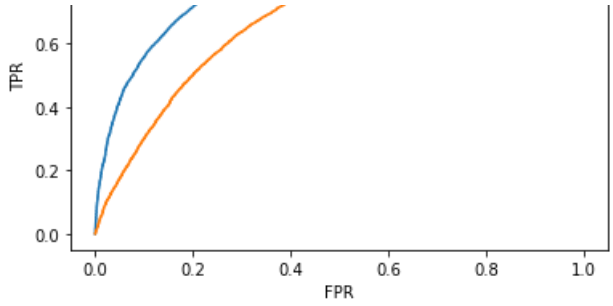
In [55]:

```
roc_data_test1, roc_data_train1 = getROC_Data(X_te, y_test,X_tr, y_train, LOR)
```

In [56]:

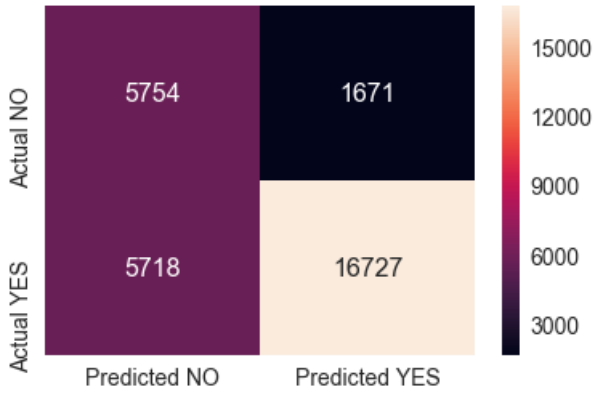
```
makeROC(roc_data_test1, roc_data_train1, plt_title1)
```





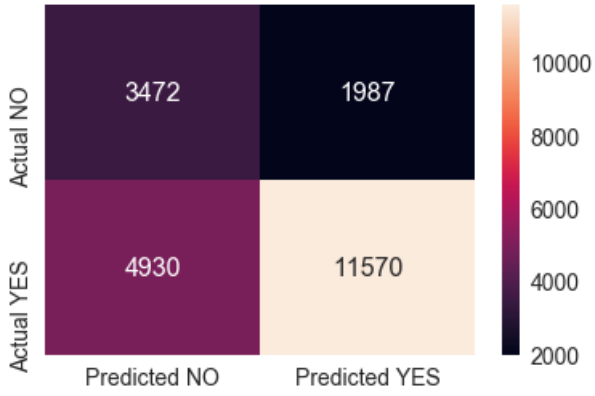
In [57]:

```
get_confusion_matrix(LOR,X_tr,y_train)
```



In [58]:

```
get_confusion_matrix(LOR,X_te,y_test)
```



In [59]:

```
predbow = (LOR.predict_proba(X_te))
predbow_train=(LOR.predict_proba(X_tr))
```

In [61]:

```
print("AUC score for Logistic regression  with Bag of Words for Testing data is ",round(metrics.ro
c_auc_score(y_test ,predbow[:,1]),3))
print("AUC score for Logistic regression  Bag of Words for Training data ",round(metrics.roc_auc_s
core(y_train ,predbow_train[:,1]),3))
```

```
AUC score for Logistic regression  with Bag of Words for Testing data is  0.725
AUC score for Logistic regression  Bag of Words for Training data  0.836
```

2.4.2 Applying Logistic Regression on TFIDF, SET 2

In [62]:

```
%%time
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['clean_essays'].values) # fit has to happen only on train data

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

```
Wall time: 19.4 s
```

In [63]:

```
from sklearn.feature_extraction.text import TfidfVectorizer


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

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_tfidf = vectorizer.transform(X_train['clean_titles'].values)
X_cv_titles_tfidf = vectorizer.transform(X_cv['clean_titles'].values)
X_test_titles_tfidf = vectorizer.transform(X_test['clean_titles'].values)
print("Train shape:",X_train_titles_tfidf.shape)
print("CV shape:",X_cv_titles_tfidf.shape)
print("Test shape:",X_test_titles_tfidf.shape)
```

```
Train shape: (29870, 2443)
CV shape: (14713, 2443)
Test shape: (21959, 2443)
```

In [64]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_summary_tfidf = vectorizer.transform(X_train['project_resource_summary'].values)
X_cv_summary_tfidf = vectorizer.transform(X_cv['project_resource_summary'].values)
X_test_summary_tfidf = vectorizer.transform(X_test['project_resource_summary'].values)

print("After vectorizations")
print(X_train_summary_tfidf.shape, y_train.shape)
print(X_cv_summary_tfidf.shape, y_cv.shape)
print(X_test_summary_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(29870, 4618) (29870,)
(14713, 4618) (14713,)
(21959, 4618) (21959,)
====================================================================================================
```

Concatinating all features (TFIDF)

In [65]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr =
hstack((X_train_essay_tfidf,X_train_titles_tfidf,X_train_summary_tfidf,X_train_clean_cat_ohe,X_trai
n_clean_subcat_ohe, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_std,X
_train_projects_std,X_train_qty_std)).tocsr()
X_cr =
hstack((X_cv_essay_tfidf,X_cv_titles_tfidf,X_cv_summary_tfidf,X_cv_clean_cat_ohe,X_cv_clean_subcat_
```

```
ohe, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_std,X_cv_projects_std,X_cv_qty_std)).tocsr()
X_te = hstack((X_test_essay_tfidf,X_test_titles_tfidf,X_test_summary_tfidf,X_test_clean_cat_ohe,X_test_clean_subcat_ohe, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_std,X_test_projects_std,X_test_qty_std)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(29870, 16989) (29870,)
(14713, 16989) (14713,)
(21959, 16989) (21959,)
====================================================================================================
```

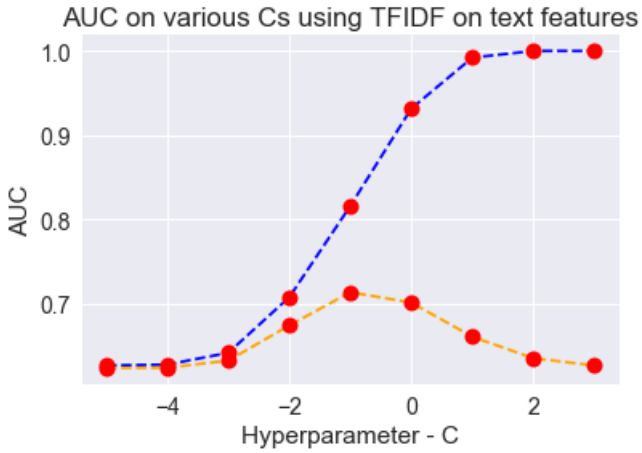
APPLYING Logistic regression ON TFIDF

In [66]:

```
Cs, auc_tr, auc_cv=predAUC(X_tr,y_train)
plt_title1 = 'TFIDF'
plotPerformance(Cs, auc_tr, auc_cv, plt_title1)
```

```
LogisticRegression(C=0.1, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
0.723496078202801
```

```
No handles with labels found to put in legend.
```



In [67]:

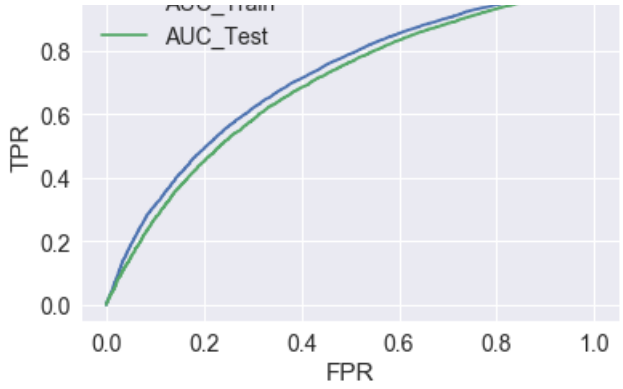
```
optimal_c=0.01
LOR=Get_LOR_OptimalC(optimal_c)
```

In [68]:

```
roc_data_test1, roc_data_train1 = getROC_Data(X_te, y_test,X_tr, y_train, LOR)
```

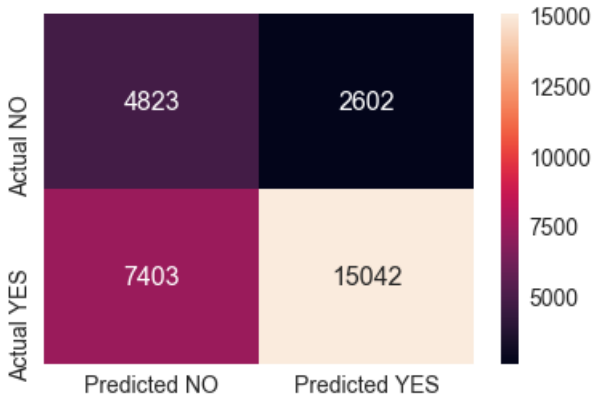
In [69]:

```
makeROC(roc_data_test1, roc_data_train1, plt_title1)
```



In [70]:

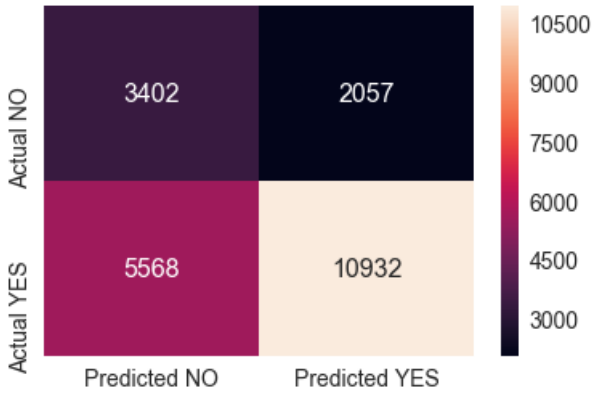
```
get_confusion_matrix(LOR,X_tr,y_train)
```



In [71]:

```
%%time
get_confusion_matrix(LOR,X_te,y_test)
```

```
Wall time: 140 ms
```



In [72]:

```
predbow = (LOR.predict_proba(X_te))
predbow_train=(LOR.predict_proba(X_tr))
```

In [73]:

```
print("AUC score for Logistic regression  with TFIDF for Testing data is
",round(metrics.roc_auc_score(y_test ,predbow[:,1]),3))
print("AUC score for Logistic regression with TFIDF for Training data
",round(metrics.roc_auc_score(y_train ,predbow_train[:,1]),3))
```

```
AUC score for Logistic regression  with TFIDF for Testing data is  0.69
```

```
AUC score for Logistic regression with TFIDF for Training data  0.714
```

Avg W2V

In [74]:

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

In [75]:

```
# average Word2Vec
# compute average word2vec for each review.
train_w2v_vectors_essays = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train['clean_essays'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_w2v_vectors_essays.append(vector)
print("train  vector")
print(len(train_w2v_vectors_essays))
print(len(train_w2v_vectors_essays[0]))
print('='*50)

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

print("Test vec")
print(len(test_w2v_vectors_essays))
print(len(test_w2v_vectors_essays[0]))
print('='*50)

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

print("CV vec")
print(len(cv_w2v_vectors_essays))
print(len(cv_w2v_vectors_essays[0]))
print('='*50)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 29870/29870
[00:14<00:00, 2090.73it/s]
```

```
train  vector
```

```
29870
50
==================================================

100%|██████████████████████████████████████████████████████████████████████████| 21959/21959
[00:11<00:00, 1915.11it/s]

Test vec
21959
50
==================================================

100%|██████████████████████████████████████████████████████████████████████████| 14713/14713
[00:07<00:00, 1888.60it/s]

CV vec
14713
50
==================================================
```

In [76]:

```
# Changing list to numpy arrays
train_w2v_vectors_essays = np.array(train_w2v_vectors_essays)
test_w2v_vectors_essays = np.array(test_w2v_vectors_essays)
cv_w2v_vectors_essays = np.array(cv_w2v_vectors_essays)
```

In [77]:

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

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

print("Test vec")
print(len(test_w2v_vectors_titles))
print(len(test_w2v_vectors_titles[0]))
print('='*50)

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

```
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_w2v_vectors_titles.append(vector)

print("CV vec")
print(len(cv_w2v_vectors_titles))
print(len(cv_w2v_vectors_titles[0]))
print('='*50)
```

```
100%|████████████████████████████████████████████████████████████████| 29870/29870
[00:00<00:00, 41829.58it/s]
```

```
train  vector
29870
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████████| 21959/21959
[00:00<00:00, 34189.12it/s]
```

```
Test vec
21959
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████████| 14713/14713
[00:00<00:00, 28589.76it/s]
```

```
CV vec
14713
50
==================================================
```

In [78]:

```
# Changing list to numpy arrays
train_w2v_vectors_titles = np.array(train_w2v_vectors_titles)
test_w2v_vectors_titles = np.array(test_w2v_vectors_titles)
cv_w2v_vectors_titles = np.array(cv_w2v_vectors_titles)
```

In [79]:

```
# average Word2Vec
# compute average word2vec for each review.
train_w2v_vectors_summary = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train['project_resource_summary'].values): # for each essay in training
data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_w2v_vectors_summary.append(vector)
print("train  vector")
print(len(train_w2v_vectors_summary))
print(len(train_w2v_vectors_summary[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
test_w2v_vectors_summary = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_test['project_resource_summary'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
```

```
            if word in glove_words:
                vector += model[word][:50]
                cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_w2v_vectors_summary.append(vector)

print("Test vec")
print(len(test_w2v_vectors_summary))
print(len(test_w2v_vectors_summary[0]))
print('='*50)

# average Word2Vec
# compute average word2vec for each review.
cv_w2v_vectors_summary = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_cv['project_resource_summary'].values): # for each essay in training data
    vector = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if word in glove_words:
            vector += model[word][:50]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_w2v_vectors_summary.append(vector)

print("CV vec")
print(len(cv_w2v_vectors_summary))
print(len(cv_w2v_vectors_summary[0]))
print('='*50)
```

```
100%|████████████████████████████████████████████████████████████████| 29870/29870
[00:02<00:00, 13227.38it/s]
```

```
train  vector
29870
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████████| 21959/21959
[00:01<00:00, 11110.13it/s]
```

```
Test vec
21959
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████████| 14713/14713
[00:01<00:00, 10963.98it/s]
```

```
CV vec
14713
50
==================================================
```

In [80]:

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

In [81]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr =
hstack((train_w2v_vectors_essays,train_w2v_vectors_titles,train_w2v_vectors_summary,X_train_clean_c
at_ohe,X_train_clean_subcat_ohe, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
X_train_price_std,X_train_projects_std,X_train_qty_std)).tocsr()
X_cr =
hstack((cv_w2v_vectors_essays,cv_w2v_vectors_titles,cv_w2v_vectors_summary,X_cv_clean_cat_ohe,X_cv
```

```
_clean_subcat_ohe, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_std,X_cv_projects_s
td,X_cv_qty_std)).tocsr()
X_te =
hstack((test_w2v_vectors_essays,test_w2v_vectors_titles,test_w2v_vectors_summary,X_test_clean_cat_o
he,X_test_clean_subcat_ohe, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
X_test_price_std,X_test_projects_std,X_test_qty_std)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

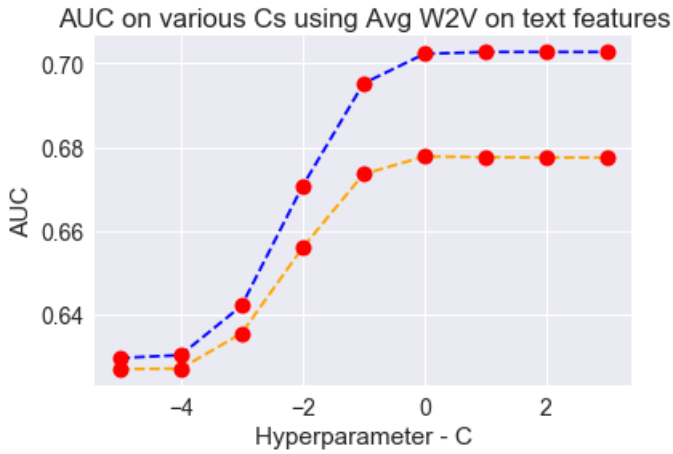
```
Final Data matrix
(29870, 253) (29870,)
(14713, 253) (14713,)
(21959, 253) (21959,)
====================================================================================================
```

In [82]:

```
Cs, auc_tr, auc_cv=predAUC(X_tr,y_train)
plt_title1 = 'Avg W2V'
plotPerformance(Cs, auc_tr, auc_cv, plt_title1)
```

```
LogisticRegression(C=1, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
0.6805203639250168
```

```
No handles with labels found to put in legend.
```



In [83]:

```
optimal_c=0.1
LOR=Get_LOR_OptimalC(optimal_c)
```

In [84]:

```
roc_data_test1, roc_data_train1 = getROC_Data(X_te, y_test,X_tr, y_train, LOR)
```

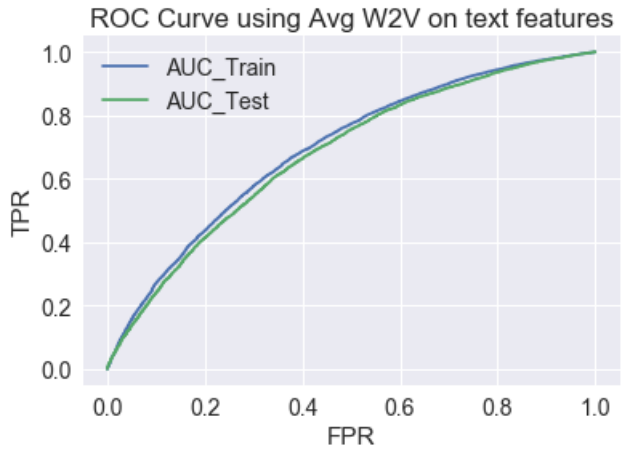
In [85]:

```
print(auc_tr,auc_cv)
```

```
[0.62968315 0.63037755 0.64213868 0.67067409 0.69516801 0.70220467
 0.70267509 0.70266767 0.70266603] [0.62705544 0.62716149 0.63557449 0.65595911 0.67352891
0.67773913
 0.6775784  0.67747909 0.67746424]
```

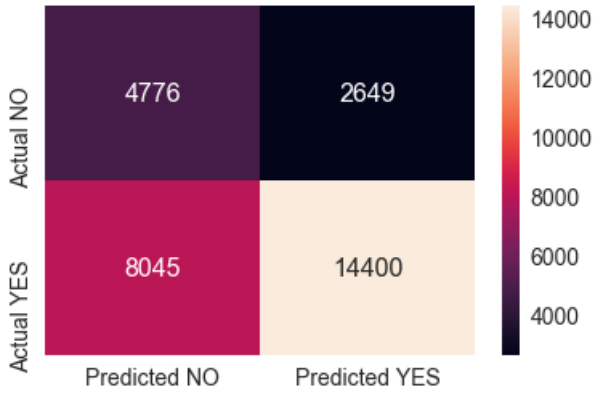
In [86]:

```
makeROC(roc_data_test1, roc_data_train1, plt_title1)
```



In [87]:

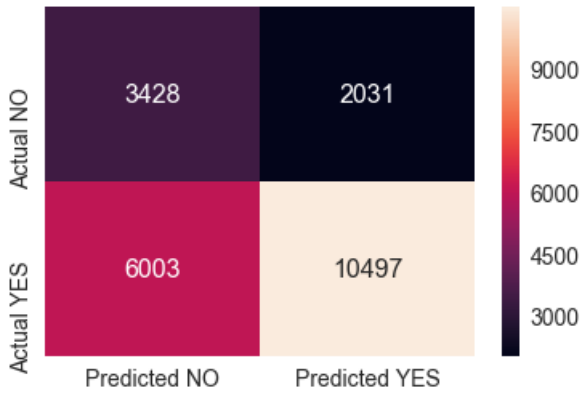
```
get_confusion_matrix(LOR,X_tr,y_train)
```



In [88]:

```
%%time
get_confusion_matrix(LOR,X_te,y_test)
```

```
Wall time: 181 ms
```



In [89]:

```
predbow = (LOR.predict_proba(X_te))
predbow_train=(LOR.predict_proba(X_tr))
```

In [90]:

In [90]:

```
print("AUC score for Logistic regression  AVG W2V for Testing data is
",round(metrics.roc_auc_score(y_test ,predbow[:,1]),3))
print("AUC score for Logistic regression AVG W2V for Training data ",round(metrics.roc_auc_score(y
_train ,predbow_train[:,1]),3))
```

```
AUC score for Logistic regression  AVG W2V for Testing data is  0.678
AUC score for Logistic regression AVG W2V for Training data  0.694
```

TFIDF W2V

In [91]:

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


# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_essays'].values): # for each review/sentence
    vector = np.zeros(50) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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
    train_tfidf_w2v_essays.append(vector)

print("Train matrix:")
print(len(train_tfidf_w2v_essays))
print(len(train_tfidf_w2v_essays[0]))
print('='*50)

cv_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_essays'].values): # for each review/sentence
    vector = np.zeros(50) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_tfidf_w2v_essays.append(vector)

print("CV matrix:")
print(len(cv_tfidf_w2v_essays))
print(len(cv_tfidf_w2v_essays[0]))
print('='*50)

test_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_essays'].values): # for each review/sentence
    vector = np.zeros(50) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word][:50] # getting the vector for each word
```

```
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_essays.append(vector)

print("Test matrix:")
print(len(test_tfidf_w2v_essays))
print(len(test_tfidf_w2v_essays[0]))
print('='*50)
```

```
100%|████████████████████████████████████████████████████████████| 29870/29870 [02:
22<00:00, 209.68it/s]
```

```
Train matrix:
29870
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████| 14713/14713 [01:
08<00:00, 214.90it/s]
```

```
CV matrix:
14713
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████| 21959/21959 [01:
46<00:00, 206.74it/s]
```

```
Test matrix:
21959
50
==================================================
```

In [92]:

```
# Changing list to numpy arrays
train_tfidf_w2v_essays = np.array(train_tfidf_w2v_essays)
test_tfidf_w2v_essays = np.array(test_tfidf_w2v_essays)
cv_tfidf_w2v_essays = np.array(cv_tfidf_w2v_essays)
```

In [93]:

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


# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles'].values): # for each review/sentence
    vector = np.zeros(50) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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
```

```
        vector /= tf_idf_weight
    train_tfidf_w2v_titles.append(vector)

print("Train matrix:")
print(len(train_tfidf_w2v_titles))
print(len(train_tfidf_w2v_titles[0]))
print('='*50)

cv_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles'].values): # for each review/sentence
    vector = np.zeros(50) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_tfidf_w2v_titles.append(vector)

print("CV matrix:")
print(len(cv_tfidf_w2v_titles))
print(len(cv_tfidf_w2v_titles[0]))
print('='*50)

test_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles'].values): # for each review/sentence
    vector = np.zeros(50) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_titles.append(vector)

print("Test matrix:")
print(len(test_tfidf_w2v_titles))
print(len(test_tfidf_w2v_titles[0]))
print('='*50)
```

```
100%|████████████████████████████████████████████████████████████████| 29870/29870
[00:02<00:00, 12704.98it/s]
```

```
Train matrix:
29870
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████████| 14713/14713
[00:01<00:00, 12599.60it/s]
```

```
CV matrix:
14713
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████████| 21959/21959
[00:01<00:00, 13612.35it/s]
```

```
Test matrix:
21959
50
```

==

In [94]:

```
# Changing list to numpy arrays
train_tfidf_w2v_titles = np.array(train_tfidf_w2v_titles)
test_tfidf_w2v_titles = np.array(test_tfidf_w2v_titles)
cv_tfidf_w2v_titles = np.array(cv_tfidf_w2v_titles)
```

In [95]:

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


# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_summary = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_resource_summary'].values): # for each review/sentence
    vector = np.zeros(50) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf 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
    train_tfidf_w2v_summary.append(vector)

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

cv_tfidf_w2v_summary = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_resource_summary'].values): # for each review/sentence
    vector = np.zeros(50) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_tfidf_w2v_summary.append(vector)

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

test_tfidf_w2v_summary = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_resource_summary'].values): # for each review/sentence
    vector = np.zeros(50) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word][:50] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
```

```
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_summary.append(vector)

print("Test matrix:")
print(len(test_tfidf_w2v_summary))
print(len(test_tfidf_w2v_summary[0]))
print('='*50)
```

```
100%|████████████████████████████████████████████████████████████████| 29870/29870
[00:07<00:00, 3834.02it/s]
```

```
Train matrix:
29870
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████████| 14713/14713
[00:04<00:00, 3375.79it/s]
```

```
CV matrix:
14713
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████████| 21959/21959
[00:06<00:00, 3164.53it/s]
```

```
Test matrix:
21959
50
==================================================
```

In [96]:

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

In [97]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr =
hstack((train_tfidf_w2v_essays,train_tfidf_w2v_titles,train_tfidf_w2v_summary,X_train_clean_cat_ohe
,X_train_clean_subcat_ohe, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
X_train_price_std,X_train_projects_std,X_train_qty_std)).tocsr()
X_cr =
hstack((cv_tfidf_w2v_essays,cv_tfidf_w2v_titles,cv_tfidf_w2v_summary,X_cv_clean_cat_ohe,X_cv_clean_
subcat_ohe, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_std,X_cv_projects_std,X_cv
_qty_std)).tocsr()
X_te =
hstack((test_tfidf_w2v_essays,test_tfidf_w2v_titles,test_tfidf_w2v_summary,X_test_clean_cat_ohe,X_
test_clean_subcat_ohe, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_std,X_t
est_projects_std,X_test_qty_std)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

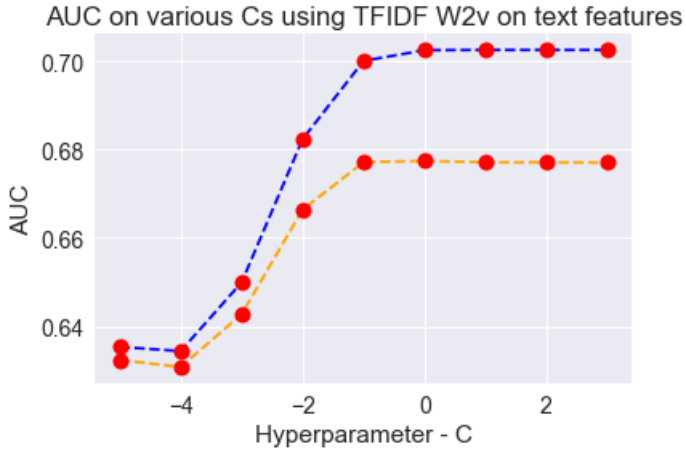
```
Final Data matrix
(29870, 253) (29870,)
(14713, 253) (14713,)
(21959, 253) (21959,)
====================================================================================================
```

In [98]:

```
Cs, auc_tr, auc_cv=predAUC(X_tr,y_train)
plt_title1 = 'TFIDF W2v'
plotPerformance(Cs, auc_tr, auc_cv, plt_title1)
```

```
LogisticRegression(C=1, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
0.6812657662908626
```

```
No handles with labels found to put in legend.
```



In [99]:

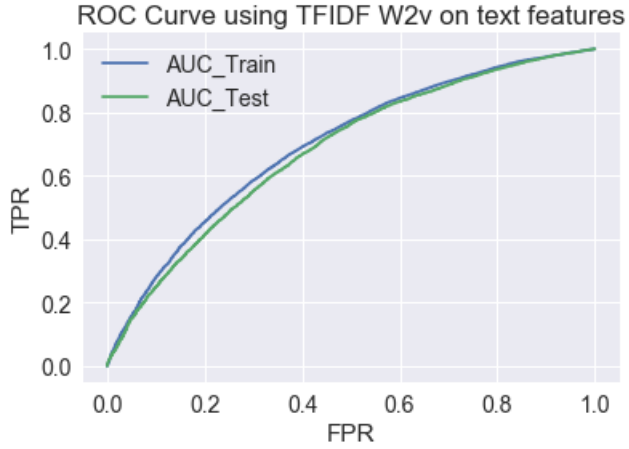
```
optimal_c=0.1
LOR=Get_LOR_OptimalC(optimal_c)
```

In [100]:

```
roc_data_test1, roc_data_train1 = getROC_Data(X_te, y_test,X_tr, y_train, LOR)
```

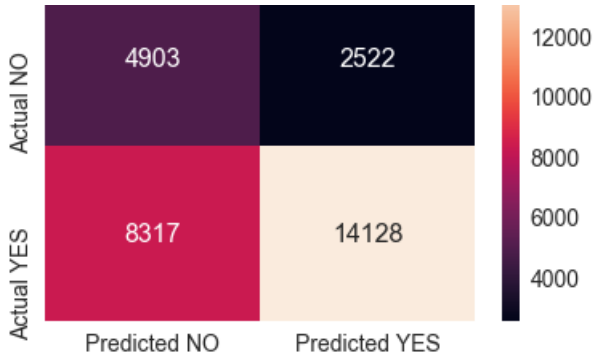
In [101]:

```
makeROC(roc_data_test1, roc_data_train1, plt_title1)
```



In [102]:

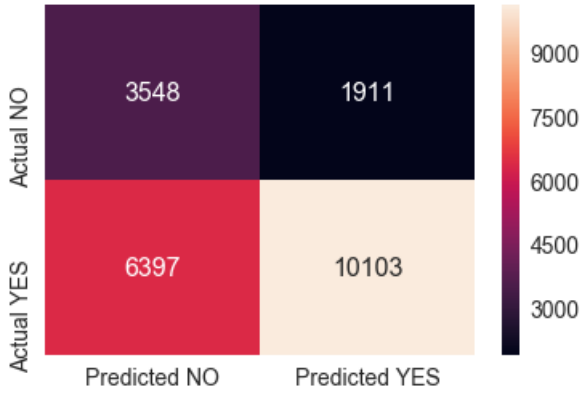
```
get_confusion_matrix(LOR,X_tr,y_train)
```



In [103]:

```
%%time
get_confusion_matrix(LOR,X_te,y_test)
```

```
Wall time: 166 ms
```



In [104]:

```
predbow = (LOR.predict_proba(X_te))
predbow_train=(LOR.predict_proba(X_tr))
```

In [105]:

```
print("AUC score for Logistic regression  with TFIDF-W2V for Testing data is ",round(metrics.roc_a
uc_score(y_test ,predbow[:,1]),3))
print("AUC score for Logistic regression  TFIDF-W2V for Training data
",round(metrics.roc_auc_score(y_train ,predbow_train[:,1]),3))
```

```
AUC score for Logistic regression  with TFIDF-W2V for Testing data is  0.68
AUC score for Logistic regression  TFIDF-W2V for Training data  0.697
```

CONCLUSION

In [107]:

```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Hyperparameter", "Test AUC","Train AUC"]
x.add_row(["Bag of Words", 0.01, 0.725,0.836])
x.add_row(["TFIDF",0.01,0.69, 0.714])
x.add_row(["AVG W2V",0.1, 0.678,0.694])
x.add_row(["TFIDF W2V", 0.1, 0.68,0.697])

print(x)
```

```
+--------------+---------------+----------+-----------+
```

Vectorizer	Hyperparameter	Test AUC	Train AUC
Bag of Words	0.01	0.725	0.836
TFIDF	0.01	0.69	0.714
AVG W2V	0.1	0.678	0.694
TFIDF W2V	0.1	0.68	0.697