DonorsChoose

In [4]:

```
%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 [5]:

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

In [6]:

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

```
109248
1541272
```

In [7]:

```
from sklearn.utils import resample
```

In [8]:

```
project_data=resample(project_data,n_samples=50000)
```

In [9]:

```
# 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[9]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
81565	95963	p155767	e50367a62524e11fbd2dc79651b6df21	Mrs.	CA	2016-04-27 01:29:58	Grades 3-5

In [10]:

```
len(project_data['project_is_approved'])
```

Out[10]:

```
50000
```

In [11]:

```
filtered = project_data.loc[project_data['project_is_approved'] == 1]
```

In [12]:

```
print(len(filtered))
```

```
42355
```

In [13]:

```
# 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 [14]:

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

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

Preprocessing data

1.2 preprocessing of `project_subject_categories`

In [15]:

```
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 [16]:

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

In [17]:

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

In [18]:

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

In [19]:

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

```
0         Grades_PreK_2
1            Grades_3_5
2            Grades_3_5
3           Grades_9_12
4         Grades_PreK_2
5         Grades_PreK_2
6            Grades_3_5
7            Grades_3_5
8            Grades_3_5
9            Grades_3_5
```

```
9           Grades_3_5
10          Grades_3_5
11       Grades_PreK_2
12       Grades_PreK_2
13       Grades_PreK_2
14       Grades_PreK_2
15       Grades_PreK_2
16       Grades_PreK_2
17          Grades_6_8
18          Grades_3_5
19          Grades_6_8
20          Grades_3_5
21       Grades_PreK_2
22          Grades_3_5
23          Grades_6_8
24          Grades_3_5
25         Grades_9_12
26          Grades_3_5
27          Grades_3_5
28       Grades_PreK_2
29       Grades_PreK_2
             ...
49970       Grades_6_8
49971    Grades_PreK_2
49972    Grades_PreK_2
49973       Grades_3_5
49974       Grades_3_5
49975       Grades_3_5
49976    Grades_PreK_2
49977       Grades_3_5
49978       Grades_3_5
49979      Grades_9_12
49980      Grades_9_12
49981    Grades_PreK_2
49982    Grades_PreK_2
49983       Grades_3_5
49984       Grades_3_5
49985       Grades_3_5
49986       Grades_3_5
49987    Grades_PreK_2
49988    Grades_PreK_2
49989       Grades_3_5
49990    Grades_PreK_2
49991    Grades_PreK_2
49992    Grades_PreK_2
49993    Grades_PreK_2
49994       Grades_3_5
49995    Grades_PreK_2
49996    Grades_PreK_2
49997       Grades_3_5
49998      Grades_9_12
49999    Grades_PreK_2
Name: preprocessed_grade, Length: 50000, dtype: object
```

In [20]:

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

```
['AppliedLearning' 'Literacy_Language Math_Science'
 'Math_Science History_Civics' 'AppliedLearning Music_Arts'
 'Math_Science AppliedLearning' 'Math_Science Literacy_Language'
 'History_Civics Literacy_Language' 'Math_Science' 'SpecialNeeds'
 'Literacy_Language' 'Music_Arts' 'AppliedLearning Literacy_Language'
 'Literacy_Language SpecialNeeds' 'AppliedLearning History_Civics'
 'AppliedLearning SpecialNeeds' 'Literacy_Language Music_Arts'
 'Health_Sports' 'History_Civics Math_Science'
 'Health_Sports SpecialNeeds' 'History_Civics Music_Arts'
 'Math_Science Health_Sports' 'Math_Science Music_Arts'
 'AppliedLearning Health_Sports' 'History_Civics SpecialNeeds'
 'Math_Science SpecialNeeds' 'Health_Sports Music_Arts' 'History_Civics'
 'Literacy_Language History_Civics' 'Literacy_Language AppliedLearning'
 'Health_Sports Literacy_Language' 'Health_Sports AppliedLearning'
 'AppliedLearning Math_Science' 'SpecialNeeds Music_Arts'
 'Music_Arts AppliedLearning' 'Music_Arts History_Civics'
 'SpecialNeeds Health_Sports' 'Music_Arts Health_Sports'
 'Literacy Language Health Sports' 'Health Sports Math Science'
```

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

1.3 preprocessing of `project_subject_subcategories`

In [21]:

```
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 [22]:

```
# 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 [23]:

```
# 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 [24]:

```
# 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', "d
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 [25]:

```
# 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%|██████████████████████████████████████████████████████████████████████████████| 50000/50000
[00:35<00:00, 1420.88it/s]
```

1.4 Preprocessing of `project_title`

In [26]:

```
# 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.lower() for e in sent.split() if e not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████| 50000/50000
[00:01<00:00, 31927.69it/s]
```

In [27]:

```
#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 [28]:

```
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%|████████████████████████████████████████████████████████████████| 50000/50000
[00:03<00:00, 14019.93it/s]
```

In [29]:

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

In [30]:

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

In [31]:

```
# 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 [32]:

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

Out[32]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
0	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
1	95963	p155767	e50367a62524e11fbd2dc79651b6df21	Mrs.	CA	2016-04-27 01:29:58	Grades 3-5

In [33]:

```
filtered_negative = project_data.loc[project_data['project_is_approved'] == 0]
print(len(filtered_negative))
#print(len(filtered_positive))
filtered_positive = project_data.loc[project_data['project_is_approved'] == 1]
sample_positive = filtered_positive.take(np.random.permutation(len(filtered_positive))[:50000])
```

```
7645
```

In [34]:

```
print(len(filtered_positive))
print(len(sample_positive))
```

```
42355
42355
```

In [35]:

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

In [36]:

```
project_data.count()
```

Out[36]:

```
Unnamed: 0                                      50000
id                                              50000
teacher_id                                      50000
teacher_prefix                                  50000
school_state                                    50000
Date                                            50000
project_grade_category                          50000
project_title                                   50000
project_resource_summary                        50000
teacher_number_of_previously_posted_projects    50000
project_is_approved                             50000
price                                           50000
quantity                                        50000
clean_categories                                50000
preprocessed_grade                              50000
clean_subcategories                             50000
essay                                           50000
clean_essays                                    50000
clean_resource_summary                          50000
clean_titles                                    50000
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 [37]:

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

In [38]:

```
# 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 [39]:

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

```
2523
```

In [40]:

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

print("="*100)
```

```
(22445, 20) (22445,)
(11055, 20) (11055,)
(16500, 20) (16500,)
====================================================================================================
```

2.Vectorizing data

BoW

2.1 Text data

In [299]:

```
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
Bow_FeatureList =vectorizer.get_feature_names()
# 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
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

In [300]:

```
type(vectorizer.get_feature_names())
```

Out[300]:

```
list
```

In [301]:

```
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
Bow_FeatureList=Bow_FeatureList + (vectorizer.get_feature_names())
# 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
(22445, 1239) (22445,)
(11055, 1239) (11055,)
(16500, 1239) (16500,)
====================================================================================================
```

In [302]:

```
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
(22445, 2508) (22445,)
(11055, 2508) (11055,)
(16500, 2508) (16500,)
====================================================================================================
```

In [303]:

```
len(Bow_FeatureList)
```

Out[303]:

```
8747
```

In [304]:

```
X_train_summary_bow.shape
```

Out[304]:

```
(22445, 2508)
```

2.2 one hot encoding the catogorical features: clean_categories

In [305]:

```
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)
Bow_FeatureList=Bow_FeatureList + (vectorizer.get_feature_names())
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
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
['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 [306]:

```
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)
Bow_FeatureList=Bow_FeatureList + (vectorizer.get_feature_names())
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
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['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 [307]:

```
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)
Bow_FeatureList=Bow_FeatureList + (vectorizer.get_feature_names())
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(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 5) (22445,)
(11055, 5) (11055,)
(16500, 5) (16500,)
['mr', 'mrs', 'ms', 'null', 'teacher']
====================================================================================================
```

2.4 one hot encoding the catogorical features: school_state

In [308]:

```
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)
Bow_FeatureList=Bow_FeatureList + (vectorizer.get_feature_names())
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
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['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']
====================================================================================================
```

In [309]:

```
len(Bow_FeatureList)
```

Out[309]:

```
8842
```

2.4 one hot encoding the catogorical features: project_grade_category

In [310]:

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

Out[310]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
42198	73167	p001623	3d31daacafc23c2e79a4f1a3a6a391c5	Mrs.	CA	2017-02-15 18:56:51	Grades 3-5

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
34351	117870	p077598	071051aaa166a06c2a2bd924efc84058	Ms.	DC	2016-12-12 10:15:50	Grades 6-8

In [311]:

```
#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 [312]:

```
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
Bow_FeatureList=Bow_FeatureList + (vectorizer.get_feature_names())
# 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
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['Grades_9_12', 'Grades_6_8', 'Grades_3_5', 'Grades_PreK_2']
====================================================================================================
```

2.5 Normalizing the numerical features: Price

In [313]:

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

Out[313]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
42198	73167	p001623	3d31daacafc23c2e79a4f1a3a6a391c5	Mrs.	CA	2017-02-15 18:56:51	Grades 3-5
34351	117870	p077598	071051aaa166a06c2a2bd924efc84058	Ms.	DC	2016-12-12 10:15:50	Grades 6-8

In [314]:

```
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)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

2.6 Vectorizing numerical features: teacher_number_of_previously _posted_projects"

In [315]:

```
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
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

In [316]:

```
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
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

In [317]:

```
Bow_FeatureList.append('price')
Bow_FeatureList.append('teacher_number_of_previously_posted_projects')
Bow_FeatureList.append('quantity')
```

In [318]:

```
len(Bow_FeatureList)
```

Out[318]:

```
8849
```

2.7 Concatinating all the features

In [319]:

```
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
(22445, 8849) (22445,)
(11055, 8849) (11055,)
(16500, 8849) (16500,)
====================================================================================================
```

Support Vector Machines

Building function to find optimal Alpha for SVM

In [320]:

```
%%time
import warnings
warnings.filterwarnings("ignore")
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
import math
from sklearn.linear_model import SGDClassifier

from sklearn.metrics import roc_auc_score
from sklearn import metrics
from sklearn import cross_validation


def find_optimal_k(X_train,y_train, myList ,Regulizer):
    cv_scores=[]
    for i in myList:
        SGD=SGDClassifier(loss='hinge', penalty=Regulizer, alpha=i)
        model = SGD.fit(X_train, y_train)
        y_pred_proba = model.predict(X_cr)
        auc = metrics.roc_auc_score(y_cv, y_pred_proba)
        cv_scores.append(auc)
    newmylist=[math.log10(i) for i in myList]
    print(newmylist)
    if Regulizer=="l1":
        plt.plot(newmylist,cv_scores,color='blue', linestyle='dashed',
marker='o',markerfacecolor='red', markersize=10)
    else:
        plt.plot(newmylist,cv_scores,color='orange', linestyle='dashed',
marker='o',markerfacecolor='red', markersize=10)

    print(cv_scores)
    #optimal_alpha= myList(cv_scores.index(min(cv_scores)))
```

```
Wall time: 0 ns
```

In [321]:

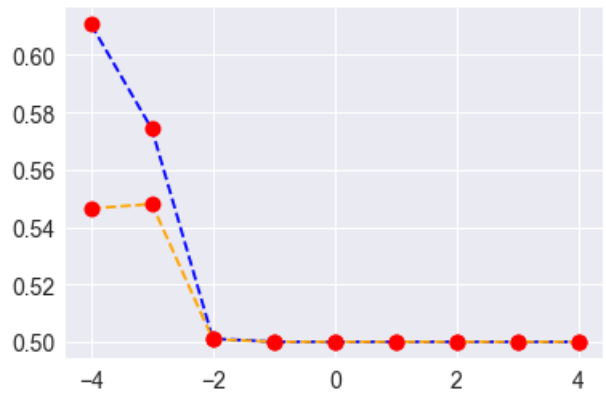
```
myList = [10**x for x in range(-4,5)]
newmylist=[math.log10(i) for i in myList]
regularizer=["l1","l2"]
print(myList)
print(newmylist)
```

```
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
[-4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
```

In [322]:

```
find_optimal_k(X_tr,y_train, myList,regularizer[0])
find_optimal_k(X_tr,y_train, myList,regularizer[1])
```

```
[-4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
[0.610791945333405, 0.5742170741493096, 0.5010742819954698, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[-4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
[0.5464868246050224, 0.5480484114021426, 0.5008875739644971, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
```



SVM with Optimal alpha

In [344]:

```
sgd = SGDClassifier(loss='hinge', penalty="l1", alpha=0.00001)
model = sgd.fit(X_tr, y_train)
```

In [345]:

```
predbow = (model.predict(X_te))
predbow_train=(model.predict(X_tr))
```

In [346]:

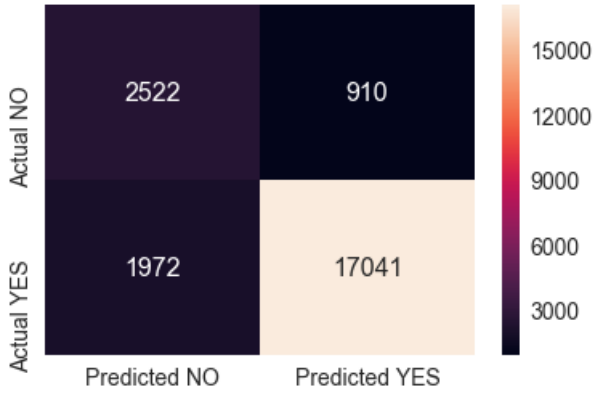
```
def get_confusion_matrix(y_test,y_pred):
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm,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

In [347]:

```
get_confusion_matrix(y_test,predbow)
```

In [348]:

```
get_confusion_matrix(y_train,predbow_train)
```



In [349]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test ,predbow))
```

```
              precision    recall  f1-score   support

           0       0.31      0.42      0.36      2523
```

```
           1       0.89      0.83      0.86     13977

avg / total       0.80      0.77      0.78     16500
```

In [350]:

```
print("AUC score for SVM model with Bag of Words is ",round(metrics.roc_auc_score(y_test ,predbow)
,3))
```

```
AUC score for SVM model with Bag of Words is  0.627
```

TFIDF vectorizer

In [72]:

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

In [73]:

```
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
TFIDF_FeatureList=TFIDF_FeatureList + vectorizer.get_feature_names()
# 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: (22445, 2118)
CV shape: (11055, 2118)
Test shape: (16500, 2118)
```

In [74]:

```
vectorizer = TfidfVectorizer(min_df=5)
vectorizer.fit(X_train['clean_resource_summary'].values) # fit has to happen only on train
datadata
TFIDF_FeatureList=TFIDF_FeatureList + vectorizer.get_feature_names()
# we use the fitted CountVectorizer to convert the text to vector
X_train_summary_tfidf = vectorizer.transform(X_train['clean_resource_summary'].values)
X_cv_summary_tfidf = vectorizer.transform(X_cv['clean_resource_summary'].values)
X_test_summary_tfidf = vectorizer.transform(X_test['clean_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
(22445, 3933) (22445,)
(11055, 3933) (11055,)
(16500, 3933) (16500,)
====================================================================================================
```

In [75]:

```
vectorizer = TfidfVectorizer(min_df=5)
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train datadata
TFIDF_FeatureList=TFIDF_FeatureList + vectorizer.get_feature_names()
# we use the fitted CountVectorizer to convert the text to vector
X_train_categories_tfidf = vectorizer.transform(X_train['clean_categories'].values)
X_cv_categories_tfidf = vectorizer.transform(X_cv['clean_categories'].values)
X_test_categories_tfidf = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_categories_tfidf.shape, y_train.shape)
print(X_cv_categories_tfidf.shape, y_cv.shape)
print(X_test_categories_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
====================================================================================================
```

In [76]:

```
vectorizer = TfidfVectorizer(min_df=5)
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train datadata
TFIDF_FeatureList=TFIDF_FeatureList + vectorizer.get_feature_names()
# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategories_tfidf = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_subcategories_tfidf = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcategories_tfidf = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subcategories_tfidf.shape, y_train.shape)
print(X_cv_subcategories_tfidf.shape, y_cv.shape)
print(X_test_subcategories_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
====================================================================================================
```

In [77]:

```
vectorizer = TfidfVectorizer(min_df=5)
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train datadata
TFIDF_FeatureList=TFIDF_FeatureList + vectorizer.get_feature_names()
# we use the fitted CountVectorizer to convert the text to vector
X_train_prefix_tfidf = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_prefix_tfidf = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_prefix_tfidf = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_prefix_tfidf.shape, y_train.shape)
print(X_cv_prefix_tfidf.shape, y_cv.shape)
print(X_test_prefix_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
====================================================================================================
```

In [78]:

```
vectorizer = TfidfVectorizer(min_df=5)
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train datadata
TFIDF_FeatureList=TFIDF_FeatureList + vectorizer.get_feature_names()
# we use the fitted CountVectorizer to convert the text to vector
X_train_school_state_tfidf = vectorizer.transform(X_train['school_state'].values)
X_cv_school_state_tfidf = vectorizer.transform(X_cv['school_state'].values)
X_test_school_state_tfidf = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_school_state_tfidf.shape, y_train.shape)
print(X_cv_school_state_tfidf.shape, y_cv.shape)
print(X_test_school_state_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
====================================================================================================
```

In [79]:

```
vectorizer = TfidfVectorizer(min_df=5)
vectorizer.fit(X_train['preprocessed_grade'].values) # fit has to happen only on train datadata
TFIDF_FeatureList=TFIDF_FeatureList + vectorizer.get_feature_names()
# we use the fitted CountVectorizer to convert the text to vector
X_train_school_grade_tfidf = vectorizer.transform(X_train['preprocessed_grade'].values)
X_cv_school_grade_tfidf = vectorizer.transform(X_cv['preprocessed_grade'].values)
X_test_school_grade_tfidf = vectorizer.transform(X_test['preprocessed_grade'].values)

print("After vectorizations")
print(X_train_school_grade_tfidf.shape, y_train.shape)
print(X_cv_school_grade_tfidf.shape, y_cv.shape)
print(X_test_school_grade_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
====================================================================================================
```

In [80]:

```
TFIDF_FeatureList.append('price')
TFIDF_FeatureList.append('teacher_number_of_previously_posted_projects')
TFIDF_FeatureList.append('quantity')
```

In [81]:

```
len(TFIDF_FeatureList)
```

Out[81]:

```
15317
```

In [82]:

```
print(type(X_train_summary_tfidf))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
```

Concatinating all features (TFIDF)

In [184]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
```

```
# 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_st
d)).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_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)
```

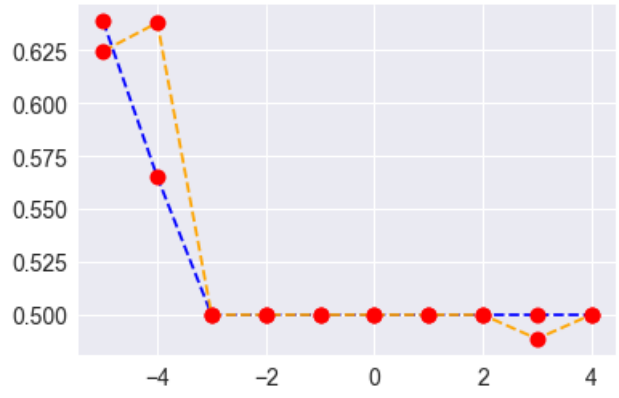
```
Final Data matrix
(22445, 15318) (22445,)
(11055, 15318) (11055,)
(16500, 15318) (16500,)
====================================================================================================
```

SVM with TFIDF

In [185]:

```
import numpy as np
myList =[10**x for x in range(-5,5)]
#myList=[1,2,3,4,5,6,7,8]
print(type(list(myList)))
find_optimal_k(X_tr,y_train, myList,regularizer[0])
find_optimal_k(X_tr,y_train, myList,regularizer[1])
```

```
<class 'list'>
[-5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
[0.6387539213425287, 0.5650783320749233, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[-5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
[0.6245337827805281, 0.638002192476709, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.48865882977345465, 0.5]
```



SVM with Optimal alpha

In [186]:

```
sgd = SGDClassifier(loss='hinge', penalty="l1", alpha=0.00001)
model = sgd.fit(X_tr, y_train)
```

In [187]:

```
predbow = (model.predict(X_te))
predbow_train=(model.predict(X_tr))
```

In [188]:

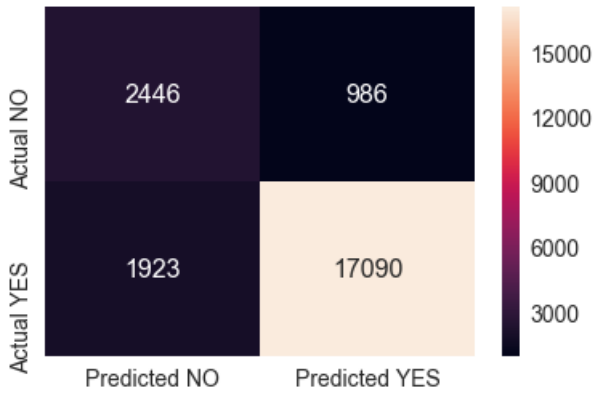
```
def get_confusion_matrix(y_test,y_pred):
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm,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

In [189]:

```
get_confusion_matrix(y_test,predbow)
```

In [190]:

```
get_confusion_matrix(y_train,predbow_train)
```



In [191]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test ,predbow))
```

```
             precision    recall  f1-score   support

          0       0.34      0.43      0.38      2523
          1       0.89      0.85      0.87     13977

avg / total       0.81      0.79      0.80     16500
```

In [192]:

```
print("AUC score for Naive Bayes model with TFIDF is ",round(metrics.roc_auc_score(y_test ,predbow
),3))
```

```
AUC score for Naive Bayes model with TFIDF is  0.642
```

AUC score for Naive Bayes model with TFIDF is 0.642

SVM with Truncated SVD and TFIDF

In [85]:

```
from sklearn.decomposition import TruncatedSVD
from sklearn.random_projection import sparse_random_matrix
variance_list=[]
#X = sparse_random_matrix(100, 100, density=0.01, random_state=42)
```

In [99]:

```
svd = TruncatedSVD(n_components=100, n_iter=3, random_state=18)
svd.fit(X_train_essay_tfidf)
print(svd.explained_variance_.sum())
```

```
0.1791584902351223
```

In [92]:

```
print(variance_list)
```

```
[0.7436603621464959, 0.8261892612340141, 0.7323082066980201, 0.7329685361646087,
0.7336289258081171, 0.7342729726397218, 0.7349274941145908, 0.7355254696360043, 0.736143657032875,
0.7367598050081593, 0.7373526741589973, 0.7379207089957389, 0.7385000623775271, 0.739053933773056,
0.7395991652508535, 0.7401149235628942, 0.74063273326777, 0.741177040897493, 0.7416761559151503, 0
.7421783359409488, 0.7426867190376211, 0.7431779113039604, 0.8685448929908284]
```

In [106]:

```
list_i=[10,100,500,1000,1500,2000,3000,4000,5000]
variance_list=[]
for i in (list_i):
    svd = TruncatedSVD(n_components=i, n_iter=3, random_state=42)
    svd.fit(X_train_essay_tfidf)
    variance_list .append(svd.explained_variance_.sum())
    print(i,svd.explained_variance_.sum())
```

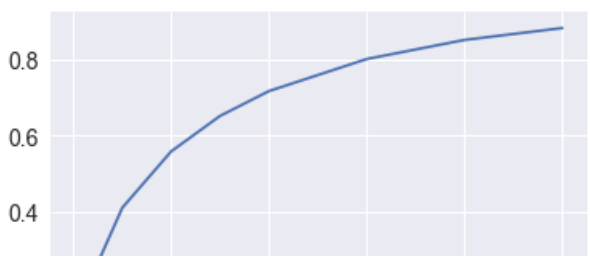
```
10 0.05070054330229548
100 0.17921697616850696
500 0.41027855969712473
1000 0.5580200331065113
1500 0.6510287144983935
2000 0.7161149526268495
3000 0.8002205649514786
4000 0.8502359708501246
5000 0.8814587510392509
```

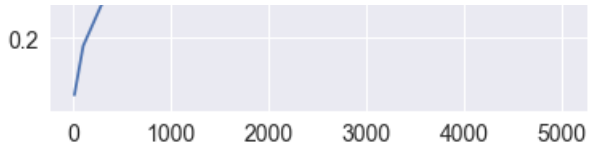
In [107]:

```
plt.plot(list_i,variance_list)
```

Out[107]:

```
[<matplotlib.lines.Line2D at 0x133ffd5ca90>]
```





In [108]:

```
svd = TruncatedSVD(n_components=5000, n_iter=3, random_state=18)
X_train_essay_SVD=svd.fit_transform(X_train_essay_tfidf)
```

In [113]:

```
X_test_essay_SVD=svd.fit_transform(X_test_essay_tfidf)
```

In [130]:

```
X_cv_essay_SVD=svd.fit_transform(X_cv_essay_tfidf)
```

In [171]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_SVD,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((X_cv_essay_SVD,X_cv_clean_cat_ohe,X_cv_clean_subcat_ohe, X_cv_state_ohe, X_cv_teache
r_ohe, X_cv_grade_ohe, X_cv_price_std,X_cv_projects_std,X_cv_qty_std)).tocsr()
X_te = hstack((X_test_essay_SVD,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
(22445, 5102) (22445,)
(11055, 5102) (11055,)
(16500, 5102) (16500,)
====================================================================================================
```

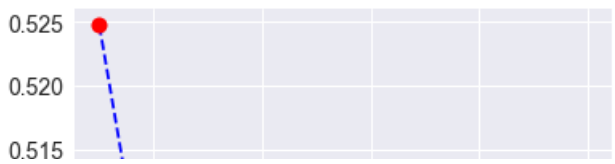
In [172]:

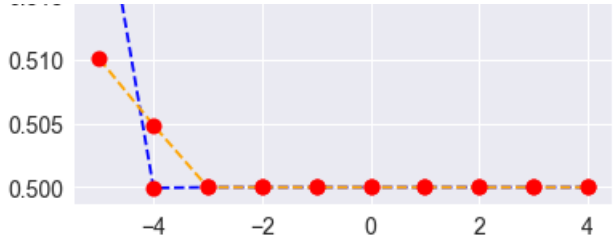
```
import numpy as np
myList =[10**x for x in range(-5,5)]
#myList=[1,2,3,4,5,6,7,# SVM with Optimal alpha
```

In [173]:

```
print(type(list(myList)))
find_optimal_k(X_tr,y_train, myList,regularizer[0])
find_optimal_k(X_tr,y_train, myList,regularizer[1])
```

```
<class 'list'>
[-5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
[0.5247486075877386, 0.4999466097170315, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[-5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
[0.5100484303572727, 0.5048555145212091, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
```





In [179]:

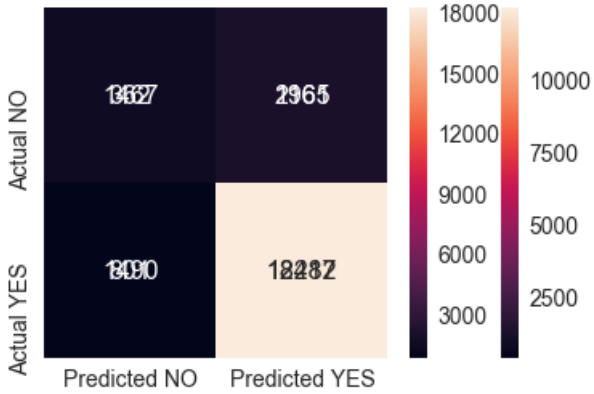
```
sgd = SGDClassifier(loss='hinge', penalty="l1", alpha=0.00001)
model = sgd.fit(X_tr, y_train)
```

In [180]:

```
predbow = (model.predict(X_te))
predbow_train=(model.predict(X_tr))
```

In [181]:

```
get_confusion_matrix(y_test,predbow)


get_confusion_matrix(y_train,predbow_train)
```



In [182]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test ,predbow))
```

```
              precision    recall  f1-score   support

           0       0.20      0.14      0.17      2523
           1       0.85      0.89      0.87     13977

avg / total       0.75      0.78      0.76     16500
```

In [183]:

```
print("AUC score for SVM model with TFIDF is ",round(metrics.roc_auc_score(y_test ,predbow),3))
```

```
AUC score for SVM model with TFIDF is  0.518
```

AVG W2V

In [193]:

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

```
    glove_words =  set(model.keys())
```

In [194]:

```
# 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%|██████████████████████████████████████████████████████████████████████████| 22445/22445
[00:09<00:00, 2356.10it/s]
```

```
train  vector
22445
50
==================================================
```

```
100%|██████████████████████████████████████████████████████████████████████████| 16500/16500
[00:05<00:00, 3157.11it/s]
```

```
Test vec
16500
50
```

```
50
==================================================

100%|██████████████████████████████████████████████████████████████████████████| 11055/11055
[00:04<00:00, 2680.71it/s]

CV vec
11055
50
==================================================
```

In [195]:

```
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 [196]:

```
# 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%|██████████████████████████████████████████████████████████████| 22445/22445
[00:00<00:00, 42316.26it/s]
```

```
train  vector
22445
50
==================================================
```

```
100%|██████████████████████████████████████████████████████████████| 16500/16500
[00:00<00:00, 51167.56it/s]
```

```
Test vec
16500
50
==================================================
```

```
100%|██████████████████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 40005.44it/s]
```

```
CV vec
11055
50
==================================================
```

In [197]:

```
# 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 [198]:

```
# 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:
            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%|████████████████████████████████████████████████████████████████████████| 22445/22445
[00:01<00:00, 20980.88it/s]
```

```
train  vector
22445
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████████████████| 16500/16500
[00:00<00:00, 18412.92it/s]
```

```
Test vec
16500
50
==================================================
```

```
100%|████████████████████████████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 21029.26it/s]
```

```
CV vec
11055
50
==================================================
```

In [199]:

```
# 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 [200]:

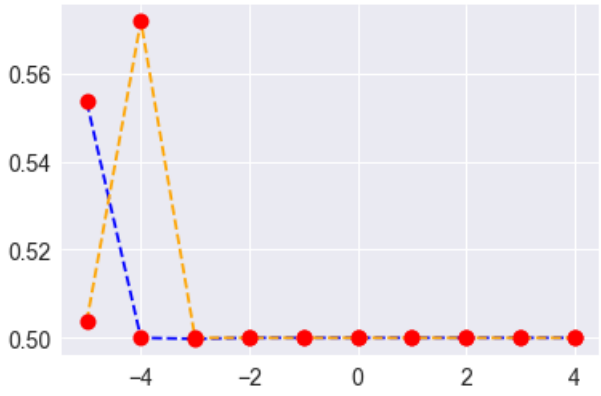
```
from scipy.sparse import hstack
X_tr =
hstack((train_w2v_vectors_essays,train_w2v_vectors_titles,train_w2v_vectors_summary,X_train_clean_c
at_ohe,X_train_clean_subcat_ohe, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
X_train_price_std,X_train_projects_std,X_train_qty_std)).tocsr()
X_cr =
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_c
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
(22445, 252) (22445,)
(11055, 252) (11055,)
(16500, 252) (16500,)
==========================================================================================================
```

In [201]:

```
import numpy as np
myList =[10**x for x in range(-5,5)]
#myList=[1,2,3,4,5,6,7,8]
print(type(list(myList)))
find_optimal_k(X_tr,y_train, myList,regularizer[0])
find_optimal_k(X_tr,y_train, myList,regularizer[1])
```

```
<class 'list'>
[-5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
[0.5536150592189855, 0.5, 0.499786438868126, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[-5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
[0.5036057396133785, 0.571875957628966, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
```

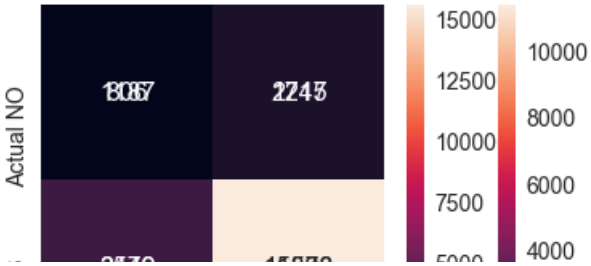


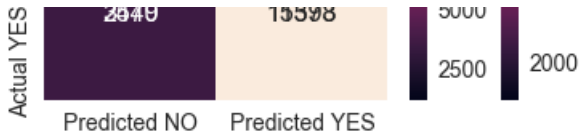
In [202]:

```
# SVM with Optimal alpha

sgd = SGDClassifier(loss='hinge', penalty="l1", alpha=0.00001)
model = sgd.fit(X_tr, y_train)

predbow = (model.predict(X_te))
predbow_train=(model.predict(X_tr))

def get_confusion_matrix(y_test,y_pred):
    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')
```

In [203]:

```
get_confusion_matrix(y_test,predbow)


get_confusion_matrix(y_train,predbow_train)
```





In [204]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test ,predbow))

print("AUC score for SVM with AVG W2V is ",round(metrics.roc_auc_score(y_test ,predbow),3))
```

```
             precision    recall  f1-score   support

          0       0.24      0.32      0.27      2523
          1       0.87      0.82      0.84     13977

avg / total       0.77      0.74      0.75     16500

AUC score for SVM with AVG W2V is  0.567
```

SVM on TFIDF - AVG W2V

In [217]:

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

# 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())
```

In [218]:

```
# 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%|██████████████████████████████████████████████████████████████████████████| 22445/22445 [01:
01<00:00, 363.07it/s]
```

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

```
100%|██████████████████████████████████████████████████████████████████████████| 11055/11055 [00:
30<00:00, 361.16it/s]
```

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

```
100%|██████████████████████████████████████████████████████████████████████████| 16500/16500 [00:
44<00:00, 369.95it/s]
```

```
Test matrix:
16500
50
==================================================
```

In [219]:

```
# 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 [220]:

```
# 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%|██████████████████████████████████████████████████████████████████████████| 22445/22445
[00:00<00:00, 44902.78it/s]
```

```
train  vector
22445
50
==================================================
```

```
100%|██████████████████████████████████████████████████████████████████████████| 16500/16500
[00:00<00:00, 38713.43it/s]
```

```
Test vec
16500
50
==================================================
```

```
100%|██████████████████████████████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 37328.52it/s]
```

```
CV vec
11055
50
```

==

In [221]:

```
# 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 [222]:

```
# 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:
            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%|████████████████████████████████████████████████████████████████████████████| 22445/22445
[00:01<00:00, 15653.57it/s]
```

```
train  vector
22445
50
```

```
==================================================

100%|████████████████████████████████████████████████████████████████| 16500/16500
[00:00<00:00, 20606.98it/s]

Test vec
16500
50
==================================================

100%|████████████████████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 16740.82it/s]

CV vec
11055
50
==================================================
```

In [223]:

```
# 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 [224]:

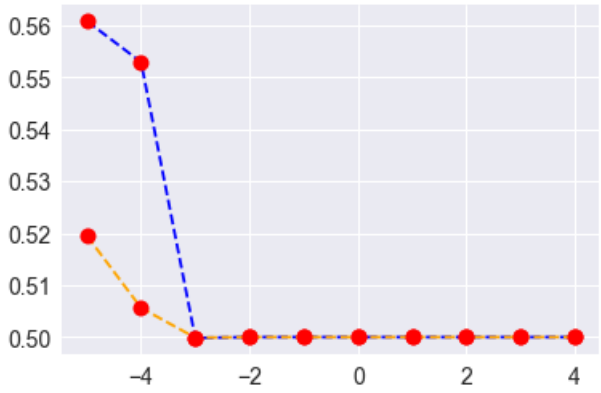
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr =
hstack((train_w2v_vectors_essays,train_w2v_vectors_titles,train_w2v_vectors_summary,X_train_clean_c
at_ohe,X_train_clean_subcat_ohe, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
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_c
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
(22445, 252) (22445,)
(11055, 252) (11055,)
(16500, 252) (16500,)
====================================================================================================
```

In [230]:

```
import numpy as np
myList =[10**x for x in range(-5,5)]
#myList=[1,2,3,4,5,6,7,8]
print(type(list(myList)))
find_optimal_k(X_tr,y_train, myList,regularizer[0])
find_optimal_k(X_tr,y_train, myList,regularizer[1])
```

```
<class 'list'>
[-5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
[0.560799053507173, 0.5528995030596738, 0.4998398291510945, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[-5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
[0.5195165178162426, 0.505618932383892, 0.499893219434063, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
```

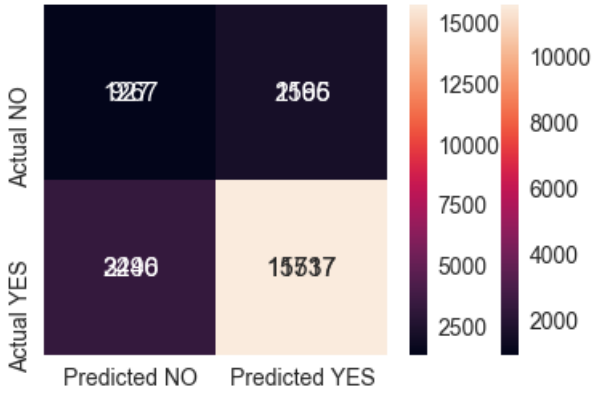


In [231]:

```
# SVM with Optimal alpha

sgd = SGDClassifier(loss='hinge', penalty="l1", alpha=0.00001)
model = sgd.fit(X_tr, y_train)

predbow = (model.predict(X_te))
predbow_train=(model.predict(X_tr))

def get_confusion_matrix(y_test,y_pred):
    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')
```

In [232]:

```
get_confusion_matrix(y_test,predbow)


get_confusion_matrix(y_train,predbow_train)
```



In [233]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test ,predbow))

print("AUC score for SVM model with TFIDF AVG W2V is ",round(metrics.roc_auc_score(y_test ,predbow
),3))
```

```
             precision    recall  f1-score   support

          0       0.28      0.37      0.31      2523
          1       0.88      0.83      0.85     13977

avg / total       0.79      0.76      0.77     16500

AUC score for SVM model with TFIDF AVG W2V is  0.596
```

In [351]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Hyperparameter","Regularizer", "AUC"]
x.add_row(["Bag of Words", 0.0001,"l1",0.627])
x.add_row(["TFIDF", 0.0001, "l1",0.642])
x.add_row(["TFIDF With Truncated SVM ", 0.0001, "l1",0.518])
x.add_row(["AVG w2v", 0.00001, "l1",0.567])
x.add_row(["TFIDF - AVG W2V", 0.00001, "l1",0.596])

print(x)
```

```
+--------------------------+----------------+-------------+-------+
|         Vectorizer       | Hyperparameter | Regularizer |  AUC  |
+--------------------------+----------------+-------------+-------+
|        Bag of Words      |     0.0001     |      l1     | 0.627 |
|           TFIDF          |     0.0001     |      l1     | 0.642 |
| TFIDF With Truncated SVM |     0.0001     |      l1     | 0.518 |
|          AVG w2v         |     1e-05      |      l1     | 0.567 |
|      TFIDF - AVG W2V     |     1e-05      |      l1     | 0.596 |
+--------------------------+----------------+-------------+-------+
```