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

```
109248
1541272
```

In [4]:

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

In [5]:

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

In [8]:

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


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


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

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[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Grades 3-5
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Grades 3-5

In [11]:

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

Out[11]:

```
50000
```

In [12]:

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

In [13]:

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

```
42484
```

In [14]:

```
# 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 [15]:

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

Preprocessing data

1.2 preprocessing of `project_subject_categories`

In [16]:

```
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 [17]:

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

In [18]:

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

In [19]:

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

In [20]:

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

```
0           Grades_3_5
1           Grades_3_5
2           Grades_3_5
3        Grades_PreK_2
4        Grades_PreK_2
5        Grades_PreK_2
6        Grades_PreK_2
7           Grades_3_5
```

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

In [21]:

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

```
['Math_Science History_Civics' 'Literacy_Language'
 'Literacy_Language Math_Science' 'Math_Science Music_Arts'
 'AppliedLearning Literacy_Language' 'Math_Science' 'Music_Arts'
 'Health_Sports' 'Literacy_Language SpecialNeeds'
 'Math_Science Literacy_Language' 'AppliedLearning'
 'AppliedLearning History_Civics' 'AppliedLearning Music_Arts'
 'History_Civics Math_Science' 'Math_Science SpecialNeeds'
 'SpecialNeeds Health_Sports' 'History_Civics Literacy_Language'
 'Literacy_Language Music_Arts' 'Math_Science Health_Sports'
 'SpecialNeeds' 'SpecialNeeds Music_Arts' 'AppliedLearning Health_Sports'
 'Literacy_Language History_Civics' 'History_Civics'
 'Health_Sports SpecialNeeds' 'Health_Sports Literacy_Language'
 'AppliedLearning SpecialNeeds' 'AppliedLearning Math_Science'
 'Math_Science AppliedLearning' 'Health_Sports AppliedLearning'
 'Literacy_Language AppliedLearning' 'History_Civics Music_Arts'
 'Health_Sports Music_Arts' 'Music_Arts Health_Sports'
 'Music_Arts AppliedLearning' 'Health_Sports Math_Science'
```

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

1.3 preprocessing of `project_subject_subcategories`

In [22]:

```
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 [23]:

```
# 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 [24]:

```
# 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 [25]:

```
# 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', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [26]:

```
# 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
[00:36<00:00, 1364.73it/s]
```

1.4 Preprocessing of `project_title`

In [27]:

```
# 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, 28079.18it/s]
```

In [28]:

```
#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 [29]:

```
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, 12608.68it/s]
```

In [30]:

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

In [31]:

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

In [32]:

```
# 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 [33]:

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

Out[33]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
0	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Grades 3-5

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
1	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Grades 3-5

In [85]:

```
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])
```

```
7516
```

In [86]:

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

```
42484
42484
```

In [87]:

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

In [88]:

```
project_data.count()
```

Out[88]:

```
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 [38]:

```
y = project_data['project_is_approved'].values
```

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

In [39]:

```
# 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 [40]:

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

```
2480
```

In [41]:

```
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 [42]:

```
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 [43]:

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

Out[43]:

```
list
```

In [44]:

```
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, 1241) (22445,)
(11055, 1241) (11055,)
(16500, 1241) (16500,)
====================================================================================================
```

In [45]:

```
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, 2530) (22445,)
(11055, 2530) (11055,)
(16500, 2530) (16500,)
====================================================================================================
```

In [46]:

```
len(Bow_FeatureList)
```

Out[46]:

```
8771
```

In [47]:

```
X_train_summary_bow.shape
```

Out[47]:

```
(22445, 2530)
```

2.2 one hot encoding the catogorical features: clean_categories

In [48]:

```
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 [49]:

```
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 [50]:

```
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(vectorizer.get_feature_names())
print("="*100)
```

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

2.4 one hot encoding the catogorical features: school_state

In [51]:

```
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 [52]:

```
len(Bow_FeatureList)
```

Out[52]:

```
8866
```

2.4 one hot encoding the catogorical features: project_grade_category

In [53]:

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

Out[53]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cat
38703	150029	p033189	bd2049000d07f47146451280b21b7917	Mrs.	NJ	2017-01-20	Grades PreK-2

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	07:21:25 Date	project_grade_cat
20089	78749	p052615	7d99840f0b143f0133e9fa22dbeec20e	Ms.	CA	2016-09-05 01:11:36	Grades 6-8

In [54]:

```
#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 [55]:

```
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 [56]:

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

Out[56]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cat
38703	150029	p033189	bd2049000d07f47146451280b21b7917	Mrs.	NJ	2017-01-20 07:21:25	Grades PreK-2

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cat
20089	78749	p052615	7d99840f0b143f0133e9fa22dbeec20e	Ms.	CA	2016-09-05 01:11:36	Grades 6-8

In [57]:

```
from sklearn.preprocessing import Normalizer
# 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.
transformer = Normalizer().fit(X_train['price'].values.reshape(1,-1))  # fit does nothing.
Normalizer(copy=True, norm='l2')
X_train_price_norm=transformer.transform(X_train['price'].values.reshape(1,-1))
X_cv_price_norm=transformer.transform(X_cv['price'].values.reshape(1,-1))
X_test_price_norm=transformer.transform(X_test['price'].values.reshape(1,-1))

print("After normalization")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

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

2.6 Vectorizing numerical features: teacher_number_of_previously _posted_projects"

In [58]:

```
# 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.
transformer =
Normalizer().fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))  # f
it does nothing.
Normalizer(copy=True, norm='l2')
X_train_previous_norm=transformer.transform(X_train['teacher_number_of_previously_posted_projects'
].values.reshape(1,-1))
X_cv_previous_norm=transformer.transform(X_cv['teacher_number_of_previously_posted_projects'].valu
es.reshape(1,-1))
X_test_previous_norm=transformer.transform(X_test['teacher_number_of_previously_posted_projects'].
values.reshape(1,-1))

print("After normalization")
print(X_train_previous_norm.shape, y_train.shape)
print(X_cv_previous_norm.shape, y_cv.shape)
print(X_test_previous_norm.shape, y_test.shape)
print("="*100)
```

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

In [59]:

```
# 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.
transformer = Normalizer().fit(X_train['quantity'].values.reshape(1,-1))  # fit does nothing.
Normalizer(copy=True, norm='l2')
X_train_quantity_norm=transformer.transform(X_train['quantity'].values.reshape(1,-1))
X_cv_quantity_norm=transformer.transform(X_cv['quantity'].values.reshape(1,-1))
X_test_quantity_norm=transformer.transform(X_test['quantity'].values.reshape(1,-1))

print("After normalization")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_cv_quantity_norm.shape, y_cv.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

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

In [60]:

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

In [61]:

```
len(Bow_FeatureList)
```

Out[61]:

```
8873
```

2.7 Concatinating all the features

In [62]:

```
# 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_norm.T,X_train_previous_norm.T,X_train_quantity_norm.T)).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_norm.T,X_cv_previous_norm.T,X_cv_quant
ity_norm.T)).tocsr()
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_norm.T,X_test_previous_norm.T,X_test_quantity_norm.T)).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, 8873) (22445,)
(11055, 8873) (11055,)
(16500, 8873) (16500,)
====================================================================================================
```

2.4 Appling NB() on different kind of featurization as mentioned in the instructions

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

2.4.1 Applying Naive Bayes on BOW, SET 1

Building function to find optimal Alpha for Naive Bayes

In [63]:

```
%%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.metrics import roc_auc_score
from sklearn import metrics
from sklearn import cross_validation


def find_optimal_k(X_train,y_train, myList):
    cv_scores=[]
    for i in myList:
        nb = MultinomialNB(alpha = i,class_prior=[0.5,0.5])
        model = nb.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)
    plt.plot(newmylist,cv_scores,color='blue', linestyle='dashed',
marker='o',markerfacecolor='red', markersize=10)
    print(cv_scores)
    #optimal_alpha= myList(cv_scores.index(min(cv_scores)))
```

```
Wall time: 71.4 ms
```

In [64]:

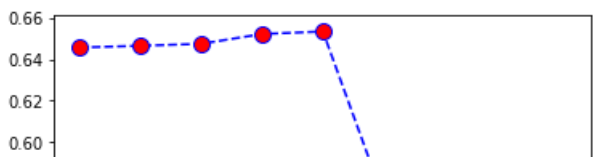
```
from sklearn.naive_bayes import MultinomialNB
import numpy as np
myList = [10**x for x in range(-4,5)]
newmylist=[math.log10(i) for i in myList]
print(myList)
print(newmylist)
```

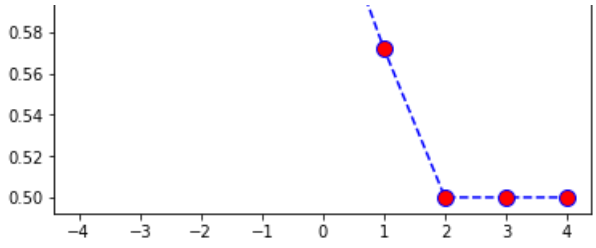
```
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]
[-4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
```

In [65]:

```
find_optimal_k(X_tr,y_train, myList)
```

```
[-4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
[0.6457350142840067, 0.6465980503954669, 0.6475280578017042, 0.652292436067876, 0.65353635340243,
0.5720518890132871, 0.5, 0.5, 0.5]
```





Observation: If we take alpha to be 100 the alpha is close to 0.51 which is very near to ideal value 0.5.

Naive Bayes with Optimal alpha

In [66]:

```
nb = MultinomialNB(alpha = 100,class_prior = [0.5,0.5] )
model = nb.fit(X_tr, y_train)
```

In [67]:

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

In [68]:

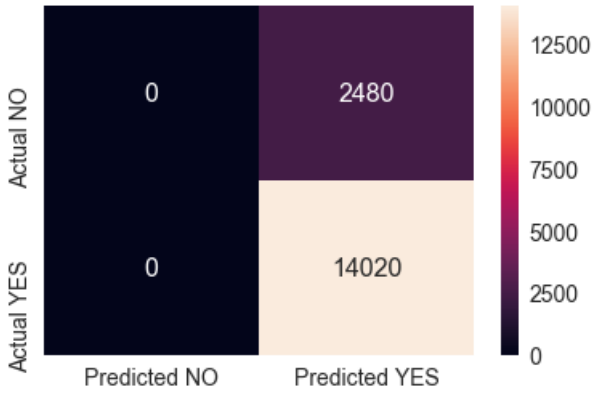
```
predbowprob = model.predict_proba(X_te)
predbowprob_train=model.predict_proba(X_tr)
```

In [69]:

```
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')
```

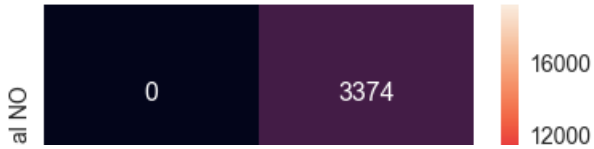
In [70]:

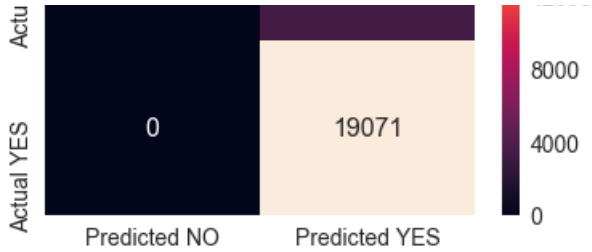
```
get_confusion_matrix(y_test,predbow)
```



In [71]:

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





In [72]:

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

```
             precision    recall  f1-score   support

          0       0.00      0.00      0.00      2480
          1       0.85      1.00      0.92     14020

avg / total       0.72      0.85      0.78     16500
```

In [73]:

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

```
AUC score for Naive Bayes model with Bag of Words is  0.5
```

Feature Importance

In [74]:

```
model.class_count_
```

Out[74]:

```
array([ 3374., 19071.])
```

In [75]:

```
df = pd.DataFrame(model.feature_log_prob_)
df1_transposed = df.T
```

In [76]:

```
fe_bow_neg = df1_transposed[0].sort_values(ascending = False)[0:10]
fe_bow_pos =df1_transposed[1].sort_values(ascending = False)[0:10]
```

2.4.1.1 Top 10 important features of negative class from SET 2

In [77]:

```
indices=fe_bow_neg.index.values
```

In [78]:

```
print(len(Bow_FeatureList))
print(len(fe_bow_neg))
```

```
8873
10
```

In [79]:

```
for i in indices:
    print(Bow_FeatureList[i],fe_bow_neg[i])
```

```
students -4.08398421924457
school -5.174963696832739
learning -5.5016710977629835
classroom -5.653028274340107
not -5.8238641435681515
learn -5.827039137076671
help -5.858598471338295
students -5.882874609191935
need -5.918418695739053
my -5.97266461608057
```

2.4.1.2 Top 10 important features of positive class from SET 2

In [80]:

```
print(len(Bow_FeatureList[i]))
print(len(fe_bow_pos))
```

```
2
10
```

In [81]:

```
indices=fe_bow_pos.index.values
for i in indices:
    print(Bow_FeatureList[i],fe_bow_pos[i])
```

```
students -3.3193492141404253
school -4.454062640695128
learning -4.834764641924977
classroom -4.855067722295763
not -5.1261256043491485
learn -5.160676443307308
help -5.181697962762474
students -5.2143223617870955
need -5.24938629765494
my -5.287888587181273
```

2.5 Appling Naive Bayes with tf-idf

In [82]:

```
%%time
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)
```

```
Wall time: 12.4 s
```

In [83]:

```
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
```

```
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, 2067)
CV shape: (11055, 2067)
Test shape: (16500, 2067)
```

In [84]:

```
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, 3893) (22445,)
(11055, 3893) (11055,)
(16500, 3893) (16500,)
====================================================================================================
```

In [89]:

```
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 [90]:

```
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 [91]:

```
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, 5) (22445,)
(11055, 5) (11055,)
(16500, 5) (16500,)
====================================================================================================
```

In [92]:

```
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 [93]:

```
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 [94]:

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

In [95]:

```
len(TFIDF_FeatureList)
```

Out[95]:

```
15282
```

Concatinating all features (TFIDF)

In [96]:

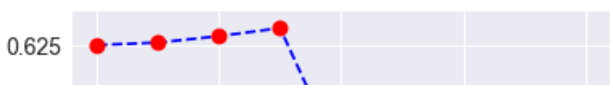
```
from scipy.sparse import hstack
X_tr =
hstack((X_train_essay_tfidf,X_train_titles_tfidf,X_train_summary_tfidf,X_train_clean_cat_ohe,X_trai
n_clean_subcat_ohe, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_norm.
T,X_train_previous_norm.T,X_train_quantity_norm.T)).tocsr()
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_norm.T,X_cv_previous_norm.T,X_cv_
quantity_norm.T)).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_norm.T,
X_test_previous_norm.T,X_test_quantity_norm.T)).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)
```

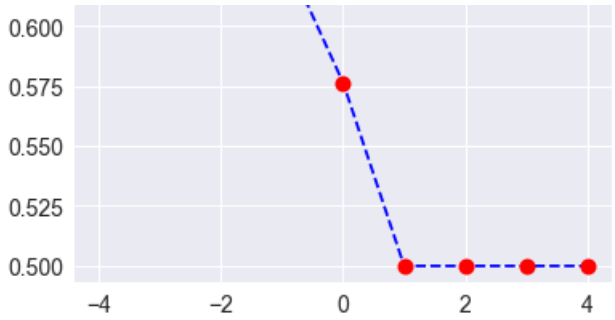
```
Final Data matrix
(22445, 15282) (22445,)
(11055, 15282) (11055,)
(16500, 15282) (16500,)
==========================================================================================================
```

In [97]:

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

```
<class 'list'>
[-4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
[0.6249521015918991, 0.626002311422478, 0.6286511846712795, 0.6320398168849143,
0.5763454504295196, 0.5, 0.5, 0.5, 0.5]
```





Observation: If we take alpha value as 10 , the AUC scores obtained is 0.57,hence taking alpha equal to 10000.

Naive Bayes with Optimal alpha

In [98]:

```
nb = MultinomialNB(alpha = 10, class_prior = [0.5,0.5] )
model_new = nb.fit(X_tr, y_train)
```

In [99]:

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

In [100]:

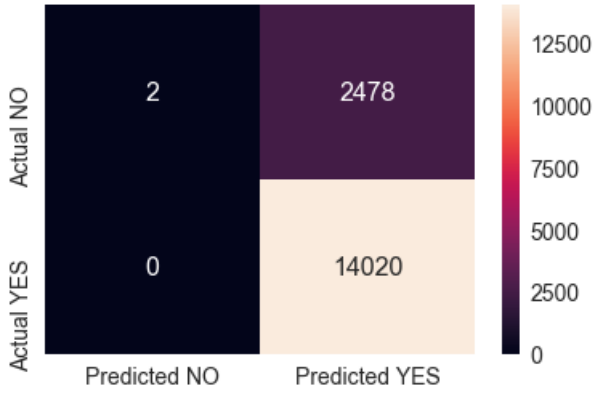
```
predbowprob = model_new.predict_proba(X_te)
predbowprob_train=model_new.predict_proba(X_tr)
```

In [101]:

```
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')
```

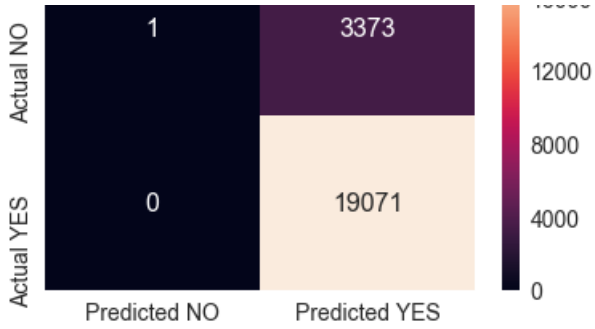
In [102]:

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



In [103]:

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



In [104]:

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

```
             precision    recall  f1-score   support

          0       1.00      0.00      0.00      2480
          1       0.85      1.00      0.92     14020

avg / total       0.87      0.85      0.78     16500
```

In [105]:

```
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  0.5
```

Feature Importance

In [106]:

```
model_new.class_count_
```

Out[106]:

```
array([ 3374., 19071.])
```

In [107]:

```
df = pd.DataFrame(model_new.feature_log_prob_)
df1_transposed = df.T
```

In [108]:

```
len(df1_transposed)
```

Out[108]:

```
15282
```

In [109]:

```
fe_tfidf_neg = df1_transposed[0].sort_values(ascending = False)[0:10]
fe_tfidf_pos =df1_transposed[1].sort_values(ascending = False)[0:10]
```

2.4.2.1 Top 10 important features of negative class from SET 2

In [110]:

```
indices=fe_tfidf_neg.index.values
```

In [111]:

```
indices
```

Out[111]:

```
array([15272, 15184, 15185, 15278, 15273, 15277, 15208, 15206, 15207,
       15276], dtype=int64)
```

In [112]:

```
fe_tfidf_neg
```

Out[112]:

```
15272   -4.870034
15184   -5.025480
15185   -5.065343
15278   -5.073364
15273   -5.164510
15277   -5.278191
15208   -5.498419
15206   -5.519925
15207   -5.793219
15276   -5.974038
Name: 0, dtype: float64
```

In [113]:

```
print(indices)
for i in indices:
    print(TFIDF_FeatureList[i],fe_tfidf_neg[i])
```

```
[15272 15184 15185 15278 15273 15277 15208 15206 15207 15276]
wi -4.87003445489963
literacy_language -5.025480278120879
math_science -5.065343207220397
grades_prek_2 -5.073363665818316
wv -5.164509760269015
grades_9_12 -5.278191354673506
mathematics -5.4984191250725525
literacy -5.519925330293516
literature_writing -5.793218665293197
grades_6_8 -5.974037592202617
```

2.4.2.2 Top 10 important features of positive class from SET 2

In [114]:

```
indices=fe_tfidf_pos.index.values
```

In [115]:

```
#print((fe_tfidf_pos))
for i in indices:
    print(TFIDF_FeatureList[i],fe_tfidf_pos[i])
```

```
wi -3.9690103465906876
literacy_language -4.02658990672588
grades_prek_2 -4.222761742991139
math_science -4.3033676723072745
wv -4.3342625780052515
grades_9_12 -4.374878802490949
literacy -4.463404184311793
mathematics -4.691934116993664
literature_writing -4.883437708248248
grades_6_8 -5.199096606415026
```

Conclusions

In [117]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Hyperparameter", "AUC"]
x.add_row(["Bag of Words", 100, 0.5])
x.add_row(["TFIDF", 10, 0.5])
print(x)
```

```
+--------------+----------------+-----+
|  Vectorizer  | Hyperparameter | AUC |
+--------------+----------------+-----+
| Bag of Words |      100       | 0.5 |
|    TFIDF     |       10       | 0.5 |
+--------------+----------------+-----+
```