Naive Bayes Algorithm on Donors_Choose dataset

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import sqlite3
import pandas as pd
import numpy as np
import nltk
import gc
gc.enable()
gc.DEBUG_SAVEALL
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os
import math
from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
gc.set_threshold(2, 1, 1)
```

2.1 Loading Input Data

In [2]:

```
# %load_ext memory_profiler

project_data = pd.read_csv('train_data.csv')
# project_data=project_data.dropna(how='any')
project_data=project_data.fillna("")
# project_data=project_data.head(15000)
# s=0
# # We are taking samples of 0's and 1's and appending them to overcome memory er
# project_data_1 = project_data[project_data['project_is_approved'] == s+1]
# project_data_0 = project_data[project_data['project_is_approved'] == s]

project_data_1=project_data.head(20000)
project_data_0=project_data.tail(8000)
project_data_1=project_data_1.append(project_data_0)
project_data=project_data_1
resource_data = pd.read_csv('resources.csv')
```

In [3]:

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

```
Number of data points in train data (28000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'scho
ol_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

```
print("Number of data points in resource data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(1)
# project_data.head(2)
```

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

Out[4]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.0

In [5]:
```
y = project_data['project_is_approved'].values
X = project_data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[5]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_subr
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016

In [6]:
```
#To make best use of the memory we are setting the variable names to 'None' and p

project_data=None
project_data_1=None
project_data_0=None

gc.collect()
gc.enable()
gc.DEBUG_SAVEALL
```

Out[6]:
```
32
```

2.2 Getting the Data Model Ready:Preprocessing and Vectorizing categorical features

2.2.1 Preprocessing:project_grade_category

In [7]:
```
sub_catogories = list(X['project_grade_category'].values)
# remove special characters from list of strings python: https://stackoverflow.c

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-

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",
        if 'The' in j.split(): # this will split each of the catogory based on s
            j=j.replace('The','') # if we have the words "The" we are going to r
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trail
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())
X['project_grade_category'] = sub_cat_list
```

In [8]:

```
sub_catogories=None
sub_cat_list=None
temp=None
i=None
j=None
catogories=None
cat_list=None
temp=None
my_counter=None
word=None
cat_dict=None
gc.collect()
gc.enable()
gc.DEBUG_SAVEALL
```

Out[8]:

```
32
```

2.2.2 Preprocessing:project_subject_categories

In [9]:

```
catogories = list(X['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.co

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-
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",
        if 'The' in j.split(): # this will split each of the catogory based on sp
            j=j.replace('The','') # if we have the words "The" we are going to re
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trail
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

X['clean_categories'] = cat_list
X.drop(['project_subject_categories'], axis=1, inplace=True)
X.head(2)
```

Out[9]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	20
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	20

2.2.3 Preprocessing:project_subject_subcategories

In [10]:
```
sub_catogories = list(X['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.c

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-

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",
        if 'The' in j.split(): # this will split each of the catogory based on sp
            j=j.replace('The','') # if we have the words "The" we are going to re
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trail
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())
```

In [11]:
```
X['clean_subcategories'] = sub_cat_list
X.drop(['project_subject_subcategories'], axis=1, inplace=True)
X.head(2)
```

Out[11]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	20
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	20

2.2.4 New Column:digits in summary

In [12]:
```
# Creating a new column 'digits_in_summary' which contains flags of 1 for /
# 'project_resource_summary' containing numeric specification in their requiremn
project_resource_summary = []
new=[]

project_resource_summary = list(X['project_resource_summary'].values)

for i in project_resource_summary:
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(' '):
        if j.isdigit():
            new.append(1)
            break
        else:
            continue
    else:
        new.append(0)


X['digits_in_summary']=new
```

In [13]:
```
#To make best use of the memory we are setting the variable names to 'None' and p
project_resource_summary=None
new=None
new1=None
i=None
j=None
a=None

gc.collect()
gc.enable()
gc.DEBUG_SAVEALL
```

Out[13]:
```
32
```

2.2.5 Preprocessing:Text features (Project Essay's)

In [14]:
```
# merge two column text dataframe:
X["essay"] = X["project_essay_1"].map(str) +\
                        X["project_essay_2"].map(str) + \
                        X["project_essay_3"].map(str) + \
                        X["project_essay_4"].map(str)
```

In [15]:
```
X = X.drop(['project_essay_1', 'project_essay_2','project_essay_3', 'project_ess
X.shape
```

Out[15]:
```
(28000, 14)
```

2.2.6 Adding column Cost per project in dataset

In [16]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).
price_data.head(2)
type(price_data)
```

Out[16]:

```
pandas.core.frame.DataFrame
```

In [17]:

```
# join two dataframes in python:
X = pd.merge(X, price_data, on='id', how='left')
X.head(2)
```

Out[17]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	20
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	20

In [18]:

```
type(price_data)
```

Out[18]:

```
pandas.core.frame.DataFrame
```

In [19]:

```
#To make best use of the memory we are setting the variable names to 'None' and
resource_data=None
price_data=None
gc.collect()
gc.enable()
gc.DEBUG_SAVEALL
```

Out[19]:

```
32
```

2.2.7 Text Preprocessing:Essay Text

In [20]:
```
# 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 [21]:
```
# print(X['essay'].values)
```

In [22]:
```
sent = decontracted(X['essay'].values[99])
print(sent)
print("="*50)
```

```
My preschool students are children who are three to five years of age.  My scho
ol is in sunny San Pedro, California. The children from San Pedro come to schoo
l each morning ready to learn and grow.  There is never a dull moment in our cl
ass; my students are busy bees moving from one interest area to another.  They
are eager to learn, explore, and experiment with the instructional materials an
d centers I set up for them.  We need more materials for the children to engage
with, materials that will foster their interest in technology, literacy, math,
science, art, and engineering. \r\nMy student is will learn number recognition
and develop counting skills while engaging with the Learn to count picture puzz
les and number Sequencing puzzles. While building with the 3-D Magnet Builders
and Crystal Building Blocks, my student is mathematical skills will be supporte
d and strengthened in concepts such as measurement, comparison, number estimati
on, symmetry and balance. My student is will build number skills as the they si
ft and make exciting number shell discoveries with every scoop at the sand tabl
e. The sort a shape activity board will allow my youngest students to learn col
ors, shapes and sorting skills as they fit various shape pieces into place.
==================================================
```

In [23]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-bre
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
My preschool students are children who are three to five years of age.  My scho
ol is in sunny San Pedro, California. The children from San Pedro come to schoo
l each morning ready to learn and grow.  There is never a dull moment in our cl
ass; my students are busy bees moving from one interest area to another.  They
are eager to learn, explore, and experiment with the instructional materials an
d centers I set up for them.  We need more materials for the children to engage
with, materials that will foster their interest in technology, literacy, math,
science, art, and engineering.   My student is will learn number recognition an
d develop counting skills while engaging with the Learn to count picture puzzle
s and number Sequencing puzzles. While building with the 3-D Magnet Builders an
d Crystal Building Blocks, my student is mathematical skills will be supported
and strengthened in concepts such as measurement, comparison, number estimatio
n, symmetry and balance. My student is will build number skills as the they sif
t and make exciting number shell discoveries with every scoop at the sand tabl
e. The sort a shape activity board will allow my youngest students to learn col
ors, shapes and sorting skills as they fit various shape pieces into place.
```

In [24]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
My preschool students are children who are three to five years of age My school
is in sunny San Pedro California The children from San Pedro come to school eac
h morning ready to learn and grow There is never a dull moment in our class my
students are busy bees moving from one interest area to another They are eager
to learn explore and experiment with the instructional materials and centers I
set up for them We need more materials for the children to engage with material
s that will foster their interest in technology literacy math science art and e
ngineering My student is will learn number recognition and develop counting ski
lls while engaging with the Learn to count picture puzzles and number Sequencin
g puzzles While building with the 3 D Magnet Builders and Crystal Building Bloc
ks my student is mathematical skills will be supported and strengthened in conc
epts such as measurement comparison number estimation symmetry and balance My s
tudent is will build number skills as the they sift and make exciting number sh
ell discoveries with every scoop at the sand table The sort a shape activity bo
ard will allow my youngest students to learn colors shapes and sorting skills a
s they fit various shape pieces into place
```

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'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itse
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'ha
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 't
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than'
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should'v
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "d
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma'
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn
            'won', "won't", 'wouldn', "wouldn't"]
```

In [26]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(X['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 not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████| 28000/28000 [00:36<00:00, 774.21it/s]
```

In [27]:

```
# after preprocesing

# X['essay'] = None
X['essay'] = preprocessed_essays

X.head(2)
```

Out[27]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	20
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	20

In [28]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_project_title = []
# tqdm is for printing the status bar
for sentance in tqdm(X['project_title'].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 not in stopwords)
    preprocessed_project_title.append(sent.lower().strip())
```

```
100%|██████████████████████████████| 28000/28000 [00:01<00:00, 16240.37it/s]
```

In [29]:

```
preprocessed_project_title[4999]
# after preprocesing

# X['project_title'] = None
X['project_title'] = preprocessed_project_title

X.head(2)
```

Out[29]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	20
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	20

2.2.8 Splitting the data into Train and Test

In [30]:

```
# train test split(67:33)
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, strati
# X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0..
```

In [31]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/
from collections import Counter
my_counter = Counter()
for word in X_train['clean_categories'].values:
    my_counter.update(word.split())
print(my_counter)
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
cat_dict = dict(my_counter)
sorted_cat_dict_train = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

```
Counter({'Literacy_Language': 8984, 'Math_Science': 7093, 'Health_Sports': 246
0, 'SpecialNeeds': 2381, 'AppliedLearning': 2129, 'Music_Arts': 1715, 'History_
Civics': 1045, 'Warmth': 235, 'Care_Hunger': 235})
```

2.2.9 Vectorizing Categorical data: clean_categories(Project subject categories)

In [32]:

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

print("="*100)

Bow_features_names1=[]

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict_train.keys()), lowe
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on t

# 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)

#Appending the features into a variable
for cnt in vectorizer.get_feature_names() :
    Bow_features_names1.append(cnt)

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

```
(18760, 16) (18760,)
(9240, 16) (9240,)
===============================================================================
=====================
After vectorizations
(18760, 9) (18760,)
(9240, 9) (9240,)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
===============================================================================
=====================
```

2.2.10 Vectorizing Categorical data: clean_subcategories(Project subject subcategories)

In [33]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4
from collections import Counter
my_counter = Counter()
for word in X_train['clean_subcategories'].values:
    my_counter.update(word.split())
sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict_train = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[
```

In [34]:

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

print("="*100)

vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict_train.keys()), 
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only o

# we use the fitted Countvectorizer to convert the text to vector
X_train_clean_sub_cat_ohe = vectorizer.transform(X_train['clean_subcategories'].
# X_cv_clean_sub_cat_ohe = vectorizer.transform(X_cv['clean_subcategories'].value
X_test_clean_sub_cat_ohe = vectorizer.transform(X_test['clean_subcategories'].va

#Appending the features into a variable
cnt=None
for cnt in vectorizer.get_feature_names() :
    Bow_features_names1.append(cnt)

print("After vectorizations")
print(X_train_clean_sub_cat_ohe.shape, y_train.shape)
# print(X_cv_clean_sub_cat_ohe.shape, y_cv.shape)
print(X_test_clean_sub_cat_ohe.shape, y_test.shape)

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

```
(18760, 16) (18760,)
(9240, 16) (9240,)
================================================================================
====================
After vectorizations
(18760, 30) (18760,)
(9240, 30) (9240,)
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Fo
reignLanguages', 'Extracurricular', 'Civics_Government', 'Warmth', 'Care_Hunge
r', 'NutritionEducation', 'SocialSciences', 'PerformingArts', 'CharacterEducati
on', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_Geography',
'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness', 'EnvironmentalS
cience', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'L
iterature_Writing', 'Mathematics', 'Literacy']
================================================================================
====================
```

2.2.11 Vectorizing Categorical data: school_state

In [35]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4
from collections import Counter
my_counter = Counter()
for word in X_train['school_state'].values:
    my_counter.update(word.split())
school_state_dict = dict(my_counter)
sorted_school_state_dict_train = dict(sorted(school_state_dict.items(), key=lambd
# sorted_school_state_dict
```

In [36]:

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

print("="*100)


vectorizer = CountVectorizer(vocabulary=list(sorted_school_state_dict_train.keys
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train

# we use the fitted Countvectorizer to convert the text to vector
X_train_School_state_ohe = vectorizer.transform(X_train['school_state'].values)
# X_cv_School_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_School_state_ohe = vectorizer.transform(X_test['school_state'].values)

#Appending the features into a variable
cnt=None
for cnt in vectorizer.get_feature_names() :
    Bow_features_names1.append(cnt)

print("After vectorizations")
print(X_train_School_state_ohe.shape, y_train.shape)
# print(X_cv_School_state_ohe.shape, y_cv.shape)
print(X_test_School_state_ohe.shape, y_test.shape)

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

```
(18760, 16) (18760,)
(9240, 16) (9240,)
================================================================================
====================
After vectorizations
(18760, 51) (18760,)
(9240, 51) (9240,)
['VT', 'ND', 'WY', 'MT', 'RI', 'NE', 'DE', 'NH', 'AK', 'SD', 'WV', 'ME', 'DC',
'HI', 'NM', 'ID', 'IA', 'KS', 'AR', 'MN', 'CO', 'KY', 'MS', 'OR', 'MD', 'NV',
'CT', 'AL', 'UT', 'TN', 'WI', 'VA', 'AZ', 'NJ', 'MA', 'OK', 'WA', 'LA', 'MO',
'OH', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX', 'CA']
================================================================================
====================
```

2.2.12 Vectorizing Categorical data: project_grade_category

In [37]:
```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4
from collections import Counter
my_counter = Counter()
for word in X_train['project_grade_category'].values:
    my_counter.update(word.split())
project_grade_dict = dict(my_counter)
sorted_project_grade_dict_train = dict(sorted(project_grade_dict.items(), key=lan
```

In [38]:
```
print(X_train.shape, y_train.shape)
# print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)



vectorizer= CountVectorizer(vocabulary=list(sorted_project_grade_dict_train.keys
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only

# we use the fitted Countvectorizer_pro_gradeto convert the text to vector
X_train_project_grade_category_ohe = vectorizer.transform(X_train['project_grade_
# X_cv_project_grade_category_ohe = vectorizer.transform(X_cv['project_grade_cate
X_test_project_grade_category_ohe = vectorizer.transform(X_test['project_grade_ca

#Appending the features into a variable
cnt=None
for cnt in vectorizer.get_feature_names() :
    Bow_features_names1.append(cnt)

print("After vectorizations")
print(X_train_project_grade_category_ohe.shape, y_train.shape)
# print(X_cv_project_grade_category_ohe.shape, y_cv.shape)
print(X_test_project_grade_category_ohe.shape, y_test.shape)

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

```
(18760, 16) (18760,)
(9240, 16) (9240,)
================================================================================
====================
After vectorizations
(18760, 4) (18760,)
(9240, 4) (9240,)
['Grades9-12', 'Grades6-8', 'Grades3-5', 'GradesPreK-2']
================================================================================
====================
```

2.2.13 Vectorizing Categorical data: teacher_prefix

In [39]:

```
#To overcome the blanks in the teacher_prefix categry the .fillna is used
X_train['teacher_prefix']=X_train['teacher_prefix'].fillna("")
# project_data1=project_data.dropna()
```

In [40]:

```
#To overcome the blanks in the teacher_prefix categry the .fillna is used
X_test['teacher_prefix']=X_test['teacher_prefix'].fillna("")
# project_data1=project_data.dropna()
```

In [41]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4
from collections import Counter
my_counter = Counter()
my_counter1=[]
# project_data['teacher_prefix']=str(project_data['teacher_prefix'])
for word in X_train['teacher_prefix'].values:
    my_counter.update(word.split())
teacher_prefix_dict = dict(my_counter)
sorted_teacher_prefix_dict_train = dict(sorted(teacher_prefix_dict.items(), key=
# teacher_prefix_dict
```

In [42]:

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

print("="*100)

vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict_train.ke
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on tra

# we use the fitted Countvectorizer to convert the text to vector
X_train_teacher_prefix_ohe = vectorizer.transform(X_train['teacher_prefix'].valu
# X_cv_teacher_prefix_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_prefix_ohe = vectorizer.transform(X_test['teacher_prefix'].values

#Appending the features into a variable
cnt=None
for cnt in vectorizer.get_feature_names() :
    Bow_features_names1.append(cnt)

print("After vectorizations")
print(X_train_teacher_prefix_ohe.shape, y_train.shape)
# print(X_cv_teacher_prefix_ohe.shape, y_cv.shape)
print(X_test_teacher_prefix_ohe.shape, y_test.shape)

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

```
(18760, 16) (18760,)
(9240, 16) (9240,)
===============================================================================
=====================
After vectorizations
(18760, 4) (18760,)
(9240, 4) (9240,)
['Teacher', 'Mr.', 'Ms.', 'Mrs.']
===============================================================================
=====================
```

2.3 Make Data Model Ready: Vectorizing Essay and Project_title feature into BOW & TFIDF

Vectorizing Text data

2.3.1 Bag of words:Essays

In [43]:
```
print(X_train.shape, y_train.shape)
# print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

# We are considering only the words which appeared in at least 10 documents(rows

vectorizer = CountVectorizer(min_df=5,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

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

#Appending the features into a variable
cnt=None
for cnt in vectorizer.get_feature_names() :
    Bow_features_names1.append(cnt)

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)
```

```
(18760, 16) (18760,)
(9240, 16) (9240,)
==============================================================================
======================
After vectorizations
(18760, 5000) (18760,)
(9240, 5000) (9240,)
==============================================================================
======================
```

2.3.2 Bag of words:Project Title

In [44]:

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

print("="*100)


vectorizer = CountVectorizer(min_df=5,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on trai

# we use the fitted Countvectorizer to convert the text to vector
X_train_project_title_bow = vectorizer.transform(X_train['project_title'].values
# X_cv_project_title_bow = vectorizer.transform(X_cv['project_title'].values)
X_test_project_title_bow = vectorizer.transform(X_test['project_title'].values)

# print(vectorizer)
print(X_train_project_title_bow.shape)

#Appending the features into a variable
cnt=None
for cnt in vectorizer.get_feature_names() :
    Bow_features_names1.append(cnt)

print("After vectorizations")
print(X_train_project_title_bow.shape, y_train.shape)
# print(X_cv_project_title_bow.shape, y_cv.shape)
print(X_test_project_title_bow.shape, y_test.shape)

print("="*100)
```

```
(18760, 16) (18760,)
(9240, 16) (9240,)
===============================================================================
=====================
(18760, 3276)
After vectorizations
(18760, 3276) (18760,)
(9240, 3276) (9240,)
===============================================================================
=====================
```

2.3.3 TFIDF vectorizer:Essay

In [45]:

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

print("="*100)

from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(min_df=5,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted Countvectorizer to convert the text to vector
X_train_essay_Tfidf = vectorizer.transform(X_train['essay'].values)
# X_cv_essay_Tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_Tfidf = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_Tfidf.shape, y_train.shape)
# print(X_cv_essay_Tfidf.shape, y_cv.shape)
print(X_test_essay_Tfidf.shape, y_test.shape)
print("="*100)
```

```
(18760, 16) (18760,)
(9240, 16) (9240,)
===============================================================================
=====================
After vectorizations
(18760, 5000) (18760,)
(9240, 5000) (9240,)
===============================================================================
=====================
```

2.3.4 TFIDF vectorizer:Project Title

In [46]:

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

print("="*100)

# We are considering only the words which appeared in at least 10 documents(rows
vectorizer = TfidfVectorizer(min_df=5,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on trai

# we use the fitted Countvectorizer to convert the text to vector
X_train_project_title_tfidf = vectorizer.transform(X_train['project_title'].valu
# X_cv_project_title_tfidf = vectorizer.transform(X_cv['project_title'].values)
X_test_project_title_tfidf = vectorizer.transform(X_test['project_title'].values

print("After vectorizations")
print(X_train_project_title_tfidf.shape, y_train.shape)
# print(X_cv_project_title_tfidf.shape, y_cv.shape)
print(X_test_project_title_tfidf.shape, y_test.shape)
print("="*100)
```

```
================================================================================
====================
After vectorizations
(18760, 3276) (18760,)
(9240, 3276) (9240,)
================================================================================
====================
```

2.4 Make Data Model Ready: Vectorizing Numerical features

2.4.1 Vectorizing Numerical features--Price

In [47]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer_test = Normalizer()
# normalizer.fit(X_train['price'].values)
# 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.
normalizer.fit(X_train['price'].values.reshape(1,-1))
# normalizer_test.fit(X_test['price'].values.reshape(1,-1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))
# X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))


X_train_price_norm=np.reshape(X_train_price_norm,(-1,1))
X_test_price_norm=np.reshape(X_test_price_norm,(-1,1))

print("After vectorizations")

# np.reshape(X_train_price_norm,
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 vectorizations
(18760, 1) (18760,)
(9240, 1) (9240,)
===============================================================================
=====================
```

2.4.2 Vectorizing Numerical features--teacher_number_of_previously_posted_projects

In [48]:

```
from sklearn.preprocessing import Normalizer
normalizer_train = Normalizer()
normalizer_test = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# 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.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.re

X_train_teacher_number_of_previously_posted_projects_norm = normalizer.transform
# X_cv_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X
X_test_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X



X_train_teacher_number_of_previously_posted_projects_norm=np.reshape(X_train_tea
X_test_teacher_number_of_previously_posted_projects_norm=np.reshape(X_test_teach


print("After vectorizations")
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.sh
# print(X_cv_teacher_number_of_previously_posted_projects_norm.shape, y_cv.shape,
print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.sha
print("="*100)
```

```
After vectorizations
(18760, 1) (18760,)
(9240, 1) (9240,)
================================================================================
====================
```

2.4.3 Vectorizing Numerical features--digits_in_summary

In [49]:

```
X_train['digits_in_summary'].fillna(X_train['digits_in_summary'].mean(), inplace=
# X_cv['digits_in_summary'].fillna(X_cv['digits_in_summary'].mean(), inplace=True
X_test['digits_in_summary'].fillna(X_test['digits_in_summary'].mean(), inplace=Tr
```

In [50]:

```
from sklearn.preprocessing import Normalizer
normalizer_train = Normalizer()
normalizer_test = Normalizer()
# normalizer.fit(X_train['digits_in_summary'].values)
# 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.
normalizer.fit(X_train['digits_in_summary'].values.reshape(1,-1))

X_train_digits_in_summary_norm = normalizer.transform(X_train['digits_in_summary
# X_cv_digits_in_summary_norm = normalizer.transform(X_cv['digits_in_summary'].va
X_test_digits_in_summary_norm = normalizer.transform(X_test['digits_in_summary']


X_train_digits_in_summary_norm=np.reshape(X_train_digits_in_summary_norm,(-1,1))
X_test_digits_in_summary_norm=np.reshape(X_test_digits_in_summary_norm,(-1,1))

print("After vectorizations")
print(X_train_digits_in_summary_norm.shape, y_train.shape)
# print(X_cv_digits_in_summary_norm.shape, y_cv.shape)
print(X_test_digits_in_summary_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(18760, 1) (18760,)
(9240, 1) (9240,)
===============================================================================
=====================
```

In [51]:

```
preprocessed_essays=None
glove_words=None
vector=None
sent=None
stopwords=None
preprocessed_essays=None
sentence=None
my_counter=None
max_features=None
sub_cat_dict=None
sorted_sub_cat_dict=None
dictionary=None
cnt_words=None
max_features=None
min_df=None
tfidf_words=None
mo=None
project_grade_dict=None


gc.collect()
gc.enable()
gc.DEBUG_SAVEALL
```

Out[51]:

```
32
```

2.5 Merging all the above features

2.5.1 Merging all the numerical and catogorical features

In [52]:

```
#catogorical
print(X_train_teacher_prefix_ohe.shape)
print(X_train_project_grade_category_ohe.shape)
print(X_train_School_state_ohe.shape)
print(X_train_clean_sub_cat_ohe.shape)
print(X_train_clean_cat_ohe.shape)

print(X_test_teacher_prefix_ohe.shape)
print(X_test_project_grade_category_ohe.shape)
print(X_test_School_state_ohe.shape)
print(X_test_clean_sub_cat_ohe.shape)
print(X_test_clean_cat_ohe.shape)

#numerical vectors
print(X_train_price_norm.shape)
print(X_train_teacher_number_of_previously_posted_projects_norm.shape)
print(X_train_price_norm.shape)

print(X_test_price_norm.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape)
print(X_test_price_norm.shape)

#text
print(X_train_project_title_bow.shape)
print(X_train_project_title_tfidf.shape)

print(X_test_project_title_bow.shape)
print(X_test_project_title_tfidf.shape)


print(X_train_essay_bow.shape)
print(X_train_essay_Tfidf.shape)


print(X_test_essay_bow.shape)
print(X_test_essay_Tfidf.shape)
```

```
(18760, 4)
(18760, 4)
(18760, 51)
(18760, 30)
(18760, 9)
(9240, 4)
(9240, 4)
(9240, 51)
(9240, 30)
(9240, 9)
(18760, 1)
(18760, 1)
(18760, 1)
(9240, 1)
(9240, 1)
(9240, 1)
(18760, 3276)
(18760, 3276)
(9240, 3276)
```

```
(9240, 3276)
(18760, 5000)
(18760, 5000)
(9240, 5000)
(9240, 5000)
```

In [53]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense
X_BOW_TRAIN = hstack((X_train_digits_in_summary_norm,X_train_teacher_number_of_pr
X_BOW_TRAIN=X_BOW_TRAIN.todense()
X_BOW_TRAIN=np.array(X_BOW_TRAIN)

# X_BOW_cv = hstack((X_cv_project_title_bow,X_cv_essay_bow ,X_cv_digits_in_summar
# X_BOW_cv=X_BOW_cv.todense()
# X_BOW_cv=np.array(X_BOW_cv)

X_BOW_test = hstack((X_test_digits_in_summary_norm,X_test_teacher_number_of_prev
X_BOW_test=X_BOW_test.todense()
X_BOW_test=np.array(X_BOW_test)

X_Tfidf_train = hstack(( X_train_project_title_tfidf,X_train_essay_Tfidf,X_train_
X_Tfidf_train=X_Tfidf_train.todense()
X_Tfidf_train=np.array(X_Tfidf_train)

# X_Tfidf_cv = hstack(( X_cv_project_title_tfidf,X_cv_essay_Tfidf,X_cv_digits_in_
# X_Tfidf_cv=X_Tfidf_cv.todense()
# X_Tfidf_cv=np.array(X_Tfidf_cv)

X_Tfidf_test = hstack(( X_test_project_title_tfidf,X_test_essay_Tfidf,X_test_dig
X_Tfidf_test=X_Tfidf_test.todense()
X_Tfidf_test=np.array(X_Tfidf_test)


# X_avg_w2v_train = hstack(( avg_w2v_vectors_Pro_title_train,avg_w2v_vectors_ess
# X_avg_w2v_train=X_avg_w2v_train.todense()
# X_avg_w2v_train=np.array(X_avg_w2v_train)
```

In [54]:

```
X_train_project_title_bow=None
X_train_essay_bow =None
X_train_digits_in_summary_norm=None
X_train_teacher_number_of_previously_posted_projects_norm=None
X_train_price_norm=NoneX_train_teacher_prefix_ohe=None
X_train_project_grade_category_ohe=None
X_train_School_state_ohe=None
X_train_clean_sub_cat_ohe=None
X_train_clean_cat_ohe=None



# X_cv_project_title_bow=None
# X_cv_essay_bow =None
# X_cv_digits_in_summary_norm=None
# X_cv_teacher_number_of_previously_posted_projects_norm=None
# X_cv_price_norm=NoneX_cv_teacher_prefix_ohe=None
# X_cv_project_grade_category_ohe=None
# X_cv_School_state_ohe=None
# X_cv_clean_sub_cat_ohe=None
# X_cv_clean_cat_ohe=None


X_test_project_title_bow=None
X_test_essay_bow =None
X_test_digits_in_summary_norm=None
X_test_teacher_number_of_previously_posted_projects_norm=None
X_test_price_norm=NoneX_test_teacher_prefix_ohe=None
X_test_project_grade_category_ohe=None
X_test_School_state_ohe=None
X_test_clean_sub_cat_ohe=None
X_test_clean_cat_ohe=None


X_train_project_title_tfidf=None
X_train_essay_Tfidf=None
X_train_digits_in_summary_norm=None
X_train_teacher_number_of_previously_posted_projects_norm=None
X_train_price_norm=NoneX_train_teacher_prefix_ohe=None
X_train_project_grade_category_ohe=None
X_train_School_state_ohe=None
X_train_clean_sub_cat_ohe=None
X_train_clean_cat_ohe=None


# X_cv_project_title_tfidf=None
# X_cv_essay_Tfidf=None
# X_cv_digits_in_summary_norm=None
# X_cv_teacher_number_of_previously_posted_projects_norm=None
# X_cv_price_norm=None
# X_cv_teacher_prefix_ohe=None
# X_cv_project_grade_category_ohe=None
# X_cv_School_state_ohe=None
# X_cv_clean_sub_cat_ohe=None
# X_cv_clean_cat_ohe=None
```

```
X_test_project_title_tfidf=None
X_test_essay_Tfidf=None
X_test_digits_in_summary_norm=None
X_test_teacher_number_of_previously_posted_projects_norm=None
X_test_price_norm=NoneX_test_teacher_prefix_ohe=None
X_test_project_grade_category_ohe=None
X_test_School_state_ohe=None
X_test_clean_sub_cat_ohe=None
X_test_clean_cat_ohe=None


# avg_w2v_vectors_Pro_title_train=None
# avg_w2v_vectors_essay_train=None
X_train_digits_in_summary_norm=None
X_train_teacher_number_of_previously_posted_projects_norm=None
X_train_price_norm=None
X_train_teacher_prefix_ohe=None
X_train_project_grade_category_ohe=None
X_train_School_state_ohe=None
X_train_clean_sub_cat_ohe=None
X_train_clean_cat_ohe=None

# avg_w2v_vectors_Pro_title_cv=None
# avg_w2v_vectors_essay_cv=None
# X_cv_digits_in_summary_norm=None
# X_cv_teacher_number_of_previously_posted_projects_norm=None
# X_cv_price_norm=NoneX_cv_teacher_prefix_ohe=None
# X_cv_project_grade_category_ohe=None
# X_cv_School_state_ohe=None
# X_cv_clean_sub_cat_ohe=None
# X_cv_clean_cat_ohe=None


# avg_w2v_vectors_Pro_title_test=None
# avg_w2v_vectors_essay_test=None
X_test_digits_in_summary_norm=None
X_test_teacher_number_of_previously_posted_projects_norm=None
X_test_price_norm=NoneX_test_teacher_prefix_ohe=None
X_test_project_grade_category_ohe=None
X_test_School_state_ohe=None
X_test_clean_sub_cat_ohe=None
X_test_clean_cat_ohe=None


# Xtfidf_w2v_vectors_train=None
# tfidf_w2v_vectors_Pro_title_train=None
X_train_digits_in_summary_norm=None
X_train_teacher_number_of_previously_posted_projects_norm=None
X_train_price_norm=NoneX_train_teacher_prefix_ohe=None
X_train_project_grade_category_ohe=None
X_train_School_state_ohe=None
X_train_clean_sub_cat_ohe=None
X_train_clean_cat_ohe=None



# Xtfidf_w2v_vectors_cv=None
# tfidf_w2v_vectors_Pro_title_cv=None
# X_cv_digits_in_summary_norm=None
# X_cv_teacher_number_of_previously_posted_projects_norm=None
# X_cv_price_norm=NoneX_cv_teacher_prefix_ohe=None
# X_cv_project_grade_category_ohe=None
# X_cv_School_state_ohe=None
# X_cv_clean_sub_cat_ohe=None
# X_cv_clean_cat_ohe=None


# Xtfidf_w2v_vectors_test=None
# tfidf_w2v_vectors_Pro_title_test=None
X_test_digits_in_summary_norm=None
X_test_teacher_number_of_previously_posted_projects_norm=None
X_test_price_norm=NoneX_test_teacher_prefix_ohe=None
X_test_project_grade_category_ohe=None
X_test_School_state_ohe=None
X_test_clean_sub_cat_ohe=None
X_test_clean_cat_ohe=None

gc.collect()
gc.enable()
gc.DEBUG_SAVEALL
```

Out[54]:
```
32
```

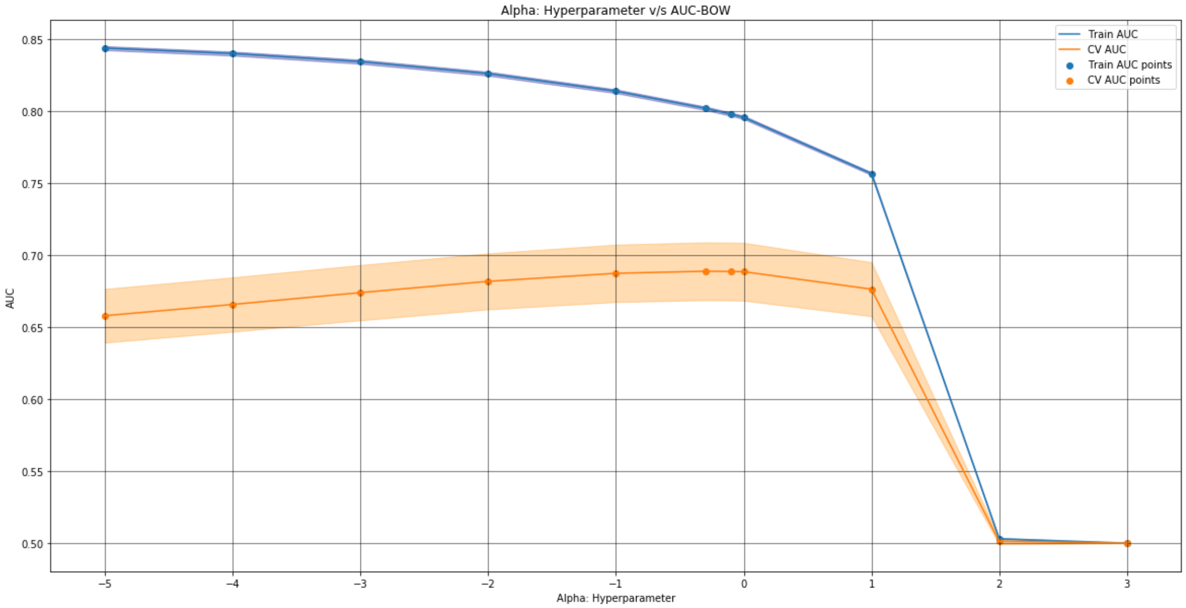
Appling Naive Bayes on all features as mentioned in the instructions

2.6 Applying Naive Bayes on `BOW`, `SET 1`

2.6.1 Applying MultinomialNB & GridSearchCV on Train data to obtain the best `aplha`

In [55]:

```
from sklearn.model_selection import GridSearchCV

from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB(class_prior=[0.5,0.5])

parameters = {'alpha':[0.00001, 0.0001,0.001, 0.01, 0.1,0.5,0.8, 1, 10,  100,  1
alpha=[*parameters.values()]
alpha=alpha[0]
log_alphas=[math.log10(num) for num in alpha]

clf = GridSearchCV(nb, parameters, cv= 10, scoring='roc_auc',return_train_score=

clf.fit(X_BOW_TRAIN, y_train)

#Selecting the best parameter
best_alpha1=clf.best_params_

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(20,10))

plt.plot(log_alphas, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,train_auc - train_auc_std,train_auc + train_au

plt.plot(log_alphas, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=

plt.scatter(log_alphas, train_auc, label='Train AUC points')
plt.scatter(log_alphas, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("Alpha: Hyperparameter")
plt.ylabel("AUC")
plt.title("Alpha: Hyperparameter v/s AUC-BOW")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

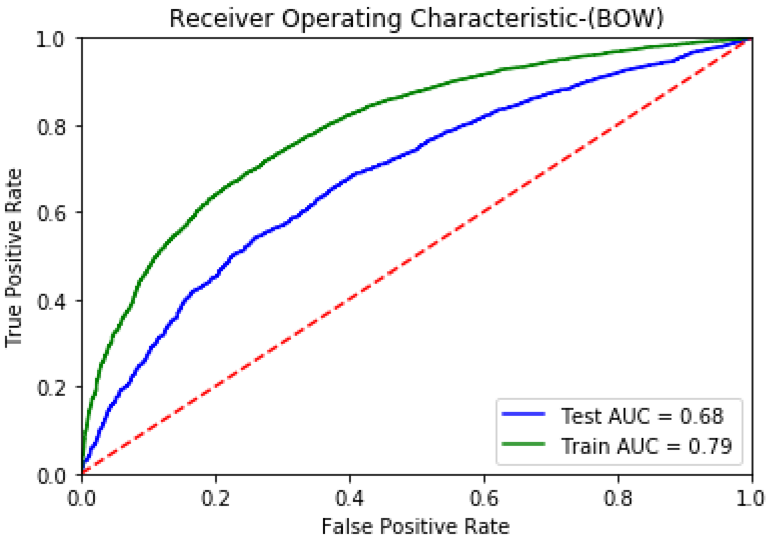


In [56]:
```
print("The best alpha from the above graph is ",best_alpha1)
# best_alpha1['alpha']
```

```
The best alpha from the above graph is  {'alpha': 0.5}
```

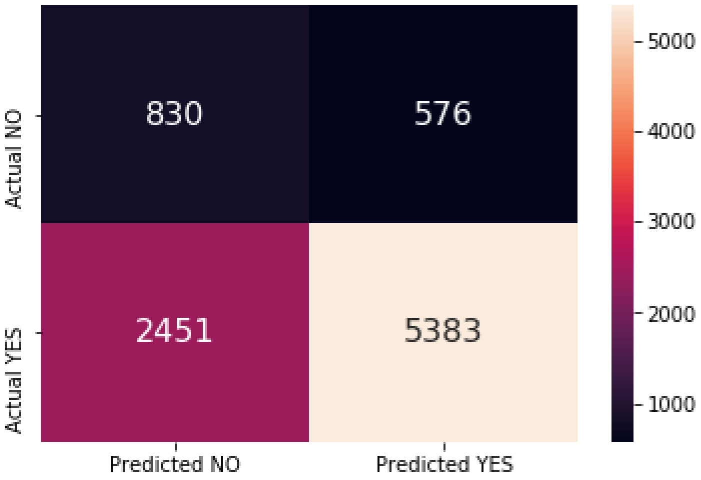
2.6.2 Receiver Operating Characteristic- (BOW)

In [57]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htm
from sklearn.metrics import roc_curve, auc

nb_bow = MultinomialNB(alpha = best_alpha1['alpha'],class_prior=[0.5,0.5])

nb_bow.fit(X_BOW_TRAIN, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
# not the predicted outputs

probs = nb_bow.predict_proba(X_BOW_test)
# print(len(probs[:,1]))
probs1 = nb_bow.predict_proba(X_BOW_TRAIN)
# print(len(probs1[:,1]))
preds = probs[:,1]
# print(preds)
preds1 = probs1[:,1]
# print(preds1)
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
# print(fpr,tpr,threshold)
fpr1, tpr1, threshold = metrics.roc_curve(y_train, preds1)
# print(fpr1,tpr1,threshold)
roc_auc = metrics.auc(fpr, tpr)
# print(roc_auc)
roc_auc1 = metrics.auc(fpr1, tpr1)
# print(roc_auc1)

# https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic-(BOW)')
plt.plot(fpr, tpr, 'b', label = 'Test AUC = %0.2f' % roc_auc)
plt.plot(fpr1, tpr1, 'g', label = 'Train AUC = %0.2f' % roc_auc1)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



2.6.3 Confusion matrix

In [58]:
```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#function to get heatmap confusion matrix
def get_confusion_matrix(clf,X_te,y_test):
    y_pred = clf.predict(X_te)
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), index =['Actual NO','
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')

# %%time
get_confusion_matrix(nb_bow,X_BOW_test,y_test)
```

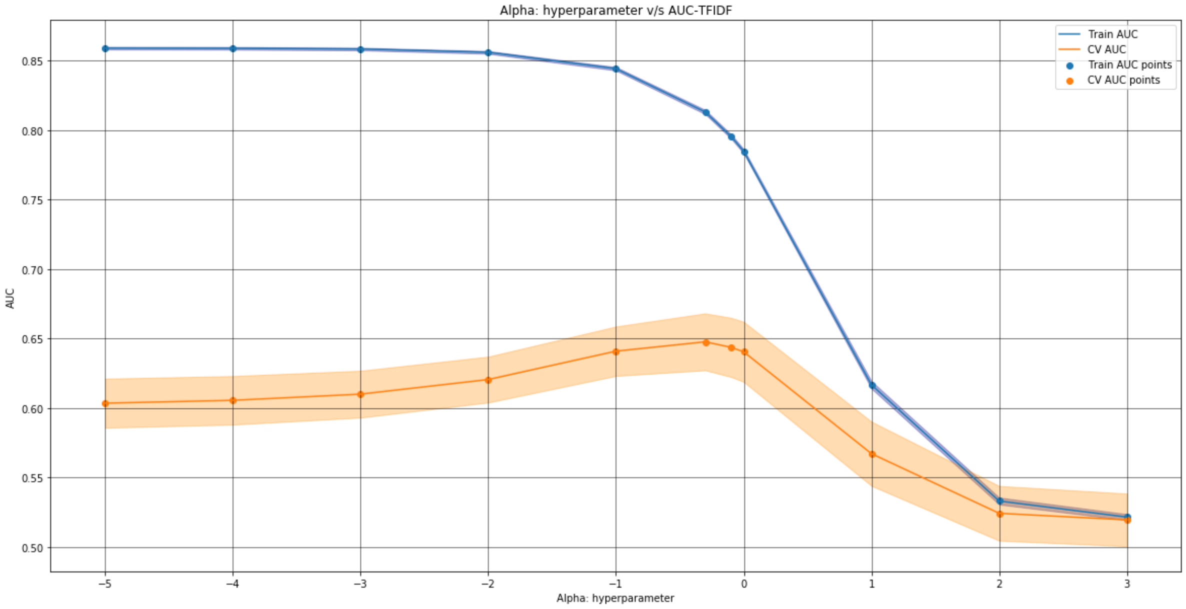


2.7 Applying Naive Bayes on `TFIDF` , `SET 2`

2.7.1 Applying MultinomialNB & GridSearchCV on Train data to obtain the best `aplha`

In [59]:

```
from sklearn.model_selection import GridSearchCV

from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB(class_prior=[0.5,0.5])

parameters = {'alpha':[0.00001, 0.0001,0.001, 0.01, 0.1,0.5,0.8, 1, 10,  100,  1
alpha=[*parameters.values()]
alpha=alpha[0]
log_alphas=[math.log10(num) for num in alpha]

clf = GridSearchCV(nb, parameters, cv= 10, scoring='roc_auc',return_train_score=

clf.fit(X_Tfidf_train, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

#Selecting the best parameter
best_alpha2=clf.best_params_

plt.figure(figsize=(20,10))

plt.plot(log_alphas, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,train_auc - train_auc_std,train_auc + train_au

plt.plot(log_alphas, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=

plt.scatter(log_alphas, train_auc, label='Train AUC points')
plt.scatter(log_alphas, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Alpha: hyperparameter v/s AUC-TFIDF")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

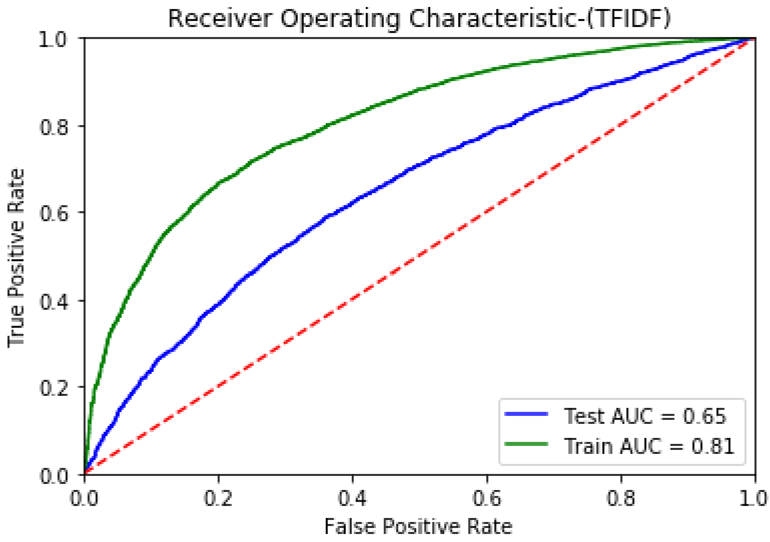


In [60]:
```
# best_alpha1=0.01
print("The best alpha from the above graph is ",best_alpha2['alpha'])
```

```
The best alpha from the above graph is  0.5
```

2.7.2 Receiver Operating Characteristic- (TFIDF)

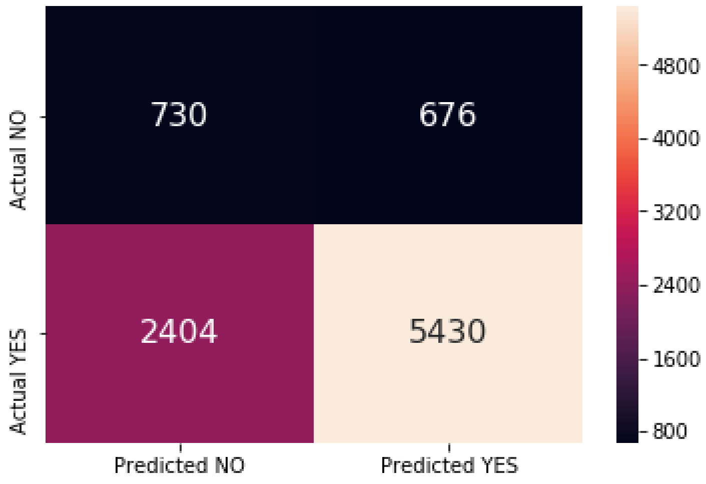
In [61]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htm
from sklearn.metrics import roc_curve, auc

nb_Tfidf = MultinomialNB(alpha = best_alpha2['alpha'],class_prior=[0.5,0.5])

nb_Tfidf.fit(X_Tfidf_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
# not the predicted outputs

probs = nb_Tfidf.predict_proba(X_Tfidf_test)
# print(len(probs[:,1]))
probs1 = nb_Tfidf.predict_proba(X_Tfidf_train)
# print(len(probs1[:,1]))
preds = probs[:,1]
# print(preds)
preds1 = probs1[:,1]
# print(preds1)
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
# print(fpr,tpr,threshold)
fpr1, tpr1, threshold = metrics.roc_curve(y_train, preds1)
# print(fpr1,tpr1,threshold)
roc_auc = metrics.auc(fpr, tpr)
# print(roc_auc)
roc_auc1 = metrics.auc(fpr1, tpr1)
# print(roc_auc1)

# https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic-(TFIDF)')
plt.plot(fpr, tpr, 'b', label = 'Test AUC = %0.2f' % roc_auc)
plt.plot(fpr1, tpr1, 'g', label = 'Train AUC = %0.2f' % roc_auc1)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



2.7.3 Confusion matrix

In [62]:

```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#function to get heatmap confusion matrix
def get_confusion_matrix(clf,X_te,y_test):
    y_pred = clf.predict(X_te)
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), index =['Actual NO','
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')

# %%time
get_confusion_matrix(nb_Tfidf,X_Tfidf_test,y_test)
```



2.8 Inserting the Numerical feature names into the list of all feature names post Vectorization.

In [63]:

```
Bow_features_names1.append("price")

Bow_features_names1.append("prev_proposed_projects")

Bow_features_names1.append("digits_in_summary")

# len(pos_class_prob_sorted)
```

In [64]:

```
len(Bow_features_names1)
```

Out[64]:

```
8377
```

2.8.1 Finding the Top 20 features in our model for `Positive class` -BOW

In [65]:

```
pos_class_prob_sorted = nb_bow.feature_log_prob_[1, :].argsort()[::-1]
for i in pos_class_prob_sorted[:20]:
    print(Bow_features_names1[i])

# len(pos_class_prob_sorted)
```

```
production
meeting needs
flexible minds
code
something new
special needs
stellar
getting ready
food
chromebooks classroom
all hands
elmo
treasure
for the
life skills
watch us
the move
discovering
students self
prefer
```

2.8.2 Finding the Top 20 features in our model for `Negative class` -BOW

In [66]:

```
pos_class_prob_sorted = nb_bow.feature_log_prob_[0, :].argsort()[::-1]
for i in pos_class_prob_sorted[:20]:
    print(Bow_features_names1[i])
```

```
production
meeting needs
code
flexible minds
something new
chromebooks classroom
getting ready
all hands
stellar
special needs
food
elmo
treasure
for the
watch us
still need
life skills
discovering
engaging books
prefer
```

3. Conclusions

In [68]:

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

x = PrettyTable()
x.field_names = ["Vectorizer","Model", "Hyper Parameter", "Test-AUC"]
x.add_row(["BOW","Naive Bayes", '0.5', 0.68])
x.add_row(["Tfidf","Naive Bayes" ,'0.5', 0.65])
# x.add_row(["avg_w2v","KNN-Brute", 41,0.66, 0.48])
# x.add_row(["tfidf_w2v","KNN-Brute", 41,0.69, 0.55])
# x.add_row(["Tfidf_K_Best","KNN-Brute", 41,0.67, 0.51])
print(x)
```

```
+------------+-------------+-----------------+----------+
| Vectorizer |    Model    | Hyper Parameter | Test-AUC |
+------------+-------------+-----------------+----------+
|    BOW     | Naive Bayes |       0.5       |   0.68   |
|   Tfidf    | Naive Bayes |       0.5       |   0.65   |
+------------+-------------+-----------------+----------+
```

In [ ]:

In [ ]:

In [ ]: