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

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 [ ]:
```
# %load_ext memory_profiler
# System Specification:i3 prcessesor 6GB RAM

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(10000)
project_data_0=project_data.tail(5000)
project_data_1=project_data_1.append(project_data_0)
project_data=project_data_1
resource_data = pd.read_csv('resources.csv')
```

In [ ]:
```
# #Sorting them by columns to spread the zeros and one's unevenly in the 'project
# project_data_1.sort_values(by=['project_essay_1'])
# project_data_1.sort_values(by=['project_essay_2'], ascending=False)
# project_data_1.sort_values(by=['project_essay_3'])
# project_data_1.sort_values(by=['project_essay_4'], ascending=False)
```

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

# project_data_1=None
# project_data_0=None
# s=None
# gc.collect()
# gc.enable()
# gc.DEBUG_SAVEALL
```

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 (15000, 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

project_data=None

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

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

2.2 Getting the Data Model Ready:encoding categorical features and numerical 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.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-

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())
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

In [10]:

```
# 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['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 = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

```
Counter({'Literacy_Language': 7178, 'Math_Science': 5769, 'Health_Sports': 193
6, 'SpecialNeeds': 1889, 'AppliedLearning': 1675, 'Music_Arts': 1425, 'History_
Civics': 801, 'Warmth': 178, 'Care_Hunger': 178})
```

2.2.3 Preprocessing:project_subject_subcategories

In [11]:

```
sub_catogories = list(X['project_subject_subcategories'].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-

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 [12]:

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

Out[12]:

	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 [14]:
```
# 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 [15]:
```
#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[15]:
```
32
```

2.2.5 Preprocessing:Text features (Project Essay's)

In [16]:
```
# 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 [17]:
```
X = X.drop(['project_essay_1', 'project_essay_2','project_essay_3', 'project_ess
X.shape
```

Out[17]:
```
(15000, 14)
```

2.2.6 Adding column Cost per project in dataset

In [18]:
```
# 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)
```

Out[18]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [19]:
```
# join two dataframes in python:
X = pd.merge(X, price_data, on='id', how='left')
X.head(2)
```

Out[19]:

	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 [20]:
```
#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[20]:
```
32
```

2.2.7 Text Preprocessing:Essay Text

In [21]:
```
# 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 [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%|██████████████████████████████████████████| 15000/15000 [00:48<00:00, 306.96it/s]
```

In [27]:

```
# after preprocesing
preprocessed_essays[4999]
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%|██████████| 15000/15000 [00:01<00:00, 10358.52it/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.
```

2.2.9 Vectorizing Categorical data: clean_categories(Project subject categories)

In [31]:

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

print("="*100)


from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=
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)

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)
```

```
(10050, 16) (10050,)
(4950, 16) (4950,)
===============================================================================
=====================
After vectorizations
(10050, 9) (10050,)
(4950, 9) (4950,)
['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 [32]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/
from collections import Counter
my_counter = Counter()
for word in X['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]))
```

In [33]:

```
# we use count vectorizer to convert the values into one hot encoded features
```

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.keys()), lowerca
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

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("="*100)
```

```
(10050, 16) (10050,)
(4950, 16) (4950,)
================================================================================
====================
After vectorizations
(10050, 30) (10050,)
(4950, 30) (4950,)
['Economics', 'FinancialLiteracy', 'CommunityService', 'ParentInvolvement', 'Ex
tracurricular', 'ForeignLanguages', 'Civics_Government', 'NutritionEducation',
'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'CharacterEducatio
n', 'Other', 'TeamSports', 'College_CareerPrep', 'Music', 'History_Geography',
'ESL', 'EarlyDevelopment', 'Health_LifeScience', '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['school_state'].values:
    my_counter.update(word.split())
school_state_dict = dict(my_counter)
sorted_school_state_dict = dict(sorted(school_state_dict.items(), key=lambda kv:
# 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.keys()), l
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)

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("="*100)
```

```
(10050, 16) (10050,)
(4950, 16) (4950,)
===============================================================================
=====================
After vectorizations
(10050, 51) (10050,)
(4950, 51) (4950,)
['VT', 'WY', 'ND', 'MT', 'RI', 'DE', 'NE', 'NH', 'SD', 'AK', 'ME', 'DC', 'WV',
'HI', 'NM', 'KS', 'IA', 'ID', 'AR', 'MN', 'CO', 'MS', 'OR', 'NV', 'KY', 'MD',
'AL', 'TN', 'CT', 'UT', 'WI', 'VA', 'AZ', 'WA', 'NJ', 'MA', 'OK', 'IN', 'MO',
'LA', 'OH', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'TX', 'NY', '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['project_grade_category'].values:
    my_counter.update(word.split())
project_grade_dict = dict(my_counter)
sorted_project_grade_dict = dict(sorted(project_grade_dict.items(), key=lambda k
```

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.keys()), l
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only

# we use the fitted CountVectorizer to 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

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("="*100)
```

```
(10050, 16) (10050,)
(4950, 16) (4950,)
================================================================================
====================
After vectorizations
(10050, 4) (10050,)
(4950, 4) (4950,)
['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['teacher_prefix']=X['teacher_prefix'].fillna("")
# project_data1=project_data.dropna()
```

In [40]:

```
# 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['teacher_prefix'].values:
    my_counter.update(word.split())
teacher_prefix_dict = dict(my_counter)
sorted_teacher_prefix_dict = dict(sorted(teacher_prefix_dict.items(), key=lambda
# teacher_prefix_dict
```

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)



vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict.keys()),
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

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("="*100)
```

```
(10050, 16) (10050,)
(4950, 16) (4950,)
===============================================================================
=====================
After vectorizations
(10050, 4) (10050,)
(4950, 4) (4950,)
['Teacher', 'Mr.', 'Ms.', 'Mrs.']
===============================================================================
=====================
```

2.3 Make Data Model Ready: encoding essay, and project_title

Vectorizing Text data

2.3.1 Bag of words:Essays

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)

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

vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=1000)
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)

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)
```

```
(10050, 16) (10050,)
(4950, 16) (4950,)
===============================================================================
=====================
After vectorizations
(10050, 1000) (10050,)
(4950, 1000) (4950,)
===============================================================================
=====================
```

2.3.2 Bag of words:Project Title

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)


vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=1000)
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("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)
```

```
(10050, 16) (10050,)
(4950, 16) (4950,)
===============================================================================
=====================
After vectorizations
(10050, 932) (10050,)
(4950, 932) (4950,)
===============================================================================
=====================
```

2.3.3 TFIDF vectorizer:Essay

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)

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=2500)
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)
```

```
(10050, 16) (10050,)
(4950, 16) (4950,)
===============================================================================
=====================
After vectorizations
(10050, 2500) (10050,)
(4950, 2500) (4950,)
===============================================================================
=====================
```

2.3.4 TFIDF vectorizer:Project Title

In [45]:

```
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=10,ngram_range=(1,4), max_features=2500)
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)
```

```
(10050, 16) (10050,)
(4950, 16) (4950,)
===============================================================================
=====================
After vectorizations
(10050, 932) (10050,)
(4950, 932) (4950,)
===============================================================================
=====================
```

2.3.5 Using Pretrained Models: Avg W2V-Essays

In [46]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to
# make sure you have the glove_vectors file
import pickle
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [47]:

```
import sys
sys.getsizeof(glove_words)
```

Out[47]:

```
2097376
```

In [48]:

```
# average Word2Vec
# compute average word2vec for each review.
from tqdm import tqdm
avg_w2v_vectors_essay = []; # the avg-w2v for each sentence/review is stored in t
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_essay.append(vector)

print(len(avg_w2v_vectors_essay))
print(len(avg_w2v_vectors_essay[0]))
```

```
100%|██████████████████████████████████| 15000/15000 [00:17<00:00, 875.04it/s]

15000
300
```

In [49]:

```
# train test split
from sklearn.model_selection import train_test_split

avg_w2v_vectors_essay_train, avg_w2v_vectors_essay_test, y_train, y_test = train_

# avg_w2v_vectors_essay_train, avg_w2v_vectors_essay_cv, y_train, y_cv = train_te
```

In [50]:

```
# # We are considering only the words which appeared in at least 10 documents(row
# vectorizer = CountVectorizer(min_df=10,max_features=1000)
# project_essay_avg_w2v = vectorizer.fit_transform(avg_w2v_vectors_essay)
# print("Shape of matrix after one hot encodig ",project_essay_avg_w2v.shape)


import scipy
avg_w2v_vectors_essay_train=scipy.sparse.csr_matrix(avg_w2v_vectors_essay_train)
type(avg_w2v_vectors_essay_train)

# import scipy
# avg_w2v_vectors_essay_cv=scipy.sparse.csr_matrix(avg_w2v_vectors_essay_cv)
# type(avg_w2v_vectors_essay_cv)

import scipy
avg_w2v_vectors_essay=scipy.sparse.csr_matrix(avg_w2v_vectors_essay_test)
type(avg_w2v_vectors_essay_test)
```

Out[50]:

```
list
```

2.3.6 Using Pretrained Models: AVG W2V on project_title

In [51]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_Pro_title = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(preprocessed_project_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_Pro_title.append(vector)

print(len(avg_w2v_vectors_Pro_title))
print(len(avg_w2v_vectors_Pro_title[0]))
```

```
100%|██████████████████████████████| 15000/15000 [00:00<00:00, 23290.60it/s]

15000
300
```

In [52]:

```
# train test split
from sklearn.model_selection import train_test_split

avg_w2v_vectors_Pro_title_train, avg_w2v_vectors_Pro_title_test, y_train, y_test

# avg_w2v_vectors_Pro_title_train, avg_w2v_vectors_Pro_title_cv, y_train, y_cv =
```

In [53]:

```
import scipy
avg_w2v_vectors_Pro_title_train=scipy.sparse.csr_matrix(avg_w2v_vectors_Pro_titl
type(avg_w2v_vectors_Pro_title_train)

import scipy
avg_w2v_vectors_Pro_title_test=scipy.sparse.csr_matrix(avg_w2v_vectors_Pro_title_
type(avg_w2v_vectors_Pro_title_test)

# import scipy
# avg_w2v_vectors_Pro_title_cv=scipy.sparse.csr_matrix(avg_w2v_vectors_Pro_title_
# type(avg_w2v_vectors_Pro_title_cv)
```

Out[53]:

```
scipy.sparse.csr.csr_matrix
```

2.3.7 Using Pretrained Models: TFIDF weighted W2V-Essay

In [54]:

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

In [55]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_essay = []; # the avg-w2v for each sentence/review is stored i
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf val
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_essay.append(vector)

print(len(tfidf_w2v_vectors_essay))
print(len(tfidf_w2v_vectors_essay[0]))
```

```
100%|██████████████████████████████| 15000/15000 [02:15<00:00, 110.87it/s]

15000
300
```

In [56]:
```
# train test split
from sklearn.model_selection import train_test_split

Xtfidf_w2v_vectors_train, Xtfidf_w2v_vectors_test, y_train, y_test = train_test_

# Xtfidf_w2v_vectors_train, Xtfidf_w2v_vectors_cv, y_train, y_cv = train_test_sp
```

In [57]:
```
import scipy
Xtfidf_w2v_vectors_train=scipy.sparse.csr_matrix(Xtfidf_w2v_vectors_train)
type(Xtfidf_w2v_vectors_train)

Xtfidf_w2v_vectors_test=scipy.sparse.csr_matrix(Xtfidf_w2v_vectors_test)
type(Xtfidf_w2v_vectors_test)

# Xtfidf_w2v_vectors_cv=scipy.sparse.csr_matrix(Xtfidf_w2v_vectors_cv)
# type(Xtfidf_w2v_vectors_cv)
```

Out[57]:
```
scipy.sparse.csr.csr_matrix
```

2.3.8 Using Pretrained Models: TFIDF weighted W2V on project_title

In [58]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_project_title)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [59]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_Pro_title = []; # the avg-w2v for each sentence/review is stor
for sentence in tqdm(preprocessed_project_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf val
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_Pro_title.append(vector)

print(len(tfidf_w2v_vectors_Pro_title))
print(len(tfidf_w2v_vectors_Pro_title[0]))
```

```
100%|██████████| 15000/15000 [00:01<00:00, 10224.37it/s]

15000
300
```

In [60]:
```
# train test split
from sklearn.model_selection import train_test_split

tfidf_w2v_vectors_Pro_title_train, tfidf_w2v_vectors_Pro_title_test, y_train, y_

# tfidf_w2v_vectors_Pro_title_train, tfidf_w2v_vectors_Pro_title_cv, y_train, y_
```

In [61]:
```
import scipy
tfidf_w2v_vectors_Pro_title_train=scipy.sparse.csr_matrix(tfidf_w2v_vectors_Pro_
type(tfidf_w2v_vectors_Pro_title_train)

tfidf_w2v_vectors_Pro_title_test=scipy.sparse.csr_matrix(tfidf_w2v_vectors_Pro_t
type(tfidf_w2v_vectors_Pro_title_test)

# tfidf_w2v_vectors_Pro_title_cv=scipy.sparse.csr_matrix(tfidf_w2v_vectors_Pro_t
# type(tfidf_w2v_vectors_Pro_title_cv)
```

Out[61]:
```
scipy.sparse.csr.csr_matrix
```

In [62]:
```
#To make best use of the memory we are setting the variable names to 'None' and
```

2.4 Make Data Model Ready: encoding Numerical features

2.4.1 Vectorizing Numerical features--Price

In [63]:

```
X_train['price'].shape
```

Out[63]:

```
(10050,)
```

In [64]:

```
from sklearn.preprocessing import Normalizer
normalizer = 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))

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))

print("After vectorizations")
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
(10050, 1) (10050,)
(4950, 1) (4950,)
===============================================================================
=====================
```

2.4.2 Vectorizing Numerical features--teacher_number_of_previously_posted_projects

In [65]:
```
from sklearn.preprocessing import Normalizer
normalizer = 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_test_teacher_number_of_previously_posted_projects_norm = normalizer.transform(

print("After vectorizations")
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.s
# 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
(10050, 1) (10050,)
(4950, 1) (4950,)
================================================================================
====================
```

2.4.3 Vectorizing Numerical features--digits_in_summary

In [66]:
```
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=T
```

In [67]:

```
from sklearn.preprocessing import Normalizer
normalizer = 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'].v
X_test_digits_in_summary_norm = normalizer.transform(X_test['digits_in_summary']

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
(10050, 1) (10050,)
(4950, 1) (4950,)
===============================================================================
=====================
```

In [68]:

```
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[68]:

```
32
```

2.5 Merging all the above features

2.5.1 we need to merge all the numerical vectors, catogorical features

In [69]:

```
#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)

#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)

#text
print(X_train_project_title_bow.shape)
print(X_train_project_title_tfidf.shape)
print(avg_w2v_vectors_Pro_title_train.shape)
print(avg_w2v_vectors_essay_train.shape)

print(X_train_essay_bow.shape)
print(X_train_essay_Tfidf.shape)
print(Xtfidf_w2v_vectors_train.shape)
print(tfidf_w2v_vectors_Pro_title_train.shape)

# type(digits_in_summary_standardized)
```

```
(10050, 4)
(10050, 4)
(10050, 51)
(10050, 30)
(10050, 9)
(10050, 1)
(10050, 1)
(10050, 1)
(10050, 932)
(10050, 932)
(10050, 300)
(10050, 300)
(10050, 1000)
(10050, 2500)
(10050, 300)
(10050, 300)
```

In [70]:

```
# 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_project_title_bow,X_train_essay_bow ,X_train_digit
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_summa
# X_BOW_cv=X_BOW_cv.todense()
# X_BOW_cv=np.array(X_BOW_cv)

X_BOW_test = hstack((X_test_project_title_bow,X_test_essay_bow ,X_test_digits_in
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_essay
X_avg_w2v_train=X_avg_w2v_train.todense()
X_avg_w2v_train=np.array(X_avg_w2v_train)

# X_avg_w2v_cv = hstack(( avg_w2v_vectors_Pro_title_cv,avg_w2v_vectors_essay_cv,
# X_avg_w2v_cv=X_avg_w2v_cv.todense()
# X_avg_w2v_cv=np.array(X_avg_w2v_cv)

X_avg_w2v_test = hstack(( avg_w2v_vectors_Pro_title_test,avg_w2v_vectors_essay_t
X_avg_w2v_test=X_avg_w2v_test.todense()
X_avg_w2v_test=np.array(X_avg_w2v_test)


X_tfidf_w2v_train = hstack((Xtfidf_w2v_vectors_train,tfidf_w2v_vectors_Pro_title
X_tfidf_w2v_train=X_tfidf_w2v_train.todense()
X_tfidf_w2v_train=np.array(X_tfidf_w2v_train)

# X_tfidf_w2v_cv = hstack((Xtfidf_w2v_vectors_cv,tfidf_w2v_vectors_Pro_title_cv 
# X_tfidf_w2v_cv=X_tfidf_w2v_cv.todense()
# X_tfidf_w2v_cv=np.array(X_tfidf_w2v_cv)

X_tfidf_w2v_test = hstack((Xtfidf_w2v_vectors_test,tfidf_w2v_vectors_Pro_title_t
X_tfidf_w2v_test=X_tfidf_w2v_test.todense()
X_tfidf_w2v_test=np.array(X_tfidf_w2v_test)
# X_All = hstack((categories_one_hot,sub_categories_one_hot,school_state_one_hot
```

In [71]:

```
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[71]: 32
```

2.5.2 To overcome memory error we are saving the variables to the disk.

```
In [73]: # import pickle

# # To overcome memory error we are saving the variables to the disk.
with open('X_BOW_TRAIN.pickle', 'wb') as f:
    pickle.dump(X_BOW_TRAIN, f)

with open('X_BOW_test.pickle', 'wb') as f:
    pickle.dump(X_BOW_test, f)

with open('X_Tfidf_train.pickle', 'wb') as f:
    pickle.dump(X_Tfidf_train, f)

with open('X_Tfidf_test.pickle', 'wb') as f:
    pickle.dump(X_Tfidf_test, f)

with open('X_avg_w2v_train.pickle', 'wb') as f:
    pickle.dump(X_avg_w2v_train, f)

with open('X_avg_w2v_test.pickle', 'wb') as f:
    pickle.dump(X_avg_w2v_test, f)

with open('X_tfidf_w2v_train.pickle', 'wb') as f:
    pickle.dump(X_tfidf_w2v_train, f)

with open('X_tfidf_w2v_test.pickle', 'wb') as f:
    pickle.dump(X_tfidf_w2v_test, f)
```

In [74]:

```

# with open('X_tfidf_w2v_cv.pickle', 'wb') as f:
#     pickle.dump(X_tfidf_w2v_cv, f)

# with open('X_avg_w2v_cv.pickle', 'wb') as f:
#     pickle.dump(X_avg_w2v_cv, f)

# with open('X_Tfidf_cv.pickle', 'wb') as f:
#     pickle.dump(X_Tfidf_cv, f)

# with open('X_BOW_cv.pickle', 'wb') as f:
#     pickle.dump(X_BOW_cv, f)
```

In [75]:

```
# X_tfidf_w2v_cv=None
# X_avg_w2v_cv=None
# X_Tfidf_cv=None
# X_BOW_cv=None
# gc.collect()
# gc.enable()
# gc.DEBUG_SAVEALL
```

In [76]:

```
#To make best use of the memory we are setting the variable names to 'None' and
X_BOW_TRAIN=None
X_BOW_test=None
X_Tfidf_train=None
X_Tfidf_test=None
X_avg_w2v_train=None
X_avg_w2v_test=None
X_tfidf_w2v_train=None
X_tfidf_w2v_test=None


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

Out[76]:

```
32
```

In [77]:

```

# # %%time
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.metrics import classification_report,confusion_matrix
# knn=KNeighborsClassifier(algorithm='brute',n_neighbors=6)
# knn.fit(X_BOW_train,y_BOW_train)
# pred=knn.predict(X_BOW_test)
# print(confusion_matrix(y_BOW_test,pred))
# print(classification_report(y_BOW_test,pred))
```

Appling KNN on different kind of featurization as mentioned in the instructions

2.6 Applying KNN brute force on BOW SET 1

2.6 Applying KNN brute force on BOW, SET 1

2.6.1 Reading the pickle file from the disk drive

In [78]:
```
import pickle

with open('X_BOW_TRAIN.pickle', 'rb') as f:
    X_BOW_TRAIN = pickle.load(f)
# gc.collect()

with open('X_BOW_test.pickle', 'rb') as f:
    X_BOW_test = pickle.load(f)
# gc.collect()
```

2.6.2 Simple `upsampling` on training dataset to ovecome the imbalance in the data

In [80]:
```
# We are performing simple sampling to overcome the imbalance in our train data.
# We are trying to match the minority class 0's with 1's with a difference of 20

from imblearn.over_sampling import RandomOverSampler
from collections import Counter

ros = RandomOverSampler(sampling_strategy='float',ratio=.80,random_state=20)
print('Original dataset shape %s' % Counter(y_train))
X_BOW_TRAIN, y_BOW_train = ros.fit_resample(X_BOW_TRAIN, y_train)
print('Post sampling dataset shape %s' % Counter(y_BOW_train))

print(X_BOW_TRAIN.shape, y_BOW_train.shape)
# print(X_BOW_cv.shape, y_cv.shape)
print(X_BOW_test.shape, y_test.shape)

print("="*100)
```

```
Original dataset shape Counter({1: 8526, 0: 1524})
Post sampling dataset shape Counter({1: 8526, 0: 6820})
(15346, 2033) (15346,)
(4950, 2033) (4950,)
===============================================================================
=====================
```

In [ ]:
```
# # %%time
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.metrics import classification_report,confusion_matrix
# knn=KNeighborsClassifier(algorithm='brute',n_neighbors=6)
# knn.fit(X_BOW_train,y_BOW_train)
# pred=knn.predict(X_BOW_test)
# print(confusion_matrix(y_BOW_test,pred))
# print(classification_report(y_BOW_test,pred))
```

2.6.3 Hyper Parameter Tuning to find the best K using GridSearchCV. (`BOW`)

In [125]:

```
# Method I (For loop)
```

In [ ]:

```
# def batch_predict(clf, data):
#     # roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
#     # not the predicted outputs

#     y_data_pred = []
#     tr_loop = data.shape[0] - data.shape[0]%1000
#     # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 4904
#     # in this for loop we will iterate unti the last 1000 multiplier
#     for i in range(0, tr_loop, 1000):
#         y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
#     # we will be predicting for the last data points
#     if data.shape[0]%1000 !=0:
#         y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

#     return y_data_pred
```

In [ ]:

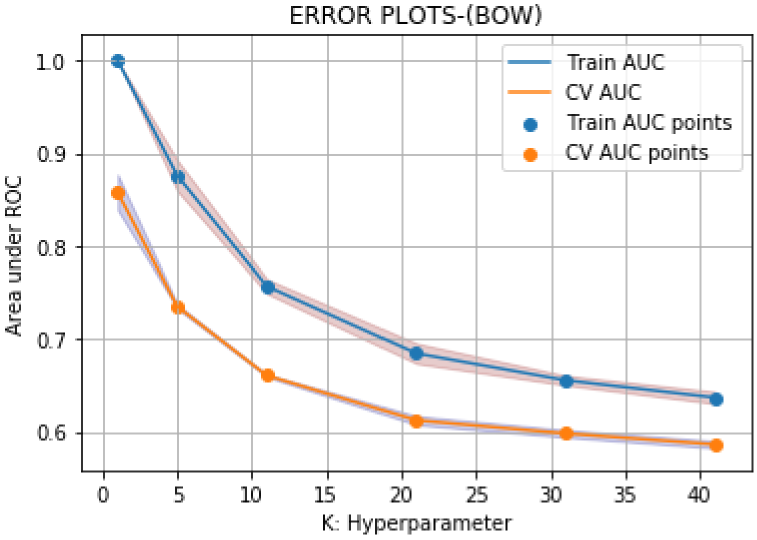
```
# import matplotlib.pyplot as plt
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.metrics import roc_auc_score
# """
# y_true : array, shape = [n_samples] or [n_samples, n_classes]
# True binary labels or binary label indicators.

# y_score : array, shape = [n_samples] or [n_samples, n_classes]
# Target scores, can either be probability estimates of the positive class, conf
# decisions (as returned by “decision_function” on some classifiers).
# For binary y_true, y_score is supposed to be the score of the class with greate

# """

# train_auc = []
# cv_auc = []
# K = [1, 5, 11, 21, 31, 41]
# for i in tqdm(K):
#     neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
#     neigh.fit(X_BOW_TRAIN, y_BOW_train)

#     y_train_pred = batch_predict(neigh, X_BOW_TRAIN)
#     y_cv_pred = batch_predict(neigh, X_BOW_cv)

#     # roc_auc_score(y_true, y_score) the 2nd parameter should be probability es
#     # not the predicted outputs
#     train_auc.append(roc_auc_score(y_BOW_train,y_train_pred))
#     cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

# plt.plot(K, train_auc, label='Train AUC')
# plt.plot(K, cv_auc, label='CV AUC')

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

# plt.legend()
# plt.xlabel("K: Hyperparameter")
# plt.ylabel("Area under ROC")
# plt.title("ERROR PLOTS-(BOW)")
# plt.grid()
# plt.show()
```

In [127]:

```
#Method II (GridSearchCV)
```

In [81]:

```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridS
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score


neigh = KNeighborsClassifier(algorithm='brute',n_jobs=-1)
parameters = {'n_neighbors':[1, 5, 11, 21, 31, 41]}
clf1 = GridSearchCV(neigh, parameters, cv=2, scoring='roc_auc', n_jobs=-1,return_
clf1.fit(X_BOW_TRAIN,y_BOW_train)

train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_

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

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: Hyperparameter")
plt.ylabel("Area under ROC")
plt.title("ERROR PLOTS-(BOW)")
plt.grid()
plt.show()
#To make best use of the memory we are setting the variable names to 'None' and p
```

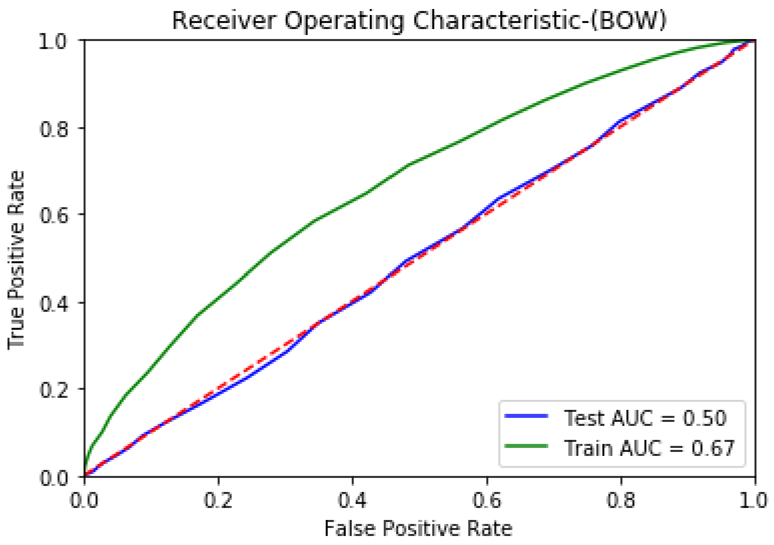


```
Wall time: 18min 7s
```

2.6.4 Plotting the `ROC curve` on both Test and Train data and obtaining the AUC on the test data using the best K- `(BOW)`

In [82]:

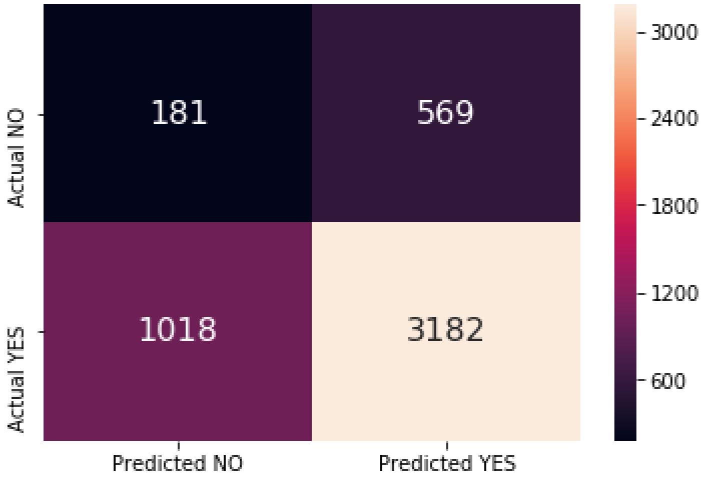
```python

from sklearn.neighbors import KNeighborsClassifier
import sklearn.metrics as metrics
# calculate the fpr and tpr for BOW thresholds of the classification

neigh = KNeighborsClassifier(algorithm='brute',n_neighbors=41)
neigh.fit(X_BOW_TRAIN,y_BOW_train)

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

# 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()



probs=None
probs1=None
preds=None
preds1=None
fpr=None
tpr=None
fpr1=None
tpr1=None
roc_auc=None
roc_auc1=None
gc.collect()
```



Out[82]: 6279

2.6.5 Confusion matrix- (BOW)

In [83]:
```
# 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(neigh,X_BOW_test,y_test)
```



In [84]:
```
#To make best use of the memory we are setting the variable names to 'None' and p
X_BOW_TRAIN=None
y_BOW_train=None
gc.collect()
```

Out[84]: 3146

2.7 Applying KNN brute force on `Tfidf`, `SET 2`

2.7.1 Unpickling to read the data from the disk

In [85]:

```
# Unpickling to read the data from the disk
import pickle
with open('X_Tfidf_train.pickle', 'rb') as f:
    X_Tfidf_train = pickle.load(f)

with open('X_Tfidf_test.pickle', 'rb') as f:
    X_Tfidf_test = pickle.load(f)
gc.collect()
```

Out[85]:

```
0
```

In [ ]:

```
# import sys
# sys.getsizeof(X_Tfidf)
```

2.7.2 Simple `upsampling` on training dataset to ovecome the imbalance in the data

In [86]:

```
# We are performing simple sampling to overcome the imbalance in our train data.
# We are trying to match the minority class 0's with 1's with a difference of 20%

from imblearn.over_sampling import RandomOverSampler
from collections import Counter

ros = RandomOverSampler(sampling_strategy='float',ratio=.80,random_state=20)
print('Original dataset shape %s' % Counter(y_train))
X_Tfidf_train, y_Tfidf_train = ros.fit_resample(X_Tfidf_train, y_train)
print('Post sampling dataset shape %s' % Counter(y_Tfidf_train))

print(X_Tfidf_train.shape, y_Tfidf_train.shape)
# print(X_Tfidf_cv.shape, y_Tfidf_cv.shape)
print(X_Tfidf_test.shape, y_test.shape)

print("="*100)
```

```
Original dataset shape Counter({1: 8526, 0: 1524})
Post sampling dataset shape Counter({1: 8526, 0: 6820})
(15346, 3533) (15346,)
(4950, 3533) (4950,)
================================================================================
====================
```

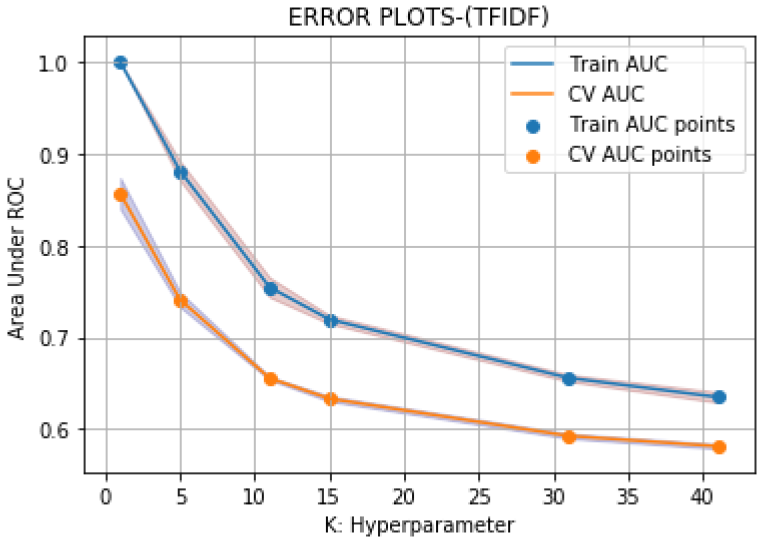
In [ ]:

```
# # %%time
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.metrics import classification_report,confusion_matrix
# knn=KNeighborsClassifier(algorithm='brute',n_neighbors=5)
# knn.fit(X_Tfidf_train,y_Tfidf_train)
# pred=knn.predict(X_Tfidf_test)
# print(confusion_matrix(y_Tfidf_test,pred))
# print(classification_report(y_Tfidf_test,pred))
```

2.7.3 Hyper Parameter Tuning to find the best K. (TFIDF)

In [87]:

```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridS
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score


neigh = KNeighborsClassifier(algorithm='brute',n_jobs=-1)
parameters = {'n_neighbors':[1, 5, 11, 15, 31, 41]}
clf1 = GridSearchCV(neigh, parameters, cv=2, scoring='roc_auc', n_jobs=-1,return_
clf1.fit(X_Tfidf_train,y_Tfidf_train)

train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_

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

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: Hyperparameter")
plt.ylabel("Area Under ROC")
plt.title("ERROR PLOTS-(TFIDF)")
plt.grid()
plt.show()
```

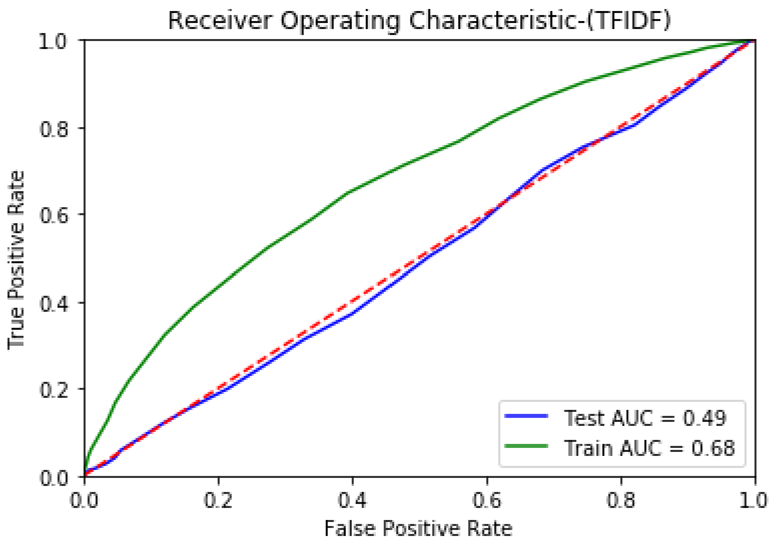


```
Wall time: 32min 22s
```

2.7.4 Plotting the `ROC curve` on both Test and Train data and obtaining the AUC on the test data usin the best K- `(TFIDF)`

In [88]:

```
import sklearn.metrics as metrics
# calculate the fpr and tpr for Tfidf thresholds of the classification

neigh = KNeighborsClassifier(algorithm='brute',n_neighbors=41)
neigh.fit(X_Tfidf_train,y_Tfidf_train)

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

# https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve E4

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()

probs=None
probs1=None
preds=None
preds1=None
fpr=None
tpr=None
fpr1=None
tpr1=None
roc_auc=None
roc_auc1=None
gc.collect()
```

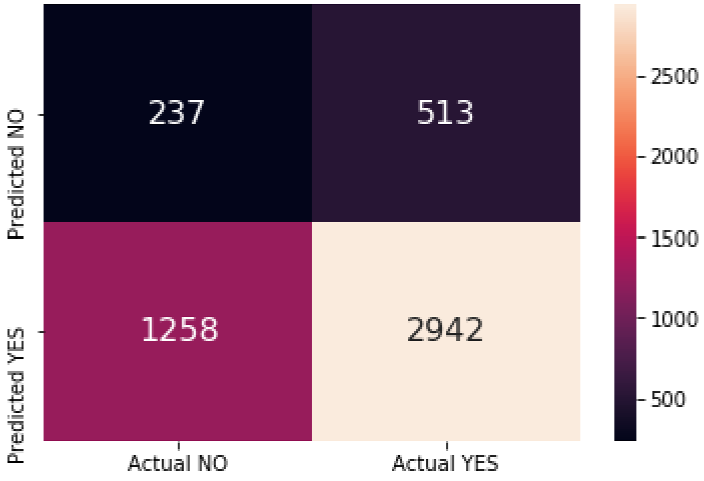


Out[88]: 6844

2.7.5 Confusion matrix- (`TFIDF`)

In [89]:

```
# 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), ['Predicted NO','Pred
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')

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



In [90]:

```
#To make best use of the memory we are setting the variable names to 'None' and p
X_Tfidf_test=None
X_Tfidf_train=None
gc.collect()
```

Out[90]: 3152

2.8 Applying KNN brute force on `AVG W2V`, `SET 3`

2.8.1 Unpickling to read the data from the disk

In [91]:

```
# # %%time
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.metrics import classification_report,confusion_matrix
# knn=KNeighborsClassifier(algorithm='brute',n_neighbors=5)
# knn.fit(X_avg_w2v_train,y_Tfidf_train)
# pred=knn.predict(X_avg_w2v_test)
# print(confusion_matrix(y_avg_w2v_test,pred))
# print(classification_report(y_avg_w2v_test,pred))

# Unpickling to read the data from the disk
with open('X_avg_w2v_train.pickle', 'rb') as f:
    X_avg_w2v_train = pickle.load(f)

with open('X_avg_w2v_test.pickle', 'rb') as f:
    X_avg_w2v_test = pickle.load(f)
```

In [ ]:

```
# # train test split (67:33) ratio
# from sklearn.model_selection import train_test_split
# X_avg_w2v_train, X_avg_w2v_test, y_avg_w2v_train, y_avg_w2v_test = train_test_s
# # X_avg_w2v_train, X_avg_w2v_cv, y_avg_w2v_train, y_avg_w2v_cv = train_test_sp
# print(X_avg_w2v_train.shape, y_avg_w2v_train.shape)
# # print(X_avg_w2v_cv.shape, y_avg_w2v_cv.shape)
# print(X_avg_w2v_test.shape, y_avg_w2v_test.shape)

# print("="*100)
```

2.8.2 Simple `upsampling` on training dataset to ovecome the imbalance in the data

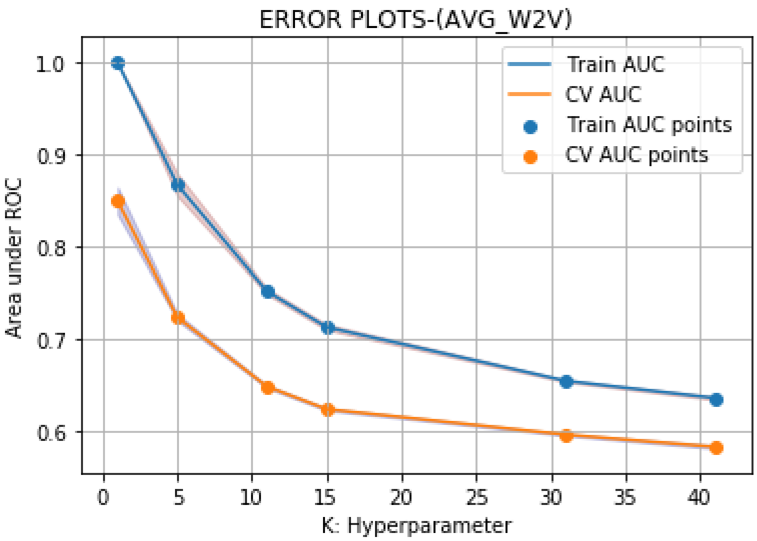
In [92]:
```
# We are performing simple sampling to overcome the imbalance in our train data.
# We are trying to match the  minority class 0's with 1's with a difference of 2

from imblearn.over_sampling import RandomOverSampler
from collections import Counter

ros = RandomOverSampler(sampling_strategy='float',ratio=.80,random_state=20)
print('Original dataset shape %s' % Counter(y_train))
X_avg_w2v_train, y_avg_w2v_train = ros.fit_resample(X_avg_w2v_train, y_train)
print('Post sampling dataset shape %s' % Counter(y_avg_w2v_train))

print(X_avg_w2v_train.shape, y_avg_w2v_train.shape)
# print(X_avg_w2v_cv.shape, y_avg_w2v_cv.shape)
print(X_avg_w2v_test.shape, y_test.shape)

print("="*100)
```

```
Original dataset shape Counter({1: 8526, 0: 1524})
Post sampling dataset shape Counter({1: 8526, 0: 6820})
(15346, 701) (15346,)
(4950, 701) (4950,)
================================================================================
====================
```

2.8.3 Hyper Parameter Tuning to find the best K. (AVG_W2V)

In [93]:

```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.Grid
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score


# lst=[[X_BOW_train,y_BOW_train],[X_BOW_train,y_BOW_train],[X_Tfidf_train,y_Tfid


neigh = KNeighborsClassifier(algorithm='brute',n_jobs=-1)
parameters = {'n_neighbors':[1, 5, 11, 15, 31, 41]}
clf1 = GridSearchCV(neigh, parameters, cv=2, scoring='roc_auc', n_jobs=-1,return_
clf1.fit(X_avg_w2v_train,y_avg_w2v_train)

train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_

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

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: Hyperparameter")
plt.ylabel("Area under ROC")
plt.title("ERROR PLOTS-(AVG_W2V)")
plt.grid()
plt.show()

#To make best use of the memory we are setting the variable names to 'None' and p
```

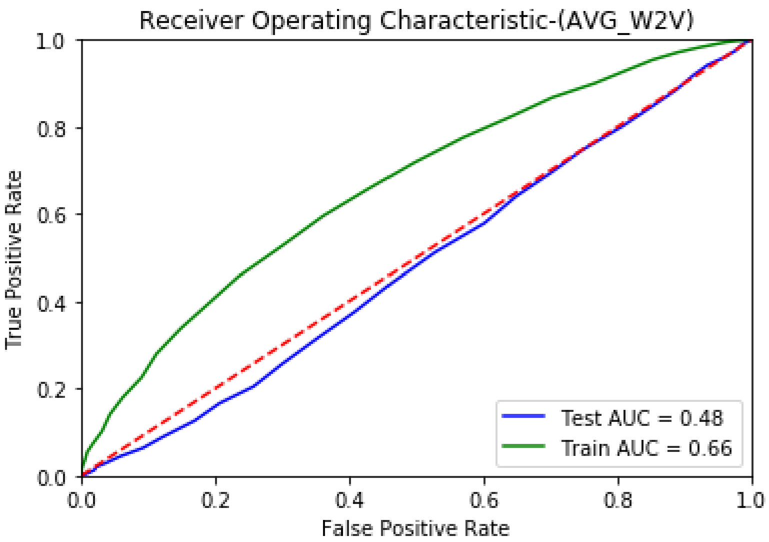


```
Wall time: 7min 34s
```

2.8.4 Plotting the `ROC curve` on both Test and Train data and obtaining the AUC on the test data using the best K- `(AVG_W2V)`

In [94]:

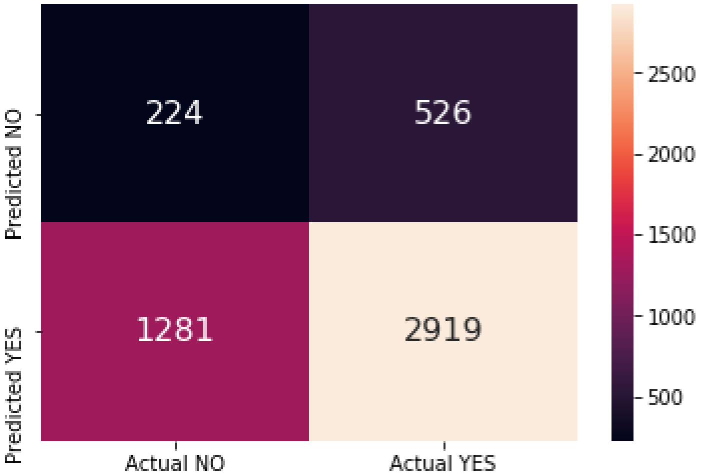
```
import sklearn.metrics as metrics
# calculate the fpr and tpr for avg_w2v thresholds of the classification

neigh = KNeighborsClassifier(algorithm='brute',n_neighbors=41)
neigh.fit(X_avg_w2v_train,y_avg_w2v_train)

# X_Tfidf_train
probs = neigh.predict_proba(X_avg_w2v_test)
probs1 = neigh.predict_proba(X_avg_w2v_train)
preds = probs[:,1]
preds1 = probs1[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
fpr1, tpr1, threshold = metrics.roc_curve(y_avg_w2v_train, preds1)
roc_auc = metrics.auc(fpr, tpr)
roc_auc1 = metrics.auc(fpr1, tpr1)

# https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve E4

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic-(AVG_W2V)')
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()


probs=None
probs1=None
preds=None
preds1=None
fpr=None
tpr=None
fpr1=None
tpr1=None
roc_auc=None
roc_auc1=None
gc.collect()
```



Out[94]: 6849

2.8.5 Confusion matrix- (AVG_W2V)

In [95]:
```
# 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),['Predicted NO','Predi
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')

# %%time
get_confusion_matrix(neigh,X_avg_w2v_test,y_test)
```



In [96]:
```
#To make best use of the memory we are setting the variable names to 'None' and p
X_avg_w2v_test=None
X_avg_w2v_train=None
y_avg_w2v_train=None
gc.collect()
```

Out[96]: 3147

2.9 Applying KNN brute force on `TFIDF W2V` , SET 4

2.9.1 Unpickling to read the data from the disk

In [100]:

```
# # %%time
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.metrics import classification_report,confusion_matrix
# knn=KNeighborsClassifier(algorithm='brute',n_neighbors=5)
# knn.fit(X_tfidf_w2v_train,y_Tfidf_train)
# pred=knn.predict(X_tfidf_w2v_test)
# print(confusion_matrix(y_tfidf_w2v_test,pred))
# print(classification_report(y_tfidf_w2v_test,pred))

# Unpickling to read the data from the disk
with open('X_tfidf_w2v_train.pickle', 'rb') as f:
    X_tfidf_w2v_train = pickle.load(f)

with open('X_tfidf_w2v_test.pickle', 'rb') as f:
    X_tfidf_w2v_test = pickle.load(f)
```

In [ ]:

```
# # Train test split (67:33) ratio.
# from sklearn.model_selection import train_test_split
# X_tfidf_w2v_train, X_tfidf_w2v_test, y_tfidf_w2v_train, y_tfidf_w2v_test = tra
# # X_tfidf_w2v_train, X_tfidf_w2v_cv, y_tfidf_w2v_train, y_tfidf_w2v_cv = train_
# print(X_tfidf_w2v_train.shape, y_tfidf_w2v_train.shape)
# # print(X_tfidf_w2v_cv.shape, y_tfidf_w2v_cv.shape)
# print(X_tfidf_w2v_test.shape, y_tfidf_w2v_test.shape)

# print("="*100)
```

2.9.2 Simple `upsampling` on training dataset to ovecome the imbalance in the data

In [101]:

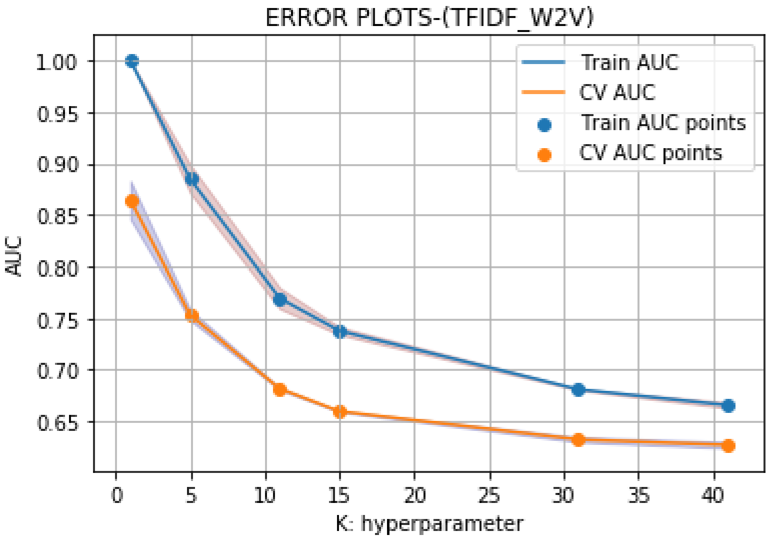
```
# We are performing simple sampling to overcome the imbalance in our train data.
# We are trying to match the 0's minority class with 1's with a difference of 20

from imblearn.over_sampling import RandomOverSampler
from collections import Counter

print('Original dataset shape %s' % Counter(y_train))
ros = RandomOverSampler(sampling_strategy='float',ratio=.80,random_state=20)
X_tfidf_w2v_train, y_tfidf_w2v_train = ros.fit_resample(X_tfidf_w2v_train, y_tra
print('Post sampling dataset shape %s' % Counter(y_tfidf_w2v_train))

print(X_tfidf_w2v_train.shape, y_tfidf_w2v_train.shape)
# print(X_tfidf_w2v_cv.shape, y_tfidf_w2v_cv.shape)
print(X_tfidf_w2v_test.shape, y_test.shape)

print("="*100)
```

```
Original dataset shape Counter({1: 8526, 0: 1524})
Post sampling dataset shape Counter({1: 8526, 0: 6820})
(15346, 701) (15346,)
(4950, 701) (4950,)
===============================================================================
=====================
```

2.9.3 Hyper Parameter Tuning to find the best K. (`TFIDF_W2V`)

In [102]:

```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridS
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score


# lst=[[X_BOW_train,y_BOW_train],[X_BOW_train,y_BOW_train],[X_Tfidf_train,y_Tfid


neigh = KNeighborsClassifier(algorithm='brute',n_jobs=-1)
parameters = {'n_neighbors':[1, 5, 11, 15, 31, 41]}
clf1 = GridSearchCV(neigh, parameters, cv=2, scoring='roc_auc', n_jobs=-1,return_
clf1.fit(X_tfidf_w2v_train,y_tfidf_w2v_train)

train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_

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

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS-(TFIDF_W2V)")
plt.grid()
plt.show()
```

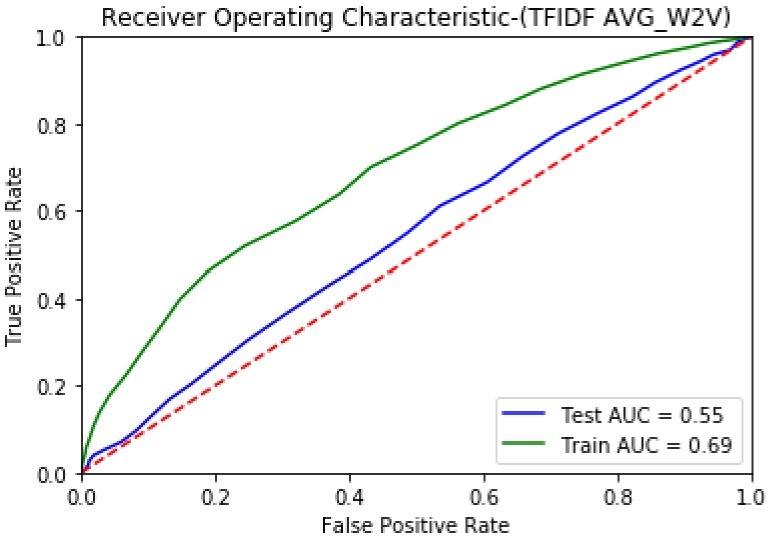


```
Wall time: 7min
```

2.9.4 Plotting the ROC curve on both Test and Train data and obtaining the AUC on the test data using the best K- `(TFIDF_W2V)`

In [103]:

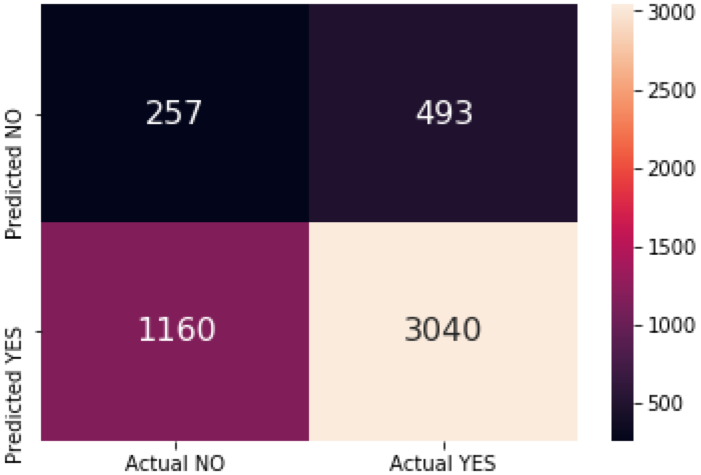
```


import sklearn.metrics as metrics
# calculate the fpr and tpr for tfidf_w2v thresholds of the classification

neigh = KNeighborsClassifier(algorithm='brute',n_neighbors=41)
neigh.fit(X_tfidf_w2v_train,y_tfidf_w2v_train)

# X_Tfidf_train
probs = neigh.predict_proba(X_tfidf_w2v_test)
probs1 = neigh.predict_proba(X_tfidf_w2v_train)
preds = probs[:,1]
preds1 = probs1[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
fpr1, tpr1, threshold = metrics.roc_curve(y_tfidf_w2v_train, preds1)
roc_auc = metrics.auc(fpr, tpr)
roc_auc1 = metrics.auc(fpr1, tpr1)

# https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve E4

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic-(TFIDF AVG_W2V)')
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()


probs=None
probs1=None
preds=None
preds1=None
fpr=None
tpr=None
fpr1=None
tpr1=None
roc_auc=None
roc_auc1=None
gc.collect()
```



Out[103]:
```
2952
```

2.9.5 Confusion matrix- (`TFIDF AVG_W2V`)

In [104]:
```
# 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), ['Predicted NO','Pred
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')

# %%time
get_confusion_matrix(neigh,X_tfidf_w2v_test,y_test)
```



In [105]:
```
#To make best use of the memory we are setting the variable names to 'None' and p
X_tfidf_w2v_test=None
X_tfidf_w2v_train=None
y_tfidf_w2v_train=None
gc.collect()
```

Out[105]:
```
3231
```

2.10 Feature selection with `SelectKBest` for `TFIDF`

2.10.1 Unpickling to read the data from the disk & selecting the best 2000 feature from Tfidf train and test data

In [107]:

```
# # Unpickling to read the data from the disk
with open('X_Tfidf_train.pickle', 'rb') as f:
    X_Tfidf_train = pickle.load(f)

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

In [110]:

```
# Selecting 2000 best features from Tfidf to see the variation in the AUC
from sklearn.feature_selection import SelectKBest, f_classif
X_Tfidf_train = SelectKBest(f_classif, k=2000).fit_transform(X_Tfidf_train,y_tra
X_Tfidf_train.shape
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\sklearn\feature_selection\univariate
_selection.py:114: UserWarning:

Features [3434 3436 3437 3438 3439 3440 3441 3442] are constant.
```

Out[110]:

```
(10050, 2000)
```

In [ ]:

```
# # train test split (67:33) ratio.
# from sklearn.model_selection import train_test_split
# X_Tfidf_train_KB, X_Tfidf_test_KB, y_Tfidf_train_KB, y_test = train_test_split
# # X_Tfidf_train_KB, X_Tfidf_cv_KB, y_Tfidf_train_KB, y_Tfidf_cv_KB = train_test
# print(X_Tfidf_train_KB.shape, y_Tfidf_train_KB.shape)
# # print(X_Tfidf_cv_KB.shape, y_Tfidf_cv_KB.shape)
# print(X_Tfidf_test_KB.shape, y_test.shape)

# print("="*100)
```

2.10.2 Simple `upsampling` on training dataset to ovecome the imbalance in the data

In [111]:

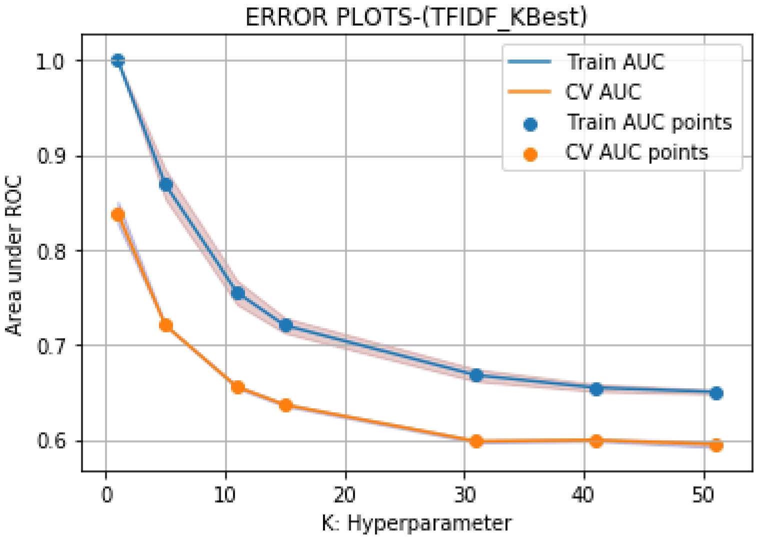
```
# We are performing simple sampling to overcome the imbalance in our train data.
# We are trying to match the 0's minority class with 1's with a difference of 20

from imblearn.over_sampling import RandomOverSampler
from collections import Counter

ros = RandomOverSampler(sampling_strategy='float',ratio=.80,random_state=20)
print('Original dataset shape %s' % Counter(y_train))
X_Tfidf_train, y_Tfidf_train = ros.fit_resample(X_Tfidf_train, y_train)
print('Post sampling dataset shape %s' % Counter(y_Tfidf_train))

print(X_Tfidf_train.shape, y_Tfidf_train.shape)
# print(X_Tfidf_cv.shape, y_Tfidf_cv.shape)
print(X_Tfidf_test.shape, y_test.shape)

print("="*100)
```

```
Original dataset shape Counter({1: 8526, 0: 1524})
Post sampling dataset shape Counter({1: 8526, 0: 6820})
(15346, 2000) (15346,)
(4950, 3533) (4950,)
================================================================================
====================
```

2.10.3 Hyper Parameter Tuning to find the best K. (TFIDF_KBest)

In [112]:

```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridS
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score


neigh = KNeighborsClassifier(n_jobs=-1)
parameters = {'n_neighbors':[1, 5, 11, 15, 31, 41, 51]}
clf1 = GridSearchCV(neigh, parameters, cv=2, scoring='roc_auc', n_jobs=-1,return_
clf1.fit(X_Tfidf_train,y_Tfidf_train)

train_auc= clf1.cv_results_['mean_train_score']
train_auc_std= clf1.cv_results_['std_train_score']
cv_auc = clf1.cv_results_['mean_test_score']
cv_auc_std= clf1.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_

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

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: Hyperparameter")
plt.ylabel("Area under ROC")
plt.title("ERROR PLOTS-(TFIDF_KBest)")
plt.grid()
plt.show()
```



```
Wall time: 1h 58min 9s
```

In [116]:

```
from sklearn.feature_selection import SelectKBest, f_classif
X_Tfidf_test = SelectKBest(f_classif, k=2000).fit_transform(X_Tfidf_test,y_test)
X_Tfidf_test.shape
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\sklearn\feature_selection\univariate
_selection.py:114: UserWarning:

Features [  87  807 3434 3436 3437 3438 3439 3440 3441 3442] are constant.
```

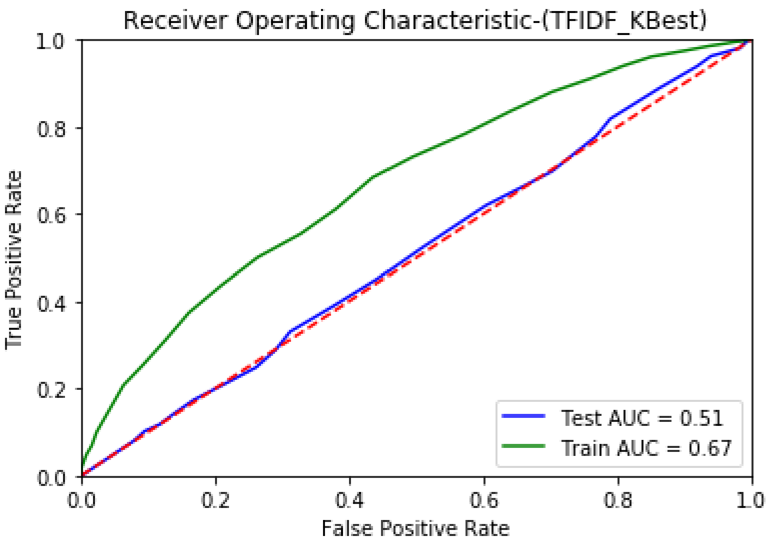
Out[116]:

```
(4950, 2000)
```

2.10.4 Plotting the `ROC curve` on both Test and Train data and obtaining the AUC on the test data using the best K- `(Tfidf_KBest)`

In [120]:

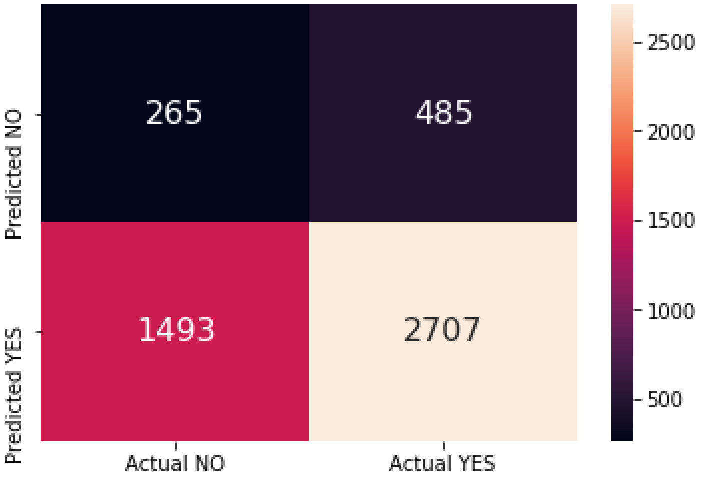
```
# Here are two ways you may try, assuming your model is an sklearn predictor:

import sklearn.metrics as metrics
# calculate the fpr and tpr for Tfidf thresholds of the classification

neigh = KNeighborsClassifier(algorithm='brute',n_neighbors=41)
neigh.fit(X_Tfidf_train,y_Tfidf_train)

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

# https://www.programcreek.com/python/example/81207/sklearn.metrics.roc_curve E4

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic-(TFIDF_KBest)')
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()

probs=None
probs1=None
preds=None
preds1=None
fpr=None
tpr=None
fpr1=None
tpr1=None
roc_auc=None
roc_auc1=None
gc.collect()
```



```
Out[120]: 2379
```

2.10.5 Confusion matrix- `(TFIDF_KBest)`

```
In [119]: # 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), ['Predicted NO','Pred
              sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')

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



```
In [ ]: #To make best use of the memory we are setting the variable names to 'None' and p
        X_Tfidf_test=None
        X_Tfidf_train=None
        gc.collect()
```

3. Conclusions

3. Conclusions

In [123]:
```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer","Model", "Hyper Parameter","Train-AUC", "Test-AUC"
x.add_row(["BOW","KNN-Brute", 41,0.67, 0.50])
x.add_row(["Tfidf","KNN-Brute" ,41,0.68, 0.49])
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 | Train-AUC | Test-AUC |
+--------------+-----------+-----------------+-----------+----------+
|     BOW      | KNN-Brute |        41       |    0.67   |   0.5    |
|    Tfidf     | KNN-Brute |        41       |    0.68   |   0.49   |
|   avg_w2v    | KNN-Brute |        41       |    0.66   |   0.48   |
|  tfidf_w2v   | KNN-Brute |        41       |    0.69   |   0.55   |
| Tfidf_K_Best | KNN-Brute |        41       |    0.67   |   0.51   |
+--------------+-----------+-----------------+-----------+----------+
```