Logistic Regression 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_1=project_data.head(30000)
project_data_0=project_data.tail(30000)
project_data_1=project_data_1.append(project_data_0)
project_data=project_data_1
resource_data = pd.read_csv('resources.csv')
project_data_1=None
project_data_0=None
```

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 (60000, 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)
project_data=None
gc.collect()
gc.enable()
gc.DEBUG_SAVEALL
```

Out[5]: 32

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

2.2.1 Preprocessing:project_grade_category

In [6]:
```
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 [7]:
```
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[7]: 32

2.2.2 Preprocessing:project_subject_categories

In [8]:
```
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[8]:

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

2.2.3 Preprocessing:project_subject_subcategories

In [9]:
```
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 [10]:
```
X['clean_subcategories'] = sub_cat_list
X.drop(['project_subject_subcategories'], axis=1, inplace=True)
X.head(2)
```

Out[10]:

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

2.2.4 New Column:digits in summary

In [11]:
```
# Creating a new column 'digits_in_summary' which contains flags of 1 for /
# 'project_resource_summary' containing numeric specification in their requiremnt
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 [12]:
```
#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[12]: 32

2.2.5 Preprocessing:Text features (Project Essay's)

In [13]:
```
# 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 [14]:
```
X = X.drop(['project_essay_1', 'project_essay_2','project_essay_3', 'project_essa
X.shape
```

Out[14]: (60000, 14)

2.2.6 Adding column Cost per project in dataset

In [15]:
```
# 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[15]:
```
pandas.core.frame.DataFrame
```

In [16]:
```
# join two dataframes in python:
X = pd.merge(X, price_data, on='id', how='left')
X.head(2)
```

Out[16]:

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

In [17]:
```
#To make best use of the memory we are setting the variable names to 'None' and p
resource_data=None
price_data=None
gc.collect()
gc.enable()
gc.DEBUG_SAVEALL
```

Out[17]:
```
32
```

2.2.7 Text Preprocessing:Essay Text

In [18]:
```
# 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 [19]:
```
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 [20]:
```
# \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 [21]:
```
#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 [22]:
```
# 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 [23]:
```
# 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%|██████████████████████████████████| 60000/60000 [01:28<00:00, 679.43it/s]
```

In [24]:
```
# after preprocesing

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

X.head(2)
```

Out[24]:

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

In [25]:
```
# 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%|██████████| 60000/60000 [00:05<00:00, 10906.48it/s]

In [26]:
```
preprocessed_project_title[4999]
# after preprocesing

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

X.head(2)
```

Out[26]:

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

2.2.8 Splitting the data into Train and Test

In [27]:
```
# 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=0.60, strati
# X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
```

In [28]:
```
# 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': 11490, 'Math_Science': 9038, 'Health_Sports': 312
6, 'SpecialNeeds': 3012, 'AppliedLearning': 2670, 'Music_Arts': 2269, 'History_
Civics': 1302, 'Warmth': 319, 'Care_Hunger': 319})
```

2.2.9 Vectorizing Categorical data: clean_categories(Project subject categories)

In [29]:
```
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)

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)
```

```
(24000, 16) (24000,)
(36000, 16) (36000,)
============================================================================
====================
After vectorizations
(24000, 9) (24000,)
(36000, 9) (36000,)
['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 [30]:
```
# 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[1
```

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)

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

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)
```

```
(24000, 16) (24000,)
(36000, 16) (36000,)
============================================================================
====================
After vectorizations
(24000, 30) (24000,)
(36000, 30) (36000,)
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Ex
tracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducation',
'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'CharacterEducatio
n', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_Geography',
'Health_LifeScience', 'EarlyDevelopment', 'Gym_Fitness', 'ESL', 'EnvironmentalS
cience', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'L
iterature_Writing', 'Mathematics', 'Literacy']
============================================================================
====================
```

2.2.11 Vectorizing Categorical data: school_state

In [32]:
```
# 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 [33]:
```
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)

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)
```

```
(24000, 16) (24000,)
(36000, 16) (36000,)
====================================================================================
====================
After vectorizations
(24000, 51) (24000,)
(36000, 51) (36000,)
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'NH', 'AK', 'DE', 'ME', 'HI', 'WV',
'DC', 'NM', 'ID', 'KS', 'IA', 'CO', 'AR', 'KY', 'MN', 'OR', 'MS', 'NV', 'MD',
'TN', 'UT', 'CT', 'AL', 'WI', 'VA', 'AZ', 'OK', 'NJ', 'WA', 'LA', 'MA', 'OH',
'IN', 'MO', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'TX', 'NY', 'CA']
====================================================================================
====================
```

2.2.12 Vectorizing Categorical data: project_grade_category

In [34]:
```
# 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=la
```

In [35]:
```
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


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)
```

```
(24000, 16) (24000,)
(36000, 16) (36000,)
============================================================================
====================
After vectorizations
(24000, 4) (24000,)
(36000, 4) (36000,)
['Grades9-12', 'Grades6-8', 'Grades3-5', 'GradesPreK-2']
============================================================================
====================
```

2.2.13 Vectorizing Categorical data: teacher_prefix

In [36]:
```
#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 [37]:
```
#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 [38]:
```
# 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=l
# teacher_prefix_dict
```

In [39]:
```
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.key
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(vectorizer_test.get_feature_names())
print("="*100)
```

```
(24000, 16) (24000,)
(36000, 16) (36000,)
====================================================================================
====================
After vectorizations
(24000, 5) (24000,)
(36000, 5) (36000,)
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
====================================================================================
====================
```

2.3 Make Data Model Ready: Vectorizing Numerical features

2.3.1 Vectorizing Numerical features--Price

In [40]:
```
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
normalizer = StandardScaler() #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
(24000, 1) (24000,)
(36000, 1) (36000,)
====================================================================================================
```

2.3.2 Vectorizing Numerical features--teacher_number_of_previously_posted_projects

In [41]:
```
from sklearn.preprocessing import Normalizer
normalizer_train = StandardScaler() #Normalizer()
normalizer_test = StandardScaler() #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(



# X_train_teacher_number_of_previously_posted_projects_norm=np.reshape(X_train_te
# X_test_teacher_number_of_previously_posted_projects_norm=np.reshape(X_test_teac


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.shap
print("="*100)
```

```
After vectorizations
(24000, 1) (24000,)
(36000, 1) (36000,)
====================================================================================================
```

2.3.3 Vectorizing Numerical features--digits_in_summary

In [42]:
```
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 [43]:
```
from sklearn.preprocessing import Normalizer
normalizer_train = StandardScaler() #Normalizer()
normalizer_test = StandardScaler() #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
(24000, 1) (24000,)
(36000, 1) (36000,)
====================================================================================
======================
```

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

Vectorizing Text data

2.4.1 Bag of words:Essays

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)

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

vectorizer = CountVectorizer(min_df=10,ngram_range=(2,2), 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)

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)
```

```
(24000, 16) (24000,)
(36000, 16) (36000,)
==========================================================================================
====================
After vectorizations
(24000, 5000) (24000,)
(36000, 5000) (36000,)
==========================================================================================
====================
```

2.4.2 Bag of words: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)


vectorizer = CountVectorizer(min_df=10,ngram_range=(2,2), 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("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)
```

```
(24000, 16) (24000,)
(36000, 16) (36000,)
============================================================================
====================
After vectorizations
(24000, 682) (24000,)
(36000, 682) (36000,)
============================================================================
====================
```

In [46]:
```
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_previ
X_BOW_test=X_BOW_test.todense()
X_BOW_test=np.array(X_BOW_test)

X_train_project_title_bow=None
X_train_essay_bow =None

X_test_project_title_bow=None
X_test_essay_bow =None
```

2.5 Applying Logistic Regression on BOW, SET 1

2.5.1 Applying Logistic Regression & GridSearchCV on Train data to obtain the best C

In [47]:
```
# Selecting 2000 best features from Tfidf to see the variation in the AUC
from sklearn.feature_selection import SelectKBest, f_classif
X_BOW_TRAIN = SelectKBest(f_classif, k=5000).fit_transform(X_BOW_TRAIN,y_train)
X_BOW_TRAIN.shape
X_BOW_test = SelectKBest(f_classif, k=5000).fit_transform(X_BOW_test,y_test)
X_BOW_test.shape
```

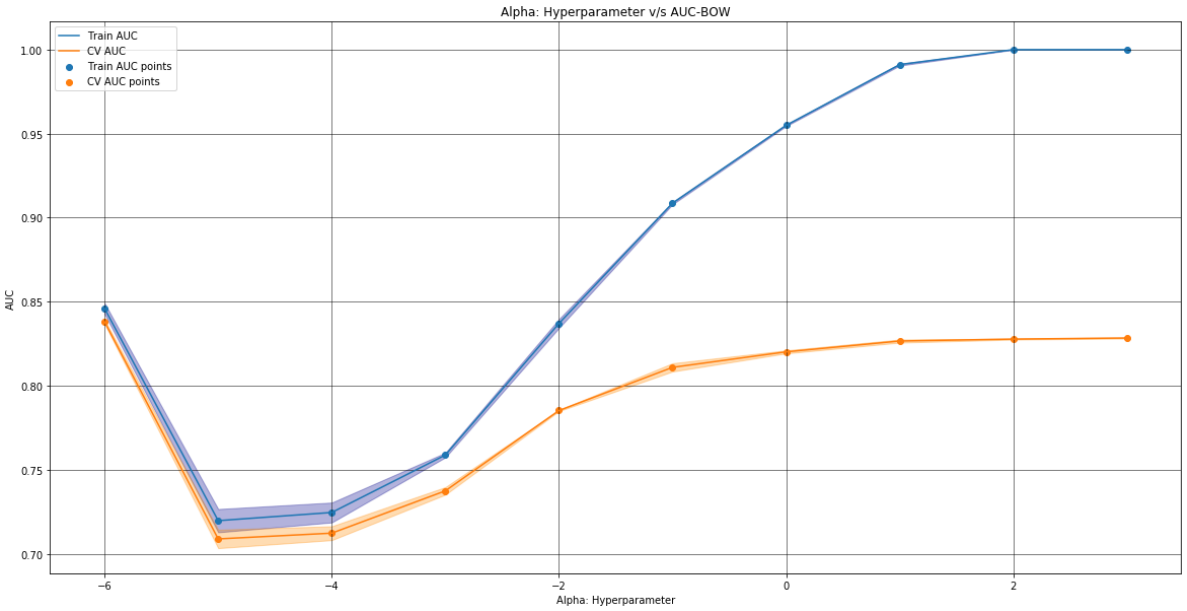
```
C:\Users\Admin\Anaconda3\lib\site-packages\sklearn\feature_selection\univariate
_selection.py:114: UserWarning:

Features [5724 5726 5727 5728 5729 5730 5731 5732] are constant.

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

Features [5724 5726 5727 5728 5729 5730 5731 5732] are constant.
```

Out[47]: (36000, 5000)

In [49]:
```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/mach
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import *
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import f1_score

# data = load_breast_cancer() #refer: http://scikit-learn.org/stable/modules/gene

tuned_parameters = {'C': [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]}
# X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, tra

# LR=SGDClassifier(loss='log',class_weight="balanced")
#Using GridSearchCV
# SGDClassifier(loss='log',class_weight="balanced",max_iter=1500)
# LogisticRegression(penalty = 'l1', C = i,class_weight="balanced")
model = GridSearchCV(LogisticRegression(penalty='l2',class_weight="balanced",solv
model.fit(X_BOW_TRAIN, y_train)

print(model.best_estimator_)
print(model.score(X_BOW_test, y_test))
```

```
LogisticRegression(C=1e-06, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=2000, multi_class='warn', n_jobs=None, penalty='l
2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
0.7103604045958951
```

In [50]:
```
# from sklearn.model_selection import GridSearchCV

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

# parameters =  [0.00001, 0.0001,0.001, 0.01, 0.1,0.5,0.8, 1, 10,  100,  1000]
alpha=[*tuned_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=model.best_params_
# print(best_alpha1)
train_auc= model.cv_results_['mean_train_score']
# print(train_auc)
train_auc_std= model.cv_results_['std_train_score']
# print(train_auc_std)
cv_auc = model.cv_results_['mean_test_score']
# print(cv_auc)
cv_auc_std= model.cv_results_['std_test_score']
# print(cv_auc_std)
# print(log_alphas)

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 [51]:
```
print("The best alpha from the above graph is ",best_alpha1)
best_alpha1['C']
```

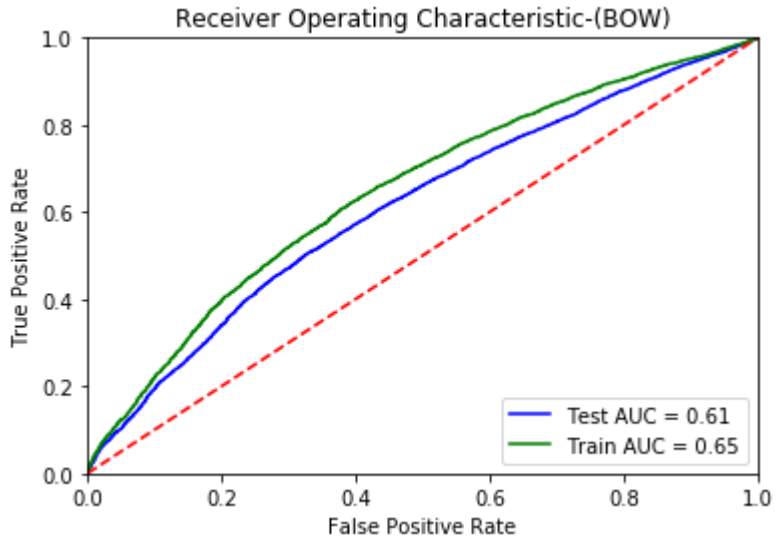
```
The best alpha from the above graph is  {'C': 1e-06}
```

Out[51]:
```
1e-06
```

2.5.2 Receiver Operating Characteristic- (BOW)

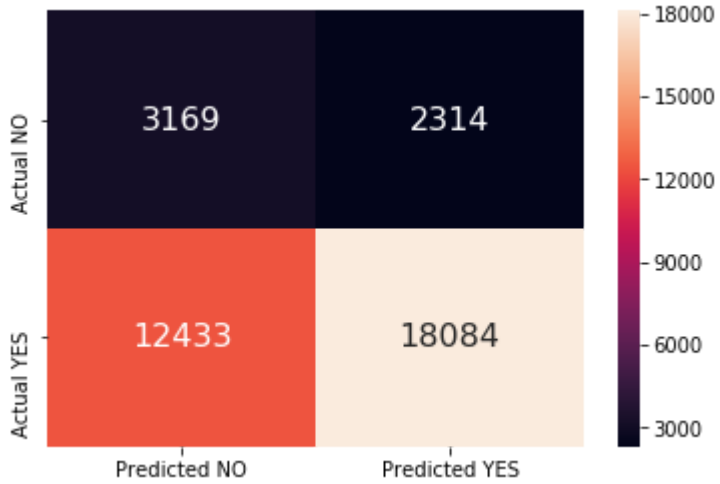
In [52]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htm
from sklearn.metrics import roc_curve, auc
from sklearn.linear_model import SGDClassifier
# train model on the best k
LR=LogisticRegression(C=best_alpha1['C'],class_weight="balanced",solver='lbfgs',m

LR.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 = LR.predict_proba(X_BOW_test)
# print(len(probs[:,1]))
probs1 = LR.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_1 = 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()

# print(model.best_estimator_)
```

```
0.6120651957976488
0.6477662769833696
```



2.5.3 Confusion matrix- BOW

In [53]:
```
# 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','A
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')

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



In [54]:
```
#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[54]: 6209

2.6 TFIDF vectorizer

2.6.1 TFIDF vectorizer:Essay

In [55]:
```
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=(2,2), 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)
```

```
(24000, 16) (24000,)
(36000, 16) (36000,)
====================================================================================================
After vectorizations
(24000, 5000) (24000,)
(36000, 5000) (36000,)
====================================================================================================
```

2.6.2 TFIDF vectorizer:Project Title

In [56]:
```
# 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=(2,2), 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
(24000, 682) (24000,)
(36000, 682) (36000,)
====================================================================================
====================
```

In [57]:
```
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_digi
X_Tfidf_test=X_Tfidf_test.todense()
X_Tfidf_test=np.array(X_Tfidf_test)
```

2.6.3 Applying Logistic Regression on `TFIDF`, `SET 2`

Applying Logistic Regression & GridSearchCV on Train data to obtain the best `C`

In [58]:
```
# 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=5000).fit_transform(X_Tfidf_train,y_tra
X_Tfidf_train.shape
X_Tfidf_test = SelectKBest(f_classif, k=5000).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 [5685 5687 5688 5689 5690 5691 5692 5693] are constant.

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

Features [5685 5687 5688 5689 5690 5691 5692 5693] are constant.
```

Out[58]:
```
(36000, 5000)
```

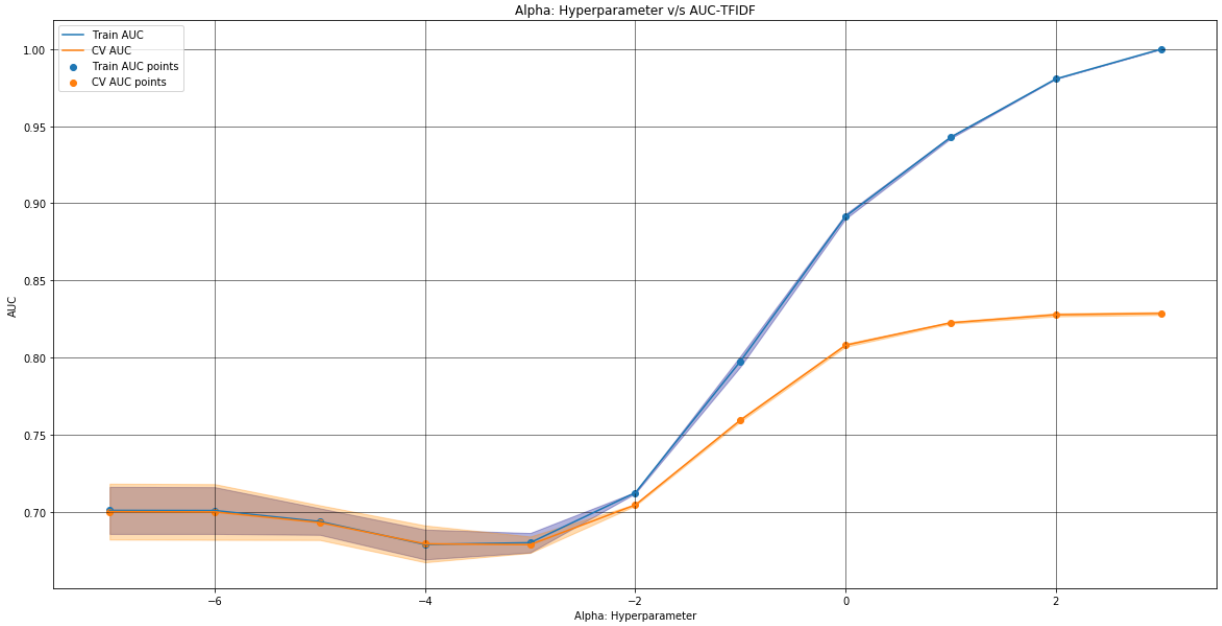
In [67]:
```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/mach
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import *
from sklearn.linear_model import LogisticRegression

# data = load_breast_cancer() #refer: http://scikit-learn.org/stable/modules/gene

tuned_parameters = {'C': [0.0000001,0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,1
# X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, tra


#Using GridSearchCV
model = GridSearchCV(LogisticRegression(class_weight="balanced",solver='lbfgs',ma
model.fit(X_Tfidf_train, y_train)

print(model.best_estimator_)
print(model.score(X_Tfidf_test, y_test))
```

```
LogisticRegression(C=1000, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=3000, multi_class='warn', n_jobs=None, penalty='l
2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
0.8038152819764411
```

In [68]:
```
# from sklearn.model_selection import GridSearchCV

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

# parameters =  [0.00001, 0.0001,0.001, 0.01, 0.1,0.5,0.8, 1, 10,  100,  1000]
alpha=[*tuned_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=model.best_params_
print(best_alpha1)
train_auc= model.cv_results_['mean_train_score']
print(train_auc)
train_auc_std= model.cv_results_['std_train_score']
print(train_auc_std)
cv_auc = model.cv_results_['mean_test_score']
print(cv_auc)
cv_auc_std= model.cv_results_['std_test_score']
print(cv_auc_std)
print(log_alphas)

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()
```

```
{'C': 1000}
[0.70092871 0.70083723 0.69367852 0.67882291 0.679956   0.71235882
 0.79759488 0.89151146 0.94280236 0.98060831 0.99990168]
[1.52488774e-02 1.51574002e-02 8.59164758e-03 9.66426426e-03
 6.32324259e-03 4.70207082e-04 3.48553124e-03 1.56493260e-03
 7.52300005e-04 5.10794055e-05 4.91642081e-05]
[0.70014074 0.69991165 0.69299079 0.67923395 0.67868945 0.7044605
```

```
 0.75943924 0.80797646 0.82256111 0.82771522 0.82859499]
[0.01812479 0.01805501 0.01124956 0.01195397 0.00531465 0.00054988
 0.00098622 0.00104338 0.00063902 0.00103893 0.00098766]
[-7.0, -6.0, -5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0]
```

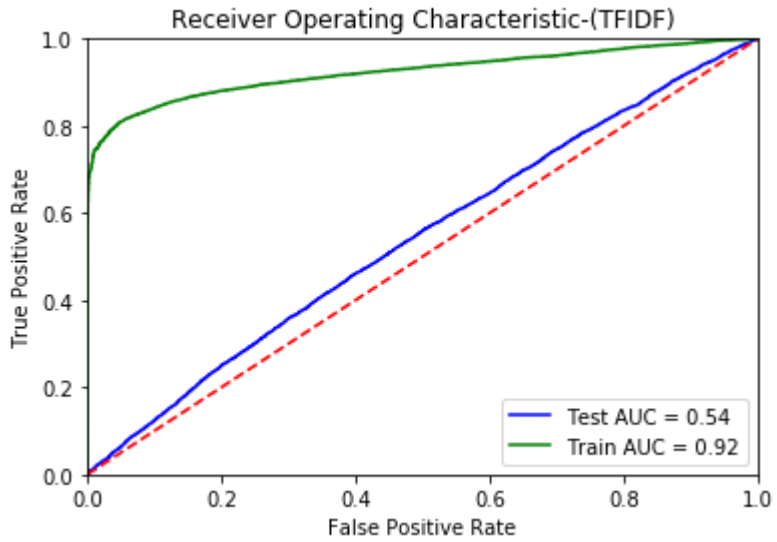


In [69]:
```
# best_alpha1=0.01
print("The best alpha from the above graph is ",best_alpha1['C'])
```

```
The best alpha from the above graph is  1000
```

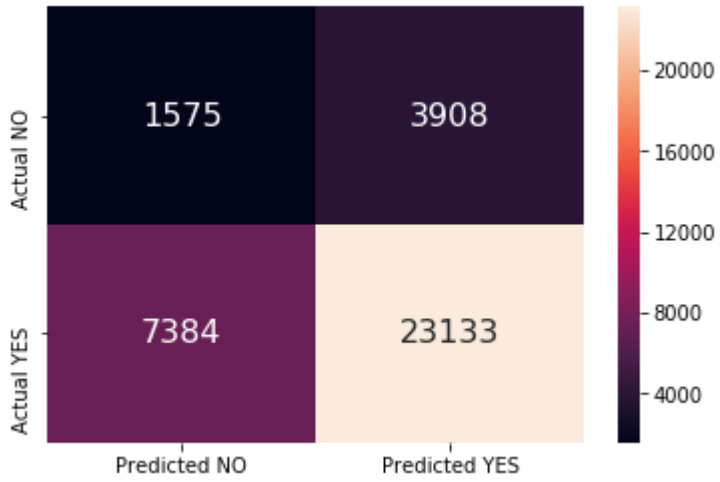
2.6.4 Receiver Operating Characteristic- (TFIDF)

In [71]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htm
from sklearn.metrics import roc_curve, auc

# train model on the best k
lr=LogisticRegression(C=best_alpha1['C'],class_weight="balanced", solver='lbfgs',
lr.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 = lr.predict_proba(X_Tfidf_test)
# print(len(probs[:,1]))
probs1 = lr.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()
```



2.6.5 Confusion matrix- TFIDF

In [72]:
```
# 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','A
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')

# %%time
get_confusion_matrix(lr,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()
```

2.7 AVG_W2V

2.7.1 Using Pretrained Models: Avg W2V-Essays

In [74]:
```
# 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 [75]:
```
# 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%|██████████████████████████████████| 60000/60000 [01:44<00:00, 573.09it/s]

60000
300
```

In [76]:
```
# # We are considering only the words which appeared in at least 10 documents(rou
# 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=scipy.sparse.csr_matrix(avg_w2v_vectors_essay)
type(avg_w2v_vectors_essay)
```

Out[76]:
```
scipy.sparse.csr.csr_matrix
```

In [77]:
```
# 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
```

2.7.2 Using Pretrained Models: AVG W2V on `project_title`

In [78]:
```
# 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%|████████████████████████████████| 60000/60000 [00:03<00:00, 16876.67it/s]

60000
300
```

In [79]:
```
# 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 [80]:
```
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[80]:
```
scipy.sparse.csr.csr_matrix
```

In [81]:
```
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
# 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_te
X_avg_w2v_test=X_avg_w2v_test.todense()
X_avg_w2v_test=np.array(X_avg_w2v_test)

avg_w2v_vectors_Pro_title_train=None
avg_w2v_vectors_essay_train=None

avg_w2v_vectors_Pro_title_test=None
avg_w2v_vectors_essay_test=None
```

2.7.3 Applying Logistic Regression on AVG_W2V , SET 3

Applying Logistic Regression & GridSearchCV on Train data to obtain the best C

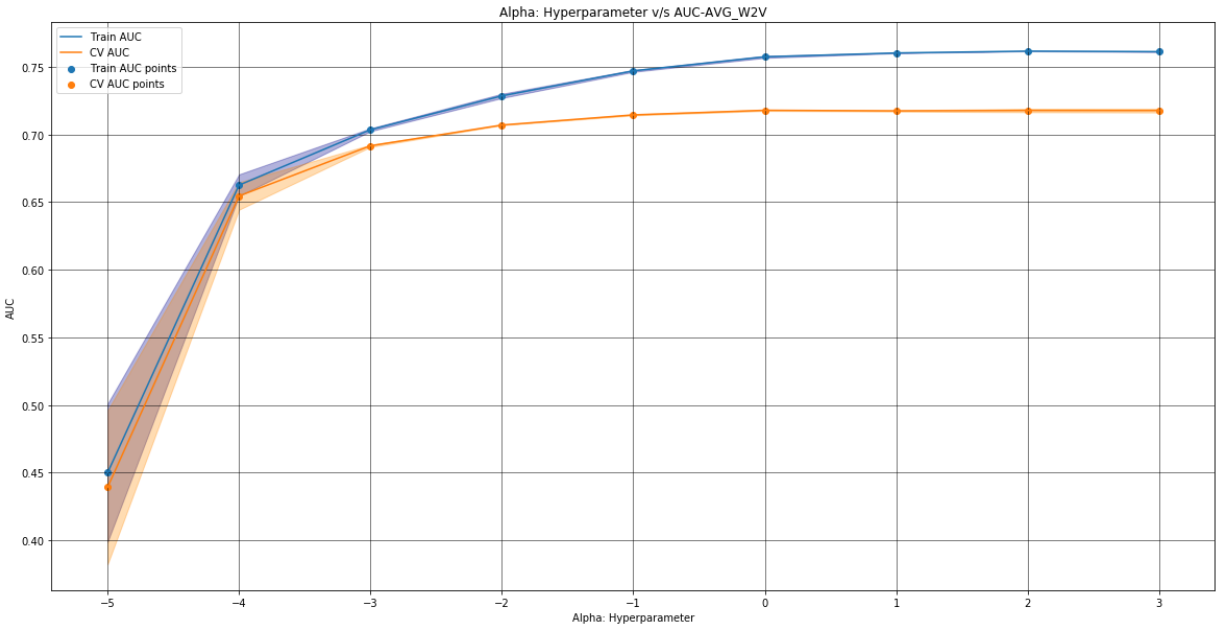
In [83]:
```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/mach
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import *
from sklearn.linear_model import LogisticRegression

# data = load_breast_cancer() #refer: http://scikit-learn.org/stable/modules/gene

tuned_parameters = {'C': [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]}
# X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, tra


#Using GridSearchCV
model = GridSearchCV(LogisticRegression(class_weight='balanced',max_iter=250),tun
model.fit(X_avg_w2v_train, y_train)

print(model.best_estimator_)
print(model.score(X_avg_w2v_test, y_test))
```

```
LogisticRegression(C=1, class_weight='balanced', dual=False, fit_intercept=Tru
e,
                   intercept_scaling=1, l1_ratio=None, max_iter=250,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
0.7040750513046029
```

In [84]:
```
# from sklearn.model_selection import GridSearchCV

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

# parameters =  [0.00001, 0.0001,0.001, 0.01, 0.1,0.5,0.8, 1, 10,  100,  1000]
alpha=[*tuned_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=model.best_params_
print(best_alpha1)
train_auc= model.cv_results_['mean_train_score']
print(train_auc)
train_auc_std= model.cv_results_['std_train_score']
print(train_auc_std)
cv_auc = model.cv_results_['mean_test_score']
print(cv_auc)
cv_auc_std= model.cv_results_['std_test_score']
print(cv_auc_std)
print(log_alphas)

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-AVG_W2V")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

```
{'C': 1}
[0.44992725 0.66246367 0.70338604 0.72842219 0.74688005 0.75725609
 0.76012336 0.76147422 0.76109297]
[5.10285032e-02 7.98809698e-03 1.08233386e-03 1.58776378e-03
 7.11915940e-04 7.67456734e-04 1.94833079e-04 8.50591166e-05
 3.07303671e-04]
[0.43920642 0.65445501 0.69151622 0.70686611 0.71435555 0.71776625
 0.71733322 0.71762543 0.71755867]
```

```
[0.05690116 0.01013633 0.00083269 0.00019631 0.00036856 0.00047788
 0.00049766 0.00130031 0.00142258]
[-5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0]
```

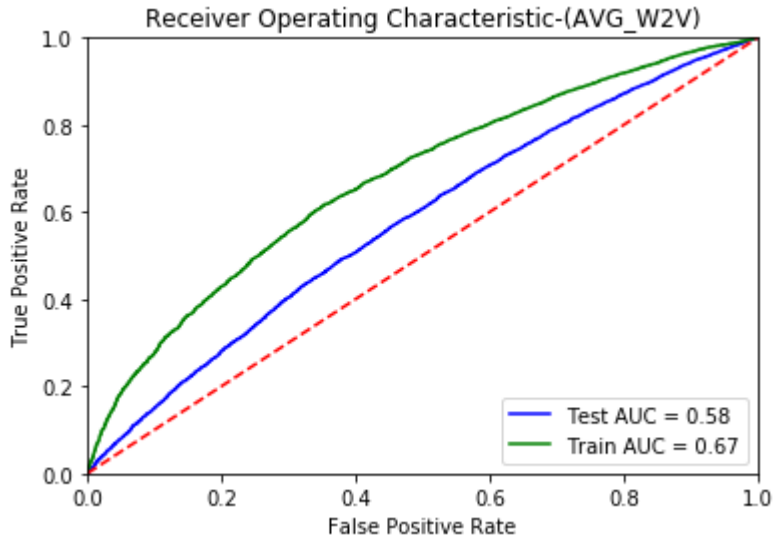


In [85]:
```
print("The best alpha from the above graph is ",best_alpha1)
# best_alpha1['alpha']
```

```
The best alpha from the above graph is  {'C': 1}
```

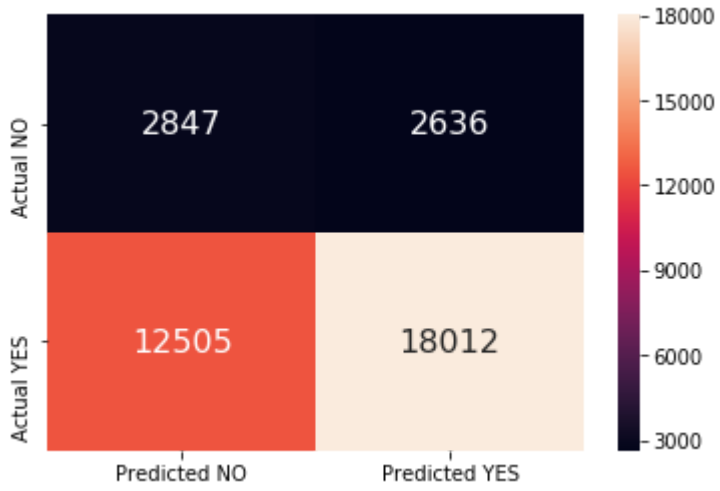
2.7.4 Receiver Operating Characteristic- (AVG_W2V)

In [86]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htm
from sklearn.metrics import roc_curve, auc

# train model on the best k
lr=LogisticRegression(C=best_alpha1['C'],class_weight="balanced", max_iter=250)
lr.fit(X_avg_w2v_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
# not the predicted outputs

probs = lr.predict_proba(X_avg_w2v_test)
# print(len(probs[:,1]))
probs1 = lr.predict_proba(X_avg_w2v_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-(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()
```



2.7.5 Confusion matrix-AVG_W2V

In [87]:
```
# 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','A
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')

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



In [88]:
```
#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[88]: 3246

2.8 TFIDF weighted W2V-Essay

2.8.1 Using Pretrained Models: TFIDF weighted W2V-Essay

In [89]:
```
# 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 [90]:
```
# 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 valu
            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%|████████████████████████████████████| 60000/60000 [13:02<00:00, 76.69it/s]

60000
300
```

In [91]:
```
# 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_s

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

In [92]:
```
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[92]:
```
scipy.sparse.csr.csr_matrix
```

2.8.2 Using Pretrained Models: TFIDF weighted W2V on project_title

In [93]:
```
# 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 [94]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_Pro_title = []; # the avg-w2v for each sentence/review is store
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 valu
            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%|██████████████████████████████████| 60000/60000 [00:14<00:00, 4192.34it/s]

60000
300
```

In [95]:
```
# 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 [96]:
```
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[96]:
```
scipy.sparse.csr.csr_matrix
```

In [97]:
```
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_te
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,
```

2.8.3 Applying Logistic Regression on `TFIDF W2V`, `SET 4`

Applying Logistic Regression & GridSearchCV on Train data to obtain the best `C`

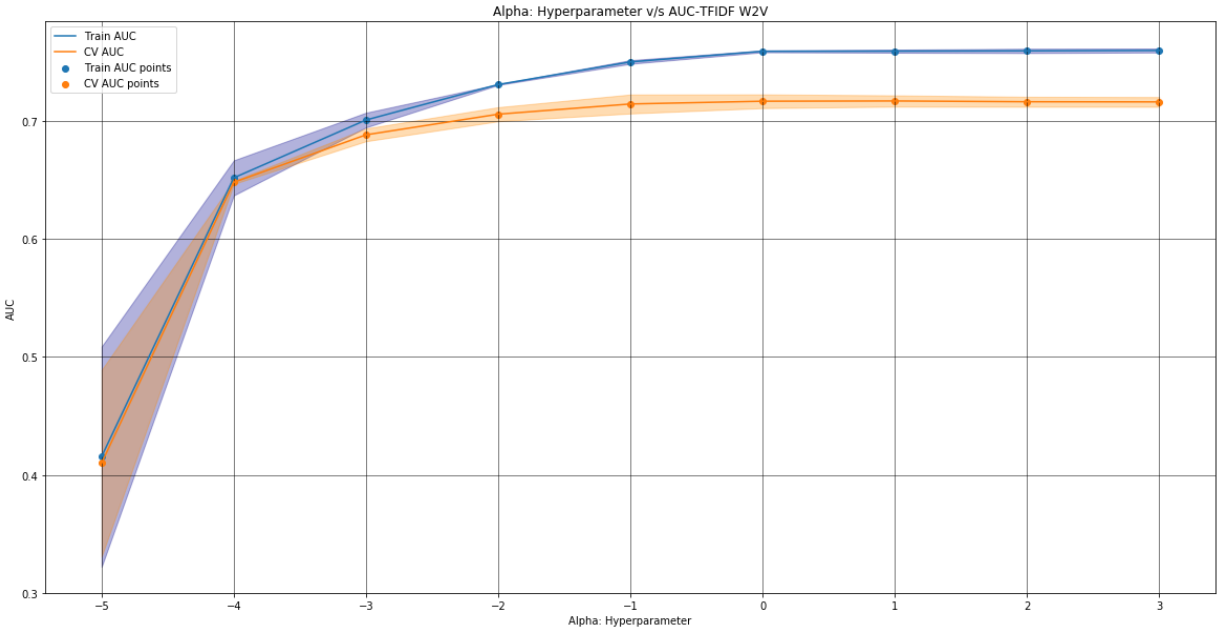
In [98]:
```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/mach
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import *
from sklearn.linear_model import LogisticRegression

# data = load_breast_cancer() #refer: http://scikit-learn.org/stable/modules/gene

tuned_parameters = {'C': [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]}
# X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, tra


#Using GridSearchCV
model = GridSearchCV(LogisticRegression(max_iter=250,class_weight='balanced'), tu
model.fit(X_tfidf_w2v_train, y_train)

print(model.best_estimator_)
print(model.score(X_tfidf_w2v_test, y_test))
```

```
LogisticRegression(C=10, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=250, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
0.7121626072971768
```

In [99]:
```
# from sklearn.model_selection import GridSearchCV

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

# parameters =  [0.00001, 0.0001,0.001, 0.01, 0.1,0.5,0.8, 1, 10,  100,  1000]
alpha=[*tuned_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=model.best_params_
print(best_alpha1)
train_auc= model.cv_results_['mean_train_score']
print(train_auc)
train_auc_std= model.cv_results_['std_train_score']
print(train_auc_std)
cv_auc = model.cv_results_['mean_test_score']
print(cv_auc)
cv_auc_std= model.cv_results_['std_test_score']
print(cv_auc_std)
print(log_alphas)

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 W2V")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

```
{'C': 10}
[0.41538416 0.65156427 0.70053822 0.73057726 0.74972521 0.75873199
 0.75886442 0.75910641 0.75942836]
[0.09331881 0.01485779 0.00624205 0.00034033 0.00157792 0.00080692
 0.00130615 0.00183829 0.00168659]
[0.40989826 0.64776464 0.68798305 0.70545418 0.71411626 0.71645341
 0.71666529 0.71601522 0.71586838]
[0.07899063 0.00133008 0.00541205 0.00596773 0.00804839 0.00582942
```

```
 0.00478909 0.00423722 0.00413162]
[-5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0]
```

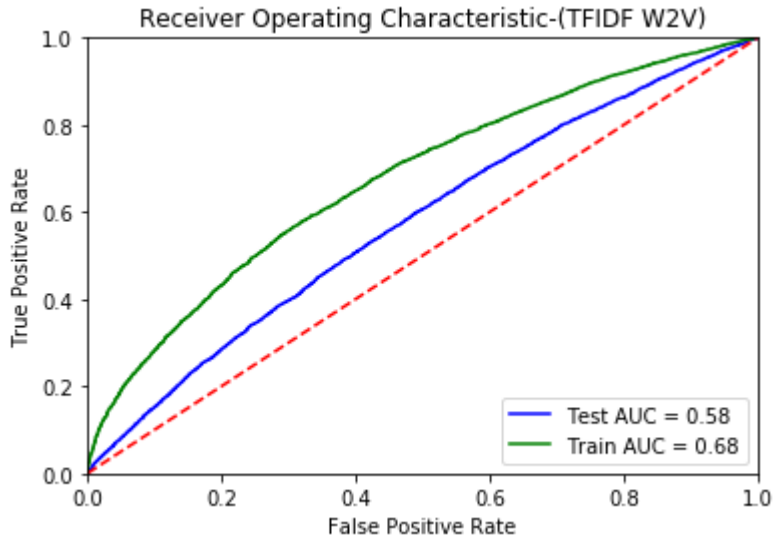


In [100]:
```
print("The best alpha from the above graph is ",best_alpha1)
# best_alpha1['alpha']
```

```
The best alpha from the above graph is  {'C': 10}
```

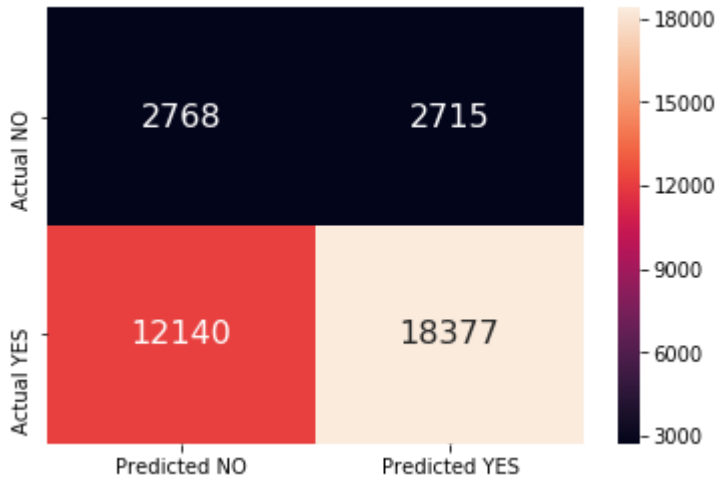
2.8.4 Receiver Operating Characteristic- TFIDF W2V

In [101]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htm
from sklearn.metrics import roc_curve, auc

# train model on the best k
lr=LogisticRegression(penalty='l2',class_weight="balanced",C=best_alpha1['C'])
lr.fit(X_tfidf_w2v_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
# not the predicted outputs

probs = lr.predict_proba(X_tfidf_w2v_test)
# print(len(probs[:,1]))
probs1 = lr.predict_proba(X_tfidf_w2v_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 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()
```



2.8.5 Confusion matrix- TFIDF W2V

In [102]:
```
# 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','A
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')

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



In [103]:
```
#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[103]: 9381

2.9 No text features, SET 5

In [104]:
```
X_NO_text_train = hstack((X_train_digits_in_summary_norm,X_train_teacher_number_o
X_NO_text_train=X_NO_text_train.todense()
X_NO_text_train=np.array(X_NO_text_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_NO_text_test = hstack((X_test_digits_in_summary_norm,X_test_teacher_number_of_p
X_NO_text_test=X_NO_text_test.todense()
X_NO_text_test=np.array(X_NO_text_test)
```

2.9.1 Applying Logistic Regression on `No text features`, `SET 5`

Applying Logistic Regression & GridSearchCV on Train data to obtain the best `C`

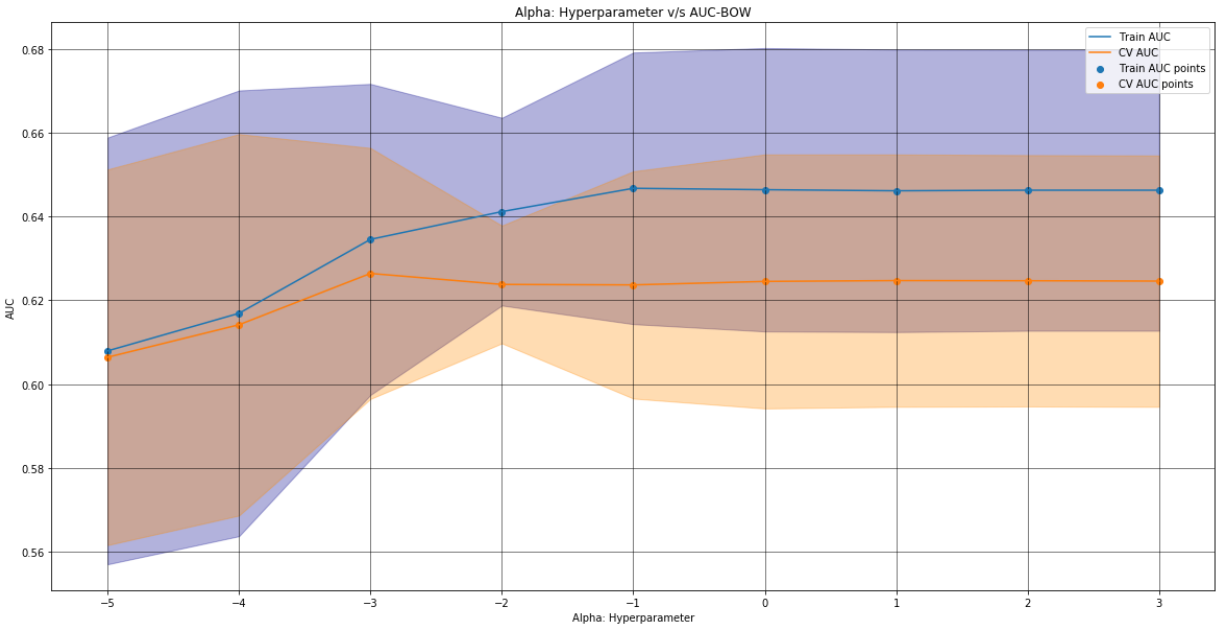
In [105]:
```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterials/mach
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import *
from sklearn.linear_model import LogisticRegression

# data = load_breast_cancer() #refer: http://scikit-learn.org/stable/modules/gene

tuned_parameters = {'C': [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]}
# X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, tra


#Using GridSearchCV
model = GridSearchCV(LogisticRegression(class_weight="balanced",max_iter=1000), t
model.fit(X_NO_text_train, y_train)

print(model.best_estimator_)
print(model.score(X_NO_text_test, y_test))
```

```
LogisticRegression(C=0.001, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=1000, multi_class='warn', n_jobs=None, penalty='l
2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
0.627733906644065
```

In [106]:
```
# from sklearn.model_selection import GridSearchCV

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

# parameters =  [0.00001, 0.0001,0.001, 0.01, 0.1,0.5,0.8, 1, 10,  100,  1000]
alpha=[*tuned_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=model.best_params_
print(best_alpha1)
train_auc= model.cv_results_['mean_train_score']
print(train_auc)
train_auc_std= model.cv_results_['std_train_score']
print(train_auc_std)
cv_auc = model.cv_results_['mean_test_score']
print(cv_auc)
cv_auc_std= model.cv_results_['std_test_score']
print(cv_auc_std)
print(log_alphas)

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()
```

```
{'C': 0.001}
[0.60794708 0.61691858 0.63455655 0.64118263 0.6467479  0.64641342
 0.64615769 0.64629461 0.64629461]
[0.05092633 0.0531876  0.03711573 0.0223995  0.03241126 0.0338146
 0.03371506 0.03353733 0.03353733]
[0.60643048 0.61418529 0.62643112 0.62380616 0.6236956  0.62452607
 0.62472556 0.62468935 0.62460423]
```

```
[0.04483079 0.04552068 0.02994884 0.01405619 0.027122   0.03031671
 0.03011722 0.02999166 0.02999589]
[-5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0]
```

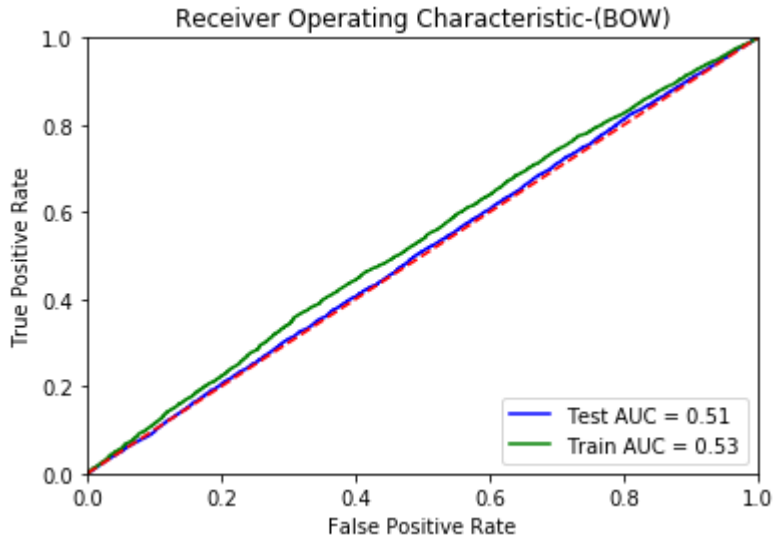


In [107]:
```
print("The best alpha from the above graph is ",best_alpha1)
# best_alpha1['alpha']
```

```
The best alpha from the above graph is  {'C': 0.001}
```

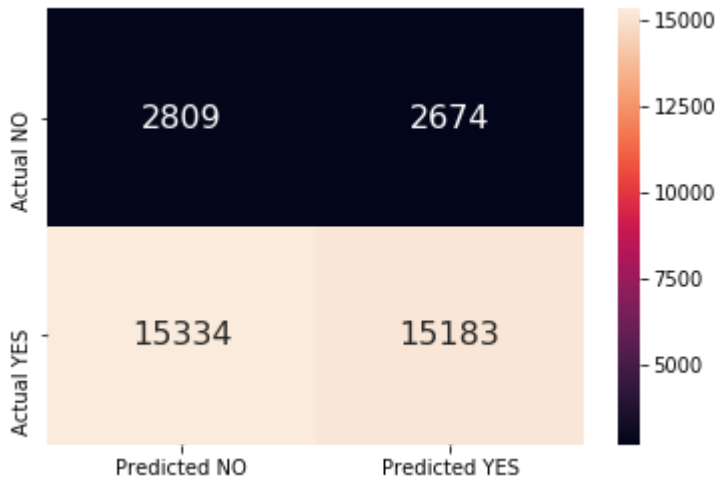
2.9.2 Receiver Operating Characteristic- (No text features)

In [108]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.htm
from sklearn.metrics import roc_curve, auc

# train model on the best k
lr=LogisticRegression(C=best_alpha1['C'],class_weight="balanced", max_iter=250)
lr.fit(X_NO_text_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimate
# not the predicted outputs

probs = lr.predict_proba(X_NO_text_test)
# print(len(probs[:,1]))
probs1 = lr.predict_proba(X_NO_text_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()
```



2.9.3 Confusion matrix- `No text features`

In [109]:
```
# 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','A
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')

# %%time
get_confusion_matrix(lr,X_NO_text_test,y_test)
```



In [110]:
```
#To make best use of the memory we are setting the variable names to 'None' and p
X_NO_text_test=None
X_NO_text_train=None
# y_tfidf_w2v_train=None
gc.collect()
```

Out[110]: 5530

2.10 Pretty table summary

In [112]:
```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer","Model", "Hyper Parameter", "Test-AUC"]
x.add_row(["BOW","LR", '0.0001', 0.61])
x.add_row(["Tfidf","LR" ,'0.0001', 0.54])
x.add_row(["avg_w2v","LR",'0.0001', 0.58])
x.add_row(["tfidf_w2v","LR",'0.0001' , 0.58])
x.add_row(["No_Text","LR",'0.0001', 0.51])
print(x)
```

```
+------------+-------+-----------------+----------+
| Vectorizer | Model | Hyper Parameter | Test-AUC |
+------------+-------+-----------------+----------+
|    BOW     |   LR  |      0.0001     |   0.61   |
|   Tfidf    |   LR  |      0.0001     |   0.54   |
|  avg_w2v   |   LR  |      0.0001     |   0.58   |
| tfidf_w2v  |   LR  |      0.0001     |   0.58   |
|  No_Text   |   LR  |      0.0001     |   0.51   |
+------------+-------+-----------------+----------+
```

In [ ]: