Table of Contents

Author : SAURABH SINGHAI

DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports`

Feature	Description
project_subject_categories	• History & Civics • Literacy & Language • Math & Science • Music & The Arts • Special Needs • Warmth **Examples:** • Music & The Arts • Literacy & Language, Math & Science
school_state	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
project_subject_subcategories	One or more (comma-separated) subject subcategories for the project. **Examples:** • Literacy • Literature & Writing, Social Sciences
project_resource_summary	An explanation of the resources needed for the project. **Example:** • My students need hands on literacy materials to manage sensory needs!
project_essay_1	First application essay*
project_essay_2	Second application essay*
project_essay_3	Third application essay*
project_essay_4	Fourth application essay*
project_submitted_datetime	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
teacher_id	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
teacher_prefix	Teacher's title. One of the following enumerated values: • nan • Dr. • Mr. • Mrs. • Ms. • Teacher.
teacher_number_of_previously_posted_projects	Number of project applications previously submitted by the same teacher. **Example:** `2`

* See the section **Notes on the Essay Data** for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
id	A `project_id` value from the `train.csv` file. **Example:** `p036502`
description	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`
quantity	Quantity of the resource required. **Example:** `3`
price	Price of the resource required. **Example:** `9.95`

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved.

Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_3:__ "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

Import Necessary Libraries

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

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
```

```
IOPub data rate exceeded.
```

```
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.
```

Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna(' ')
```

Approved And Non Approved Projects

In [3]:

```
y_value_counts = project_data['project_is_approved'].value_counts()

print("Project Not Approved & Approved Count=\n",y_value_counts)

print("Number of projects approved for funding ", y_value_counts[1], "=", (y_value_counts[1]/(y_val
ue_counts[1]+y_value_counts[0]))*100,"%")

print("Number of projects not approved for funding ", y_value_counts[0], "=", (y_value_counts[0]/(y
_value_counts[1]+y_value_counts[0]))*100,"%")
```

```
Project Not Approved & Approved Count=
 1    92706
0    16542
Name: project_is_approved, dtype: int64
Number of projects approved for funding  92706 = 84.8583040422 %
Number of projects not approved for funding  16542 = 15.1416959578 %
```

Project Dataframe shape and Column Values

In [4]:

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

```
Number of data points in project data (109248, 17)
**************************************************
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_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']
```

Resource data shape and Column Values

In [5]:

```
print("Number of data points in resource data", resource_data.shape)
print('*'*50)
print(resource_data.columns.values)
resource_data.head(5)
```

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

Out[5]:

Out[5]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95
2	p069063	Cory Stories: A Kid's Book About Living With Adhd	1	8.45
3	p069063	Dixon Ticonderoga Wood-Cased #2 HB Pencils, Bo...	2	13.59
4	p069063	EDUCATIONAL INSIGHTS FLUORESCENT LIGHT FILTERS...	3	24.95

Preprocessing of `project_subject_categories`

In [6]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

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

Preprocessing of `project_subject_subcategories`

In [7]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
        j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

Preprocessing of `project_grade_category`

In [8]:

```
print(project_data["project_grade_category"].values[0:10])
```

```
['Grades PreK-2' 'Grades 6-8' 'Grades 6-8' 'Grades PreK-2' 'Grades PreK-2'
 'Grades 3-5' 'Grades 6-8' 'Grades 3-5' 'Grades PreK-2' 'Grades PreK-2']
```

In [9]:

```
project_data["project_grade_category"] = project_data["project_grade_category"].str.replace(" ", "_
")
project_data["project_grade_category"] = project_data["project_grade_category"].str.replace("-", "_
")

print(project_data["project_grade_category"].values[0:10])
```

```
['Grades_PreK_2' 'Grades_6_8' 'Grades_6_8' 'Grades_PreK_2' 'Grades_PreK_2'
 'Grades_3_5' 'Grades_6_8' 'Grades_3_5' 'Grades_PreK_2' 'Grades_PreK_2']
```

Create new column 'Essay' by merging all project Essays

In [10]:

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

In [11]:

```
project_data['essay'].head(5)
```

Out[11]:

```
0    My students are English learners that are work...
1    Our students arrive to our school eager to lea...
2    \r\n\"True champions aren't always the ones th...
3    I work at a unique school filled with both ESL...
4    Our second grade classroom next year will be m...
Name: essay, dtype: object
```

In [12]:

```
# printing some random reviews
print(project_data['essay'].values[0])
```

```
My students are English learners that are working on English as their second or third languages. W
e are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of langua
ge to our school. \r\n\r\n We have over 24 languages represented in our English Learner program wi
th students at every level of mastery.  We also have over 40 countries represented with the
families within our school.  Each student brings a wealth of knowledge and experiences to us that
open our eyes to new cultures, beliefs, and respect.\"The limits of your language are the limits o
f your world.\"-Ludwig Wittgenstein  Our English learner's have a strong support system at home th
at begs for more resources.  Many times our parents are learning to read and speak English along s
ide of their children.  Sometimes this creates barriers for parents to be able to help their child
learn phonetics, letter recognition, and other reading skills.\r\n\r\nBy providing these dvd's and
players, students are able to continue their mastery of the English language even if no one at hom
e is able to assist.  All families with students within the Level 1 proficiency status, will be a
offered to be a part of this program.  These educational videos will be specially chosen by the En
glish Learner Teacher and will be sent home regularly to watch.  The videos are to help the child
develop early reading skills.\r\n\r\nParents that do not have access to a dvd player will have the
opportunity to check out a dvd player to use for the year.  The plan is to use these videos and ed
ucational dvd's for the years to come for other EL students.\r\nnannan
```

Use Decontraction function to decontract project essay

In [13]:

```
# 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 [14]:

```
sent = decontracted(project_data['essay'].values[1])
print(sent)
print("="*50)
```

```
Our students arrive to our school eager to learn. They are polite, generous, and strive to be the
best they can be. They know that with an education, they can succeed in life and help improve on t
he lives that they have now.\r\n\r\nOur school focuses on families with low incomes and tries to g
ive each student the education they deserve. While we do not have much, the students use the mater
ials they are given and do the very best they can with them.The projector we need for our school i
s very crucial for the academic improvement of our students. As technology continues to grow, ther
e are so many resources in the internet that we as teachers use to further the growth of our stude
nts. However, our school is very limited with resources - particularly, technology - and without i
t, we are at a disadvantage. One of the things that could really help our classrooms is a
projector.\r\n\r\n\r\nWith a projector, not only is it crucial with instruction, but also for the
growth of our students. With a projector, we can show presentations, documentaries, photos of hist
orical land sites, math problems and so much more. With a projector, we can make teaching and lear
ning easier while also targeting the different types of learners we have in our classrooms:  audit
ory, visual, kinesthetic, etc. \r\nnannan
==================================================
```

Remove line breaks

In [15]:

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

```
Our students arrive to our school eager to learn. They are polite, generous, and strive to be the
best they can be. They know that with an education, they can succeed in life and help improve on t
he lives that they have now.    Our school focuses on families with low incomes and tries to give
each student the education they deserve. While we do not have much, the students use the materials
they are given and do the very best they can with them.The projector we need for our school is ver
y crucial for the academic improvement of our students. As technology continues to grow, there are
so many resources in the internet that we as teachers use to further the growth of our students. H
owever, our school is very limited with resources - particularly, technology - and without it, we
are at a disadvantage. One of the things that could really help our classrooms is a projector.
With a projector, not only is it crucial with instruction, but also for the growth of our students
. With a projector, we can show presentations, documentaries, photos of historical land sites, mat
h problems and so much more. With a projector, we can make teaching and learning easier while also
targeting the different types of learners we have in our classrooms:  auditory, visual,
kinesthetic, etc.   nannan
```

Remove Special Chars

In [16]:

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

```
Our students arrive to our school eager to learn They are polite generous and strive to be the bes
t they can be They know that with an education they can succeed in life and help improve on the li
ves that they have now Our school focuses on families with low incomes and tries to give each stud
ent the education they deserve While we do not have much the students use the materials they are g
iven and do the very best they can with them The projector we need for our school is very crucial
for the academic improvement of our students As technology continues to grow there are so many res
ources in the internet that we as teachers use to further the growth of our students However our s
chool is very limited with resources particularly technology and without it we are at a
disadvantage One of the things that could really help our classrooms is a projector With a project
or not only is it crucial with instruction but also for the growth of our students With a projecto
r we can show presentations documentaries photos of historical land sites math problems and so
much more With a projector we can make teaching and learning easier while also targeting the diffe
rent types of learners we have in our classrooms auditory visual kinesthetic etc nannan
```

Remove Stopwards and Join the essays

In [17]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn', \
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [18]:

```
# Combining all the above stundents
def Text_cleaner(data):
    from tqdm import tqdm
    preprocessed_essays = []
    # tqdm is for printing the status bar
    for sentance in tqdm(data.values):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\"', ' ')
        sent = sent.replace('\\n', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
        preprocessed_essays.append(sent.lower().strip())
    return preprocessed_essays
```

In [19]:

```
# after preprocesing
preprocessed_essays=Text_cleaner(project_data['essay'])
```

```
100%|██████████████████████████████████████████████████████████████████████████| 109248/109248
[01:30<00:00, 1212.73it/s]
```

In [20]:

```
preprocessed_essays[1]
```

Out[20]:

```
'students arrive school eager learn polite generous strive best know education succeed life help i
mprove lives school focuses families low incomes tries give student education deserve not much stu
dents use materials given best projector need school crucial academic improvement students technol
ogy continues grow many resources internet teachers use growth students however school limited res
ources particularly technology without disadvantage one things could really help classrooms
projector projector not crucial instruction also growth students projector show presentations docu
mentaries photos historical land sites math problems much projector make teaching learning easier
also targeting different types learners classrooms auditory visual kinesthetic etc nannan'
```

Drop essay columns 1, 2, 3, 4

In [21]:

```
project_data['essay'] = preprocessed_essays
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
project_data.head()
```

Out[21]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Gra
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	2016-10-06 21:16:17	Gra
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	2016-07-11 01:10:09	Gra

Preprocessing of `project_title`

In [22]:

```
# similarly you can preprocess the titles also
preprocessed_project_title=Text_cleaner(project_data['project_title'])
```

```
100%|████████████████████████████████████████████████████████████████████████████| 109248/109248
[00:03<00:00, 27653.10it/s]
```

In [23]:

```
preprocessed_project_title[1]
```

Out[23]:

```
'wanted projector hungry learners'
```

Drop column project_title and use Cleaned_Title

In [24]:

```
project_data['Cleaned_title']= preprocessed_project_title
project_data.drop(['project_title'], axis=1, inplace=True)
```

Add up the price based on project id

In [25]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [26]:

```
project_data.columns
```

Out[26]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'Cleaned_title',
       'price', 'quantity'],
      dtype='object')
```

In [27]:

```
project_data.drop(['project_resource_summary'], axis=1, inplace=True)
project_data.drop(['Unnamed: 0'], axis=1, inplace=True)
project_data.drop(['id'], axis=1, inplace=True)
project_data.drop(['teacher_id'], axis=1, inplace=True)
```

Adding the word count of essay and title as new columns

In [28]:

```
project_data['essay_count']=project_data['essay'].str.len()
project_data['title_count']=project_data['Cleaned_title'].str.len()
```

we are going to consider

```
       - school_state : categorical data
       - clean_categories : categorical data
       - clean_subcategories : categorical data
       - project_grade_category : categorical data
       - teacher_prefix : categorical data

       - project_title : text data
       - text : text data
       - project_resource_summary: text data (optinal)

       - quantity : numerical (optinal)
       - teacher_number_of_previously_posted_projects : numerical
       - price : numerical

       -Length of words in essay
       -Length of words in title
```

Separate out the Dependant and independant variables

In [29]:

```
#https://stackoverflow.com/questions/29763620/how-to-select-all-columns-except-one-column-in-panda
s
X=project_data.loc[:, project_data.columns != 'project_is_approved']
y=project_data['project_is_approved']
X.shape
```

Out[29]:

```
(109248, 13)
```

Splitting data into Test,Train,CV

In [30]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split

X_train,X_test,y_train, y_test=train_test_split(X, y, test_size=0.3, random_state=0,stratify=y)
X_train,X_cv,y_train,y_cv=train_test_split(X_train, y_train, test_size=0.3,
random_state=0,stratify=y_train)

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

```
(53531, 13)
(32775, 13)
(22942, 13)
(53531,)
(32775,)
(22942,)
```

In [31]:

```
X.head(2)
```

Out[31]:

	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	teacher_number_of_previously_poste
0	Mrs.	IN	2016-12-05 13:43:57	Grades_PreK_2	0
1	Mr.	FL	2016-10-25 09:22:10	Grades_6_8	7

Vectorize the features

Vectorize the Categorical Features - categories

In [32]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['clean_categories'].values)

# we use the fitted CountVectorizer to transform the text to vector
X_train_clean_categories=vectorizer.transform(X_train['clean_categories'].values)
X_test_clean_categories=vectorizer.transform(X_test['clean_categories'].values)
X_cv_clean_categories=vectorizer.transform(X_cv['clean_categories'].values)


print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_clean_categories.shape)
```

```
['AppliedLearning', 'Care_Hunger', 'Health_Sports', 'History_Civics', 'Literacy_Language',
'Math_Science', 'Music_Arts', 'SpecialNeeds', 'Warmth']
Shape of matrix after one hot encodig  (53531, 9)
```

Vectorize the Categorical Features - subcategories

In [33]:

```
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_sub_categories=vectorizer.transform(X_train['clean_subcategories'].values)
X_test_clean_sub_categories=vectorizer.transform(X_test['clean_subcategories'].values)
X_cv_clean_sub_categories=vectorizer.transform(X_cv['clean_subcategories'].values)


print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_clean_sub_categories.shape)
```

```
['AppliedSciences', 'Care_Hunger', 'CharacterEducation', 'Civics_Government',
'College_CareerPrep', 'CommunityService', 'ESL', 'EarlyDevelopment', 'Economics',
'EnvironmentalScience', 'Extracurricular', 'FinancialLiteracy', 'ForeignLanguages', 'Gym_Fitness',
'Health_LifeScience', 'Health_Wellness', 'History_Geography', 'Literacy', 'Literature_Writing', 'M
athematics', 'Music', 'NutritionEducation', 'Other', 'ParentInvolvement', 'PerformingArts', 'Socia
lSciences', 'SpecialNeeds', 'TeamSports', 'VisualArts', 'Warmth']
Shape of matrix after one hot encodig  (53531, 30)
```

Vectorize the Categorical Features - school state

In [34]:

```
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['school_state'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_skl_state=vectorizer.transform(X_train['school_state'].values)
X_test_skl_state=vectorizer.transform(X_test['school_state'].values)
X_cv_skl_state=vectorizer.transform(X_cv['school_state'].values)


print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_skl_state.shape)
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'K
S', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV
', 'WY']
Shape of matrix after one hot encodig  (53531, 51)
```

Vectorize the Categorical Features - teacher prefix

In [35]:

```
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['teacher_prefix'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix=vectorizer.transform(X_train['teacher_prefix'].values)
X_test_teacher_prefix=vectorizer.transform(X_test['teacher_prefix'].values)
X_cv_teacher_prefix=vectorizer.transform(X_cv['teacher_prefix'].values)


print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_teacher_prefix.shape)
```

```
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher']
Shape of matrix after one hot encodig  (53531, 5)
```

Vectorize the Categorical Features - project_grade_category

In [36]:

```
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['project_grade_category'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade_category=vectorizer.transform(X_train['project_grade_category'].values)
X_test_project_grade_category=vectorizer.transform(X_test['project_grade_category'].values)
X_cv_project_grade_category=vectorizer.transform(X_cv['project_grade_category'].values)


print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_project_grade_category.shape)
```

```
['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
Shape of matrix after one hot encodig  (53531, 4)
```

Vectorize the Numerical Features - teacher_number_of_previously_posted_projects

In [37]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
# https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html#sklearn.preprocessing.Nor
er

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
from sklearn import preprocessing

X=X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)
ppp_scalar = Normalizer().fit(X)

# Now standardize the data with above maen and variance.
X_train_prev_proj = ppp_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].v
alues.reshape(1,-1))
X_test_prev_proj =
ppp_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_cv_prev_proj = ppp_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.
reshape(1,-1))

# Make sure to reshape it back Else it will give error while merging
X_train_prev_proj_standardized=X_train_prev_proj.reshape(-1,1)

X_test_prev_proj_standardized=X_test_prev_proj.reshape(-1,1)

X_cv_prev_proj_standardized=X_cv_prev_proj.reshape(-1,1)


print("X_train_price_standardized dimension is-->", X_train_prev_proj_standardized.ndim)
print("=="*25)
print("X_train_price_standardized shape is--> ", X_train_prev_proj_standardized.shape)
print("=="*25)
print("After normalize-->\n",X_test_prev_proj_standardized[:10])
print("=="*25)
print("After normalize-->\n",X_cv_prev_proj_standardized[:10])
print("=="*25)
```

```
X_train_price_standardized dimension is--> 2
==================================================
X_train_price_standardized shape is-->  (53531, 1)
==================================================
After normalize-->
 [[ 0.00107381]
 [ 0.00071588]
 [ 0.00876948]
 [ 0.00053691]
 [ 0.00053691]
 [ 0.0028635 ]
 [ 0.00304247]
 [ 0.        ]
 [ 0.00196866]
 [ 0.00071588]]
==================================================
After normalize-->
 [[ 0.        ]
 [ 0.0011172 ]
 [ 0.00044688]
 [ 0.00044688]
 [ 0.        ]
 [ 0.        ]
 [ 0.        ]
 [ 0.00022344]
 [ 0.        ]
 [ 0.00558599]]
==================================================
```

Vectorize the Numerical Features - price

In [38]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
# https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html#sklearn.preprocessing.Nor
er

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
from sklearn import preprocessing

X=X_train['price'].values.reshape(1,-1)
price_scalar = Normalizer().fit(X)

# Now standardize the data with above maen and variance.
X_train_price_standardized = price_scalar.transform(X_train['price'].values.reshape(1,-1))
X_test_price_standardized = price_scalar.transform(X_test['price'].values.reshape(1,-1))
X_cv_price_standardized = price_scalar.transform(X_cv['price'].values.reshape(1,-1))

# Make sure to reshape it back Else it will give error while merging
X_train_price_standardized=X_train_price_standardized.reshape(-1,1)

X_test_price_standardized=X_test_price_standardized.reshape(-1,1)

X_cv_price_standardized=X_cv_price_standardized.reshape(-1,1)


print("X_train_price_standardized dimension is-->", X_train_price_standardized.ndim)
print("=="*25)
print("X_train_price_standardized shape is--> ", X_train_price_standardized.shape)
print("=="*25)
print("After normalize-->\n",X_test_price_standardized[:10])
print("=="*25)
print("After normalize-->\n",X_cv_price_standardized[:10])
print("=="*25)
```

```
X_train_price_standardized dimension is--> 2
==================================================
X_train_price_standardized shape is-->  (53531, 1)
==================================================
After normalize-->
 [[ 0.00434408]
 [ 0.00157362]
 [ 0.00158861]
 [ 0.00549255]
 [ 0.00454119]
 [ 0.00117084]
 [ 0.00208285]
 [ 0.00181301]
 [ 0.00119778]
 [ 0.00173794]]
==================================================
After normalize-->
 [[ 0.00239887]
 [ 0.00199007]
 [ 0.00138773]
 [ 0.01330296]
 [ 0.00199856]
 [ 0.00334289]
 [ 0.00516327]
 [ 0.00019611]
 [ 0.00305634]
 [ 0.0001918 ]]
==================================================
```

Vectorize the Numerical Features - quantity

In [39]:

```
X=X_train['quantity'].values.reshape(1,-1)
quantity_scalar = Normalizer().fit(X)

# Now standardize the data with above maen and variance.
X_train_quantity_standardized = quantity_scalar.transform(X_train['quantity'].values.reshape(1,-1))
X_test_quantity_standardized = quantity_scalar.transform(X_test['quantity'].values.reshape(1,-1))
X_cv_quantity_standardized = quantity_scalar.transform(X_cv['quantity'].values.reshape(1,-1))

X_train_quantity_standardized =X_train_quantity_standardized.reshape(-1,1)
X_test_quantity_standardized = X_test_quantity_standardized.reshape(-1,1)
X_cv_quantity_standardized =X_cv_quantity_standardized.reshape(-1,1)
```

Vectorize the Numerical Features - essay count

In [40]:

```
X=X_train['essay_count'].values.reshape(1,-1)
count_scalar = Normalizer().fit(X)


# Now standardize the data with above maen and variance.
X_train_essay_count_standardized = count_scalar.transform(X_train['essay_count'].values.reshape(1,
-1))
X_test_essay_count_standardized = count_scalar.transform(X_test['essay_count'].values.reshape(1,-1)
)
X_cv_essay_count_standardized = count_scalar.transform(X_cv['essay_count'].values.reshape(1,-1))

X_train_essay_count_standardized = X_train_essay_count_standardized.reshape(-1,1)
X_test_essay_count_standardized =X_test_essay_count_standardized.reshape(-1,1)
X_cv_essay_count_standardized = X_cv_essay_count_standardized.reshape(-1,1)
```

Vectorize the Numerical Features - title count

In [41]:

```
X=X_train['title_count'].values.reshape(1,-1)
count_scalar = Normalizer().fit(X)

# Now standardize the data with above maen and variance.
X_train_title_count_standardized = count_scalar.transform(X_train['title_count'].values.reshape(1,
-1))
X_test_title_count_standardized = count_scalar.transform(X_test['title_count'].values.reshape(1,-1)
)
X_cv_title_count_standardized = count_scalar.transform(X_cv['title_count'].values.reshape(1,-1))

X_train_title_count_standardized = X_train_title_count_standardized.reshape(-1,1)
X_test_title_count_standardized = X_test_title_count_standardized.reshape(-1,1)
X_cv_title_count_standardized = X_cv_title_count_standardized.reshape(-1,1)
```

Vectorizing Text data

Bag of words - essay

In [42]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'])

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


print("Shape of matrix after one hot encodig ",X_train_essay_bow.shape)
```

```
Shape of matrix after one hot encodig  (53531, 5000)
```

Bag of words - cleaned title

In [43]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer = CountVectorizer(min_df=5)
vectorizer.fit(X_train['Cleaned_title'])

# we use the fitted CountVectorizer to convert the text to vector
X_train_cleaned_title_bow=vectorizer.transform(X_train['Cleaned_title'].values)
X_test_cleaned_title_bow=vectorizer.transform(X_test['Cleaned_title'].values)
X_cv_cleaned_title_bow=vectorizer.transform(X_cv['Cleaned_title'].values)


print("Shape of matrix after one hot encodig ",X_train_cleaned_title_bow.shape)
```

```
Shape of matrix after one hot encodig  (53531, 3346)
```

TFIDF vectorizer - essay

In [44]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'])

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf=vectorizer.transform(X_train['essay'].values)
X_test_essay_tfidf=vectorizer.transform(X_test['essay'].values)
X_cv_essay_tfidf=vectorizer.transform(X_cv['essay'].values)


print("Shape of matrix after one hot encodig ",X_train_essay_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (53531, 5000)
```

TFIDF vectorizer - cleaned tittle

In [45]:

```
# Similarly you can vectorize for title alsovectorizer = TfidfVectorizer(min_df=10)
vectorizer = TfidfVectorizer(min_df=5)

# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer.fit(X_train['Cleaned_title'])

# we use the fitted CountVectorizer to convert the text to vector
X_train_cleaned_title_tfidf=vectorizer.transform(X_train['Cleaned_title'].values)
X_test_cleaned_title_tfidf=vectorizer.transform(X_test['Cleaned_title'].values)
X_cv_cleaned_title_tfidf=vectorizer.transform(X_cv['Cleaned_title'].values)
```

```
print("Shape of matrix after one hot encodig ",X_train_cleaned_title_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (53531, 3346)
```

Using Pretrained Models: Avg W2V

In [46]:

```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

# ============================
Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

# ============================

words = []
for i in preproced_texts:
    words.extend(i.split(' '))

for i in preproced_titles:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and our coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/

import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_courpus, f)


'''
```

Out[46]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading G
```

```
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
=============================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus",
len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\r
print("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [47]:

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

In [48]:

```python
# average Word2Vec
# compute average word2vec for each review.
def avg_w2v_vectors(preprocessed_essays):
    avg_w2v_vectors_text = []; # the avg-w2v for each sentence/review is stored in this list
    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_text.append(vector)
    return avg_w2v_vectors_text

X_train_essay_w2v=avg_w2v_vectors(X_train['essay'])
X_test_essay_w2v=avg_w2v_vectors(X_test['essay'])
X_cv_essay_w2v=avg_w2v_vectors(X_cv['essay'])

X_train_cleaned_title_w2v=avg_w2v_vectors(X_train['Cleaned_title'])
X_test_cleaned_title_w2v=avg_w2v_vectors(X_test['Cleaned_title'])
X_cv_cleaned_title_w2v=avg_w2v_vectors(X_cv['Cleaned_title'])
```

```
100%|██████████████████████████████████████████████████████████████████████████| 53531/53531
[00:16<00:00, 3293.93it/s]
100%|██████████████████████████████████████████████████████████████████████████| 32775/32775
[00:10<00:00, 3086.73it/s]
100%|██████████████████████████████████████████████████████████████████████████| 22942/22942
[00:07<00:00, 3265.88it/s]
100%|█████████████████████████████████████████████████████████████████████████| 53531/53531
[00:00<00:00, 58675.92it/s]
100%|█████████████████████████████████████████████████████████████████████████| 32775/32775
[00:00<00:00, 55678.92it/s]
100%|█████████████████████████████████████████████████████████████████████████| 22942/22942
[00:00<00:00, 51208.30it/s]
```

Using Pretrained Models: TFIDF weighted W2V - Essay

In [49]:

```python
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'])
# 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_essay = set(tfidf_model.get_feature_names())
print(len(tfidf_words_essay))
```

```
42499
```

In [50]:

```python
# average Word2Vec
# compute average word2vec for each review.
def tfidf_w2v_vectors(tfidf_words,preprocessed_essays):
    tfidf_w2v_vectors_text = []; # the avg-w2v for each sentence/review is stored in this list
    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
value((sentence.count(word)/len(sentence.split())))
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting
the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors_text.append(vector)
    return tfidf_w2v_vectors_text

X_train_essay_tfidf_w2v=tfidf_w2v_vectors(tfidf_words_essay,X_train['essay'])
X_test_essay_tfidf_w2v=tfidf_w2v_vectors(tfidf_words_essay,X_test['essay'])
X_cv_essay_tfidf_w2v=tfidf_w2v_vectors(tfidf_words_essay,X_cv['essay'])
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 53531/53531 [01:
58<00:00, 452.16it/s]
100%|██████████████████████████████████████████████████████████████████████████████| 32775/32775 [01:
12<00:00, 454.01it/s]
100%|██████████████████████████████████████████████████████████████████████████████| 22942/22942 [00:
51<00:00, 447.76it/s]
```

Using Pretrained Models: TFIDF weighted W2V - Cleaned Title

In [51]:

```python
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['Cleaned_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_Cleaned_title = set(tfidf_model.get_feature_names())
print(len(tfidf_words_Cleaned_title))
```

```
12119
```

In [52]:

```python
X_train_cleaned_title_tfidf_w2v=tfidf_w2v_vectors(tfidf_words_Cleaned_title,X_train['Cleaned_title'
])
X_test_cleaned_title_tfidf_w2v=tfidf_w2v_vectors(tfidf_words_Cleaned_title,X_test['Cleaned_title']
)
X_cv_cleaned_title_tfidf_w2v=tfidf_w2v_vectors(tfidf_words_Cleaned_title,X_cv['Cleaned_title'])
```

```
100%|████████████████████████████████████████████████████████████████████████████| 53531/53531
[00:01<00:00, 26779.73it/s]
100%|████████████████████████████████████████████████████████████████████████████| 32775/32775
[00:01<00:00, 29829.80it/s]
100%|████████████████████████████████████████████████████████████████████████████| 22942/22942
[00:00<00:00, 28951.86it/s]
```

In [53]:

```python
X_train_prev_proj=X_train['teacher_number_of_previously_posted_projects'][:,np.newaxis]

X_test_prev_proj=X_test['teacher_number_of_previously_posted_projects'][:,np.newaxis]
```

```
X_cv_prev_proj=X_cv['teacher_number_of_previously_posted_projects'][:,np.newaxis]
```

Merging all the above features

BOW

In [54]:

```
# 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 matirx :)
X_train_bow = hstack((X_train_clean_categories,
X_train_clean_sub_categories,X_train_skl_state,X_train_teacher_prefix,

X_train_project_grade_category,X_train_prev_proj_standardized,X_train_price_standardized,X_train_qu
antity_standardized,X_train_prev_proj,
            X_train_essay_bow,X_train_cleaned_title_bow,X_train_essay_count_standardized,X_train_ti
tle_count_standardized
            )).toarray()


X_test_bow = hstack((X_test_clean_categories,
X_test_clean_sub_categories,X_test_skl_state,X_test_teacher_prefix,
            X_test_project_grade_category,X_test_prev_proj_standardized,X_test_price_standardized,X
_test_quantity_standardized,X_test_prev_proj,
            X_test_essay_bow,X_test_cleaned_title_bow,X_test_essay_count_standardized,X_test_title_
count_standardized
            )).toarray()


X_cv_bow = hstack((X_cv_clean_categories,
X_cv_clean_sub_categories,X_cv_skl_state,X_cv_teacher_prefix,
            X_cv_project_grade_category,X_cv_prev_proj_standardized,X_cv_price_standardized,X_cv_qu
antity_standardized,X_cv_prev_proj,

X_cv_essay_bow,X_cv_cleaned_title_bow,X_cv_essay_count_standardized,X_cv_title_count_standardized
            )).toarray()


print(X_train_bow.shape)
print(X_test_bow.shape)
print(X_cv_bow.shape)
```

```
(53531, 8451)
(32775, 8451)
(22942, 8451)
```

TFIDF

In [55]:

```
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_tfidf = hstack((X_train_clean_categories, X_train_clean_sub_categories,X_train_skl_state,X
_train_teacher_prefix,

X_train_project_grade_category,X_train_prev_proj_standardized,X_train_price_standardized,X_train_qu
antity_standardized,X_train_prev_proj,
            X_train_essay_tfidf,X_train_cleaned_title_tfidf,X_train_essay_count_standardized,X_trai
n_title_count_standardized
            )).toarray()
```

In [56]:

```
X_test_tfidf = hstack((X_test_clean_categories,
X_test_clean_sub_categories,X_test_skl_state,X_test_teacher_prefix,
            X_test_project_grade_category,X_test_prev_proj_standardized,X_test_price_standardized,X
_test_quantity_standardized,X_test_prev_proj,
            X_test_essay_tfidf,X_test_cleaned_title_tfidf,X_test_essay_count_standardized,X_test_ti
            X_test_essay_tfidf,X_test_cleaned_title_tfidf,X_test_essay_count_standardized,X_test_ti
tle_count_standardized
            )).toarray()
```

In [57]:

```
X_cv_tfidf = hstack((X_cv_clean_categories,
X_cv_clean_sub_categories,X_cv_skl_state,X_cv_teacher_prefix,
            X_cv_project_grade_category,X_cv_prev_proj_standardized,X_cv_price_standardized,X_cv_qu
antity_standardized,X_cv_prev_proj,

X_cv_essay_tfidf,X_cv_cleaned_title_tfidf,X_cv_essay_count_standardized,X_cv_title_count_standardiz
d
            )).toarray()


print(X_train_tfidf.shape)
print(X_test_tfidf.shape)
print(X_cv_tfidf.shape)
```

```
(53531, 8451)
(32775, 8451)
(22942, 8451)
```

Word2Vec

In [58]:

```
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_w2v = hstack((X_train_clean_categories,
X_train_clean_sub_categories,X_train_skl_state,X_train_teacher_prefix,

X_train_project_grade_category,X_train_prev_proj_standardized,X_train_price_standardized,X_train_qu
antity_standardized,X_train_prev_proj,
            X_train_essay_w2v,X_train_cleaned_title_w2v,X_train_essay_count_standardized,X_train_ti
tle_count_standardized
            )).toarray()


X_test_w2v = hstack((X_test_clean_categories,
X_test_clean_sub_categories,X_test_skl_state,X_test_teacher_prefix,
            X_test_project_grade_category,X_test_prev_proj_standardized,X_test_price_standardized,X
_test_quantity_standardized,X_test_prev_proj,
            X_test_essay_w2v,X_test_cleaned_title_w2v,X_test_essay_count_standardized,X_test_title_
count_standardized
            )).toarray()


X_cv_w2v = hstack((X_cv_clean_categories,
X_cv_clean_sub_categories,X_cv_skl_state,X_cv_teacher_prefix,
            X_cv_project_grade_category,X_cv_prev_proj_standardized,X_cv_price_standardized,X_cv_qu
antity_standardized,X_cv_prev_proj,

X_cv_essay_w2v,X_cv_cleaned_title_w2v,X_cv_essay_count_standardized,X_cv_title_count_standardized
            )).toarray()


print(X_train_w2v.shape)
print(X_test_w2v.shape)
print(X_cv_w2v.shape)
```

```
(53531, 705)
(32775, 705)
(22942, 705)
```

TFIDF- WORD2VEC

In [59]:

```
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_tfidf_w2v = hstack((X_train_clean_categories,
X_train_tfidf_w2v = hstack((X_train_clean_categories,
X_train_clean_sub_categories,X_train_skl_state,X_train_teacher_prefix,

X_train_project_grade_category,X_train_prev_proj_standardized,X_train_price_standardized,X_train_qu
antity_standardized,X_train_prev_proj,

X_train_essay_tfidf_w2v,X_train_cleaned_title_tfidf_w2v,X_train_essay_count_standardized,X_train_ti
tle_count_standardized
            )).toarray()


X_test_tfidf_w2v = hstack((X_test_clean_categories, X_test_clean_sub_categories,X_test_skl_state,X
_test_teacher_prefix,
            X_test_project_grade_category,X_test_prev_proj_standardized,X_test_price_standardized,X
_test_quantity_standardized,X_test_prev_proj,
            X_test_essay_tfidf_w2v,X_test_cleaned_title_tfidf_w2v,X_test_essay_count_standardized,X
_test_title_count_standardized
            )).toarray()


X_cv_tfidf_w2v = hstack((X_cv_clean_categories,
X_cv_clean_sub_categories,X_cv_skl_state,X_cv_teacher_prefix,
            X_cv_project_grade_category,X_cv_prev_proj_standardized,X_cv_price_standardized,X_cv_qu
antity_standardized,X_cv_prev_proj,
            X_cv_essay_tfidf_w2v,X_cv_cleaned_title_tfidf_w2v,X_cv_essay_count_standardized,X_cv_ti
tle_count_standardized
            )).toarray()


print(X_train_tfidf_w2v.shape)
print(X_test_tfidf_w2v.shape)
print(X_cv_tfidf_w2v.shape)
```

```
(53531, 705)
(32775, 705)
(22942, 705)
```

MERGE ONLY CATEGORICAL AND NUMERICAL DATA

In [60]:

```
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_no_text = hstack((X_train_clean_categories, X_train_clean_sub_categories,X_train_skl_state
,X_train_teacher_prefix,

X_train_project_grade_category,X_train_prev_proj_standardized,X_train_price_standardized,X_train_qu
antity_standardized,X_train_prev_proj
           )).toarray()


X_test_no_text = hstack((X_test_clean_categories,
X_test_clean_sub_categories,X_test_skl_state,X_test_teacher_prefix,
            X_test_project_grade_category,X_test_prev_proj_standardized,X_test_price_standardized,X
_test_quantity_standardized,X_test_prev_proj
            )).toarray()


X_cv_no_text = hstack((X_cv_clean_categories,
X_cv_clean_sub_categories,X_cv_skl_state,X_cv_teacher_prefix,
            X_cv_project_grade_category,X_cv_prev_proj_standardized,X_cv_price_standardized,X_cv_qu
antity_standardized,X_cv_prev_proj
            )).toarray()


print(X_train_no_text.shape)
print(X_test_no_text.shape)
print(X_cv_no_text.shape)
```

```
(53531, 103)
(32775, 103)
(22942, 103)
```

Shapes after merge

In [61]:

```
print(X_train_bow.shape)
print(X_cv_bow.shape)
print(X_test_bow.shape)

print('='*50)
print(X_train_tfidf.shape)
print(X_cv_tfidf.shape)
print(X_test_tfidf.shape)

print('='*50)
print(X_train_w2v.shape)
print(X_cv_w2v.shape)
print(X_test_w2v.shape)

print('='*50)
print(X_train_tfidf_w2v.shape)
print(X_cv_tfidf_w2v.shape)
print(X_test_tfidf_w2v.shape)

print('='*50)
print(y_train.shape)
print(y_test.shape)
print(y_cv.shape)
```

```
(53531, 8451)
(22942, 8451)
(32775, 8451)
==================================================
(53531, 8451)
(22942, 8451)
(32775, 8451)
==================================================
(53531, 705)
(22942, 705)
(32775, 705)
==================================================
(53531, 705)
(22942, 705)
(32775, 705)
==================================================
(53531,)
(32775,)
(22942,)
```

In [62]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

summary=[]
```

Assignment 5: Logistic Regression

Applying Logistic Regression on BOW, SET 1

Hyperparameter tuning for finiding optimal ALPHA using ROC_AUC_Score

In [63]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

summary=[]
roc_auc_score_cv_bow_dict={}
roc_auc_score_train_bow_dict={}
cValues = [.0001,.001, .01, .1, 1, 10, 100, 1000]
```

```
for i in tqdm(cValues):
    lr = LogisticRegression(penalty='l2',C= i,class_weight='balanced')

    # 0.fitting the model on X_train
    lr.fit(X_train_bow, y_train)

    # 1.predict the response on the CV data
    pred_bow_cv = lr.predict_proba(X_cv_bow)

    # 2.evaluate CV roc_auc
    roc_auc_cv = roc_auc_score(y_cv,pred_bow_cv[:,1])

    # 3.insert into dict of CV data
    roc_auc_score_cv_bow_dict[i]=roc_auc_cv

    # 1.predict the response on the TRAIN data
    pred_bow_train = lr.predict_proba(X_train_bow)

    # 2.evaluate train roc_auc
    roc_auc_train =roc_auc_score(y_train,pred_bow_train[:,1])

    # 3.insert into dict of train data
    roc_auc_score_train_bow_dict[i]=roc_auc_train



print(roc_auc_score_cv_bow_dict)
print(roc_auc_score_train_bow_dict)
```

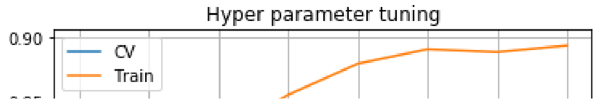
```
100%|██████████████████████████████████████████████████████████| 8/8 [13:58<00:00, 104.84s/it]
```

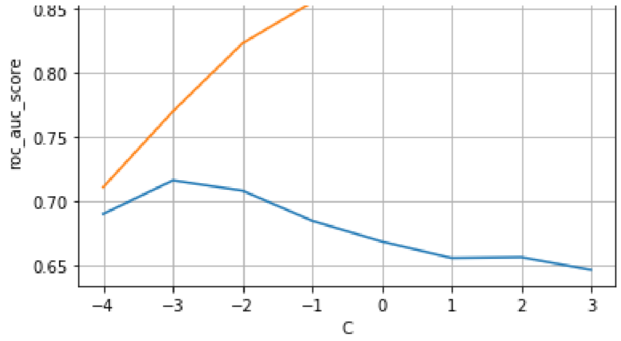
```
{0.0001: 0.69022771703123464, 0.001: 0.71625750134936461, 0.01: 0.70827603782786785, 0.1: 0.68484780953442159, 1: 0.66866066854436246, 10: 0.65591414113992952, 100: 0.65655488971524534, 1000: 0.64670435069687304}
{0.0001: 0.71101637787815142, 0.001: 0.76999552906146707, 0.01: 0.82298649350681807, 0.1: 0.85502805397816983, 1: 0.87938099623787669, 10: 0.89065178657057831, 100: 0.88846243633475597, 1000: 0.89350069054964298}
```

Plot ROC_AUC_score VS different ALPHA values (Train and CV set)

In [64]:

```
#https://stackoverflow.com/questions/37266341/plotting-a-python-dict-in-order-of-key-values/37266356
import math

lists1 = sorted(roc_auc_score_cv_bow_dict.items())
x1, y1 = zip(*lists1) # unpack a list of pairs into two tuples
lists2 = sorted(roc_auc_score_train_bow_dict.items())
x2, y2 = zip(*lists2) # unpack a list of pairs into two tuples

x1=[math.log10(i) for i in list(x1)]
x2=[math.log10(i) for i in list(x2)]

plt.xlabel('C')
plt.ylabel('roc_auc_score')

plt.title('Hyper parameter tuning')

plt.plot(x1, y1,label="CV")
plt.plot(x2, y2,label='Train')

plt.legend()
plt.grid()
plt.show()
```





Find the best ALPHA value from the result

In [65]:

```
# This function returns the key (K) with max value in the dictionary
def find_highest_hyperparam(mydict):
    #print dictionary contents
    print("Dictionary Contents-->\n",mydict)
    print("="*50)
    #find maximum value from dixtionary
    maxval=max(mydict.values())

    # make a list of keys
    keys = list(mydict.keys())
    print("Keys in the list are-->\n", keys)
    print("="*50)
    # make a list of values
    vals = list(mydict.values())
    print("Values in the list are-->\n", vals)
    print("="*50)
    # find the index of max value and use it to find corresponding key
    k=(keys[vals.index(maxval)])
    print("Maximum k is {0} ".format(k))
    print("="*50)
    return(k)

print(find_highest_hyperparam(roc_auc_score_cv_bow_dict))
```

```
Dictionary Contents-->
 {0.0001: 0.69022771703123464, 0.001: 0.71625750134936461, 0.01: 0.70827603782786785, 0.1:
0.68484780953442159, 1: 0.66866066854436246, 10: 0.65591414113992952, 100: 0.65655488971524534, 10
00: 0.64670435069687304}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.69022771703123464, 0.71625750134936461, 0.70827603782786785, 0.68484780953442159,
0.66866066854436246, 0.65591414113992952, 0.65655488971524534, 0.64670435069687304]
==================================================
Maximum k is 0.001
==================================================
0.001
```

Train the model on the optimal ALPHA value and run the Test Dataset

```
    -- Plot the ROC curve for BOW using the test and train Dataset
```

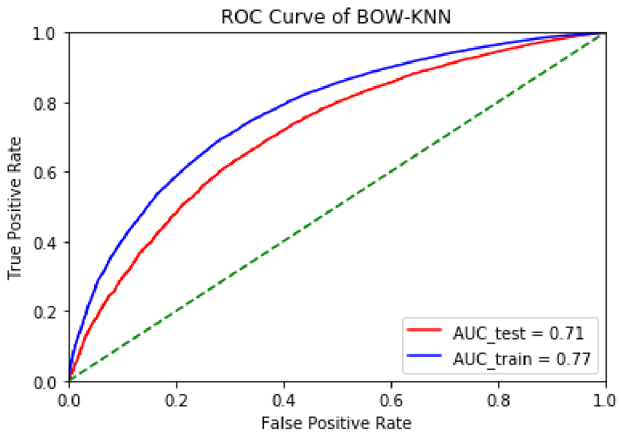
In [66]:

```
# train model on the best k
lr = LogisticRegression(penalty='l2', C=
find_highest_hyperparam(roc_auc_score_cv_bow_dict),class_weight='balanced')
# fitting the model on train

lr.fit(X_train_bow, y_train)

pred_bow_test = lr.predict(X_test_bow) # **we will use it in confusion matrix
```

```
pred_bow_train = lr.predict(X_train_bow) # **we will use it in confusion matrix

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-scip_scores = knn.predict_proba(X_test)
pred_bow_test_scores=lr.predict_proba(X_test_bow)
print("pred_bow_test_scores-->\n", pred_bow_test_scores)

pred_bow_train_scores=lr.predict_proba(X_train_bow)
print("pred_bow_train_scores-->\n", pred_bow_train_scores)

# Calculate fpr, tpr for test
fpr_test, tpr_test, threshold_test = roc_curve(y_test, pred_bow_test_scores[:,1])

# Calculate fpr, tpr for train
fpr_train, tpr_train, threshold_train = roc_curve(y_train, pred_bow_train_scores[:,1])

# Calculate auc from fpr, tpr for test
roc_auc_test = auc(fpr_test, tpr_test)

# Calculate auc from fpr, tpr for train
roc_auc_train = auc(fpr_train, tpr_train)

#save ALPHA & roc_auc_test score (AUC of fpr_test and tpr_test) for final summary
summary.append(['BOW',find_highest_hyperparam(roc_auc_score_cv_bow_dict),roc_auc_test])

plt.title('Receiver Operating Characteristic')
plt.plot(fpr_test, tpr_test, 'r', label = 'AUC_test = %0.2f' % roc_auc_test)

plt.plot(fpr_train, tpr_train, 'b', label = 'AUC_train = %0.2f' % roc_auc_train)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of BOW-KNN')
plt.show()
```

```
Dictionary Contents-->
 {0.0001: 0.69022771703123464, 0.001: 0.71625750134936461, 0.01: 0.70827603782786785, 0.1: 0.68484780953442159, 1: 0.66866066854436246, 10: 0.65591414113992952, 100: 0.65655488971524534, 1000: 0.64670435069687304}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.69022771703123464, 0.71625750134936461, 0.70827603782786785, 0.68484780953442159, 0.66866066854436246, 0.65591414113992952, 0.65655488971524534, 0.64670435069687304]
==================================================
Maximum k is 0.001
==================================================
pred_bow_test_scores-->
 [[ 0.68719591  0.31280409]
 [ 0.2906323   0.7093677 ]
 [ 0.26707153  0.73292847]
 ...,
 [ 0.54952012  0.45047988]
 [ 0.51186613  0.48813387]
 [ 0.35549141  0.64450859]]
pred_bow_train_scores-->
 [[ 0.33848587  0.66151413]
 [ 0.24350352  0.75649648]
 [ 0.28349582  0.71650418]
 ...,
 [ 0.39005275  0.60994725]
 [ 0.35908812  0.64091188]
 [ 0.5219915   0.4780085 ]]
Dictionary Contents-->
 {0.0001: 0.69022771703123464, 0.001: 0.71625750134936461, 0.01: 0.70827603782786785, 0.1: 0.68484780953442159, 1: 0.66866066854436246, 10: 0.65591414113992952, 100: 0.65655488971524534, 1000: 0.64670435069687304}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
```

```
==================================================
Values in the list are-->
 [0.69022771703123464, 0.71625750134936461, 0.70827603782786785, 0.68484780953442159, 
0.66866066854436246, 0.65591414113992952, 0.65655488971524534, 0.64670435069687304]
==================================================
Maximum k is 0.001
==================================================
```



Get the confusion matrix for the BOW

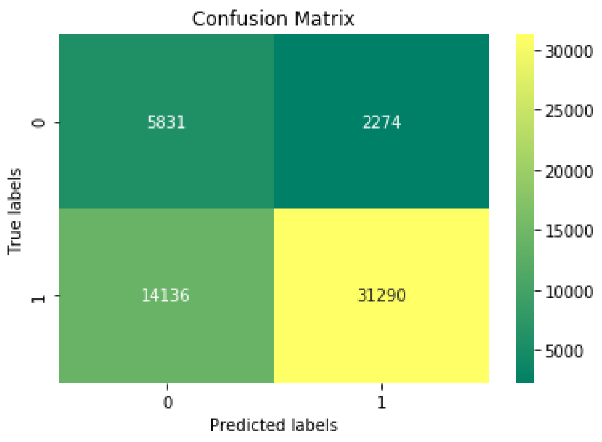
In [67]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
# Make confusion matrix for y_train vs predicted(X_train_bow)
cm = confusion_matrix(y_train, pred_bow_train)
print(cm)

sns.heatmap(cm, annot=True, ax = ax,cmap="summer", fmt='g')
print("Training CM for BOW")
# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
```

```
[[ 5831  2274]
 [14136 31290]]
Training CM for BOW
```

Out[67]:

```
Text(0.5,1,'Confusion Matrix')
```



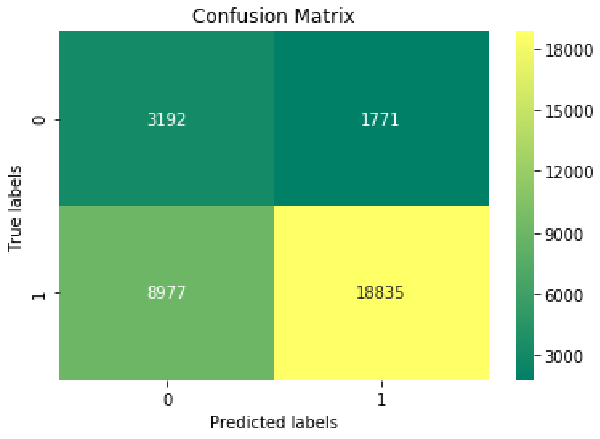
In [68]:

```
print("="*50)
print("="*50)
print("Testing CM for BOW")
ax= plt.subplot()
# Make confusion matrix for y_test vs predicted(X_test_bow)
cm = confusion_matrix(y_test, pred_bow_test)
print(cm)

sns.heatmap(cm, annot=True, ax = ax,cmap="summer",fmt='g')
# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
```

```
==================================================
Testing CM for BOW
[[ 3192  1771]
 [ 8977 18835]]
```

Out[68]:

```
Text(0.5,1,'Confusion Matrix')
```



Applying Logistic Regression on TFIDF, SET 2

Hyperparameter tuning for finiding optimal ALPHA using ROC_AUC_Score

In [69]:

```
roc_auc_score_cv_tfidf_dict={}
roc_auc_score_train_tfidf_dict={}

cValues = [.0001,.001, .01, .1, 1, 10, 100, 1000]

for i in tqdm(cValues):
    lr = LogisticRegression(penalty='l2',C= i,class_weight='balanced')

     # fitting the model on crossvalidation train
    lr.fit(X_train_tfidf, y_train)

    # predict the response on the crossvalidation train
    pred_tfidf_cv = lr.predict_proba(X_cv_tfidf)

    #evaluate CV roc_auc
    roc_auc_cv =roc_auc_score(y_cv,pred_tfidf_cv[:,1])

    #insert into dict
    roc_auc_score_cv_tfidf_dict[i]=roc_auc_cv

    # predict the response on the train
    pred_tfidf_train = lr.predict_proba(X_train_tfidf)

    #evaluate train roc_auc
    roc_auc_train =roc_auc_score(y_train,pred_tfidf_train[:,1])
    #insert into dict
    roc_auc_score_train_tfidf_dict[i]=roc_auc_train



print(roc_auc_score_cv_tfidf_dict)
print(roc_auc_score_train_tfidf_dict)
```

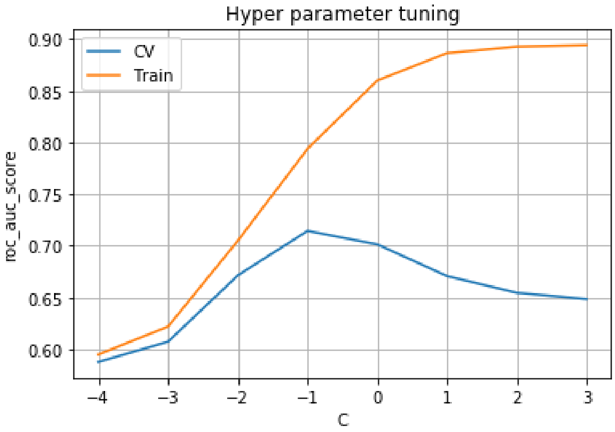
```
100%|████████████████████████████████████████████████████████████████████| 8/8 [05
:19<00:00, 39.92s/it]
```

```
{0.0001: 0.58771578152725479, 0.001: 0.60728353477102326, 0.01: 0.67138731359517223, 0.1:
0.71448349942671963, 1: 0.70146013492581427, 10: 0.67076763497993075, 100: 0.65464457624037153, 10
00: 0.64850134475138865}
{0.0001: 0.5949227809080142, 0.001: 0.62190124861707419, 0.01: 0.70489413631834819, 0.1:
0.79387372777815757, 1: 0.85989935078365554, 10: 0.88643172144062055, 100: 0.89267312012597833, 10
00: 0.89409949102570652}
```

Plot ROC_AUC_score VS different ALPHA values (Train and CV set)

In [70]:

```
#https://stackoverflow.com/questions/37266341/plotting-a-python-dict-in-order-of-key-
values/37266356

lists1 = sorted(roc_auc_score_cv_tfidf_dict.items())
x1, y1 = zip(*lists1) # unpack a list of pairs into two tuples
lists2 = sorted(roc_auc_score_train_tfidf_dict.items())
x2, y2 = zip(*lists2) # unpack a list of pairs into two tuples

x1=[math.log10(i) for i in list(x1)]
x2=[math.log10(i) for i in list(x2)]

plt.xlabel('C')
plt.ylabel('roc_auc_score')
plt.title('Hyper parameter tuning')

plt.plot(x1, y1,label="CV")
plt.plot(x2, y2,label='Train')
plt.legend()
plt.grid()
plt.show()
```



Find the best ALPHA value from the result

In [71]:

```
print(find_highest_hyperparam(roc_auc_score_cv_tfidf_dict))
```

```
Dictionary Contents-->
 {0.0001: 0.58771578152725479, 0.001: 0.60728353477102326, 0.01: 0.67138731359517223, 0.1:
0.71448349942671963, 1: 0.70146013492581427, 10: 0.67076763497993075, 100: 0.65464457624037153, 10
00: 0.64850134475138865}
```

```
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.58771578152725479, 0.60728353477102326, 0.67138731359517223, 0.71448349942671963,
0.70146013492581427, 0.67076763497993075, 0.65464457624037153, 0.64850134475138865]
==================================================
Maximum k is 0.1
==================================================
0.1
```

Train the model on the optimal ALPHA value and run the Test Dataset

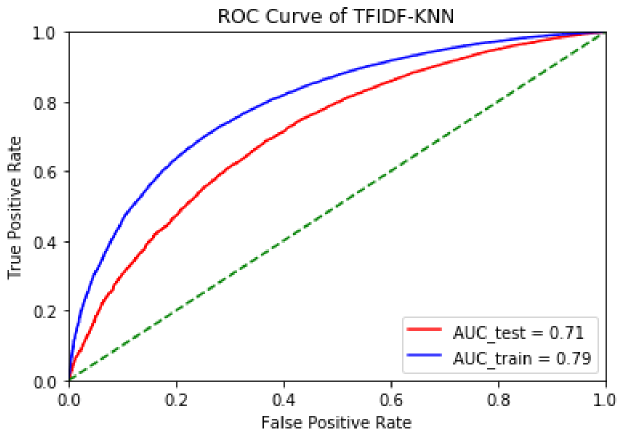
```
   -- Plot the ROC curve for TFIDF using the test and train Dataset
```

In [72]:

```
# train model on the best k

lr = LogisticRegression(penalty='l2', C=
find_highest_hyperparam(roc_auc_score_cv_tfidf_dict),class_weight='balanced')

# fitting the model on crossvalidation train

lr.fit(X_train_tfidf, y_train)

pred_tfidf_test = lr.predict(X_test_tfidf) # **we will use it in confusion matrix
pred_tfidf_train = lr.predict(X_train_tfidf) # **we will use it in confusion matrix

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-al
gorithm-using-python-and-scip_scores = knn.predict_proba(X_test)
pred_tfidf_test_scores=lr.predict_proba(X_test_tfidf)
pred_tfidf_train_scores=lr.predict_proba(X_train_tfidf)

fpr_test, tpr_test, threshold_test = roc_curve(y_test, pred_tfidf_test_scores[:,1])
fpr_train, tpr_train, threshold_train = roc_curve(y_train, pred_tfidf_train_scores[:,1])

roc_auc_test = auc(fpr_test, tpr_test)
roc_auc_train = auc(fpr_train, tpr_train)

#save ALPHA & roc_auc_test score (AUC of fpr_test and tpr_test) for final summary
summary.append(['TFIDF',find_highest_hyperparam(roc_auc_score_cv_tfidf_dict),roc_auc_test])

plt.title('Receiver Operating Characteristic')
plt.plot(fpr_test, tpr_test, 'r', label = 'AUC_test = %0.2f' % roc_auc_test)
plt.plot(fpr_train, tpr_train, 'b', label = 'AUC_train = %0.2f' % roc_auc_train)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of TFIDF-KNN')
plt.show()
```

```
Dictionary Contents-->
 {0.0001: 0.58771578152725479, 0.001: 0.60728353477102326, 0.01: 0.67138731359517223, 0.1:
0.71448349942671963, 1: 0.70146013492581427, 10: 0.67076763497993075, 100: 0.65464457624037153, 10
00: 0.64850134475138865}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.58771578152725479, 0.60728353477102326, 0.67138731359517223, 0.71448349942671963,
0.70146013492581427, 0.67076763497993075, 0.65464457624037153, 0.64850134475138865]
==================================================
Maximum k is 0.1
==================================================
Dictionary Contents-->
 {0.0001: 0.58771578152725479, 0.001: 0.60728353477102326, 0.01: 0.67138731359517223, 0.1:
0.71448349942671963, 1: 0.70146013492581427, 10: 0.67076763497993075, 100: 0.65464457624037153, 10
00: 0.64850134475138865}
```

```
00: 0.64850134475138865}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.5877157815272547 9, 0.60728353477102326, 0.67138731359517223, 0.71448349942671963,
0.70146013492581427, 0.67076763497993075, 0.65464457624037153, 0.64850134475138865]
==================================================
Maximum k is 0.1
==================================================
```



Get the confusion matrix for the TFIDF

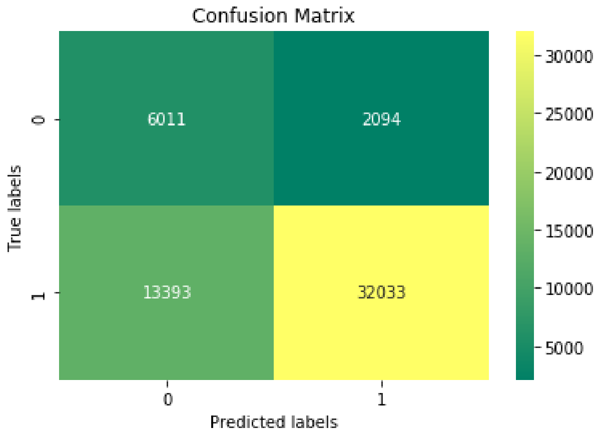
In [73]:

```python
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
# Make confusion matrix for y_train vs predicted(X_train_bow)
cm = confusion_matrix(y_train, pred_tfidf_train)
print(cm)

sns.heatmap(cm, annot=True, ax = ax,cmap="summer", fmt='g')
print("Training CM for tfidf")
# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
```

```
[[ 6011  2094]
 [13393 32033]]
Training CM for tfidf
```

Out[73]:

```
Text(0.5,1,'Confusion Matrix')
```



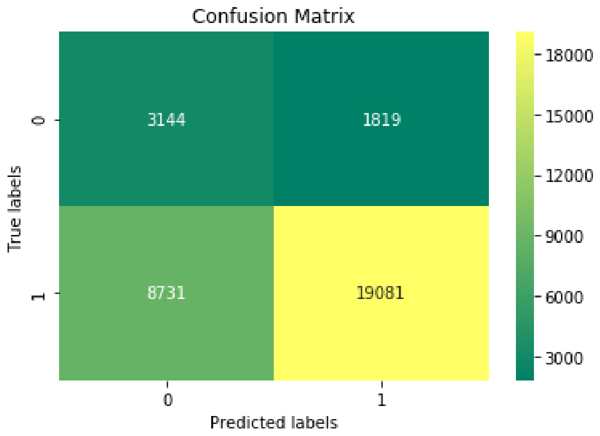
In [74]:

```
print("="*50)
print("Testing CM for tfidf")
ax= plt.subplot()
# Make confusion matrix for y_test vs predicted(X_test_bow)
cm = confusion_matrix(y_test, pred_tfidf_test)
print(cm)

sns.heatmap(cm, annot=True, ax = ax,cmap="summer",fmt='g')
# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
```

```
==================================================
Testing CM for tfidf
[[ 3144  1819]
 [ 8731 19081]]
```

Out[74]:

```
Text(0.5,1,'Confusion Matrix')
```



Applying Logistic Regression on AVG W2V, SET 3

Hyperparameter tuning for finiding optimal ALPHA using ROC_AUC_Score

In [75]:

```
roc_auc_score_cv_w2v_dict={}
roc_auc_score_train_w2v_dict={}

cValues = [.0001,.001, .01, .1, 1, 10, 100, 1000]

for i in tqdm(cValues):
    lr = LogisticRegression(penalty='l2',C= i,class_weight='balanced')

     # fitting the model on crossvalidation train
    lr.fit(X_train_w2v, y_train)

    # predict the response on the crossvalidation train
    pred_w2v_cv = lr.predict_proba(X_cv_w2v)

    #evaluate CV roc_auc
    roc_auc_cv =roc_auc_score(y_cv,pred_w2v_cv[:,1])

    #insert into dict
    roc_auc_score_cv_w2v_dict[i]=roc_auc_cv

    # predict the response on the train
    pred_w2v_train = lr.predict_proba(X_train_w2v)

    #evaluate train roc auc
    #evaluate train roc_auc
    roc_auc_train =roc_auc_score(y_train,pred_w2v_train[:,1])

    #insert into dict
    roc_auc_score_train_w2v_dict[i]=roc_auc_train



print(roc_auc_score_cv_w2v_dict)
print(roc_auc_score_train_w2v_dict)
```

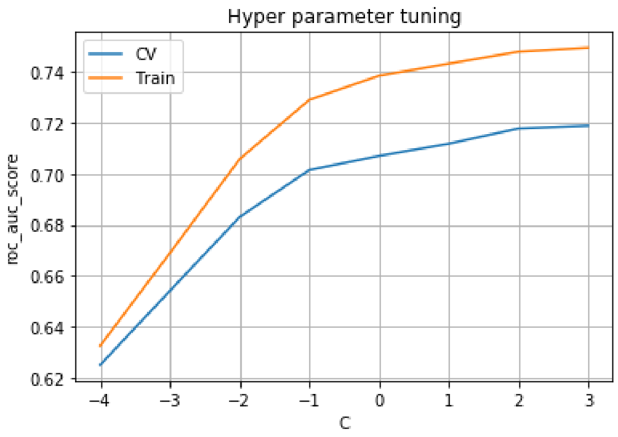
```
100%|████████████████████████████████████████████████████████████████████████████████| 8/8 [19:
07<00:00, 143.39s/it]
```

```
{0.0001: 0.6252287236578l845, 0.001: 0.65418415695141907, 0.01: 0.68302523580907881, 0.1:
0.70144981434186204, 1: 0.70694910349315998, 10: 0.71168424063982183, 100: 0.71766118948249102, 10
00: 0.71873645238532058}
{0.0001: 0.63271295360531454, 0.001: 0.66895104437739894, 0.01: 0.70568386903792368, 0.1:
0.72888044043293976, 1: 0.73827848034154586, 10: 0.74302348759660175, 100: 0.74770649218788976, 10
00: 0.74925494543083859}
```

Plot ROC_AUC_score VS different ALPHA values (Train and CV set)

In [76]:

```
#https://stackoverflow.com/questions/37266341/plotting-a-python-dict-in-order-of-key-
values/37266356

lists1 = sorted(roc_auc_score_cv_w2v_dict.items())
x1, y1 = zip(*lists1) # unpack a list of pairs into two tuples
lists2 = sorted(roc_auc_score_train_w2v_dict.items())
x2, y2 = zip(*lists2) # unpack a list of pairs into two tuples

x1=[math.log10(i) for i in list(x1)]
x2=[math.log10(i) for i in list(x2)]

plt.xlabel('C')
plt.ylabel('roc_auc_score')
plt.title('Hyper parameter tuning')
plt.plot(x1, y1,label="CV")
plt.plot(x2, y2,label='Train')
plt.legend()
plt.grid()
plt.show()
```



Find the best ALPHA value from the result

In [77]:

```
print(find_highest_hyperparam(roc_auc_score_cv_w2v_dict))
```

```
Dictionary Contents-->
 {0.0001: 0.62522872365781845, 0.001: 0.65418415695141907, 0.01: 0.68302523580907881, 0.1:
0.70144981434186204, 1: 0.70694910349315998, 10: 0.71168424063982183, 100: 0.71766118948249102, 10
00: 0.71873645238532058}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.62522872365781845, 0.65418415695141907, 0.68302523580907881, 0.70144981434186204,
0.70694910349315998, 0.71168424063982183, 0.71766118948249102, 0.71873645238532058]
==================================================
Maximum k is 1000
==================================================
1000
```

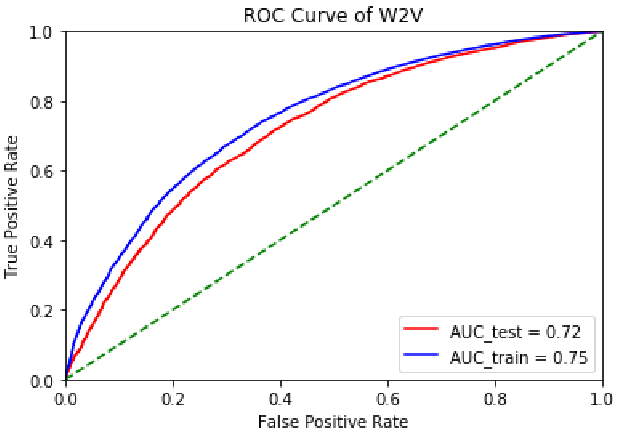
Train the model on the optimal ALPHA value and run the Test Dataset

```
    -- Plot the ROC curve for w2v using the test and train Dataset
```

In [78]:

```
# train model on the best k
lr = LogisticRegression(penalty='l2', C=
find_highest_hyperparam(roc_auc_score_cv_w2v_dict),class_weight='balanced')

# fitting the model on crossvalidation train

lr.fit(X_train_w2v, y_train)

pred_w2v_test = lr.predict(X_test_w2v) # **we will use it in confusion matrix
pred_w2v_train = lr.predict(X_train_w2v) # **we will use it in confusion matrix

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-al
gorithm-using-python-and-scip_scores = knn.predict_proba(X_test)
pred_w2v_test_scores=lr.predict_proba(X_test_w2v)
pred_w2v_train_scores=lr.predict_proba(X_train_w2v)

fpr_test, tpr_test, threshold_test = roc_curve(y_test, pred_w2v_test_scores[:,1])
fpr_train, tpr_train, threshold_train = roc_curve(y_train, pred_w2v_train_scores[:,1])

roc_auc_test = auc(fpr_test, tpr_test)
roc_auc_train = auc(fpr_train, tpr_train)

#save ALPHA & roc_auc_test score (AUC of fpr_test and tpr_test) for final summary
summary.append(['W2V',find_highest_hyperparam(roc_auc_score_cv_w2v_dict),roc_auc_test])

plt.title('Receiver Operating Characteristic')
plt.plot(fpr_test, tpr_test, 'r', label = 'AUC_test = %0.2f' % roc_auc_test)
plt.plot(fpr_train, tpr_train, 'b', label = 'AUC_train = %0.2f' % roc_auc_train)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of W2V')
plt.show()
```

```
Dictionary Contents-->
 {0.0001: 0.62522872365781845, 0.001: 0.65418415695141907, 0.01: 0.68302523580907881, 0.1:
0.70144981434186204, 1: 0.70694910349315998, 10: 0.71168424063982183, 100: 0.71766118948249102, 10
00: 0.71873645238532058}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.62522872365781845, 0.65418415695141907, 0.68302523580907881, 0.70144981434186204,
0.70694910349315998, 0.71168424063982183, 0.71766118948249102, 0.71873645238532058]
==================================================
Maximum k is 1000
==================================================
Dictionary Contents-->
 {0.0001: 0.62522872365781845, 0.001: 0.65418415695141907, 0.01: 0.68302523580907881, 0.1:
0.70144981434186204, 1: 0.70694910349315998, 10: 0.71168424063982183, 100: 0.71766118948249102, 10
00: 0.71873645238532058}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.62522872365781845, 0.65418415695141907, 0.68302523580907881, 0.70144981434186204,
0.70694910349315998, 0.71168424063982183, 0.71766118948249102, 0.71873645238532058]
==================================================
Maximum k is 1000
==================================================
```



Get the confusion matrix for the W2V

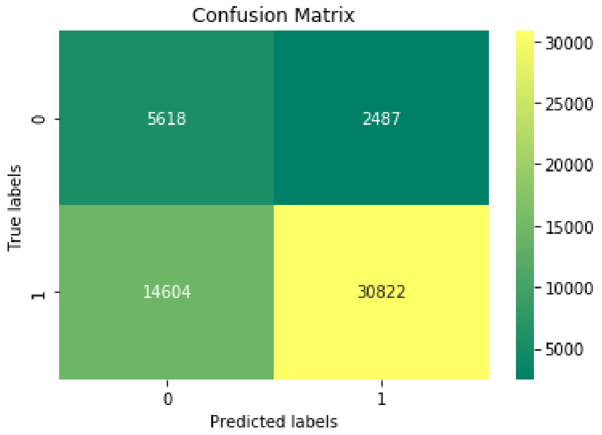
In [79]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
# Make confusion matrix for y_train vs predicted(X_train_bow)
cm = confusion_matrix(y_train, pred_w2v_train)
print(cm)

sns.heatmap(cm, annot=True, ax = ax,cmap="summer", fmt='g')
print("Training CM for BOW")
# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
```

```
[[ 5618  2487]
 [14604 30822]]
Training CM for BOW
```

Out[79]:

```
Text(0.5,1,'Confusion Matrix')
```



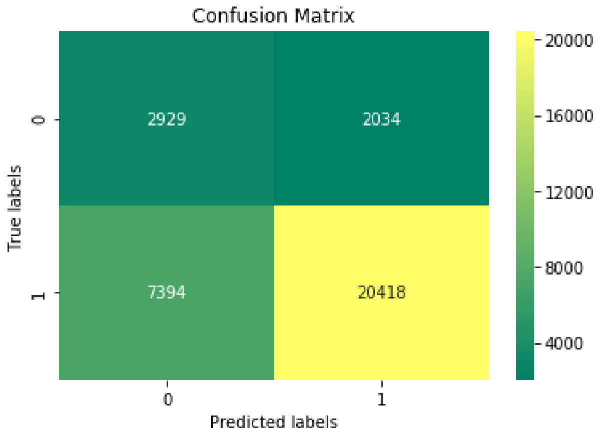
In [80]:

```
print("="*50)
print("Testing CM for BOW")
ax= plt.subplot()
# Make confusion matrix for y_test vs predicted(X_test_bow)
cm = confusion_matrix(y_test, pred_w2v_test)
print(cm)

sns.heatmap(cm, annot=True, ax = ax,cmap="summer",fmt='g')
# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
```

```
==================================================
Testing CM for BOW
[[ 2929  2034]
 [ 7394 20418]]
```

Out[80]:

```
Text(0.5,1,'Confusion Matrix')
```



Applying Logistic Regression on TFIDF W2V, SET 4

Hyperparameter tuning for finiding optimal ALPHA using ROC_AUC_Score

In [81]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

roc_auc_score_cv_tfidf_w2v_dict={}
roc_auc_score_train_tfidf_w2v_dict={}

cValues = [.0001,.001, .01, .1, 1, 10, 100, 1000]

for i in tqdm(cValues):
    lr = LogisticRegression(penalty='l2',C= i,class_weight='balanced')

     # fitting the model on crossvalidation train
    lr.fit(X_train_tfidf_w2v, y_train)

    # predict the response on the crossvalidation train
    pred_tfidf_w2v_cv = lr.predict_proba(X_cv_tfidf_w2v)

    #evaluate CV roc_auc
    roc_auc_cv =roc_auc_score(y_cv,pred_tfidf_w2v_cv[:,1])

    #insert into dict
    roc_auc_score_cv_tfidf_w2v_dict[i]=roc_auc_cv

    # predict the response on the train
    # predict the response on the train
    pred_tfidf_w2v_train = lr.predict_proba(X_train_tfidf_w2v)

    #evaluate train roc_auc
    roc_auc_train =roc_auc_score(y_train,pred_tfidf_w2v_train[:,1])

    #insert into dict
    roc_auc_score_train_tfidf_w2v_dict[i]=roc_auc_train



print(roc_auc_score_cv_tfidf_w2v_dict)
print(roc_auc_score_train_tfidf_w2v_dict)
```

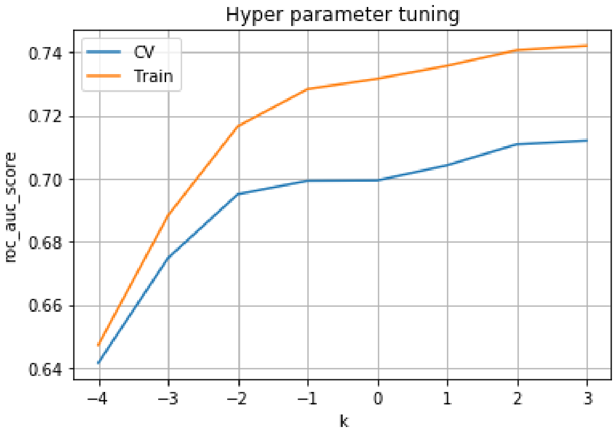
```
100%|██████████████████████████████████████████████████████████████████| 8/8 [19:
12<00:00, 144.11s/it]
```

```
{0.0001: 0.64157314265862264, 0.001: 0.67477537796107601, 0.01: 0.69497410627587319, 0.1:
0.69922555106891093, 1: 0.6993460712405366, 10: 0.7041526983920825, 100: 0.71077425493959712, 1000
: 0.71194138286835118}
{0.0001: 0.64722754415374339, 0.001: 0.68824002472936097, 0.01: 0.71648901197799231, 0.1:
0.7282782502896088, 1: 0.73153410718241973, 10: 0.7356641424238235, 100: 0.74061008524334149,
1000: 0.74200810570481812}
```

Plot ROC_AUC_score VS different ALPHA values (Train and CV set)

In [82]:

```
#https://stackoverflow.com/questions/37266341/plotting-a-python-dict-in-order-of-key-
values/37266356
import math

lists1 = sorted(roc_auc_score_cv_tfidf_w2v_dict.items())
x1, y1 = zip(*lists1) # unpack a list of pairs into two tuples
lists2 = sorted(roc_auc_score_train_tfidf_w2v_dict.items())
x2, y2 = zip(*lists2) # unpack a list of pairs into two tuples

x1=[math.log10(i) for i in list(x1)]
x2=[math.log10(i) for i in list(x2)]

plt.xlabel('k')
plt.ylabel('roc_auc_score')
plt.title('Hyper parameter tuning')
plt.plot(x1, y1,label="CV")
plt.plot(x2, y2,label='Train')
plt.legend()
plt.grid()
plt.show()
```



Find the best ALPHA value

In [83]:

```
print(find_highest_hyperparam(roc_auc_score_cv_tfidf_w2v_dict))
```

```
Dictionary Contents-->
 {0.0001: 0.64157314265862264, 0.001: 0.67477537796107601, 0.01: 0.69497410627587319, 0.1:
0.69922555106891093, 1: 0.6993460712405366, 10: 0.7041526983920825, 100: 0.71077425493959712, 1000
: 0.71194138286835118}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.64157314265862264, 0.67477537796107601, 0.69497410627587319, 0.69922555106891093,
0.6993460712405366, 0.7041526983920825, 0.71077425493959712, 0.71194138286835118]
==================================================
Maximum k is 1000
==================================================
1000
```

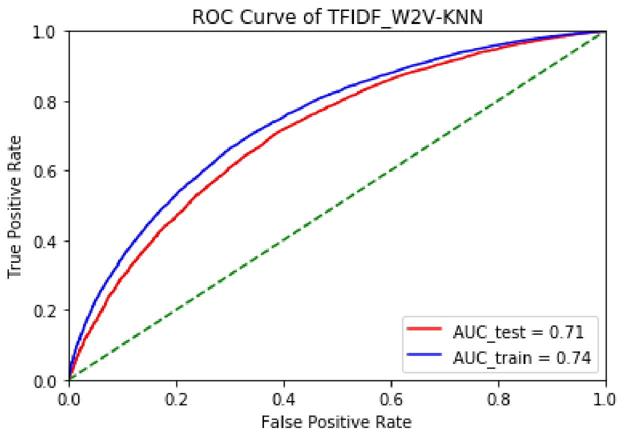
Train the model on the optimal ALPHA value and run the Test Dataset

In [84]:

```
# train model on the best k

lr = LogisticRegression(penalty='l2', C= find_highest_hyperparam(roc_auc_score_cv_tfidf_w2v_dict),
class_weight='balanced')
# fitting the model on crossvalidation train

lr.fit(X_train_tfidf_w2v, y_train)

pred_tfidf_w2v_test = lr.predict(X_test_tfidf_w2v) # **we will use it in confusion matrix
pred_tfidf_w2v_train = lr.predict(X_train_tfidf_w2v) # **we will use it in confusion matrix

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-al
gorithm-using-python-and-scip_scores = knn.predict_proba(X_test)
pred_tfidf_w2v_test_scores=lr.predict_proba(X_test_tfidf_w2v)
pred_tfidf_w2v_train_scores=lr.predict_proba(X_train_tfidf_w2v)

fpr_test, tpr_test, threshold_test = roc_curve(y_test, pred_tfidf_w2v_test_scores[:,1])
fpr_train, tpr_train, threshold_train = roc_curve(y_train, pred_tfidf_w2v_train_scores[:,1])

roc_auc_test = auc(fpr_test, tpr_test)
roc_auc_train = auc(fpr_train, tpr_train)

#save ALPHA & roc_auc_test score (AUC of fpr_test and tpr_test) for final summary
summary.append(['TFIDF_W2V',find_highest_hyperparam(roc_auc_score_cv_tfidf_w2v_dict),roc_auc_test]
)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr_test, tpr_test, 'r', label = 'AUC_test = %0.2f' % roc_auc_test)
plt.plot(fpr_train, tpr_train, 'b', label = 'AUC_train = %0.2f' % roc_auc_train)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of TFIDF_W2V-KNN')
plt.show()
```

```
Dictionary Contents-->
 {0.0001: 0.64157314265862264, 0.001: 0.67477537796107601, 0.01: 0.69497410627587319, 0.1:
0.69922555106891093, 1: 0.6993460712405366, 10: 0.7041526983920825, 100: 0.71077425493959712, 1000
: 0.71194138286835118}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.64157314265862264, 0.67477537796107601, 0.69497410627587319, 0.69922555106891093,
0.6993460712405366, 0.7041526983920825, 0.71077425493959712, 0.71194138286835118]
==================================================
Maximum k is 1000
==================================================
Dictionary Contents-->
```

```
 {0.0001: 0.64157314265862264, 0.001: 0.67477537796107601, 0.01: 0.69497410627587319, 0.1:
0.69922555106891093, 1: 0.6993460712405366, 10: 0.7041526983920825, 100: 0.71077425493959712, 1000
: 0.71194138286835118}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.64157314265862264, 0.67477537796107601, 0.69497410627587319, 0.69922555106891093,
0.6993460712405366, 0.7041526983920825, 0.71077425493959712, 0.71194138286835118]
==================================================
Maximum k is 1000
==================================================
```



Get the confusion matrix for the TFIDF W2V

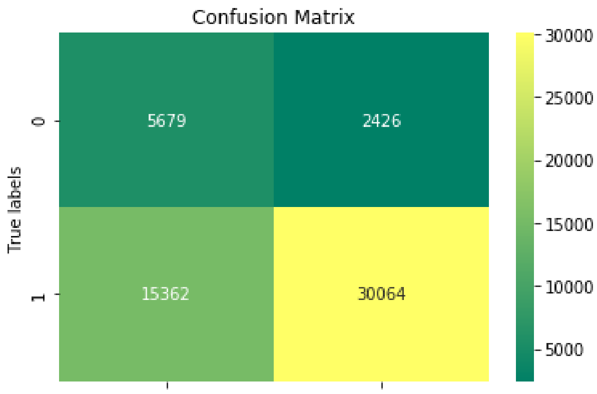
In [85]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
# Make confusion matrix for y_train vs predicted(X_train_bow)
cm = confusion_matrix(y_train, pred_tfidf_w2v_train)
print(cm)

sns.heatmap(cm, annot=True, ax = ax,cmap="summer", fmt='g')
print("Training CM for BOW")
# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
```

```
[[ 5679  2426]
 [15362 30064]]
Training CM for BOW
```

Out[85]:

```
Text(0.5,1,'Confusion Matrix')
```



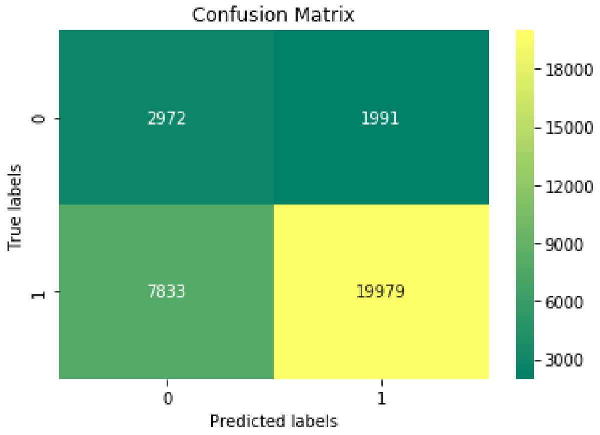
In [86]:

```
print("="*50)
print("Testing CM for tfidf")
ax= plt.subplot()
# Make confusion matrix for y_test vs predicted(X_test_bow)
cm = confusion_matrix(y_test, pred_tfidf_w2v_test)
print(cm)

sns.heatmap(cm, annot=True, ax = ax,cmap="summer",fmt='g')
# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
```

```
==================================================
Testing CM for tfidf
[[ 2972  1991]
 [ 7833 19979]]
```

Out[86]:

```
Text(0.5,1,'Confusion Matrix')
```



Applying Logistic Regression on Non Text Data

In [87]:

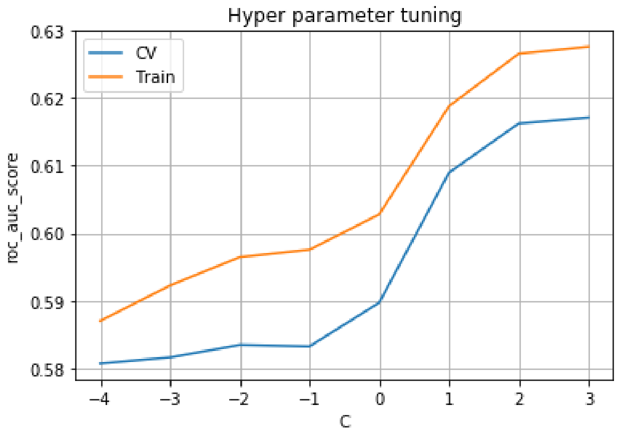
```
roc_auc_score_cv_no_text_dict={}
roc_auc_score_train_no_text_dict={}

cValues = [.0001,.001, .01, .1, 1, 10, 100, 1000]

for i in tqdm(cValues):
    lr = LogisticRegression(penalty='l2',C= i,class_weight='balanced')

     # fitting the model on crossvalidation train
    lr.fit(X_train_no_text, y_train)

    # predict the response on the crossvalidation train
    pred_no_text_cv = lr.predict_proba(X_cv_no_text)

    #evaluate CV roc_auc
    roc_auc_cv =roc_auc_score(y_cv,pred_no_text_cv[:,1])

    #insert into dict
    roc_auc_score_cv_no_text_dict[i]=roc_auc_cv

    # predict the response on the train
    pred_no_text_train = lr.predict_proba(X_train_no_text)

    #evaluate train roc_auc
    roc_auc_train =roc_auc_score(y_train,pred_no_text_train[:,1])

    #insert into dict
    roc_auc_score_train_no_text_dict[i]=roc_auc_train



print(roc_auc_score_cv_no_text_dict)
print(roc_auc_score_train_no_text_dict)
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 8/8 [00
:22<00:00,  2.76s/it]
```

```
{0.0001: 0.58072467266597183, 0.001: 0.58162689574932702, 0.01: 0.58346224452414652, 0.1:
0.58323950917077028, 1: 0.58972518295822596, 10: 0.60896465883106643, 100: 0.61624303626730093, 10
00: 0.61712148356412999}
{0.0001: 0.58702980351364542, 0.001: 0.59226767056225804, 0.01: 0.59646306282566308, 0.1:
0.59755855819959569, 1: 0.60279398349270064, 10: 0.6188016762991071, 100: 0.62657291765039669, 100
0: 0.62761228795668877}
```

In [88]:

```
#https://stackoverflow.com/questions/37266341/plotting-a-python-dict-in-order-of-key-
values/37266356

lists1 = sorted(roc_auc_score_cv_no_text_dict.items())
x1, y1 = zip(*lists1) # unpack a list of pairs into two tuples
lists2 = sorted(roc_auc_score_train_no_text_dict.items())
x2, y2 = zip(*lists2) # unpack a list of pairs into two tuples

x1=[math.log10(i) for i in list(x1)]
x2=[math.log10(i) for i in list(x2)]

plt.xlabel('C')
plt.ylabel('roc_auc_score')
plt.title('Hyper parameter tuning')
plt.plot(x1, y1,label="CV")
plt.plot(x2, y2,label='Train')
plt.legend()
plt.grid()
plt.show()
```



In [89]:

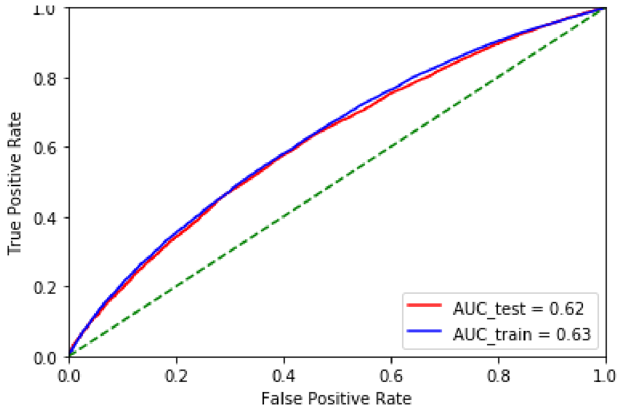
```
print(find_highest_hyperparam(roc_auc_score_cv_no_text_dict))
```

```
Dictionary Contents-->
 {0.0001: 0.58072467266597183, 0.001: 0.58162689574932702, 0.01: 0.58346224452414652, 0.1:
0.58323950917077028, 1: 0.58972518295822596, 10: 0.60896465883106643, 100: 0.61624303626730093, 10
00: 0.61712148356412999}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.58072467266597183, 0.58162689574932702, 0.58346224452414652, 0.58323950917077028
 [0.58072467266597183, 0.58162689574932702, 0.58346224452414652, 0.58323950917077028,
0.58972518295822596, 0.60896465883106643, 0.61624303626730093, 0.61712148356412999]
==================================================
Maximum k is 1000
==================================================
1000
```

In [90]:

```
# train model on the best k
lr = LogisticRegression(penalty='l2', C=
find_highest_hyperparam(roc_auc_score_cv_no_text_dict),class_weight='balanced')
# fitting the model on crossvalidation train

lr.fit(X_train_no_text, y_train)

pred_no_text_test = lr.predict(X_test_no_text) # **we will use it in confusion matrix
pred_no_text_train = lr.predict(X_train_no_text) # **we will use it in confusion matrix

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-al
gorithm-using-python-and-scip_scores = knn.predict_proba(X_test)
pred_no_text_test_scores=lr.predict_proba(X_test_no_text)
pred_no_text_train_scores=lr.predict_proba(X_train_no_text)

fpr_test, tpr_test, threshold_test = roc_curve(y_test, pred_no_text_test_scores[:,1])
fpr_train, tpr_train, threshold_train = roc_curve(y_train, pred_no_text_train_scores[:,1])
roc_auc_test = auc(fpr_test, tpr_test)
roc_auc_train = auc(fpr_train, tpr_train)

#save ALPHA & roc_auc_test score (AUC of fpr_test and tpr_test) for final summary
summary.append(['No_Text',find_highest_hyperparam(roc_auc_score_cv_no_text_dict),roc_auc_test])

plt.title('Receiver Operating Characteristic')
plt.plot(fpr_test, tpr_test, 'r', label = 'AUC_test = %0.2f' % roc_auc_test)
plt.plot(fpr_train, tpr_train, 'b', label = 'AUC_train = %0.2f' % roc_auc_train)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'g--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

plt.title('ROC Curve of no_text')
plt.show()
```

```
Dictionary Contents-->
 {0.0001: 0.58072467266597183, 0.001: 0.58162689574932702, 0.01: 0.58346224452414652, 0.1:
0.58323950917077028, 1: 0.58972518295822596, 10: 0.60896465883106643, 100: 0.61624303626730093, 10
00: 0.61712148356412999}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.58072467266597183, 0.58162689574932702, 0.58346224452414652, 0.58323950917077028,
0.58972518295822596, 0.60896465883106643, 0.61624303626730093, 0.61712148356412999]
==================================================
Maximum k is 1000
==================================================
Dictionary Contents-->
 {0.0001: 0.58072467266597183, 0.001: 0.58162689574932702, 0.01: 0.58346224452414652, 0.1:
0.58323950917077028, 1: 0.58972518295822596, 10: 0.60896465883106643, 100: 0.61624303626730093, 10
00: 0.61712148356412999}
==================================================
Keys in the list are-->
 [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
==================================================
Values in the list are-->
 [0.58072467266597183, 0.58162689574932702, 0.58346224452414652, 0.58323950917077028,
0.58972518295822596, 0.60896465883106643, 0.61624303626730093, 0.61712148356412999]
==================================================
Maximum k is 1000
==================================================
```



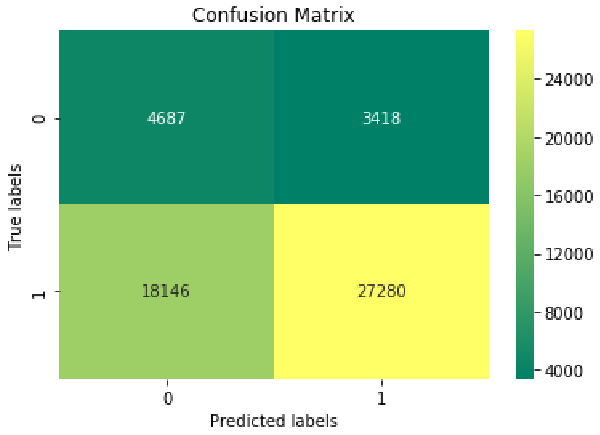
In [91]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
# Make confusion matrix for y_train vs predicted(X_train_bow)
cm = confusion_matrix(y_train, pred_no_text_train)
print(cm)

sns.heatmap(cm, annot=True, ax = ax,cmap="summer", fmt='g')
print("Training CM for NO_TEXT")
# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
```

```
[[ 4687  3418]
 [18146 27280]]
Training CM for NO_TEXT
```

Out[91]:

```
Text(0.5,1,'Confusion Matrix')
```



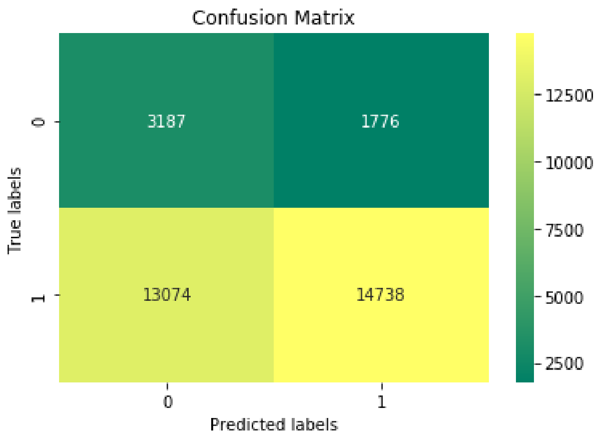
In [92]:

```
print("="*50)
print("Testing CM for NO_TEXT")
ax= plt.subplot()
# Make confusion matrix for y_test vs predicted(X_test_bow)
cm = confusion_matrix(y_test, pred_no_text_test)
print(cm)

sns.heatmap(cm, annot=True, ax = ax,cmap="summer",fmt='g')
# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
```

```
==================================================
Testing CM for NO_TEXT
[[ 3187  1776]
 [13074 14738]]
```

Out[92]:

```
Text(0.5,1,'Confusion Matrix')
```



Conclusions

In [93]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Hyper parameter", "AUC"]

for each in summary:
    x.add_row(each)

print(x)
```

```
+------------+-----------------+----------------+
| Vectorizer | Hyper parameter |      AUC       |
+------------+-----------------+----------------+
|    BOW     |      0.001      | 0.713624228611 |
|   TFIDF    |       0.1       | 0.713494033179 |
|    W2V     |       1000      | 0.717755656927 |
| TFIDF_W2V  |       1000      | 0.710858378029 |
|  No_Text   |       1000      | 0.619591557418 |
+------------+-----------------+----------------+
```