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
project_subject_categories	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • Applied Learning • Care & Hunger • Health & Sports • 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.

In [1]:
```
%config IPCompleter.greedy=True
%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
from sklearn.metrics import confusion_matrix
```

1.1 Reading Data

In [2]:
```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [3]:
```
print("Number of data points in train data", project_data.shape)
```

```
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (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']
```

In [4]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/
4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project
_data.columns)]
#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702
492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetim
e'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)
# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4
084039
project_data = project_data[cols]
project_data.head(2)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school
86221	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
18308	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

In [5]:

```
# 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 [6]:

```
# 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', 'it
self', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
hat', "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', 'becau
se', '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', 'a
ll', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'tha
n', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
d've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
"didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'm
a', 'mightn', "mightn't", 'mustn',\
```

```
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

1.2 preprocessing of `project_subject_categories`

In [7]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-fr
om-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-strin
g-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 & Scienc
e", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on
space "Math & Science"=> "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 ''(emp
ty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the tra
iling spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

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

Out[8]:
```
{'Warmth': 1388,
 'Care_Hunger': 1388,
 'History_Civics': 5914,
 'Music_Arts': 10293,
 'AppliedLearning': 12135,
 'SpecialNeeds': 13642,
 'Health_Sports': 14223,
 'Math_Science': 41421,
 'Literacy_Language': 52239}
```

1.3 preprocessing of `project_subject_subcategories`

In [9]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflo
w.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-fr
om-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-strin
g-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 & Scienc
e", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on
space "Math & Science"=> "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 ''(emp
ty) ex:"Math & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the tra
iling 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/2289859
5/4084039
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]))
```

In [10]:
```
sorted_sub_cat_dict
```

Out[10]:
```
{'Economics': 269,
 'CommunityService': 441,
 'FinancialLiteracy': 568,
 'ParentInvolvement': 677,
 'Extracurricular': 810,
 'Civics_Government': 815,
 'ForeignLanguages': 890,
 'NutritionEducation': 1355,
 'Warmth': 1388,
 'Care_Hunger': 1388,
 'SocialSciences': 1920,
 'PerformingArts': 1961,
 'CharacterEducation': 2065,
 'TeamSports': 2192,
 'Other': 2372,
 'College_CareerPrep': 2568,
 'Music': 3145,
 'History_Geography': 3171,
 'Health_LifeScience': 4235,
 'EarlyDevelopment': 4254,
 'ESL': 4367,
 'Gym_Fitness': 4509,
 'EnvironmentalScience': 5591,
 'VisualArts': 6278,
 'Health_Wellness': 10234,
 'AppliedSciences': 10816,
 'SpecialNeeds': 13642,
 'Literature_Writing': 22179,
 'Mathematics': 28074,
 'Literacy': 33700}
```

1.3 Text preprocessing

In [11]:
```
# 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 [12]:
```
project_data.head(2)
```

Out[12]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school
86221	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
18308	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

In [13]:
```
# printing some random reviews
print(project_data['essay'].values[0])
```

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classroo
m as well as the STEM journals, which my students really enjoyed.  I would
love to implement more of the Lakeshore STEM kits in my classroom for the n
ext school year as they provide excellent and engaging STEM lessons.My stud
ents come from a variety of backgrounds, including language and socioeconom
ic status.  Many of them don't have a lot of experience in science and engi
neering and these kits give me the materials to provide these exciting oppo
rtunities for my students.Each month I try to do several science or STEM/ST
EAM projects.  I would use the kits and robot to help guide my science inst
ruction in engaging and meaningful ways.  I can adapt the kits to my curren
t language arts pacing guide where we already teach some of the material in
the kits like tall tales (Paul Bunyan) or Johnny Appleseed.  The following
units will be taught in the next school year where I will implement these k
its: magnets, motion, sink vs. float, robots.  I often get to these units a
```

```
nd don't know If I am teaching the right way or using the right materials.
The kits will give me additional ideas, strategies, and lessons to prepare
my students in science.It is challenging to develop high quality science ac
tivities.  These kits give me the materials I need to provide my students w
ith science activities that will go along with the curriculum in my classro
om.  Although I have some things (like magnets) in my classroom, I don't kn
ow how to use them effectively.  The kits will provide me with the right am
ount of materials and show me how to use them in an appropriate way.
```

In [14]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
len_essay=[]
# tqdm is for printing the status bar
for sentance1 in tqdm(project_data['essay'].values):
    sent= sentance1.lower()
    sent = decontracted(sent)
    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())
    len_essay.append(len(sent.split()))
num_essay=np.array(len_essay)
```

```
100%|██████████| 109248/109248 [01:23<00:00, 1310.07it/s]
```

In [15]:
```
# after preprocesing
project_data['essay']=preprocessed_essays
project_data['num_essay']=num_essay
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)
print(project_data['essay'].values[0])
```

```
fortunate enough use fairy tale stem kits classroom well stem journals stud
ents really enjoyed would love implement lakeshore stem kits classroom next
school year provide excellent engaging stem lessons students come variety b
ackgrounds including language socioeconomic status many not lot experience
science engineering kits give materials provide exciting opportunities stud
```

```
ents month try several science stem steam projects would use kits robot hel
p guide science instruction engaging meaningful ways adapt kits current lan
guage arts pacing guide already teach material kits like tall tales paul bu
nyan johnny appleseed following units taught next school year implement kit
s magnets motion sink vs float robots often get units not know teaching rig
ht way using right materials kits give additional ideas strategies lessons
prepare students science challenging develop high quality science activitie
s kits give materials need provide students science activities go along cur
riculum classroom although things like magnets classroom not know use effec
tively kits provide right amount materials show use appropriate way
```

1.4 Preprocessing of `project_title`

In [16]:
```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_titles = []
len_project=[]
# tqdm is for printing the status bar
for sentence2 in tqdm(project_data['project_title'].values):
    sent = sentence2.lower()
    sent = decontracted(sent)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
    len_project.append(len(sent.split()))
num_project=np.array(len_project)
```

```
100%|
| 109248/109248 [00:03<00:00, 30898.00it/s]
```

In [17]:
```
# after preprocesing
project_data['project_title']=preprocessed_titles
project_data['num_project']=num_project
print(project_data['project_title'][0])
```

```
not 21st century learners across ocean
```

In [18]:
```
#Preprocessing the project_grade_category
project_grade_category_cleaned=[]
```

```
for grade in tqdm(project_data['project_grade_category'].values):
    grade = grade.replace(' ', '_')
    grade = grade.replace('-', '_')
    project_grade_category_cleaned.append(grade)
project_data['Project_grade_category']=project_grade_category_cleaned
```

```
100%|
| 109248/109248 [00:00<00:00, 614144.46it/s]
```

In [19]:
```
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')
```

Assignment 7: SVM

1. **[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) on these feature sets**
    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
    - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
    - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

2. **The hyper paramter tuning (best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**
    - Find the best hyper parameter which will give the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.

- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.

- Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

4. **[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3**

   - Consider these set of features Set 5 :
     - **school_state** : categorical data
     - **clean_categories** : categorical data
     - **clean_subcategories** : categorical data
     - **project_grade_category** :categorical data
     - **teacher_prefix** : categorical data
     - **quantity** : numerical data
     - **teacher_number_of_previously_posted_projects** : numerical data
     - **price** : numerical data
     - **sentiment score's of each of the essay** : numerical data
     - **number of words in the title** : numerical data
     - **number of words in the combine essays** : numerical data
     - **Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`) using elbow method** : numerical data

   - **Conclusion**
     - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

In [20]:
```
project_data.head(2)
```

Out[20]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_stat
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

2. Support Vector Machines

2.1 Splitting data into Train and cross validation(or test): Stratified Sampling

In [21]:
```
from sklearn.model_selection import train_test_split
# split the data set into train and test respectively 80% and 20%
y=project_data['project_is_approved']
project_data.drop(['project_is_approved'],axis=1, inplace=True)
x=project_data
X_temp,X_test,Y_temp,Y_test=train_test_split(x,y,test_size=0.33,random_state=1
)
# split the data set into train and cv respectively 60% and 20%
```

```
X_train,X_cv,Y_train,Y_cv=train_test_split(X_temp,Y_temp,test_size=0.33,random
_state=1)
print("Shape of Train data set X={} Y={}".format(X_train.shape,Y_train.shape))
print("Shape of Test data set X={} Y={}".format(X_test.shape,Y_test.shape))
print("Shape of CV data set X={} Y={}".format(X_cv.shape,Y_cv.shape))
```

```
Shape of Train data set X=(49041, 18) Y=(49041,)
Shape of Test data set X=(36052, 18) Y=(36052,)
Shape of CV data set X=(24155, 18) Y=(24155,)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

Vectorizing Categorical data

In [22]:

```
# we use count vectorizer to convert the values into one hot encoded features
# Project categories
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_categories = CountVectorizer(vocabulary=list(sorted_cat_dict.keys
()),lowercase=False, binary=True)

tr_categories_one_hot=vectorizer_categories.fit_transform(X_train['clean_categ
ories'].values)
print(vectorizer_categories.get_feature_names())

cv_categories_one_hot =vectorizer_categories.transform(X_cv['clean_categories'
].values)
te_categories_one_hot =vectorizer_categories.transform(X_test['clean_categorie
s'].values)

print(tr_categories_one_hot.toarray()[0:1])
print("\nShape of matrix after one hot encodig for 'Project categories'\nTrain
data-{},\nCV data\t-{}\nTest data-{}".format(tr_categories_one_hot.shape,cv_ca
tegories_one_hot.shape,te_categories_one_hot.shape))
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearnin
g', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
[[0 0 0 0 0 0 1 0 0]]

Shape of matrix after one hot encodig for 'Project categories'
Train data-(49041, 9),
CV data -(24155, 9)
Test data-(36052, 9)
```

In [23]:
```
# we use count vectorizer to convert the values into one hot encoded features
# Project subcategories
vectorizer_subcategories = CountVectorizer(vocabulary=list(sorted_sub_cat_dict
.keys()), lowercase=False, binary=True)

tr_sub_categories_one_hot=vectorizer_subcategories.fit_transform(X_train['clea
n_subcategories'].values)
print(vectorizer_subcategories.get_feature_names())

cv_sub_categories_one_hot = vectorizer_subcategories.transform(X_cv['clean_sub
categories'].values)
te_sub_categories_one_hot = vectorizer_subcategories.transform(X_test['clean_s
ubcategories'].values)

print(tr_sub_categories_one_hot.toarray()[0:2])
print("\nShape of matrix after one hot encodig for 'Project sub categories'\nT
rain data-{},\nCV data\t-{}\nTest data-{}".format(tr_sub_categories_one_hot.sh
ape,cv_sub_categories_one_hot.shape,te_sub_categories_one_hot.shape))
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement',
'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEduca
tion', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'Charac
terEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'Histo
ry_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitnes
s', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedScienc
es', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
[[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]]

Shape of matrix after one hot encodig for 'Project sub categories'
Train data-(49041, 30),
CV data -(24155, 30)
Test data-(36052, 30)
```

In [24]:
```
# you can do the similar thing with state, teacher_prefix and project_grade_ca
tegory also
# we use count vectorizer to convert the values into one hot encoded features
#teacher_prefix
vectorizer_teacher_prefix = CountVectorizer(lowercase=False, binary=True)
tr_teacher_prefix_one_hot=vectorizer_teacher_prefix.fit_transform(X_train['tea
cher_prefix'].values.astype('str'))
print(vectorizer_teacher_prefix.get_feature_names())

cv_teacher_prefix_one_hot = vectorizer_teacher_prefix.transform(X_cv['teacher_
```

```
prefix'].values.astype('str'))
te_teacher_prefix_one_hot = vectorizer_teacher_prefix.transform(X_test['teache
r_prefix'].values.astype('str'))

print(tr_teacher_prefix_one_hot.toarray()[0:1])
print("\nShape of matrix after one hot encodig for 'teacher_prefix'\nTrain dat
a-{},\nCV data\t-{}\nTest data-{}".format(tr_teacher_prefix_one_hot.shape,cv_t
eacher_prefix_one_hot.shape,te_teacher_prefix_one_hot.shape))
```

```
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher', 'nan']
[[0 0 0 1 0 0]]

Shape of matrix after one hot encodig for 'teacher_prefix'
Train data-(49041, 6),
CV data -(24155, 6)
Test data-(36052, 6)
```

In [25]:

```
# we use count vectorizer to convert the values into one hot encoded features
#school_state
vectorizer_school_state = CountVectorizer(lowercase=False, binary=True)
tr_school_state_one_hot=vectorizer_school_state.fit_transform(X_train['school_
state'].values.astype('str'))
print(vectorizer_school_state.get_feature_names())

cv_school_state_one_hot = vectorizer_school_state.transform(X_cv['school_stat
e'].values.astype('str'))
te_school_state_one_hot = vectorizer_school_state.transform(X_test['school_sta
te'].values.astype('str'))

print(tr_school_state_one_hot.toarray()[0:1])
print("\nShape of matrix after one hot encodig for 'teacher_prefix'\nTrain dat
a-{},\nCV data\t-{}\nTest data-{}".format(tr_school_state_one_hot.shape,cv_sch
ool_state_one_hot.shape,te_school_state_one_hot.shape))
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'I
A', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO',
'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY', 'OH', 'OK', 'O
R', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV',
'WY']
[[0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]

Shape of matrix after one hot encodig for 'teacher_prefix'
Train data-(49041, 51),
CV data -(24155, 51)
Test data-(36052, 51)
```

In [26]:
```
# we use count vectorizer to convert the values into one hot encoded features
#project_grade_category
vectorizer_grade_category = CountVectorizer(lowercase=False, binary=True)
tr_grade_category_one_hot=vectorizer_grade_category.fit_transform(X_train['Pro
ject_grade_category'])
print(vectorizer_grade_category.get_feature_names())

cv_grade_category_one_hot = vectorizer_grade_category.transform(X_cv['Project_
grade_category'])
te_grade_category_one_hot = vectorizer_grade_category.transform(X_test['Projec
t_grade_category'])

print(tr_grade_category_one_hot.toarray()[0:1])
print(cv_grade_category_one_hot.toarray()[0:1])
print(te_grade_category_one_hot.toarray()[0:1])
print("\nShape of matrix after one hot encodig for 'project_grade_category'\nT
rain data-{},\nCV data\t-{}\nTest data-{}".format(tr_grade_category_one_hot.sh
ape,cv_grade_category_one_hot.shape,te_grade_category_one_hot.shape))
```

```
['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
[[1 0 0 0]]
[[1 0 0 0]]
[[0 0 0 1]]

Shape of matrix after one hot encodig for 'project_grade_category'
Train data-(49041, 4),
CV data -(24155, 4)
Test data-(36052, 4)
```

standardizing Numerical features

In [27]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/s
klearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(X_train['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 32
9.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()
```

```
tr_price_standardized=price_scalar.fit_transform(X_train['price'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
cv_price_standardized = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
te_price_standardized = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
```

```
Mean : 298.5818657857711, Standard deviation : 363.8573751232583
```

In [28]:
```
print("\nShape of matrix after column standardization for 'price'\nTrain data-{},\nCV data\t-{}\nTest data-{}".format(tr_price_standardized.shape,cv_price_standardized.shape,te_price_standardized.shape))
```

```
Shape of matrix after column standardization for 'price'
Train data-(49041, 1),
CV data -(24155, 1)
Test data-(36052, 1)
```

In [29]:
```
#quantity
quantity_scalar = StandardScaler()
tr_quantity_standardized=quantity_scalar.fit_transform(X_train['quantity'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {quantity_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
cv_quantity_standardized = quantity_scalar.transform(X_cv['quantity'].values.reshape(-1, 1))
te_quantity_standardized = quantity_scalar.transform(X_test['quantity'].values.reshape(-1, 1))
print("\nShape of matrix after column standardization for 'quantity'\nTrain data-{},\nCV data\t-{}\nTest data-{}".format(tr_quantity_standardized.shape,cv_quantity_standardized.shape,te_quantity_standardized.shape))
```

```
C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
```

```
Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Mean : 16.96853653065802, Standard deviation : 26.262737421015874
```

```
C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Shape of matrix after column standardization for 'quantity'
Train data-(49041, 1),
CV data -(24155, 1)
Test data-(36052, 1)
```

In [30]:

```
#teacher_number_of_previously_posted_projects
teacher_number_of_previously_posted_projects_scalar = StandardScaler()
tr_teacher_number_of_previously_posted_projects_standardized=teacher_number_of
_previously_posted_projects_scalar.fit_transform(X_train['teacher_number_of_pr
eviously_posted_projects'].values.reshape(-1,1)) # finding the mean and standa
rd deviation of this data
print(f"Mean : {teacher_number_of_previously_posted_projects_scalar.mean_[0]},
Standard deviation : {np.sqrt(teacher_number_of_previously_posted_projects_sca
lar.var_[0])}")

# Now standardize the data with above maen and variance.
cv_teacher_number_of_previously_posted_projects_standardized = teacher_number_
of_previously_posted_projects_scalar.transform(X_cv['teacher_number_of_previou
sly_posted_projects'].values.reshape(-1, 1))
te_teacher_number_of_previously_posted_projects_standardized = teacher_number_
of_previously_posted_projects_scalar.transform(X_test['teacher_number_of_previ
ously_posted_projects'].values.reshape(-1, 1))
print("\nShape of matrix after column standardization for 'teacher_number_of_p
reviously_posted_projects'\nTrain data-{},\nCV data\t-{}\nTest data-{}".format
(tr_teacher_number_of_previously_posted_projects_standardized.shape,cv_teacher
_number_of_previously_posted_projects_standardized.shape,te_teacher_number_of_
previously_posted_projects_standardized.shape))
```

```
C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:
```

```
Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Mean : 11.03756040863767, Standard deviation : 27.38081956899988
```

```
C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Shape of matrix after column standardization for 'teacher_number_of_previou
sly_posted_projects'
Train data-(49041, 1),
CV data -(24155, 1)
Test data-(36052, 1)
```

In [31]:

```
#Number of words in essay
num_essay_scalar = StandardScaler()
tr_num_essay_standardized=num_essay_scalar.fit_transform(X_train['num_essay'].
values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {num_essay_scalar.mean_[0]}, Standard deviation : {np.sqrt(num_
essay_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
cv_num_essay_standardized = num_essay_scalar.transform(X_cv['num_essay'].value
s.reshape(-1, 1))
te_num_essay_standardized = num_essay_scalar.transform(X_test['num_essay'].val
ues.reshape(-1, 1))
print("\nShape of matrix after column standardization for 'num_essay'\nTrain d
ata-{},\nCV data\t-{}\nTest data-{}".format(tr_num_essay_standardized.shape,cv
_num_essay_standardized.shape,te_num_essay_standardized.shape))
```

```
C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.
```

```
C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.
```

```
Mean : 138.1287086315532, Standard deviation : 36.41300099141212
```

```
C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.

C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.
```

```
Shape of matrix after column standardization for 'num_essay'
Train data-(49041, 1),
CV data -(24155, 1)
Test data-(36052, 1)
```

In [32]:
```
#Number of words in essay
num_project_scalar = StandardScaler()
tr_num_project_standardized=num_project_scalar.fit_transform(X_train['num_proj
ect'].values.reshape(-1,1)) # finding the mean and standard deviation of this
 data
print(f"Mean : {num_project_scalar.mean_[0]}, Standard deviation : {np.sqrt(nu
m_project_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
cv_num_project_standardized = num_project_scalar.transform(X_cv['num_project']
.values.reshape(-1, 1))
te_num_project_standardized = num_project_scalar.transform(X_test['num_projec
t'].values.reshape(-1, 1))
print("\nShape of matrix after column standardization for 'num_project'\nTrain
data-{},\nCV data\t-{}\nTest data-{}".format(tr_num_project_standardized.shape
,cv_num_project_standardized.shape,te_num_project_standardized.shape))
```

```
C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.
```

```
C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.
```

```
Mean : 3.6965600212067455, Standard deviation : 1.523906739030445
```

```
C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.

C:\Users\nnagari\AppData\Local\Continuum\anaconda\lib\site-packages\sklearn
\utils\validation.py:475: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.
```

```
Shape of matrix after column standardization for 'num_project'
Train data-(49041, 1),
CV data -(24155, 1)
Test data-(36052, 1)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

Vectorizing Text data

Bag of Words on `preprocessed_essay`

In [33]:
```
#Bag of words of Project essays
# We are considering only the words which appeared in at least 10 documents(ro
ws or projects) and max feature is 8000.
#Fitting train data because we need all and transforming  train ,cv and test v
ector shape should be same.
vectorizer_essays = CountVectorizer(min_df=10,max_features=5000)#max_features=
8000
tr_text_bow=vectorizer_essays.fit_transform(X_train['essay']) # fitting train
 data

#transforming train,cv and test data
```

```
cv_text_bow = vectorizer_essays.transform(X_cv['essay'])
te_text_bow = vectorizer_essays.transform(X_test['essay'])
print("Shape of matrix after one hot encodig \nTrain data-{},\nCV data\t-{}\nTest data-{}".format(tr_text_bow.shape,cv_text_bow.shape,te_text_bow.shape))
```

```
Shape of matrix after one hot encodig
Train data-(49041, 5000),
CV data -(24155, 5000)
Test data-(36052, 5000)
```

In [34]:
```
print('Some feature names of bag of words of the essays')
print('='*50)
print(vectorizer_essays.get_feature_names()[1000:1020])
print(tr_text_bow.toarray()[0:1])
```

```
Some feature names of bag of words of the essays
==================================================
['consistently', 'consisting', 'consists', 'constant', 'constantly', 'const
raints', 'construct', 'constructing', 'construction', 'constructive', 'cons
umable', 'consumers', 'consuming', 'contact', 'contagious', 'contain', 'con
tained', 'containers', 'contains', 'contemporary']
[[0 0 0 ... 0 0 0]]
```

Bag of Words on `project_title`

In [35]:
```
#Bag of words project_title
# We are considering only the words which appeared in at least 5 documents(row
s or projects) and max number of feature is 5000.
#Fitting train data and transforming  train ,cv and test vector shape should b
e same.
vectorizer_title = CountVectorizer(min_df=10,max_features=5000)
tr_text_bow_title=vectorizer_title.fit_transform(X_train['project_title'])
cv_text_bow_title = vectorizer_title.transform(X_cv['project_title'])
te_text_bow_title = vectorizer_title.transform(X_test['project_title'])
print("Shape of matrix after one hot encodig \nTrain data-{},\nCV data\t-{}\nTest data-{}".format(tr_text_bow_title.shape,cv_text_bow_title.shape,te_text_bo
w_title.shape))
```

```
Shape of matrix after one hot encodig
Train data-(49041, 1978),
CV data -(24155, 1978)
Test data-(36052, 1978)
```

In [36]:
```
print('Some feature names of bag of words of the project title')
print('='*50)
print(vectorizer_title.get_feature_names()[1000:1020])
print(tr_text_bow_title.toarray()[0:2])
```

```
Some feature names of bag of words of the project title
==================================================
['la', 'lab', 'labs', 'lakeshore', 'laminate', 'laminating', 'land', 'langu
age', 'lap', 'laptop', 'laptops', 'large', 'last', 'lead', 'leader', 'leade
rs', 'leadership', 'leading', 'leads', 'league']
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

TFIDF vectorizer

TFIDF Vectorizer on `preprocessed_essay`

In [37]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer_essays = TfidfVectorizer(min_df=10,max_features=3000)
#Fitting train data and transforming  train ,cv and test vector shape should b
e same.
tr_text_tfidf=tfidf_vectorizer_essays.fit_transform(X_train['essay'])
cv_text_tfidf = tfidf_vectorizer_essays.transform(X_cv['essay'])
te_text_tfidf = tfidf_vectorizer_essays.transform(X_test['essay'])
print("Shape of matrix TFIDF Vectorizer on essays \nTrain data-{},\nCV data\t-
{}\nTest data-{}".format(tr_text_tfidf.shape,cv_text_tfidf.shape,te_text_tfidf
.shape))
```

```
Shape of matrix TFIDF Vectorizer on essays
Train data-(49041, 3000),
CV data -(24155, 3000)
Test data-(36052, 3000)
```

In [38]:
```
print('Sample of TFIDF Vectorizer on essays')
print('='*50)
print(tr_text_tfidf.toarray()[0:1])
print(tfidf_vectorizer_essays.get_feature_names()[300:310])
```

```
Sample of TFIDF Vectorizer on essays
==================================================
[[0. 0. 0. ... 0. 0. 0.]]
['becoming', 'began', 'begin', 'beginning', 'begins', 'begun', 'behavior',
'behavioral', 'behaviors', 'behind']
```

1.4.2.4 TFIDF Vectorizer on `project_title`

In [39]:
```
# Similarly you can vectorize for title also
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer_title = TfidfVectorizer(min_df=10)
#Fitting train data and transforming  train ,cv and test vector shape should b
e same.
tr_title_tfidf=tfidf_vectorizer_title.fit_transform(X_train['project_title'])

cv_title_tfidf = tfidf_vectorizer_title.transform(X_cv['project_title'])
te_title_tfidf = tfidf_vectorizer_title.transform(X_test['project_title'])

print("Shape of matrix TFIDF Vectorizer on essays \nTrain data-{},\nCV data\t-
{}\nTest data-{}".format(tr_title_tfidf.shape,cv_title_tfidf.shape,te_title_tf
idf.shape))
```

```
Shape of matrix TFIDF Vectorizer on essays
Train data-(49041, 1978),
CV data -(24155, 1978)
Test data-(36052, 1978)
```

In [40]:
```
print('Sample of TFIDF Vectorizer on `project_title`')
print('='*50)
print(tr_title_tfidf.toarray()[0:1,180:200])
print(tfidf_vectorizer_title.get_feature_names()[180:200])
```

```
Sample of TFIDF Vectorizer on `project_title`
==================================================
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
['bookshelf', 'bookshelves', 'bookworms', 'boom', 'boost', 'boosting', 'bor
ed', 'boredom', 'boring', 'bot', 'bots', 'bounce', 'bouncing', 'bouncy', 'b
ound', 'box', 'boxes', 'boys', 'brain', 'brains']
```

In [41]:
```
'''# Reading glove vectors in python: https://stackoverflow.com/a/38230349/408
4039
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 preprocessed_essays:
    words.extend(i.split(' '))

for i in preprocessed_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 coup
us", \
      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-save-and-load-variables-in-python/

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

Out[41]:

```
'\nwords = []\nfor i in preprocessed_essays:\n    words.extend(i.split(\'
\'))\n\nfor i in preprocessed_titles:\n    words.extend(i.split(\' \'))\npr
int("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.k
eys()).intersection(words)\nprint("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),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words
_courpus[i] = model[i]\nprint("word 2 vec length", len(words_courpus))\n\n
\n# stronging variables into pickle files python: http://www.jessicayung.co
m/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pickle
\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpu
s, f)'
```

In [42]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how
-to-use-pickle-to-save-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())
```

Using Pretrained Models: AVG W2V on `preprocessed_essay`

In [43]:
```
# average Word2Vec
# compute average word2vec for each review.
def AVG_w2v(preprocessed_data):
    avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in
 this list
    for sentence in tqdm(preprocessed_data): # 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/revie
w
        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.append(vector)
    return(avg_w2v_vectors)

#print(len(avg_w2v_vectors))
#print(len(avg_w2v_vectors[1]))
```

In [44]:
```
#using above defined function "AVG_w2v" to compute average word2vec for each r
eview in train,cv and test data.
tr_avg_w2v_vectors=AVG_w2v(X_train['essay'])
```

```
cv_avg_w2v_vectors=AVG_w2v(X_cv['essay'])
te_avg_w2v_vectors=AVG_w2v(X_test['essay'])
```

```
100%|
| 49041/49041 [00:22<00:00, 2222.35it/s]
100%|
| 24155/24155 [00:09<00:00, 2465.80it/s]
100%|
| 36052/36052 [00:14<00:00, 2442.99it/s]
```

In [45]:
```
print(len(tr_avg_w2v_vectors),len(cv_avg_w2v_vectors),len(te_avg_w2v_vectors))
```

```
49041 24155 36052
```

Using Pretrained Models: AVG W2V on `project_title`

In [46]:
```
#using above defined function "AVG_w2v" to compute average word2vec for each r
eview in train,cv and test data.
tr_avg_w2v_vectors_project_title=AVG_w2v(X_train['project_title'])
cv_avg_w2v_vectors_project_title=AVG_w2v(X_cv['project_title'])
te_avg_w2v_vectors_project_title=AVG_w2v(X_test['project_title'])
```

```
100%|
| 49041/49041 [00:01<00:00, 27246.62it/s]
100%|
| 24155/24155 [00:00<00:00, 27465.80it/s]
100%|
| 36052/36052 [00:01<00:00, 32646.15it/s]
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [47]:
```
# 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 = set(tfidf_model.get_feature_names())
```

In [48]:
```
# average Word2Vec
# compute average word2vec for each review.
def tfidf_w2v(preprocessed_data,words):
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored i
```

```
n this list
    for sentence in tqdm(preprocessed_data): # 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/r
eview
        for word in sentence.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in 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.s
plit())) # 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.append(vector)
    return(tfidf_w2v_vectors)
```

In [49]:
```
#using above defined function "tfidf w2v" to compute average word2vec for each
review in train,cv and test data.
words=tfidf_words
tr_tfidf_w2v_vectors=tfidf_w2v(X_train['essay'],words)
cv_tfidf_w2v_vectors=tfidf_w2v(X_cv['essay'],words)
te_tfidf_w2v_vectors=tfidf_w2v(X_test['essay'],words)
```

```
100%|██████████| 49041/49041 [02:11<00:00, 371.84it/s]
100%|██████████| 24155/24155 [01:08<00:00, 352.49it/s]
100%|██████████| 36052/36052 [01:46<00:00, 340.11it/s]
```

Using Pretrained Models: TFIDF weighted W2V on `project_title`

In [50]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_project_title = TfidfVectorizer()
tfidf_model_project_title.fit(X_train['project_title'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_project_title.get_feature_names(), list(tfid
f_model_project_title.idf_)))
tfidf_project_title_words = set(tfidf_model_project_title.get_feature_names())
```

In [51]:
```
#using above defined function "tfidf w2v" to compute average word2vec for each
```

```
review in train,cv and test data.
words=tfidf_project_title_words
tr_tfidf_w2v_project_title_vectors=tfidf_w2v(X_train['project_title'],words)
cv_tfidf_w2v_project_title_vectors=tfidf_w2v(X_cv['project_title'],words)
te_tfidf_w2v_project_title_vectors=tfidf_w2v(X_test['project_title'],words)
```

```
100%|
| 49041/49041 [00:02<00:00, 21235.24it/s]
100%|
| 24155/24155 [00:00<00:00, 27037.96it/s]
100%|
| 36052/36052 [00:01<00:00, 22125.91it/s]
```

In [52]:
```
print(len(tr_tfidf_w2v_project_title_vectors),len(cv_tfidf_w2v_project_title_v
ectors),len(te_tfidf_w2v_project_title_vectors))
```

```
49041 24155 36052
```

1.5.4 Merging all the above features

- we need to merge all the numerical vectors i.e catogorical, text, numerical vectors

In [53]:
```
%%time
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
#categorical, numerical features + project_title(BOW)
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a den
se matirx :)
tr_X_BOW= hstack((tr_school_state_one_hot,tr_categories_one_hot,tr_sub_categor
ies_one_hot,tr_teacher_prefix_one_hot,tr_grade_category_one_hot,tr_price_stand
ardized,tr_teacher_number_of_previously_posted_projects_standardized,tr_text_b
ow_title,tr_text_bow)).tocsr()
cv_X_BOW= hstack((cv_school_state_one_hot,cv_categories_one_hot,cv_sub_categor
ies_one_hot,cv_teacher_prefix_one_hot,cv_grade_category_one_hot,cv_price_stand
ardized,cv_teacher_number_of_previously_posted_projects_standardized,cv_text_b
ow_title,cv_text_bow)).tocsr()
te_X_BOW= hstack((te_school_state_one_hot,te_categories_one_hot,te_sub_categor
ies_one_hot,te_teacher_prefix_one_hot,te_grade_category_one_hot,te_price_stand
ardized,te_teacher_number_of_previously_posted_projects_standardized,te_text_b
ow_title,te_text_bow)).tocsr()
tr_X_BOW=tr_X_BOW.toarray()
cv_X_BOW=cv_X_BOW.toarray()
te_X_BOW=te_X_BOW.toarray()
print(tr_X_BOW.shape)
```

```
print(cv_X_BOW.shape)
print(te_X_BOW.shape)
```

```
(49041, 7080)
(24155, 7080)
(36052, 7080)
Wall time: 2.99 s
```

In [54]:

```
%%time
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
#categorical, numerical features + project_title(TFIDF)
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a den
se matirx :)
tr_X_TFIDF= hstack((tr_school_state_one_hot,tr_categories_one_hot, tr_sub_cate
gories_one_hot,tr_teacher_prefix_one_hot,tr_grade_category_one_hot,tr_price_st
andardized,tr_teacher_number_of_previously_posted_projects_standardized,tr_tit
le_tfidf,tr_text_tfidf))
cv_X_TFIDF= hstack((cv_school_state_one_hot,cv_categories_one_hot, cv_sub_cate
gories_one_hot,cv_teacher_prefix_one_hot,cv_grade_category_one_hot,cv_price_st
andardized,cv_teacher_number_of_previously_posted_projects_standardized,cv_tit
le_tfidf,cv_text_tfidf))
te_X_TFIDF= hstack((te_school_state_one_hot,te_categories_one_hot, te_sub_cate
gories_one_hot,te_teacher_prefix_one_hot,te_grade_category_one_hot,te_price_st
andardized,te_teacher_number_of_previously_posted_projects_standardized,te_tit
le_tfidf,te_text_tfidf))

tr_X_TFIDF=tr_X_TFIDF.toarray()
cv_X_TFIDF=cv_X_TFIDF.toarray()
te_X_TFIDF=te_X_TFIDF.toarray()
print(tr_X_TFIDF.shape)
print(cv_X_TFIDF.shape)
print(te_X_TFIDF.shape)
```

```
(49041, 5080)
(24155, 5080)
(36052, 5080)
Wall time: 6.85 s
```

In [55]:

```
%%time
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# categorical, numerical features + project_title(AVG W2V)
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a den
se matirx :)
tr_X_AVG_W2V= hstack((tr_school_state_one_hot,tr_categories_one_hot, tr_sub_ca
```

```
tegories_one_hot,tr_teacher_prefix_one_hot,tr_grade_category_one_hot,tr_price_
standardized,tr_teacher_number_of_previously_posted_projects_standardized,tr_a
vg_w2v_vectors_project_title,tr_avg_w2v_vectors))
cv_X_AVG_W2V= hstack((cv_school_state_one_hot,cv_categories_one_hot, cv_sub_ca
tegories_one_hot,cv_teacher_prefix_one_hot,cv_grade_category_one_hot,cv_price_
standardized,cv_teacher_number_of_previously_posted_projects_standardized,cv_a
vg_w2v_vectors_project_title,cv_avg_w2v_vectors))
te_X_AVG_W2V= hstack((te_school_state_one_hot,te_categories_one_hot, te_sub_ca
tegories_one_hot,te_teacher_prefix_one_hot,te_grade_category_one_hot,te_price_
standardized,te_teacher_number_of_previously_posted_projects_standardized,te_a
vg_w2v_vectors_project_title,te_avg_w2v_vectors))
tr_X_AVG_W2V=tr_X_AVG_W2V.toarray()
cv_X_AVG_W2V=cv_X_AVG_W2V.toarray()
te_X_AVG_W2V=te_X_AVG_W2V.toarray()
print(tr_X_AVG_W2V.shape,cv_X_AVG_W2V.shape,te_X_AVG_W2V.shape)
```

```
(49041, 702) (24155, 702) (36052, 702)
Wall time: 15.4 s
```

In [56]:

```
%%time
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# categorical, numerical features + project_title(TFIDF W2V)
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a den
se matirx :)
tr_X_tfidf_w2v= hstack((tr_school_state_one_hot,tr_categories_one_hot, tr_sub_
categories_one_hot,tr_teacher_prefix_one_hot,tr_grade_category_one_hot,tr_pric
e_standardized,tr_teacher_number_of_previously_posted_projects_standardized,tr
_tfidf_w2v_project_title_vectors,tr_tfidf_w2v_vectors))
cv_X_tfidf_w2v= hstack((cv_school_state_one_hot,cv_categories_one_hot, cv_sub_
categories_one_hot,cv_teacher_prefix_one_hot,cv_grade_category_one_hot,cv_pric
e_standardized,cv_teacher_number_of_previously_posted_projects_standardized,cv
_tfidf_w2v_project_title_vectors,cv_tfidf_w2v_vectors))
te_X_tfidf_w2v= hstack((te_school_state_one_hot,te_categories_one_hot, te_sub_
categories_one_hot,te_teacher_prefix_one_hot,te_grade_category_one_hot,te_pric
e_standardized,te_teacher_number_of_previously_posted_projects_standardized,te
_tfidf_w2v_project_title_vectors,te_tfidf_w2v_vectors))
tr_X_tfidf_w2v=tr_X_tfidf_w2v.toarray()
cv_X_tfidf_w2v=cv_X_tfidf_w2v.toarray()
te_X_tfidf_w2v=te_X_tfidf_w2v.toarray()
print(tr_X_tfidf_w2v.shape)
print(cv_X_tfidf_w2v.shape)
print(te_X_tfidf_w2v.shape)
```

```
(49041, 702)
(24155, 702)
```

```
(36052, 702)
Wall time: 6.89 s
```

2.4 Appling LR on different kind of featurization as mentioned in the instructions

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

Hyper paramter tuning to find best Alpha(α)

In [57]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#Drawing confusion matrix
def draw_confusion_matrix(clf,threshold,y_true,y_hat,tpr,fpr,t):
    result=[]
    y_pred=[]

    #finding threshold which maximises the tpr and minimises the fpr
    thr=threshold[np.argmax((tpr*(1-fpr)))]

    for probab in y_hat:

        if probab >= thr:
            y_pred.append(1)
        else:
            y_pred.append(0)

    result=confusion_matrix(y_true,y_pred,labels=[0,1])
    df_cm = pd.DataFrame(result,range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    plt.figure(figsize = (5,3))
    plt.title(t)
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 12}, fmt='g')
```

In [58]:
```
#function to plot lines
def plot_curve(train_auc_scores_tmp,validation_auc_scores_tmp,C,title):
    plt.xscale('log')
    plt.plot(C,train_auc_scores_tmp,label="Train curve")
```

```
    plt.plot(C,validation_auc_scores_tmp,label="Validation curve")
    plt.scatter(C, train_auc_scores_tmp, label='Train AUC points')
    plt.scatter(C, validation_auc_scores_tmp, label='CV AUC points')
    plt.title(title)
    plt.xlabel("Alpha(α)-hyper paramters")
    plt.ylabel("AUC")
    plt.legend()
    plt.show()
```

In [59]:

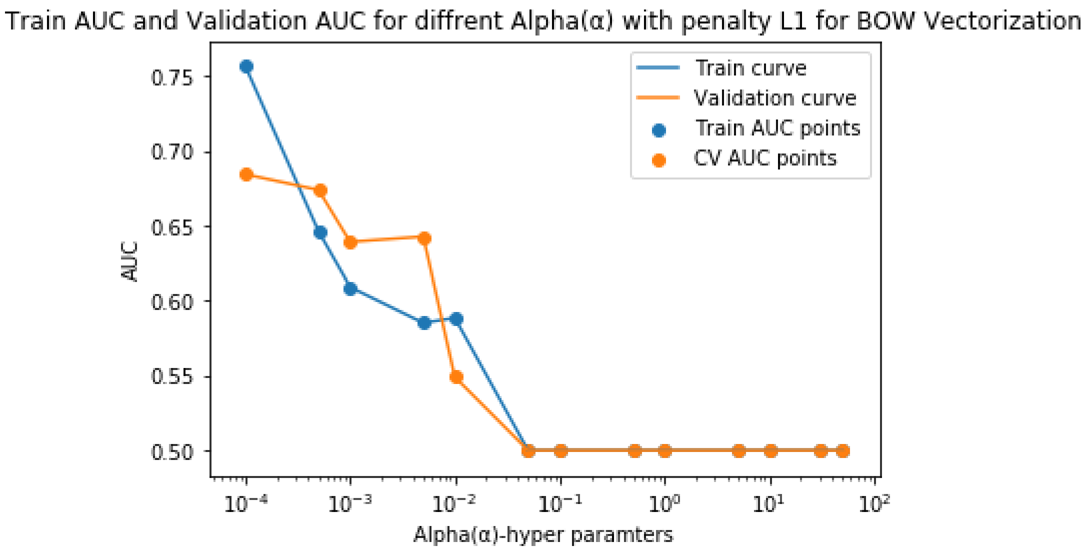
```
#refered link :https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
def roc_auc_compute(x_train,y_train,x_test_temp,y_test_temp,C,title):
    n_neighbors=C
    regularization=['l1','l2']
    i=0
    for reg in regularization:
        train_auc_scores=[]
        validation_auc_scores=[]
        train_cv_scores=[]
        validation_cv_scores=[]
        best_cv_auc_scores=0
        for c in tqdm(n_neighbors):
            class_w={0:0.5,1:0.5}
            parameters = {'alpha':[c]}
            trained_SVM = SGDClassifier(alpha=c,class_weight=class_w,penalty=reg)
            #trainning model
            trained_SVM.fit(x_train,y_train)
            calibrated = CalibratedClassifierCV(trained_SVM, method='isotonic', cv=5)
            calibrated.fit(x_train, y_train)
            # predict the response on the cross validation
            pradicted_labels=calibrated.predict_proba(x_test_temp)
            #Calculating validation auc scores
            validation_auc=roc_auc_score(y_test_temp,pradicted_labels[:,1]) #1-roc_auc_score for validation error
            # predict the response on the train and calculating the train auc
            train_auc=roc_auc_score(y_train,calibrated.predict_proba(x_train)[:,1]) #1-roc_auc_score for train error
```

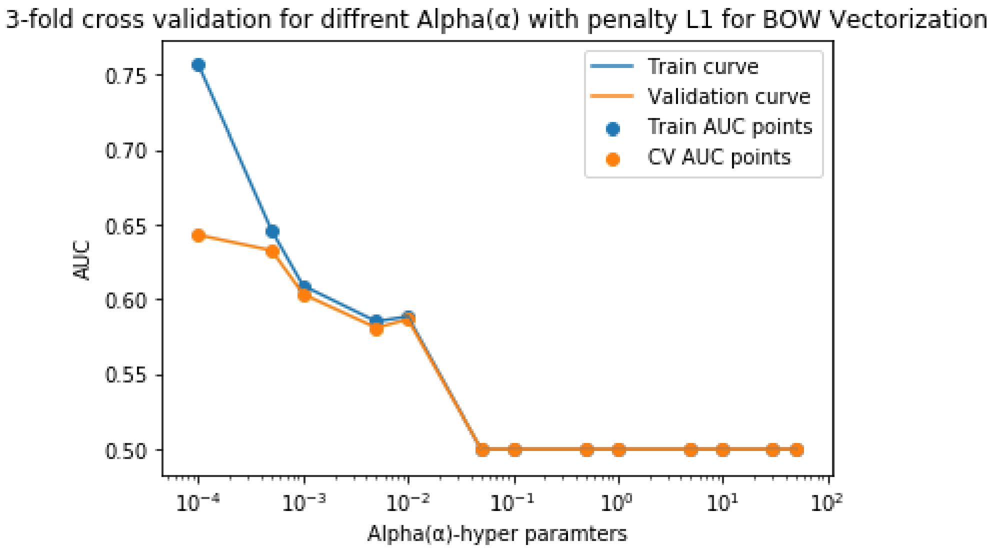
```
            # K-flod cross validation
            gs = GridSearchCV(trained_SVM,parameters,cv=3,scoring='roc_auc')
            gs.fit(x_train,y_train)
            train_auc= gs.cv_results_['mean_train_score']
            cv_auc = gs.cv_results_['mean_test_score']

            train_cv_scores.append(train_auc)
            validation_cv_scores.append(cv_auc)
            train_auc_scores.append(train_auc)
            validation_auc_scores.append(validation_auc)

            if cv_auc>=best_cv_auc_scores:
                best_cv_auc_scores=cv_auc

        plot_curve(train_auc_scores,validation_auc_scores,n_neighbors,title[i
])
        i=i+1
        plot_curve(train_cv_scores,validation_cv_scores,n_neighbors,title[i])
        print("best AUC",best_cv_auc_scores)
        print("best Alpha(α)",C[validation_cv_scores.index(best_cv_auc_scores
)])
        i=i+1
```

Applying SVM on BOW

In [150]:
```
%%time
title=["Train AUC and Validation AUC for diffrent Alpha(α) with penalty L1 for
BOW Vectorization","3-fold cross validation for diffrent Alpha(α) with penalty
L1 for BOW Vectorization","Train AUC and Validation AUC for diffrent Alpha(α)
 with penalty L2 for BOW Vectorization","3-fold cross validation for diffrent
 Alpha(α) with penalty L2 for BOW Vectorization"]
roc_auc_compute(tr_X_BOW,Y_train,cv_X_BOW,Y_cv,[50,30,10,5,1,0.5,0.1,0.05,0.0
1,0.005,0.001,0.0005,0.0001],title)
```

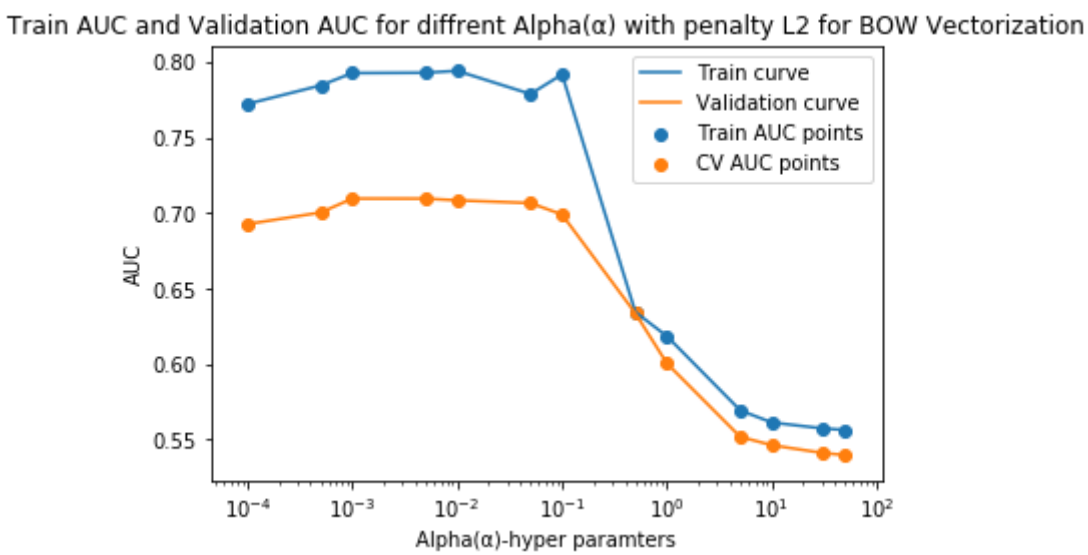
```
100%|
| 13/13 [20:00<00:00, 97.08s/it]
```

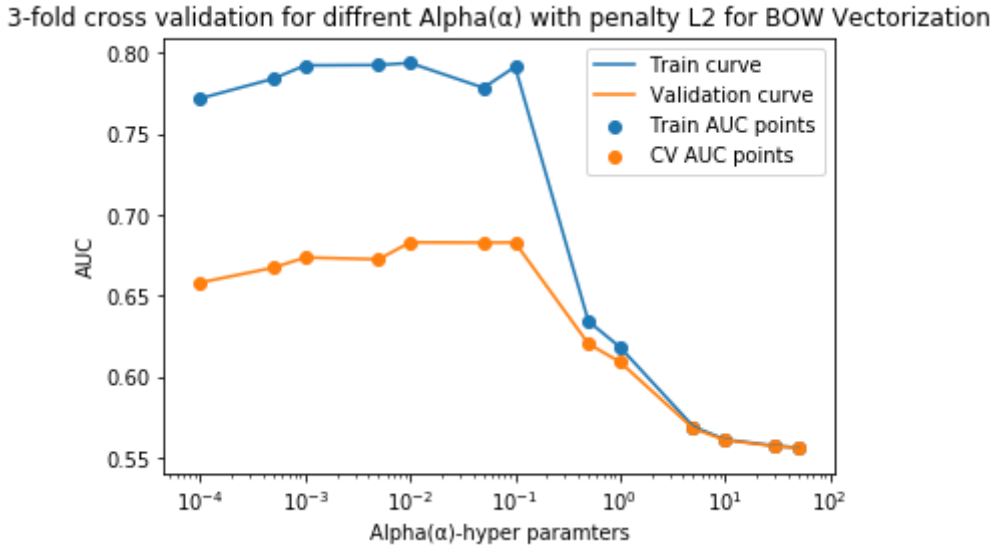





```
best AUC [0.64291932]
best Alpha(α) 0.0001
```

```
100%|[redacted]| 13/13 [11:27<00:00, 49.28s/it]
```

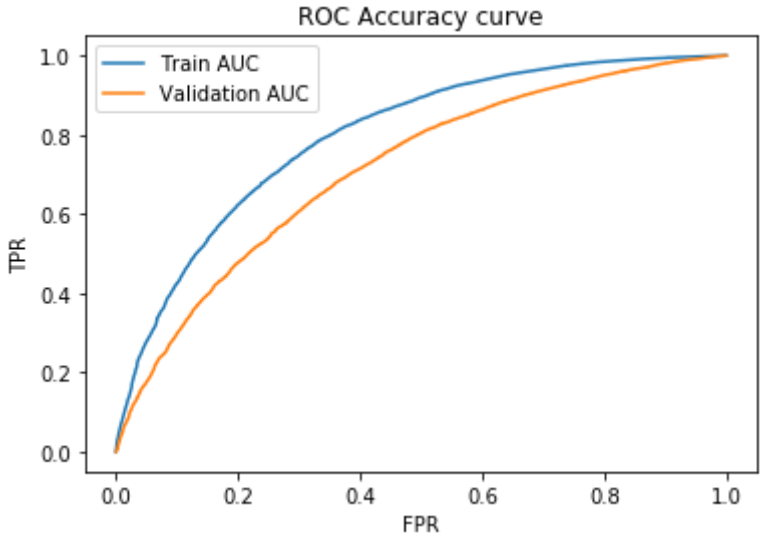




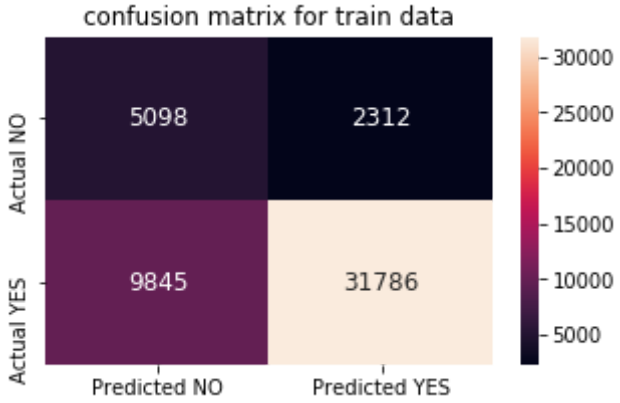

```
best AUC [0.68314397]
best Alpha(α) 0.01
Wall time: 31min 29s
```

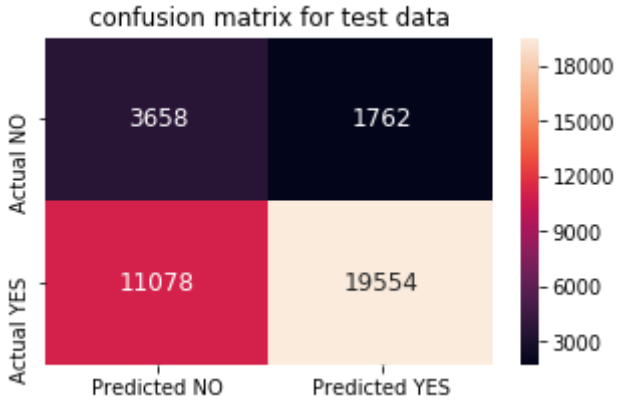
In [156]:
```
%%time
class_w={0:0.5,1:0.5}
```

```
#https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for
-classification-in-python/
from sklearn.metrics import roc_curve, auc
#training model with best lambda(λ) paramter
trained_SVM_BOW=SGDClassifier(alpha=0.01,class_weight=class_w,penalty='l2')
#trainning model
trained_SVM_BOW.fit(tr_X_BOW,Y_train)
calibrated = CalibratedClassifierCV(trained_SVM_BOW, method='isotonic')
calibrated.fit(tr_X_BOW, Y_train)
# predict the response on the train data
predicted_labels_train=calibrated.predict_proba(tr_X_BOW)
# predict the response on the test data
predicted_labels_test=calibrated.predict_proba(te_X_BOW)
#Calculating FPR and TPR for train and test data
tr_fpr,tr_tpr,tr_threshold=roc_curve(Y_train,predicted_labels_train[:,1])
te_fpr,te_tpr,te_threshold=roc_curve(Y_test,predicted_labels_test[:,1])
#drawing ROC ROC Accuracy curve for test and train data
plt.plot(tr_fpr,tr_tpr,label="Train AUC")
plt.plot(te_fpr,te_tpr,label="Validation AUC")
plt.title("ROC Accuracy curve")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.legend()
plt.show()
print("Train AUC =",round(auc(tr_fpr,tr_tpr),2))
print("Test AUC =",round(auc(te_fpr,te_tpr),2))
#drawing confusion matrix for test and train data
t2="confusion matrix for train data"
draw_confusion_matrix(calibrated,tr_threshold,Y_train,predicted_labels_train
[:,1],tr_tpr,tr_fpr,t2)
t1="confusion matrix for test data"
draw_confusion_matrix(calibrated,tr_threshold,Y_test,predicted_labels_test[:,
1],te_tpr,te_fpr,t1)
```




```
Train AUC = 0.8
Test AUC = 0.71
Wall time: 21.3 s
```

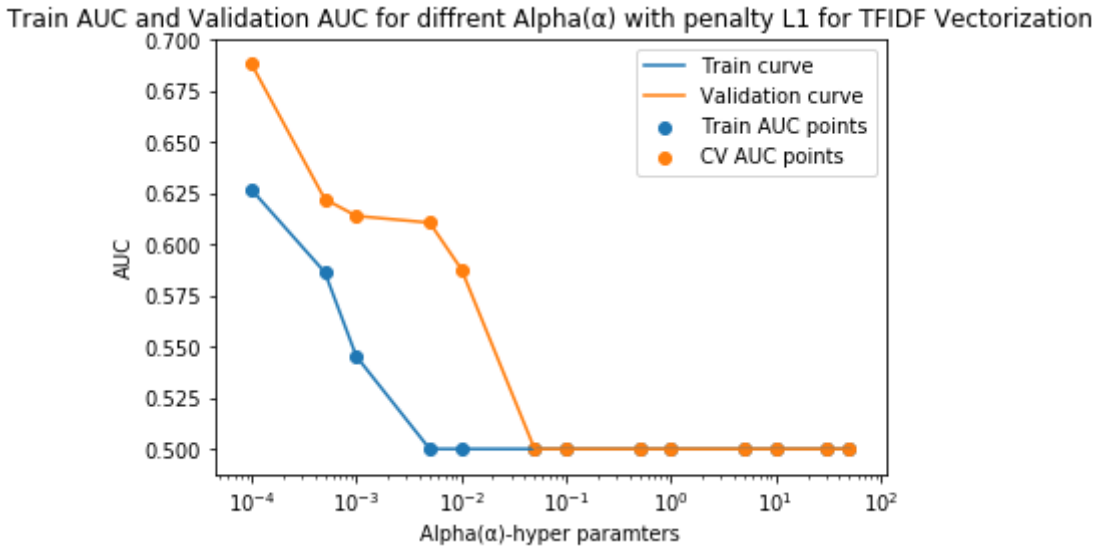


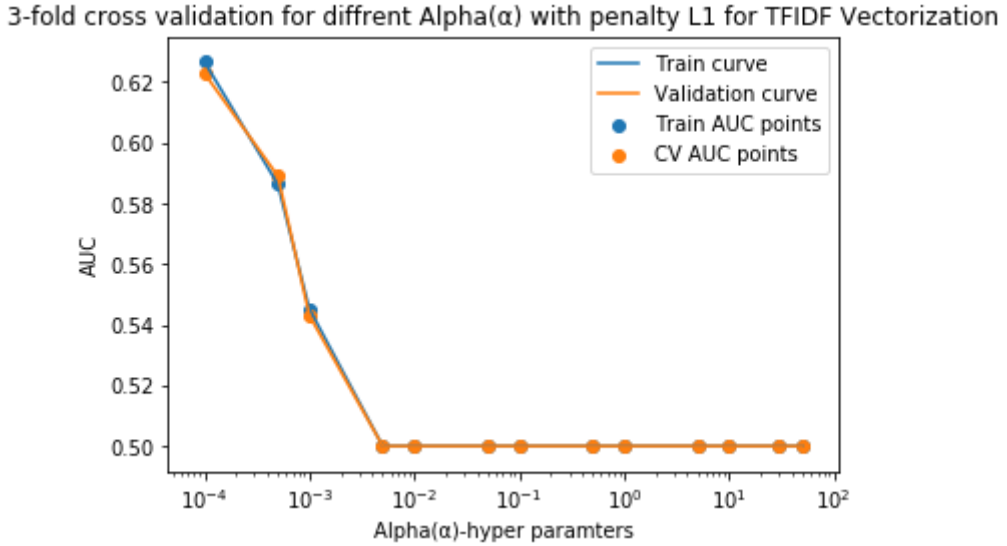


Applying SVM on TFIDF

In [60]:
```
%%time
title=["Train AUC and Validation AUC for diffrent Alpha(α) with penalty L1 for TFIDF Vectorization","3-fold cross validation for diffrent Alpha(α) with penalty L1 for TFIDF Vectorization","Train AUC and Validation AUC for diffrent Alpha(α) with penalty L2 for TFIDF Vectorization","3-fold cross validation for diffrent Alpha(α) with penalty L2 for TFIDF Vectorization"]
roc_auc_compute(tr_X_TFIDF,Y_train,cv_X_TFIDF,Y_cv,[50,30,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],title)
```

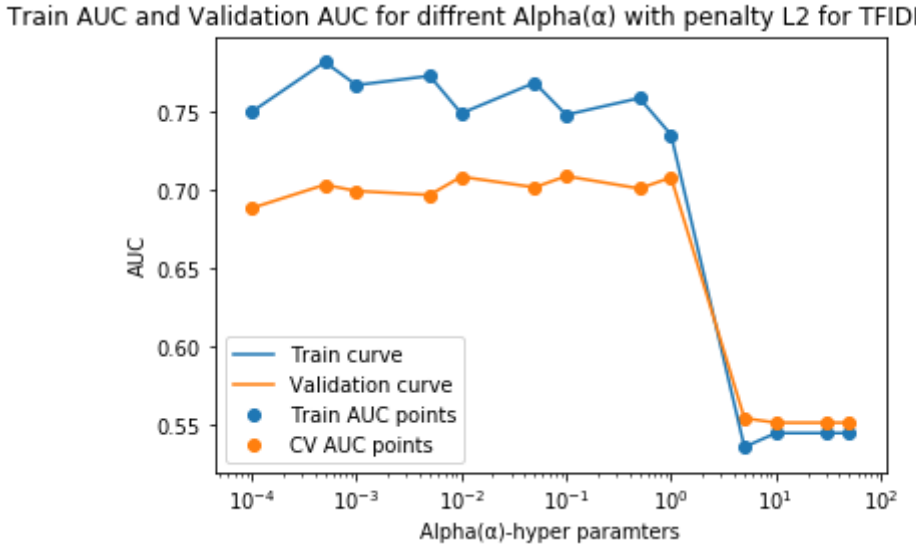
```
100%|██████████| 13/13 [14:34<00:00, 61.71s/it]
```

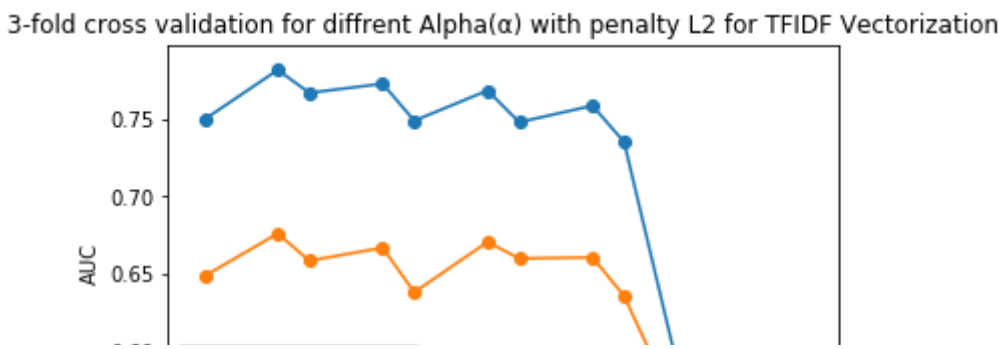





```
best AUC [0.62288283]
best Alpha(α) 0.0001
```

```
100%|██████████| 13/13 [09:01<00:00, 40.44s/it]
```



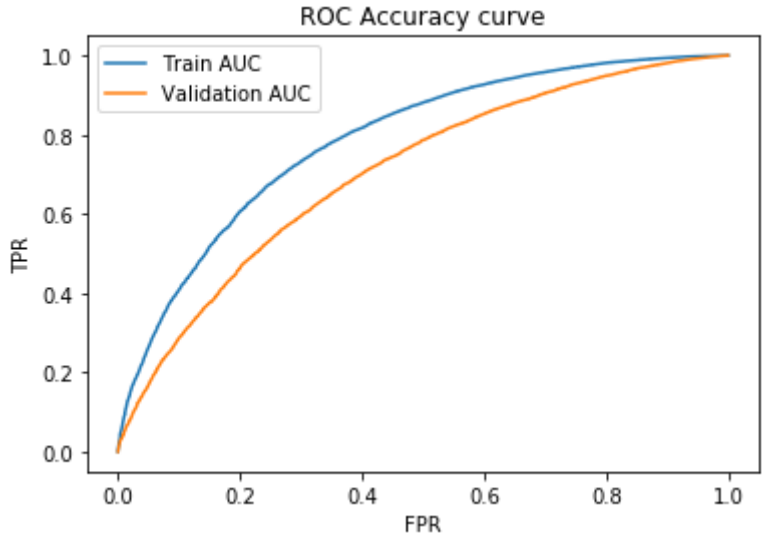



```
best AUC [0.67581356]
best Alpha(α) 0.0005
Wall time: 23min 39s
```

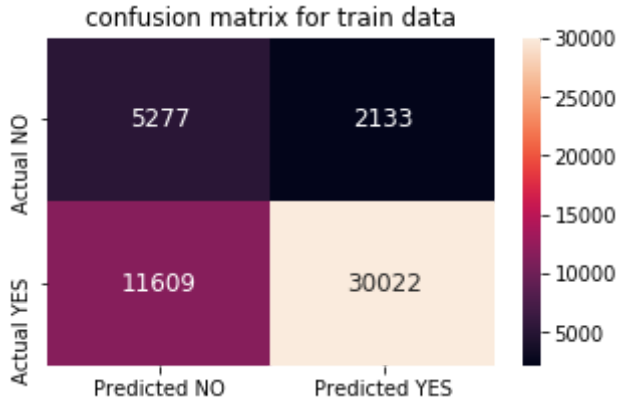
In [73]:

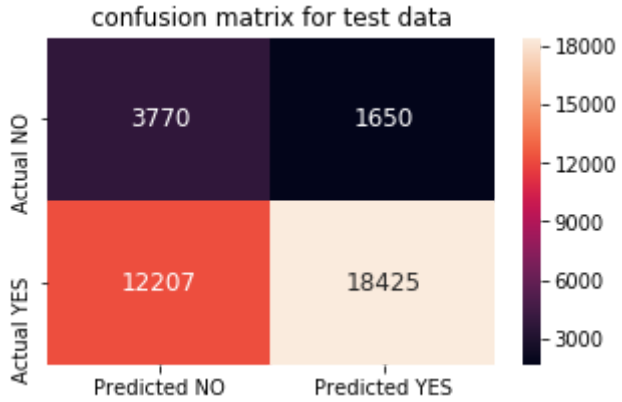
```
%%time
class_w={0:0.5,1:0.5}
#https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for
-classification-in-python/
from sklearn.metrics import roc_curve, auc
#training model with best lambda(λ) paramter
trained_SVM_TFIDF=SGDClassifier(alpha=0.0005,class_weight=class_w,penalty='l
2')
#trainning model
trained_SVM_TFIDF.fit(tr_X_TFIDF,Y_train)
calibrated = CalibratedClassifierCV(trained_SVM_TFIDF, method='isotonic')
calibrated.fit(tr_X_TFIDF, Y_train)
# predict the response on the train data
predicted_labels_train=calibrated.predict_proba(tr_X_TFIDF)
# predict the response on the test data
predicted_labels_test=calibrated.predict_proba(te_X_TFIDF)
#Calculating FPR and TPR for train and test data
tr_fpr,tr_tpr,tr_threshold=roc_curve(Y_train,predicted_labels_train[:,1])
te_fpr,te_tpr,te_threshold=roc_curve(Y_test,predicted_labels_test[:,1])
#drawing ROC ROC Accuracy curve for test and train data
plt.plot(tr_fpr,tr_tpr,label="Train AUC")
plt.plot(te_fpr,te_tpr,label="Validation AUC")
plt.title("ROC Accuracy curve")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.legend()
plt.show()
print("Train AUC =",round(auc(tr_fpr,tr_tpr),2))
print("Test AUC =",round(auc(te_fpr,te_tpr),2))
#drawing confusion matrix for test and train data
t2="confusion matrix for train data"
```

```
draw_confusion_matrix(calibrated,tr_threshold,Y_train,predicted_labels_train
[:,1],tr_tpr,tr_fpr,t2)
t1="confusion matrix for test data"
draw_confusion_matrix(calibrated,tr_threshold,Y_test,predicted_labels_test[:,
1],te_tpr,te_fpr,t1)
```




```
Train AUC = 0.79
Test AUC = 0.71
Wall time: 16.6 s
```

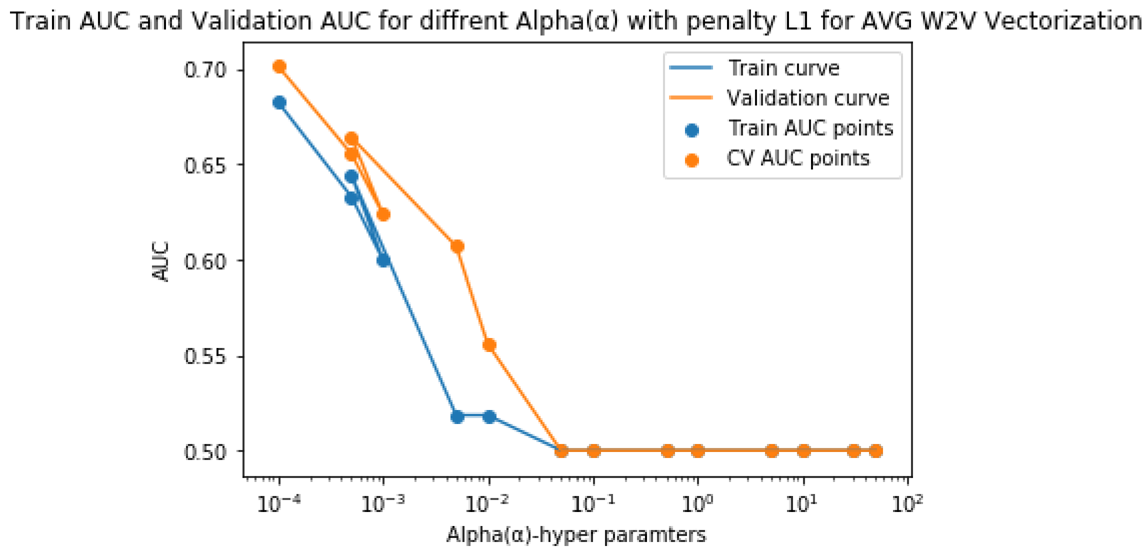


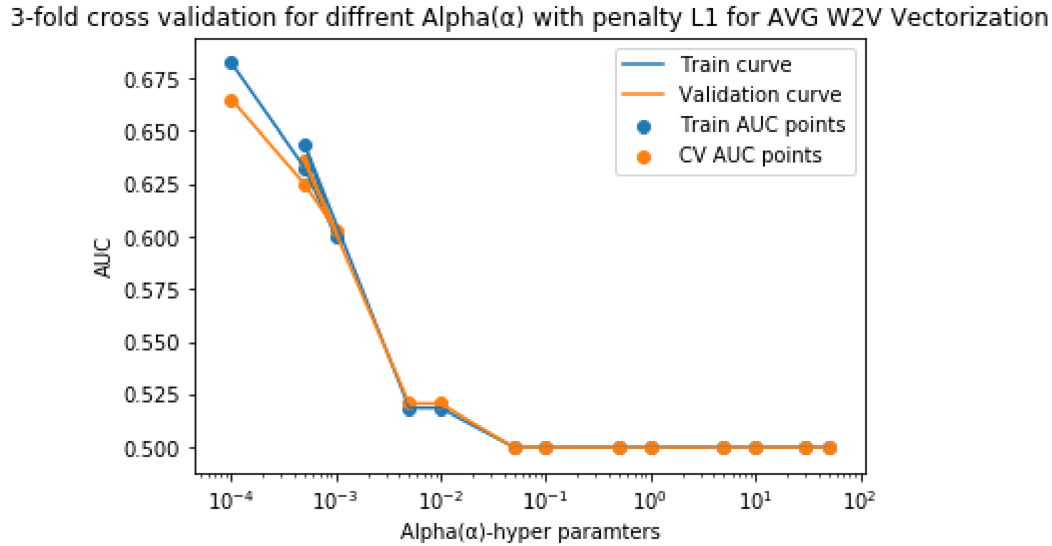


Applying SVM on AVG W2V

In [78]:
```
%%time
title=["Train AUC and Validation AUC for diffrent Alpha(α) with penalty L1 for
AVG W2V Vectorization","3-fold cross validation for diffrent Alpha(α) with pen
alty L1 for AVG W2V Vectorization","Train AUC and Validation AUC for diffrent
 Alpha(α) with penalty L2 for AVG W2V Vectorization","3-fold cross validation
 for diffrent Alpha(α) with penalty L2 for AVG W2V Vectorization"]
roc_auc_compute(tr_X_AVG_W2V,Y_train,cv_X_AVG_W2V,Y_cv,[50,30,10,5,1,0.5,0.1,
0.05,0.01,0.005,0.0005,0.001,0.0005,0.0001],title)
```

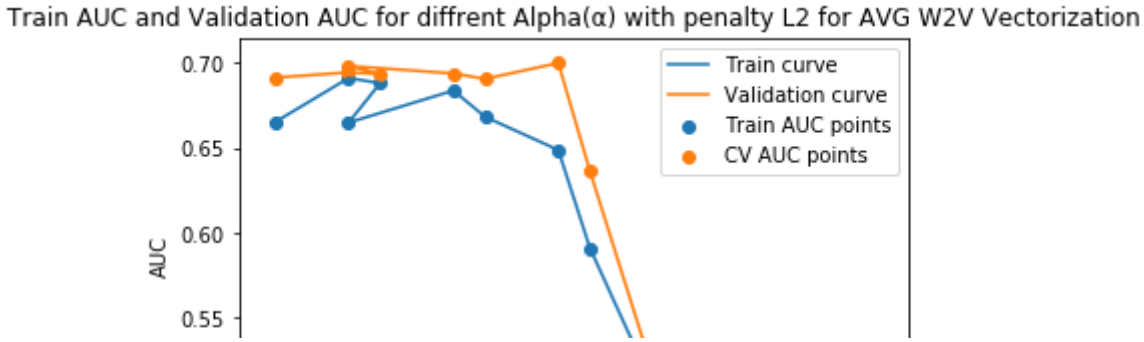
```
100%|██████████| 14/14 [02:56<00:00, 11.48s/it]
```

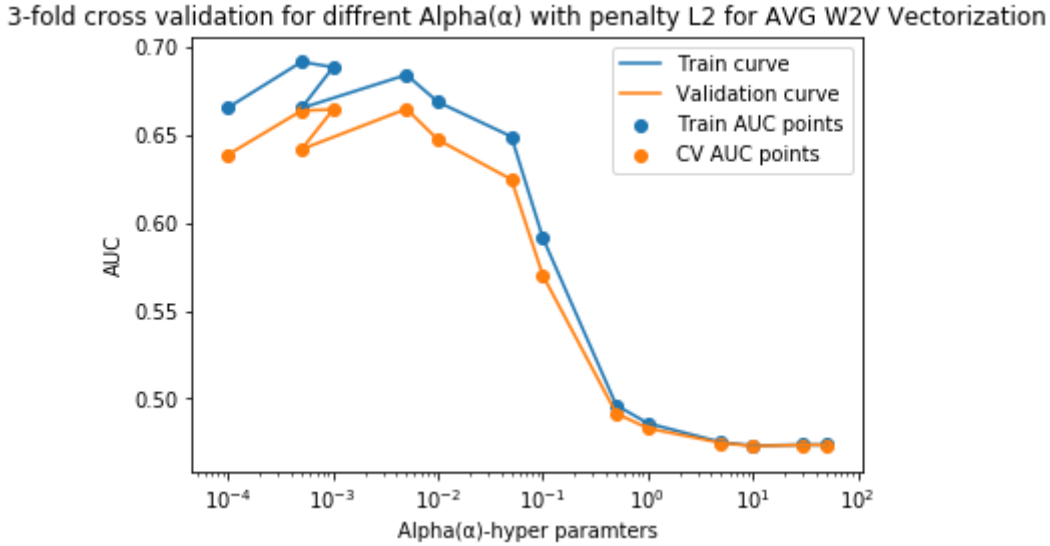





```
best AUC [0.66511433]
best Alpha(α) 0.0001
```

```
100%|████████████████████████████████████| 14/14 [01:20<00:00,  5.54s/it]
```



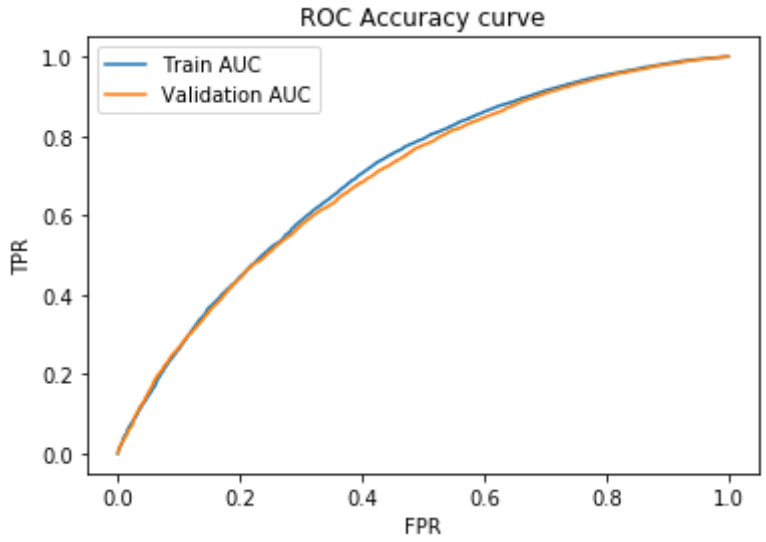



```
best AUC [0.66454829]
best Alpha(α) 0.005
Wall time: 4min 18s
```

In [143]:

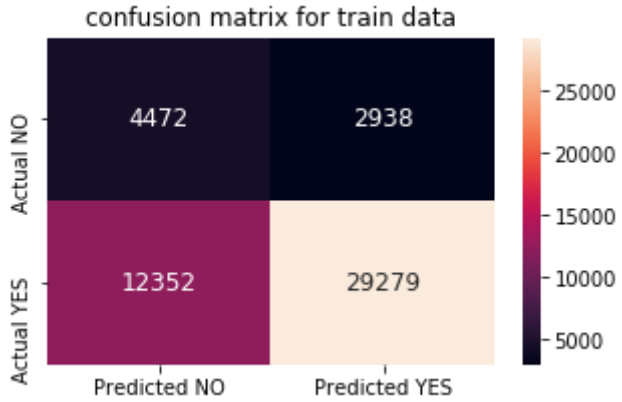
```
%%time
class_w={0:0.5,1:0.5}
#https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for
-classification-in-python/
from sklearn.metrics import roc_curve, auc
#training model with best lambda(λ) paramter
trained_SVM_AVG_W2V=SGDClassifier(alpha=0.0001,class_weight=class_w,penalty='l
1')
#trainning model
```

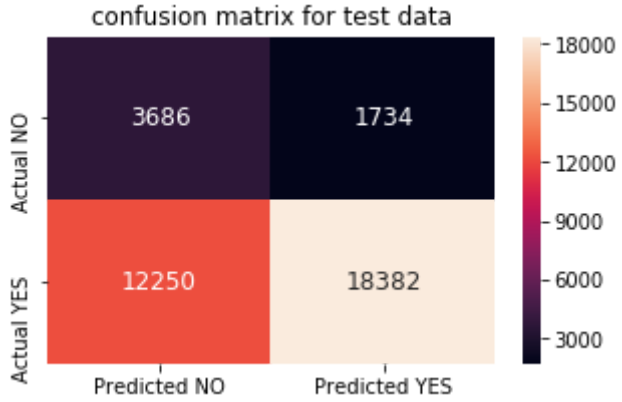
```
trained_SVM_AVG_W2V.fit(tr_X_AVG_W2V,Y_train)
calibrated = CalibratedClassifierCV(trained_SVM_AVG_W2V, method='isotonic')
calibrated.fit(tr_X_AVG_W2V, Y_train)
# predict the response on the train data
predicted_labels_train=calibrated.predict_proba(tr_X_AVG_W2V)
# predict the response on the test data
predicted_labels_test=calibrated.predict_proba(te_X_AVG_W2V)
#Calculating FPR and TPR for train and test data
tr_fpr,tr_tpr,tr_threshold=roc_curve(Y_train,predicted_labels_train[:,1])
te_fpr,te_tpr,te_threshold=roc_curve(Y_test,predicted_labels_test[:,1])
#drawing ROC ROC Accuracy curve for test and train data
plt.plot(tr_fpr,tr_tpr,label="Train AUC")
plt.plot(te_fpr,te_tpr,label="Validation AUC")
plt.title("ROC Accuracy curve")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.legend()
plt.show()
print("Train AUC =",round(auc(tr_fpr,tr_tpr),2))
print("Test AUC =",round(auc(te_fpr,te_tpr),2))
#drawing confusion matrix for test and train data
t2="confusion matrix for train data"
draw_confusion_matrix(calibrated,tr_threshold,Y_train,predicted_labels_train
[:,1],tr_tpr,tr_fpr,t2)
t1="confusion matrix for test data"
draw_confusion_matrix(calibrated,tr_threshold,Y_test,predicted_labels_test[:,
1],te_tpr,te_fpr,t1)
```




```
Train AUC = 0.7
Test AUC = 0.7
```

```
Wall time: 6.67 s
```

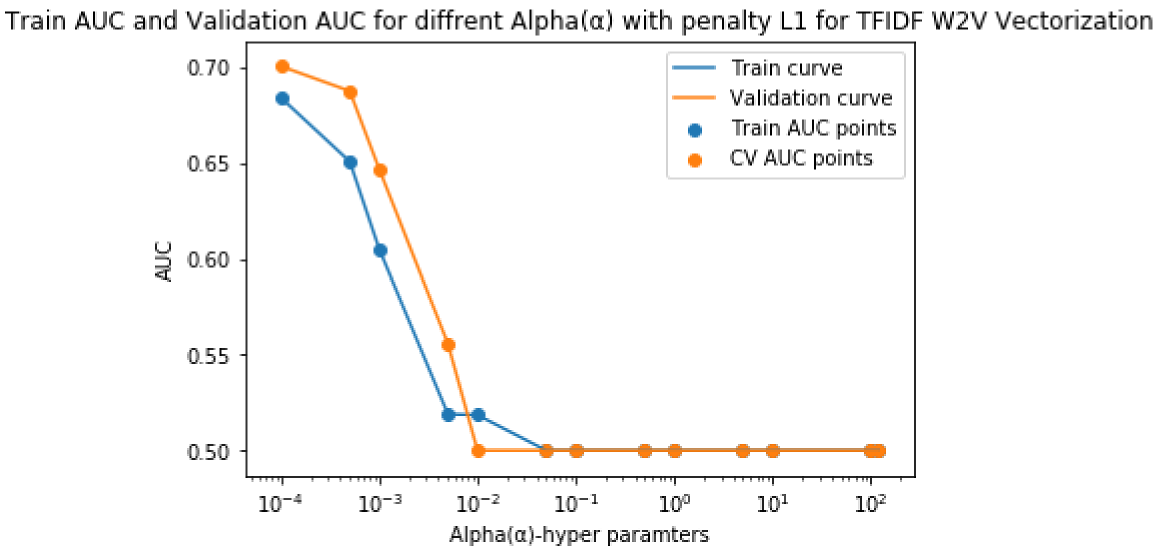


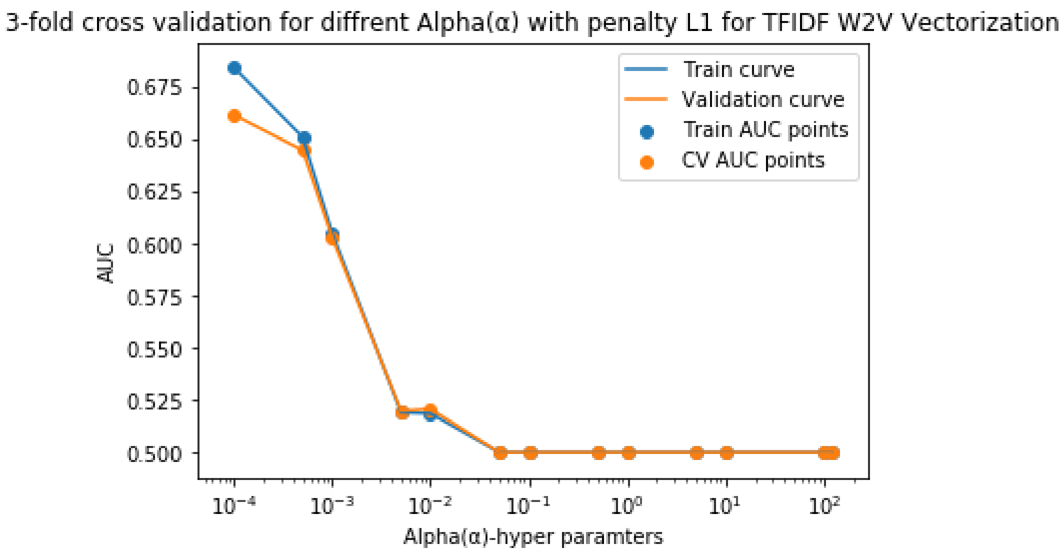


Applying SVM on TFIDF W2V

In [162]:
```
%%time
title=["Train AUC and Validation AUC for diffrent Alpha(α) with penalty L1 for TFIDF W2V Vectorization","3-fold cross validation for diffrent Alpha(α) with penalty L1 for TFIDF W2V Vectorization","Train AUC and Validation AUC for diffrent Alpha(α) with penalty L2 for TFIDF W2V Vectorization","3-fold cross validation for diffrent Alpha(α) with penalty L2 for TFIDF W2V Vectorization"]
roc_auc_compute(tr_X_tfidf_w2v,Y_train,cv_X_tfidf_w2v,Y_cv,[120,100,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001],title)
```

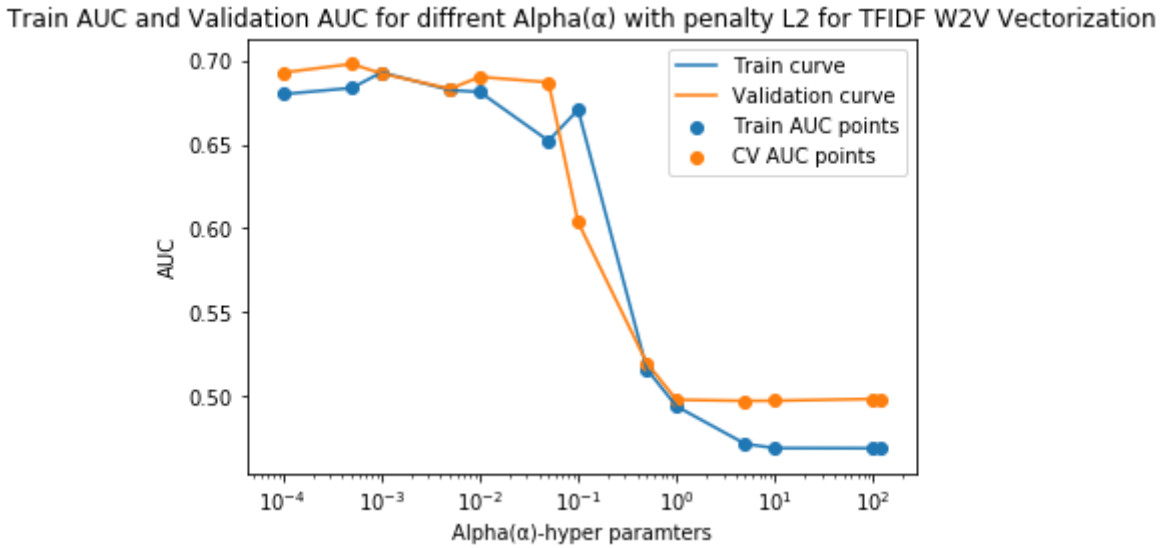
```
100%|██████████| 13/13 [03:01<00:00, 13.64s/it]
```

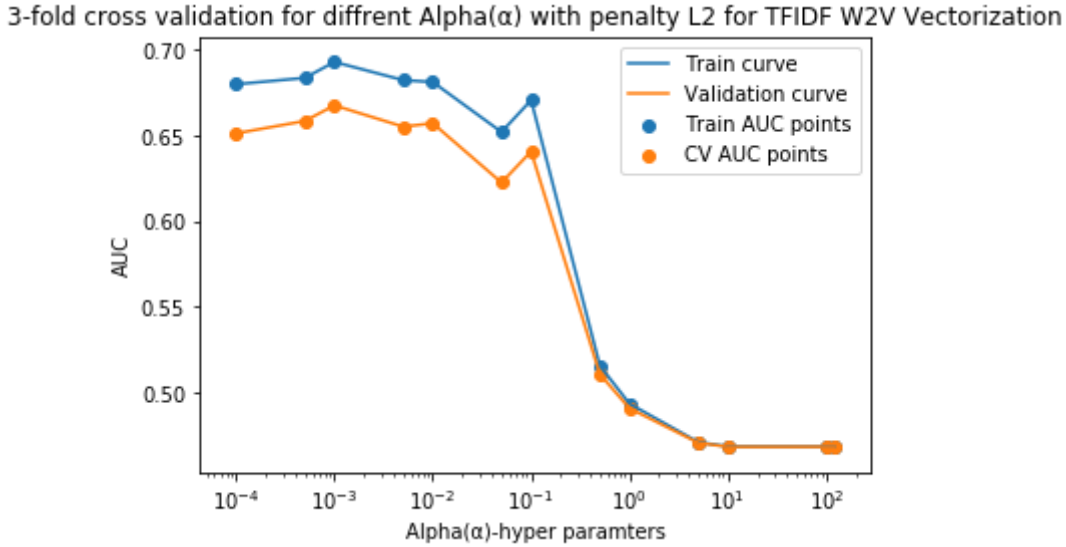





```
best AUC [0.66126891]
best Alpha(α) 0.0001
```

```
100%|[redacted]| 13/13 [01:30<00:00,  7.10s/it]
```



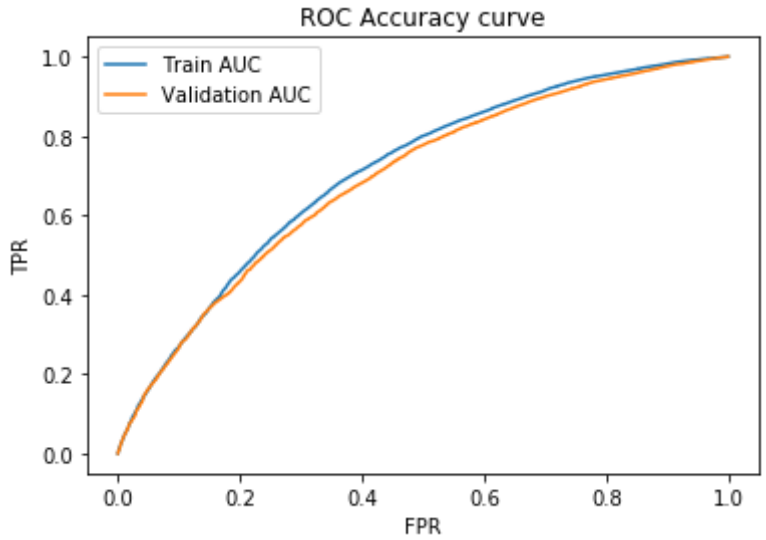



```
best AUC [0.66774208]
best Alpha(α) 0.001
Wall time: 4min 33s
```

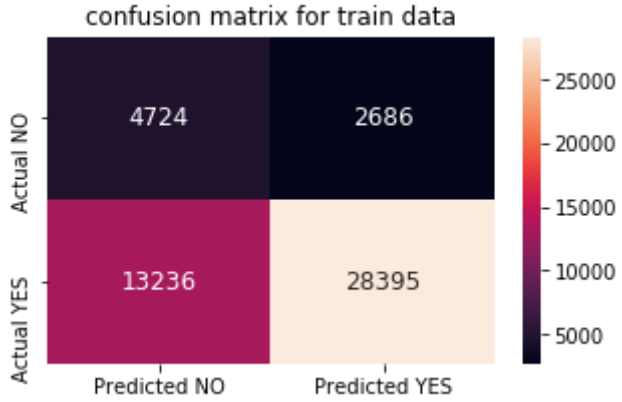
In [165]:

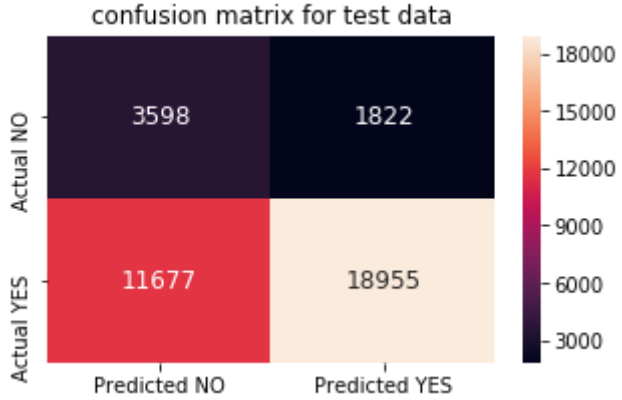
```
%%time
class_w={0:0.5,1:0.5}
#https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for
-classification-in-python/
from sklearn.metrics import roc_curve, auc
#training model with best lambda(λ) paramter
trained_SVM_tfidf_w2v=SGDClassifier(alpha=0.0005,class_weight=class_w,penalty
```

```
='l2')
#trainning model
trained_SVM_tfidf_w2v.fit(tr_X_tfidf_w2v,Y_train)
calibrated = CalibratedClassifierCV(trained_SVM_tfidf_w2v, method='isotonic')
calibrated.fit(tr_X_tfidf_w2v, Y_train)
# predict the response on the train data
predicted_labels_train=calibrated.predict_proba(tr_X_tfidf_w2v)
# predict the response on the test data
predicted_labels_test=calibrated.predict_proba(te_X_tfidf_w2v)
#Calculating FPR and TPR for train and test data
tr_fpr,tr_tpr,tr_threshold=roc_curve(Y_train,predicted_labels_train[:,1])
te_fpr,te_tpr,te_threshold=roc_curve(Y_test,predicted_labels_test[:,1])
#drawing ROC ROC Accuracy curve for test and train data
plt.plot(tr_fpr,tr_tpr,label="Train AUC")
plt.plot(te_fpr,te_tpr,label="Validation AUC")
plt.title("ROC Accuracy curve")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.legend()
plt.show()
print("Train AUC =",round(auc(tr_fpr,tr_tpr),2))
print("Test AUC =",round(auc(te_fpr,te_tpr),2))
#drawing confusion matrix for test and train data
t2="confusion matrix for train data"
draw_confusion_matrix(calibrated,tr_threshold,Y_train,predicted_labels_train
[:,1],tr_tpr,tr_fpr,t2)
t1="confusion matrix for test data"
draw_confusion_matrix(calibrated,tr_threshold,Y_test,predicted_labels_test[:,
1],te_tpr,te_fpr,t1)
```



```
Train AUC = 0.71
Test AUC = 0.69
Wall time: 3.23 s
```





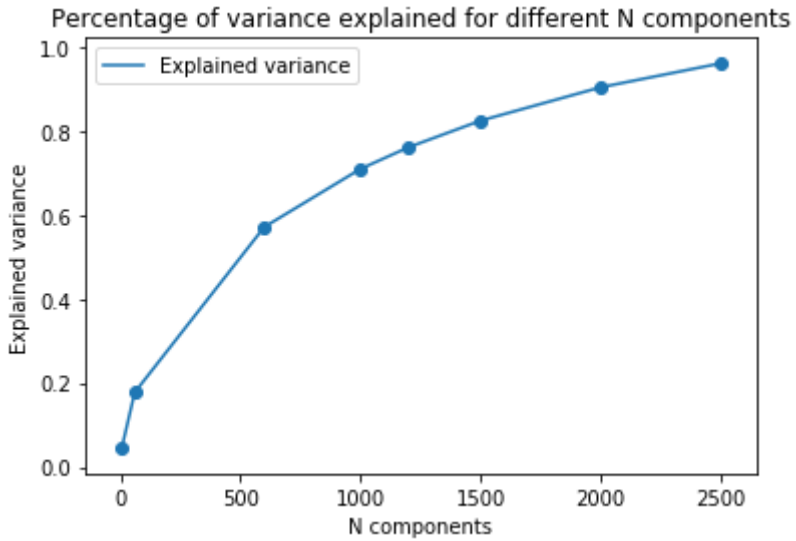
2.5 Support Vector Machines with added Features `Set 5`

In [117]:
```
from sklearn.decomposition import TruncatedSVD
def dimension_reduction(components):
    explained_variance=[]
    for component in tqdm(components):
        svd = TruncatedSVD(n_components=component, n_iter=7, random_state=42)
        svd.fit(tr_text_tfidf)
        explained_variance.append(svd.explained_variance_ratio_.sum())
    plt.plot(components,explained_variance,label="Explained variance")
    plt.scatter(components,explained_variance)
    plt.title("Percentage of variance explained for different N components")
    plt.xlabel("N components")
    plt.ylabel("Explained variance")
```

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

In [118]:
```
%%time
components=[6,60,600,1000,1200,1500,2000,2500]
dimension_reduction(components)
```

```
100%|██████████| 8/8 [19:10<00:00, 229.22s/it]
```




```
Wall time: 19min 10s
```

Reason for choosing 1000 components because from the above figure we can see that 70% of the variance explained if we choose 1000 components.

In [137]:
```
%%time
svd = TruncatedSVD(n_components=1000, n_iter=7, random_state=42)
tr_text_tfidf_new=svd.fit_transform(tr_text_tfidf)
cv_text_tfidf_new=svd.transform(cv_text_tfidf)
te_text_tfidf_new=svd.transform(te_text_tfidf)
print("Explained variance ",svd.explained_variance_ratio_.sum())
```

```
Explained variance  0.7094333966603623
Wall time: 1min 59s
```

In [138]:
```
%%time
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
#categorical, numerical features
```

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a den
se matirx :)
tr_X_num= hstack((tr_school_state_one_hot,tr_categories_one_hot,tr_sub_categor
ies_one_hot,tr_teacher_prefix_one_hot,tr_grade_category_one_hot,tr_price_stand
ardized,tr_teacher_number_of_previously_posted_projects_standardized,tr_num_pr
oject_standardized,tr_num_essay_standardized,tr_text_tfidf_new)).tocsr()
cv_X_num= hstack((cv_school_state_one_hot,cv_categories_one_hot,cv_sub_categor
ies_one_hot,cv_teacher_prefix_one_hot,cv_grade_category_one_hot,cv_price_stand
ardized,cv_teacher_number_of_previously_posted_projects_standardized,cv_num_pr
oject_standardized,cv_num_essay_standardized,cv_text_tfidf_new)).tocsr()
te_X_num= hstack((te_school_state_one_hot,te_categories_one_hot,te_sub_categor
ies_one_hot,te_teacher_prefix_one_hot,te_grade_category_one_hot,te_price_stand
ardized,te_teacher_number_of_previously_posted_projects_standardized,te_num_pr
oject_standardized,te_num_essay_standardized,te_text_tfidf_new)).tocsr()
tr_X_num=tr_X_num.toarray()
cv_X_num=cv_X_num.toarray()
te_X_num=te_X_num.toarray()
print(tr_X_num.shape)
print(cv_X_num.shape)
print(te_X_num.shape)
```

```
(49041, 1104)
(24155, 1104)
(36052, 1104)
Wall time: 8.77 s
```

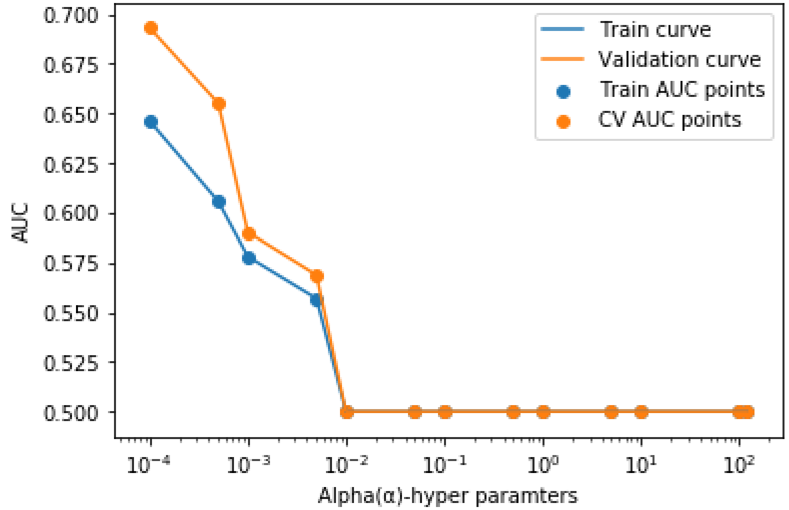
Applying SVM on Set 5

In [149]:

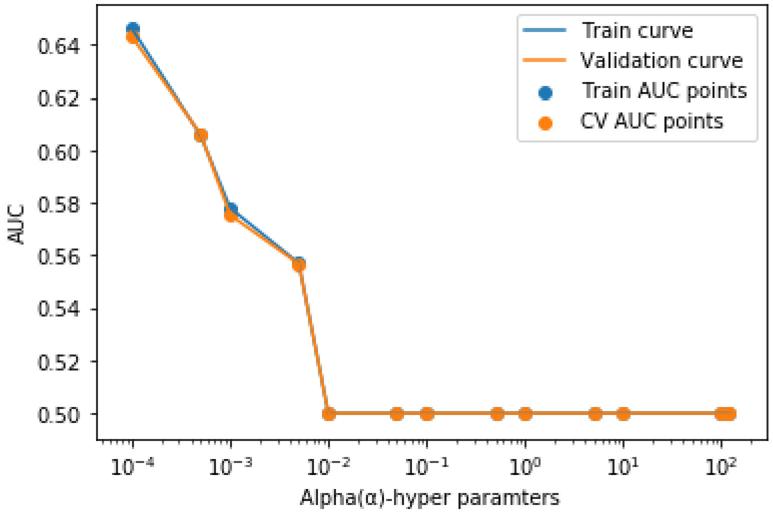
```
%%time
title=["Train AUC and Validation AUC for diffrent Alpha(α) with penalty L1 for
TFIDF W2V Vectorization","3-fold cross validation for diffrent Alpha(α with pe
nalty L1 for TFIDF W2V Vectorization","Train AUC and Validation AUC for diffre
nt Alpha(α) with penalty L2 for TFIDF W2V Vectorization","3-fold cross validat
ion for diffrent Alpha(α) with penalty L2 for TFIDF W2V Vectorization"]
roc_auc_compute(tr_X_num,Y_train,cv_X_num,Y_cv,[120,100,10,5,1,0.5,0.1,0.05,0.
01,0.005,0.001,0.0005,0.0001],title)
```

```
100%|[redacted]| 13/13 [04:24<00:00, 17.83s/it]
```

Train AUC and Validation AUC for diffrent Alpha(α) with penalty L1 for TFIDF W2V Vectorization

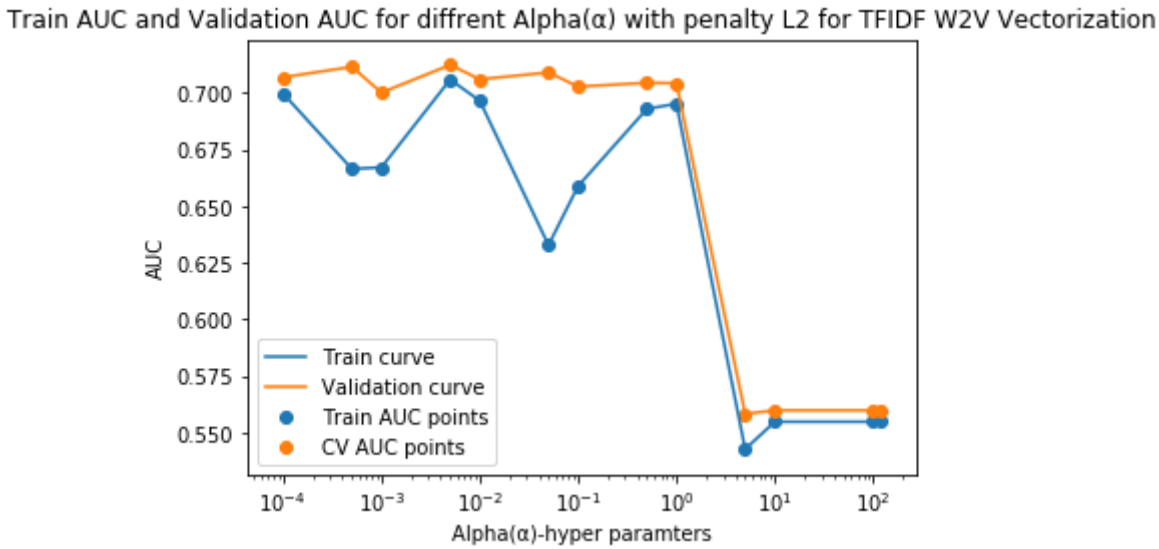


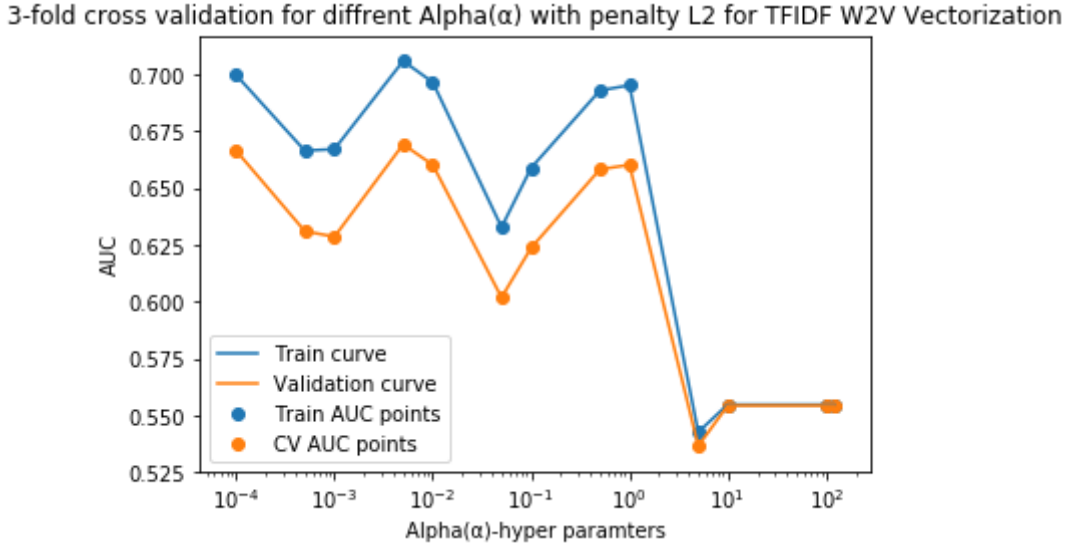
3-fold cross validation for diffrent Alpha(α with penalty L1 for TFIDF W2V Vectorization




```
best AUC [0.64303473]
best Alpha(α) 0.0001
```

```
100%|
| 13/13 [01:59<00:00,  8.70s/it]
```



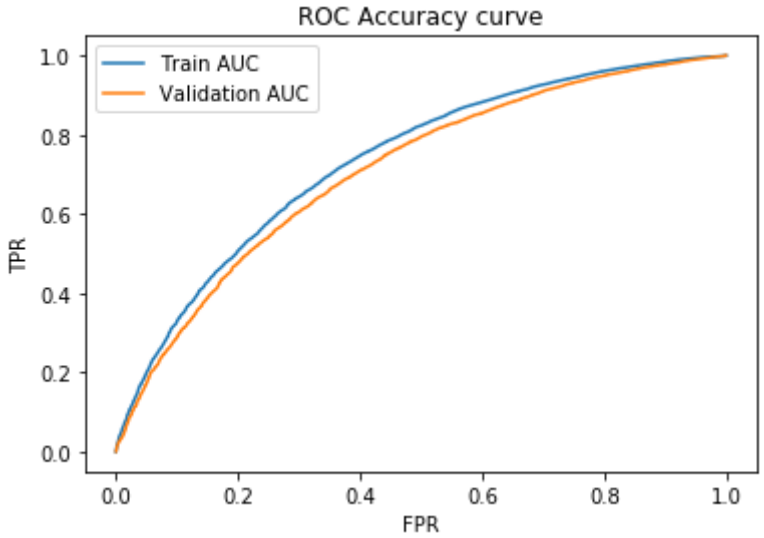



```
best AUC [0.66917125]
best Alpha(α) 0.005
Wall time: 6min 26s
```

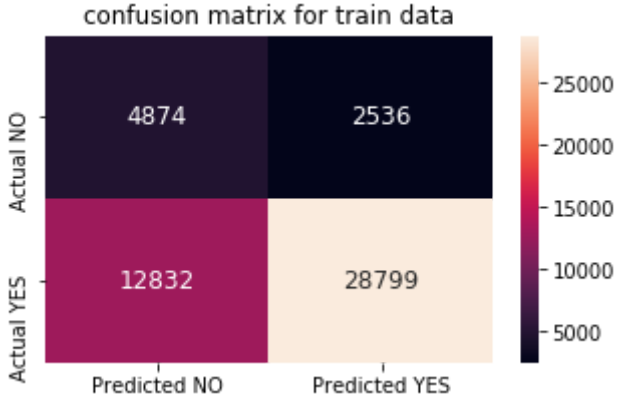
In [160]:

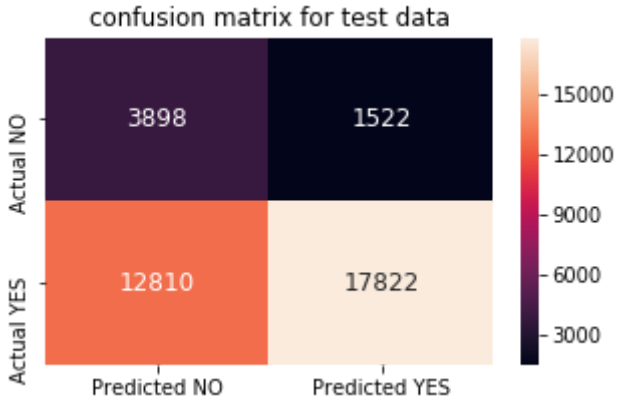
```
%%time
class_w={0:0.5,1:0.5}
#https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for
-classification-in-python/
from sklearn.metrics import roc_curve, auc
```

```
#training model with best lambda(λ) paramter
trained_SVM_new=SGDClassifier(alpha=0.0005,class_weight=class_w,penalty='l2')
#trainning model
trained_SVM_new.fit(tr_X_num,Y_train)
calibrated = CalibratedClassifierCV(trained_SVM_new, method='isotonic')
calibrated.fit(tr_X_num, Y_train)
# predict the response on the train data
predicted_labels_train=calibrated.predict_proba(tr_X_num)
# predict the response on the test data
predicted_labels_test=calibrated.predict_proba(te_X_num)
#Calculating FPR and TPR for train and test data
tr_fpr,tr_tpr,tr_threshold=roc_curve(Y_train,predicted_labels_train[:,1])
te_fpr,te_tpr,te_threshold=roc_curve(Y_test,predicted_labels_test[:,1])
#drawing ROC ROC Accuracy curve for test and train data
plt.plot(tr_fpr,tr_tpr,label="Train AUC")
plt.plot(te_fpr,te_tpr,label="Validation AUC")
plt.title("ROC Accuracy curve")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.legend()
plt.show()
print("Train AUC =",round(auc(tr_fpr,tr_tpr),2))
print("Test AUC =",round(auc(te_fpr,te_tpr),2))
#drawing confusion matrix for test and train data
t2="confusion matrix for train data"
draw_confusion_matrix(calibrated,tr_threshold,Y_train,predicted_labels_train
[:,1],tr_tpr,tr_fpr,t2)
t1="confusion matrix for test data"
draw_confusion_matrix(calibrated,tr_threshold,Y_test,predicted_labels_test[:,
1],te_tpr,te_fpr,t1)
```




```
Train AUC = 0.73
Test AUC = 0.71
Wall time: 3.77 s
```





3. Conclusions

In [161]:
```
from prettytable import PrettyTable

table = PrettyTable()

table.field_names = ["Vectorizer",  "Hyper parameter","Penalty", "Test AUC"]

table.add_row(["BOW",  "0.01","l2","0.71" ])
table.add_row(["TFIDF", "0.0005","l2", "0.71"])
table.add_row(["AVG W2V",  "0.0001","l1","0.70" ])
table.add_row(["TFIDF W2V",  "0.0001","l1","0.69" ])
table.add_row(["categorical,numerical and TFIDF-1000 features",  "0.0001","l1"
,"0.71" ])

print(table)
```

```
+-----------------------------------------------+-----------------+--------
-+----------+
|                   Vectorizer                  | Hyper parameter | Penalty
| Test AUC |
+-----------------------------------------------+-----------------+--------
-+----------+
|                      BOW                      |       0.01      |    l2
|   0.71   |
|                     TFIDF                     |      0.0005     |    l2
|   0.71   |
|                    AVG W2V                    |      0.0001     |    l1
|   0.70   |
|                   TFIDF W2V                   |      0.0001     |    l1
|   0.69   |
| categorical,numerical and TFIDF-1000 features |      0.0001     |    l1
|   0.71   |
+-----------------------------------------------+-----------------+--------
-+----------+
```