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**
`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 targete enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories f 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 ((https://en.wikipedia.org/wiki/List_of_U.S._state_abbr **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategori **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the projec • `My students need hands on literacy mate sensory needs!`

Feature	Description
`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. **Exa** `12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed pro `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated value • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted b **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]:

```
%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 chart_studio.plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

from collections import Counter
from prettytable import PrettyTable

from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import math
```

In [2]:

```
df1 = pd.read_csv('train_data.csv',nrows=50000)
df2 =pd.read_csv('resources.csv', nrows =50000)
```

In [3]:

```
print("Number of data points in train data", df1.shape)
print('-'*50)
print("The attributes of data :", df1.columns.values)
```

```
Number of data points in train data (50000, 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]:

```
print("Number of data points in train data", df2.shape)
print(df2.columns.values)
print(df2.head(2))
```

```
Number of data points in train data (50000, 4)
['id' 'description' 'quantity' 'price']
        id                                        description  quantity  \
0  p233245  LC652 - Lakeshore Double-Space Mobile Drying Rack         1
1  p069063        Bouncy Bands for Desks (Blue support pipes)         3

    price
0  149.00
1   14.95
```

In [5]:

```
df1.head()
```

Out[5]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_s
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX

In [6]:

```
# merge two column text dataframe:
df1["essay"] = df1["project_essay_1"].map(str) +\
                        df1["project_essay_2"].map(str) + \
                        df1["project_essay_3"].map(str) + \
                        df1["project_essay_4"].map(str)
```

preprocessing of project subject categories

In [7]:

```
categories = list(df1['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-stri
ng
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-pyth
on
cat_list = []
for i in categories:
    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", "Warmt
h", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "M
ath & 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 ''(empty) ex:"M
ath & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spa
ces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

df1['clean_categories'] = cat_list
df1.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in df1['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]))
sorted_cat_dict
```

Out[7]:

```
{'Warmth': 643,
 'Care_Hunger': 643,
 'History_Civics': 2689,
 'Music_Arts': 4699,
 'AppliedLearning': 5569,
 'SpecialNeeds': 6233,
 'Health_Sports': 6538,
 'Math_Science': 18874,
 'Literacy_Language': 23998}
```

preprocessing project subject subcategories

In [8]:

```
sub_catogories = list(df1['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

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

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", "Warmt
h", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "M
ath & 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 ''(empty) ex:"M
ath & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spa
ces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

df1['clean_subcategories'] = sub_cat_list
df1.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 df1['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]))
sorted_sub_cat_dict
```

Out[8]:

```
{'Economics': 127,
 'CommunityService': 214,
 'FinancialLiteracy': 253,
 'ParentInvolvement': 302,
 'Extracurricular': 373,
 'Civics_Government': 380,
 'ForeignLanguages': 388,
 'NutritionEducation': 617,
 'Warmth': 643,
 'Care_Hunger': 643,
 'SocialSciences': 864,
 'PerformingArts': 910,
 'CharacterEducation': 958,
 'TeamSports': 995,
 'Other': 1128,
 'College_CareerPrep': 1168,
 'Music': 1432,
 'History_Geography': 1433,
 'Health_LifeScience': 1876,
 'EarlyDevelopment': 1937,
 'ESL': 1999,
 'Gym_Fitness': 2068,
 'EnvironmentalScience': 2533,
 'VisualArts': 2865,
 'Health_Wellness': 4732,
 'AppliedSciences': 4901,
 'SpecialNeeds': 6233,
 'Literature_Writing': 10127,
 'Mathematics': 12832,
 'Literacy': 15611}
```

In [9]:

```
df1['teacher_prefix']= df1['teacher_prefix'].fillna(df1['teacher_prefix'].mode().iloc[0
])

prefix = list(df1['teacher_prefix'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

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

prefix_list = []
for i in prefix:
    temp = ""
    if "." in i:
        i = i.replace('.','')
        temp +=i.strip()+" "#" abc ".strip() will return "abc", remove the trailing spa
ces
    prefix_list.append(temp.strip())

df1['teachers_prefix'] = prefix_list
df1.drop(['teacher_prefix'],inplace = True,axis = 1)
```

In [10]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in df1['teachers_prefix'].values:
    my_counter.update(word.split())

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

Out[10]:

```
{'Dr': 2, 'Mr': 4859, 'Ms': 17936, 'Mrs': 26142}
```

preprocessing project grade category

In [11]:

```
grade_categories = list(df1['project_grade_category'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

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

grade_cat_list = []
for i in grade_categories:
    temp = ""
    if "-" in i:
        i = i.replace('-','_')
        i = i.replace(' ','_')
        temp +=i.strip()+' '#" abc ".strip() will return "abc", remove the trailing spa
ces
    grade_cat_list.append(temp.strip())

df1['grade_category'] = grade_cat_list
df1.drop(['project_grade_category'], axis=1, inplace=True)
```

In [12]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in df1['grade_category'].values:
    my_counter.update(word.split())

grade_dict = dict(my_counter)
sorted_grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))

sorted_grade_dict
```

Out[12]:

```
{'Grades_9_12': 4966,
 'Grades_6_8': 7750,
 'Grades_3_5': 16968,
 'Grades_PreK_2': 20316}
```

train test splitting

In [13]:

```
y = df1['project_is_approved'].values
X = df1.drop(['Unnamed: 0','teacher_id','project_submitted_datetime'], axis=1)
X.head(1)
```

Out[13]:

	id	school_state	project_title	project_essay_1	project_essay_2	project_ess
0	p253737	IN	Educational Support for English Learners at Home	My students are English learners that are work...	\"The limits of your language are the limits o...	NaN

In [14]:

```
from sklearn.model_selection import train_test_split

# split the data into test and train by maintaining same distribution of output varaibl
e 'y' [stratify=y]
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2)
# split the train data into train and cross validation by maintaining same distribution
of output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, tes
t_size=0.2)
```

In [15]:

```
# 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 [16]:

```
sent = decontracted(df1['essay'].values[20000])
print(sent)
print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and
language delays, cognitive delays, gross/fine motor delays, to autism. The
y are eager beavers and always strive to work their hardest working past t
heir limitations. \r\n\r\nThe materials we have are the ones I seek out fo
r my students. I teach in a Title I school where most of the students rece
ive free or reduced price lunch.  Despite their disabilities and limitatio
ns, my students love coming to school and come eager to learn and explore.
Have you ever felt like you had ants in your pants and you needed to groov
e and move as you were in a meeting? This is how my kids feel all the tim
e. The want to be able to move as they learn or so they say.Wobble chairs
are the answer and I love then because they develop their core, which enha
nces gross motor and in Turn fine motor skills. \r\nThey also want to lear
n through games, my kids do not want to sit and do worksheets. They want t
o learn to count by jumping and playing. Physical engagement is the key to
our success. The number toss and color and shape mats can make that happe
n. My students will forget they are doing work and just have the fun a 6 y
ear old deserves.nannan
==================================================
```

In [17]:

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

```
My kindergarten students have varied disabilities ranging from speech and
language delays, cognitive delays, gross/fine motor delays, to autism. The
y are eager beavers and always strive to work their hardest working past t
heir limitations.     The materials we have are the ones I seek out for my
students. I teach in a Title I school where most of the students receive f
ree or reduced price lunch.  Despite their disabilities and limitations, m
y students love coming to school and come eager to learn and explore.Have
you ever felt like you had ants in your pants and you needed to groove and
move as you were in a meeting? This is how my kids feel all the time. The
want to be able to move as they learn or so they say.Wobble chairs are the
answer and I love then because they develop their core, which enhances gro
ss motor and in Turn fine motor skills.   They also want to learn through
games, my kids do not want to sit and do worksheets. They want to learn to
count by jumping and playing. Physical engagement is the key to our succes
s. The number toss and color and shape mats can make that happen. My stude
nts will forget they are doing work and just have the fun a 6 year old des
erves.nannan
```

In [18]:

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

```
My kindergarten students have varied disabilities ranging from speech and
language delays cognitive delays gross fine motor delays to autism They ar
e eager beavers and always strive to work their hardest working past their
limitations The materials we have are the ones I seek out for my students
I teach in a Title I school where most of the students receive free or red
uced price lunch Despite their disabilities and limitations my students lo
ve coming to school and come eager to learn and explore Have you ever felt
like you had ants in your pants and you needed to groove and move as you w
ere in a meeting This is how my kids feel all the time The want to be able
to move as they learn or so they say Wobble chairs are the answer and I lo
ve then because they develop their core which enhances gross motor and in
Turn fine motor skills They also want to learn through games my kids do no
t want to sit and do worksheets They want to learn to count by jumping and
playing Physical engagement is the key to our success The number toss and
color and shape mats can make that happen My students will forget they are
doing work and just have the fun a 6 year old deserves nannan
```

In [19]:

```
# 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'r
e", "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', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', '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", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [20]:

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

```
100%|██████████████████████████████████████████████████████████████████████
██████| 32000/32000 [00:35<00:00, 909.61it/s]
```

In [21]:

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

```
100%|█████████████████████████████████████████████████████████████████████
████████| 8000/8000 [00:08<00:00, 943.94it/s]
```

In [22]:

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

```
100%|█████████████████████████████████████████████████████████████████████
██████| 10000/10000 [00:12<00:00, 829.94it/s]
```

In [23]:

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

```
100%|█████████████████████████████████████████████████████████████████████
████| 32000/32000 [00:01<00:00, 23100.99it/s]
```

In [24]:

```
preprocessed_cv_title = []
# tqdm is for printing the status bar
for sentance in tqdm(X_cv['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_cv_title.append(sent.lower().strip())
```

```
100%|█████████████████████████████████████████████████████████████████████
██████| 8000/8000 [00:00<00:00, 25821.45it/s]
```

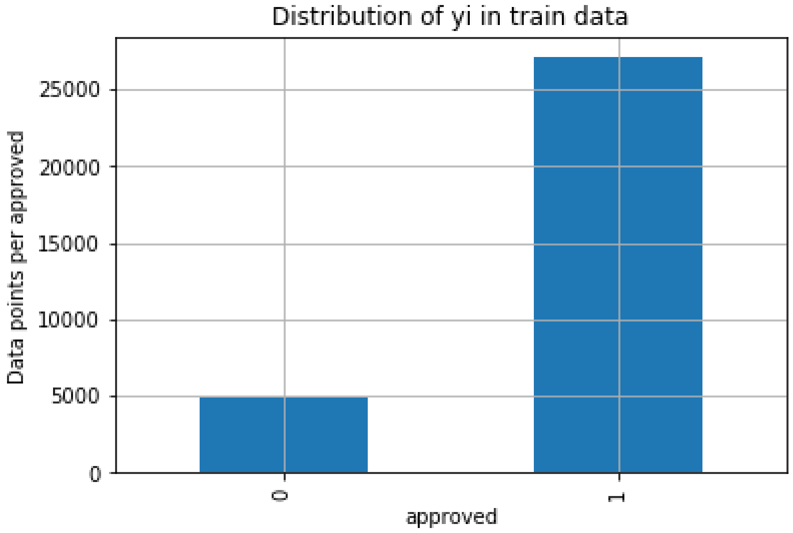
In [25]:

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

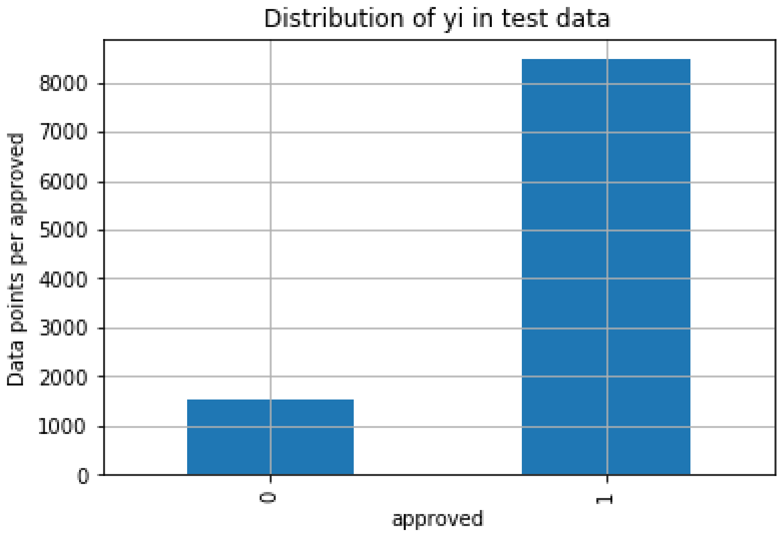
```
100%|█████████████████████████████████████████████████████████████████████
████| 10000/10000 [00:00<00:00, 25422.74it/s]
```

In [26]:

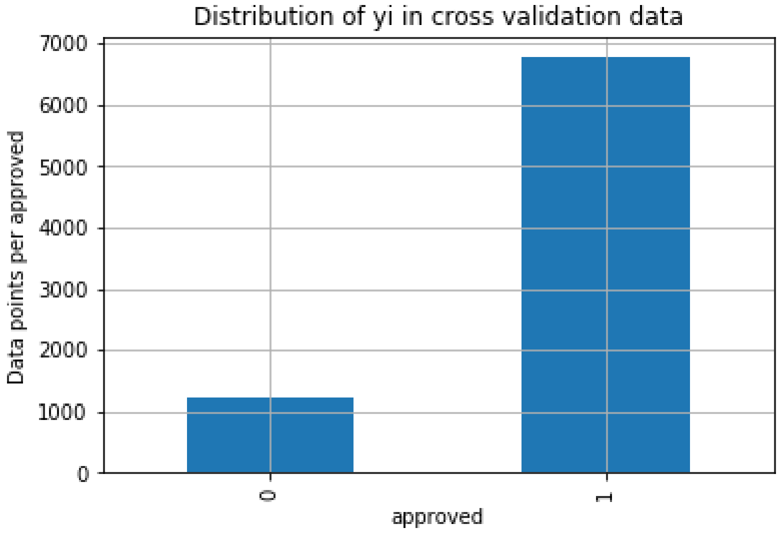
```
# it returns a dict, keys as aprvd labels and values as the number of data points in th
at aprvd
train_aprvd_distribution = X_train['project_is_approved'].value_counts().sort_index()
test_aprvd_distribution = X_test['project_is_approved'].value_counts().sort_index()
cv_aprvd_distribution = X_cv['project_is_approved'].value_counts().sort_index()

my_colors = 'rgbkymc'
train_aprvd_distribution.plot(kind='bar')
plt.xlabel('approved')
plt.ylabel('Data points per approved')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_aprvd_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_aprvd_distribution.values)
for i in sorted_yi:
    print('Number of data points in approved', i+1, ':',train_aprvd_distribution.values
[i], '(', np.round((train_aprvd_distribution.values[i]/X_train.shape[0]*100), 3), '%)')


print('-'*80)
my_colors = 'rgbkymc'
test_aprvd_distribution.plot(kind='bar')
plt.xlabel('approved')
plt.ylabel('Data points per approved')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_aprvd_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_aprvd_distribution.values)
for i in sorted_yi:
    print('Number of data points in approved', i+1, ':',test_aprvd_distribution.values[
i], '(', np.round((test_aprvd_distribution.values[i]/X_test.shape[0]*100), 3), '%)')

print('-'*80)
my_colors = 'rgbkymc'
cv_aprvd_distribution.plot(kind='bar')
plt.xlabel('approved')
plt.ylabel('Data points per approved')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_aprvd_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_aprvd_distribution.values)
for i in sorted_yi:
    print('Number of data points in approved', i+1, ':',cv_aprvd_distribution.values[i
], '(', np.round((cv_aprvd_distribution.values[i]/X_cv.shape[0]*100), 3), '%)')
```



```
Number of data points in approved 2 : 27063 ( 84.572 %)
Number of data points in approved 1 : 4937 ( 15.428 %)
--------------------------------------------------------------------------
------
```



```
Number of data points in approved 2 : 8457 ( 84.57 %)
Number of data points in approved 1 : 1543 ( 15.43 %)
--------------------------------------------------------------------------
------
```



```
Number of data points in approved 2 : 6766 ( 84.575 %)
Number of data points in approved 1 : 1234 ( 15.425 %)
```

Response encoding

for school state

In [27]:

```
def get_project_fea_dict(alpha, feature, df):

    value_count = X_train[feature].value_counts()
    project_dict = dict()

    # denominator will contain the number of time that particular feature occured in wh
ole data
    for i, denominator in value_count.items():
        vec = []
        for k in range(0,2):
            aprvd_cnt = X_train.loc[(X_train['project_is_approved']==k) & (X_train[feat
ure]==i)]

            vec.append((aprvd_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

        project_dict[i]=vec
    return project_dict

def get_project_feature(alpha, feature, df):
    project_dict = get_project_fea_dict(alpha, feature, df)
    value_count = X_train[feature].value_counts()
    project_fea = []
    # for every feature values in the given data frame we will check if it is there in
 the train data then we will add the feature to project_fea
    # if not we will add [1/2,1/2] to project_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            project_fea.append(project_dict[row[feature]])
        else:
            project_fea.append([1/2,1/2])
    return project_fea
```

In [28]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
from collections import Counter
my_counter = Counter()
for word in df1['school_state'].values:
    my_counter.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
school_dict = dict(my_counter)
sorted_school_dict = dict(sorted(school_dict.items(), key=lambda kv: kv[1]))
sorted_school_dict
```

Out[28]:

```
{'VT': 32,
 'WY': 51,
 'ND': 63,
 'MT': 106,
 'RI': 126,
 'NH': 141,
 'SD': 142,
 'NE': 144,
 'AK': 153,
 'DE': 155,
 'WV': 218,
 'ME': 222,
 'NM': 236,
 'HI': 239,
 'DC': 247,
 'KS': 285,
 'ID': 302,
 'IA': 306,
 'AR': 446,
 'CO': 538,
 'MN': 556,
 'OR': 577,
 'MS': 598,
 'KY': 614,
 'NV': 665,
 'MD': 668,
 'CT': 774,
 'TN': 774,
 'AL': 790,
 'UT': 792,
 'WI': 833,
 'VA': 916,
 'AZ': 994,
 'NJ': 1005,
 'OK': 1074,
 'MA': 1076,
 'LA': 1094,
 'WA': 1103,
 'MO': 1166,
 'IN': 1171,
 'OH': 1180,
 'PA': 1419,
 'MI': 1468,
 'GA': 1828,
 'SC': 1830,
 'IL': 1967,
 'NC': 2340,
 'FL': 2839,
 'TX': 3320,
 'NY': 3393,
 'CA': 7024}
```

In [29]:

```
#response-coding of the feature
# alpha is used for laplace smoothing
alpha = 1
# train feature
train_state_responseCoding = np.array(get_project_feature(alpha, "school_state", X_trai
n))
# test feature
test_state_responseCoding = np.array(get_project_feature(alpha, "school_state", X_test
))
# cross validation feature
cv_state_responseCoding = np.array(get_project_feature(alpha, "school_state", X_cv))
```

In [30]:

```
print("Shape of X_train after response encodig ",train_state_responseCoding.shape, y_tr
ain.shape)
print("Shape of X_cv after response encodig ",cv_state_responseCoding.shape, y_cv.shape
)
print("Shape of X_test after response encodig ",test_state_responseCoding.shape, y_test
.shape)
```

```
Shape of X_train after response encodig  (32000, 2) (32000,)
Shape of X_cv after response encodig  (8000, 2) (8000,)
Shape of X_test after response encodig  (10000, 2) (10000,)
```

for teacher_prefix

In [31]:

```
#response-coding of the feature
# alpha is used for laplace smoothing
alpha = 1
# train feature
train_teacher_responseCoding = np.array(get_project_feature(alpha, "teachers_prefix", X
_train))
# test feature
test_teacher_responseCoding = np.array(get_project_feature(alpha, "teachers_prefix", X_
test))
# cross validation feature
cv_teacher_responseCoding = np.array(get_project_feature(alpha, "teachers_prefix", X_cv
))
```

In [32]:

```
print("Shape of X_train after response encodig ",train_teacher_responseCoding.shape, y_
train.shape)
print("Shape of X_cv after response encodig ",cv_teacher_responseCoding.shape, y_cv.sha
pe)
print("Shape of X_test after response encodig ",test_teacher_responseCoding.shape, y_te
st.shape)
```

```
Shape of X_train after response encodig  (32000, 2) (32000,)
Shape of X_cv after response encodig  (8000, 2) (8000,)
Shape of X_test after response encodig  (10000, 2) (10000,)
```

for project grade category

In [33]:

```
#response-coding of the feature
# alpha is used for laplace smoothing
alpha = 1
# train feature
train_grade_responseCoding = np.array(get_project_feature(alpha, "grade_category", X_tr
ain))
# test feature
test_grade_responseCoding = np.array(get_project_feature(alpha, "grade_category", X_tes
t))
# cross validation feature
cv_grade_responseCoding = np.array(get_project_feature(alpha, "grade_category", X_cv))
```

In [34]:

```
print("Shape of X_train after response encodig ",train_grade_responseCoding.shape, y_tr
ain.shape)
print("Shape of X_cv after response encodig ",cv_grade_responseCoding.shape, y_cv.shape
)
print("Shape of X_test after response encodig ",test_grade_responseCoding.shape, y_test
.shape)
```

```
Shape of X_train after response encodig  (32000, 2) (32000,)
Shape of X_cv after response encodig  (8000, 2) (8000,)
Shape of X_test after response encodig  (10000, 2) (10000,)
```

for project subject categories

In [35]:

```
#response-coding of the feature
# alpha is used for laplace smoothing
alpha = 1
# train feature
train_category_responseCoding = np.array(get_project_feature(alpha, "clean_categories",
X_train))
# test feature
test_category_responseCoding = np.array(get_project_feature(alpha, "clean_categories",
X_test))
# cross validation feature
cv_category_responseCoding = np.array(get_project_feature(alpha, "clean_categories", X_
cv))
```

In [36]:

```
print("Shape of X_train after response encodig ",train_category_responseCoding.shape, y
_train.shape)
print("Shape of X_cv after response encodig ",cv_category_responseCoding.shape, y_cv.sh
ape)
print("Shape of X_test after response encodig ",test_category_responseCoding.shape, y_t
est.shape)
```

```
Shape of X_train after response encodig  (32000, 2) (32000,)
Shape of X_cv after response encodig  (8000, 2) (8000,)
Shape of X_test after response encodig  (10000, 2) (10000,)
```

for project categories

In [37]:

```
#response-coding of the feature
# alpha is used for laplace smoothing
alpha = 1
# train feature
train_subcategory_responseCoding = np.array(get_project_feature(alpha, "clean_subcatego
ries", X_train))
# test feature
test_subcategory_responseCoding = np.array(get_project_feature(alpha, "clean_subcategor
ies", X_test))
# cross validation feature
cv_subcategory_responseCoding = np.array(get_project_feature(alpha, "clean_subcategorie
s", X_cv))
```

In [38]:

```
print("Shape of X_train after response encodig ",train_subcategory_responseCoding.shape
, y_train.shape)
print("Shape of X_cv after response encodig ",cv_subcategory_responseCoding.shape, y_cv
.shape)
print("Shape of X_test after response encodig ",test_subcategory_responseCoding.shape,
y_test.shape)
```

```
Shape of X_train after response encodig  (32000, 2) (32000,)
Shape of X_cv after response encodig  (8000, 2) (8000,)
Shape of X_test after response encodig  (10000, 2) (10000,)
```

bag of words on essay

In [39]:

```
vec1 = CountVectorizer(min_df=10)
 # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vec1.fit_transform(preprocessed_train_essay)
X_cv_essay_bow = vec1.transform(preprocessed_cv_essay)
X_test_essay_bow = vec1.transform(preprocessed_test_essay)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
After vectorizations
(32000, 10155) (32000,)
(8000, 10155) (8000,)
(10000, 10155) (10000,)
```

bag of words on title

In [40]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vec2 = CountVectorizer(min_df=10)

# fit has to happen only on train data
# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vec2.fit_transform(preprocessed_train_title)
X_cv_title_bow = vec2.transform(preprocessed_cv_title)
X_test_title_bow = vec2.transform(preprocessed_test_title)

print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
```

```
After vectorizations
(32000, 1494) (32000,)
(8000, 1494) (8000,)
(10000, 1494) (10000,)
```

tfidf on essay

In [41]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vec3 = TfidfVectorizer(min_df=10, max_features =5000)

# fit has to happen only on train data
X_train_ess_tfidf = vec3.fit_transform(preprocessed_train_essay)

# we use the fitted Tfidf Vectorizer to convert the text to vector
X_cv_ess_tfidf = vec3.transform(preprocessed_cv_essay)
X_test_ess_tfidf = vec3.transform(preprocessed_test_essay)

print("After vectorizations")
print(X_train_ess_tfidf.shape, y_train.shape)
print(X_cv_ess_tfidf.shape, y_cv.shape)
print(X_test_ess_tfidf.shape, y_test.shape)
```

```
After vectorizations
(32000, 5000) (32000,)
(8000, 5000) (8000,)
(10000, 5000) (10000,)
```

tfidf on title

In [42]:

```
vec4 = TfidfVectorizer(min_df=10)
X_train_title_tfidf = vec4.fit_transform(preprocessed_train_title)

# we use the fitted Tfidf Vectorizer to convert the text to vector
X_cv_title_tfidf = vec4.transform(preprocessed_cv_title)
X_test_title_tfidf = vec4.transform(preprocessed_test_title)

print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
```

```
After vectorizations
(32000, 1494) (32000,)
(8000, 1494) (8000,)
(10000, 1494) (10000,)
```

Numerical features

In [43]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-al
l-groups-in-one-step
price_data = df2.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)
```

Out[43]:

	id	price	quantity
0	p000027	782.13	15
1	p000052	114.98	2

In [44]:

```
# join two dataframes in python:
df1 = pd.merge(df1, price_data, on='id', how='left')
```

In [45]:

```
# https://stackoverflow.com/questions/32617811/imputation-of-missing-values-for-categor
ies-in-pandas
#replacing nan with most frequently occuring element
df1['price'] = df1['price'].fillna(df1['price'].mode().iloc[0])
```

In [46]:

```
X_train =pd.merge(X_train,price_data, how ='left', on = 'id')
X_cv =pd.merge(X_cv,price_data, how ='left', on ='id')
X_test = pd.merge(X_test,price_data,how ='left',on='id')
```

In [47]:

```
# https://stackoverflow.com/questions/32617811/imputation-of-missing-values-for-categor
ies-in-pandas
#replacing nan with most frequently occuring element
X_train['price'] = X_train['price'].fillna(X_train['price'].mode().iloc[0])
X_cv['price'] = X_cv['price'].fillna(X_train['price'].mode().iloc[0])
X_test['price'] = X_test['price'].fillna(X_test['price'].mode().iloc[0])
print(X_train['price'].isnull().sum())
print(X_cv['price'].isnull().sum())
print(X_test['price'].isnull().sum())
```

```
0
0
0
```

In [48]:

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

# price_standardized = standardScalar.fit(df1['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ...
 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()

X_train_price_stndrd = price_scalar.fit_transform(X_train['price'].values.reshape(-1,1
))
X_cv_price_stndrd = price_scalar.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_stndrd = price_scalar.transform(X_test['price'].values.reshape(-1,1))
```

In [49]:

```
X_train_price_stndrd
```

Out[49]:

```
array([[-0.07294389],
       [-0.07294389],
       [-0.07294389],
       ...,
       [-0.07294389],
       [-0.07294389],
       [-0.07294389]])
```

In [50]:

```
X_cv_price_stndrd
```

Out[50]:

```
array([[-0.07294389],
       [-0.07294389],
       [-0.07294389],
       ...,
       [-0.07294389],
       [-0.07294389],
       [-0.07294389]])
```

In [51]:

```
X_test_price_stndrd
```

Out[51]:

```
array([[2.69872935],
       [2.69872935],
       [2.69872935],
       ...,
       [2.69872935],
       [2.69872935],
       [2.69872935]])
```

In [52]:

```
df1['quantity'] = df1['quantity'].fillna(df1['quantity'].mode().iloc[0])
```

In [53]:

```
# https://stackoverflow.com/questions/32617811/imputation-of-missing-values-for-categor
ies-in-pandas
#replacing nan with most frequently occuring element

X_train['quantity'] = X_train['quantity'].fillna(X_train['quantity'].mode().iloc[0])
X_cv['quantity'] = X_cv['quantity'].fillna(X_cv['quantity'].mode().iloc[0])
X_test['quantity'] = X_test['quantity'].fillna(X_test['quantity'].mode().iloc[0])
```

In [54]:

```
# quantity
quantity_scalar = StandardScaler()
X_train_quantity_stndrd = quantity_scalar.fit_transform(X_train['quantity'].values.resh
ape(-1,1))

X_cv_quantity_stndrd = quantity_scalar.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity_stndrd = quantity_scalar.transform(X_test['quantity'].values.reshape(-1
,1))
```

In [55]:

```
X_train_quantity_stndrd
```

Out[55]:

```
array([[-0.07618667],
       [-0.07618667],
       [-0.07618667],
       ...,
       [-0.07618667],
       [-0.07618667],
       [-0.07618667]])
```

In [56]:

```
X_cv_quantity_stndrd
```

Out[56]:

```
array([[-0.07618667],
       [-0.07618667],
       [-0.07618667],
       ...,
       [-0.07618667],
       [-0.07618667],
       [-0.07618667]])
```

In [57]:

```
X_test_quantity_stndrd
```

Out[57]:

```
array([[0.05542494],
       [0.05542494],
       [0.05542494],
       ...,
       [0.05542494],
       [0.05542494],
       [0.05542494]])
```

In [58]:

```
# previous_year_projects
# finding the mean and standard deviation of this data
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape
(-1,1))
X_train_prev_proj_stndrd =price_scalar.transform(X_train['teacher_number_of_previously_
posted_projects'].values.reshape(-1,1))

X_test_prev_proj_stndrd =price_scalar.transform(X_test['teacher_number_of_previously_po
sted_projects'].values.reshape(-1, 1))
X_cv_prev_proj_stndrd = price_scalar.transform(X_cv['teacher_number_of_previously_poste
d_projects'].values.reshape(-1, 1))
```

In [59]:

```
X_train_prev_proj_stndrd
```

Out[59]:

```
array([[-0.07792049],
       [-0.40075831],
       [ 0.28078819],
       ...,
       [-0.36488744],
       [-0.40075831],
       [-0.40075831]])
```

In [60]:

```
X_cv_prev_proj_stndrd
```

Out[60]:

```
array([[ 0.78298035],
       [-0.1855331 ],
       [-0.25727484],
       ...,
       [-0.32901658],
       [-0.22140397],
       [-0.25727484]])
```

In [61]:

```
X_test_prev_proj_stndrd
```

Out[61]:

```
array([[-0.1855331 ],
       [-0.25727484],
       [-0.40075831],
       ...,
       [ 0.24491732],
       [-0.40075831],
       [ 4.47767983]])
```

Concatinating all

In [62]:

```
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_title_bow,X_train_essay_bow,train_category_responseCoding
,train_subcategory_responseCoding,
                      train_grade_responseCoding,train_teacher_responseCoding,train_sta
te_responseCoding,
                      X_train_price_stndrd, X_train_quantity_stndrd, X_train_prev_proj_
stndrd))


print(X_train_bow.shape, y_train.shape)
```

```
(32000, 11662) (32000,)
```

In [63]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_cv_bow = hstack((X_cv_title_bow,X_cv_essay_bow,cv_category_responseCoding,cv_subcateg
ory_responseCoding,
                      cv_grade_responseCoding,cv_teacher_responseCoding,cv_state_respon
seCoding,
                      X_cv_price_stndrd, X_cv_quantity_stndrd, X_cv_prev_proj_stndrd))

print(X_cv_bow.shape, y_cv.shape)
```

```
(8000, 11662) (8000,)
```

In [64]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_test_bow = hstack((X_test_title_bow,X_test_essay_bow,test_category_responseCoding,tes
t_subcategory_responseCoding,
                      test_grade_responseCoding,test_teacher_responseCoding,test_state_
responseCoding,
                      X_test_price_stndrd, X_test_quantity_stndrd, X_test_prev_proj_stn
drd))

print(X_test_bow.shape, y_test.shape)
```

```
(10000, 11662) (10000,)
```

In [65]:

```
X_train_bow = X_train_bow.tocsr()
X_cv_bow = X_cv_bow.tocsr()
X_test_bow = X_test_bow.tocsr()
```

In [66]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_train_tfidf = hstack((X_train_title_tfidf,X_train_ess_tfidf,train_category_responseCo
ding,train_subcategory_responseCoding,
                         train_grade_responseCoding,train_teacher_responseCoding,train_
state_responseCoding,
                         X_train_price_stndrd,X_train_quantity_stndrd, X_train_prev_pro
j_stndrd))


print(X_train_tfidf.shape, y_train.shape)
```

```
(32000, 6507) (32000,)
```

In [67]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_cv_tfidf = hstack((X_cv_title_tfidf,X_cv_ess_tfidf,cv_category_responseCoding,cv_subc
ategory_responseCoding,
                         cv_grade_responseCoding,cv_teacher_responseCoding,cv_state_res
ponseCoding,
                         X_cv_price_stndrd,X_cv_quantity_stndrd, X_cv_prev_proj_stndrd
))

print(X_cv_tfidf.shape, y_cv.shape)
```

```
(8000, 6507) (8000,)
```

In [68]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_test_tfidf = hstack((X_test_title_tfidf,X_test_ess_tfidf,test_category_responseCoding
,test_subcategory_responseCoding,
                         test_grade_responseCoding,test_teacher_responseCoding,test_sta
te_responseCoding,
                         X_test_price_stndrd,X_test_quantity_stndrd, X_test_prev_proj_s
tndrd))

print(X_test_tfidf.shape, y_test.shape)
```

```
(10000, 6507) (10000,)
```

In [69]:

```
X_train_tfidf = X_train_tfidf.tocsr()
X_cv_tfidf = X_cv_tfidf.tocsr()
X_test_tfidf = X_test_tfidf.tocsr()
```

Pretrained avg_w2v model

In [70]:

```
# 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
```

In [71]:

```
model = loadGloveModel('glove.42B.300d.txt')
```

```
0it [00:00, ?it/s]

Loading Glove Model

1917494it [07:47, 4105.22it/s]

Done. 1917494  words loaded!
```

In [72]:

```
glove_words = set(model.keys())
```

In [73]:

```
# compute average word2vec for each review.
def func(wordlist):



    train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in thi
s list
    for sentence in tqdm(wordlist): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length    # we are taking
 the 300 dimensions  very large
        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
        train_avg_w2v_vectors.append(vector)

    print(len(train_avg_w2v_vectors))
    print(len(train_avg_w2v_vectors[0]))
    return train_avg_w2v_vectors
```

In [74]:

```
# FOR ESSAYS
X_train_avg_w2v_ess=func(preprocessed_train_essay)
X_cv_avg_w2v_ess=func(preprocessed_cv_essay)
X_test_avg_w2v_ess=func(preprocessed_test_essay)
```

```
100%|██████████████████████████████████████████████████████████████████████
█████| 32000/32000 [00:13<00:00, 2307.13it/s]
  6%|█████
| 502/8000 [00:00<00:02, 2501.91it/s]

32000
300

100%|██████████████████████████████████████████████████████████████████████
███████| 8000/8000 [00:03<00:00, 2415.39it/s]
  7%|██████
| 715/10000 [00:00<00:03, 2347.09it/s]

8000
300

100%|██████████████████████████████████████████████████████████████████████
█████| 10000/10000 [00:04<00:00, 2422.11it/s]

10000
300
```

In [75]:

```
X_train_avg_w2v_ess=np.array(X_train_avg_w2v_ess)
X_cv_avg_w2v_ess=np.array(X_cv_avg_w2v_ess)
X_test_avg_w2v_ess =np.array(X_test_avg_w2v_ess)
```

In [76]:

```
# FOR TITLES
X_train_avg_w2v_title=func(preprocessed_train_title)
X_cv_avg_w2v_title=func(preprocessed_cv_title)
X_test_avg_w2v_title=func(preprocessed_test_title)
```

```
100%|█████████████████████████████████████████████████████████████████████
████| 32000/32000 [00:00<00:00, 42892.53it/s]
100%|█████████████████████████████████████████████████████████████████████
██████| 8000/8000 [00:00<00:00, 46539.26it/s]
  0%|
| 0/10000 [00:00<?, ?it/s]

32000
300
8000
300

100%|█████████████████████████████████████████████████████████████████████
████| 10000/10000 [00:00<00:00, 44028.49it/s]

10000
300
```

In [77]:

```
X_train_avg_w2v_title=np.array(X_train_avg_w2v_title)
X_cv_avg_w2v_title=np.array(X_cv_avg_w2v_title)
X_test_avg_w2v_title =np.array(X_test_avg_w2v_title)
```

In [78]:

```
X_train_avg_w2v = np.hstack((X_train_avg_w2v_ess,X_train_avg_w2v_title,train_category_r
esponseCoding,train_subcategory_responseCoding,
                            train_grade_responseCoding,train_teacher_responseCoding,tra
in_state_responseCoding,
                            X_train_price_stndrd, X_train_quantity_stndrd, X_train_prev
_proj_stndrd ))
```

In [79]:

```
X_cv_avg_w2v = np.hstack((X_cv_avg_w2v_ess,X_cv_avg_w2v_title,cv_category_responseCodin
g,cv_subcategory_responseCoding,
                            cv_grade_responseCoding,cv_teacher_responseCoding,cv_state_
responseCoding,
                            X_cv_price_stndrd, X_cv_quantity_stndrd, X_cv_prev_proj_stn
drd ))
```

In [80]:

```
X_test_avg_w2v = np.hstack((X_test_avg_w2v_ess,X_test_avg_w2v_title,test_category_respo
nseCoding,test_subcategory_responseCoding,
                            test_grade_responseCoding,test_teacher_responseCoding,test_
state_responseCoding,
                            X_test_price_stndrd, X_test_quantity_stndrd, X_test_prev_pr
oj_stndrd ))
```

In [81]:

```
print(X_train_avg_w2v.shape, y_train.shape)
print(X_cv_avg_w2v.shape, y_cv.shape)
print(X_test_avg_w2v.shape, y_test.shape)
```

```
(32000, 613) (32000,)
(8000, 613) (8000,)
(10000, 613) (10000,)
```

In [231]:

```
X_train_avg_w2v = X_train_avg_w2v[0:20000]
X_cv_avg_w2v = X_cv_avg_w2v[0:20000]
X_test_avg_w2v = X_test_avg_w2v[0:20000]
```

In [83]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_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 [84]:

```
def func(wordlist):
    # average Word2Vec
# compute average word2vec for each review.
    tfidf_w2v_ess = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(wordlist): # 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_ess.append(vector)

    print(len(tfidf_w2v_ess))
    print(len(tfidf_w2v_ess[0]))
    return tfidf_w2v_ess
```

In [85]:

```
#For essays
X_train_tfidf_w2v_ess =func(preprocessed_train_essay)
X_test_tfidf_w2v_ess =func(preprocessed_test_essay)
X_cv_tfidf_w2v_ess =func(preprocessed_cv_essay)
```

```
100%|█████████████████████████████████████████████████████████████████████
██████| 32000/32000 [01:44<00:00, 306.97it/s]
  2%|█
| 164/10000 [00:00<00:45, 217.89it/s]

32000
300

100%|█████████████████████████████████████████████████████████████████████
██████| 10000/10000 [00:33<00:00, 299.01it/s]
  4%|███
| 290/8000 [00:01<00:30, 249.92it/s]

10000
300

100%|█████████████████████████████████████████████████████████████████████
████████| 8000/8000 [00:26<00:00, 303.46it/s]

8000
300
```

In [86]:

```
X_train_tfidf_w2v_ess =np.array(X_train_tfidf_w2v_ess)
X_test_tfidf_w2v_ess = np.array(X_test_tfidf_w2v_ess)
X_cv_tfidf_w2v_ess = np.array(X_cv_tfidf_w2v_ess)
```

In [87]:

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

In [88]:

```
#For essays
X_train_tfidf_w2v_title =func(preprocessed_train_title)
X_test_tfidf_w2v_title =func(preprocessed_test_title)
X_cv_tfidf_w2v_title =func(preprocessed_cv_title)
```

```
100%|█████████████████████████████████████████████████████████████████████
████| 32000/32000 [00:01<00:00, 17917.39it/s]
 28%|████████████████████▌
| 2791/10000 [00:00<00:00, 14303.60it/s]

32000
300

100%|█████████████████████████████████████████████████████████████████████
████| 10000/10000 [00:00<00:00, 18878.77it/s]
 59%|████████████████████████████████████████████▏
| 4725/8000 [00:00<00:00, 23624.64it/s]

10000
300

100%|█████████████████████████████████████████████████████████████████████
██████| 8000/8000 [00:00<00:00, 16171.04it/s]

8000
300
```

In [89]:

```
X_train_tfidf_w2v_title =np.array(X_train_tfidf_w2v_title)
X_test_tfidf_w2v_title =np.array(X_test_tfidf_w2v_title)
X_cv_tfidf_w2v_title =np.array(X_cv_tfidf_w2v_title)
```

In [90]:

```
X_train_tfidf_w2v = np.hstack((train_category_responseCoding,train_subcategory_response
Coding,train_grade_responseCoding
                          ,train_teacher_responseCoding,train_state_responseCoding,
                            X_train_tfidf_w2v_title,X_train_tfidf_w2v_ess,X_train_price
_stndrd, X_train_quantity_stndrd, X_train_prev_proj_stndrd))
```

In [91]:

```
X_cv_tfidf_w2v = np.hstack((cv_category_responseCoding,cv_subcategory_responseCoding,cv
_grade_responseCoding
                          ,cv_teacher_responseCoding,cv_state_responseCoding,
                            X_cv_tfidf_w2v_title,X_cv_tfidf_w2v_ess,X_cv_price_stndrd,
X_cv_quantity_stndrd, X_cv_prev_proj_stndrd))
```

In [92]:

```
X_test_tfidf_w2v = np.hstack((test_category_responseCoding,test_subcategory_responseCod
ing,test_grade_responseCoding
                          ,test_teacher_responseCoding,test_state_responseCoding,
                            X_test_tfidf_w2v_title,X_test_tfidf_w2v_ess,X_test_price_st
ndrd, X_test_quantity_stndrd, X_test_prev_proj_stndrd))
```

In [93]:

```
print(X_train_tfidf_w2v.shape, y_train.shape)
print(X_cv_tfidf_w2v.shape, y_cv.shape)
print(X_test_tfidf_w2v.shape, y_test.shape)
```

```
(32000, 613) (32000,)
(8000, 613) (8000,)
(10000, 613) (10000,)
```

In [216]:

```
X_train_tfidf_w2v = X_train_tfidf_w2v[0:20000]
X_cv_tfidf_w2v = X_cv_tfidf_w2v[0:5000]
X_test_tfidf_w2v = X_test_tfidf_w2v[0:5000]
```

In [240]:

```
X_train_tfidf_w2v
```

Out[240]:

```
array([[ 0.14262948,  0.80159363,  0.12903226, ..., -0.07294389,
        -0.07618667, -0.07792049],
       [ 0.17905405,  0.76182432,  0.12096774, ..., -0.07294389,
        -0.07618667, -0.40075831],
       [ 0.13598091,  0.8541959 ,  0.13522925, ..., -0.07294389,
        -0.07618667,  0.28078819],
       ...,
       [ 0.13188799,  0.85230352,  0.12965964, ..., -0.07294389,
        -0.07618667, -0.36488744],
       [ 0.13188799,  0.85230352,  0.1296875 , ..., -0.07294389,
        -0.07618667,  3.3656829 ],
       [ 0.13598091,  0.8541959 ,  0.13522925, ..., -0.07294389,
        -0.07618667,  4.08310027]])
```

In [245]:

```
from scipy.sparse import csr_matrix
X_train_tfidf_w2v = csr_matrix(X_train_tfidf_w2v)
X_cv_tfidf_w2v = csr_matrix(X_cv_tfidf_w2v)
X_test_tfidf_w2v = csr_matrix(X_test_tfidf_w2v)
```

In [244]:

```
X_train_tfidf_w2v
```

Out[244]:

```
<20000x613 sparse matrix of type '<class 'numpy.float64'>'
        with 12254900 stored elements in Compressed Sparse Row format>
```

In [246]:

```
y_train1 = y_train[0:20000]
y_cv1 = y_cv[0:5000]
y_test1 = y_test[0:5000]
```

XGBoost Classifier on bag of words

In [119]:

```
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV
from xgboost.sklearn import XGBClassifier
from scipy.stats import uniform

xgb = XGBClassifier(min_samples_split = 20, class_weights = "balanced" )

params = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4
, 5, 6, 7, 8, 9, 10]}

clf = GridSearchCV(xgb,params ,cv=3, scoring='roc_auc',n_jobs=-1,return_train_score = T
rue)
clf.fit(X_train_bow, y_train)
```

Out[119]:

```
GridSearchCV(cv=3, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     class_weights='balanced',
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=No
ne,
                                     max_depth=None, min_child_weight=Non
e,
                                     min_samples_split=20...
                                     random_state=None, reg_alpha=None,
                                     reg_lambda=None, scale_pos_weight=Non
e,
                                     subsample=None, tree_method=None,
                                     validate_parameters=False,
                                     verbosity=None),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [10, 50, 100, 150, 200, 300, 500,
                                          1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [120]:

```
clf.best_estimator_
```

Out[120]:

```
XGBClassifier(base_score=0.5, booster=None, class_weights='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints=None, learning_rate=0.300000012,
              max_delta_step=0, max_depth=2, min_child_weight=1,
              min_samples_split=20, missing=nan, monotone_constraints=Non
e,
              n_estimators=200, n_jobs=0, num_parallel_tree=1,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=N
one,
              validate_parameters=False, verbosity=None)
```

In [121]:

```
depth = clf.best_params_['max_depth']
est = clf.best_params_['n_estimators']
```

In [122]:

```
train_auc
```

Out[122]:

```
[[0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263]]
```

In [124]:

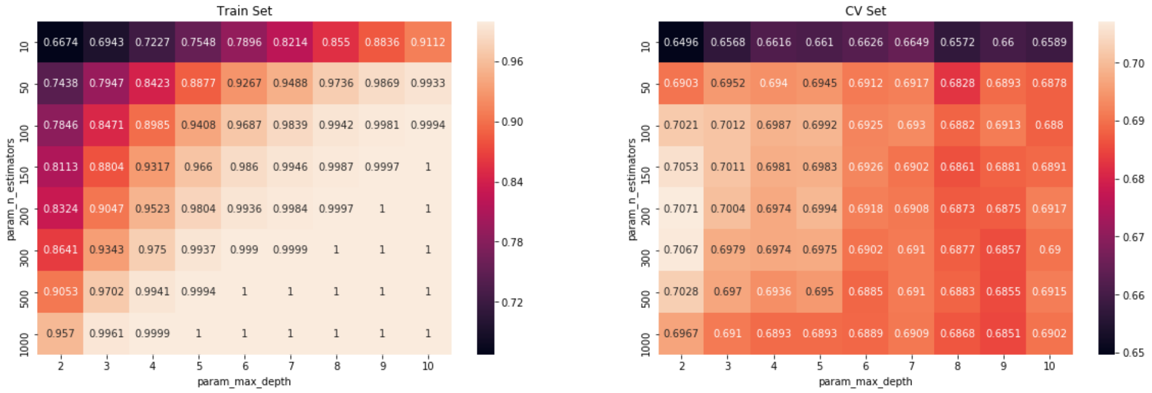
```
cv_auc
```

Out[124]:

```
[[0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611]]
```

In [125]:

```
max_scores = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_de
pth']).max().unstack()[['mean_test_score', 'mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(20,6))

sns.heatmap(max_scores.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores.mean_test_score, annot = True, fmt='.4g', ax=ax[1])

ax[0].set_title('Train Set')
ax[1].set_title('CV Set')

plt.show()
```



In [133]:

```
clf.cv_results_['mean_train_score']
```

Out[133]:

```
array([0.6674354 , 0.74383242, 0.78460117, 0.8112808 , 0.83235042,
       0.86409668, 0.9052849 , 0.9570029 , 0.69431047, 0.79467003,
       0.84706255, 0.88041158, 0.90469305, 0.93427165, 0.97020813,
       0.99607067, 0.72271532, 0.84231548, 0.89846177, 0.93165425,
       0.95225074, 0.97500793, 0.9940574 , 0.99990685, 0.75483218,
       0.88768815, 0.94075888, 0.96595575, 0.98039659, 0.99368657,
       0.99942786, 0.99999993, 0.78959635, 0.92671994, 0.9686866 ,
       0.98597084, 0.99361491, 0.99899802, 0.99998191, 1.        ,
       0.82140517, 0.94882265, 0.98386481, 0.99461066, 0.9983725 ,
       0.99987329, 0.99999969, 1.        , 0.85497459, 0.97362512,
       0.99415887, 0.99869697, 0.99968997, 0.99998957, 0.99999999,
       1.        , 0.88364093, 0.98691781, 0.998068  , 0.99967866,
       0.99996698, 0.99999973, 1.        , 1.        , 0.91122045,
       0.99330788, 0.99941022, 0.99995387, 0.99999798, 0.99999999,
       1.        , 1.        ])
```

In [149]:

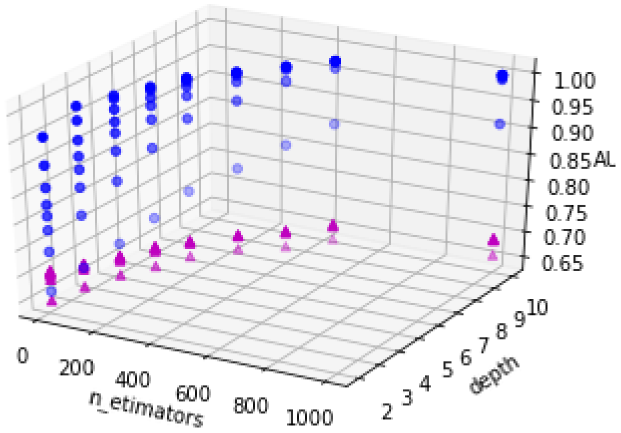
```
clf.cv_results_['mean_test_score']
```

Out[149]:

```
array([0.64963126, 0.69027054, 0.70206004, 0.70532383, 0.70710229,
       0.70666195, 0.70277032, 0.69666827, 0.6568322 , 0.69518936,
       0.70116077, 0.7010744 , 0.70040751, 0.69785515, 0.696977  ,
       0.69095007, 0.66159207, 0.69404992, 0.69865929, 0.69806145,
       0.6973679 , 0.69743385, 0.69363659, 0.68927876, 0.66095335,
       0.6945325 , 0.69916041, 0.6982655 , 0.69941131, 0.69746561,
       0.69499147, 0.6893401 , 0.66260728, 0.69115897, 0.6924569 ,
       0.69262815, 0.69177866, 0.69023135, 0.68854228, 0.68885363,
       0.66487424, 0.69168465, 0.69298485, 0.69023711, 0.69082064,
       0.69098274, 0.6909924 , 0.6909021 , 0.65722175, 0.68277705,
       0.68824675, 0.68607758, 0.68727211, 0.68768611, 0.68831133,
       0.68676703, 0.66003998, 0.68934701, 0.69126131, 0.68809056,
       0.68747151, 0.68573917, 0.68546194, 0.68507471, 0.65885477,
       0.68781924, 0.68799872, 0.68913144, 0.6917303 , 0.69004957,
       0.69150451, 0.69019197])
```

In [195]:

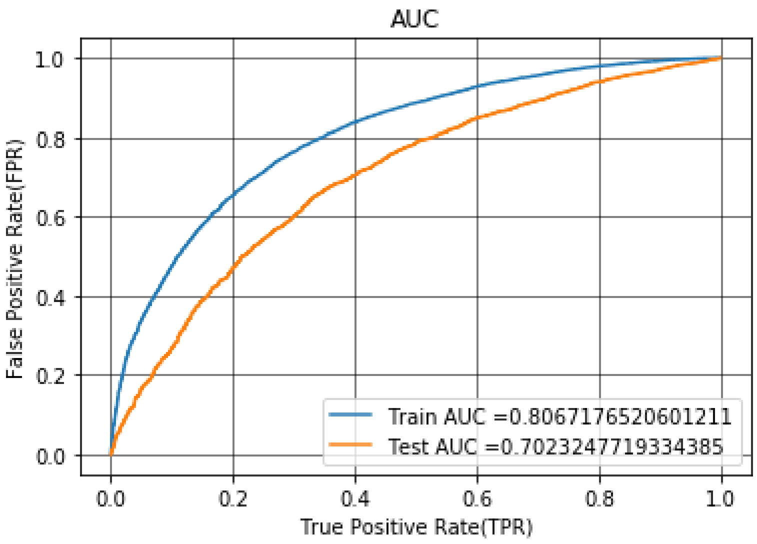
```
# https://pythonprogramming.net/3d-scatter-plot-customizing/
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x1=[10,10,10,10,10,10,10,10,
      50,50,50,50,50,50,50,50,
      100,100,100,100,100,100,100,100,
      150,150,150,150,150,150,150,150,
      200,200,200,200,200,200,200,200,
      300,300,300,300,300,300,300,300,
      400,400,400,400,400,400,400,400,
      500,500,500,500,500,500,500,500,
      1000,1000,1000,1000,1000,1000,1000,1000]
y1=[2,2,2,2,2,2,2,2,
      3,3,3,3,3,3,3,3,
      4,4,4,4,4,4,4,4,
      5,5,5,5,5,5,5,5,
      6,6,6,6,6,6,6,6,
      7,7,7,7,7,7,7,7,
      8,8,8,8,8,8,8,8,
      9,9,9,9,9,9,9,9,
      10,10,10,10,10,10,10,10]
z1=(list(clf.cv_results_['mean_train_score']))

x2=[10,10,10,10,10,10,10,10,
      50,50,50,50,50,50,50,50,
      100,100,100,100,100,100,100,100,
      150,150,150,150,150,150,150,150,
      200,200,200,200,200,200,200,200,
      300,300,300,300,300,300,300,300,
      400,400,400,400,400,400,400,400,
      500,500,500,500,500,500,500,500,
      1000,1000,1000,1000,1000,1000,1000,1000]
y2=[2,2,2,2,2,2,2,2,
      3,3,3,3,3,3,3,3,
      4,4,4,4,4,4,4,4,
      5,5,5,5,5,5,5,5,
      6,6,6,6,6,6,6,6,
      7,7,7,7,7,7,7,7,
      8,8,8,8,8,8,8,8,
      9,9,9,9,9,9,9,9,
      10,10,10,10,10,10,10,10]
z2 =(list(clf.cv_results_['mean_test_score']))

ax.scatter(x1, y1, z1,c='b', marker ='o')
ax.scatter(x2, y2, z2, c ='m', marker='^')

ax.set_xlabel('n_etimators')
ax.set_ylabel('depth')
ax.set_zlabel('AUC')

plt.show()
```



Hyperparameter tuning

In [197]:

```
from sklearn.metrics import roc_auc_score

import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

model = XGBClassifier(max_depth = depth, n_estimators = est, class_weight= 'balanced')
model.fit(X_train_bow, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs

y_train_pred = batch_predict(model, X_train_bow)
y_test_pred = batch_predict(model, X_test_bow)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



best threshold

In [198]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou
nd(t,3))
    return t

def predict_with_best_t(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

confusion matrix

In [199]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
print('='*100)
```

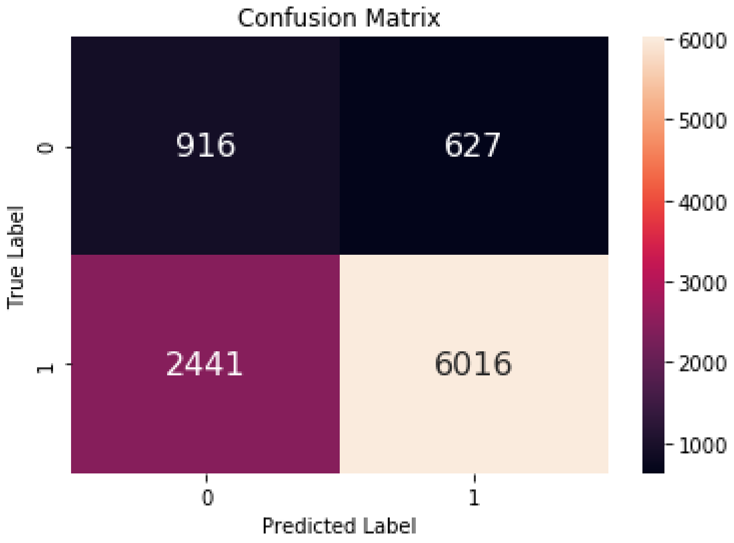
```
the maximum value of tpr*(1-fpr) 0.5390480749158721 for threshold 0.821
Train confusion matrix
[[ 3595  1342]
 [ 7029 20034]]
Test confusion matrix
[[ 916  627]
 [2441 6016]]
==========================================================================
==========================
```

In [200]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[200]:

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

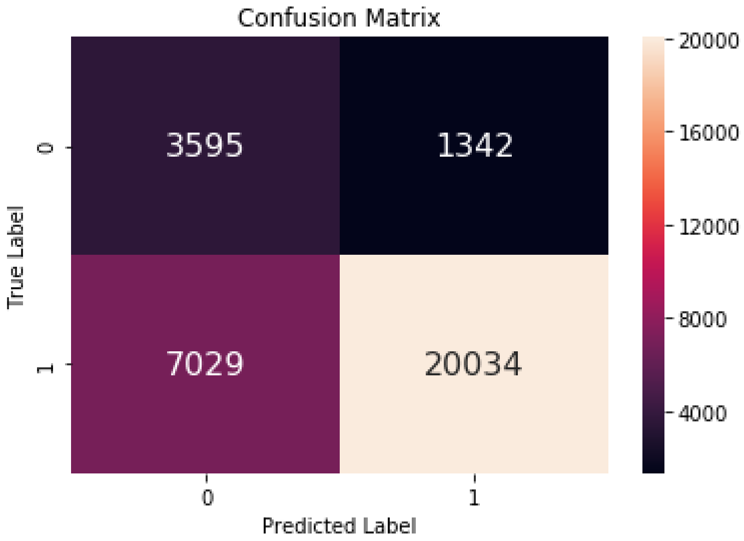


In [201]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_train,  predict_with_best_t(y_train_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[201]:

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



XGBoost classifier on tfidf

In [ ]:

```
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV
from xgboost.sklearn import XGBClassifier
from scipy.stats import uniform

xgb = XGBClassifier(min_samples_split = 20, class_weights = "balanced" )

params = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4
, 5, 6, 7, 8, 9, 10]}

clf = GridSearchCV(xgb,params ,cv=3, scoring='roc_auc',n_jobs=-1,return_train_score = T
rue)
clf.fit(X_train_tfidf, y_train)
```

In [203]:

```
clf.best_estimator_
```

Out[203]:

```
XGBClassifier(base_score=0.5, booster=None, class_weights='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints=None, learning_rate=0.300000012,
              max_delta_step=0, max_depth=2, min_child_weight=1,
              min_samples_split=20, missing=nan, monotone_constraints=Non
e,
              n_estimators=300, n_jobs=0, num_parallel_tree=1,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=N
one,
              validate_parameters=False, verbosity=None)
```

In [204]:

```
depth = clf.best_params_['max_depth']
est = clf.best_params_['n_estimators']
```

In [205]:

```
train_auc
```

Out[205]:

```
[[0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263]]
```

In [206]:

```
cv_auc
```

Out[206]:

```
[[0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611]]
```

In [207]:

```
clf.cv_results_['mean_train_score']
```

Out[207]:

```
array([0.67629006, 0.76316635, 0.81068928, 0.84233379, 0.86826341,
       0.90384192, 0.94666161, 0.98811185, 0.70674544, 0.82005868,
       0.87647088, 0.91324393, 0.93680478, 0.96692794, 0.9912518 ,
       0.9998823 , 0.73972631, 0.87075021, 0.93005908, 0.96055601,
       0.97753041, 0.99302483, 0.99948451, 0.99999985, 0.77808393,
       0.91715298, 0.9660312 , 0.98569085, 0.99418522, 0.99915792,
       0.99999347, 0.99999999, 0.812395  , 0.94915143, 0.98621277,
       0.99637824, 0.99908452, 0.99996546, 0.99999983, 1.        ,
       0.84826302, 0.97325032, 0.99503156, 0.99918454, 0.99989726,
       0.9999988 , 0.99999998, 1.        , 0.88539066, 0.98570201,
       0.99840982, 0.99987965, 0.99999448, 0.99999995, 1.        ,
       1.        , 0.91201397, 0.99377718, 0.99969024, 0.99998538,
       0.99999941, 0.99999998, 1.        , 1.        , 0.93235751,
       0.99762537, 0.99995125, 0.99999927, 0.99999993, 1.        ,
       1.        , 1.        ])
```

In [208]:

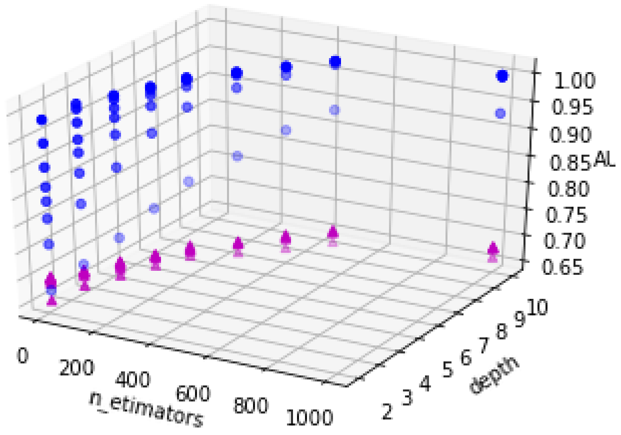
```
clf.cv_results_['mean_test_score']
```

Out[208]:

```
array([0.65756178, 0.69510097, 0.7001974 , 0.70070709, 0.70105049,
       0.70154165, 0.69694046, 0.69035468, 0.66683983, 0.6943592 ,
       0.69537866, 0.69389225, 0.69454365, 0.69348192, 0.68983891,
       0.68509986, 0.66872286, 0.69191426, 0.69221809, 0.69469937,
       0.69427113, 0.68942628, 0.68550168, 0.68111929, 0.67038094,
       0.68957966, 0.68983544, 0.68989128, 0.6881504 , 0.68899857,
       0.68551759, 0.68362983, 0.67130831, 0.69146323, 0.6883848 ,
       0.68569785, 0.68492183, 0.6824504 , 0.67946117, 0.68288426,
       0.66580371, 0.68396633, 0.68487531, 0.68685498, 0.68458381,
       0.6840108 , 0.68210294, 0.68253265, 0.66241563, 0.6847651 ,
       0.68732686, 0.6874619 , 0.68412464, 0.68123277, 0.67934134,
       0.6819155 , 0.66216764, 0.67937391, 0.68132866, 0.68112908,
       0.68066683, 0.68246963, 0.68400049, 0.68566901, 0.66238538,
       0.68139227, 0.67906907, 0.67923872, 0.67962506, 0.68064388,
       0.68167393, 0.68239771])
```

In [209]:

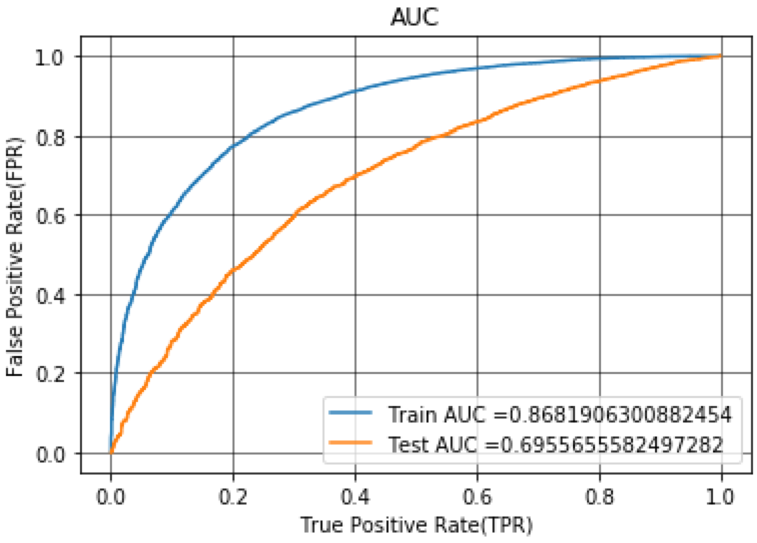
```
# https://pythonprogramming.net/3d-scatter-plot-customizing/
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x1=[10,10,10,10,10,10,10,10,
      50,50,50,50,50,50,50,50,
      100,100,100,100,100,100,100,100,
      150,150,150,150,150,150,150,150,
      200,200,200,200,200,200,200,200,
      300,300,300,300,300,300,300,300,
      400,400,400,400,400,400,400,400,
      500,500,500,500,500,500,500,500,
      1000,1000,1000,1000,1000,1000,1000,1000]
y1=[2,2,2,2,2,2,2,2,
      3,3,3,3,3,3,3,3,
      4,4,4,4,4,4,4,4,
      5,5,5,5,5,5,5,5,
      6,6,6,6,6,6,6,6,
      7,7,7,7,7,7,7,7,
      8,8,8,8,8,8,8,8,
      9,9,9,9,9,9,9,9,
      10,10,10,10,10,10,10,10]
z1=(list(clf.cv_results_['mean_train_score']))

x2=[10,10,10,10,10,10,10,10,
      50,50,50,50,50,50,50,50,
      100,100,100,100,100,100,100,100,
      150,150,150,150,150,150,150,150,
      200,200,200,200,200,200,200,200,
      300,300,300,300,300,300,300,300,
      400,400,400,400,400,400,400,400,
      500,500,500,500,500,500,500,500,
      1000,1000,1000,1000,1000,1000,1000,1000]
y2=[2,2,2,2,2,2,2,2,
      3,3,3,3,3,3,3,3,
      4,4,4,4,4,4,4,4,
      5,5,5,5,5,5,5,5,
      6,6,6,6,6,6,6,6,
      7,7,7,7,7,7,7,7,
      8,8,8,8,8,8,8,8,
      9,9,9,9,9,9,9,9,
      10,10,10,10,10,10,10,10]
z2 =(list(clf.cv_results_['mean_test_score']))

ax.scatter(x1, y1, z1,c='b', marker ='o')
ax.scatter(x2, y2, z2, c ='m', marker='^')

ax.set_xlabel('n_etimators')
ax.set_ylabel('depth')
ax.set_zlabel('AUC')

plt.show()
```



Hyperparameter tuning

In [210]:

```
from sklearn.metrics import roc_auc_score

import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

model = XGBClassifier(max_depth = depth, n_estimators = est, class_weight= 'balanced')
model.fit(X_train_tfidf, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs

y_train_pred = batch_predict(model, X_train_tfidf)
y_test_pred = batch_predict(model, X_test_tfidf)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



best threshold

In [211]:

```
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou
nd(t,3))
    return t

def predict_with_best_t(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

Confusion matrix

In [212]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
print('='*100)
```

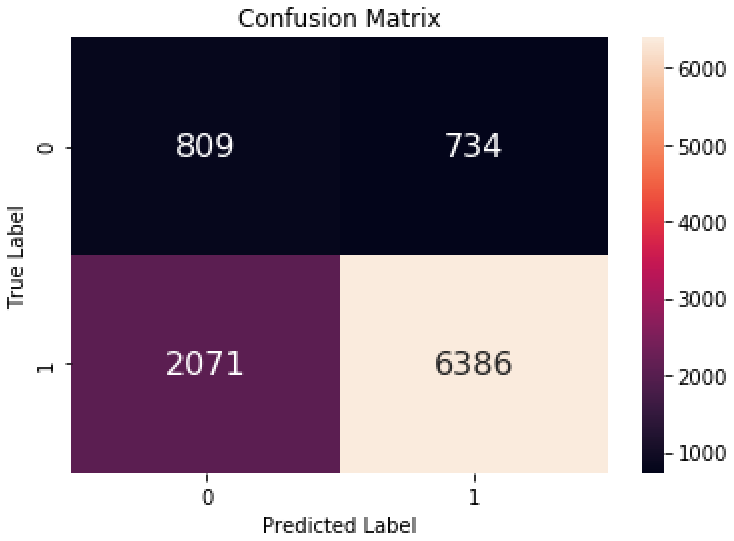
```
the maximum value of tpr*(1-fpr) 0.6199354672704177 for threshold 0.812
Train confusion matrix
[[ 3799  1138]
 [ 5260 21803]]
Test confusion matrix
[[ 809  734]
 [2071 6386]]
==========================================================================
==========================
```

In [213]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[213]:

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

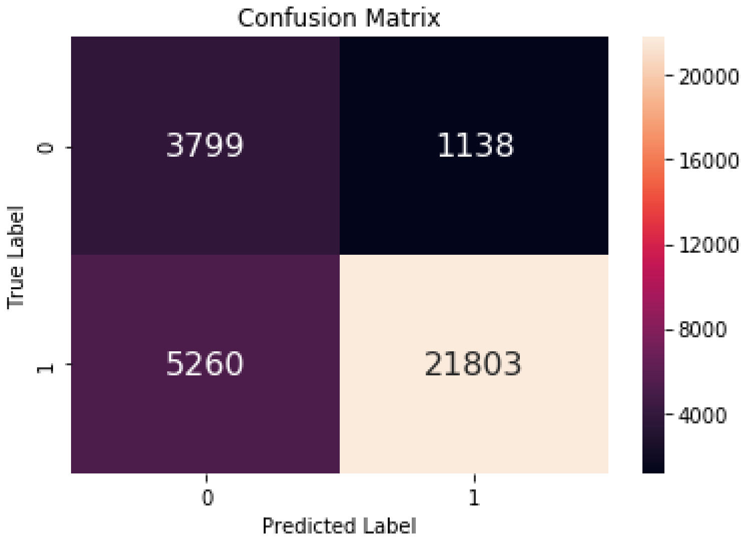


In [214]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_train,  predict_with_best_t(y_train_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[214]:

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



XGBoost classifier on avgw2v

In [218]:

```
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV
from xgboost.sklearn import XGBClassifier
from scipy.stats import uniform

xgb = XGBClassifier(min_samples_split = 20, class_weights = "balanced" )

params = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4
, 5, 6, 7, 8, 9, 10]}

clf = GridSearchCV(xgb,params ,cv=3, scoring='roc_auc',n_jobs=-1,return_train_score = T
rue)
clf.fit(X_train_avg_w2v, y_train1)
```

Out[218]:

```
GridSearchCV(cv=3, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     class_weights='balanced',
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=No
ne,
                                     max_depth=None, min_child_weight=Non
e,
                                     min_samples_split=20...
                                     random_state=None, reg_alpha=None,
                                     reg_lambda=None, scale_pos_weight=Non
e,
                                     subsample=None, tree_method=None,
                                     validate_parameters=False,
                                     verbosity=None),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [10, 50, 100, 150, 200, 300, 500,
                                          1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [219]:

```
clf.best_estimator_
```

Out[219]:

```
XGBClassifier(base_score=0.5, booster=None, class_weights='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints=None, learning_rate=0.300000012,
              max_delta_step=0, max_depth=2, min_child_weight=1,
              min_samples_split=20, missing=nan, monotone_constraints=Non
e,
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=N
one,
              validate_parameters=False, verbosity=None)
```

In [220]:

```
depth = clf.best_params_['max_depth']
est = clf.best_params_['n_estimators']
```

In [221]:

```
train_auc
```

Out[221]:

```
[[0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263]]
```

In [222]:

```
cv_auc
```

Out[222]:

```
[[0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611]]
```

In [223]:

```
clf.cv_results_['mean_train_score']
```

Out[223]:

```
array([0.69770178, 0.7963364 , 0.85434836, 0.89177822, 0.91922667,
       0.95417752, 0.98679375, 0.999806  , 0.73991573, 0.87742672,
       0.94458274, 0.97580665, 0.99098738, 0.9993085 , 0.99999973,
       0.99999999, 0.7906739 , 0.95359075, 0.99424043, 0.99957509,
       0.99999269, 0.99999997, 1.        , 1.        , 0.84913213,
       0.99116146, 0.99997079, 0.99999999, 0.99999999, 1.        ,
       1.        , 1.        , 0.905759  , 0.99968949, 0.99999999,
       0.99999999, 1.        , 1.        , 1.        , 1.        ,
       0.95517446, 0.99999956, 0.99999999, 1.        , 1.        ,
       1.        , 1.        , 1.        , 0.98434838, 0.99999999,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 0.99662433, 0.99999999, 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 0.99934134,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        ])
```

In [224]:

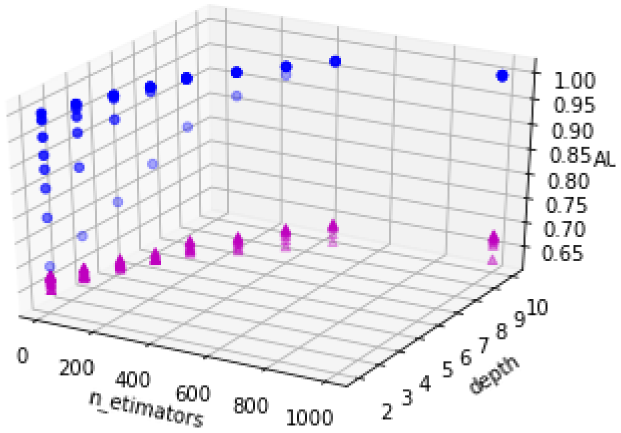
```
clf.cv_results_['mean_test_score']
```

Out[224]:

```
array([0.65032557, 0.67743854, 0.68036651, 0.6775126 , 0.67413214,
       0.67328752, 0.66743535, 0.66029134, 0.65369709, 0.67472831,
       0.67211933, 0.66699724, 0.66052666, 0.65925772, 0.65781973,
       0.65817404, 0.65366275, 0.66695183, 0.66442038, 0.66113159,
       0.6599653 , 0.66200818, 0.66904735, 0.67354963, 0.65661614,
       0.65579333, 0.65418927, 0.65308488, 0.65402444, 0.65806702,
       0.66473008, 0.66844757, 0.648934  , 0.65401801, 0.6560947 ,
       0.65895004, 0.66406963, 0.66985806, 0.6721163 , 0.67466681,
       0.64071368, 0.64704056, 0.65456803, 0.65975646, 0.66279515,
       0.66501774, 0.66749752, 0.66949223, 0.63591125, 0.64913714,
       0.65872289, 0.6652786 , 0.66751217, 0.66997802, 0.67234865,
       0.67353118, 0.63304688, 0.64931898, 0.66215575, 0.66545828,
       0.6666352 , 0.66842002, 0.6694545 , 0.67161014, 0.62766333,
       0.65869225, 0.66679016, 0.67105562, 0.67438673, 0.6763753 ,
       0.67809179, 0.67966925])
```

In [225]:

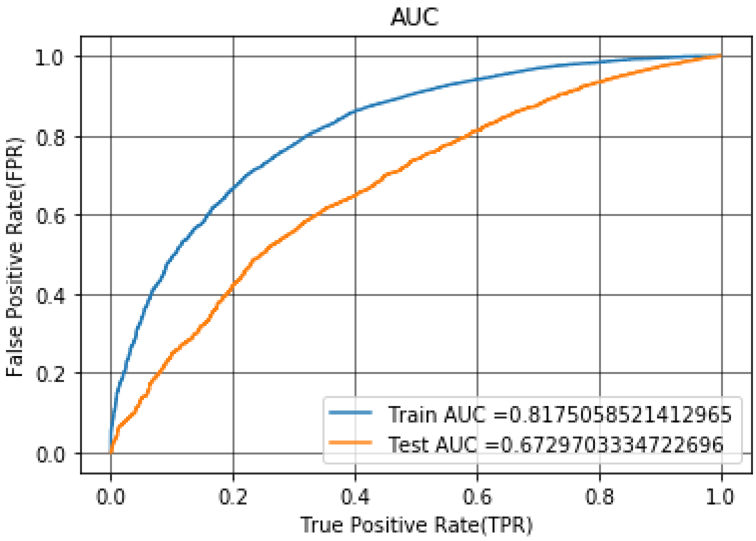
```
# https://pythonprogramming.net/3d-scatter-plot-customizing/
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x1=[10,10,10,10,10,10,10,10,
      50,50,50,50,50,50,50,50,
      100,100,100,100,100,100,100,100,
      150,150,150,150,150,150,150,150,
      200,200,200,200,200,200,200,200,
      300,300,300,300,300,300,300,300,
      400,400,400,400,400,400,400,400,
      500,500,500,500,500,500,500,500,
      1000,1000,1000,1000,1000,1000,1000,1000]
y1=[2,2,2,2,2,2,2,2,
      3,3,3,3,3,3,3,3,
      4,4,4,4,4,4,4,4,
      5,5,5,5,5,5,5,5,
      6,6,6,6,6,6,6,6,
      7,7,7,7,7,7,7,7,
      8,8,8,8,8,8,8,8,
      9,9,9,9,9,9,9,9,
      10,10,10,10,10,10,10,10]
z1=(list(clf.cv_results_['mean_train_score']))

x2=[10,10,10,10,10,10,10,10,
      50,50,50,50,50,50,50,50,
      100,100,100,100,100,100,100,100,
      150,150,150,150,150,150,150,150,
      200,200,200,200,200,200,200,200,
      300,300,300,300,300,300,300,300,
      400,400,400,400,400,400,400,400,
      500,500,500,500,500,500,500,500,
      1000,1000,1000,1000,1000,1000,1000,1000]
y2=[2,2,2,2,2,2,2,2,
      3,3,3,3,3,3,3,3,
      4,4,4,4,4,4,4,4,
      5,5,5,5,5,5,5,5,
      6,6,6,6,6,6,6,6,
      7,7,7,7,7,7,7,7,
      8,8,8,8,8,8,8,8,
      9,9,9,9,9,9,9,9,
      10,10,10,10,10,10,10,10]
z2 =(list(clf.cv_results_['mean_test_score']))

ax.scatter(x1, y1, z1,c='b', marker ='o')
ax.scatter(x2, y2, z2, c ='m', marker='^')

ax.set_xlabel('n_etimators')
ax.set_ylabel('depth')
ax.set_zlabel('AUC')

plt.show()
```



Hyperparameter tuning

In [233]:

```
from sklearn.metrics import roc_auc_score

import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

model = XGBClassifier(max_depth = depth, n_estimators = est, class_weight= 'balanced')
model.fit(X_train_avg_w2v, y_train1)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs

y_train_pred = batch_predict(model, X_train_avg_w2v)
y_test_pred = batch_predict(model, X_test_avg_w2v)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train1, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test1, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



best threshold

In [234]:

```
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou
nd(t,3))
    return t

def predict_with_best_t(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

confusion matrix

In [235]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train1, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test1, predict_with_best_t(y_test_pred, best_t)))
print('='*100)
```

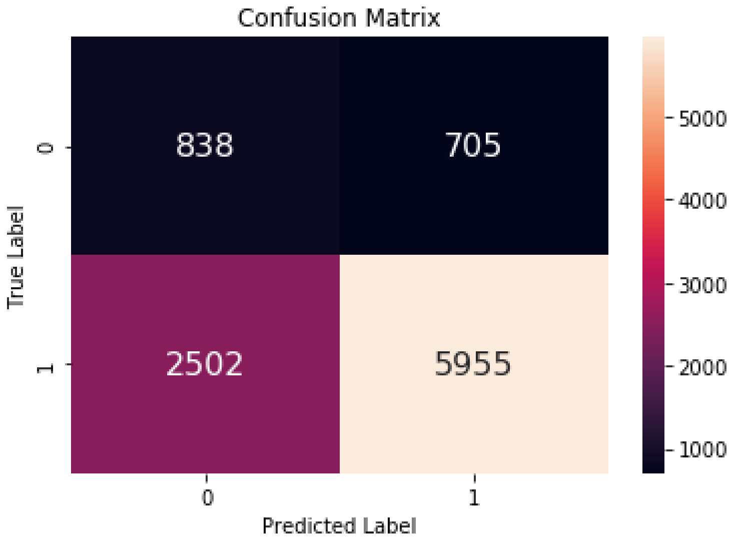
```
the maximum value of tpr*(1-fpr) 0.5475283597036675 for threshold 0.826
Train confusion matrix
[[ 2311   842]
 [ 4262 12585]]
Test confusion matrix
[[ 838  705]
 [2502 5955]]
==========================================================================
==========================
```

In [236]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_test1, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[236]:

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

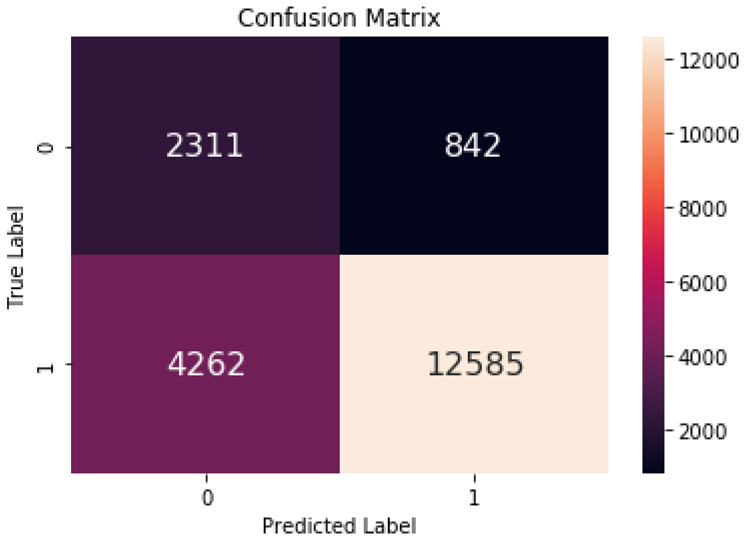


In [237]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_train1,  predict_with_best_t(y_train_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[237]:

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



XGBoost classifier on tfidf_w2v

In [247]:

```
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV
from xgboost.sklearn import XGBClassifier
from scipy.stats import uniform

xgb = XGBClassifier(min_samples_split = 20, class_weights = "balanced" )

params = {'n_estimators': [10, 50, 100, 150, 200, 300, 500, 1000], 'max_depth':[2, 3, 4
, 5, 6, 7, 8, 9, 10]}

clf = GridSearchCV(xgb,params ,cv=3, scoring='roc_auc',n_jobs=-1,return_train_score = T
rue)
clf.fit(X_train_tfidf_w2v, y_train1)
```

Out[247]:

```
GridSearchCV(cv=3, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     class_weights='balanced',
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=No
ne,
                                     max_depth=None, min_child_weight=Non
e,
                                     min_samples_split=20...
                                     random_state=None, reg_alpha=None,
                                     reg_lambda=None, scale_pos_weight=Non
e,
                                     subsample=None, tree_method=None,
                                     validate_parameters=False,
                                     verbosity=None),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [10, 50, 100, 150, 200, 300, 500,
                                          1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [250]:

```
clf.best_estimator_
```

Out[250]:

```
XGBClassifier(base_score=0.5, booster=None, class_weights='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints=None, learning_rate=0.300000012,
              max_delta_step=0, max_depth=2, min_child_weight=1,
              min_samples_split=20, missing=nan, monotone_constraints=Non
e,
              n_estimators=50, n_jobs=0, num_parallel_tree=1,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=N
one,
              validate_parameters=False, verbosity=None)
```

In [251]:

```
depth = clf.best_params_['max_depth']
est = clf.best_params_['n_estimators']
```

In [252]:

```
train_auc
```

Out[252]:

```
[[0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263,
  0.9421437227269263]]
```

In [253]:

```
cv_auc
```

Out[253]:

```
[[0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611,
  0.6854302018242611]]
```

In [254]:

```
clf.cv_results_['mean_train_score']
```

Out[254]:

```
array([0.6964544 , 0.79067334, 0.84825124, 0.88601246, 0.91466223,
       0.95247255, 0.98632219, 0.99981954, 0.74048164, 0.87527534,
       0.94377892, 0.97740463, 0.99169535, 0.9992485 , 0.99999952,
       0.99999999, 0.79137876, 0.95276599, 0.99395409, 0.99966728,
       0.99999413, 0.99999999, 0.99999999, 1.        , 0.84982265,
       0.9914555 , 0.99998293, 0.99999997, 0.99999999, 1.        ,
       1.        , 1.        , 0.91298801, 0.99979219, 0.99999997,
       0.99999999, 1.        , 1.        , 1.        , 1.        ,
       0.9546729 , 0.99999987, 0.99999999, 1.        , 1.        ,
       1.        , 1.        , 1.        , 0.98327509, 0.99999999,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 0.99700743, 0.99999999, 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 0.99921626,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        ])
```

In [255]:

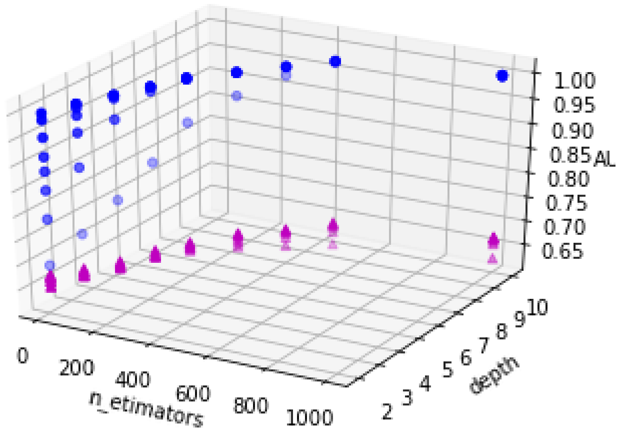
```
clf.cv_results_['mean_test_score']
```

Out[255]:

```
array([0.65320563, 0.67687186, 0.67494745, 0.67139323, 0.66998025,
       0.66432332, 0.6627946 , 0.65193693, 0.6639518 , 0.66870321,
       0.66316717, 0.66257322, 0.657548  , 0.65464573, 0.65289986,
       0.65537833, 0.6607079 , 0.65913476, 0.65420918, 0.65178642,
       0.65021361, 0.65388793, 0.65920432, 0.6650629 , 0.6566556 ,
       0.65782059, 0.65622165, 0.65469607, 0.65795895, 0.66229287,
       0.66732464, 0.66834573, 0.64923595, 0.65411263, 0.65585368,
       0.65720709, 0.66155589, 0.66278314, 0.66518703, 0.66636371,
       0.64442384, 0.65302305, 0.65878735, 0.66515033, 0.66853089,
       0.67053425, 0.67189905, 0.67265468, 0.63338063, 0.65077006,
       0.65901417, 0.66337805, 0.66474817, 0.66608135, 0.66713213,
       0.66819139, 0.62360272, 0.64962291, 0.66021927, 0.66306351,
       0.66510069, 0.66750746, 0.66862697, 0.66917512, 0.62601897,
       0.65689034, 0.66350524, 0.6671952 , 0.66953263, 0.67001895,
       0.67068038, 0.67050187])
```

In [256]:

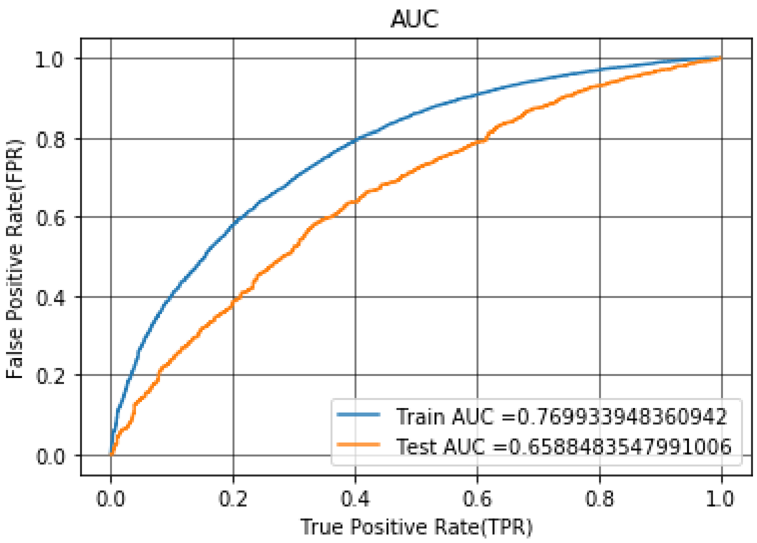
```
# https://pythonprogramming.net/3d-scatter-plot-customizing/
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt


fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x1=[10,10,10,10,10,10,10,10,
      50,50,50,50,50,50,50,50,
      100,100,100,100,100,100,100,100,
      150,150,150,150,150,150,150,150,
      200,200,200,200,200,200,200,200,
      300,300,300,300,300,300,300,300,
      400,400,400,400,400,400,400,400,
      500,500,500,500,500,500,500,500,
      1000,1000,1000,1000,1000,1000,1000,1000]
y1=[2,2,2,2,2,2,2,2,
      3,3,3,3,3,3,3,3,
      4,4,4,4,4,4,4,4,
      5,5,5,5,5,5,5,5,
      6,6,6,6,6,6,6,6,
      7,7,7,7,7,7,7,7,
      8,8,8,8,8,8,8,8,
      9,9,9,9,9,9,9,9,
      10,10,10,10,10,10,10,10]
z1=(list(clf.cv_results_['mean_train_score']))

x2=[10,10,10,10,10,10,10,10,
      50,50,50,50,50,50,50,50,
      100,100,100,100,100,100,100,100,
      150,150,150,150,150,150,150,150,
      200,200,200,200,200,200,200,200,
      300,300,300,300,300,300,300,300,
      400,400,400,400,400,400,400,400,
      500,500,500,500,500,500,500,500,
      1000,1000,1000,1000,1000,1000,1000,1000]
y2=[2,2,2,2,2,2,2,2,
      3,3,3,3,3,3,3,3,
      4,4,4,4,4,4,4,4,
      5,5,5,5,5,5,5,5,
      6,6,6,6,6,6,6,6,
      7,7,7,7,7,7,7,7,
      8,8,8,8,8,8,8,8,
      9,9,9,9,9,9,9,9,
      10,10,10,10,10,10,10,10]
z2 =(list(clf.cv_results_['mean_test_score']))

ax.scatter(x1, y1, z1,c='b', marker ='o')
ax.scatter(x2, y2, z2, c ='m', marker='^')

ax.set_xlabel('n_etimators')
ax.set_ylabel('depth')
ax.set_zlabel('AUC')

plt.show()
```



Hyperparameter tuning

In [258]:

```
from sklearn.metrics import roc_auc_score

import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

model = XGBClassifier(max_depth = depth, n_estimators = est, class_weight= 'balanced')
model.fit(X_train_tfidf_w2v, y_train1)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs

y_train_pred = batch_predict(model, X_train_tfidf_w2v)
y_test_pred = batch_predict(model, X_test_tfidf_w2v)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train1, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test1, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



best threshold

In [259]:

```
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou
nd(t,3))
    return t

def predict_with_best_t(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

confusion matrix

In [261]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train1, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test1, predict_with_best_t(y_test_pred, best_t)))
print('='*100)
```

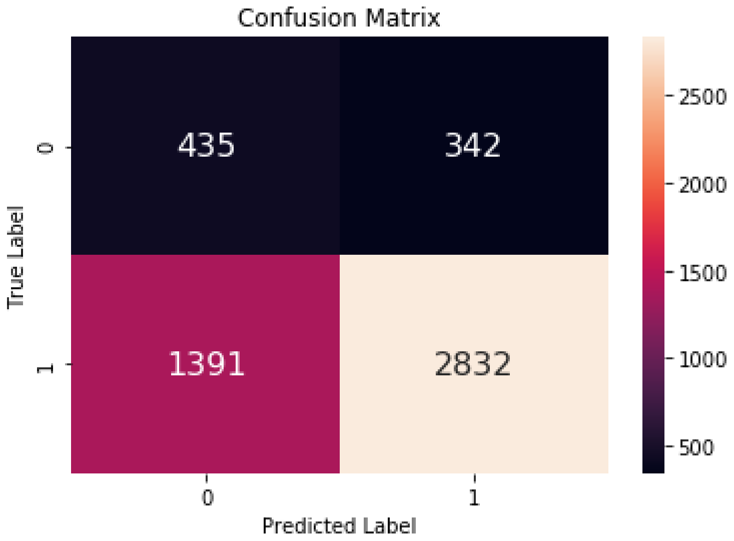
```
the maximum value of tpr*(1-fpr) 0.48895446040728 for threshold 0.831
Train confusion matrix
[[ 2194   959]
 [ 5009 11838]]
Test confusion matrix
[[ 435  342]
 [1391 2832]]
===========================================================================
=========================
```

In [262]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_test1, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[262]:

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

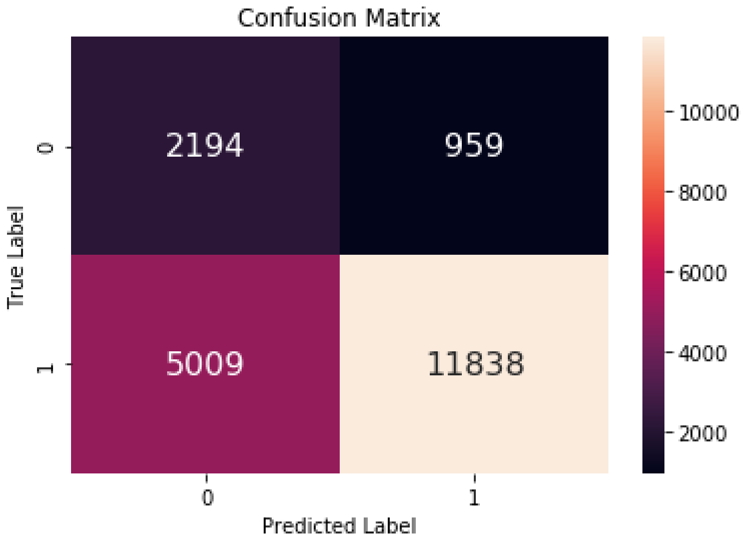


In [263]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_train1,  predict_with_best_t(y_train_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[263]:

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



In [264]:

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

ptable = PrettyTable()
ptable.title = 'Classification Report'

ptable.field_names = ["Vectorization", "Model", "max_depth", "n_estimators", "AUC"]

ptable.add_row(["BOW","XGBoost",2,200,70.23])
ptable.add_row(["tf-idf", "XGBoost",2,300,69.55])
ptable.add_row(["avg-w2v", "XGBoost",2,100,67.29])
ptable.add_row(["tf-idf-w2v", "XGBoost",2,50,65.88])

print(ptable)
```

```
+---------------+---------+-----------+--------------+-------+
| Vectorization |  Model  | max_depth | n_estimators |  AUC  |
+---------------+---------+-----------+--------------+-------+
|      BOW      | XGBoost |     2     |     200      | 70.23 |
|     tf-idf    | XGBoost |     2     |     300      | 69.55 |
|    avg-w2v    | XGBoost |     2     |     100      | 67.29 |
|   tf-idf-w2v  | XGBoost |     2     |      50      | 65.88 |
+---------------+---------+-----------+--------------+-------+
```

Summary

- Tried to plot the performance using 3d scatter plot.
- With bag of words more accuracy is obtained.
- By using xgboost classifier obtained more accuracy than random forest classifier.