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 mater 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. **Ex** 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_4:__ "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]:

```
## Importing the required libraries

%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

import re
from tqdm import tqdm

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

from collections import Counter
```

1.1 Reading the Data

In [2]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [3]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
print('='*100)
project_data.head(2)
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix'
'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
====================================================================
==========================
```

Out[3]:

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

Renaming and sorting the column [project_submitted_datetime]

In [4]:

```
project_data = project_data.rename(columns = {'project_submitted_datetime': 'Date'})

# Sorting the dataframe according to time
project_data['Date'] = pd.to_datetime(project_data['Date'])
project_data.sort_values(by = ['Date'], inplace = True)

project_data.head(2)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	scho
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

In [5]:

```
# Modifying values of project_data['project_grade_category']

project_grade_category = []

for i in tqdm(range(len(project_data))):
    a = project_data['project_grade_category'][i].replace(" ", "-")
    project_grade_category.append(a)
```

```
100%|████████████████████████████████████████████████████████████
██| 109248/109248 [00:02<00:00, 45643.41it/s]
```

In [6]:

```
project_grade_category[0:5]
```

Out[6]:

```
['Grades-PreK-2', 'Grades-6-8', 'Grades-6-8', 'Grades-PreK-2', 'Grades-Pre
K-2']
```

In [7]:

```
project_data.drop('project_grade_category', axis = 1, inplace=True)
```

In [8]:

```
project_data["project_grade_category"] = project_grade_category
```

In [9]:

```
# Printing the shape and features of resource data

print("Number of data points in resource data", resource_data.shape)
print('-'*50)
print("The attributes of resource data :", resource_data.columns.values)
print('='*100)
resource_data.head(2)
```

```
Number of data points in resource data (1541272, 4)
--------------------------------------------------
The attributes of resource data : ['id' 'description' 'quantity' 'price']
====================================================================================================
```

Out[9]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

1.2 preprocessing of project_subject_categories`

In [10]:

```
catogories = list(project_data['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 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('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.3 preprocessing of project_subject_subcategories`

In [11]:

```
sub_catogories = list(project_data['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())

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

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

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

1.4 Text preprocessing (project essays)

1.3.1 Combine all Project essays into 1 Essay

In [12]:

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

In [13]:

```
project_data.head(2)
```

Out[13]:

	Unnamed: 0	id	teacher_id	teacher_prefix	scho
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

In [14]:

```
# 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 [15]:

```
from nltk.corpus import stopwords
stopwords = stopwords.words('english')   # To remove words that comes under the stopwor
d.
print(stopwords)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "yo
u're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselv
es', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'hersel
f', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'the
mselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'thes
e', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'hav
e', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'th
e', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'a
t', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through',
'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'on
ce', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'no
t', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can',
'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll',
'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",
'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',
"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "would
n't"]
```

In [16]:

```
# Combining all the above statements.

from tqdm import tqdm
preprocessed_essays = []

for sentence in tqdm(project_data['essay'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\\r',' ')
    sent = sent.replace('\\"',' ')
    sent = sent.replace('\\n',' ')
    sent = re.sub('[^A-Za-z0-9]+',' ',sent)
    sent = sent.lower()
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.strip())
```

```
100%|████████████████████████████████████████████████████████████████
███| 109248/109248 [01:41<00:00, 1072.05it/s]
```

In [17]:

```
project_data['preprocessed_essays'] = preprocessed_essays
```

In [18]:

```
project_data.drop(["essay"], axis = 1, inplace=True)
```

In [19]:

```
project_data.head(2)
```

Out[19]:

	Unnamed: 0	id	teacher_id	teacher_prefix	scho
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

1.5 Adding New feature " essay_word_count" .

In [20]:

```
essay_word_count = []

for sent in tqdm(project_data['preprocessed_essays']):
    count = len(sent.split())
    essay_word_count.append(count)
```

```
100%|████████████████████████████████████████████████████████████████
██| 109248/109248 [00:01<00:00, 55531.98it/s]
```

In [21]:

```
project_data["essay_word_count"] = essay_word_count
```

In [22]:

```
project_data.head(2)
```

Out[22]:

	Unnamed: 0	id	teacher_id	teacher_prefix	scho
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

1.6 Sentiment Scores of the essays

In [23]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

In [24]:

```
sentiment = SentimentIntensityAnalyzer()

sentiment_neg = []
sentiment_neu = []
sentiment_pos = []
sentiment_compound = []

for sent in tqdm(project_data['preprocessed_essays']):
    neg = sentiment.polarity_scores(sent)['neg']
    neu = sentiment.polarity_scores(sent)['neu']
    pos = sentiment.polarity_scores(sent)['pos']
    com = sentiment.polarity_scores(sent)['compound']

    sentiment_neg.append(neg)
    sentiment_neu.append(neu)
    sentiment_pos.append(pos)
    sentiment_compound.append(com)
```

```
100%|██████████████████████████████████████████████████████████████████████
█████| 109248/109248 [24:08<00:00, 75.42it/s]
```

In [25]:

```
project_data['sentiment_neg'] = sentiment_neg
project_data['sentiment_neu'] = sentiment_neu
project_data['sentiment_pos'] = sentiment_pos
project_data['sentiment_compound'] = sentiment_compound
```

In [26]:

```
project_data.head(3)
```

Out[26]:

	Unnamed: 0	id	teacher_id	teacher_prefix	scho
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT
51140	74477	p189804	4a97f3a390bfe21b99cf5e2b81981c73	Mrs.	CA

3 rows × 23 columns

1.7 Preprocessing of project titles

In [27]:

```
from tqdm import tqdm
preprocessed_title = []

for title in tqdm(project_data['project_title'].values):
    sent = decontracted(title)
    sent = sent.replace('\\r',' ')
    sent = sent.replace('\\"',' ')
    sent = sent.replace('\\n',' ')
    sent = re.sub('[^A-Za-z0-9]+',' ',sent)
    sent = sent.lower()
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_title.append(sent.strip())
```

```
100%|████████████████████████████████████████████████████████████████
██| 109248/109248 [00:04<00:00, 22208.16it/s]
```

In [28]:

```
project_data.drop('project_title', axis = 1, inplace=True)
```

In [29]:

```
project_data['preprocessed_title'] = preprocessed_title
```

In [30]:

```
project_data.head(1)
```

Out[30]:

	Unnamed: 0	id	teacher_id	teacher_prefix	schoo
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA

1 rows × 23 columns

1.8 Adding New feature " title_word_count"

In [31]:

```
title_word_count = []

for i in tqdm(project_data['preprocessed_title']):
    count = len(i.split())
    title_word_count.append(count)
```

```
100%|██████████████████████████████████████████████████████████████████████
██| 109248/109248 [00:00<00:00, 616814.73it/s]
```

In [32]:

```
project_data['title_word_count'] = title_word_count
```

1.9 Splitting the dataset

In [33]:

```
from sklearn.model_selection import train_test_split

X = project_data.drop(['project_is_approved'],axis = 1)
y = project_data['project_is_approved']
```

In [34]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.33, stratify = y
)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size = 0.33, str
atify = y_train)
```

In [35]:

```
print('The shape of X_train & y_train: ',X_train.shape, y_train.shape)
print('The shape of X_cv & y_cv:       ',X_cv.shape, y_cv.shape)
print('The shape of X_test & y_test:   ',X_test.shape, y_test.shape)
```

```
The shape of X_train & y_train:  (49041, 23) (49041,)
The shape of X_cv & y_cv:        (24155, 23) (24155,)
The shape of X_test & y_test:    (36052, 23) (36052,)
```

1.10 Preparing Data for Models :

In [36]:

```
project_data.columns
```

Out[36]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       'project_grade_category', 'clean_categories', 'clean_subcategorie
s',
       'preprocessed_essays', 'essay_word_count', 'sentiment_neg',
       'sentiment_neu', 'sentiment_pos', 'sentiment_compound',
       'preprocessed_title', 'title_word_count'],
      dtype='object')
```

we are going to consider

- school_state - Categorical data
- clean_categories - Categorical data
- clean_subcategories - Categorical data
- project_grade_category - Categorical data
- teacher_prefix - Categorical data

- preprocessed_essay - Text data
- preprocesed_title - Text data
- project_resource_summary - Text data (optional)

- quantity - numerical (optional)
- teacher_number_of_previously_posted_projects - numerical
- price - numerical
- title_word_count : numerical
- essay_word_count : numerical
- sentiment_neg : numerical
- sentiment_neu : numerical
- sentiment_pos : numerical
- sentiment_compound : numerical

2. Vectorizing Categorical data:

2.1. One Hot Encoding of School State :

In [37]:

```
my_counter = Counter()
for state in project_data['school_state'].values:
    my_counter.update(state.split())

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

In [38]:

```
# Using CountVectorizer to convert the categorical data into one hot encoders:

from sklearn.feature_extraction.text import CountVectorizer

vectorizer_state = CountVectorizer(vocabulary = list(sorted_dict_state_cat.keys()), low
ercase=False, binary=True)
vectorizer_state.fit(X_train['school_state'].values)

school_state_one_hot_train = vectorizer_state.transform(X_train['school_state'].values)
school_state_one_hot_cv = vectorizer_state.transform(X_cv['school_state'].values)
school_state_one_hot_test = vectorizer_state.transform(X_test['school_state'].values)

print(vectorizer_state.get_feature_names())
print("="*90)

print("The shape of school_state_one_hot_train : ",school_state_one_hot_train.shape)
print("The shape of school_state_one_hot_cv    : ",school_state_one_hot_cv.shape)
print("The shape of school_state_one_hot_test  : ",school_state_one_hot_test.shape)
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'WV', 'ME',
'HI', 'DC', 'NM', 'KS', 'IA', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'N
V', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ', 'NJ', 'OK', 'WA', 'M
A', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'N
Y', 'TX', 'CA']
==========================================================================
================
The shape of school_state_one_hot_train :  (49041, 51)
The shape of school_state_one_hot_cv    :  (24155, 51)
The shape of school_state_one_hot_test  :  (36052, 51)
```

2.2. One Hot Encoding of clean_categories :

In [39]:

```
vectorizer_cat = CountVectorizer(vocabulary = list(sorted_cat_dict.keys()), lowercase =
False, binary = True)
vectorizer_cat.fit(X_train['clean_categories'].values)

categories_one_hot_train = vectorizer_cat.transform(X_train['clean_categories'].values)
categories_one_hot_cv = vectorizer_cat.transform(X_cv['clean_categories'].values)
categories_one_hot_test = vectorizer_cat.transform(X_test['clean_categories'].values)

print(vectorizer_cat.get_feature_names())
print("="*90)

print("The shape of categories_one_hot_train: ", categories_one_hot_train.shape)
print("The shape of categories_one_hot_cv   : ", categories_one_hot_cv.shape)
print("The shape of categories_one_hot_test : ", categories_one_hot_test.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearnin
g', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
==========================================================================
================
The shape of categories_one_hot_train:  (49041, 9)
The shape of categories_one_hot_cv   :  (24155, 9)
The shape of categories_one_hot_test :  (36052, 9)
```

2.3. One Hot Encoding of clean_subcategories :

In [40]:

```
vectorizer_subcat = CountVectorizer(vocabulary = list(sorted_sub_cat_dict.keys()), lowe
rcase = False, binary = True)
vectorizer_subcat.fit(X_train['clean_subcategories'].values)

subcategories_one_hot_train = vectorizer_subcat.transform(X_train['clean_subcategories'
].values)
subcategories_one_hot_cv = vectorizer_subcat.transform(X_cv['clean_subcategories'].valu
es)
subcategories_one_hot_test = vectorizer_subcat.transform(X_test['clean_subcategories'].
values)

print(vectorizer_subcat.get_feature_names())
print("="*90)

print("The shape of subcategories_one_hot_train: ", subcategories_one_hot_train.shape)
print("The shape of subcategories_one_hot_cv   : ", subcategories_one_hot_cv.shape)
print("The shape of subcategories_one_hot_test : ", subcategories_one_hot_test.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvemen
t', 'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'Nutrition
Education', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts',
'CharacterEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Musi
c', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'A
ppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Lit
eracy']
==========================================================================
================
The shape of subcategories_one_hot_train:  (49041, 30)
The shape of subcategories_one_hot_cv   :  (24155, 30)
The shape of subcategories_one_hot_test :  (36052, 30)
```

2.4. One Hot Encoding of project_grade_category :

In [41]:

```
my_counter = Counter()

for grades in project_data['project_grade_category'].values:
    my_counter.update(grades.split())

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

In [42]:

```
vectorizer_grade = CountVectorizer(vocabulary=list(sorted_project_grade_category.keys
()), lowercase=False, binary=True)
vectorizer_grade.fit(X_train['project_grade_category'].values)

project_grade_category_one_hot_train = vectorizer_grade.transform(X_train['project_grad
e_category'].values)
project_grade_category_one_hot_cv = vectorizer_grade.transform(X_cv['project_grade_cate
gory'].values)
project_grade_category_one_hot_test = vectorizer_grade.transform(X_test['project_grade_
category'].values)

print(vectorizer_grade.get_feature_names())
print("="*80)


print("The shape of project_grade_category_one_hot_train: ", project_grade_category_one
_hot_train.shape)
print("The shape of project_grade_category_one_hot_cv   : ", project_grade_category_one
_hot_cv.shape)
print("The shape of project_grade_category_one_hot_test : ", project_grade_category_one
_hot_test.shape)
```

```
['Grades-9-12', 'Grades-6-8', 'Grades-3-5', 'Grades-PreK-2']
================================================================================
======
The shape of project_grade_category_one_hot_train:  (49041, 4)
The shape of project_grade_category_one_hot_cv   :  (24155, 4)
The shape of project_grade_category_one_hot_test :  (36052, 4)
```

2.5. One Hot Encoding of teacher_prefix :

In [43]:

```
my_counter = Counter()

for prefix in project_data['teacher_prefix'].values:
    prefix = str(prefix)
    my_counter.update(prefix.split())

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

In [44]:

```
vectorizer_prefix = CountVectorizer(vocabulary=list(sorted_teacher_prefix.keys()), lowe
rcase=False, binary=True)
vectorizer_prefix.fit(X_train['teacher_prefix'].values.astype('unicode'))

teacher_prefix_one_hot_train = vectorizer_prefix.transform(X_train['teacher_prefix'].va
lues.astype('unicode'))
teacher_prefix_one_hot_cv = vectorizer_prefix.transform(X_cv['teacher_prefix'].values.a
stype('unicode'))
teacher_prefix_one_hot_test = vectorizer_prefix.transform(X_test['teacher_prefix'].valu
es.astype('unicode'))

print(vectorizer_prefix.get_feature_names())
print("="*80)

print("The shape of teacher_prefix_one_hot_train: ",teacher_prefix_one_hot_train.shape)
print("The shape of teacher_prefix_one_hot_cv   : ",teacher_prefix_one_hot_cv.shape)
print("The shape of teacher_prefix_one_hot_test : ",teacher_prefix_one_hot_test.shape)
```

```
['nan', 'Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
================================================================================
======
The shape of teacher_prefix_one_hot_train:  (49041, 6)
The shape of teacher_prefix_one_hot_cv   :  (24155, 6)
The shape of teacher_prefix_one_hot_test :  (36052, 6)
```

3. Vectorizing Text data:

3.1. Bag of Words (BOW) with bi-grams with min_df=10 and max_features=5000)

A. Preprocessed essays:

A.1. Bag of word - X_train

In [45]:

```
# We are considering the bigrams of words which appeared in at least 10 documents(rows
 or projects) of max_feat = 5000
vectorizer_bow_essay = CountVectorizer(ngram_range =(2,2) ,min_df=10, max_features = 50
00)
vectorizer_bow_essay.fit(X_train['preprocessed_essays'])

essay_BOW_train = vectorizer_bow_essay.transform(X_train['preprocessed_essays'])
print("The shape of Data matrix after processing: ", essay_BOW_train.shape)
```

```
The shape of Data matrix after processing:  (49041, 5000)
```

A.2. Bag of word - X_cv

In [46]:

```
essay_BOW_cv = vectorizer_bow_essay.transform(X_cv['preprocessed_essays'])
print("The shape of Data matrix after processing:: ", essay_BOW_cv.shape)
```

```
The shape of Data matrix after processing::  (24155, 5000)
```

A.3. Bag of word - X_test

In [47]:

```
essay_BOW_test = vectorizer_bow_essay.transform(X_test['preprocessed_essays'])
print("The shape of Data matrix after processing:: ", essay_BOW_test.shape)
```

```
The shape of Data matrix after processing::  (36052, 5000)
```

B. Preprocessed titles:

B.1. Bag of word - X_train

In [48]:

```
# We are considering the bigrams of words which appeared in at least 10 documents(rows
 or projects) of max_feat = 5000

vectorizer_bow_title = CountVectorizer(ngram_range =(2,2) ,min_df=10, max_features = 50
00)
vectorizer_bow_title.fit(X_train['preprocessed_title'])

title_BOW_train = vectorizer_bow_title.transform(X_train['preprocessed_title'])
print("The shape of Data matrix after processing: ", title_BOW_train.shape)
```

```
The shape of Data matrix after processing:  (49041, 1233)
```

B.2. Bag of word - X_cv

In [49]:

```
title_BOW_cv = vectorizer_bow_title.transform(X_cv['preprocessed_title'])
print("The shape of Data matrix after processing: ", title_BOW_cv.shape)
```

```
The shape of Data matrix after processing:  (24155, 1233)
```

B.3. Bag of word - X_test

In [50]:

```
title_BOW_test = vectorizer_bow_title.transform(X_test['preprocessed_title'])
print("The shape of Data matrix after processing: ", title_BOW_test.shape)
```

```
The shape of Data matrix after processing:  (36052, 1233)
```

3.2. TFIDF vectorizer with bi-grams with min_df=10 and max_features=5000

A. Preprocessed essays:

A.1 TFIDF of X_train

In [51]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_essay = TfidfVectorizer(ngram_range = (2,2) ,min_df=10, max_features=
5000)

vectorizer_tfidf_essay.fit(X_train['preprocessed_essays'])

essay_tfidf_train = vectorizer_tfidf_essay.transform(X_train['preprocessed_essays'])
print("The shape of Data matrix after processing: ", essay_tfidf_train.shape)
```

```
The shape of Data matrix after processing:  (49041, 5000)
```

A.2 TFIDF of X_cv

In [52]:

```
essay_tfidf_cv = vectorizer_tfidf_essay.transform(X_cv['preprocessed_essays'])
print("The shape of Data matrix after processing: ", essay_tfidf_cv.shape)
```

```
The shape of Data matrix after processing:  (24155, 5000)
```

A.3 TFIDF of X_test

In [53]:

```
essay_tfidf_test = vectorizer_tfidf_essay.transform(X_test['preprocessed_essays'])
print("The shape of Data matrix after processing: ", essay_tfidf_test.shape)
```

```
The shape of Data matrix after processing:  (36052, 5000)
```

B. Preprocessed titles:

B.1 TFIDF of X_train

In [54]:

```
vectorizer_tfidf_title = TfidfVectorizer(ngram_range = (2,2) ,min_df=10, max_features=
5000)
vectorizer_tfidf_title.fit(X_train['preprocessed_title'])

title_tfidf_train = vectorizer_tfidf_title.transform(X_train['preprocessed_title'])
print("The shape of Data matrix after processing: ", title_tfidf_train.shape)
```

```
The shape of Data matrix after processing:  (49041, 1233)
```

B.2 TFIDF of X_cv

In [55]:

```
title_tfidf_cv = vectorizer_tfidf_title.transform(X_cv['preprocessed_title'])
print("The shape of Data matrix after processing: ", title_tfidf_cv.shape)
```

```
The shape of Data matrix after processing:  (24155, 1233)
```

B.3 TFIDF of X_test

In [56]:

```
title_tfidf_test = vectorizer_tfidf_title.transform(X_test['preprocessed_title'])
print("The shape of Data matrix after processing: ", title_tfidf_test.shape)
```

```
The shape of Data matrix after processing:  (36052, 1233)
```

3.3 Using pre-trained :-- AVG W2V

In [57]:

```
# 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 [58]:

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

```
Loading Glove Model

1917495it [10:06, 3160.09it/s]

Done. 1917495  words loaded!
```

In [59]:

```
# Lets find the total number of words present in training essay

total_words_train_essay = []
for words in tqdm(X_train['preprocessed_essays']):
    total_words_train_essay.extend(words.split(' '))

print("Total words in the train essay corpus: ", len(total_words_train_essay))
```

```
100%|████████████████████████████████████████████████████████████████████
█████| 49041/49041 [00:39<00:00, 1227.01it/s]

Total words in the train essay corpus:  6704548
```

In [60]:

```
# finding the unique words in the train essay corpus

unique_words_train_essay_corpus = set(total_words_train_essay)
print("The unique words in Train essay corpus : ", len(unique_words_train_essay_corpus
))
```

```
The unique words in Train essay corpus :  41176
```

In [61]:

```
# Common words present in both Globe

common_word = set(model.keys()).intersection(unique_words_train_essay_corpus)

print("The number of words present in Glove vectors and in our corpus ",len(common_word
),"& is {} of train essay corpus"
      .format(np.round(len(common_word)/len(unique_words_train_essay_corpus)*100,3)))
```

```
The number of words present in Glove vectors and in our corpus  37753 & is
91.687 of train essay corpus
```

In [62]:

```
word2vec_corpus_train = {}

words_glove = set(model.keys())

for i in unique_words_train_essay_corpus:
    if i in words_glove:
        word2vec_corpus_train[i] = model[i]

print("Word2Vec length is ", len(word2vec_corpus_train))
```

```
Word2Vec length is  37753
```

In [63]:

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

import pickle
with open('glove_vectors','wb') as f :       # glove_vectors is our filename
    pickle.dump(word2vec_corpus_train,f)
```

In [64]:

```
# stronging variables into pickle files python:
# http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file

with open("glove_vectors", 'rb') as f :
    model = pickle.load(f)
    glove_words = set(model.keys())
```

A. Preprocessed essays:

A.1 Avg w2v of X_train

In [65]:

```
# Computing average w2v of train essay
essay_avgw2v_train = []

for sentence in tqdm(X_train['preprocessed_essays']):  # for each sentence
    vector = np.zeros(300)  # as word vectors are of zero length  # 300 dimensions.
    count_words = 0         # num of words with a valid vector in the sentence.

    for word in sentence.split(): # for each word in a sentence
        if word in glove_words:
            vector = vector + model[word]
            count_words = count_words + 1
    if count_words != 0:
        vector = vector/count_words

    essay_avgw2v_train.append(vector)

print(len(essay_avgw2v_train))
print(len(essay_avgw2v_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████
█████| 49041/49041 [00:23<00:00, 2076.64it/s]

49041
300
```

A.2 Avg w2v of X_cv

In [66]:

```
essay_avgw2v_cv = []

for sentence in tqdm(X_cv['preprocessed_essays']):  # for each sentence
    vector = np.zeros(300)  # as word vectors are of zero length  # 300 dimensions.
    count_words = 0         # num of words with a valid vector in the sentence.

    for word in sentence.split(): # for each word in a sentence
        if word in glove_words:
            vector = vector + model[word]
            count_words = count_words + 1
    if count_words != 0:
        vector = vector/count_words

    essay_avgw2v_cv.append(vector)

print(len(essay_avgw2v_cv))
print(len(essay_avgw2v_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█████| 24155/24155 [00:11<00:00, 2017.40it/s]

24155
300
```

A.3 Avg w2v of X_test

In [67]:

```
essay_avgw2v_test = []

for sentence in tqdm(X_test['preprocessed_essays']):  # for each sentence
    vector = np.zeros(300)  # as word vectors are of zero length  # 300 dimensions.
    count_words = 0         # num of words with a valid vector in the sentence.

    for word in sentence.split(): # for each word in a sentence
        if word in glove_words:
            vector = vector + model[word]
            count_words = count_words + 1
    if count_words != 0:
        vector = vector/count_words

    essay_avgw2v_test.append(vector)

print(len(essay_avgw2v_test))
print(len(essay_avgw2v_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█████| 36052/36052 [00:17<00:00, 2101.07it/s]

36052
300
```

B. Preprocessed titles:

B.1 Avg w2v of X_train

In [68]:

```
# Computing average w2v of train title
title_avgw2v_train = []

for sentence in tqdm(X_train['preprocessed_title']):  # for each sentence
    vector = np.zeros(300)  # as word vectors are of zero length  # 300 dimensions.
    count_words = 0         # num of words with a valid vector in the sentence.

    for word in sentence.split(): # for each word in a sentence
        if word in glove_words:
            vector = vector + model[word]
            count_words = count_words + 1
    if count_words != 0:
        vector = vector/count_words

    title_avgw2v_train.append(vector)

print(len(title_avgw2v_train))
print(len(title_avgw2v_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████| 49041/49041 [00:01<00:00, 24570.67it/s]

49041
300
```

B.2 Avg w2v of X_cv

In [69]:

```
title_avgw2v_cv = []

for sentence in tqdm(X_cv['preprocessed_title']):  # for each sentence
    vector = np.zeros(300)  # as word vectors are of zero length  # 300 dimensions.
    count_words = 0         # num of words with a valid vector in the sentence.

    for word in sentence.split(): # for each word in a sentence
        if word in glove_words:
            vector = vector + model[word]
            count_words = count_words + 1
    if count_words != 0:
        vector = vector/count_words

    title_avgw2v_cv.append(vector)

print(len(title_avgw2v_cv))
print(len(title_avgw2v_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████| 24155/24155 [00:00<00:00, 31520.77it/s]

24155
300
```

B.3 Avg w2v of X_test

In [70]:

```
title_avgw2v_test = []

for sentence in tqdm(X_test['preprocessed_title']):  # for each sentence
    vector = np.zeros(300)  # as word vectors are of zero length  # 300 dimensions.
    count_words = 0         # num of words with a valid vector in the sentence.

    for word in sentence.split(): # for each word in a sentence
        if word in glove_words:
            vector = vector + model[word]
            count_words = count_words + 1
    if count_words != 0:
        vector = vector/count_words

    title_avgw2v_test.append(vector)

print(len(title_avgw2v_test))
print(len(title_avgw2v_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████| 36052/36052 [00:01<00:00, 27367.54it/s]

36052
300
```

3.3 Using pre-trained :-- TFIDF weighted avg w2v

A. Preprocessed essays:

In [71]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]

tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_essays'])
# we are converting a dictionary with features 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())
```

A.1 TFIDF weighted w2v of X_train

In [72]:

```
# TFIDF weighted W2W
# computing TFIDF weighted for pre-processed essay train
essay_tfidf_w2v_train = []   # to store the tfidf w2v of each sentence in list format.

for sentence in tqdm(X_train['preprocessed_essays']):
    vector = np.zeros(300)       # length of word vectors
    tf_idf_weight = 0            # num of words with a valid vector in the sentence.

    for word in sentence.split():  # for each word in the 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((sent
ence.count(word)/len(sentence.split()
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # tf
idf values of each word.
            vector = vector + (vec * tf_idf)
            tf_idf_weight = tf_idf_weight + tf_idf
    if tf_idf_weight != 0:
        vector = vector/tf_idf_weight
    essay_tfidf_w2v_train.append(vector)

print(len(essay_tfidf_w2v_train))
print(len(essay_tfidf_w2v_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██████| 49041/49041 [03:10<00:00, 256.91it/s]

49041
300
```

A.2 TFIDF weighted w2v of X_cv

In [73]:

```
essay_tfidf_w2v_cv = []   # to store the tfidf w2v of each sentence in list format.

for sentence in tqdm(X_cv['preprocessed_essays']):
    vector = np.zeros(300)       # length of word vectors
    tf_idf_weight = 0            # num of words with a valid vector in the sentence.

    for word in sentence.split():  # for each word in the 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((sent
ence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # tf
idf values of each word.
            vector = vector + (vec * tf_idf)
            tf_idf_weight = tf_idf_weight + tf_idf
    if tf_idf_weight != 0:
        vector = vector/tf_idf_weight
    essay_tfidf_w2v_cv.append(vector)

print(len(essay_tfidf_w2v_cv))
print(len(essay_tfidf_w2v_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██████| 24155/24155 [01:35<00:00, 254.02it/s]

24155
300
```

A.3 TFIDF weighted w2v of X_test

In [74]:

```
essay_tfidf_w2v_test = []   # to store the tfidf w2v of each sentence in list format.

for sentence in tqdm(X_test['preprocessed_essays']):
    vector = np.zeros(300)       # length of word vectors
    tf_idf_weight = 0            # num of words with a valid vector in the sentence.

    for word in sentence.split():  # for each word in the 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((sent
ence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # tf
idf values of each word.
            vector = vector + (vec * tf_idf)
            tf_idf_weight = tf_idf_weight + tf_idf
    if tf_idf_weight != 0:
        vector = vector/tf_idf_weight
    essay_tfidf_w2v_test.append(vector)

print(len(essay_tfidf_w2v_test))
print(len(essay_tfidf_w2v_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██████| 36052/36052 [02:27<00:00, 244.22it/s]

36052
300
```

B. Preprocessed titles:

B.1 TFIDF weighted w2v of X_train

In [75]:

```
title_tfidf_w2v_train = []   # to store the tfidf w2v of each sentence in list format.

for sentence in tqdm(X_train['preprocessed_title']):
    vector = np.zeros(300)       # length of word vectors
    tf_idf_weight = 0            # num of words with a valid vector in the sentence.

    for word in sentence.split():  # for each word in the 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((sent
ence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # tf
idf values of each word.
            vector = vector + (vec * tf_idf)
            tf_idf_weight = tf_idf_weight + tf_idf
    if tf_idf_weight != 0:
        vector = vector/tf_idf_weight
    title_tfidf_w2v_train.append(vector)

print(len(title_tfidf_w2v_train))
print(len(title_tfidf_w2v_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████| 49041/49041 [00:03<00:00, 14207.05it/s]

49041
300
```

B.2 TFIDF weighted w2v of X_cv

In [76]:

```
title_tfidf_w2v_cv = []   # to store the tfidf w2v of each sentence in list format.

for sentence in tqdm(X_cv['preprocessed_title']):
    vector = np.zeros(300)       # length of word vectors
    tf_idf_weight = 0            # num of words with a valid vector in the sentence.

    for word in sentence.split():  # for each word in the 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((sent
ence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # tf
idf values of each word.
            vector = vector + (vec * tf_idf)
            tf_idf_weight = tf_idf_weight + tf_idf
    if tf_idf_weight != 0:
        vector = vector/tf_idf_weight
    title_tfidf_w2v_cv.append(vector)

print(len(title_tfidf_w2v_cv))
print(len(title_tfidf_w2v_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████| 24155/24155 [00:01<00:00, 14101.45it/s]

24155
300
```

B.3 TFIDF weighted w2v of X_test

In [77]:

```
title_tfidf_w2v_test = []   # to store the tfidf w2v of each sentence in list format.

for sentence in tqdm(X_test['preprocessed_title']):
    vector = np.zeros(300)       # length of word vectors
    tf_idf_weight = 0            # num of words with a valid vector in the sentence.

    for word in sentence.split():  # for each word in the 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((sent
ence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # tf
idf values of each word.
            vector = vector + (vec * tf_idf)
            tf_idf_weight = tf_idf_weight + tf_idf
    if tf_idf_weight != 0:
        vector = vector/tf_idf_weight
    title_tfidf_w2v_test.append(vector)

print(len(title_tfidf_w2v_test))
print(len(title_tfidf_w2v_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████| 36052/36052 [00:02<00:00, 14309.48it/s]

36052
300
```

4. Vectorizing Numerical features:

4.1. price

In [78]:

```
resource_data.head()
```

Out[78]:

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

In [79]:

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

price_data.head()
```

Out[79]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21
2	p000003	298.97	4
3	p000004	1113.69	98
4	p000005	485.99	8

In [80]:

```
# We need to join the X_train, X_cv and X_test with price_data to proceed further.

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

In [81]:

```
X_train.head(2)
```

Out[81]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_s
0	104774	p099754	c2f954198b175005a18a9e66e6603728	Mrs.	NY
1	44872	p253412	43663cf9f0fd3f4932504aee9debc49e	Mrs.	CA

2 rows × 25 columns

In [82]:

```
from sklearn.preprocessing import Normalizer

normalizer = Normalizer(norm='l1')
normalizer.fit(X_train['price'].values.reshape(1,-1))
```

Out[82]:

```
Normalizer(copy=True, norm='l1')
```

In [83]:

```
price_train = normalizer.transform(X_train['price'].values.reshape(1,-1)).T
price_cv = normalizer.transform(X_cv['price'].values.reshape(1,-1)).T
price_test = normalizer.transform(X_test['price'].values.reshape(1,-1)).T
```

In [84]:

```
print("After vectorizations")
print(price_train.shape, y_train.shape)
print(price_cv.shape, y_cv.shape)
print(price_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

4.2. Quantity

In [85]:

```
normalizer.fit(X_train['quantity'].values.reshape(1,-1))
```

Out[85]:

```
Normalizer(copy=True, norm='l1')
```

In [86]:

```
quantity_train = normalizer.transform(X_train['quantity'].values.reshape(1,-1)).T
quantity_cv = normalizer.transform(X_cv['quantity'].values.reshape(1,-1)).T
quantity_test = normalizer.transform(X_test['quantity'].values.reshape(1,-1)).T
```

In [87]:

```
print("After vectorizations")
print(quantity_train.shape, y_train.shape)
print(quantity_cv.shape, y_cv.shape)
print(quantity_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

In [88]:

```
X_train.columns.values
```

Out[88]:

```
array(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects',
       'project_grade_category', 'clean_categories',
       'clean_subcategories', 'preprocessed_essays', 'essay_word_count',
       'sentiment_neg', 'sentiment_neu', 'sentiment_pos',
       'sentiment_compound', 'preprocessed_title', 'title_word_count',
       'price', 'quantity'], dtype=object)
```

4.3. Teacher_number_of_previously_posted_projects

In [89]:

```
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1
,-1))
```

Out[89]:

```
Normalizer(copy=True, norm='l1')
```

In [90]:

```
pre_posted_project_train = normalizer.transform(X_train['teacher_number_of_previously_p
osted_projects']
.values.reshape(1,-1)).T

pre_posted_project_cv = normalizer.transform(X_cv['teacher_number_of_previously_posted_
projects']
.values.reshape(1,-1)).T

pre_posted_project_test = normalizer.transform(X_test['teacher_number_of_previously_pos
ted_projects']
.values.reshape(1,-1)).T
```

In [91]:

```
print("After vectorizations")
print(pre_posted_project_train.shape, y_train.shape)
print(pre_posted_project_cv.shape, y_train.shape)
print(pre_posted_project_test.shape, y_train.shape)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (49041,)
(36052, 1) (49041,)
```

4.4. essay_word_count

In [92]:

```
normalizer.fit(X_train['essay_word_count'].values.reshape(1,-1))
```

Out[92]:

```
Normalizer(copy=True, norm='l1')
```

In [93]:

```
essay_count_train = normalizer.transform(X_train['essay_word_count'].values.reshape(1,-
1)).T
essay_count_cv = normalizer.transform(X_cv['essay_word_count'].values.reshape(1,-1)).T
essay_count_test = normalizer.transform(X_test['essay_word_count'].values.reshape(1,-1
)).T
```

In [94]:

```
print("After vectorizations")
print(essay_count_train.shape, y_train.shape)
print(essay_count_cv.shape, y_train.shape)
print(essay_count_test.shape, y_train.shape)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (49041,)
(36052, 1) (49041,)
```

4.5. title_word_count

In [95]:

```
normalizer.fit(X_train['title_word_count'].values.reshape(1,-1))
```

Out[95]:

```
Normalizer(copy=True, norm='l1')
```

In [96]:

```
title_count_train = normalizer.transform(X_train['title_word_count'].values.reshape(1,-
1)).T
title_count_cv = normalizer.transform(X_cv['title_word_count'].values.reshape(1,-1)).T
title_count_test = normalizer.transform(X_test['title_word_count'].values.reshape(1,-1
)).T
```

In [97]:

```
print("After vectorizations")
print(title_count_train.shape, y_train.shape)
print(title_count_cv.shape, y_train.shape)
print(title_count_test.shape, y_train.shape)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (49041,)
(36052, 1) (49041,)
```

4.6. sentiment_neg

In [98]:

```
normalizer.fit(X_train['sentiment_neg'].values.reshape(1,-1))
```

Out[98]:

```
Normalizer(copy=True, norm='l1')
```

In [99]:

```
sentiment_neg_train = normalizer.transform(X_train['sentiment_neg'].values.reshape(1,-1
)).T
sentiment_neg_cv = normalizer.transform(X_cv['sentiment_neg'].values.reshape(1,-1)).T
sentiment_neg_test = normalizer.transform(X_test['sentiment_neg'].values.reshape(1,-1))
.T
```

In [100]:

```
print("After vectorizations")
print(sentiment_neg_train.shape, y_train.shape)
print(sentiment_neg_cv.shape, y_train.shape)
print(sentiment_neg_test.shape, y_train.shape)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (49041,)
(36052, 1) (49041,)
```

4.7. sentiment_neu

In [101]:

```
normalizer.fit(X_train['sentiment_neu'].values.reshape(1,-1))
```

Out[101]:

```
Normalizer(copy=True, norm='l1')
```

In [102]:

```
sentiment_neu_train = normalizer.transform(X_train['sentiment_neu'].values.reshape(1,-1
)).T
sentiment_neu_cv = normalizer.transform(X_cv['sentiment_neu'].values.reshape(1,-1)).T
sentiment_neu_test = normalizer.transform(X_test['sentiment_neu'].values.reshape(1,-1))
.T
```

In [103]:

```
print("After vectorizations")
print(sentiment_neu_train.shape, y_train.shape)
print(sentiment_neu_cv.shape, y_train.shape)
print(sentiment_neu_test.shape, y_train.shape)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (49041,)
(36052, 1) (49041,)
```

4.8. sentiment_pos

In [104]:

```
normalizer.fit(X_train['sentiment_pos'].values.reshape(1,-1))
```

Out[104]:

```
Normalizer(copy=True, norm='l1')
```

In [105]:

```
sentiment_pos_train = normalizer.transform(X_train['sentiment_pos'].values.reshape(1,-1
)).T
sentiment_pos_cv = normalizer.transform(X_cv['sentiment_pos'].values.reshape(1,-1)).T
sentiment_pos_test = normalizer.transform(X_test['sentiment_pos'].values.reshape(1,-1))
.T
```

In [106]:

```
print("After vectorizations")
print(sentiment_pos_train.shape, y_train.shape)
print(sentiment_pos_cv.shape, y_train.shape)
print(sentiment_pos_test.shape, y_train.shape)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (49041,)
(36052, 1) (49041,)
```

4.9. sentiment_compound

In [107]:

```
normalizer.fit(X_train['sentiment_compound'].values.reshape(1,-1))
```

Out[107]:

```
Normalizer(copy=True, norm='l1')
```

In [108]:

```
sentiment_compound_train = normalizer.transform(X_train['sentiment_compound'].values.re
shape(1,-1)).T
sentiment_compound_cv = normalizer.transform(X_cv['sentiment_compound'].values.reshape(
1,-1)).T
sentiment_compound_test = normalizer.transform(X_test['sentiment_compound'].values.resh
ape(1,-1)).T
```

In [109]:

```
print("After vectorizations")
print(sentiment_compound_train.shape, y_train.shape)
print(sentiment_compound_cv.shape, y_train.shape)
print(sentiment_compound_test.shape, y_train.shape)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (49041,)
(36052, 1) (49041,)
```

Assignment 5: Logistic Regression

1. **[Task-1] Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on these feature sets**

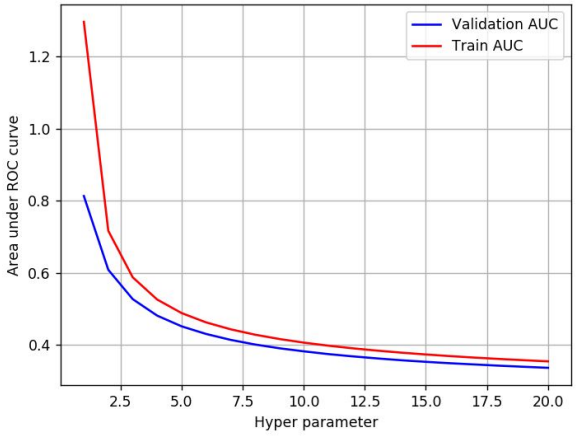
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (`BOW with bi-grams` with `min_df=10` and `max_features=5000`)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (`TFIDF with bi-grams` with `min_df=10` and `max_features=5000`)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**

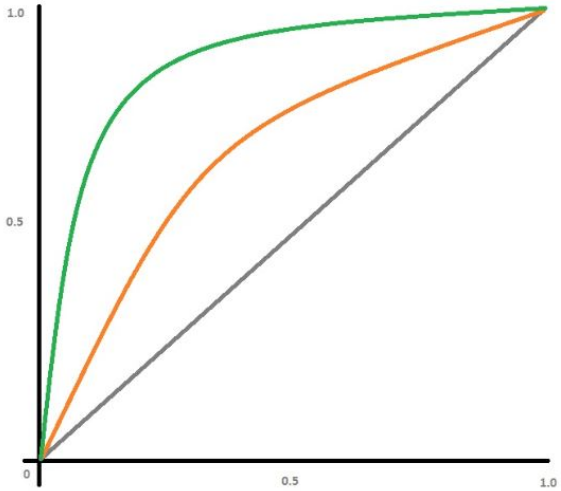
   - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Representation of results**

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



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



- Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)
(https://seaborn.pydata.org/generated/seaborn.heatmap.html)
(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

4. **[Task-2] Apply Logistic Regression on the below feature set Set 5 by finding the best hyper parameter as suggested in step 2 and step 3.**
5. Consider these set of features Set 5 :

   - **school_state** : categorical data
   - **clean_categories** : categorical data
   - **clean_subcategories** : categorical data
   - **project_grade_category** :categorical data
   - **teacher_prefix** : categorical data
   - **quantity** : numerical data
   - **teacher_number_of_previously_posted_projects** : numerical data
   - **price** : numerical data
   - **sentiment score's of each of the essay** : numerical data
   - **number of words in the title** : numerical data
   - **number of words in the combine essays** : numerical data

   And apply the Logistic regression on these features by finding the best hyper paramter as suggested in step 2 and step 3

   (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
6. **Conclusion** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

   (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library (https://seaborn.pydata.org/generated/seaborn.heatmap.html) link (http://zetcode.com/python/prettytable/)

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

Note: Data Leakage

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

5. Logistic Regression

Set 1: Categorical, Numerical features + Project_title(BOW) +Preprocessed_essay (BOW with bi-grams with min_df=10 and max_features=5000)

In [110]:

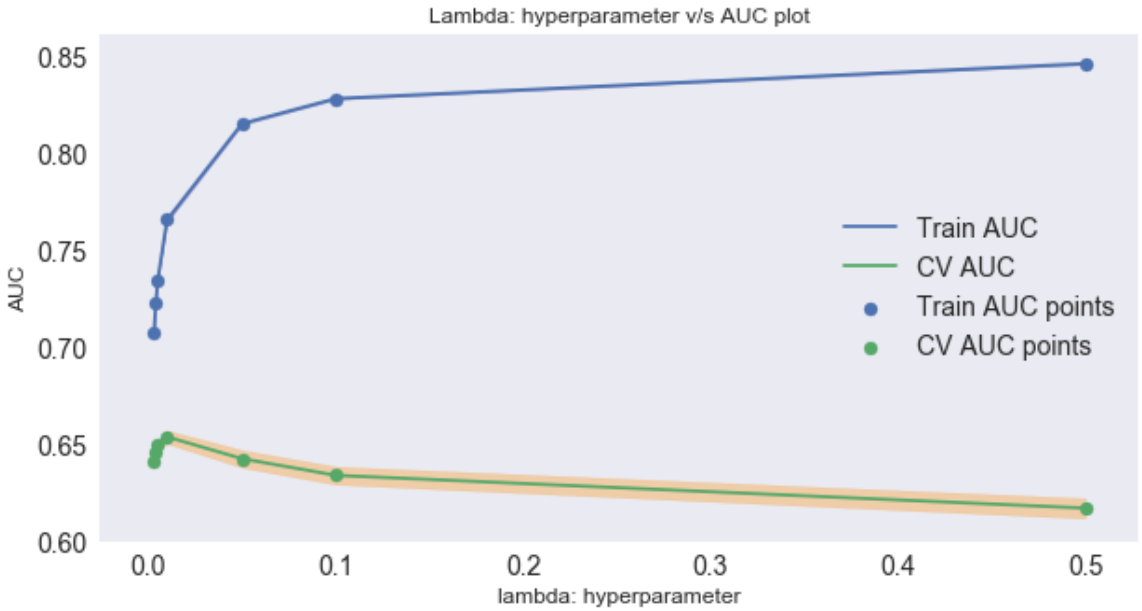
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((school_state_one_hot_train,categories_one_hot_train,subcategories_one_ho
t_train,
project_grade_category_one_hot_train,teacher_prefix_one_hot_train,price_train,quantity_
train,
pre_posted_project_train,essay_count_train,title_count_train,essay_BOW_train,title_BOW_
train)).tocsr()


X_cr = hstack((school_state_one_hot_cv,categories_one_hot_cv,subcategories_one_hot_cv,
project_grade_category_one_hot_cv,teacher_prefix_one_hot_cv,price_cv,quantity_cv,
pre_posted_project_cv,essay_count_cv,title_count_cv,essay_BOW_cv,title_BOW_cv)).tocsr()


X_te = hstack((school_state_one_hot_test,categories_one_hot_test,subcategories_one_hot_
test,
project_grade_category_one_hot_test,teacher_prefix_one_hot_test,price_test,quantity_tes
t,
pre_posted_project_test,essay_count_test,title_count_test,essay_BOW_test,title_BOW_test
)).tocsr()


print("The Final Data Matrix :----")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print('='*50)
```

```
The Final Data Matrix :----
(49041, 6338) (49041,)
(24155, 6338) (24155,)
(36052, 6338) (36052,)
==================================================
```

Set 1.1 Finding best hyperparameter using GridSearchCV (K fold Cross Validation)

In [256]:

```
lr = LogisticRegression()
parameters = {'C':[0.5, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003]}
clf = GridSearchCV(lr, parameters, cv= 5, scoring='roc_auc')
clf.fit(X_tr, y_train)
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
plt.figure(figsize=(10,5))
plt.plot(parameters['C'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')
plt.plot(parameters['C'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
3,color='darkorange')
plt.scatter(parameters['C'], train_auc, label='Train AUC points')
plt.scatter(parameters['C'], cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("lambda: hyperparameter", fontsize = 12)
plt.ylabel("AUC", fontsize = 12)
plt.title("Lambda: hyperparameter v/s AUC plot", fontsize = 12)
plt.grid()
plt.show()
```



In [257]:

```
clf.best_estimator_
```

Out[257]:

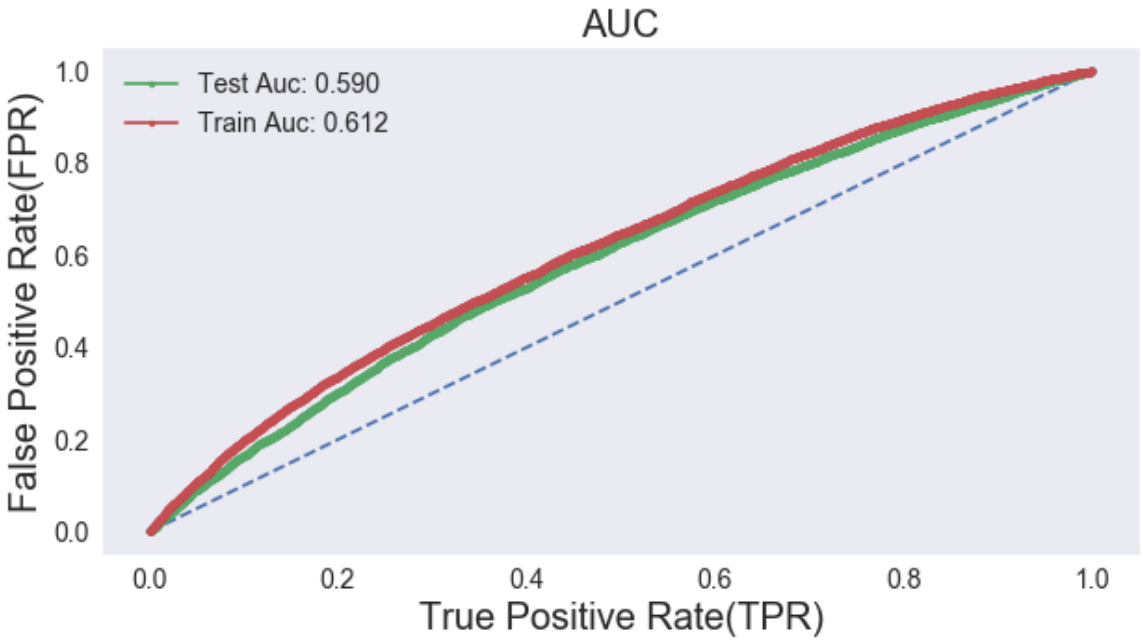
```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Best hyperparameter is 0.01

Set 1.2 Train the model using the best hyper parameter value

In [274]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
model = LogisticRegression(C = 0.01)
model.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs
# predict probabilities for train dataset.
y_train_prob = model.predict_proba(X_tr)
# keep probabilities for the positive outcome only
y_train_prob = y_train_prob[:, 1]
auc_train = roc_auc_score(y_train, y_train_prob)
train_fpr,train_tpr,train_threshold = roc_curve(y_train, y_train_prob)
# predict probabilities for test dataset.
y_test_prob = model.predict_proba(X_te)
# keep probabilities for the positive outcome only
y_test_prob  = y_test_prob [:, 1]
# calculate AUC
auc = roc_auc_score(y_test, y_test_prob )
#print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_threshold = roc_curve(y_test, y_test_prob )
# plot no skill
plt.figure(figsize=(10,5))
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr,label = 'Test Auc: %.3f' % auc, marker='.')
plt.plot(train_fpr, train_tpr,label = 'Train Auc: %.3f' % auc_train, marker='.')
plt.legend()
plt.xlabel("True Positive Rate(TPR)", fontsize = 20)
plt.ylabel("False Positive Rate(FPR)",fontsize = 20)
plt.title("AUC",fontsize = 20)
plt.grid()
# show the plot
plt.show()
```



Set 1.3 Confusion Matrix

In [259]:

```
from sklearn.metrics import confusion_matrix
```

In [260]:

```
def predict(proba, threshould, fpr, tpr):
    t = threshould[np.argmax(fpr*(1-tpr))]
# (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))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [261]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(train_threshold, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_prob, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_prob, best_t)))
```

```
====================================================================================
==========================
the maximum value of tpr*(1-fpr) 0.48218581453067516 for threshold 0.827
Train confusion matrix
[[ 5066  2360]
 [12201 29414]]
Test confusion matrix
[[ 3048  2411]
 [ 9894 20699]]
```

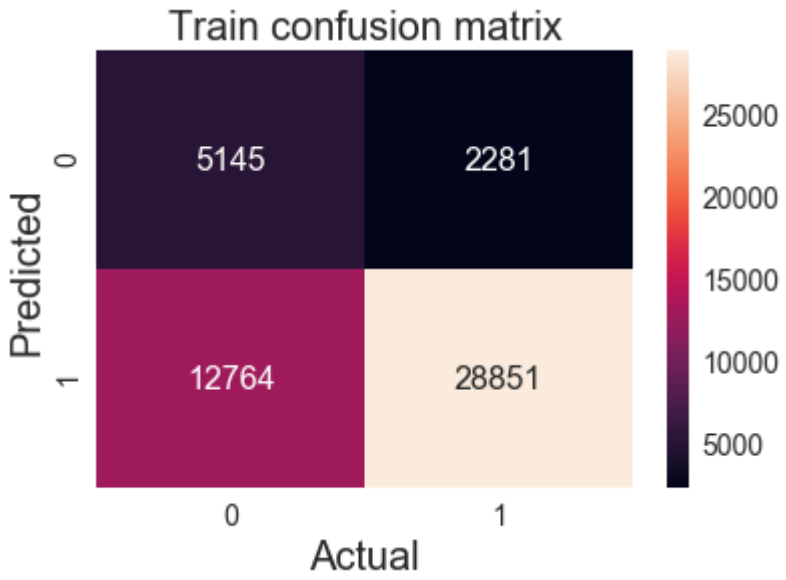
In [262]:

```
CM_df_tr = pd.DataFrame(confusion_matrix(y_train, predict(y_train_prob, train_threshold
,train_fpr, train_tpr)))
```

```
the maximum value of tpr*(1-fpr) 0.48218581453067516 for threshold 0.83
```

In [263]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(CM_df_tr, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Actual', fontsize = 20)
plt.ylabel('Predicted', fontsize = 20)
plt.title('Train confusion matrix', fontsize = 20)
plt.show()
```



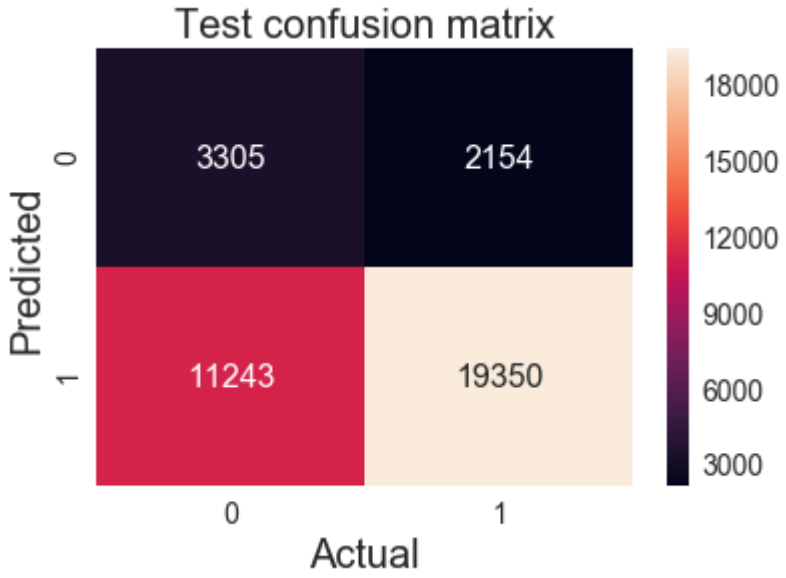
In [264]:

```
CM_df_te = pd.DataFrame(confusion_matrix(y_test, predict(y_test_prob, test_threshold,te
st_fpr, test_tpr)))
```

```
the maximum value of tpr*(1-fpr) 0.3855663648774588 for threshold 0.836
```

In [265]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(CM_df_te, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Actual', fontsize = 20)
plt.ylabel('Predicted', fontsize = 20)
plt.title('Test confusion matrix', fontsize = 20)
plt.show()
```



Set 2 : Categorical, Numerical features + Project_title(TFIDF) + Preprocessed_essay (TFIDF with bi-grams with min_df=10 and max_features=5000)

In [266]:

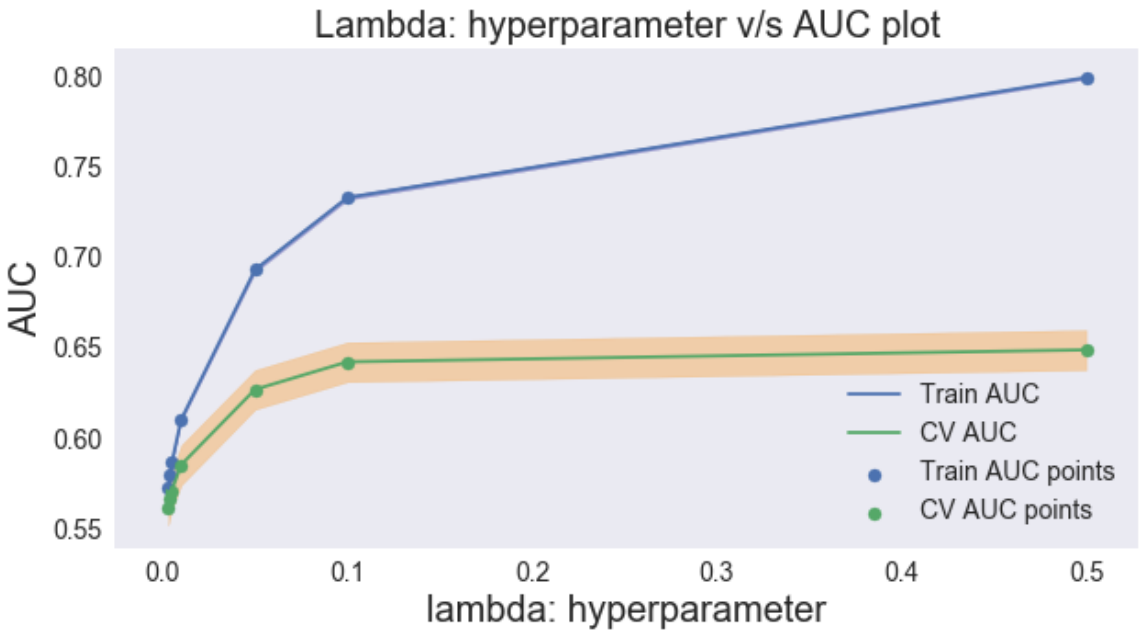
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((school_state_one_hot_train,categories_one_hot_train,subcategories_one_ho
t_train,
project_grade_category_one_hot_train,teacher_prefix_one_hot_train,price_train,quantity_
train,
pre_posted_project_train,essay_count_train,title_count_train,essay_tfidf_train,title_tf
idf_train)).tocsr()


X_cr = hstack((school_state_one_hot_cv,categories_one_hot_cv,subcategories_one_hot_cv,
project_grade_category_one_hot_cv,teacher_prefix_one_hot_cv,price_cv,quantity_cv,
pre_posted_project_cv,essay_count_cv,title_count_cv,essay_tfidf_cv,title_tfidf_cv)).toc
sr()


X_te = hstack((school_state_one_hot_test,categories_one_hot_test,subcategories_one_hot_
test,
project_grade_category_one_hot_test,teacher_prefix_one_hot_test,price_test,quantity_tes
t,
pre_posted_project_test,essay_count_test,title_count_test,essay_tfidf_test,title_tfidf_
test)).tocsr()


print("The Final Data Matrix :----")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print('='*50)
```

```
The Final Data Matrix :----
(49041, 6338) (49041,)
(24155, 6338) (24155,)
(36052, 6338) (36052,)
==================================================
```

Set 2.1 Finding best hyperparameter using GridSearchCV (K fold Cross Validation)

In [275]:

```
lr = LogisticRegression()
parameters = {'C':[0.5, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003]}
clf = GridSearchCV(lr, parameters, cv= 10, scoring='roc_auc')
clf.fit(X_tr, y_train)
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
plt.figure(figsize=(10,5))
plt.plot(parameters['C'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')
plt.plot(parameters['C'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
3,color='darkorange')
plt.scatter(parameters['C'], train_auc, label='Train AUC points')
plt.scatter(parameters['C'], cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("lambda: hyperparameter", fontsize = 20)
plt.ylabel("AUC", fontsize = 20)
plt.title("Lambda: hyperparameter v/s AUC plot", fontsize = 20)
plt.grid()
plt.show()
```



In [276]:

```
clf.best_estimator_
```

Out[276]:

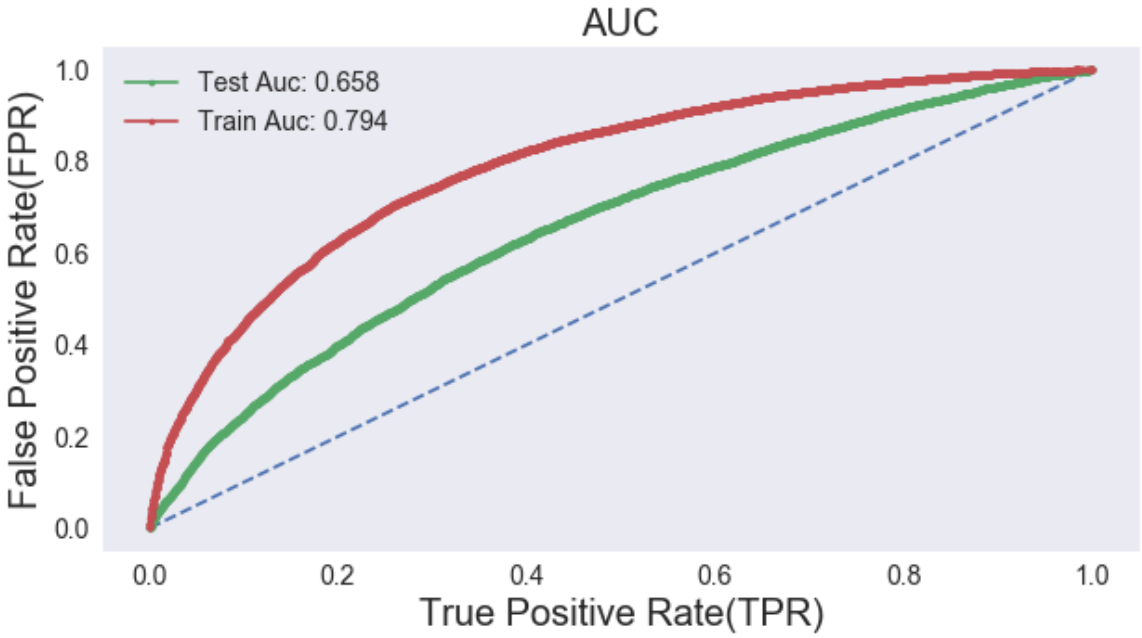
```
LogisticRegression(C=0.5, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

best hyperparameter = 0.5

Set 2.2 Train the model using the best hyper parameter value

In [278]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklea
rn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
model = LogisticRegression(C = 0.5)
model.fit(X_tr, 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
# predict probabilities for train dataset.
y_train_prob = model.predict_proba(X_tr)
# keep probabilities for the positive outcome only
y_train_prob = y_train_prob[:, 1]
auc_train = roc_auc_score(y_train, y_train_prob)
train_fpr,train_tpr,train_threshold = roc_curve(y_train, y_train_prob)
# predict probabilities for test dataset.
y_test_prob = model.predict_proba(X_te)
# keep probabilities for the positive outcome only
y_test_prob  = y_test_prob [:, 1]
# calculate AUC
auc = roc_auc_score(y_test, y_test_prob )
#print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_threshold = roc_curve(y_test, y_test_prob )
# plot no skill
plt.figure(figsize=(10,5))
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr,label = 'Test Auc: %.3f' % auc, marker='.')
plt.plot(train_fpr, train_tpr,label = 'Train Auc: %.3f' % auc_train, marker='.')
plt.legend()
plt.xlabel("True Positive Rate(TPR)", fontsize = 20)
plt.ylabel("False Positive Rate(FPR)",fontsize = 20)
plt.title("AUC",fontsize = 20)
plt.grid()
# show the plot
plt.show()
```



Set 2.3 Confusion Matrix

In [279]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(train_threshold, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_prob, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_prob, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.5229469805149282 for threshold 0.835
Train confusion matrix
[[ 5471  1955]
 [12076 29539]]
Test confusion matrix
[[ 3048  2411]
 [10053 20540]]
```

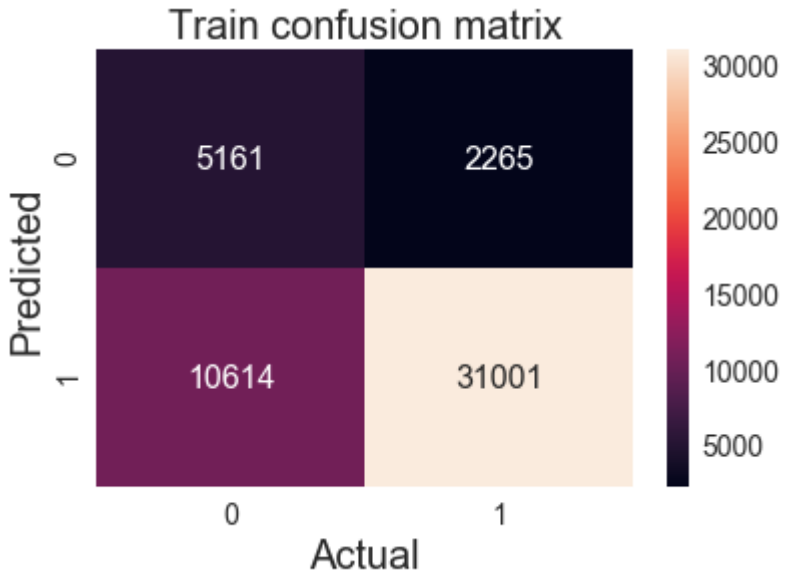
In [280]:

```
CM_df_tr = pd.DataFrame(confusion_matrix(y_train, predict(y_train_prob, train_threshold
,train_fpr, train_tpr)))
```

```
the maximum value of tpr*(1-fpr) 0.5229469805149282 for threshold 0.827
```

In [281]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(CM_df_tr, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Actual', fontsize = 20)
plt.ylabel('Predicted', fontsize = 20)
plt.title('Train confusion matrix', fontsize = 20)
plt.show()
```



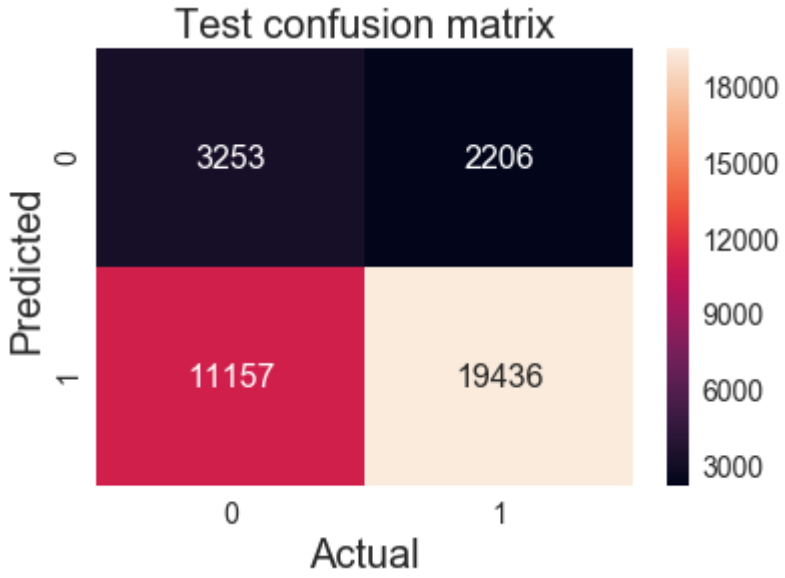
In [282]:

```
CM_df_te = pd.DataFrame(confusion_matrix(y_test, predict(y_test_prob, test_threshold,te
st_fpr, test_tpr)))
```

```
the maximum value of tpr*(1-fpr) 0.3811194724212677 for threshold 0.844
```

In [283]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(CM_df_te, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Actual', fontsize = 20)
plt.ylabel('Predicted', fontsize = 20)
plt.title('Test confusion matrix', fontsize = 20)
plt.show()
```



Set 3 : Categorical, Numerical features + Project_title(AVG W2V) + Preprocessed_essay (AVG W2V)

In [284]:

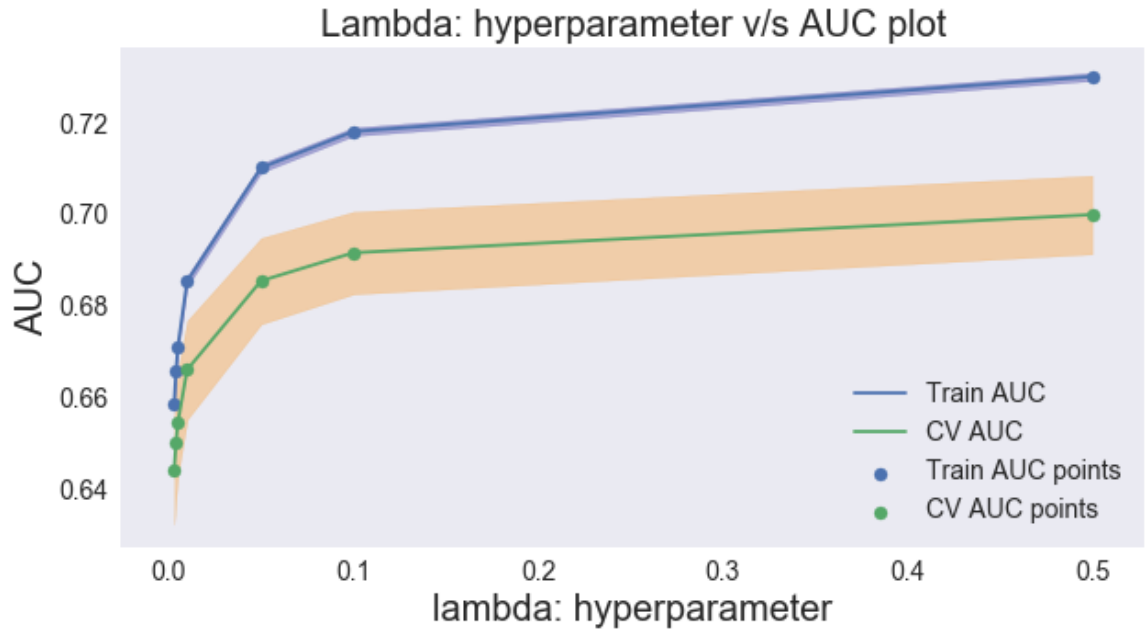
```
X_tr = hstack((school_state_one_hot_train,categories_one_hot_train,subcategories_one_ho
t_train,
project_grade_category_one_hot_train,teacher_prefix_one_hot_train,price_train,quantity_
train,
pre_posted_project_train,essay_count_train,title_count_train,essay_avgw2v_train,title_a
vgw2v_train)).tocsr()


X_cr = hstack((school_state_one_hot_cv,categories_one_hot_cv,subcategories_one_hot_cv,
project_grade_category_one_hot_cv,teacher_prefix_one_hot_cv,price_cv,quantity_cv,
pre_posted_project_cv,essay_count_cv,title_count_cv,essay_avgw2v_cv,title_avgw2v_cv)).t
ocsr()


X_te = hstack((school_state_one_hot_test,categories_one_hot_test,subcategories_one_hot_
test,
project_grade_category_one_hot_test,teacher_prefix_one_hot_test,price_test,quantity_tes
t,
pre_posted_project_test,essay_count_test,title_count_test,essay_avgw2v_test,title_avgw2
v_test)).tocsr()


print("The Final Data Matrix :----")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print('='*50)
```

```
The Final Data Matrix :----
(49041, 705) (49041,)
(24155, 705) (24155,)
(36052, 705) (36052,)
==================================================
```

Set 3.1 Finding best hyperparameter using GridSearchCV (K fold Cross Validation)

In [285]:

```
lr = LogisticRegression()
parameters = {'C':[0.5, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003]}
clf = GridSearchCV(lr, parameters, cv= 10, scoring='roc_auc')
clf.fit(X_tr, y_train)
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
plt.figure(figsize=(10,5))
plt.plot(parameters['C'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')
plt.plot(parameters['C'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
3,color='darkorange')
plt.scatter(parameters['C'], train_auc, label='Train AUC points')
plt.scatter(parameters['C'], cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("lambda: hyperparameter", fontsize = 20)
plt.ylabel("AUC", fontsize = 20)
plt.title("Lambda: hyperparameter v/s AUC plot", fontsize = 20)
plt.grid()
plt.show()
```



In [287]:

```
clf.best_estimator_
```

Out[287]:

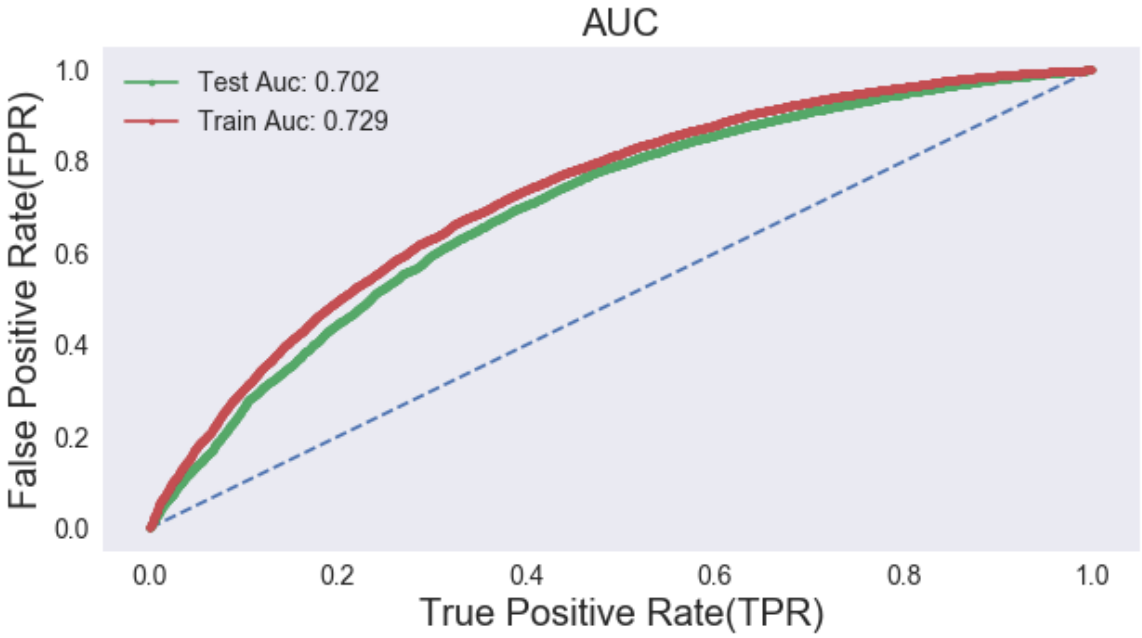
```
LogisticRegression(C=0.5, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

best hyperparameter = 0.5

Set 3.2 Train the model using the best hyper parameter value

In [288]:

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklea
rn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
model = LogisticRegression(C = 0.5)
model.fit(X_tr, 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
# predict probabilities for train dataset.
y_train_prob = model.predict_proba(X_tr)
# keep probabilities for the positive outcome only
y_train_prob = y_train_prob[:, 1]
auc_train = roc_auc_score(y_train, y_train_prob)
train_fpr,train_tpr,train_threshold = roc_curve(y_train, y_train_prob)
# predict probabilities for test dataset.
y_test_prob = model.predict_proba(X_te)
# keep probabilities for the positive outcome only
y_test_prob  = y_test_prob [:, 1]
# calculate AUC
auc = roc_auc_score(y_test, y_test_prob )
#print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_threshold = roc_curve(y_test, y_test_prob )
# plot no skill
plt.figure(figsize=(10,5))
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr,label = 'Test Auc: %.3f' % auc, marker='.')
plt.plot(train_fpr, train_tpr,label = 'Train Auc: %.3f' % auc_train, marker='.')
plt.legend()
plt.xlabel("True Positive Rate(TPR)", fontsize = 20)
plt.ylabel("False Positive Rate(FPR)",fontsize = 20)
plt.title("AUC",fontsize = 20)
plt.grid()
# show the plot
plt.show()
```



Set 3.3 Confusion Matrix

In [289]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(train_threshold, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_prob, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_prob, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.44965382498483414 for threshold 0.842
Train confusion matrix
[[ 4958  2468]
 [13588 28027]]
Test confusion matrix
[[ 3512  1947]
 [10407 20186]]
```

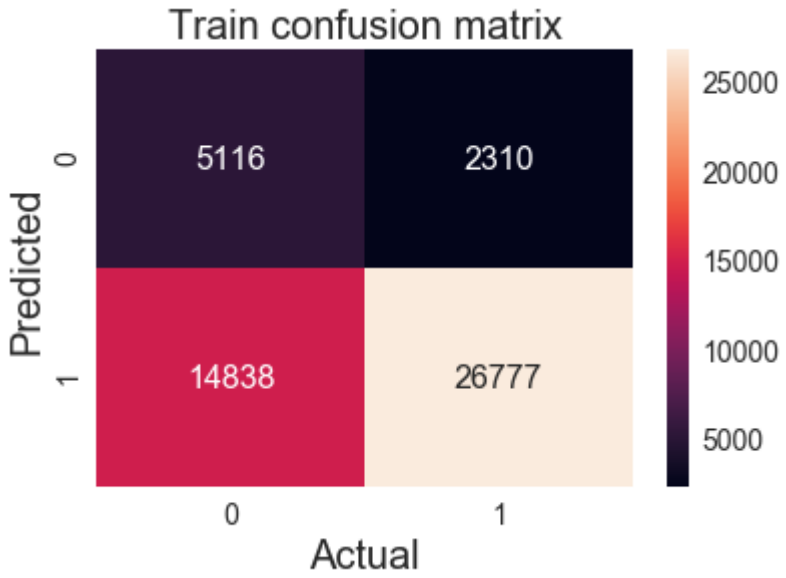
In [290]:

```
CM_df_tr = pd.DataFrame(confusion_matrix(y_train, predict(y_train_prob, train_threshold
,train_fpr, train_tpr)))
```

```
the maximum value of tpr*(1-fpr) 0.44965382498483414 for threshold 0.851
```

In [291]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(CM_df_tr, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Actual', fontsize = 20)
plt.ylabel('Predicted', fontsize = 20)
plt.title('Train confusion matrix', fontsize = 20)
plt.show()
```



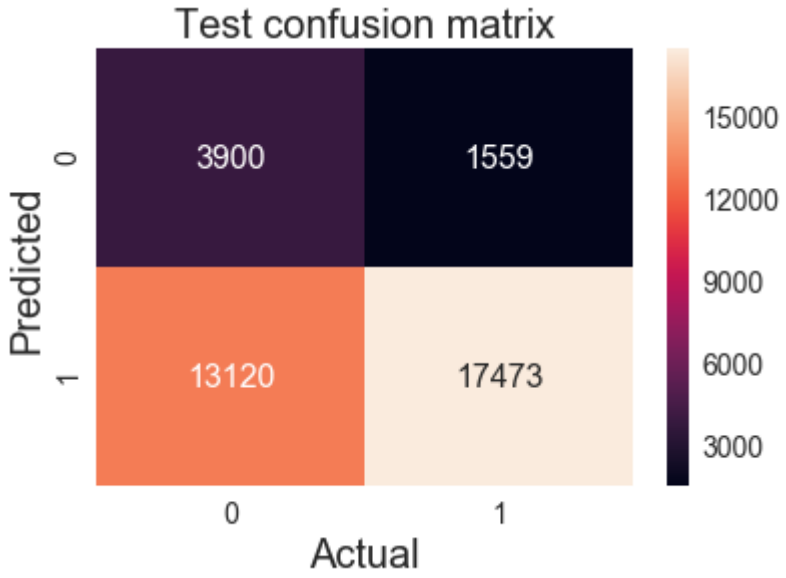
In [292]:

```
CM_df_te = pd.DataFrame(confusion_matrix(y_test, predict(y_test_prob, test_threshold,te
st_fpr, test_tpr)))
```

```
the maximum value of tpr*(1-fpr) 0.4261336250157905 for threshold 0.865
```

In [293]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(CM_df_te, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Actual', fontsize = 20)
plt.ylabel('Predicted', fontsize = 20)
plt.title('Test confusion matrix', fontsize = 20)
plt.show()
```



Set 4 : Categorical, Numerical features + Project_title(TFIDF W2V) + Preprocessed_essay (TFIDF W2V)

In [294]:

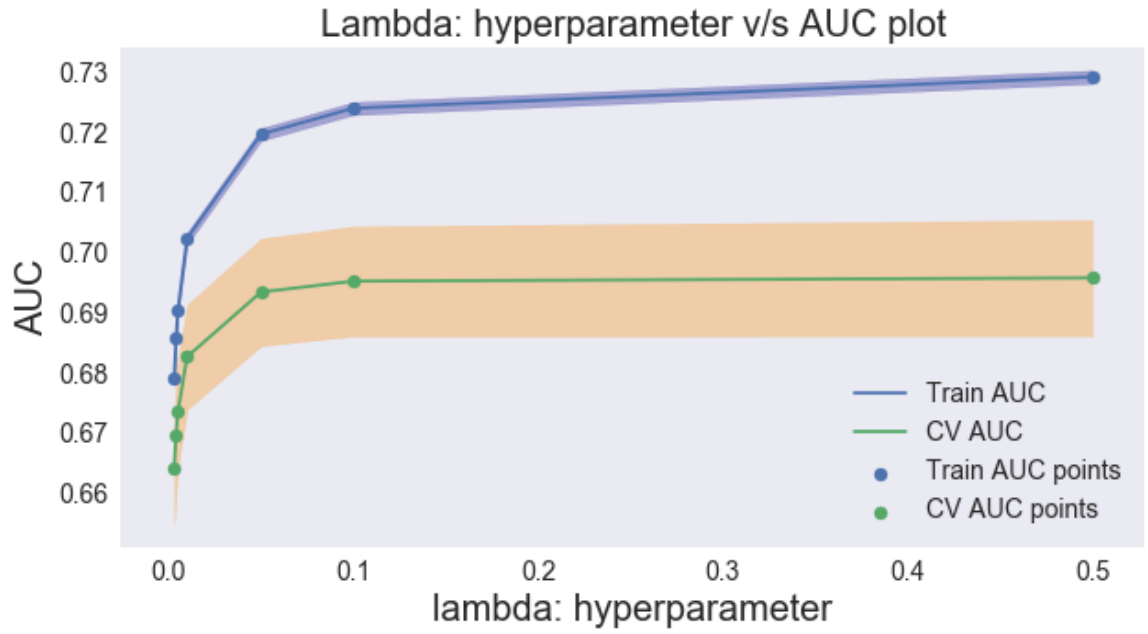
```
X_tr = hstack((school_state_one_hot_train,categories_one_hot_train,subcategories_one_ho
t_train,
project_grade_category_one_hot_train,teacher_prefix_one_hot_train,price_train,quantity_
train,
pre_posted_project_train,essay_count_train,title_count_train,essay_tfidf_w2v_train,titl
e_tfidf_w2v_train)).tocsr()


X_cr = hstack((school_state_one_hot_cv,categories_one_hot_cv,subcategories_one_hot_cv,
project_grade_category_one_hot_cv,teacher_prefix_one_hot_cv,price_cv,quantity_cv,
pre_posted_project_cv,essay_count_cv,title_count_cv,essay_tfidf_w2v_cv,title_tfidf_w2v_
cv)).tocsr()


X_te = hstack((school_state_one_hot_test,categories_one_hot_test,subcategories_one_hot_
test,
project_grade_category_one_hot_test,teacher_prefix_one_hot_test,price_test,quantity_tes
t,
pre_posted_project_test,essay_count_test,title_count_test,essay_tfidf_w2v_test,title_tf
idf_w2v_test)).tocsr()


print("The Final Data Matrix :----")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print('='*50)
```

```
The Final Data Matrix :----
(49041, 705) (49041,)
(24155, 705) (24155,)
(36052, 705) (36052,)
==================================================
```

Set 4.1 Finding best hyperparameter using GridSearchCV (K fold Cross Validation)

In [295]:

```
lr = LogisticRegression()
parameters = {'C':[0.5, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003]}
clf = GridSearchCV(lr, parameters, cv= 10, scoring='roc_auc')
clf.fit(X_tr, y_train)
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
plt.figure(figsize=(10,5))
plt.plot(parameters['C'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')
plt.plot(parameters['C'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
3,color='darkorange')
plt.scatter(parameters['C'], train_auc, label='Train AUC points')
plt.scatter(parameters['C'], cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("lambda: hyperparameter", fontsize = 20)
plt.ylabel("AUC", fontsize = 20)
plt.title("Lambda: hyperparameter v/s AUC plot", fontsize = 20)
plt.grid()
plt.show()
```



In [296]:

```
clf.best_estimator_
```

Out[296]:

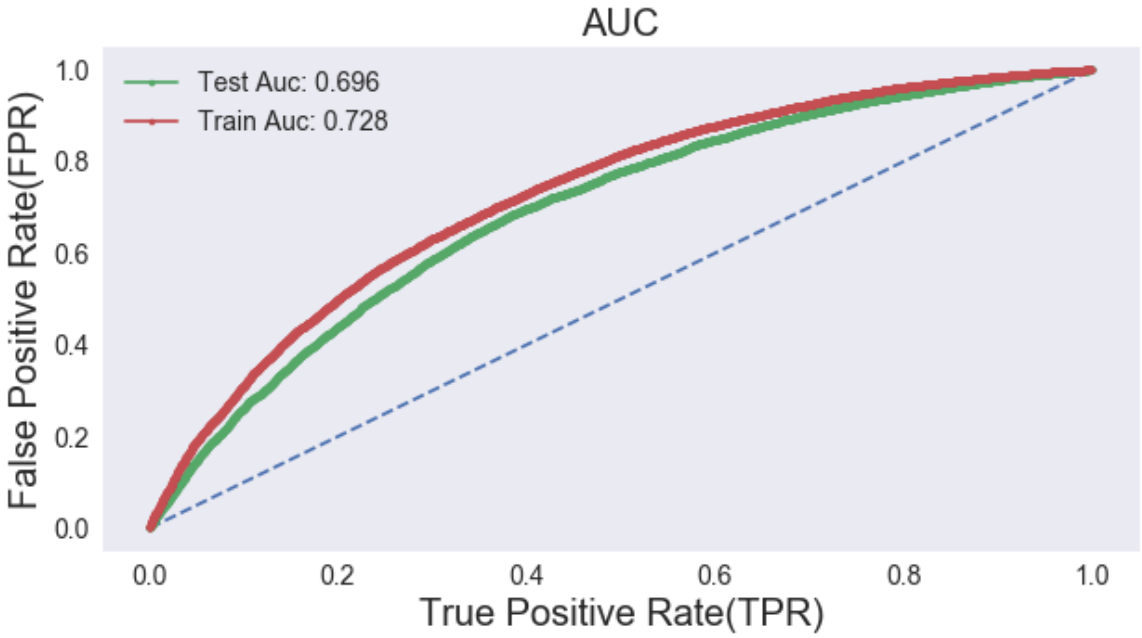
```
LogisticRegression(C=0.5, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

best hyperparameter = 0.5

Set 4.2 Train the model using the best hyper parameter value

In [297]:

```
from sklearn.metrics import roc_curve, auc
model = LogisticRegression(C = 0.5)
model.fit(X_tr, 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
# predict probabilities for train dataset.
y_train_prob = model.predict_proba(X_tr)
# keep probabilities for the positive outcome only
y_train_prob = y_train_prob[:, 1]
auc_train = roc_auc_score(y_train, y_train_prob)
train_fpr,train_tpr,train_threshold = roc_curve(y_train, y_train_prob)
# predict probabilities for test dataset.
y_test_prob = model.predict_proba(X_te)
# keep probabilities for the positive outcome only
y_test_prob  = y_test_prob [:, 1]
# calculate AUC
auc = roc_auc_score(y_test, y_test_prob )
#print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_threshold = roc_curve(y_test, y_test_prob )
# plot no skill
plt.figure(figsize=(10,5))
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr,label = 'Test Auc: %.3f' % auc, marker='.')
plt.plot(train_fpr, train_tpr,label = 'Train Auc: %.3f' % auc_train, marker='.')
plt.legend()
plt.xlabel("True Positive Rate(TPR)", fontsize = 20)
plt.ylabel("False Positive Rate(FPR)",fontsize = 20)
plt.title("AUC",fontsize = 20)
plt.grid()
# show the plot
plt.show()
```



Set 4.3 Confusion Matrix

In [298]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(train_threshold, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_prob, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_prob, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.4438440310207658 for threshold 0.842
Train confusion matrix
[[ 4987  2439]
 [14111 27504]]
Test confusion matrix
[[ 3544  1915]
 [10814 19779]]
```

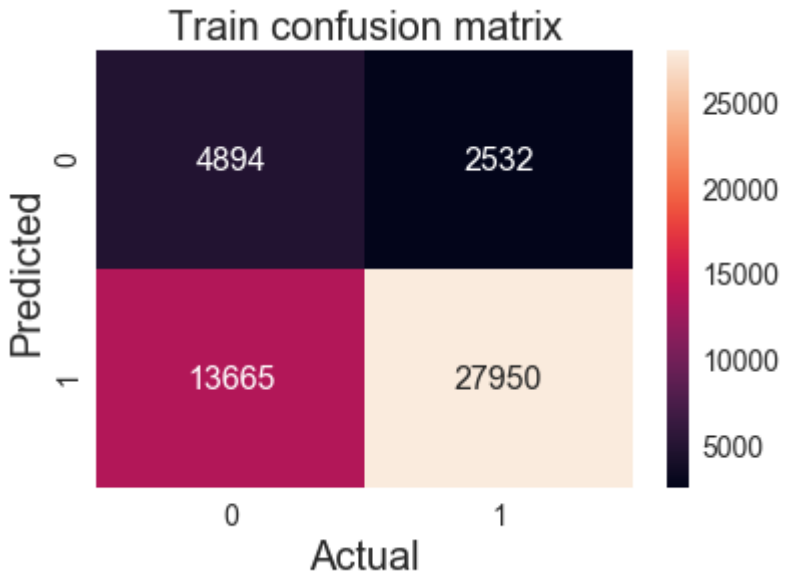
In [299]:

```
CM_df_tr = pd.DataFrame(confusion_matrix(y_train, predict(y_train_prob, train_threshold
,train_fpr, train_tpr)))
```

```
the maximum value of tpr*(1-fpr) 0.4438440310207658 for threshold 0.839
```

In [300]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(CM_df_tr, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Actual', fontsize = 20)
plt.ylabel('Predicted', fontsize = 20)
plt.title('Train confusion matrix', fontsize = 20)
plt.show()
```



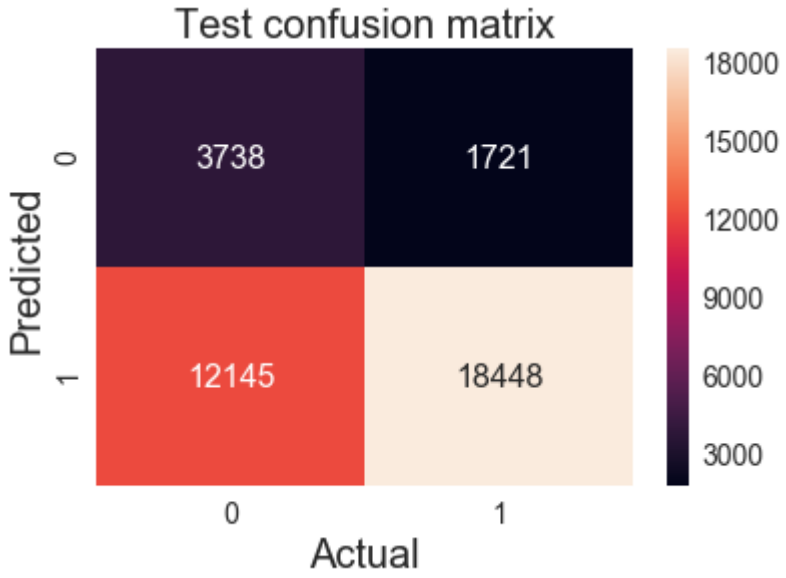
In [301]:

```
CM_df_te = pd.DataFrame(confusion_matrix(y_test, predict(y_test_prob, test_threshold,te
st_fpr, test_tpr)))
```

```
the maximum value of tpr*(1-fpr) 0.42139451160266533 for threshold 0.854
```

In [302]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(CM_df_te, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Actual', fontsize = 20)
plt.ylabel('Predicted', fontsize = 20)
plt.title('Test confusion matrix', fontsize = 20)
plt.show()
```



Set 5 : Categorical features, Numerical features & Sentiment Score (without text features)

In [303]:

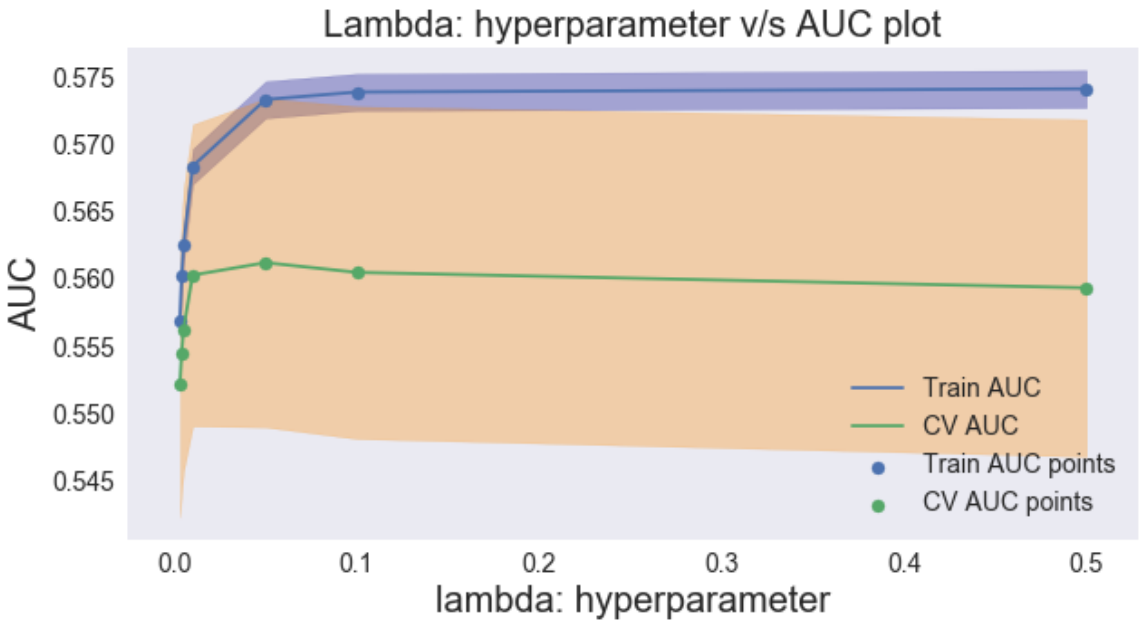
```
X_tr = hstack((school_state_one_hot_train,categories_one_hot_train,subcategories_one_ho
t_train,
project_grade_category_one_hot_train,teacher_prefix_one_hot_train,price_train,quantity_
train,
pre_posted_project_train,essay_count_train,title_count_train,sentiment_neg_train, senti
ment_neu_train,
sentiment_pos_train, sentiment_compound_train)).tocsr()


X_cr = hstack((school_state_one_hot_cv,categories_one_hot_cv,subcategories_one_hot_cv,
project_grade_category_one_hot_cv,teacher_prefix_one_hot_cv,price_cv,quantity_cv,
pre_posted_project_cv,essay_count_cv,title_count_cv,sentiment_neg_cv, sentiment_neu_cv,
sentiment_pos_cv, sentiment_compound_cv)).tocsr()


X_te = hstack((school_state_one_hot_test,categories_one_hot_test,subcategories_one_hot_
test,
project_grade_category_one_hot_test,teacher_prefix_one_hot_test,price_test,quantity_tes
t,
pre_posted_project_test,essay_count_test,title_count_test,sentiment_neg_test, sentiment
_neu_test,
sentiment_pos_test, sentiment_compound_test)).tocsr()


print("The Final Data Matrix :----")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print('='*50)
```

```
The Final Data Matrix :----
(49041, 109) (49041,)
(24155, 109) (24155,)
(36052, 109) (36052,)
==================================================
```

Set 5.1 Finding best hyperparameter using GridSearchCV (K fold Cross Validation)

In [304]:

```
lr = LogisticRegression()
parameters = {'C':[0.5, 0.1, 0.05, 0.01, 0.005, 0.004, 0.003]}
clf = GridSearchCV(lr, parameters, cv= 10, scoring='roc_auc')
clf.fit(X_tr, y_train)
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
plt.figure(figsize=(10,5))
plt.plot(parameters['C'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')
plt.plot(parameters['C'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['C'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
3,color='darkorange')
plt.scatter(parameters['C'], train_auc, label='Train AUC points')
plt.scatter(parameters['C'], cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("lambda: hyperparameter", fontsize = 20)
plt.ylabel("AUC", fontsize = 20)
plt.title("Lambda: hyperparameter v/s AUC plot", fontsize = 20)
plt.grid()
plt.show()
```



In [305]:

```
clf.best_estimator_
```

Out[305]:

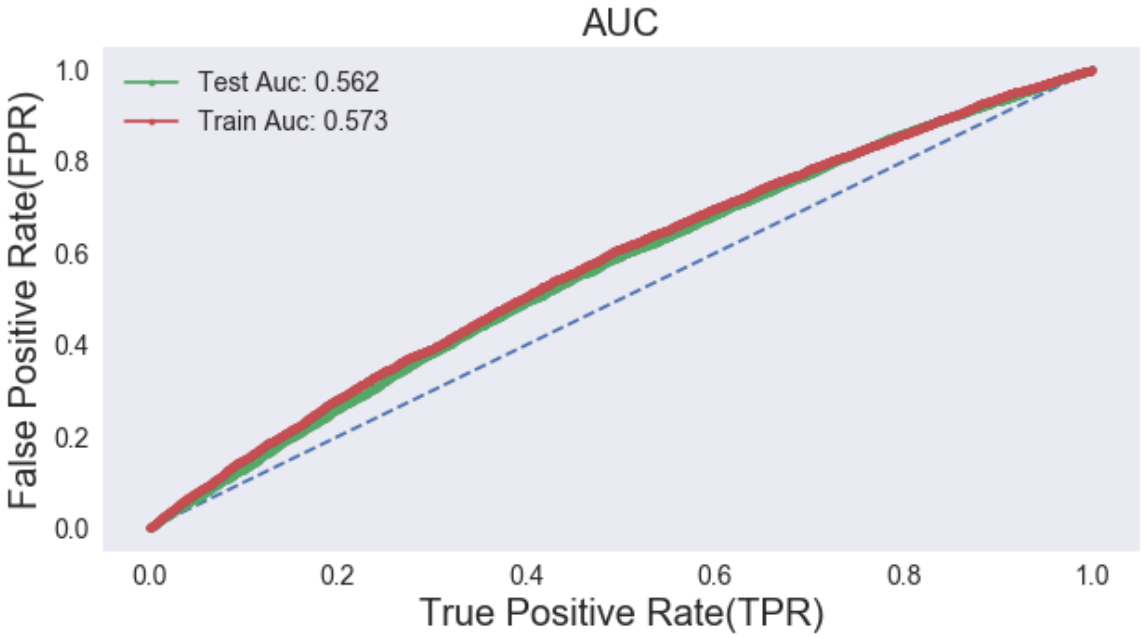
```
LogisticRegression(C=0.05, class_weight=None, dual=False, fit_intercept=Tr
ue,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

best hyperparameter = 0.05

Set 5.2 Train the model using the best hyper parameter value

In [306]:

```
from sklearn.metrics import roc_curve, auc
model = LogisticRegression(C = 0.05)
model.fit(X_tr, 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
# predict probabilities for train dataset.
y_train_prob = model.predict_proba(X_tr)
# keep probabilities for the positive outcome only
y_train_prob = y_train_prob[:, 1]
auc_train = roc_auc_score(y_train, y_train_prob)
train_fpr,train_tpr,train_threshold = roc_curve(y_train, y_train_prob)
# predict probabilities for test dataset.
y_test_prob = model.predict_proba(X_te)
# keep probabilities for the positive outcome only
y_test_prob  = y_test_prob [:, 1]
# calculate AUC
auc = roc_auc_score(y_test, y_test_prob )
#print('AUC: %.3f' % auc)
# calculate roc curve
test_fpr, test_tpr, test_threshold = roc_curve(y_test, y_test_prob )
# plot no skill
plt.figure(figsize=(10,5))
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(test_fpr, test_tpr,label = 'Test Auc: %.3f' % auc, marker='.')
plt.plot(train_fpr, train_tpr,label = 'Train Auc: %.3f' % auc_train, marker='.')
plt.legend()
plt.xlabel("True Positive Rate(TPR)", fontsize = 20)
plt.ylabel("False Positive Rate(FPR)",fontsize = 20)
plt.title("AUC",fontsize = 20)
plt.grid()
# show the plot
plt.show()
```



Set 5.3 Confusion Matrix

In [307]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(train_threshold, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_prob, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_prob, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.30906590587626265 for threshold 0.85
Train confusion matrix
[[ 4241  3185]
 [19094 22521]]
Test confusion matrix
[[ 3073  2386]
 [14287 16306]]
```

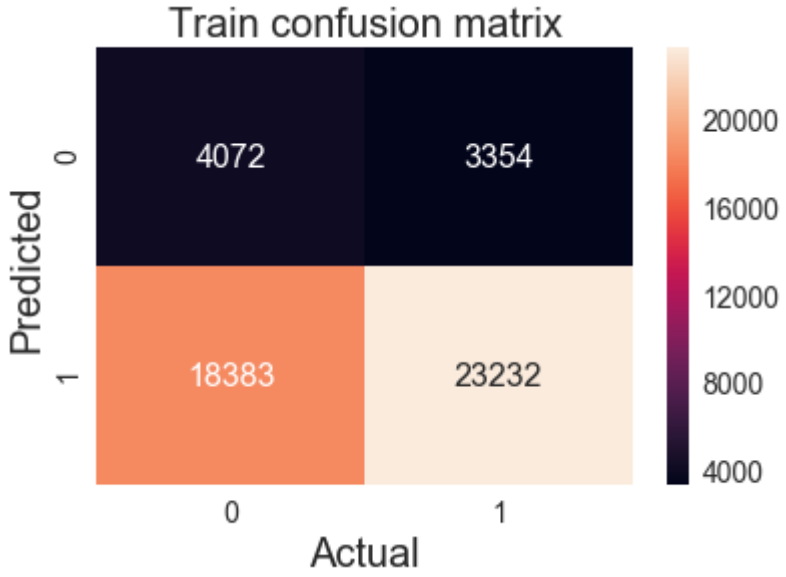
In [308]:

```
CM_df_tr = pd.DataFrame(confusion_matrix(y_train, predict(y_train_prob, train_threshold
,train_fpr, train_tpr)))
```

```
the maximum value of tpr*(1-fpr) 0.30906590587626265 for threshold 0.848
```

In [309]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(CM_df_tr, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Actual', fontsize = 20)
plt.ylabel('Predicted', fontsize = 20)
plt.title('Train confusion matrix', fontsize = 20)
plt.show()
```



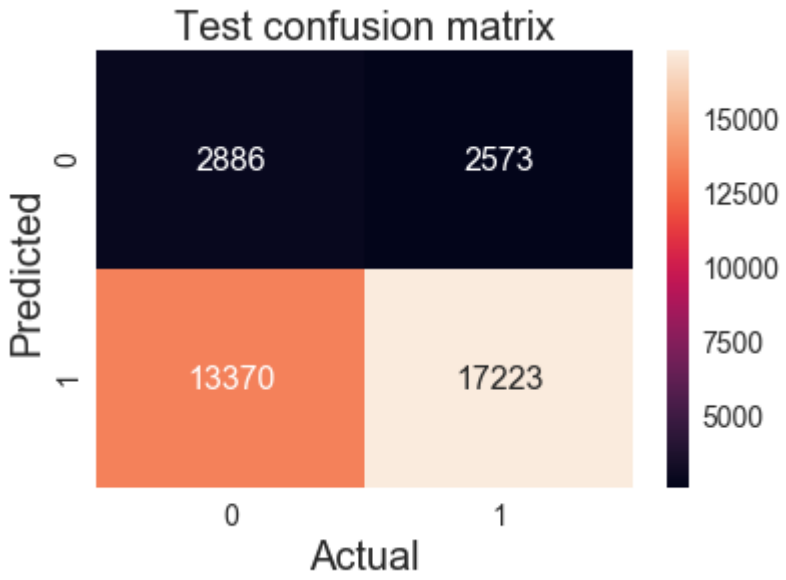
In [310]:

```
CM_df_te = pd.DataFrame(confusion_matrix(y_test, predict(y_test_prob, test_threshold,te
st_fpr, test_tpr)))
```

```
the maximum value of tpr*(1-fpr) 0.3014098548944484 for threshold 0.847
```

In [311]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(CM_df_te, annot=True,annot_kws={"size": 16}, fmt='g')
plt.xlabel('Actual', fontsize = 20)
plt.ylabel('Predicted', fontsize = 20)
plt.title('Test confusion matrix', fontsize = 20)
plt.show()
```



6. Conclusion

In [313]:

```
from prettytable import PrettyTable


x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Alpha:Hyper Parameter", "AUC"]
x.add_row(["BOW", "Logistic Regression", 0.01, 0.59])
x.add_row(["TFIDF", "Logistic Regression", 0.5, 0.66])
x.add_row(["AVG W2V", "Logistic Regression", 0.5, 0.7])
x.add_row(["TFIDF W2V", "Logistic Regression", 0.5, 0.696])
x.add_row(["WITHOUT TEXT", "Logistic Regression", 0.05, 0.56])

print(x)
```

```
+--------------+---------------------+-----------------------+-------+
|  Vectorizer  |        Model        | Alpha:Hyper Parameter |  AUC  |
+--------------+---------------------+-----------------------+-------+
|     BOW      | Logistic Regression |          0.01         |  0.59 |
|    TFIDF     | Logistic Regression |          0.5          |  0.66 |
|   AVG W2V    | Logistic Regression |          0.5          |  0.7  |
|  TFIDF W2V   | Logistic Regression |          0.5          | 0.696 |
| WITHOUT TEXT | Logistic Regression |          0.05         |  0.56 |
+--------------+---------------------+-----------------------+-------+
```

1. Logistic Regression with Avg W2v performs better than others.

2. Model became worst without text features.

3. Time complexity is way better than KNN.