DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy`

Feature	Description
	• `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."

your neighborhood, and your school are all helpful.

- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

In [1]:

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

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

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
##from sklearn import cross_validation
from sklearn.metrics import accuracy_score
##from sklearn.cross_validation import cross_val_score
from sklearn.model_selection import train_test_split
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

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

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
## os.chdir('C:/Users/kingsubham27091995/Desktop/AppliedAiCouse/DonorsChoose')
```

Reading Data

In [0]:

```
project_data = pd.read_csv('train_data.csv' ,nrows=50000)
```

In [0]:

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

In [4]:

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

```
Number of data points in train data (50000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [6]:

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

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

Out[6]:

	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

Preprocessing project_grade_categories

In [0]:

```
project_grade_category = []

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

In [8]:

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

Out[8]:

```
['Grades_PreK-2', 'Grades_6-8', 'Grades_6-8', 'Grades_PreK-2', 'Grades_PreK-2']
```

In [0]:

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

In [0]:

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

In [11]:

```
project_data.head(5)
```

Out[11]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_subject_ca
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Applied Learning

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	2016 Date	project_subject_cat
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	04-27 01:05:25	Literacy & Language
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Math & Science, Hist Civics
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Literacy & Language
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	IL	2016-04-27 07:19:44	Literacy & Language

Preprocessing of `project_subject_categories`

In [0]:

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

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

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

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

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

Preprocessing of `project_subject_subcategories`

In [0]:

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

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

```
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

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

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

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

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

Text preprocessing

Finding number of words in title and introducing it in a new column

- This can be used as Numerical Feature for Vectorisation

In [0]:

```
title_word_count = []
for a in project_data["project_title"] :
    b = len(a.split())
    title_word_count.append(b)
```

In [15]:

```
project_data["title_word_count"] = title_word_count
project_data.head(5)
```

Out[15]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	proje
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Flexible Seating for Flexible Learning	I rece article giving
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Going Deep: The Art of Inner Thinking!	My stu crave they e obsta
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Breakout Box to Ignite Engagement!	It's the schoo Routir
						2016-		Never

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	
23374	72911	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	04-27 02:04:15	iPad for Learners	societ proje Techr
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	IL	2016-04-27 07:19:44	A flexible classroom for flexible minds!	My st yearn classr enviro

In [0]:

```
# 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 [17]:

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

Out[17]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	project
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Flexible Seating for Flexible Learning	I recent article a giving s
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Going Deep: The Art of Inner Thinking!	My stud crave c they ea obstacl

Finding number of words in essay and introducing it in a new column

- This can be used as Numerical Feature for Vectorisation

In [0]:

```
essay_word_count = []

for ess in project_data["essay"] :
    c = len(ess.split())
    essay_word_count.append(c)
```

In [19]:

```
project_data["essay_word_count"] = essay_word_count
project_data.head(5)
```

Out[19]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_title	projec
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Flexible Seating for Flexible	I recei article giving

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	Learning project_title	proje
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Going Deep: The Art of Inner Thinking!	My st crave they e obsta
29891	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Breakout Box to Ignite Engagement!	It's the schoo Routir
23374	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	iPad for Learners	Never societ chang Techr
49228	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	IL	2016-04-27 07:19:44	A flexible classroom for flexible minds!	My st yearn classr enviro

Splitting Project_Data into Train and Test Datasets

In [0]:

```
from sklearn.model_selection import train_test_split
#Splitting into Train and Test Data
X_train, X_test, y_train, y_test = train_test_split(project_data,
project_data['project_is_approved'], test_size=0.33, stratify = project_data['project_is_approved'
])
#Splitting Train data into Train and Cross Validation Data
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

We don't need the 'project_is_approved' feature now

In [0]:

```
X_train.drop(['project_is_approved'], axis=1, inplace=True)
X_test.drop(['project_is_approved'], axis=1, inplace=True)
X_cv.drop(['project_is_approved'], axis=1, inplace=True)
```

In [22]:

```
# printing some random reviews
print(X_train['essay'].values[0])
print("="*50)
print(X_train['essay'].values[500])
print("="*50)
print(X_train['essay'].values[1000])
print("="*50)
print(X_train['essay'].values[10000])
print("="*50)
print(X_train['essay'].values[20000])
print("="*50)
```

```
In Room 316, my 8th graders learn about the past, the present, and why it all matters to them as c
itizens and fellow human beings. Teaching 8th grade social studies in an urban, low income/high po
verty school district, my classroom serves as a positive, safe space for my students to grapple wi
th all the challenges of 8th grade, both academic and social, as well as issues in society, both p
ast and present. \r\n\r\nMy curriculum covers some heated and sometimes sensitive topics, but my 8
th graders are always eager to jump in and immerse themselves in hard work, even if it means
stepping outside their comfort zone or embracing new ideas. They think critically about the
content as well as make connections between the past and current day issues.  One of the many thin
```

content as well as make connections between the past and current day issues. One of the many things I love about my students is their curiosity - they ask so many questions that demonstrate their desire to engage and grapple with challenging material and topics. They also absolutely LOVE debating! Any chance they can get to talk about a topic or an issue will bring to light rich conversations, and, of course, respectful disagreements. Every day in 316 is vibrant and lively!Middle schoolers need change and movement to stay engaged and focused - sitting at one table for our 75 minute class period is definitely not an ideal learning environment for my 8th graders. Providing my students with the opportunity to choose different types of seats during independent and partner work time would help foster a positive learning environment in so many ways. \r\n\r\n My 8th graders would jump at the chance to start their work if they knew that they had options for where and how they could sit. Many of my students love to sit on the carpet during work time, but often find it uncomfortable after awhile, and this hinders their focus. Body pillows and bedrest pillows would keep them comfortable and, thus, engaged. For students that don't like to sit at the carpet, ottomans would be a great alternative to being stuck sitting at the tables every day. Giving my students flexible seating options would help build a student-centered classroom environment and create an exciting, positive buzz about the room as students prepared to tackle the challenges of the day's work.nannan
==========
Our school is filled with students from pre-kindergarten to 8th grade. As a school we are working on earning Class Dojo points for making good choices showing READY TO LEARN, RESPECT, RESPONSIBILITY! The Dojo points will earn students rewards for making the right choices!Our students are young men and young ladies who live in the \"most dangerous\" city in the US as well as being labels one of the \"highest poverty rates\". They struggle daily in many ways to make it to school but once they arrive they are ready to go. They come to school to learn. They have plans for college and to reach those goals we push everyday to overcome the struggles they face.We will use the supplies to reward students making the right choices every Friday in our weekly raffle. Then once a month all those who were in the weekly drawings will be in the end of the month reward also! We plan on making big announcements each Friday and then at the end of the month a huge deal to reward the wonderful choices our students are making!Our project will change our school! We are all working together to help our students make positive choices with the behavior and school work. This project will help reward the students for those wonderful choices and therefore help improve the positive attitude and behaviors on our school!
==========
I have a kindergarten class with 27 students, they are bright eyed and full of energy. My students are excited to come to school and learn new things everyday. Our school is located in the south side of Los Angeles, we are a Title I school where all of our students qualify for free lunch. My students welcome all and every single donation that we receive, and we need each and everything we can get. My students need materials to stay engaged and have the right learning conditions to maximize engagement in the classroom.Most of my students have never been to school, they cannot sit still. When I asked my students what they needed to sit still, they all said Hokki stools. My students have seen these in other classrooms and would like some in our classroom. \r\nMy students said that Hokki stools would help them have fun will they learn because they would not have to sit still. \r\nBy having an alternative seating option, my little ones will be more focused on learning than on trying to get up from their seat. We are also requesting healthy snacks. My students always ask me if I have any snacks for them since they have an early lunch. They told me that they can focus more on learning if they're not hungry. Your donations will be greatly appreciated, thank you!\r\nnannan
==========
These Gifted and Talented students love to ask questions and do student-led learning projects. Several projects students did last year involved engineering bottle cars, hover craft, and kinds of chemical reactions got the G.A.T.E. class excited about more science! \r\nThe students told me that they want to dig into more science and design things this year. \r\nMany of these students are English learners. Our school is on the east side of San Jose, and many of our students are from low socio-economic backgrounds. In spite of the challenges, this is a group who loves to learn. I enjoy teaching science even though my main assignment is reading intervention and I have a degree in art. Getting science, technology, engineering, art and math into our learning time will be super exciting.Every year my G.A.T.E. class of 3-5th grade students engage in a whole class project of their choice and then make up teams or partnerships to do their own \"genius hour\" personal learning projects. I see them once a week and teach reading intervention the rest of the week.\r\nAt the end of last school year they were buzzed about chemistry experiments and wanted to build more things. \r\nSo, I've pulled out materials from my 5th grade teaching days for doing chemical reactions, and other favorite discrepant event type science demos to lead off. We'll ask questions! We'll get very curious! Then, they'll build and create math and art problems off their engineering discoveries. And, this grant will give them a taste of some environmental science experimentation. That way, students will be ready midyear to launch their own inquiries for Personal Learning Time.nannan
==========
I have three diverse classes of 75 students that come from myriad of backgrounds. I am an ESL classroom, so for about 1/2 of my students, English is not their first language. I work hard to create a classroom culture that instills the love for learning as well as the love for reading in all my students.\r\n\r\nOur school is a high poverty, urban school that is one of the most diverse schools in our district. We have a staff that is caring and passionate about our students' educations and futures. We are always seeking opportunities to provide high quality, engaging activities to enhance the learning experience for our students.\r\nBreakout EDU creates ultra-engaging learning games for students. Games (Breakouts) teach teamwork, problem-solving, critical thinking, and troubleshooting by presenting students with challenges that ignite their natural drive to problem-solve.\r\n\r\nBreakout EDU creates ultra-engaging learning games for all students. Top ten reasons to use Breakout EDU: 1. It's fun and engaging for all students; 2. adaptable to all subjec

t areas; 3. promotes collaboration and team building; 4. creates problem-solving and critical thin
king skills; 5. enhances communication skills; 6. challenges students to persevere; 7. builds
inference skills; 8. students learn to work under pressure; 9. student-centered; 10. inquiry-based
learning at its best.nannan
==

In [0]:

```
# 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 [24]:

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

I have three diverse classes of 75 students that come from myriad of backgrounds. I am an ESL
classroom, so for about 1/2 of my students, English is not their first language. I work hard to cr
eate a classroom culture that instills the love for learning as well as the love for reading in al
l my students.\r\n\r\nOur school is a high poverty, urban school that is one of the most diverse s
chools in our district. We have a staff that is caring and passionate about our students'
educations and futures. We are always seeking opportunities to provide high quality, engaging acti
vities to enhance the learning experience for our students.\r\nBreakout EDU creates ultra-engaging
learning games for students. Games (Breakouts) teach teamwork, problem-solving, critical thinking,
and troubleshooting by presenting students with challenges that ignite their natural drive to
problem-solve.\r\n\r\nBreakout EDU creates ultra-engaging learning games for all students. Top ten
reasons to use Breakout EDU: 1. It is fun and engaging for all students; 2. adaptable to all subje
ct areas; 3. promotes collaboration and team building; 4. creates problem-solving and critical thi
nking skills; 5. enhances communication skills; 6. challenges students to persevere; 7. builds
inference skills; 8. students learn to work under pressure; 9. student-centered; 10. inquiry-based
learning at its best.nannan
==

In [25]:

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

I have three diverse classes of 75 students that come from myriad of backgrounds. I am an ESL
classroom, so for about 1/2 of my students, English is not their first language. I work hard to cr
eate a classroom culture that instills the love for learning as well as the love for reading in al
l my students. Our school is a high poverty, urban school that is one of the most diverse
schools in our district. We have a staff that is caring and passionate about our students'
educations and futures. We are always seeking opportunities to provide high quality, engaging acti
vities to enhance the learning experience for our students. Breakout EDU creates ultra-engaging l
earning games for students. Games (Breakouts) teach teamwork, problem-solving, critical thinking,
and troubleshooting by presenting students with challenges that ignite their natural drive to
problem-solve. Breakout EDU creates ultra-engaging learning games for all students. Top ten rea
sons to use Breakout EDU: 1. It is fun and engaging for all students; 2. adaptable to all subject
areas; 3. promotes collaboration and team building; 4. creates problem-solving and critical thinki
ng skills; 5. enhances communication skills; 6. challenges students to persevere; 7. builds
inference skills; 8. students learn to work under pressure; 9. student-centered; 10. inquiry-based
learning at its best.nannan

learning at its best.nannan

In [26]:

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

```
I have three diverse classes of 75 students that come from myriad of backgrounds I am an ESL
classroom so for about 1 2 of my students English is not their first language I work hard to creat
e a classroom culture that instills the love for learning as well as the love for reading in all m
y students Our school is a high poverty urban school that is one of the most diverse schools in ou
r district We have a staff that is caring and passionate about our students educations and futures
We are always seeking opportunities to provide high quality engaging activities to enhance the lea
rning experience for our students Breakout EDU creates ultra engaging learning games for students
Games Breakouts teach teamwork problem solving critical thinking and troubleshooting by presenting
students with challenges that ignite their natural drive to problem solve Breakout EDU creates
ultra engaging learning games for all students Top ten reasons to use Breakout EDU 1 It is fun and
engaging for all students 2 adaptable to all subject areas 3 promotes collaboration and team build
ing 4 creates problem solving and critical thinking skills 5 enhances communication skills 6 chall
enges students to persevere 7 builds inference skills 8 students learn to work under pressure 9 st
udent centered 10 inquiry based learning at its best nannan
```

In [0]:

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

Preprocessing Train Dataset(Essay feature)

In [28]:

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

```
    preprocessed_essays_train.append(sent.lower().strip())
```

```
100%|██████████| 22445/22445 [00:13<00:00, 1633.94it/s]
```

In [29]:

```
# after preprocesing
preprocessed_essays_train[20000]
```

Out[29]:

```
'three diverse classes 75 students come myriad backgrounds esl classroom 1 2 students english not
first language work hard create classroom culture instills love learning well love reading student
s school high poverty urban school one diverse schools district staff caring passionate students e
ducations futures always seeking opportunities provide high quality engaging activities enhance le
arning experience students breakout edu creates ultra engaging learning games students games break
outs teach teamwork problem solving critical thinking troubleshooting presenting students
challenges ignite natural drive problem solve breakout edu creates ultra engaging learning games s
tudents top ten reasons use breakout edu 1 fun engaging students 2 adaptable subject areas 3 promo
tes collaboration team building 4 creates problem solving critical thinking skills 5 enhances comm
unication skills 6 challenges students persevere 7 builds inference skills 8 students learn work p
ressure 9 student centered 10 inquiry based learning best nannan'
```

Preprocessing Test Dataset(Essay feature)

In [30]:

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

```
100%|██████████| 16500/16500 [00:10<00:00, 1635.70it/s]
```

In [31]:

```
preprocessed_essays_test[2000]
```

Out[31]:

```
'5th grade rock stars need say anymore students group special education students attending low eco
nomic public school order many students learn need consistent practice math social skills order le
arn get better something need exposure adding technology classroom giving students exposure needed
help academically well life skills appropriate social behavior use technology world ever growing u
se technology students growing age technology vital skill success life yet already falling behind
school one computer lab 30 computers 500 students share several classrooms also laptop carts many
not receive regular maintenance cases less 50 laptops work given time classroom two desktop
computers work approximately half time want students opportunity learn explore technology skills a
pply learning life hope get 2 ipads classroom ipads allow students work iep goals reading math wel
l math related skills social skills take necessary sensory breaks nannan'
```

Preprocessing Cross Validation Dataset(Essay feature)

In [32]:

```
# Combining all the above steps
from tqdm import tqdm
preprocessed_essays_cv = []
# tqdm is for printing the status bar
for sentance in tqdm(X_cv['essay'].values):
```

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

```
100%|██████████| 11055/11055 [00:06<00:00, 1629.77it/s]
```

In [33]:

```
preprocessed_essays_cv[2000]
```

Out[33]:

```
'fortunate one one school 21st century skills goal school originally built 990 students 2003 howev
er currently educating 1 500 students 50 student population econimically disadvantaged per federal
guidelines last year saw significant gains end grade test scores first year one one school year lo
oking even higher scores school committed providing educational setting allow students develop pra
ctical real world 21st century skills striving become 21st century school increasing use
technology within classroom partnership 21st century skills feels within context core knowledge in
struction students must also learn essential skills success today world critical thinking problem
solving communication collaboration www p21 org string bags allow chrombooks safe traveling class
class last year school high rate chrombooks damaged traveling hallway chrombook damaged may
prevent student using technology fixed school uses chomebooks incorporate 21st century skills
every class somepoint day strategic plan 2018 better tomorrow relies heavily technology achieve fi
nal goal technology driven instruction engages students authentic 21st century experiences mold cr
eativity collaboration focusing analysis evaluation chrombooks allow students invested following a
reas creativity innovation students required use wide range idea techniques open new diverse persp
ectives view failure opportunity learn critical thinking problem solving students use different ty
pes reasoning skills understand parts whole interact hypothesize possible resolutions ask
questions clarify various points view communication collaboration students develop skills needed e
ffective communication articulate thoughts ideas various mediums well improve demonstrate ability
work effectively group nannan'
```

Preprocessing Train Dataset(Title feature)

In [34]:

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

```
100%|██████████| 22445/22445 [00:00<00:00, 33257.21it/s]
```

In [35]:

```
preprocessed_titles_train[20000]
```

Out[35]:

```
'the breakout experience engaging all learners'
```

Preprocessing Test Dataset(Title feature)

In [36]:

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

```
100%|██████████| 16500/16500 [00:00<00:00, 33673.98it/s]
```

In [37]:

```
preprocessed_titles_test[2000]
```

Out[37]:

```
'help my students reach their potential through ipads'
```

Preprocessing Cross Validation Dataset(Title feature)

In [38]:

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

```
100%|██████████| 11055/11055 [00:00<00:00, 32964.41it/s]
```

In [39]:

```
preprocessed_titles_cv[1000]
```

Out[39]:

```
'tell i forget involve i learn'
```

1.5 Preparing data for models

In [0]:

```
project_data.columns
```

Out[0]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', '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',
       'project_grade_category', 'clean_categories', 'clean_subcategories',
       'title_word_count', 'essay', 'essay_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

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

- quantity : numerical (optinal)
- teacher_number_of_previously_posted_projects : numerical
- price : numerical
- title_word_count-numrical
-essay_word_count-numerical
```

1.5.1 Vectorizing Categorical data

- https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

One Hot Encoding of Clean_Categories

In [40]:

```
from sklearn.feature_extraction.text import CountVectorizer

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

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

print(vectorizer.get_feature_names())

print("Shape of matrix of Train data after one hot encoding ",categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",categories_one_hot_test.shape)
print("Shape of matrix of CV data after one hot encoding ",categories_one_hot_cv.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix of Train data after one hot encoding  (22445, 9)
Shape of matrix of Test data after one hot encoding  (16500, 9)
Shape of matrix of CV data after one hot encoding  (11055, 9)
```

One Hot Encoding of Clean_Sub_Categories

In [41]:

```
# we use count vectorizer to convert the values into one

vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
vectorizer.fit(X_train['clean_subcategories'].values)

sub_categories_one_hot_train = vectorizer.transform(X_train['clean_subcategories'].values)
sub_categories_one_hot_test = vectorizer.transform(X_test['clean_subcategories'].values)
sub_categories_one_hot_cv = vectorizer.transform(X_cv['clean_subcategories'].values)


print(vectorizer.get_feature_names())
```

```
print("Shape of matrix of Train data after one hot encoding ",sub_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",sub_categories_one_hot_test.shape)
print("Shape of matrix of Cross Validation data after one hot encoding ",sub_categories_one_hot_cv
.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix of Train data after one hot encoding  (22445, 30)
Shape of matrix of Test data after one hot encoding  (16500, 30)
Shape of matrix of Cross Validation data after one hot encoding  (11055, 30)
```

Performing One-Hot-Encoding for School-State

In [0]:

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

In [0]:

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

In [44]:

```
## we use count vectorizer to convert the values into one hot encoded features

vectorizer = CountVectorizer(vocabulary=list(sorted_school_state_dict.keys()), lowercase=False, bi
nary=True)
vectorizer.fit(X_train['school_state'].values)

school_state_categories_one_hot_train = vectorizer.transform(X_train['school_state'].values)
school_state_categories_one_hot_test = vectorizer.transform(X_test['school_state'].values)
school_state_categories_one_hot_cv = vectorizer.transform(X_cv['school_state'].values)

print(vectorizer.get_feature_names())

print("Shape of matrix of Train data after one hot encoding
",school_state_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",school_state_categories_one_hot_test.
shape)
print("Shape of matrix of Cross Validation data after one hot encoding
",school_state_categories_one_hot_cv.shape)
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'NH', 'SD', 'NE', 'AK', 'DE', 'WV', 'ME', 'NM', 'HI', 'DC', 'KS', 'I
D', 'IA', 'AR', 'CO', 'MN', 'OR', 'MS', 'KY', 'NV', 'MD', 'TN', 'CT', 'AL', 'UT', 'WI', 'VA', 'AZ',
'NJ', 'OK', 'MA', 'LA', 'WA', 'MO', 'IN', 'OH', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'TX', 'NY
', 'CA']
Shape of matrix of Train data after one hot encoding  (22445, 51)
Shape of matrix of Test data after one hot encoding  (16500, 51)
Shape of matrix of Cross Validation data after one hot encoding  (11055, 51)
```

Performing One-Hot-Encoding for Project_Grade_Category

In [0]:

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

In [0]:

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

In [47]:

```
## we use count vectorizer to convert the values into one hot encoded features

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

project_grade_categories_one_hot_train = vectorizer.transform(X_train['project_grade_category'].va
lues)
project_grade_categories_one_hot_test =
vectorizer.transform(X_test['project_grade_category'].values)
project_grade_categories_one_hot_cv = vectorizer.transform(X_cv['project_grade_category'].values)

print(vectorizer.get_feature_names())

print("Shape of matrix of Train data after one hot encoding
",project_grade_categories_one_hot_train.shape)
print("Shape of matrix of Test data after one hot encoding ",project_grade_categories_one_hot_test
.shape)
print("Shape of matrix of Cross Validation data after one hot encoding
",project_grade_categories_one_hot_cv.shape)
```

```
['Grades_9-12', 'Grades_6-8', 'Grades_3-5', 'Grades_PreK-2']
Shape of matrix of Train data after one hot encoding  (22445, 4)
Shape of matrix of Test data after one hot encoding  (16500, 4)
Shape of matrix of Cross Validation data after one hot encoding  (11055, 4)
```

Performing One-Hot_encoding for Teacher_Prefix

In [0]:

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

In [0]:

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

In [50]:

```
## ValueError: np.nan is an invalid document, expected byte or unicode string.
## The link below explains h0w to tackle such discrepancies.
## https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np-nan-
is-an-invalid-document/39308809#39308809

vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict.keys()), lowercase=False,
binary=True)
vectorizer.fit(X_train['teacher_prefix'].values.astype("U"))

teacher_prefix_categories_one_hot_train =
vectorizer.transform(X_train['teacher_prefix'].values.astype("U"))
teacher_prefix_categories_one_hot_test =
vectorizer.transform(X_test['teacher_prefix'].values.astype("U"))
teacher_prefix_categories_one_hot_cv =
vectorizer.transform(X_cv['teacher_prefix'].values.astype("U"))

print(vectorizer.get_feature_names())

print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_train.shape)
print("Shape of matrix after one hot encoding ",teacher prefix categories one hot test.shape)
```

```
print("Shape of matrix after one hot encoding ",teacher_prefix_categories_one_hot_cv.shape)
```

```
['nan', 'Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encoding  (22445, 6)
Shape of matrix after one hot encoding  (16500, 6)
Shape of matrix after one hot encoding  (11055, 6)
```

1.5.2 Vectorizing Text data

Bag of words for Train Data(Essay Feature)

In [0]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(preprocessed_essays_train)
text_bow_train = vectorizer.transform(preprocessed_essays_train)
print("Shape of matrix after one hot encodig ",text_bow_train.shape)
```

```
Shape of matrix after one hot encodig  (22445, 8811)
```

Bag of words for Test Data(Essay Feature)

In [0]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
#Transforming test data size to equalise with train data
text_bow_test=vectorizer.transform(preprocessed_essays_test)
print("Shape of matrix after one hot encodig ",text_bow_test.shape)
```

```
Shape of matrix after one hot encodig  (16500, 8811)
```

Bag of words for Cross Validation Data(Essay Feature)

In [0]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).

text_bow_cv= vectorizer.transform(preprocessed_essays_cv)
print("Shape of matrix after one hot encodig ",text_bow_cv.shape)
```

```
Shape of matrix after one hot encodig  (11055, 8811)
```

Bag of words for Train Data(Titles Feature)

In [0]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer.fit(preprocessed_titles_train)
title_bow_train= vectorizer.transform(preprocessed_titles_train)
print("Shape of matrix after one hot encodig ",title_bow_train.shape)
```

```
Shape of matrix after one hot encodig  (22445, 1246)
```

Bag of words for Test Data(Titles Feature)

In [0]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
title_bow_test= vectorizer.transform(preprocessed_titles_test)
```

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

```
Shape of matrix after one hot encodig  (16500, 1246)
```

Bag of words for Cross Validation Data(Titles Feature)

In [0]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).

title_bow_cv= vectorizer.transform(preprocessed_titles_cv)
print("Shape of matrix after one hot encodig ",title_bow_cv.shape)
```

```
Shape of matrix after one hot encodig  (11055, 1246)
```

TFIDF vectorizer for Train Data(Essay feature)

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(preprocessed_essays_train)
text_tfidf_train = vectorizer.transform(preprocessed_essays_train)
print("Shape of matrix after one hot encodig ",text_tfidf_train.shape)
```

```
Shape of matrix after one hot encodig  (22445, 8811)
```

TFIDF vectorizer for Test Data(Essay feature)

In [0]:

```
text_tfidf_test = vectorizer.transform(preprocessed_essays_test)
print("Shape of matrix after one hot encodig ",text_tfidf_test.shape)
```

```
Shape of matrix after one hot encodig  (16500, 8811)
```

TFIDF vectorizer for Cross Validation Data(Essay feature)

In [0]:

```
text_tfidf_cv = vectorizer.transform(preprocessed_essays_cv)
print("Shape of matrix after one hot encodig ",text_tfidf_cv.shape)
```

```
Shape of matrix after one hot encodig  (11055, 8811)
```

TFIDF vectorizer for Train Data(Titles feature)

In [0]:

```
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(preprocessed_titles_train)
title_tfidf_train = vectorizer.transform(preprocessed_titles_train)
print("Shape of matrix after one hot encodig ",title_tfidf_train.shape)
```

```
Shape of matrix after one hot encodig  (22445, 1246)
```

TFIDF vectorizer for Test Data(Titles feature)

In [0]:

```
title_tfidf_test = vectorizer.transform(preprocessed_titles_test)
print("Shape of matrix after one hot encodig ",title_tfidf_test.shape)
```

```
Shape of matrix after one hot encodig  (16500, 1246)
```

TFIDF vectorizer for Cross Validation Data(Titles feature)

In [0]:

```
title_tfidf_cv = vectorizer.transform(preprocessed_titles_cv)
print("Shape of matrix after one hot encodig ",title_tfidf_cv.shape)
```

```
Shape of matrix after one hot encodig  (11055, 1246)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [0]:

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

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

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

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

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

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

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

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


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

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

```
    pickle.dump(words_courpus, f)


'''
```

Out[0]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
============================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus",
len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\r
print("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [0]:

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

Avg_W2V for Train Data(Essays feature)

In [52]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
print(len(avg_w2v_vectors_train[0]))
```

```
100%|██████████| 22445/22445 [00:07<00:00, 3072.21it/s]
```

```
22445
300
```

Avg_W2V for Test Data(Essays feature)

In [53]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_test): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
```

```
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)

print(len(avg_w2v_vectors_test))
print(len(avg_w2v_vectors_test[0]))
```

```
100%|██████████| 16500/16500 [00:05<00:00, 3057.20it/s]
```

```
16500
300
```

Avg_W2V for Cross Validation Data(Essays feature)

In [54]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_cv): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv.append(vector)

print(len(avg_w2v_vectors_cv))
print(len(avg_w2v_vectors_cv[0]))
```

```
100%|██████████| 11055/11055 [00:03<00:00, 3011.84it/s]
```

```
11055
300
```

Avg_W2V for Train Data(Titles feature)

In [55]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_train.append(vector)

print(len(avg_w2v_vectors_titles_train))
print(len(avg_w2v_vectors_titles_train[0]))
```

```
100%|██████████| 22445/22445 [00:00<00:00, 53341.96it/s]
```

```
22445
300
```

300

Avg_W2V for Test Data(Titles feature)

In [56]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_test): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_test.append(vector)

print(len(avg_w2v_vectors_titles_test))
print(len(avg_w2v_vectors_titles_test[0]))
```

```
100%|██████████| 16500/16500 [00:00<00:00, 54168.26it/s]
```

```
16500
300
```

Avg_W2V for Cross Validation Data(Titles feature)

In [57]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_titles_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_cv): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_cv.append(vector)

print(len(avg_w2v_vectors_titles_cv))
print(len(avg_w2v_vectors_titles_cv[0]))
```

```
100%|██████████| 11055/11055 [00:00<00:00, 55960.97it/s]
```

```
11055
300
```

TFIDF weighted W2V for Train Data(Essays feature)

In [0]:

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

In [64]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|██████████| 22445/22445 [00:42<00:00, 524.96it/s]
```

```
22445
300
```

TFIDF weighted W2V for Test Data(Essays feature)

In [0]:

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

In [66]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_test): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

print(len(tfidf_w2v_vectors_test))
print(len(tfidf_w2v_vectors_test[0]))
```

```
100%|██████████| 16500/16500 [00:33<00:00, 497.94it/s]
```

```
16500
300
```

TFIDF weighted W2V for Cross Validation Data(Essays feature)

In [0]:

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

In [68]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_cv): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_cv.append(vector)

print(len(tfidf_w2v_vectors_cv))
print(len(tfidf_w2v_vectors_cv[0]))
```

```
100%|██████████| 11055/11055 [00:21<00:00, 520.08it/s]
```

```
11055
300
```

TFIDF weighted W2V for Train Data(Titles feature)

In [0]:

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

In [70]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_train): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
```

```
    tfidf_w2v_vectors_titles_train.append(vector)

print(len(tfidf_w2v_vectors_titles_train))
print(len(tfidf_w2v_vectors_titles_train[0]))
```

```
100%|██████████| 22445/22445 [00:01<00:00, 22292.46it/s]
```

```
22445
300
```

TFIDF weighted W2V for Test Data(Titles feature)

In [0]:

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

In [72]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_test): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_test.append(vector)

print(len(tfidf_w2v_vectors_titles_test))
print(len(tfidf_w2v_vectors_titles_test[0]))
```

```
100%|██████████| 16500/16500 [00:00<00:00, 21829.90it/s]
```

```
16500
300
```

TFIDF weighted W2V for Cross Validation Data(Titles feature)

In [0]:

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

In [74]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_titles_cv = []; # the avg-w2v for each sentence/review is stored in this list
```

```
for sentence in tqdm(preprocessed_titles_cv): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_cv.append(vector)

print(len(tfidf_w2v_vectors_titles_cv))
print(len(tfidf_w2v_vectors_titles_cv[0]))
```

```
100%|██████████| 11055/11055 [00:00<00:00, 21159.98it/s]
```

```
11055
300
```

1.5.3 Vectorizing Numerical features

Price Feature

In [0]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
#Now join price data to Train,Test and Cross Validation Data
X_train = pd.merge(X_train, price_data, on='id', how='left')
X_test = pd.merge(X_test, price_data, on='id', how='left')
X_cv = pd.merge(X_cv, price_data, on='id', how='left')
```

In [59]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

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

print("After Column Standardisation: ")
print(price_standardized_train.shape, y_train.shape)
print(price_standardized_cv.shape, y_cv.shape)
print(price_standardized_test.shape, y_test.shape)
```

```
Mean : 299.4865769659167, Standard deviation : 374.7471682242887
After Column Standardisation:
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Quantity feature

In [60]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

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

print("After Column Standardisation: ")
print(quantity_standardized_train.shape, y_train.shape)
print(quantity_standardized_cv.shape, y_cv.shape)
print(quantity_standardized_test.shape, y_test.shape)
```

```
Mean : 17.15388728001782, Standard deviation : 27.295131682392313
After Column Standardisation:
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Number of Previosly Proposed Project by Teacher Feature

In [61]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

ppt_scalar = StandardScaler()
ppt_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)) # find
ing the mean and standard deviation of this data
print(f"Mean : {ppt_scalar.mean_[0]}, Standard deviation : {np.sqrt(ppt_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
ppt_standardized_train =
ppt_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
1))
ppt_standardized_test = ppt_scalar.transform(X_test['teacher_number_of_previously_posted_projects'
].values.reshape(-1, 1))
ppt_standardized_cv = ppt_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].va
lues.reshape(-1, 1))

print("After Column Standardisation: ")
```

```
print(ppt_standardized_train.shape, y_train.shape)
print(ppt_standardized_cv.shape, y_cv.shape)
print(ppt_standardized_test.shape, y_test.shape)
```

```
Mean : 10.846825573624415, Standard deviation : 26.720872959931416
After Column Standardisation:
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Title Word Count Feature

In [62]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
twc_standardized_train = twc_scalar.transform(X_train['title_word_count'].values.reshape(-1, 1))
twc_standardized_test = twc_scalar.transform(X_test['title_word_count'].values.reshape(-1, 1))
twc_standardized_cv = twc_scalar.transform(X_cv['title_word_count'].values.reshape(-1, 1))

print("After Column Standardisation: ")
print(twc_standardized_train.shape, y_train.shape)
print(twc_standardized_cv.shape, y_cv.shape)
print(twc_standardized_test.shape, y_test.shape)
```

```
Mean : 5.161149476498107, Standard deviation : 2.098510884516714
After Column Standardisation:
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Essay Word Count Feature

In [63]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above mean and variance.
```

```
ewc_standardized_train = ewc_scalar.transform(X_train['essay_word_count'].values.reshape(-1, 1))
ewc_standardized_test = ewc_scalar.transform(X_test['essay_word_count'].values.reshape(-1, 1))
ewc_standardized_cv = ewc_scalar.transform(X_cv['essay_word_count'].values.reshape(-1, 1))

print("After Column Standardisation: ")
print(ewc_standardized_train.shape, y_train.shape)
print(ewc_standardized_cv.shape, y_cv.shape)
print(ewc_standardized_test.shape, y_test.shape)
```

```
Mean : 254.8482512809089, Standard deviation : 64.89292554906126
After Column Standardisation:
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)
2. **Hyper paramter tuning to find best K**
   - Find the best hyper parameter which results in the maximum AUC value
   - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
   - Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task
3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
   - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points
4. **[Task-2]**
   - Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features
     - ```
       from sklearn.datasets import load_digits
       from sklearn.feature_selection import SelectKBest, chi2
       X, y = load_digits(return_X_y=True)
       X.shape
       X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
       X_new.shape
       ========
       output:
       (1797, 64)
       (1797, 20)
       ```
   - Repeat the steps 2 and 3 on the data matrix after feature selection
5. **Conclusion**
   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage
       ```

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

2. K Nearest Neighbor

Task 1:

Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)

In [0]:

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

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_standardized_train, quantity_standardized_train, pp
t_standardized_train, twc_standardized_train, ewc_standardized_train, title_bow_train,
text_bow_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_standardized_test, quantity_standardized_test,
ppt_standardized_test, twc_standardized_test, ewc_standardized_test, text_bow_test, title_bow_test)
).tocsr()
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv,price_standardized_cv, quantity_standardized_cv,
ppt_standardized_cv, twc_standardized_cv, ewc_standardized_cv, title_bow_cv, text_bow_cv)).tocsr()
```

In [0]:

```
print("Final Shape of the Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final Shape of the Data matrix
(22445, 10095) (22445,)
(11055, 10095) (11055,)
(16500, 10095) (16500,)
```

A] Find the best hyper parameter which results in the maximum AUC value:

In [0]:

```
def batch_predict(clf, data):
    #clf=classifier ; data= training,test data

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 10134 then your cr_loop will be 10134 - 10134%1000 = 10000
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [0]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
```

```
train_auc = []
cv_auc = []


K = [1, 5, 10, 31,91,121,151]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)    #for train error
    y_cv_pred = batch_predict(neigh, X_cr)      #for cv error


    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

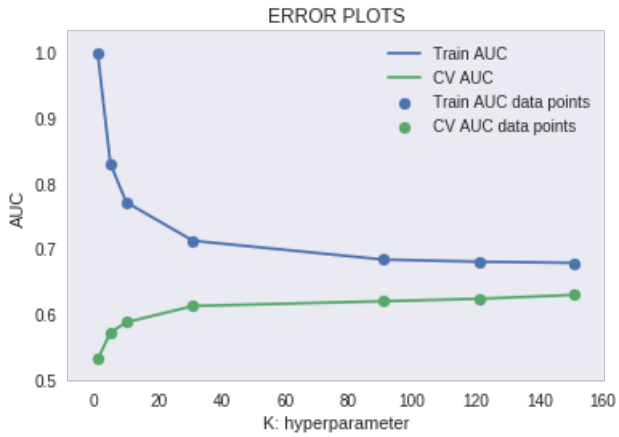


plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC data points')
plt.scatter(K, cv_auc, label='CV AUC data points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 7/7 [13:47<00:00, 119.05s/it]
```



In [0]:

```
print(K)
print(cv_auc)
```

```
[1, 5, 10, 31, 91, 121, 151]
[0.530060376056581, 0.5708020135806922, 0.5867667288723007, 0.6116452083366005,
0.6190102404064798, 0.6226820769363463, 0.6287132843006571]
```

In [0]:

```
best_k=121
```

B) GridsearchCV

In [0]:

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

neigh = KNeighborsClassifier()

parameters = {'n_neighbors':[1, 5, 10, 31,91,121,151]}

clf = GridSearchCV(neigh, parameters, cv= 5, scoring='roc_auc')

clf.fit(X_tr, y_train)

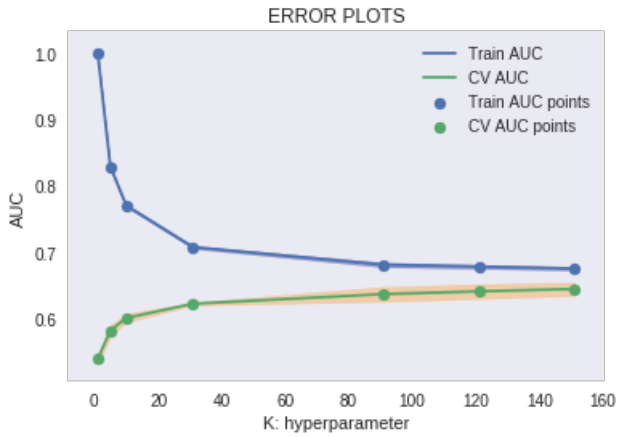
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



RandomizedSearchCV

In [0]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import RandomizedSearchCV

neigh = KNeighborsClassifier()

parameters = {'n_neighbors':[1, 5, 10, 31,91,121,151]}

clf = RandomizedSearchCV(neigh, parameters, cv= 5, scoring='roc_auc')

clf.fit(X_tr, y_train)

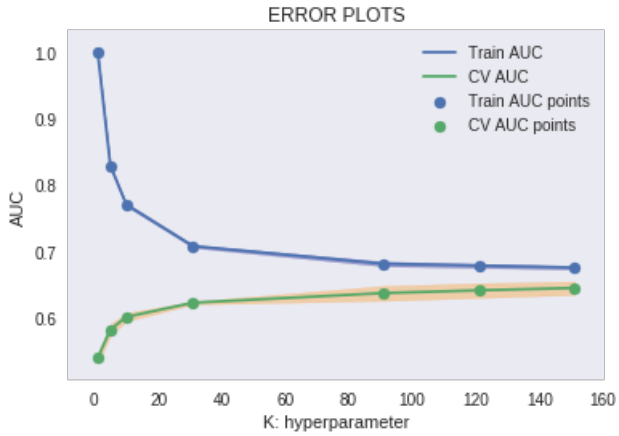
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
```

```
plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



C] Train model using the best hyper-parameter value

In [0]:

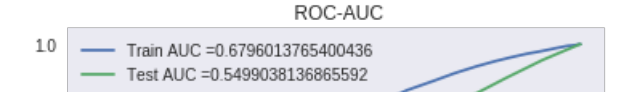
```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

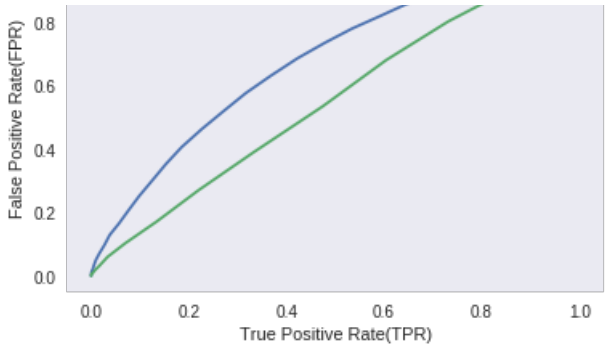

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

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





D) Confusion Matrix

In [0]:

```
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.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

Training Data

In [0]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24954952603609845 for threshold 0.777
[[ 1805  1658]
 [ 5058 13924]]
```

In [0]:

```
conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,
train_fpr, train_fpr)), range(2),range(2))
```

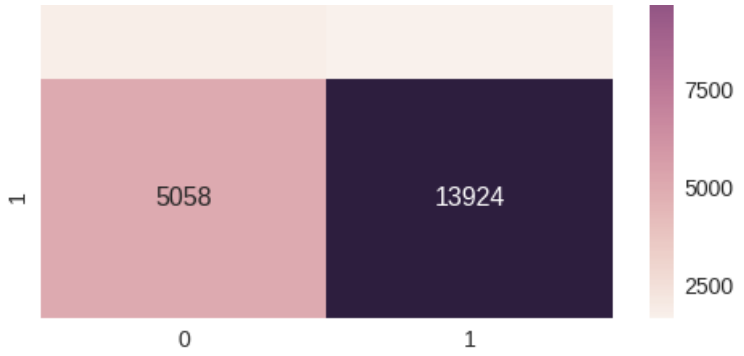
```
the maximum value of tpr*(1-fpr) 0.24954952603609845 for threshold 0.777
```

In [0]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f321b22b0>
```



Test Data

In [0]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24936810757474873 for threshold 0.769
[[ 2369   177]
 [12532  1422]]
```

In [0]:

```
conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_
fpr, test_fpr)), range(2),range(2))
```

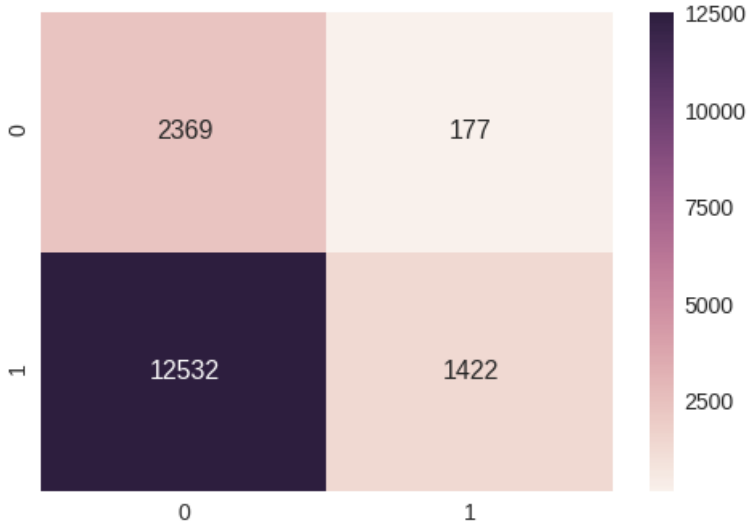
```
the maximum value of tpr*(1-fpr) 0.24936810757474873 for threshold 0.769
```

In [0]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f320998d0>
```



Set 2 : categorical, numerical features + project_title(TFIDF) + preprocessed_essay (TFIDF)

In [0]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
```

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

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_standardized_train, quantity_standardized_train, pp
t_standardized_train, twc_standardized_train, ewc_standardized_train, title_tfidf_train,
text_tfidf_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_standardized_test, quantity_standardized_test,
ppt_standardized_test, twc_standardized_test, ewc_standardized_test, text_tfidf_test,
title_tfidf_test)).tocsr()
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv,price_standardized_cv, quantity_standardized_cv,
ppt_standardized_cv, twc_standardized_cv, ewc_standardized_cv, title_tfidf_cv, text_tfidf_cv)).tocs
r()
```

In [0]:

```
print("Final Tfidf Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final Tfidf Data matrix
(22445, 10095) (22445,)
(11055, 10095) (11055,)
(16500, 10095) (16500,)
```

A] Find the best hyper parameter which results in the maximum AUC value:

In [0]:

```
train_auc = []
cv_auc = []
K = [1, 5, 10, 31,91,121,151]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr, y_train)

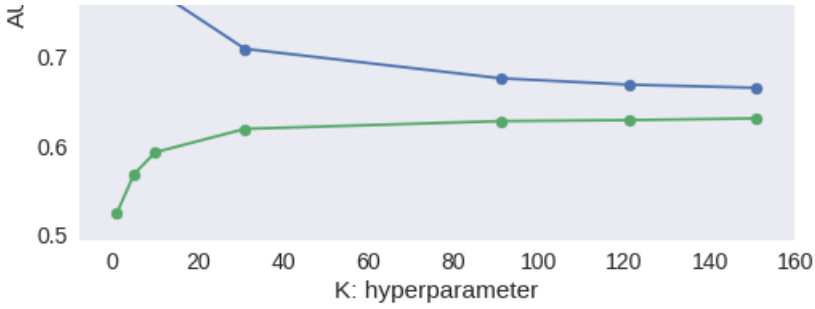
    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC vs K: hyperparameter PLOT")
plt.grid()
plt.show()
```

```
100%|██████████| 7/7 [14:02<00:00, 121.31s/it]
```



In [0]:

```
print(K)
print(cv_auc)
```

```
[1, 5, 10, 31, 91, 121, 151]
[0.51761396 0.54762143 0.56810793 0.60089699 0.62108335 0.62815848
 0.62870977]
```

In [0]:

```
best_k_tfidf = 121
```

RandomisedSearchCV

In [0]:

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

neigh = KNeighborsClassifier()

parameters = {'n_neighbors':[1, 5, 10, 31,91,121,151]}

clf = RandomizedSearchCV(neigh, parameters, cv= 5, scoring='roc_auc')

clf.fit(X_tr, y_train)

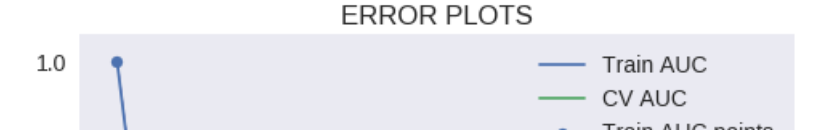
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

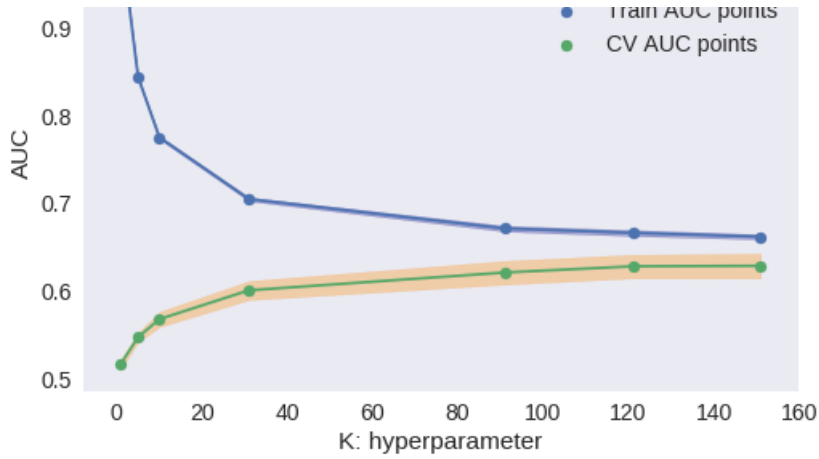
plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```





C) Train model using the best hyper-parameter value

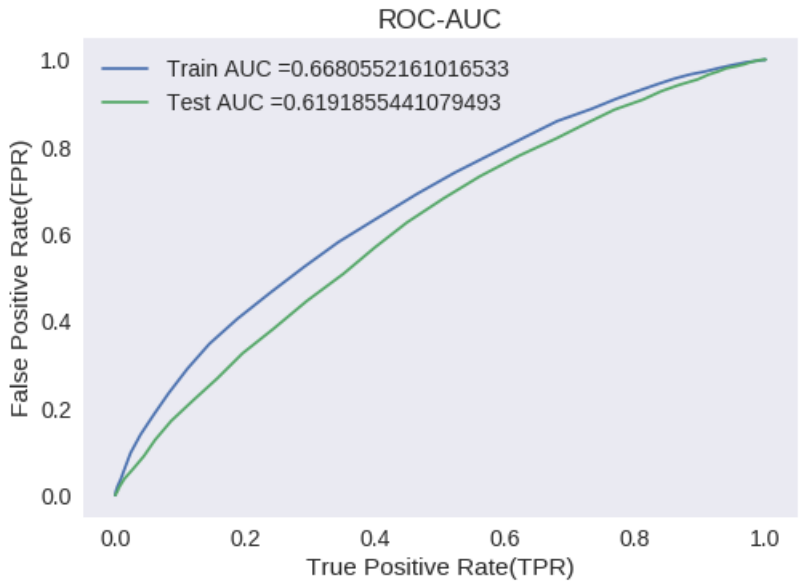
In [0]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
neigh = KNeighborsClassifier(n_neighbors=best_k_tfidf)
neigh.fit(X_tr, y_train)


y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

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



D) Confusion Matrix

Training Data

In [0]:

```
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24954952603609845 for threshold 0.826
[[ 1658  1805]
 [ 4941 14041]]
```

In [0]:

```
conf_matr_df_train1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, t
rain_fpr, train_fpr)), range(2),range(2))
```

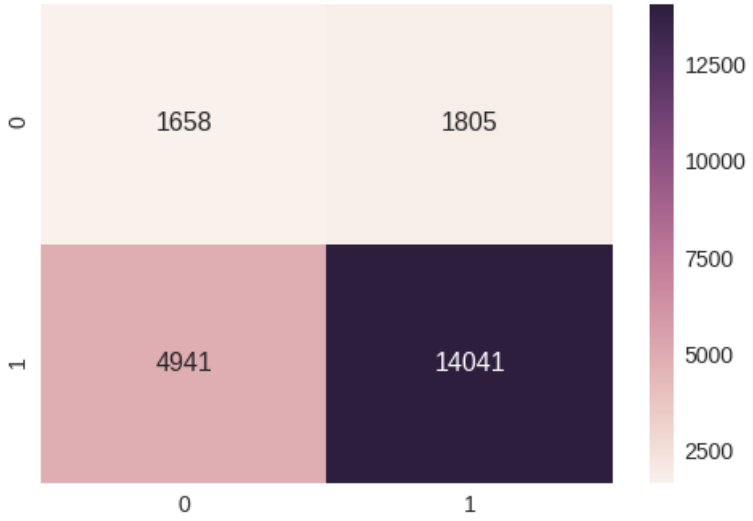
```
the maximum value of tpr*(1-fpr) 0.24954952603609845 for threshold 0.826
```

In [0]:

```
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f322137f0>
```



Test Data

In [0]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24997778503192475 for threshold 0.835
[[1261 1285]
 [4450 9504]]
```

In [0]:

```
conf_matr_df_test1 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)), range(2),range(2))
```

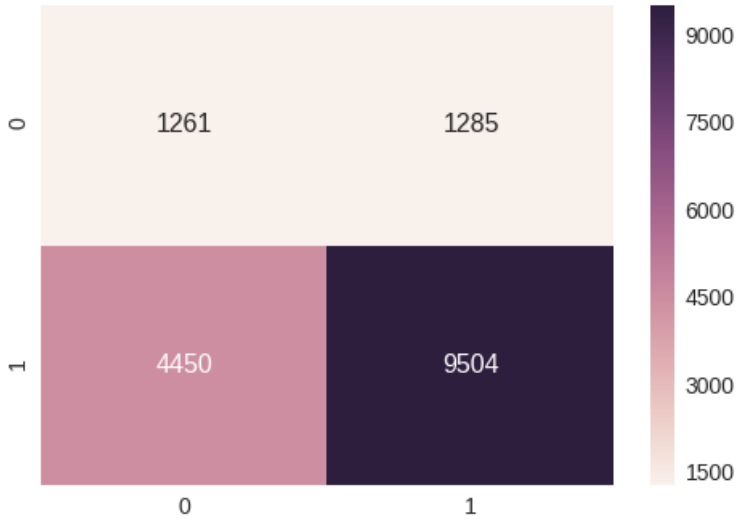
```
the maximum value of tpr*(1-fpr) 0.24997778503192475 for threshold 0.835
```

In [0]:

```
sns.set(font_scale=1.4)
sns.heatmap(conf_matr_df_test1, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f324a6748>
```



Set 3 : categorical, numerical features + project_title(AVG W2V) + preprocessed_essay (AVG W2V)

In [0]:

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

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_standardized_train, quantity_standardized_train, pp
t_standardized_train, twc_standardized_train, ewc_standardized_train, avg_w2v_vectors_train,
avg_w2v_vectors_titles_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_standardized_test, quantity_standardized_test,
ppt_standardized_test, twc_standardized_test, ewc_standardized_test, avg_w2v_vectors_test,
avg_w2v_vectors_titles_test)).tocsr()
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv,price_standardized_cv, quantity_standardized_cv,
ppt_standardized_cv, twc_standardized_cv, ewc_standardized_cv, avg_w2v_vectors_cv, avg_w2v_vectors_
titles_cv)).tocsr()
```

In [65]:

```
print("Final Avg W2V Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final Avg W2V Data matrix
(22445, 705) (22445,)
(11055, 705) (11055,)
(16500, 705) (16500,)
```

RandomisedSearch CV

In [67]:

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

```
from sklearn.model_selection import RandomizedSearchCV

neigh = KNeighborsClassifier()

parameters = {'n_neighbors':[1, 5, 10, 31,91,121,151]}

clf = RandomizedSearchCV(neigh, 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']

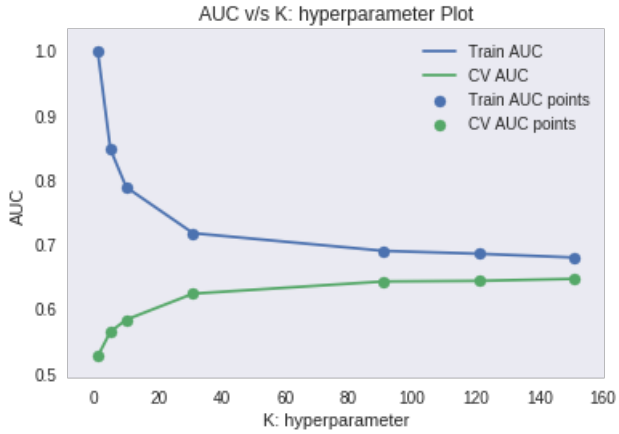
plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC v/s K: hyperparameter Plot")
plt.grid()
plt.show()
```

```
100%|██████████| 7/7 [4:39:51<00:00, 2399.81s/it]
```



In [0]:

```
K=[1, 5, 10, 31, 91, 121, 151]
print(K)
print(cv_auc)
```

```
[1, 5, 10, 31, 91, 121, 151]
[0.5297843712264965, 0.5678350557498394, 0.5826314237771889, 0.6020778145435727,
0.6251916822180751, 0.630472030987815, 0.6301186820769363]
```

In [0]:

```
best_k_avgw2v = 121
```

C) Train model using the best hyper-parameter value

In [0]:

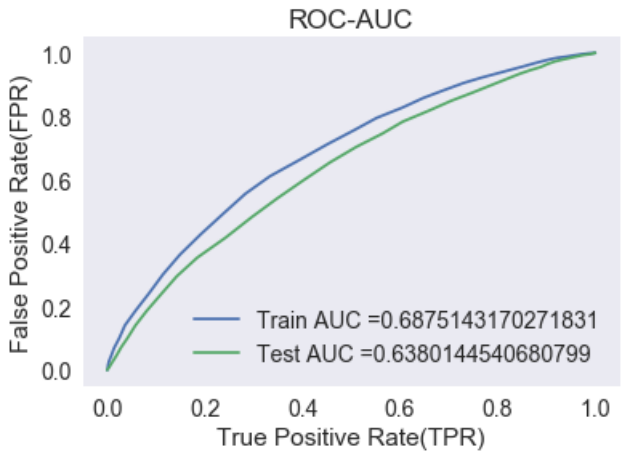
```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve

neigh = KNeighborsClassifier(n_neighbors=best_k_avgw2v)
neigh.fit(X_tr, y_train)

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

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



D) Confusion Matrix

Training Data

In [0]:

```
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24997446292721648 for threshold 0.826
[[ 1714  1749]
 [ 4638 14344]]
```

In [0]:

```
conf_matr_df_train2 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, t
rain_fpr, train_fpr)), range(2),range(2))
```

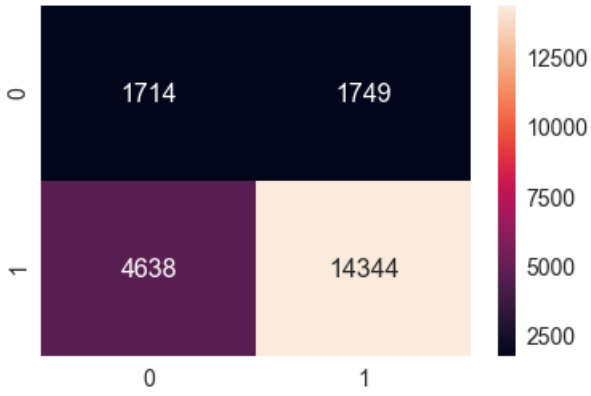
```
the maximum value of tpr*(1-fpr) 0.24997446292721648 for threshold 0.826
```

In [0]:

```
sns.set(font_scale=1.4)
sns.heatmap(conf_matr_df_train2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20d5065dd8>
```



Test Data

In [0]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24989571306653569 for threshold 0.843
[[1385 1161]
 [4824 9130]]
```

In [0]:

```
conf_matr_df_test2 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)), range(2),range(2))
```

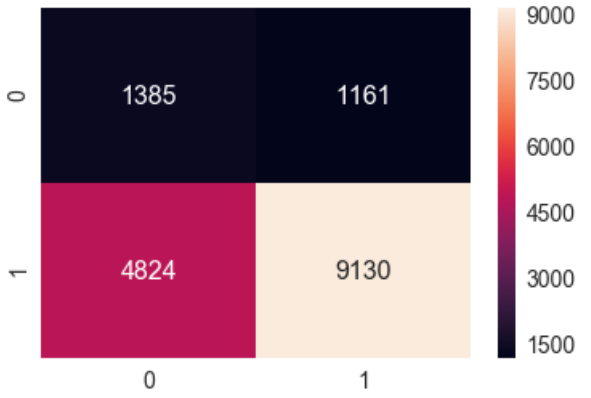
```
the maximum value of tpr*(1-fpr) 0.24989571306653569 for threshold 0.843
```

In [0]:

```
sns.set(font_scale=1.4)
sns.heatmap(conf_matr_df_test2, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20cadd7278>
```



Set 4 : categorical, numerical features + project_title(TFIDF W2V) + preprocessed_essay (TFIDF W2V)

In [0]:

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

```
X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_standardized_train, quantity_standardized_train, pp
t_standardized_train, twc_standardized_train, ewc_standardized_train, tfidf_w2v_vectors_train, tfi
df_w2v_vectors_titles_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_standardized_test, quantity_standardized_test,
ppt_standardized_test, twc_standardized_test, ewc_standardized_test, tfidf_w2v_vectors_test,
tfidf_w2v_vectors_titles_test)).tocsr()
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv,price_standardized_cv, quantity_standardized_cv,
ppt_standardized_cv, twc_standardized_cv, ewc_standardized_cv, tfidf_w2v_vectors_cv, tfidf_w2v_vect
ors_titles_cv)).tocsr()
```

In [82]:

```
print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data matrix
(22445, 705) (22445,)
(11055, 705) (11055,)
(16500, 705) (16500,)
```

RandomisedSearchCV

In [0]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import RandomizedSearchCV

neigh = KNeighborsClassifier()

parameters = {'n_neighbors':[1, 5, 10, 31,91,121,151]}

clf = RandomizedSearchCV(neigh, 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']

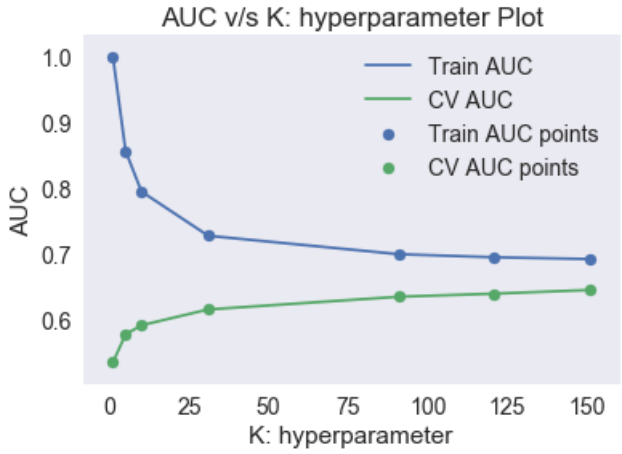
plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.3,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC v/s K: hyperparameter")
plt.grid()
plt.show()
```

```
100%|██████████| 7/7 [3:14:23<00:00, 1432.34s/it]
```



In [0]:

```
K=[1, 5, 10, 31,91,121,151]
print(K)
print(cv_auc)
```

```
[1, 5, 10, 31, 91, 121, 151]
[0.5356494738657926, 0.578330233506359, 0.5920988912760519, 0.6161443379804602,
0.6354036100177208, 0.6400678407326674, 0.6456380573023663]
```

In [0]:

```
best_k_tfidfw2v = 121
```

C) Train model using the best hyper-parameter value

In [0]:

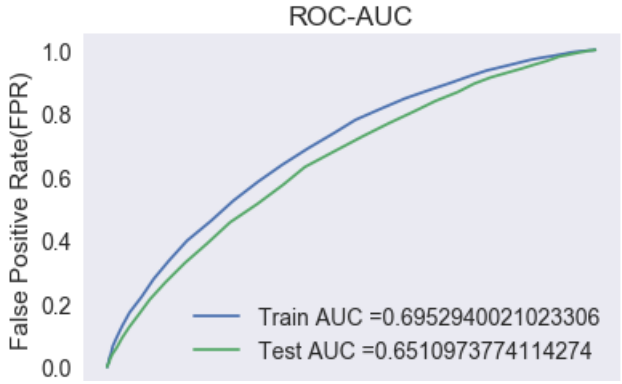
```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve

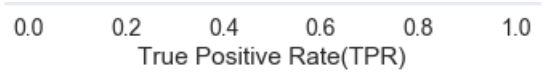
neigh = KNeighborsClassifier(n_neighbors=best_k_tfidfw2v)
neigh.fit(X_tr, y_train)

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

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





D) Confusion Matrix

Training Data

In [0]:

```
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24992743302011472 for threshold 0.818
[[ 1702  1761]
 [ 4189 14793]]
```

In [0]:

```
conf_matr_df_train3 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, t
rain_fpr, train_fpr)), range(2),range(2))
```

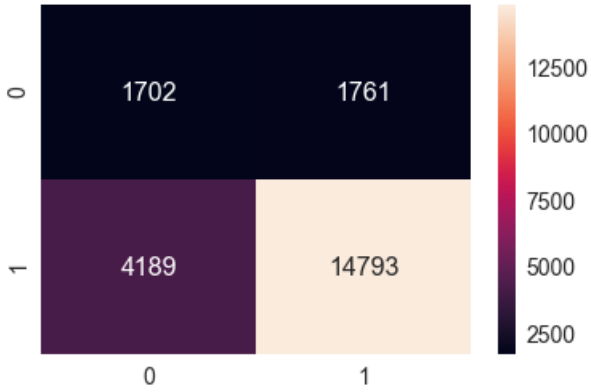
```
the maximum value of tpr*(1-fpr) 0.24992743302011472 for threshold 0.818
```

In [0]:

```
sns.set(font_scale=1.4)
sns.heatmap(conf_matr_df_train3, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20d66441d0>
```



Test Data

In [0]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2494069837688804 for threshold 0.835
[[1356 1190]
 [4439 9515]]
```

In [0]:

```
conf_matr_df_test3 = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test
_fpr, test_fpr)), range(2),range(2))
```

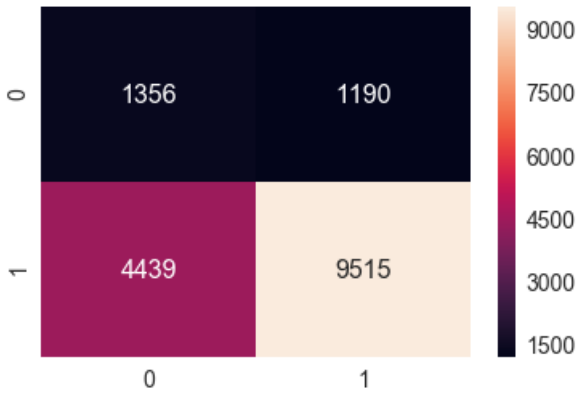
```
the maximum value of tpr*(1-fpr) 0.2494069837688804 for threshold 0.835
```

In [0]:

```
sns.set(font_scale=1.4)
sns.heatmap(conf_matr_df_test3, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20d5035828>
```



Task-2

Feature selection with `SelectKBest`

In [0]:

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

X_tr = hstack((categories_one_hot_train, sub_categories_one_hot_train,
school_state_categories_one_hot_train, project_grade_categories_one_hot_train,
teacher_prefix_categories_one_hot_train, price_standardized_train, quantity_standardized_train, pp
t_standardized_train, twc_standardized_train, ewc_standardized_train, title_tfidf_train,
text_tfidf_train)).tocsr()
X_te = hstack((categories_one_hot_test, sub_categories_one_hot_test,
school_state_categories_one_hot_test, project_grade_categories_one_hot_test,
teacher_prefix_categories_one_hot_test, price_standardized_test, quantity_standardized_test,
ppt_standardized_test, twc_standardized_test, ewc_standardized_test, text_tfidf_test,
title_tfidf_test)).tocsr()
X_cr = hstack((categories_one_hot_cv, sub_categories_one_hot_cv,
school_state_categories_one_hot_cv, project_grade_categories_one_hot_cv,
teacher_prefix_categories_one_hot_cv,price_standardized_cv, quantity_standardized_cv,
ppt_standardized_cv, twc_standardized_cv, ewc_standardized_cv, title_tfidf_cv, text_tfidf_cv)).tocs
r()
```

In [0]:

```
from sklearn.feature_selection import SelectKBest, chi2

X_tr_new = SelectKBest(chi2, k=2000).fit_transform(abs(X_tr), y_train)
X_te_new = SelectKBest(chi2, k=2000).fit_transform(abs(X_te), y_test)
X_cr_new = SelectKBest(chi2, k=2000).fit_transform(abs(X_cr), y_cv)
```

In [0]:

```
print("Final Data matrix")
print(X_tr_new.shape, y_train.shape)
print(X_cr_new.shape, y_cv.shape)
print(X_te_new.shape, y_test.shape)
```

```
Final Data matrix
```

```
Final Data matrix
(22445, 2000) (22445,)
(11055, 2000) (11055,)
(16500, 2000) (16500,)
```

RandomisedSearchCV

In [0]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import RandomizedSearchCV
neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 31,91,121,151]}
clf = RandomizedSearchCV(neigh, parameters, cv=5, scoring='roc_auc')
clf.fit(X_tr, y_train)

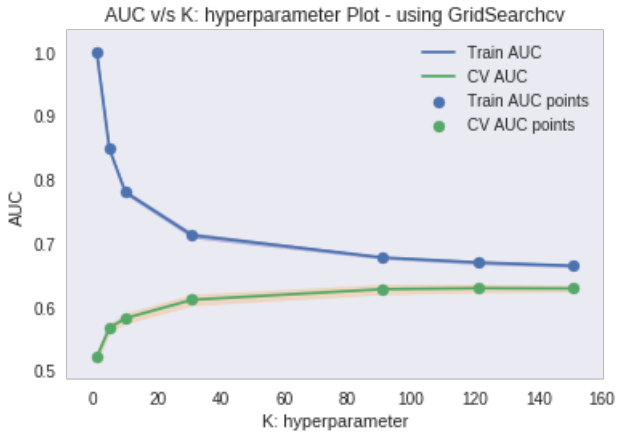
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='darkblue')

plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,
color='darkorange')

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC v/s K: hyperparameter Plot - using GridSearchcv")
plt.grid()
plt.show()
```



In [0]:

```
best_k_select = 91
```

C) Train model using the best hyper-parameter value

In [0]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve

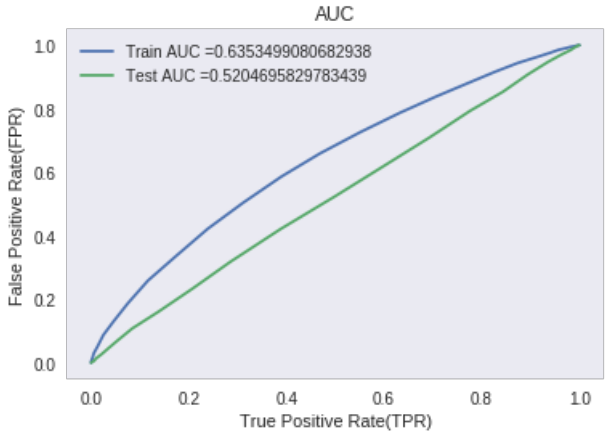
neigh = KNeighborsClassifier(n_neighbors=best_k_select)
```

```
neigh.fit(X_tr_new, y_train)

y_train_pred = batch_predict(neigh, X_tr_new)
y_test_pred = batch_predict(neigh, X_te_new)

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

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



D) Confusion Matrix

Training Data

In [0]:

```
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24995778565519455 for threshold 0.824
[[ 1709  1754]
 [ 5710 13272]]
```

In [0]:

```
conf_matr_df_train_sk = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds
, train_fpr, train_fpr)), range(2),range(2))
```

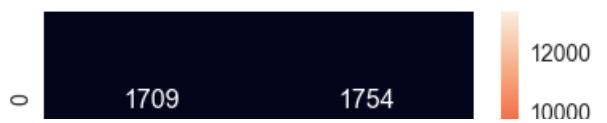
```
the maximum value of tpr*(1-fpr) 0.24995778565519455 for threshold 0.824
```

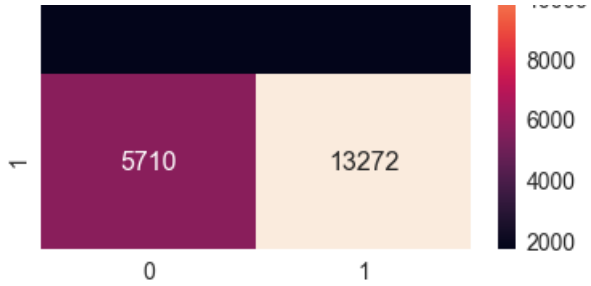
In [0]:

```
sns.set(font_scale=1.4)
sns.heatmap(conf_matr_df_train_sk, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20d6657ef0>
```





Test Data

In [0]:

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24888539483094718 for threshold 0.868
[[1770  776]
 [9429 4525]]
```

In [0]:

```
conf_matr_df_test_sk = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, te
st_fpr, test_fpr)), range(2),range(2))
```

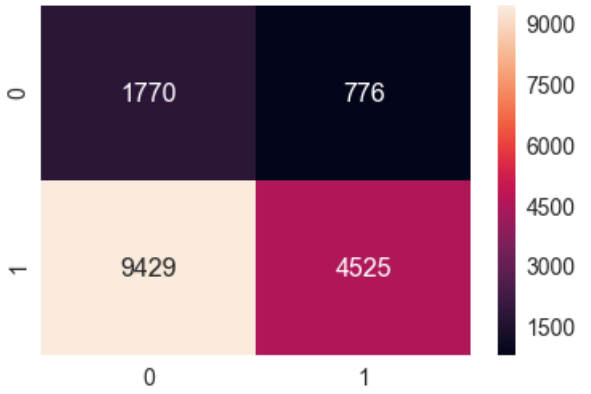
```
the maximum value of tpr*(1-fpr) 0.24888539483094718 for threshold 0.868
```

In [0]:

```
sns.set(font_scale=1.4)
sns.heatmap(conf_matr_df_test_sk, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[0]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20d66f40f0>
```



Conclusion

In [0]:

```
# Compare all your models using Prettytable library

# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]
```

```
x.add_row(["BOW", "Brute Force", 121, 0.63])
x.add_row(["TFIDF", "Brute Force", 121, 0.62])
x.add_row(["AVG W2V", "Brute Force", 91, 0.63])
x.add_row(["TFIDF W2V", "Brute Force", 121, 0.69])
x.add_row(["TFIDF", "Top 2000", 91, 0.51])

print(x)
```

```
+------------+-------------+-----------------+------+
| Vectorizer |    Model    | Hyper Parameter | AUC  |
+------------+-------------+-----------------+------+
|    BOW     | Brute Force |       121       | 0.63 |
|   TFIDF    | Brute Force |       121       | 0.62 |
|  AVG W2V   | Brute Force |        91       | 0.63 |
| TFIDF W2V  | Brute Force |       121       | 0.69 |
|   TFIDF    |   Top 2000  |        91       | 0.51 |
+------------+-------------+-----------------+------+
```