# 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:**<br>• `Art Will Make You Happy!`<br>• `First Grade Fun` |
| `project_grade_category` | Grade level of students for which the project is targeted. One of the following enumerated values:<br>• `Grades PreK-2`<br>• `Grades 3-5`<br>• `Grades 6-8`<br>• `Grades 9-12` |
| `project_subject_categories` | One or more (comma-separated) subject categories for the project from the following enumerated list of values:<br>• `Applied Learning`<br>• `Care & Hunger`<br>• `Health & Sports`<br>• `History & Civics`<br>• `Literacy & Language`<br>• `Math & Science`<br>• `Music & The Arts`<br>• `Special Needs`<br>• `Warmth`<br>**Examples:**<br>• `Music & The Arts`<br>• `Literacy & Language, Math & Science` |
| `school_state` | State where school is located (Two-letter U.S. postal code (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **Example:** `WY` |
| `project_subject_subcategories` | One or more (comma-separated) subject subcategories for the project. **Examples:**<br>• `Literacy`<br>• `Literature & Writing, Social Sciences` |
| `project_resource_summary` | An explanation of the resources needed for the project. **Example:**<br>• `My students need hands on literacy materials to manage sensory needs!</code` |

| Feature | Description |
|---|---|
| **project_essay_1** | First application essay* |
| **project_essay_2** | Second application essay* |
| **project_essay_3** | Third application essay* |
| **project_essay_4** | Fourth application essay* |
| **project_submitted_datetime** | Datetime when project application was submitted. **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:<br>• `nan`<br>• `Dr.`<br>• `Mr.`<br>• `Mrs.`<br>• `Ms.`<br>• `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. |

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

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.model_selection import train_test_split
from nltk.stem.wordnet import WordNetLemmatizer
from scipy.sparse  import hstack
import xgboost as xgb
from sklearn.metrics import confusion_matrix
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
from sklearn.metrics import roc_auc_score
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from chart_studio import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

## 1.1 Reading Data

In [2]:

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

In [3]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

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

In [4]:
```
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[4]:

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

## 1.1 Preprocessing Categorical Features: project_grade_category

In [5]:
```
print("Project grade" ,project_data['project_grade_category'].value_counts(dropna=False))
## visulaize how project grade looks like
print('-'*50)
print(project_data['project_grade_category'].values[1000])
print(project_data['project_grade_category'].values[1500])
```

```
Project grade Grades PreK-2    44225
Grades 3-5       37137
Grades 6-8       16923
Grades 9-12      10963
Name: project_grade_category, dtype: int64
--------------------------------------------------
Grades 3-5
Grades PreK-2
```

In [6]:
```
# https://stackoverflow.com/questions/36383821/pandas-dataframe-apply-function-to-column-strings-based-on-other-column-value
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace('Grades ','')
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace(' ','_')
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace('-','_')
project_data['project_grade_category'] = project_data['project_grade_category'].str.lower()
project_data['project_grade_category'].value_counts()
```

Out[6]:
```
prek_2    44225
3_5       37137
6_8       16923
9_12      10963
Name: project_grade_category, dtype: int64
```

### 1.2 Preprocessing Categorical Features: project_subject_category

In [7]:
```
categories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039
# reference from course material : reference_EDA.ipynb
# 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

def _process_cat_subcat(categories):
    cat_list = []
    for i in categories:
        temp = ""
        # consider we have text like this "Math & Science, Warmth, Care & Hunger"
        for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
            if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Science"
                j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
            j = j.replace(' ','') # we are placeing all the ' '(space) with ''(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())
    return cat_list



project_data['clean_categories'] = _process_cat_subcat(categories)
project_data.drop(['project_subject_categories'], axis=1, inplace=True)
project_data.head(2)

### maintain a dict that
my_counter=Counter()
for word in project_data['clean_categories'].values:
        my_counter.update(word.split())
cat_dict=dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))


```

### 1.3 Preprocessing Categorical Features: project_subject_category

In [8]:

```
sub_categories = list(project_data['project_subject_subcategories'].values)
project_data['clean_subcategories'] = _process_cat_subcat(sub_categories)
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)

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

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

## 1.4 Preprocessing Categorical Features: school_state

In [9]:

```
project_data['school_state'].value_counts()
## Convert it to lower
project_data['school_state'] = project_data['school_state'].str.lower()
print(project_data['school_state'].value_counts(dropna=False))
print('No nan values in this feature')
```

```
ca    15388
tx     7396
ny     7318
fl     6185
nc     5091
il     4350
ga     3963
sc     3936
mi     3161
pa     3109
in     2620
mo     2576
oh     2467
la     2394
ma     2389
wa     2334
ok     2276
nj     2237
az     2147
va     2045
wi     1827
al     1762
ut     1731
tn     1688
ct     1663
md     1514
nv     1367
ms     1323
ky     1304
or     1242
mn     1208
co     1111
ar     1049
id      693
ia      666
ks      634
nm      557
dc      516
hi      507
me      505
wv      503
nh      348
ak      345
de      343
ne      309
sd      300
ri      285
mt      245
nd      143
wy       98
vt       80
Name: school_state, dtype: int64
No nan values in this feature
```

## 1.5 Preprocessing Categorical Features: Teacher_prefix

In [10]:
```
print(project_data['teacher_prefix'].value_counts(dropna=False))
# try to remove the dots from the teacher prefix and replace nan with mrs.
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna('Mrs.')
project_data['teacher_prefix']=project_data['teacher_prefix'].str.replace('.','')
project_data['teacher_prefix']=project_data['teacher_prefix'].str.lower()
project_data['teacher_prefix']=project_data['teacher_prefix'].str.strip()
```

```
Mrs.       57269
Ms.        38955
Mr.        10648
Teacher     2360
Dr.           13
NaN            3
Name: teacher_prefix, dtype: int64
```

## 1.6 Combining all the essays

In [11]:
```
print('Number of nan values in essay1 is ' ,len(project_data[project_data["project_essay_1"].isna()==True]))
print('Number of nan values in essay2 is ' ,len(project_data[project_data["project_essay_2"].isna()==True]))
print('Number of nan values in essay3 is ' ,len(project_data[project_data["project_essay_3"].isna()==True]))
print('Number of nan values in essay4 is ' ,len(project_data[project_data["project_essay_4"].isna()==True]))
```

```
Number of nan values in essay1 is  0
Number of nan values in essay2 is  0
Number of nan values in essay3 is  105490
Number of nan values in essay4 is  105490
```

In [12]:
```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

## 1.7 Number of Words in the Essay and Title

In [13]:
```
#source:''''https://www.geeksforgeeks.org/python-program-to-count-words-in-a-sentence/'''
words_counter=[]
for string in project_data['essay']:
    res = len(re.findall(r'\w+', string))
    words_counter.append(res)

project_data["words_in_essay"] = words_counter

words_counter=[]

for string in project_data['project_title']:
    res = len(re.findall(r'\w+', string))
    words_counter.append(res)
project_data["words_in_title"] = words_counter
```

## 1.8. Preprocessing Numerical Values: price

In [14]:
```
## calculate the overall count of resources and the total price for each project id
price_data=resource_data.groupby('id',as_index=False).agg({'price':'sum','quantity':'sum' })
##merge into the project_Data
project_data = pd.merge(project_data,price_data,on='id',how='left')
```

### 1.9 Preprocessing Text Features: project_title , essay

In [15]:
```
# https://stackoverflow.com/a/47091490/4084039
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
# 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', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]

print("printing some random reviews")
print(9, project_data['project_title'].values[9])
print(34, project_data['project_title'].values[34])
print(147, project_data['project_title'].values[147])
```

```
printing some random reviews
9 Just For the Love of Reading--\r\nPure Pleasure
34 \"Have A Ball!!!\"
147 Who needs a Chromebook?\r\nWE DO!!
```

In [16]:
```
# Combining all the above statements
# stemming the words
from nltk.stem import SnowballStemmer
sno=SnowballStemmer('english')
def preprocess_text(text_data):
    preprocessed_text = []
    # tqdm is for printing the status bar
    for sentance in tqdm(text_data):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\n', ' ')
        sent = sent.replace('\\"', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(sno.stem(e) for e in sent.split() if e.lower() not in stopwords)
        preprocessed_text.append(sent.lower().strip())
    return preprocessed_text
```

In [17]:
```
preprocessed_titles = preprocess_text(project_data['project_title'].values)
#merge the column in the project_data
project_data['processed_title']=preprocessed_titles
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 109248/109248 [00:08<00:00, 13383.18it/s]
```

In [18]:
```
print("printing some random reviews")
print(9, preprocessed_titles[9])
print(34, preprocessed_titles[34])
print(147, preprocessed_titles[147])
```

```
printing some random reviews
9 love read pure pleasur
34 ball
147 need chromebook
```

In [19]:
```
print("printing some random reviews")
print(9, project_data['project_title'].values[9])
print(34, project_data['project_title'].values[34])
print(147, project_data['project_title'].values[147])
preprocessed_essays = preprocess_text(project_data['essay'].values)
```

```
printing some random reviews
9 Just For the Love of Reading--\r\nPure Pleasure
34 \"Have A Ball!!!\"
147 Who needs a Chromebook?\r\nWE DO!!

100%|██████████████████████████████████████████████████████████████████████████████| 109248/109248 [04:36<00:00, 394.77it/s]
```

In [20]:
```
print("printing some random essay")
print(9, preprocessed_essays[9])
print('-'*50)
print(34, preprocessed_essays[34])
print('-'*50)
print(147, preprocessed_essays[147])

#merge the column in the project_data
project_data['processed_essay']=preprocessed_essays
```

```
printing some random essay
9 95 student free reduc lunch homeless despit come school eager learn student inquisit eager learner embrac challeng not great book resourc everi day mani not afford opportun e
ngag big color page book regular basi home not travel public librari duti teacher provid student opportun succeed everi aspect life read fundament student read book boost compr
ehens skill book use read aloud partner read independ read engag read build love read read pure enjoy introduc new author well old favorit want student readi 21st centuri know
pleasur hold good hard back book hand noth like good book read student soar read consider generous fund contribut help build stamina prepar 3rd grade thank much read propos nan
nan
--------------------------------------------------
34 student main come extrem low incom famili major come home parent work full time student school 7 30 6 00 pm 2 30 6 00 pm school program receiv free reduc meal breakfast lunc
h want student feel comfort classroom home mani student take multipl role home well school sometim caretak younger sibl cook babysitt academ friend develop go becom adult consi
d essenti part job model help other gain knowledg posit manner result communiti student love help outsid classroom consist look opportun support learn kind help way excit exper
i altern seat classroom school year studi shown give student option sit classroom increas focus well motiv allow student choic classroom abl explor creat welcom environ altern
classroom seat experi frequent recent year believ along mani other everi child learn differ not appli multipl memor paper written appli space ask work student past ask work lib
rari work carpet answer alway long learn work wherev want yoga ball lap desk abl increas option seat classroom expand imagin space nannan
--------------------------------------------------
147 student eager learn make mark world come titl 1 school need extra love fourth grade student high poverti area still come school everi day get educ tri make fun educ get sch
ool creat care environ student bloom deserv best thank request 1 chromebook access onlin intervent differenti instruct get extra practic chromebook use supplement ela math inst
ruct student play ela math game engag fun well particip assign onlin turn help student improv skill chromebook classroom would not allow student use program pace would ensur st
udent get adequ time use program onlin program especi benefici student special need abl work level well challeng differ materi make student confid abil chromebook would allow s
tudent daili access comput increas comput skill chang live better becom success school access technolog classroom would help bridg achiev gap nannan
```

### 1.9 Creating Sentiment Columns

In [21]:
```
## craete the sentiment columns using
neg=[]
pos=[]
neu=[]
compound=[]
sentiment_model=SentimentIntensityAnalyzer()
for text in tqdm(project_data['processed_essay']):
    pol_scores = sentiment_model.polarity_scores(text)
    neg.append(pol_scores['neg'])
    pos.append(pol_scores['pos'])
    neu.append(pol_scores['neu'])
    compound.append(pol_scores['compound'])

project_data['pos']=pos
project_data['neg']=neg
project_data['neu']=neu
project_data['compound']=compound
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 109248/109248 [02:38<00:00, 687.96it/s]
```

## 2 Train,Test,CV Split

In [22]:
```
# train test split using sklearn.model selection
sampled_data =  project_data.sample(50000)
X_train, X_test, y_train, y_test = train_test_split(sampled_data, sampled_data['project_is_approved'], test_size=0.20, stratify = sampled_data['project_is_approved'],random_
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train,random_state=0)
```

In [23]:
```
## drop the y labels from splits
X_train.drop(['project_is_approved'], axis=1, inplace=True)
```

In [24]:
```
X_train.head(2)
```

Out[24]:

| | Unnamed: 0 | id | teacher_id | teacher_prefix | school_state | project_submitted_datetime | project_grade_category | project_title | project_essay_1 | project_essay_2 | ... | words_in_essay | words |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **69191** | 102275 | p200146 | db88eb4e11230df8462b304830e627e2 | mrs | ca | 2016-08-25 19:19:04 | 6_8 | Classroom Chromebook Set | I teach middle school and they are already tec... | Our school shares a set of Chromebooks between... | ... | 232 | |
| **90301** | 23896 | p218117 | 4d283fd47f0fe5396e141369003fd47d | mr | va | 2016-06-13 22:13:59 | 3_5 | SHOW IT MOVE IT | My students learn by best when they are presen... | Having an interactive whiteboard in our classr... | ... | 211 | |

2 rows × 27 columns

## 3. VECTORIZING DATA

### 3.1 One hot encoding on Categorical : (categories,subcategories,schoolstate,teacher_prefix,projectgrade,sentiment_columns)

In [25]:
```
# we use count vectorizer to convert the values into one hot vectors
## clean categories

cat_vectorize = CountVectorizer(lowercase=False, binary=True)
cat_vectorize.fit(X_train['clean_categories'].values)

train_categories = cat_vectorize.transform(X_train['clean_categories'].values)
test_categories = cat_vectorize.transform(X_test['clean_categories'].values)
cv_categories = cat_vectorize.transform(X_cv['clean_categories'].values)

print(cat_vectorize.get_feature_names())
print("Shape of matrix of Train data after one hot encoding ",train_categories.shape)
print("Shape of matrix of Test data after one hot encoding ",test_categories.shape)
print("Shape of matrix of CV data after one hot encoding ",cv_categories.shape)
```

```
['AppliedLearning', 'Care_Hunger', 'Health_Sports', 'History_Civics', 'Literacy_Language', 'Math_Science', 'Music_Arts', 'SpecialNeeds', 'Warmth']
Shape of matrix of Train data after one hot encoding  (26800, 9)
Shape of matrix of Test data after one hot encoding  (10000, 9)
Shape of matrix of CV data after one hot encoding  (13200, 9)
```

In [26]:

```
# we use count vectorizer to convert the values into one hot vectors
## clean subcategories

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

train_subcategories = subcat_vectorize.transform(X_train['clean_subcategories'].values)
test_subcategories = subcat_vectorize.transform(X_test['clean_subcategories'].values)
cv_subcategories = subcat_vectorize.transform(X_cv['clean_subcategories'].values)

print(subcat_vectorize.get_feature_names())
print("Shape of matrix of Train data after one hot encoding ",train_subcategories.shape)
print("Shape of matrix of Test data after one hot encoding ",test_subcategories.shape)
print("Shape of matrix of CV data after one hot encoding ",cv_subcategories.shape)

```

```
['AppliedSciences', 'Care_Hunger', 'CharacterEducation', 'Civics_Government', 'College_CareerPrep', 'CommunityService', 'ESL', 'EarlyDevelopment', 'Economics', 'EnvironmentalSc
ience', 'Extracurricular', 'FinancialLiteracy', 'ForeignLanguages', 'Gym_Fitness', 'Health_LifeScience', 'Health_Wellness', 'History_Geography', 'Literacy', 'Literature_Writin
g', 'Mathematics', 'Music', 'NutritionEducation', 'Other', 'ParentInvolvement', 'PerformingArts', 'SocialSciences', 'SpecialNeeds', 'TeamSports', 'VisualArts', 'Warmth']
Shape of matrix of Train data after one hot encoding  (26800, 30)
Shape of matrix of Test data after one hot encoding  (10000, 30)
Shape of matrix of CV data after one hot encoding  (13200, 30)
```

In [27]:

```
# we use count vectorizer to convert the values into one hot vectors
##  school state


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

sklstate_train = sklstate_vectorize.transform(X_train['school_state'].values)
sklstate_test = sklstate_vectorize.transform(X_test['school_state'].values)
sklstate_cv = sklstate_vectorize.transform(X_cv['school_state'].values)

print(sklstate_vectorize.get_feature_names())
print("Shape of matrix of Train data after one hot encoding ",sklstate_train.shape)
print("Shape of matrix of Test data after one hot encoding ",sklstate_test.shape)
print("Shape of matrix of CV data after one hot encoding ",sklstate_cv.shape)
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd',
'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
Shape of matrix of Train data after one hot encoding  (26800, 51)
Shape of matrix of Test data after one hot encoding  (10000, 51)
Shape of matrix of CV data after one hot encoding  (13200, 51)
```

In [28]:
```
# we use count vectorizer to convert the values into one hot vectors
## teacher_prefix

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

teacher_prefix_train = teacher_prefix_vectorize.transform(X_train['teacher_prefix'].values)
teacher_prefix_test = teacher_prefix_vectorize.transform(X_test['teacher_prefix'].values)
teacher_prefix_cv = teacher_prefix_vectorize.transform(X_cv['teacher_prefix'].values)

print(teacher_prefix_vectorize.get_feature_names())
print("Shape of matrix of Train data after one hot encoding ",teacher_prefix_train.shape)
print("Shape of matrix of Test data after one hot encoding ",teacher_prefix_test.shape)
print("Shape of matrix of CV data after one hot encoding ",teacher_prefix_cv.shape)
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
Shape of matrix of Train data after one hot encoding  (26800, 5)
Shape of matrix of Test data after one hot encoding  (10000, 5)
Shape of matrix of CV data after one hot encoding  (13200, 5)
```

In [29]:
```
# we use count vectorizer to convert the values into one hot vectors
## project_grade

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

proj_grade_train = proj_grade_vectorize.transform(X_train['project_grade_category'].values)
proj_grade_test = proj_grade_vectorize.transform(X_test['project_grade_category'].values)
proj_grade_cv = proj_grade_vectorize.transform(X_cv['project_grade_category'].values)

print(proj_grade_vectorize.get_feature_names())
print("Shape of matrix of Train data after one hot encoding ",proj_grade_train.shape)
print("Shape of matrix of Test data after one hot encoding ",proj_grade_test.shape)
print("Shape of matrix of CV data after one hot encoding ",proj_grade_cv.shape)
```

```
['3_5', '6_8', '9_12', 'prek_2']
Shape of matrix of Train data after one hot encoding  (26800, 4)
Shape of matrix of Test data after one hot encoding  (10000, 4)
Shape of matrix of CV data after one hot encoding  (13200, 4)
```

## 3.2. Vectorizing Numerical Features :

### 3.2.1 Price , Quantity, teacher_number_of_previously_posted_projects,words_in_essay,words_in_title

In [30]:

```
def normalize_numerical(tr,te,cv):
    normalizer = Normalizer()
    # normalizer.fit(tr.values)
    # this will rise an error Expected 2D array, got 1D array instead:
    # array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
    # Reshape your data either using
    # array.reshape(-1, 1) if your data has a single feature
    # array.reshape(1, -1)  if it contains a single sample.
    normalizer.fit(tr.values.reshape(1,-1))

    tr_norm = normalizer.transform(tr.values.reshape(1,-1))
    cv_norm = normalizer.transform(cv.values.reshape(1,-1))
    te_norm = normalizer.transform(te.values.reshape(1,-1))

    ## reshaping
    tr_norm=tr_norm.reshape(-1,1)
    cv_norm=cv_norm.reshape(-1,1)
    te_norm=te_norm.reshape(-1,1)

    print("After vectorizations")
    print(tr_norm.shape, y_train.shape)
    print(cv_norm.shape, y_cv.shape)
    print(te_norm.shape, y_test.shape)
    print("="*100)
    return tr_norm,cv_norm,te_norm
```

In [31]:
```
X_train_price_norm,X_cv_price_norm,X_test_price_norm=normalize_numerical(X_train['price'],X_test['price'],X_cv['price'])
quantity_train_norm,quantity_cv_norm,quantity_test_norm=normalize_numerical(X_train['quantity'],X_test['quantity'],X_cv['quantity'])
prev_projects_train_norm,prev_projects_cv_norm,prev_projects_test_norm=normalize_numerical(X_train['teacher_number_of_previously_posted_projects'],
                                                                                           X_test['teacher_number_of_previously_posted_projects'],
                                                                                           X_cv['teacher_number_of_previously_posted_projects'])
essay_word_count_train_norm,essay_word_count_cv_norm,essay_word_count_test_norm=normalize_numerical(X_train['words_in_essay'],
                                                                                                    X_test['words_in_essay'],
                                                                                                    X_cv['words_in_essay'])
title_word_count_train_norm,title_word_count_cv_norm,title_word_count_test_norm=normalize_numerical(X_train['words_in_title'],
                                                                                                   X_test['words_in_title'],
                                                                                                   X_cv['words_in_title'])



```

```
After vectorizations
(26800, 1) (26800,)
(13200, 1) (13200,)
(10000, 1) (10000,)
==========================================================================================================
After vectorizations
(26800, 1) (26800,)
(13200, 1) (13200,)
(10000, 1) (10000,)
==========================================================================================================
After vectorizations
(26800, 1) (26800,)
(13200, 1) (13200,)
(10000, 1) (10000,)
==========================================================================================================
After vectorizations
(26800, 1) (26800,)
(13200, 1) (13200,)
(10000, 1) (10000,)
==========================================================================================================
```

### 3.2.2 Vectorize Sentiment Columns

In [32]:
```
### vectorize pos
Normalize=Normalizer()
Normalize.fit(X_train['pos'].values.reshape(1,-1))
sentiment_pos_train_norm=Normalize.transform(X_train['pos'].values.reshape(1,-1))
sentiment_pos_test_norm=Normalize.transform(X_test['pos'].values.reshape(1,-1))
sentiment_pos_cv_norm=Normalize.transform(X_cv['pos'].values.reshape(1,-1))

sentiment_pos_train_norm=sentiment_pos_train_norm.reshape(-1,1)
sentiment_pos_test_norm=sentiment_pos_test_norm.reshape(-1,1)
sentiment_pos_cv_norm=sentiment_pos_cv_norm.reshape(-1,1)
print('After vectorizations : ')
print(sentiment_pos_train_norm.shape)
print(sentiment_pos_test_norm.shape)
print(sentiment_pos_cv_norm.shape)
```

```
After vectorizations :
(26800, 1)
(10000, 1)
(13200, 1)
```

In [33]:
```
### vectorize neu
Normalize=Normalizer()
Normalize.fit(X_train['neu'].values.reshape(1,-1))
sentiment_neu_train_norm=Normalize.transform(X_train['neu'].values.reshape(1,-1))
sentiment_neu_test_norm=Normalize.transform(X_test['neu'].values.reshape(1,-1))
sentiment_neu_cv_norm=Normalize.transform(X_cv['neu'].values.reshape(1,-1))
sentiment_neu_train_norm=sentiment_neu_train_norm.reshape(-1,1)
sentiment_neu_test_norm=sentiment_neu_test_norm.reshape(-1,1)
sentiment_neu_cv_norm=sentiment_neu_cv_norm.reshape(-1,1)
print('After vectorizations : ')
print(sentiment_neu_train_norm.shape)
print(sentiment_neu_test_norm.shape)
print(sentiment_neu_cv_norm.shape)
```

```
After vectorizations :
(26800, 1)
(10000, 1)
(13200, 1)
```

In [34]:
```
### vectorize compound
Normalize=Normalizer()
Normalize.fit(X_train['compound'].values.reshape(1,-1))
sentiment_compound_train_norm=Normalize.transform(X_train['compound'].values.reshape(1,-1))
sentiment_compound_test_norm=Normalize.transform(X_test['compound'].values.reshape(1,-1))
sentiment_compound_cv_norm=Normalize.transform(X_cv['compound'].values.reshape(1,-1))
sentiment_compound_train_norm=sentiment_compound_train_norm.reshape(-1,1)
sentiment_compound_test_norm=sentiment_compound_test_norm.reshape(-1,1)
sentiment_compound_cv_norm=sentiment_compound_cv_norm.reshape(-1,1)
print('After vectorizations : ')
print(sentiment_compound_train_norm.shape)
print(sentiment_compound_test_norm.shape)
print(sentiment_compound_cv_norm.shape)
```

```
After vectorizations :
(26800, 1)
(10000, 1)
(13200, 1)
```

In [35]:
```
### vectorize neg
Normalize=Normalizer()
Normalize.fit(X_train['neg'].values.reshape(1,-1))
sentiment_neg_train_norm=Normalize.transform(X_train['neg'].values.reshape(1,-1))
sentiment_neg_test_norm=Normalize.transform(X_test['neg'].values.reshape(1,-1))
sentiment_neg_cv_norm=Normalize.transform(X_cv['neg'].values.reshape(1,-1))
sentiment_neg_train_norm=sentiment_neg_train_norm.reshape(-1,1)
sentiment_neg_test_norm=sentiment_neg_test_norm.reshape(-1,1)
sentiment_neg_cv_norm=sentiment_neg_cv_norm.reshape(-1,1)
print('After vectorizations : ')
print(sentiment_neg_train_norm.shape)
print(sentiment_neg_test_norm.shape)
print(sentiment_neg_cv_norm.shape)
```

```
After vectorizations :
(26800, 1)
(10000, 1)
(13200, 1)
```

## Assignment_11: DonorsChoose_TruncatedSVD

- step 1 Select the top 2k words from essay text and project_title (concatinate essay text with project title and then find the top 2k words) based on their `idf_` (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) values
- step 2 Compute the co-occurance matrix with these 2k words, with window size=5 (ref (https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/))

- step 3 Use TruncatedSVD (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html) on calculated co-occurance matrix and reduce its dimensions, choose the number of components ( `n_components` ) using elbow method (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/pca-code-example-using-non-visualization/)

  - The shape of the matrix after TruncatedSVD will be 2000*n, i.e. each row represents a vector form of the corresponding word.
  - Vectorize the essay text and project titles using these word vectors. (while vectorizing, do ignore all the words which are not in top 2k words)

- step 4 Concatenate these truncatedSVD matrix, with the matrix with features
  - **school_state** : categorical data
  - **clean_categories** : categorical data
  - **clean_subcategories** : categorical data
  - **project_grade_category** :categorical data
  - **teacher_prefix** : categorical data
  - **quantity** : numerical data
  - **teacher_number_of_previously_posted_projects** : numerical data
  - **price** : numerical data
  - **sentiment score's of each of the essay** : numerical data
  - **number of words in the title** : numerical data
  - **number of words in the combine essays** : numerical data
  - **word vectors calculated in** step 3 : numerical data
- step 5: Apply GBDT on matrix that was formed in step 4 of this assignment, **DO REFER THIS BLOG: XGBOOST DMATRIX (https://www.kdnuggets.com/2017/03/simple-xgboost-tutorial-iris-dataset.html)**
- step 6:Hyper parameter tuning (Consider any two hyper parameters)
  - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
  - Find the best hyper paramter using k-fold cross validation or simple cross validation data
  - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

### 3.3 Vectorizing Essay and Title data : Creating custom Word Vectors

In [36]:
```
## To create Cooccurance matrix first select top 2k words from essay and title
def top_k_words(data,k):
    top_k=[]
    # concatenate words from essay and title
    corpus=pd.DataFrame(columns=['title_essay'])
    corpus['title_essay']=data['processed_essay'].map(str)  + \
                          data['processed_title'].map(str)
    # using tfidf model we are calculating idf for each word
    tfidf_model = TfidfVectorizer()
    tfidf_model.fit(corpus['title_essay'].values)
    # 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())
    # sort the dictionary based on idf values in descending order (i.e) selecting rare words
    sort_d=sorted(dictionary.items(),key=lambda x:(x[1],x[0]),reverse=True)
    #append top k words
    top_2k=[sort_d[i][0] for i in range(k)]
    return top_2k
```

In [37]:
```
top_2000_words=top_k_words(X_train,2000)
```

### Build Co-occurance matrix from scratch

In [38]:

```
# build co-occurance matrix
def get_indices(var,w):
    idx=[]
    for i in range(len(var)):
        if var[i] == w :
            idx.append(i)
    return idx
def get_co_matrix(win_size,entire_corpus,wrds):
    df=pd.DataFrame(columns=wrds,index=wrds)
    df=df.fillna(0)
    for i in tqdm(range(len(entire_corpus))):
        split_=entire_corpus[i].split()
        imp_words=[]
        for i in split_:
            ## store the words that are present in top 2000 words
            if i in wrds and i not in imp_words:imp_words.append(i)
        for wrd1 in imp_words :
            for wrd2 in imp_words :
                # find all the indices where word exists in the corpus
                indices=get_indices(split_,wrd1)
                for id_ in indices:
                    ## if the word exists in the start
                    if id_ == 0 :
                        if wrd2 in split_[id_+1:id_+win_size+1]:
                            df.at[wrd1,wrd2]+=1
                    ## if the word exists in  the end
                    elif id_ == len(split_)-1 :
                        if wrd2 in split_[id_-1:id_-win_size-1:-1] :
                            df.at[wrd1,wrd2]+=1
                    ## if the word exists in between
                    else:
                          ## if the word exists in the starting range less than the window size i.e
                          ## if window=5 and index of word is at 2 ,we cant slice 5 words before as it will show error
                        if id_ <=win_size :
                            if wrd2 in split_[id_-1::-1]:
                                    df.at[wrd1,wrd2]+=1
                            if wrd2 in split_[id_+1:id_+win_size+1]:
                                    df.at[wrd1,wrd2]+=1
                          ## if the word exists in the ending range less than the window size i.e
                          ## if window=5 and index of word is at 2 ,we cant slice 5 words after as it will show error
                        elif id_ >= len(split_)-win_size-1:
                            if wrd2 in split_[id_+1:]:
                                    df.at[wrd1,wrd2]+=1
                            if wrd2 in split_[id_-1:id_-win_size-1:-1]:
                                    df.at[wrd1,wrd2]+=1
                        # if words are anywhere in between
                        else :
                            if wrd2 in split_[id_+1:id_+win_size+1] :
                                    df.at[wrd1,wrd2]+=1
                            if wrd2 in split_[id_-1:id_-win_size-1:-1] :
                                    df.at[wrd1,wrd2]+=1
    return df
```

In [39]:
```
### output of trials
win_size=2
entire_corpus = ['ABC DEF IJK PQR','PQR KLM OPQ','LMN PQR XYZ ABC DEF PQR ABC']
wrds=['ABC','PQR','DEF']
trial_df=get_co_matrix(win_size,entire_corpus,wrds)
print(trial_df)
```

```
100%|██████████| 3/3 [00:00<00:00, 1002.86it/s]

     ABC  PQR  DEF
ABC    0    3    3
PQR    3    0    2
DEF    3    2    0
```

## Create co-occurance matrix with top 2k words

In [40]:
```
win_size=5
X_train['essay_title_merge'] = X_train['processed_title'].map(str) + X_train['processed_essay'].map(str)
entire_corpus = X_train['essay_title_merge'].values
wrds=top_2000_words
df=get_co_matrix(win_size,entire_corpus,wrds)
```

```
100%|██████████| 26800/26800 [01:41<00:00, 264.79it/s]
```

In [64]:
```
df.head(10)
```

Out[64]:

| | zynergi | zydeco | zy | zusak | zuma | zulu | zui | zuckerberg | zucchini | zot | ... | sithat | sistema | siskel | sisk | sis | sire | sipp | sinusoid | sinner | sinhales |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **zynergi** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **zydeco** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **zy** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **zusak** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **zuma** | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **zulu** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **zui** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **zuckerberg** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **zucchini** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **zot** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

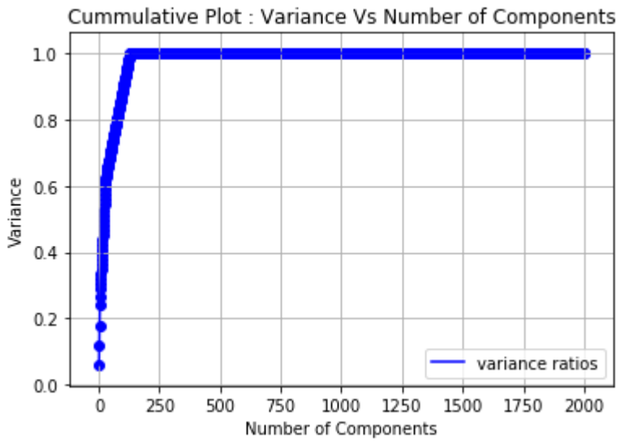
10 rows × 2000 columns

## Reducing dimensions of co-occurance matrix using TruncatedSVD

In [42]:
```
from sklearn.decomposition import TruncatedSVD
trunc_svd=TruncatedSVD(n_components=1999)
trunc_svd.fit(df)
```

Out[42]:
```
TruncatedSVD(algorithm='randomized', n_components=1999, n_iter=5,
       random_state=None, tol=0.0)
```

In [43]:
```
no_components=list(range(1,df.shape[1]))
#percentage of variance contributed by each feature
percent_variance=trunc_svd.explained_variance_ / np.sum(trunc_svd.explained_variance_)
#cummulative plot to determine number of features required to preserve maximum varaince
plt.plot(no_components,np.cumsum(percent_variance),color='b',label='variance ratios')
plt.scatter(no_components,np.cumsum(percent_variance),color='b')
plt.grid()
plt.legend()
plt.xlabel('Number of Components')
plt.ylabel('Variance')
plt.title('Cummulative Plot : Variance Vs Number of Components')
plt.show()
```




## Observations :

1. We see that 250 features are preserving 99% of variance

In [44]:
```
## Training with 250 features
trunc_svd=TruncatedSVD(n_components=250)
trunc_svd.fit(df)
co_occur_mat=trunc_svd.transform(df)
```

In [45]:
```
print('*'*20)
print('Shape of Co-occurance matrix after applying truncated SVD: ' ,co_occur_mat.shape)
```

```
********************
Shape of Co-occurance matrix after applying truncated SVD:  (2000, 250)
```

In [46]:
```
#vectorize(essay ,title) using final co ocuurance matrix
# 1. initiaize ur vector A 1 x 250 as zeros
# 2. word_Count=0
#    for every word in the essay
# 3. if word in co_ocur_matrix :
#     get the word vector from co_occur and add to A
#     word ocunt ++
# 4.divide by word count

idx_dict=dict(zip(top_2000_words,range(0,2001)))
def vectorize_text(data,co_mat,col_name):
    vectorized_text=[]
    for sentence in tqdm(data[col_name]):
        vector=np.zeros(250)
        word_count=0
        for word in sentence.split():
            if word in top_2000_words :
                vector+=co_mat[idx_dict[word]]
                word_count+=1
        if word_count!=0:
            vector = vector/word_count
        vectorized_text.append(vector)
    return vectorized_text
```

In [47]:
```
vectorize_tr_essay=vectorize_text(X_train,co_occur_mat,'processed_essay')
vectorize_te_essay=vectorize_text(X_test,co_occur_mat,'processed_essay')
vectorize_cv_essay=vectorize_text(X_cv,co_occur_mat,'processed_essay')
```

```
100%|████████████████████████████████████████████████████████████| 26800/26800 [01:51<00:00, 241.22it/s]
100%|████████████████████████████████████████████████████████████| 10000/10000 [00:41<00:00, 241.18it/s]
100%|████████████████████████████████████████████████████████████| 13200/13200 [00:54<00:00, 240.49it/s]
```

In [48]:
```
vectorize_tr_title=vectorize_text(X_train,co_occur_mat,'processed_title')
vectorize_te_title=vectorize_text(X_test,co_occur_mat,'processed_title')
vectorize_cv_title=vectorize_text(X_cv,co_occur_mat,'processed_title')
```

```
100%|████████████████████████████████████████████████████████████| 26800/26800 [00:03<00:00, 8722.78it/s]
100%|████████████████████████████████████████████████████████████| 10000/10000 [00:01<00:00, 8821.56it/s]
100%|████████████████████████████████████████████████████████████| 13200/13200 [00:01<00:00, 8058.04it/s]
```

In [49]:
```
### print shapes
print('*'*50)
print('Shape of vectorized essay train ',len(vectorize_tr_essay),len(vectorize_tr_essay[0]))
print('Shape of vectorized essay test ',len(vectorize_te_essay),len(vectorize_te_essay[0]))
print('Shape of vectorized essay cv ',len(vectorize_cv_essay),len(vectorize_cv_essay[0]))
```

```
**************************************************
Shape of vectorized essay train  26800 250
Shape of vectorized essay test  10000 250
Shape of vectorized essay cv  13200 250
```

## 4. Apply XGBoost after stacking all the features

In [50]:
```
X_tr = hstack((train_categories, train_subcategories,sklstate_train,teacher_prefix_train,
            proj_grade_train,vectorize_tr_title,vectorize_tr_essay,
               sentiment_neg_train_norm,sentiment_pos_train_norm,
            sentiment_neu_train_norm,sentiment_compound_train_norm,
            X_train_price_norm,quantity_train_norm,
            prev_projects_train_norm,title_word_count_train_norm,
            essay_word_count_train_norm)).tocsr()

X_te = hstack((test_categories, test_subcategories,sklstate_test,teacher_prefix_test,
            proj_grade_test,vectorize_te_title,vectorize_te_essay,
            sentiment_neg_test_norm,sentiment_pos_test_norm,
            sentiment_neu_test_norm,sentiment_compound_test_norm,
            X_test_price_norm,quantity_test_norm,
            prev_projects_test_norm,title_word_count_test_norm,
            essay_word_count_test_norm)).tocsr()

X_cr = hstack((cv_categories, cv_subcategories,sklstate_cv,teacher_prefix_cv,
            proj_grade_cv,vectorize_cv_title,vectorize_cv_essay
            ,sentiment_neg_cv_norm,sentiment_pos_cv_norm,
            sentiment_neu_cv_norm,sentiment_compound_cv_norm,
            X_cv_price_norm,quantity_cv_norm,
            prev_projects_cv_norm,title_word_count_cv_norm,
            essay_word_count_cv_norm)).tocsr()


print(X_tr.shape)
print(X_te.shape)
print(X_cr.shape)
```

```
(26800, 608)
(10000, 608)
(13200, 608)
```

## 4.1 Writing own loop to find best hyper parameter using simple cross validation data

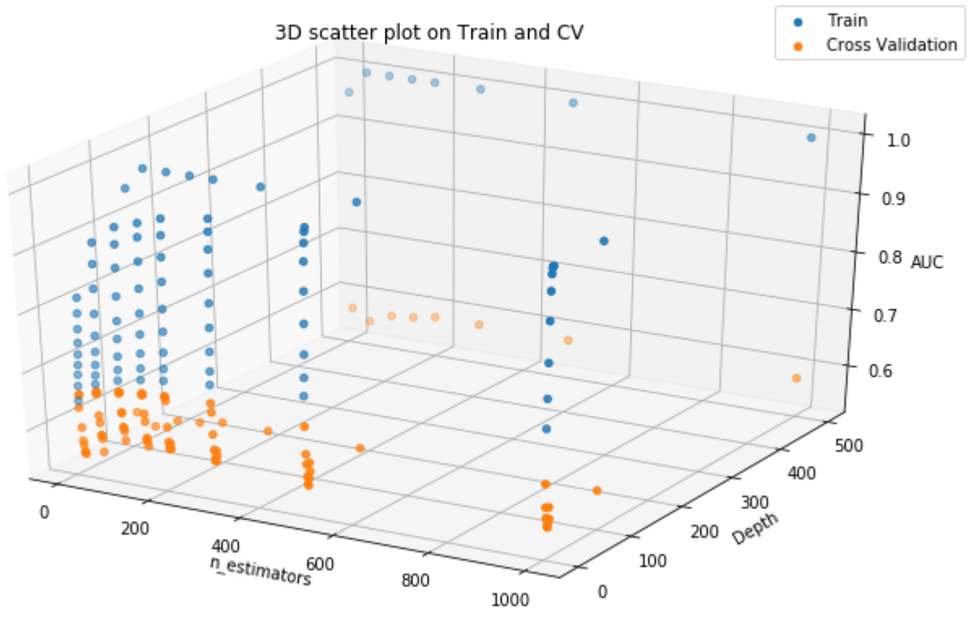
In [51]:
```
## Iterative looop to find best parameter
n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]
max_depth = [2, 3, 4, 5, 6, 7, 8, 9, 10,100,500]

def hyperparam_auc(max_depth,n_estimators,X_tr,X_cr,y_train,y_cv):
    train_auc=[]
    cv_auc=[]
    es,de=[],[]
    for estimators in tqdm(n_estimators):
        for depth in max_depth:
            xg=xgb.XGBClassifier(n_estimators=estimators,max_depth=depth,learning_rate=0.05,random_state=4)
            xg.fit(X_tr,y_train)
            #predict train and cv
            train_predicts=xg.predict_proba(X_tr)[:,1]
            cv_predicts=xg.predict_proba(X_cr)[:,1]
            #Store train and cv auc score in dict
            train_auc.append(roc_auc_score(y_train,train_predicts))
            cv_auc.append(roc_auc_score(y_cv,cv_predicts))
            es.append(estimators)
            de.append(depth)
    return train_auc,cv_auc,es,de

train_auc,cv_auc,es,de=hyperparam_auc(max_depth,n_estimators,X_tr,X_cr,y_train,y_cv)
```

```
100%|██████████| 8/8 [23:21<00:00, 175.14s/it]
```

## 4.2 Representation using 3D scatter plot

In [52]:
```
from mpl_toolkits.mplot3d import Axes3D
plt.figure(figsize=(12,7))
ax = plt.axes(projection='3d')
zipped_=list(map(list,zip(es,de)))
x1=[i[0] for i in zipped_]
y1=[i[1] for i in zipped_]
# Data for three-dimensional scattered points
# reference : https://jakevdp.github.io/PythonDataScienceHandbook/04.12-three-dimensional-plotting.html
ax.scatter3D(x1, y1, train_auc,label='Train')
ax.scatter3D(x1, y1, cv_auc,label='Cross Validation')
ax.set_xlabel('n_estimators')
ax.set_ylabel('Depth')
ax.set_zlabel('AUC')
ax.set_title('3D scatter plot on Train and CV')
ax.legend()
```

Out[52]: <matplotlib.legend.Legend at 0x257189d6518>




### 4.3 Finding best parameter for training the model

In [53]:
```
## finding the best pair of hyperparameter with max AUC using a function

def find_best_hyperparam(x1,y1,cv_auc):
    zipped_=list(zip(x1,y1,cv_auc))
    max_auc=max(cv_auc)
    print('Max_auc is',max_auc)
    for i in zipped_:
        if i[2] == max_auc:
            best_params={'n_estimators':i[0],'depth':i[1]}
        else:
            pass
    return best_params
```

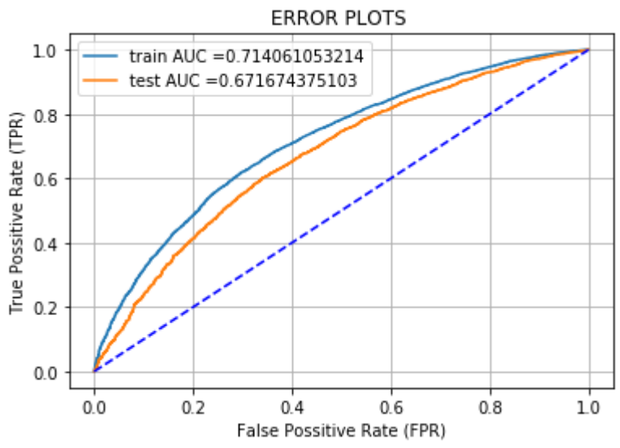
In [54]:
```
best_params=find_best_hyperparam(x1,y1,cv_auc)
best_params
```

```
Max_auc is 0.682584717305
```

Out[54]:
```
{'n_estimators': 300, 'depth': 2}
```

In [55]:
```
xg=xgb.XGBClassifier(learning_rate=0.05,n_estimators=best_params['n_estimators'],
                     max_depth=best_params['depth'],random_state=4)
xg.fit(X_tr,y_train)

## Predict the test
train_predicts=xg.predict_proba(X_tr)[:,1]
test_predicts=xg.predict_proba(X_te)[:,1]

## Store fpr and tpr rates

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, train_predicts)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, test_predicts)
```

In [56]:
```
#plot
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("False Possitive Rate (FPR)")
plt.ylabel("True Possitive Rate (TPR)")
plt.title("ERROR PLOTS ")
plt.plot([0, 1], [0, 1],'b--')
plt.grid()
plt.show()

```




## 4.4 Confusion Matrix on train and test data

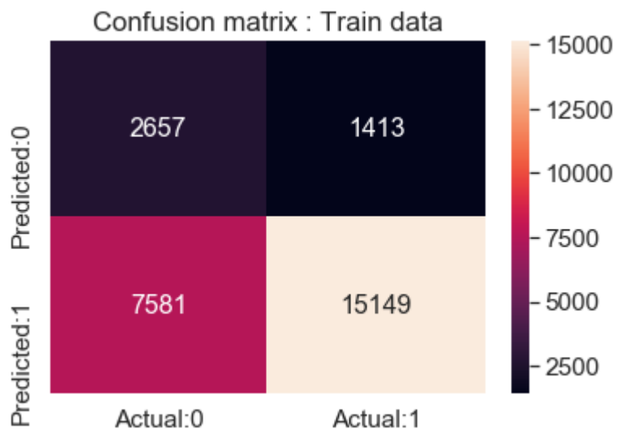
In [57]:
```
## Finding best threshold for predictions
def best_threshold(thresholds,fpr,tpr):
    t=thresholds[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

In [58]:
```
print("="*100)
best_t=best_threshold(tr_thresholds,train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(train_predicts, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(test_predicts, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.43509257808 for threshold 0.84
Train confusion matrix
[[ 2657  1413]
 [ 7581 15149]]
Test confusion matrix
[[ 689  830]
 [1841 6640]]
```

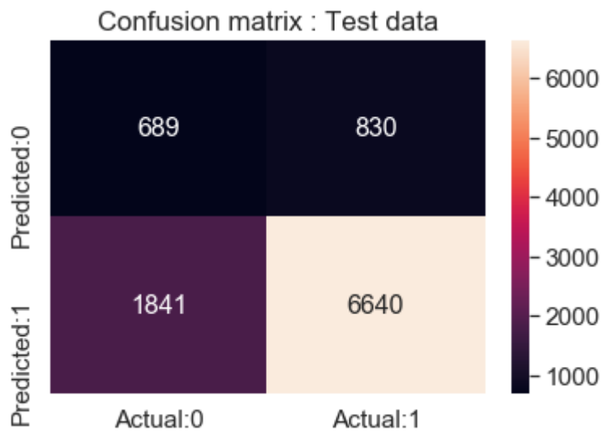
In [59]:

```
### PLOT the matrix for Train
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
# source : https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
df_cm = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(train_predicts, best_t))
                     , range(2), range(2))
# plt.figure(figsize=(10,7))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16},fmt='g',
            xticklabels=['Actual:0','Actual:1']
           ,yticklabels=['Predicted:0','Predicted:1']) # font size
plt.title('Confusion matrix : Train data')
plt.show()
```




In [60]:

```
### PLOT the matrix for Train
# source : https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
df_cm = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(test_predicts, best_t)), range(2), range(2))
# plt.figure(figsize=(10,7))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16},fmt='g',xticklabels=['Actual:0','Actual:1']
           ,yticklabels=['Predicted:0','Predicted:1']) # font size
plt.title('Confusion matrix : Test data')
plt.show()
```




## Observations :

1.We can observe from train and test we are getting majority True positives
2.Least number of data falls in False negative,which refers as least
number of projects were incorrectly predicted as not approved in both Test and Train.
3.For a model to perform well we need High True Positive Rate and
Low False Positive Rate.From the above our train data has True Positive Rate as
91.4% and False Positive Rate as 74%
4.In our test data : True Positive Rate as 88.8% and False Positive Rate as 72.7%.

## Conclusions :

1.Due to memory issue we sampled 50000 data and split into train,test and cv
2.Found top 2000 words with highest idf_ values and constructed co-occurance matrix.
3.Using TruncatedSVD we found optimal number of components that preserves maximum variance of co-occurance matrix
to be 250
4.After applying XGBoost we found optimal number of depth to be 2 and number of estimators to be 300.
5.After vectorizing text data using co-occurance matrix adds significant value to our model.
6.On the cv data we found max AUC to be 0.68.
7.After hyperparameter tuning and training model .Found train AUC to be 0.71 and test AUC to be 0.67

In [61]:
```
## https://xgboost.readthedocs.io/en/latest/python/python_intro.html
# reference : https://jakevdp.github.io/PythonDataScienceHandbook/04.12-three-dimensional-plotting.html
```

In [ ]:
```

```