Copy_of_Copy_of_7_DonorsChoose_SVM

March 21, 2019

1 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 p

How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly a
How to increase the consistency of project vetting across different volunteers to improve the experience for teache
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.

1.1 About the DonorsChoose Data Set

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

Feature	Description
project_id	A unique identifier for the proposed project. **Example:** p036502

project_title | Title of the project. **Examples:**
Art Will Make You Happy!
First Grade Fun
project_grade_category | Grade level of students for which the project is targeted. One of the following enumerated values:
Grades PreK-2
Grades 3-5
Grades 6-8
Grades 9-12
project_subject_categories | One or more (comma-separated) subject categories for the project from the following enumerated list of values:

Applied Learning
Care & Hunger
Health & Sports
History & Civics
Literacy & Language
Math & Science
Music & The Arts
Special Needs
Warmth
Examples:
Music & The Arts
Literacy & Language, Math & Science

school_state | State where school is located (Two-letter U.S. postal code). **Example:** WY

project_subject_subcategories | One or more (comma-separated) subject subcategories for the project. **Examples:**

Literacy
Literature & Writing, Social Sciences

project_resource_summary | An explanation of the resources needed for the project. **Example:**
My students need hands on literacy materials to manage sensory needs!

project_essay_1 | First application essay

project_essay_2 | *Second application essay* project_essay_3 | Third application essay

project_essay_4 | *Fourth application essay* project_submitted_datetime | Datetime when project application was submitted. **Example:** 2016-04-28 12:43:56.245

teacher_id | A unique identifier for the teacher of the proposed project. **Example:** bdf8baa8fedef6bfeec7ae4ff1c15c56

teacher_prefix | Teacher's title. One of the following enumerated values:

nan
Dr.
Mr.
Mrs.
Ms.
Teacher.

teacher_number_of_previously_posted_projects | Number of project applications previously submitted by the same teacher. **Example:** 2

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

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

Feature	Description
id	A project_id value from the train.csv file. **Example:** p036502
description	Desciption of the resource. **Example:** Tenor Saxophone Reeds, Box of 25

Feature	Description
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.

1.1.1 Notes on the Essay Data

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

project_essay_1: "Introduce us to your classroom"

project_essay_2: "Tell us more about your students"

project_essay_3: "Describe how your students will use the materials you're requesting"

project_essay_3: "Close by sharing why your project will make a difference"

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

project_essay_1: "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."

project_essay_2: "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
```

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

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

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

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

from tqdm import tqdm
import os

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

1.2 1.1 Reading Data

In [2]: #since im using google_colab, i have to mount the gdrive folder for accessing the files

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", fo

In [0]: #reading the datasets, i have taken only 5000 datapoints into consideration for avoiding mermory issues

```
project_data = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/Assignments_DonorsChoose_2
resource_data = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/Assignments_DonorsChoose_
```

```
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 [5]: # how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
        cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]


        #sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
        project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
        project_data.drop('project_submitted_datetime', axis=1, inplace=True)
        project_data.sort_values(by=['Date'], inplace=True)


        # how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
        project_data = project_data[cols]


        project_data.head(2)
```

Out[5]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_subject_categories	project_subject_subcategories	project_title	project_essay_1
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	Applied Learning	Early Development	Flexible Seating for Flexible Learning	I recently read an article about giving studen...
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	Literacy & Language	Literacy	Going Deep: The Art of Inner Thinking!	

```
41558  My students crave challenge, they eat obstacle...

                                        project_essay_2  \
473    I teach at a low-income (Title 1) school. Ever...
41558  We are an urban, public k-5 elementary school...

                                        project_essay_3  \
473    We need a classroom rug that we can use as a c...
41558  With the new common core standards that have b...

                                        project_essay_4  \
473    Benjamin Franklin once said, \"Tell me and I f...
41558  These remarkable gifts will provide students w...

                                 project_resource_summary  \
473    My students need flexible seating in the class...
41558  My students need copies of the New York Times ...

       teacher_number_of_previously_posted_projects  project_is_approved
473                                               2                   1
41558                                             2                   1
```

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

1.3 1.2 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 & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing "
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&
        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]))
```

1.4 1.3 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

        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 & Hunger"]
                if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math
                    j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing "
                j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&
                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]))
```

1.5 1.3 Text preprocessing

```
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 [10]: project_data.head(2)
```

```
Out[10]:
        Unnamed: 0       id                         teacher_id teacher_prefix  \
473         100660  p234804  cbc0e38f522143b86d372f8b43d4cff3           Mrs.
41558        33679  p137682  06f6e62e17de34fcf81020c77549e1d5           Mrs.

      school_state                 Date project_grade_category  \
473             GA 2016-04-27 00:53:00          Grades PreK-2
41558           WA 2016-04-27 01:05:25              Grades 3-5

                                project_title  \
473    Flexible Seating for Flexible Learning
41558  Going Deep: The Art of Inner Thinking!

                                   project_essay_1  \
473    I recently read an article about giving studen...
41558  My students crave challenge, they eat obstacle...

                                   project_essay_2  \
473    I teach at a low-income (Title 1) school. Ever...
41558  We are an urban, public k-5 elementary school...

                                   project_essay_3  \
473    We need a classroom rug that we can use as a c...
41558  With the new common core standards that have b...

                                   project_essay_4  \
473    Benjamin Franklin once said, \"Tell me and I f...
41558  These remarkable gifts will provide students w...

                               project_resource_summary  \
473    My students need flexible seating in the class...
41558  My students need copies of the New York Times ...

       teacher_number_of_previously_posted_projects  project_is_approved  \
473                                               2                    1
```

```
41558                                2                 1

       clean_categories clean_subcategories  \
473      AppliedLearning    EarlyDevelopment
41558  Literacy_Language            Literacy

                                                   essay
473    I recently read an article about giving studen...
41558  My students crave challenge, they eat obstacle...
```

In [0]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [12]:
```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
#print(project_data['essay'].values[99999])
#print("="*50)
```

```
I recently read an article about giving students a choice about how they learn. We already set goals; why not let t
==================================================
At the beginning of every class we start out with a Math Application problem to help students see the relevance o
==================================================
My students love coming to school and they love learning. I strive daily to make our classroom a relaxed, comfort
==================================================
I teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school provides free brea
==================================================
```

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 [14]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
I teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school provides free brea
==================================================
```

In [15]:
```
# \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 teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school provides free brea
```

In [16]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
I teach at a Title 1 school with 73 of my students who receive free reduced lunch Our school provides free breakfa
```

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', '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', "
            'won', "won't", 'wouldn', "wouldn't"]
```

In [18]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
```

```
    # tqdm is for printing the status bar
    for sentance in tqdm(project_data['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.append(sent.lower().strip())
```

100%|| 50000/50000 [00:31<00:00, 1566.54it/s]

In [19]:
```
# after preprocesing
preprocessed_essays[20000]
```

Out[19]: 'teach title 1 school 73 students receive free reduced lunch school provides free breakfast students special

1.4 Preprocessing of project_title

In [20]:
```
# similarly you can preprocess the titles also

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

100%|| 50000/50000 [00:01<00:00, 32546.04it/s]

1.6 1.5 Preparing data for models

In [21]:
```
project_data.columns
```

Out[21]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', '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',
       'clean_categories', 'clean_subcategories', 'essay'],
      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

1.6.1 Modifying DataSet (essay & project_title)

```
In [0]: project_data['clean_essay'] = preprocessed_essays
        project_data['clean_project_title'] = preprocessed_project_title
        project_data.drop(['essay'], axis=1, inplace=True)
        project_data.drop(['project_title'], axis=1, inplace=True)
```

```
In [23]: project_data.head(1)
```

Out[23]:

	Unnamed: 0	id	teacher_id	teacher_prefix	\
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	

	school_state	Date	project_grade_category	\
473	GA	2016-04-27 00:53:00	Grades PreK-2	

	project_essay_1	\
473	I recently read an article about giving studen...	

	project_essay_2	\
473	I teach at a low-income (Title 1) school. Ever...	

	project_essay_3	\
473	We need a classroom rug that we can use as a c...	

	project_essay_4	\
473	Benjamin Franklin once said, \"Tell me and I f...	

	project_resource_summary	\
473	My students need flexible seating in the class...	

	teacher_number_of_previously_posted_projects	project_is_approved	\
473	2	1	

```
     clean_categories clean_subcategories  \
473  AppliedLearning     EarlyDevelopment

                                        clean_essay  \
473  recently read article giving students choice l...

                    clean_project_title
473  flexible seating flexible learning
```

In [24]:
```
y = project_data['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
project_data.head(1)
```

Out[24]:
```
     Unnamed: 0       id                         teacher_id teacher_prefix  \
473      100660  p234804  cbc0e38f522143b86d372f8b43d4cff3           Mrs.

    school_state                Date project_grade_category  \
473           GA 2016-04-27 00:53:00          Grades PreK-2

                                     project_essay_1  \
473  I recently read an article about giving studen...

                                     project_essay_2  \
473  I teach at a low-income (Title 1) school. Ever...

                                     project_essay_3  \
473  We need a classroom rug that we can use as a c...

                                     project_essay_4  \
473  Benjamin Franklin once said, \"Tell me and I f...

                           project_resource_summary  \
473  My students need flexible seating in the class...

     teacher_number_of_previously_posted_projects clean_categories  \
473                                             2  AppliedLearning

    clean_subcategories                                       clean_essay  \
473    EarlyDevelopment  recently read article giving students choice l...

                    clean_project_title
473  flexible seating flexible learning
```

In [0]:
```
X = project_data
```

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

In [0]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
```

```
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label


# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

In [27]:
```
X_train.head(2)
```

Out[27]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_essay_1	project_essay_2	project_essay_3	project_essay_4	project_resource_summary	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	clean_essay
3953	152779	p187939	f079e996501bdd78cff2dac333a50604	Ms.	PA	2016-11-27 17:36:22	Grades 3-5	This year I moved into a new position teaching...	Each day we meet as a class on the carpet/rug ...	NaN	NaN	My students need a new rug. Each morning we g...	2	Literacy_Language SpecialNeeds	ESL SpecialNeeds	year moved new position teaching supplemental ...
3149	133005	p129723	1a80811e4236ac144ea3227e1f48d748	Mrs.	IN	2016-11-08 18:58:10	Grades 3-5	My 4th grade class is comprised of 27 students...	A class set of stability balls will ensure tha...	NaN	NaN	My students need ways to help them to focus in...	1	Literacy_Language Math_Science	Literacy Mathematics	4th grade class comprised 27 students differen...

```
                    clean_project_title
3953  around world our classroom new rug
3149                       mission move
```

In [28]:
```
resource_data.head(1)
```

Out[28]:
```
        id                                   description  quantity  price
0  p233245  LC652 - Lakeshore Double-Space Mobile Drying Rack         1  149.0
```

2.2 Make Data Model Ready: encoding numerical, categorical features

1.6.2 clean_categories

In [29]:
```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_cc_ohe = vectorizer.transform(X_train['clean_categories'].values)
#X_cv_cc_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_cc_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_cc_ohe.shape, y_train.shape)
#print(X_cv_cc_ohe.shape, y_cv.shape)
print(X_test_cc_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 9) (33500,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_a
====================================================================================================
```

1.6.3 clean_subcategories

In [30]:
```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_csc_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
#X_cv_csc_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_csc_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_csc_ohe.shape, y_train.shape)
#print(X_cv_csc_ohe.shape, y_cv.shape)
print(X_test_csc_ohe.shape, y_test.shape)
```

```
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 30) (33500,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityserv
====================================================================================================
```

1.6.4 school_state

```
In [31]: vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
#X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
#print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 51) (33500,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'm
====================================================================================================
```

1.6.5 teacher_prefix

```
In [32]: vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values.astype('U')) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values.astype('U'))
#X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values.astype('U'))
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values.astype('U'))

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
#print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
```

```
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 6) (33500,)
(16500, 6) (16500,)
['dr', 'mr', 'mrs', 'ms', 'nan', 'teacher']
====================================================================================================
```

1.6.6 project_grade_category

```
In [33]: vectorizer = CountVectorizer()
         vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
         #X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
         X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

         print("After vectorizations")
         print(X_train_grade_ohe.shape, y_train.shape)
         #print(X_cv_grade_ohe.shape, y_cv.shape)
         print(X_test_grade_ohe.shape, y_test.shape)
         print(vectorizer.get_feature_names())
         print("="*100)
```

```
After vectorizations
(33500, 3) (33500,)
(16500, 3) (16500,)
['12', 'grades', 'prek']
====================================================================================================
```

2.3 Make Data Model Ready: encoding eassay, and project_title

1.5.2.1 Bag of words

1.6.7 essays

```
In [34]: from sklearn.feature_extraction.text import CountVectorizer
         vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4))
         vectorizer.fit(X_train['clean_essay'].values) # fit has to happen only on train data

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_essay_bow = vectorizer.transform(X_train['clean_essay'].values)
         #X_cv_essay_bow = vectorizer.transform(X_cv['clean_essay'].values)
         X_test_essay_bow = vectorizer.transform(X_test['clean_essay'].values)
```

```
print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
#print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 98890) (33500,)
(16500, 98890) (16500,)
====================================================================================================
```

1.6.8 project_title

```
In [35]: from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(X_train['clean_project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_pt_bow = vectorizer.transform(X_train['clean_project_title'].values)
#X_cv_pt_bow = vectorizer.transform(X_cv['clean_project_title'].values)
X_test_pt_bow = vectorizer.transform(X_test['clean_project_title'].values)

print("After vectorizations")
print(X_train_pt_bow.shape, y_train.shape)
#print(X_cv_pt_bow.shape, y_cv.shape)
print(X_test_pt_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1636) (33500,)
(16500, 1636) (16500,)
====================================================================================================
```

1.5.2.2 TFIDF vectorizer

1.6.9 essays

```
In [36]: from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
X_train_essay_tfidf = vectorizer.fit_transform(X_train['clean_essay'].values)
#X_cv_essay_tfidf = vectorizer.transform(X_cv['clean_essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['clean_essay'].values)
```

```
print("Shape of matrix after one hot encodig ",X_train_essay_tfidf.shape)
#print("Shape of matrix after one hot encodig ",X_cv_essay_tfidf.shape)
print("Shape of matrix after one hot encodig ",X_test_essay_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (33500, 5000)
Shape of matrix after one hot encodig  (16500, 5000)
```

1.6.10 project_title

```
In [37]: from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
X_train_pt_tfidf = vectorizer.fit_transform(X_train['clean_project_title'].values)
#X_cv_pt_tfidf = vectorizer.transform(X_cv['clean_project_title'].values)
X_test_pt_tfidf = vectorizer.transform(X_test['clean_project_title'].values)
```

```
print("Shape of matrix after one hot encodig ",X_train_pt_tfidf.shape)
#print("Shape of matrix after one hot encodig ",X_cv_pt_tfidf.shape)
print("Shape of matrix after one hot encodig ",X_test_pt_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (33500, 1636)
Shape of matrix after one hot encodig  (16500, 1636)
```

1.5.2.3 Using Pretrained Models: Avg W2V

1.6.11 essays

Train

```
In [0]: i=0
list_of_sentanceTrain=[]
for sentance in X_train['clean_essay']:
    list_of_sentanceTrain.append(sentance.split())
```

```
In [39]: is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentanceTrain,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    #print(w2v_model.wv.most_similar('worst'))
```

```
    elif want_to_use_google_w2v and is_your_ram_gt_16g:
        if os.path.isfile('GoogleNews-vectors-negative300.bin'):
            w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=T
            #print(w2v_model.wv.most_similar('great'))
            #print(w2v_model.wv.most_similar('worst'))
        else:
            print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w
```

```
[('amazing', 0.7186568975448608), ('wonderful', 0.7023840546607971), ('incredible', 0.6747049689292908), ('excelle
==========================================================
```

```
In [40]: w2v_words = list(w2v_model.wv.vocab)
         print("number of words that occured minimum 5 times ",len(w2v_words))
         print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  14650
sample words  ['year', 'moved', 'new', 'position', 'teaching', 'supplemental', 'learning', 'support', 'classroom', '3rd'
```

```
In [41]: sent_vectorsPPE_train = []; # the avg-w2v for each sentence/review is stored in this list
         for sent in tqdm(list_of_sentanceTrain): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300
             cnt_words =0; # num of words with a valid vector in the sentence/review
             for word in sent: # for each word in a review/sentence
                 if word in w2v_words:
                     vec = w2v_model.wv[word]
                     sent_vec += vec
                     cnt_words += 1
             if cnt_words != 0:
                 sent_vec /= cnt_words
             sent_vectorsPPE_train.append(sent_vec)
         print(len(sent_vectorsPPE_train))
         print(len(sent_vectorsPPE_train[0]))
```

```
100%|| 33500/33500 [02:44<00:00, 203.67it/s]
```

```
33500
50
```

Test

```
In [0]: i=0
        list_of_sentanceTest=[]
        for sentance in X_test['clean_essay']:
            list_of_sentanceTest.append(sentance.split())
```

In [43]:
```
is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentanceTest,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=T
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w
```

```
[('awesome', 0.6969257593154907), ('wonderful', 0.6902042031288147), ('amazing', 0.6873944401741028), ('excellen
==================================================
[('camden', 0.8393195271492004), ('perks', 0.8388530611991882), ('prestigious', 0.8364563584327698), ('nj', 0.8321
```

In [44]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  11088
sample words  ['privilege', 'teaching', '20', 'pre', 'k', 'students', 'general', 'special', 'education', 'setting', 'classroon
```

In [45]:
```
sent_vectorsPPE_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentanceTest): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectorsPPE_test.append(sent_vec)
print(len(sent_vectorsPPE_test))
print(len(sent_vectorsPPE_test[0]))
```

```
100%|| 16500/16500 [01:13<00:00, 223.39it/s]
```

16500
50

1.6.12 project_title

Train

In [0]: # Similarly you can vectorize for title also

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentancePTtrain=[]
for sentance in X_train['clean_project_title']:
    list_of_sentancePTtrain.append(sentance.split())
```

In [47]:
```
is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

try :


  if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentancePTtrain,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

  elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=T
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w

except KeyError :

  pass

finally :

  print("Execution Done")
```

```
[('chapter', 0.9933265447616577), ('clubs', 0.9913930296897888), ('comic', 0.9912685751914978), ('nook', 0.991213
==================================================
Execution Done
```

```
In [48]: w2v_words = list(w2v_model.wv.vocab)
         print("number of words that occured minimum 5 times ",len(w2v_words))
         print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  2703
sample words  ['around', 'world', 'our', 'classroom', 'new', 'rug', 'mission', 'move', 'inspired', 'for', 'success', 'act',
```

```
In [49]: sent_vectorsPT_train = []; # the avg-w2v for each sentence/review is stored in this list
         for sent in tqdm(list_of_sentancePTtrain): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300
             cnt_words =0; # num of words with a valid vector in the sentence/review
             for word in sent: # for each word in a review/sentence
                 if word in w2v_words:
                     vec = w2v_model.wv[word]
                     sent_vec += vec
                     cnt_words += 1
             if cnt_words != 0:
                 sent_vec /= cnt_words
             sent_vectorsPT_train.append(sent_vec)
         print(len(sent_vectorsPT_train))
         print(len(sent_vectorsPT_train[0]))
```

```
100%|| 33500/33500 [00:03<00:00, 10650.03it/s]

33500
50
```

Test

```
In [0]: i=0
        list_of_sentancePT_test=[]
        for sentance in X_test['clean_project_title']:
            list_of_sentancePT_test.append(sentance.split())
```

```
In [51]: # Using Google News Word2Vectors
```

```
# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentancePT_test,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    #print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=T
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your own w
```

```
[('music', 0.9995664358139038), ('becoming', 0.9995652437210083), ('arts', 0.9995505809783936), ('literature', 0.99
==================================================
```

In [52]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  1786
sample words  ['supplies', 'get', 'ready', 'success', 'sports', 'prek', 'we', 'love', 'read', 'strengthen', 'core', 'little', 'm
```

In [53]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectorsPT_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentancePT_test): # for each review/sentence
```

```
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectorsPT_test.append(sent_vec)
print(len(sent_vectorsPT_test))
print(len(sent_vectorsPT_test[0]))
```

```
100%|| 16500/16500 [00:01<00:00, 12185.06it/s]

16500
50
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

1.6.13 essays

Train

```
In [0]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        model = TfidfVectorizer()
        model.fit(X_train['clean_essay'])
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [55]: # TF-IDF weighted Word2Vec
         tfidf_feat = model.get_feature_names() # tfidf words/col-names
         # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

         tfidf_sent_vectors_essay_train = []; # the tfidf-w2v for each sentence/review is stored in this list
         row=0;
         for sent in tqdm(list_of_sentanceTrain): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length
             weight_sum =0; # num of words with a valid vector in the sentence/review
             for word in sent: # for each word in a review/sentence
                 if word in w2v_words and word in tfidf_feat:
                     vec = w2v_model.wv[word]
         #            tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                     # to reduce the computation we are
                     # dictionary[word] = idf value of word in whole courpus
```

```
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors_essay_train.append(sent_vec)
        row += 1
```

100%|| 33500/33500 [27:48<00:00, 19.39it/s]

Test

```
In [0]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        model = TfidfVectorizer()
        model.fit(X_test['clean_essay'])
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
In [57]: # TF-IDF weighted Word2Vec
         tfidf_feat = model.get_feature_names() # tfidf words/col-names
         # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

         tfidf_sent_vectors_essay_test = []; # the tfidf-w2v for each sentence/review is stored in this list
         row=0;
         for sent in tqdm(list_of_sentanceTest): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length
             weight_sum =0; # num of words with a valid vector in the sentence/review
             for word in sent: # for each word in a review/sentence
                 if word in w2v_words and word in tfidf_feat:
                     vec = w2v_model.wv[word]
         #            tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                     # to reduce the computation we are
                     # dictionary[word] = idf value of word in whole courpus
                     # sent.count(word) = tf valeus of word in this review
                     tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                     sent_vec += (vec * tf_idf)
                     weight_sum += tf_idf
             if weight_sum != 0:
                 sent_vec /= weight_sum
             tfidf_sent_vectors_essay_test.append(sent_vec)
             row += 1
```

100%|| 16500/16500 [10:53<00:00, 25.26it/s]

1.6.14 project_title

Train

```
In [0]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        model = TfidfVectorizer()
        model.fit(X_train['clean_project_title'])
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

In [59]: # TF-IDF weighted Word2Vec
         tfidf_feat = model.get_feature_names() # tfidf words/col-names
         # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

         tfidf_sent_vectorsPT_train = []; # the tfidf-w2v for each sentence/review is stored in this list
         row=0;
         for sent in tqdm(list_of_sentancePTtrain): # for each review/sentence
             sent_vec = np.zeros(50) # as word vectors are of zero length
             weight_sum =0; # num of words with a valid vector in the sentence/review
             for word in sent: # for each word in a review/sentence
                 if word in w2v_words and word in tfidf_feat:
                     vec = w2v_model.wv[word]
         #            tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                     # to reduce the computation we are
                     # dictionary[word] = idf value of word in whole courpus
                     # sent.count(word) = tf valeus of word in this review
                     tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                     sent_vec += (vec * tf_idf)
                     weight_sum += tf_idf
             if weight_sum != 0:
                 sent_vec /= weight_sum
             tfidf_sent_vectorsPT_train.append(sent_vec)
             row += 1

100%|| 33500/33500 [00:21<00:00, 1532.53it/s]
```

Test

```
In [0]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
        model = TfidfVectorizer()
        model.fit(X_test['clean_project_title'])
        # we are converting a dictionary with word as a key, and the idf as a value
        dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

In [61]: # TF-IDF weighted Word2Vec
         tfidf_feat = model.get_feature_names() # tfidf words/col-names
         # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

         tfidf_sent_vectorsPT_test = []; # the tfidf-w2v for each sentence/review is stored in this list
         row=0;
         for sent in tqdm(list_of_sentancePT_test): # for each review/sentence
```

```
        sent_vec = np.zeros(50) # as word vectors are of zero length
        weight_sum =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_words and word in tfidf_feat:
                vec = w2v_model.wv[word]
    #             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
                # to reduce the computation we are
                # dictionary[word] = idf value of word in whole courpus
                # sent.count(word) = tf valeus of word in this review
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectorsPT_test.append(sent_vec)
        row += 1
```

```
100%|| 16500/16500 [00:08<00:00, 1888.69it/s]
```

1.6.15 1.5.3 Vectorizing Numerical features

1.6.16 price

```
In [0]: price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
        X_train = pd.merge(X_train, price_data, on='id', how='left')
        #X_cv = pd.merge(X_cv, price_data, on='id', how='left')
        X_test = pd.merge(X_test, price_data, on='id', how='left')
```

```
In [63]: X_train.head(1)
```

```
Out[63]:   Unnamed: 0       id                         teacher_id teacher_prefix  \
        0      152779  p187939  f079e996501bdd78cff2dac333a50604            Ms.

          school_state                 Date project_grade_category  \
        0           PA 2016-11-27 17:36:22              Grades 3-5

                                     project_essay_1  \
        0  This year I moved into a new position teaching...

                                     project_essay_2 project_essay_3  \
        0  Each day we meet as a class on the carpet/rug ...          NaN

          project_essay_4                       project_resource_summary  \
        0             NaN  My students need a new rug.  Each morning we g...

           teacher_number_of_previously_posted_projects  \
        0                                             2
```

```
                clean_categories clean_subcategories  \
0  Literacy_Language SpecialNeeds     ESL SpecialNeeds

                                        clean_essay  \
0  year moved new position teaching supplemental ...

                clean_project_title   price  quantity
0  around world our classroom new rug  195.97         1
```

In [64]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].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(X_train['price'].values.reshape(-1,1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
#print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

1.6.17 tnppp

In [65]:
```
from sklearn.preprocessing import Normalizer
normalizerT = Normalizer()
# normalizer.fit(X_train['price'].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.
normalizerT.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_tnppp_norm = normalizerT.transform(X_train['teacher_number_of_previously_posted_proje
```

```
#X_cv_tnppp_norm = normalizerT.transform(X_cv['teacher_number_of_previously_posted_projects'
X_test_tnppp_norm = normalizerT.transform(X_test['teacher_number_of_previously_posted_projects

print("After vectorizations")
print(X_train_tnppp_norm.shape, y_train.shape)
#print(X_cv_tnppp_norm.shape, y_cv.shape)
print(X_test_tnppp_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

1.6.18 1.5.4 Merging all the above features

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

SET1

In [66]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr1 = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_ohe, X_train_price_norm, X_tr
#X_cr1 = hstack((X_cv_cc_ohe, X_cv_csc_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_tnppp
X_te1 = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe, X_test_price_norm, X_test_t

print("Final Data matrix")
print(X_tr1.shape, y_train.shape)
#print(X_cr1.shape, y_cv.shape)
print(X_te1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 100570) (33500,)
(16500, 100570) (16500,)
====================================================================================================
```

SET2

In [67]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr2 = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_ohe, X_train_price_norm, X_tr
#X_cr2 = hstack((X_cv_cc_ohe, X_cv_csc_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_tnppp
X_te2 = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe, X_test_price_norm, X_test_t

print("Final Data matrix")
print(X_tr2.shape, y_train.shape)
```

```
#print(X_cr2.shape, y_cv.shape)
print(X_te2.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 6680) (33500,)
(16500, 6680) (16500,)
====================================================================================================
```

SET3

In [68]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr3 = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_ohe, X_train_price_norm, X_tr
#X_cr3 = hstack((X_cv_cc_ohe, X_cv_csc_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_tnppp
X_te3 = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe, X_test_price_norm, X_test_t

print("Final Data matrix")
print(X_tr3.shape, y_train.shape)
#print(X_cr3.shape, y_cv.shape)
print(X_te3.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 144) (33500,)
(16500, 144) (16500,)
====================================================================================================
```

SET4

In [69]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr4 = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_ohe, X_train_price_norm, X_tr
#X_cr4 = hstack((X_cv_cc_ohe, X_cv_csc_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_tnppp
X_te4 = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe, X_test_price_norm, X_test_t

print("Final Data matrix")
print(X_tr4.shape, y_train.shape)
#print(X_cr4.shape, y_cv.shape)
print(X_te4.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 144) (33500,)
(16500, 144) (16500,)
====================================================================================================
```

1.6.19 Computing Sentiment Scores

```
In [0]: essay_List_tr = X_train['clean_essay']
        essay_List_te = X_test['clean_essay']
```

Train

```
In [71]: import nltk
         from nltk.sentiment.vader import SentimentIntensityAnalyzer
         nltk.download('vader_lexicon')


         sia = SentimentIntensityAnalyzer()

         ss_tr = []
         ss_neg_tr = []
         ss_neu_tr = []
         ss_pos_tr = []
         ss_com_tr = []

         for i in range(len(essay_List_tr)) :
           ss_tr.append(sia.polarity_scores(essay_List_tr[i]))
           ss_neg_tr.append(ss_tr[i]['neg'])
           ss_neu_tr.append(ss_tr[i]['neu'])
           ss_pos_tr.append(ss_tr[i]['pos'])
           ss_com_tr.append(ss_tr[i]['compound'])
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
In [0]: X_train['negative_sentiment_score'] = ss_neg_tr
        X_train['neutral_sentiment_score'] = ss_neu_tr
        X_train['positive_sentiment_score'] = ss_pos_tr
        X_train['compound_sentiment_score'] = ss_com_tr
```

Test

```
In [0]: import nltk
        from nltk.sentiment.vader import SentimentIntensityAnalyzer

        sia = SentimentIntensityAnalyzer()

        ss_te = []
        ss_neg_te = []
        ss_neu_te = []
        ss_pos_te = []
        ss_com_te = []
```

```
    for i in range(len(essay_List_te)) :
      ss_te.append(sia.polarity_scores(essay_List_te[i]))
      ss_neg_te.append(ss_te[i]['neg'])
      ss_neu_te.append(ss_te[i]['neu'])
      ss_pos_te.append(ss_te[i]['pos'])
      ss_com_te.append(ss_te[i]['compound'])
```

```
In [0]: X_test['negative_sentiment_score'] = ss_neg_te
        X_test['neutral_sentiment_score'] = ss_neu_te
        X_test['positive_sentiment_score'] = ss_pos_te
        X_test['compound_sentiment_score'] = ss_com_te
```

1.6.20 Computing no of words in essay

```
In [0]: def num_Words(s) :
          return len(s.split())
```

Train

```
In [0]: num_wo_tr = []
        for i in X_train['clean_essay'] :
          num_wo_tr.append(num_Words(i))
```

```
In [0]: X_train['num_words_essay'] = num_wo_tr
```

Test

```
In [0]: num_wo_te = []
        for i in X_test['clean_essay'] :
          num_wo_te.append(num_Words(i))
```

```
In [0]: X_test['num_words_essay'] = num_wo_te
```

1.6.21 Computing number of words in project_title

Train

```
In [0]: num_wopt_tr = []
        for i in X_train['clean_project_title'] :
          num_wopt_tr.append(num_Words(i))
```

```
In [0]: X_train['num_words_project_title'] = num_wopt_tr
```

Test

```
In [0]: num_wopt_te = []
        for i in X_test['clean_project_title'] :
          num_wopt_te.append(num_Words(i))
```

In [0]: X_test['num_words_project_title'] = num_wopt_te

In [84]: X_test.head(1)

Out[84]:
```
   Unnamed: 0       id                         teacher_id teacher_prefix  \
0      138774  p254722  89ff808505388fa0d302a14443957d25            Ms.

  school_state             Date project_grade_category  \
0           NY 2016-07-26 17:43:28          Grades PreK-2

                                  project_essay_1  \
0  I have the privilege of teaching 20 Pre-K stud...

                                  project_essay_2 project_essay_3  \
0  Our school's mission is to \"To empower, prepa...            NaN

           ...                                          clean_essay  \
0          ...            privilege teaching 20 pre k students general s...

         clean_project_title   price quantity negative_sentiment_score  \
0  supplies get ready success  335.32       35                   0.019

  neutral_sentiment_score positive_sentiment_score  compound_sentiment_score  \
0                   0.711                    0.27                    0.9928

   num_words_essay  num_words_project_title
0              166                       4

[1 rows x 25 columns]
```

2 Assignment 7: SVM

[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) on these feature

Set 1: categorical, numerical features + project_title(BOW) + preprocessed_e
Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_
Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocess
Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preproce

The hyper paramter tuning (best alpha in range [10^-
4 to 10^4], and the best penalty among 'l1', 'l2')

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

```
    </ul>
        </li>
<br>
<li><strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for each hyper parameter
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test d
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.com/course/appli
ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/'>confusion matrix</a> with predicted and original lab
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as su
    <ul>
<li> Consider these set of features <font color='red'> Set 5 :</font>
        <ul>
            <li><strong>school_state</strong> : categorical data</li>
            <li><strong>clean_categories</strong> : categorical data</li>
            <li><strong>clean_subcategories</strong> : categorical data</li>
            <li><strong>project_grade_category</strong> :categorical data</li>
            <li><strong>teacher_prefix</strong> : categorical data</li>
            <li><strong>quantity</strong> : numerical data</li>
        <li><strong>teacher_number_of_previously_posted_projects</strong> : numerical data</li>
            <li><strong>price</strong> : numerical data</li>
            <li><strong>sentiment score's of each of the essay</strong> : numerical data</li>
            <li><strong>number of words in the title</strong> : numerical data</li>
            <li><strong>number of words in the combine essays</strong> : numerical data</li>
                                                    <li><strong>Apply      <a      href='http://scikit-
learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html'>TruncatedSVD</a> on <a href
learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html'>TfidfVectorizer</a> o
ai-course-online/lessons/pca-code-example-using-non-visualization/'>elbow method</a></strong> : numerical da
        </ul>
     <br>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a ta
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

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. Support Vector Machines

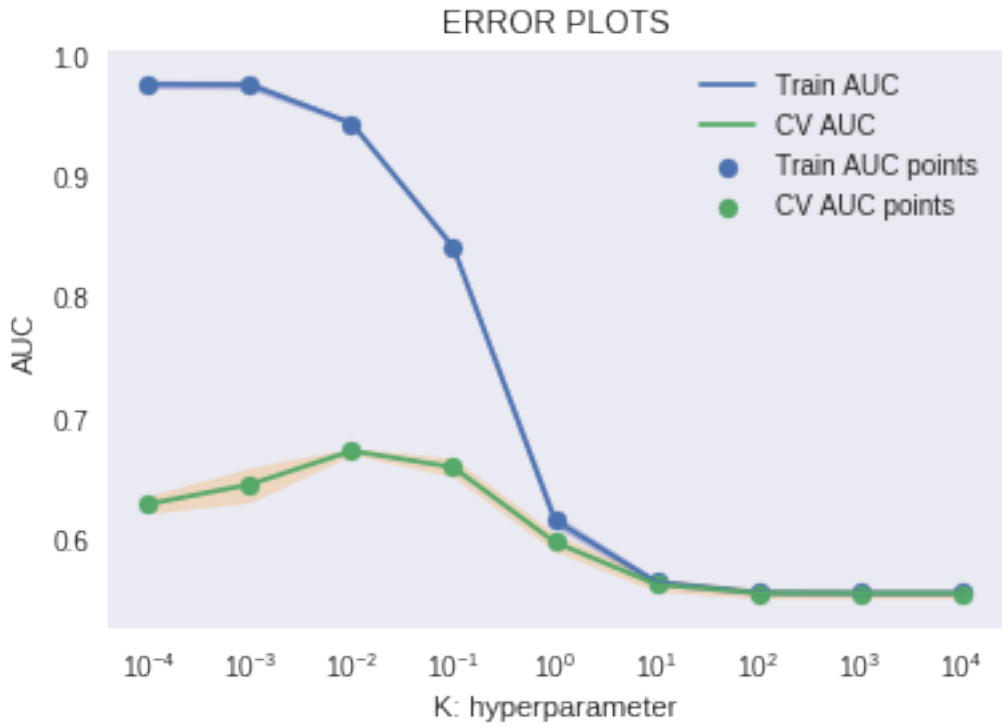
2.4 Applying Logistic Regression on different kind of featurization as mentioned in the instructions

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

2.0.1 SET1

In [85]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier

sgc1 = SGDClassifier()

parameters = {'alpha': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
#parameters = {'alpha':[0.0001, 0.0025, 0.0005, 0.0075, 0.001, 0.025, .005, 0.075, 0.1, 0.25, 0.5, 0.75, 1]}

clf = GridSearchCV(sgc1, parameters, cv=3, scoring='roc_auc')
clr1 = clf.fit(X_tr1, y_train)

train_auc1= clf.cv_results_['mean_train_score']
train_auc_std1= clf.cv_results_['std_train_score']
cv_auc1 = clf.cv_results_['mean_test_score']
cv_auc_std1 = clf.cv_results_['std_test_score']

plt.plot(parameters['alpha'], train_auc1, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc1 - train_auc_std1,train_auc1 + train_auc_std1,alpl

plt.plot(parameters['alpha'], cv_auc1, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc1 - cv_auc_std1, cv_auc1 + cv_auc_std1,alpha=0.2,col

plt.scatter(parameters['alpha'], train_auc1, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc1, label='CV AUC points')

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

```
plt.show()

print(train_auc1)
print(cv_auc1)      #C=10**-2
```

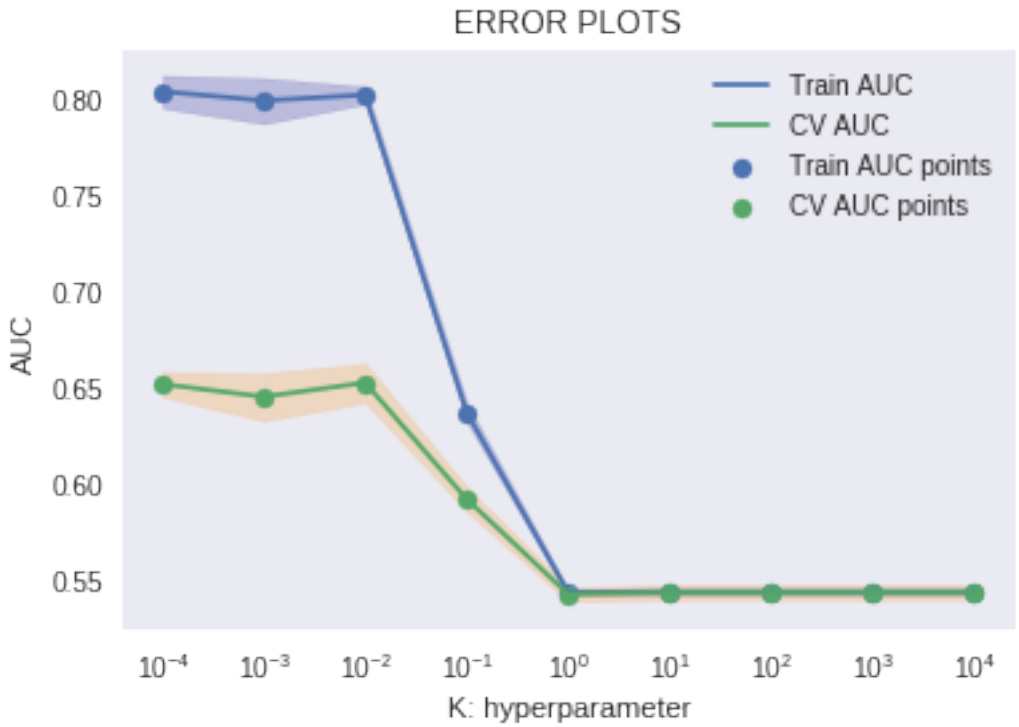



```
[0.97598135 0.97558355 0.94367137 0.84119173 0.61538446 0.56416933
 0.55560259 0.55534541 0.55534559]
[0.62869318 0.64487918 0.67263469 0.65916132 0.59716627 0.56227663
 0.5551812  0.55495923 0.55495897]
```

2.0.2 SET2

In [86]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier

sgc2 = SGDClassifier()

parameters = {'alpha': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
#parameters = {'alpha':[0.0001, 0.0025, 0.0005, 0.0075, 0.001, 0.025, .005, 0.075, 0.1, 0.25, 0.5, 0.75, 1]}

clf = GridSearchCV(sgc2, parameters, cv=3, scoring='roc_auc')
clr2 = clf.fit(X_tr2, y_train)
```

```
train_auc2= clf.cv_results_['mean_train_score']
train_auc_std2= clf.cv_results_['std_train_score']
cv_auc2 = clf.cv_results_['mean_test_score']
cv_auc_std2 = clf.cv_results_['std_test_score']

plt.plot(parameters['alpha'], train_auc2, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc2 - train_auc_std2,train_auc2 + train_auc_std2,alph

plt.plot(parameters['alpha'], cv_auc2, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc2 - cv_auc_std2, cv_auc2 + cv_auc_std2,alpha=0.2,col

plt.scatter(parameters['alpha'], train_auc2, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc2, label='CV AUC points')

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

print(train_auc2)
print(cv_auc2)       #k_best = 10**-2
```

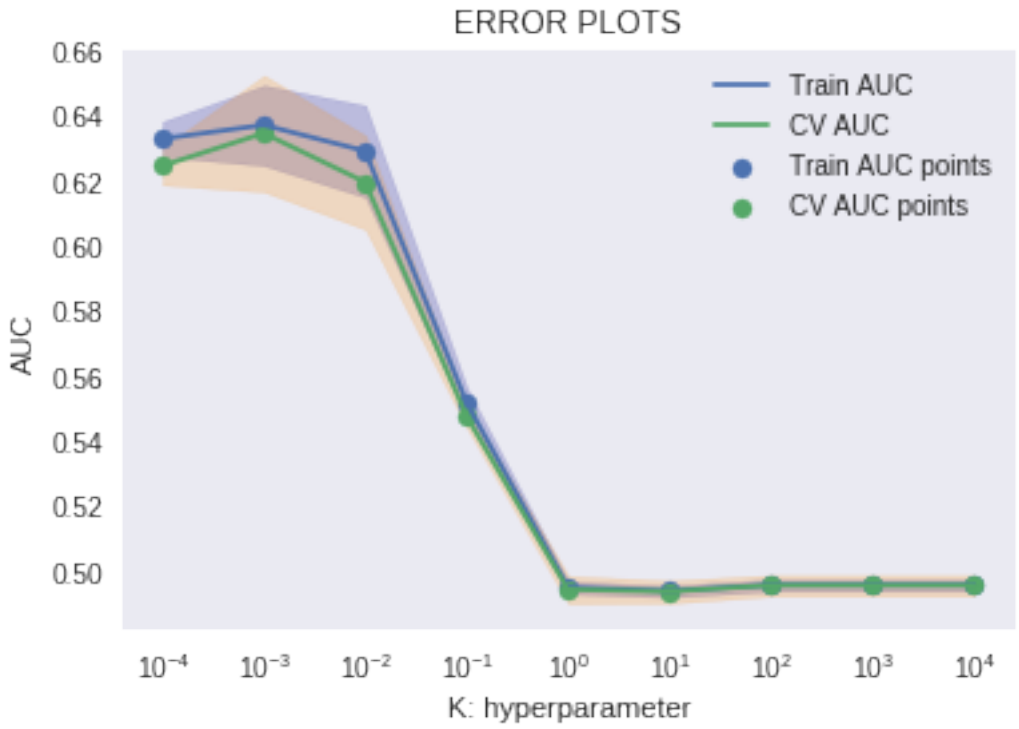


[0.80408012 0.79934247 0.80232514 0.63771475 0.54423078 0.54456665
 0.54456671 0.54456674 0.54456674]
[0.65226498 0.64570178 0.6530051 0.59265236 0.54336355 0.54428443
 0.54428417 0.5442845 0.5442845]

2.0.3 SET3

In [87]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier

sgc3 = SGDClassifier()

parameters = {'alpha': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
#parameters = {'alpha':[0.0001, 0.0025, 0.0005, 0.0075, 0.001, 0.025, .005, 0.075, 0.1, 0.25, 0.5, 0.75, 1]}

clf = GridSearchCV(sgc3, parameters, cv=3, scoring='roc_auc')
clr3 = clf.fit(X_tr3, y_train)

train_auc3= clf.cv_results_['mean_train_score']
train_auc_std3= clf.cv_results_['std_train_score']
cv_auc3 = clf.cv_results_['mean_test_score']
```

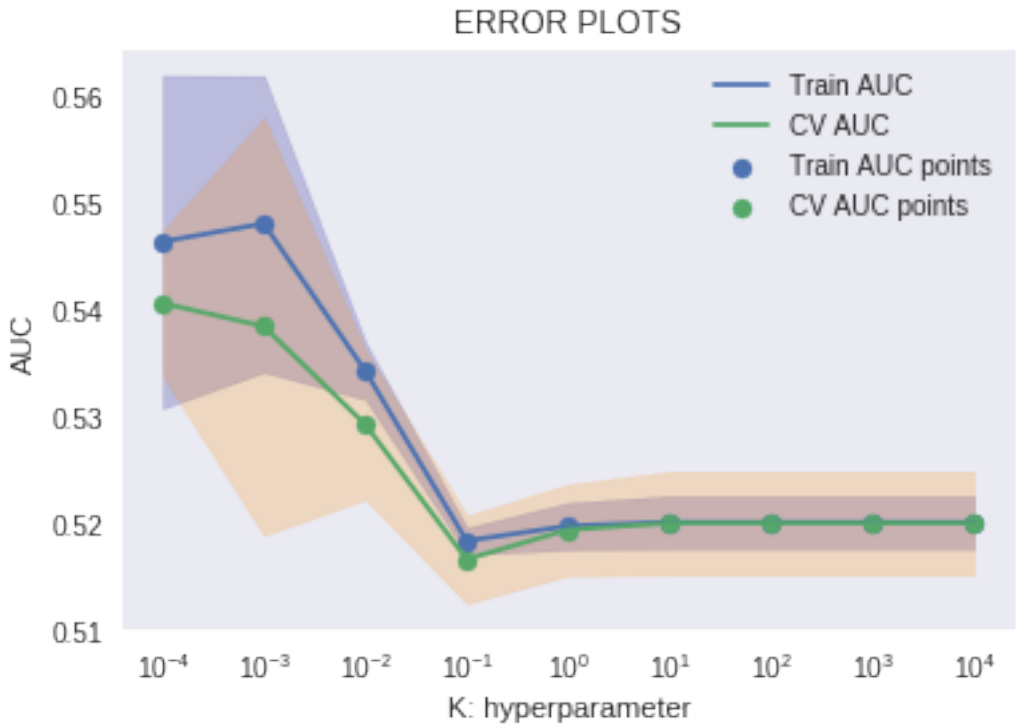
```
cv_auc_std3 = clf.cv_results_['std_test_score']

plt.plot(parameters['alpha'], train_auc3, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc3 - train_auc_std3,train_auc3 + train_auc_std3,alph

plt.plot(parameters['alpha'], cv_auc3, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc3 - cv_auc_std3, cv_auc3 + cv_auc_std3,alpha=0.2,col

plt.scatter(parameters['alpha'], train_auc3, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc3, label='CV AUC points')

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

print(train_auc3)
print(cv_auc3)      #k_best = 10**-2
```



```
[0.63285571 0.63705494 0.62909305 0.55183998 0.49513768 0.49401589
 0.4958747  0.4958747  0.49587443]
[0.62453652 0.63454626 0.61953308 0.54763071 0.49446758 0.49386046
 0.49575889 0.4957584  0.49575902]
```

2.0.4 SET4

In [88]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier

sgc4 = SGDClassifier()

parameters = {'alpha': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
#parameters = {'alpha':[0.0001, 0.0025, 0.0005, 0.0075, 0.001, 0.025, .005, 0.075, 0.1, 0.25, 0.5, 0.75, 1]}

clf = GridSearchCV(sgc4, parameters, cv=3, scoring='roc_auc')
clr4 = clf.fit(X_tr4, y_train)

train_auc4= clf.cv_results_['mean_train_score']
train_auc_std4= clf.cv_results_['std_train_score']
cv_auc4 = clf.cv_results_['mean_test_score']
cv_auc_std4 = clf.cv_results_['std_test_score']

plt.plot(parameters['alpha'], train_auc4, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],train_auc4 - train_auc_std4,train_auc4 + train_auc_std4,alpl

plt.plot(parameters['alpha'], cv_auc4, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['alpha'],cv_auc4 - cv_auc_std4, cv_auc4 + cv_auc_std4,alpha=0.2,col

plt.scatter(parameters['alpha'], train_auc4, label='Train AUC points')
plt.scatter(parameters['alpha'], cv_auc4, label='CV AUC points')

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

print(train_auc4)
print(cv_auc4)       #k_best = 10**-2
```



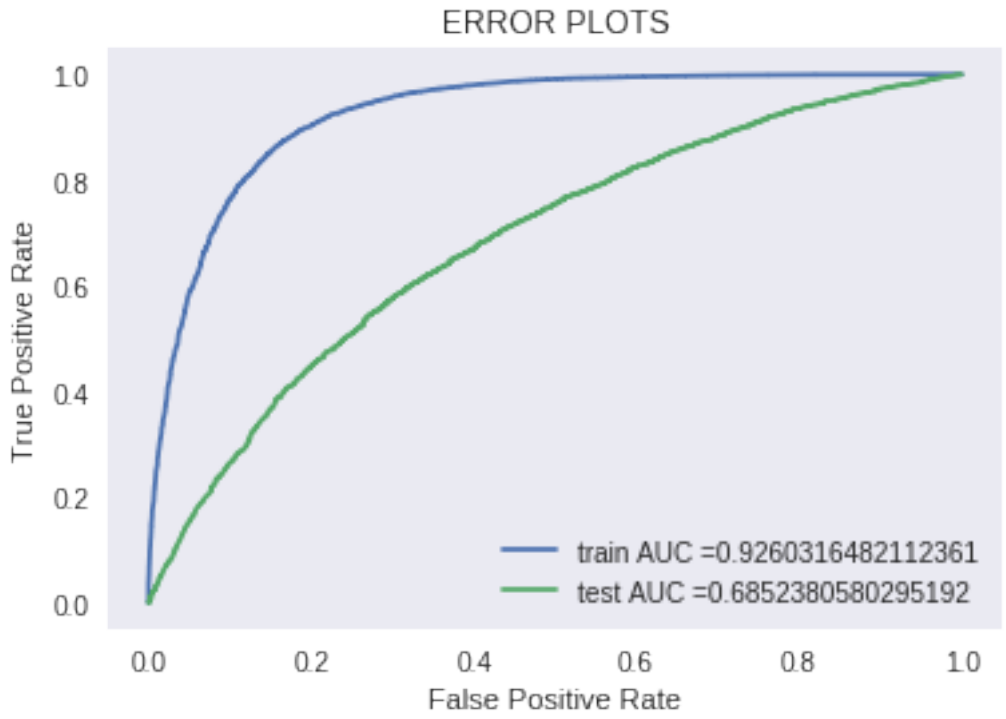

```
[0.54637571 0.54806802 0.53429212 0.51833951 0.51972722 0.52007569
 0.52007612 0.52007591 0.52007598]
[0.54061314 0.53844178 0.52925865 0.51660564 0.51937684 0.51999594
 0.51999668 0.51999637 0.51999662]
```

2.0.5 Plot of AUC on Test and Train

SET1

In [0]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive cl
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    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 [90]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_

from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.calibration import CalibratedClassifierCV



clf = SGDClassifier(loss='hinge', penalty='l2', alpha=10**-2)
clf.fit(X_tr1, y_train)

cc = CalibratedClassifierCV(base_estimator = clf, cv = 'prefit')
cc.fit(X_tr1, 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_pred1 = batch_predict(cc, X_tr1)
y_test_pred1 = batch_predict(cc, X_te1)

train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)

plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



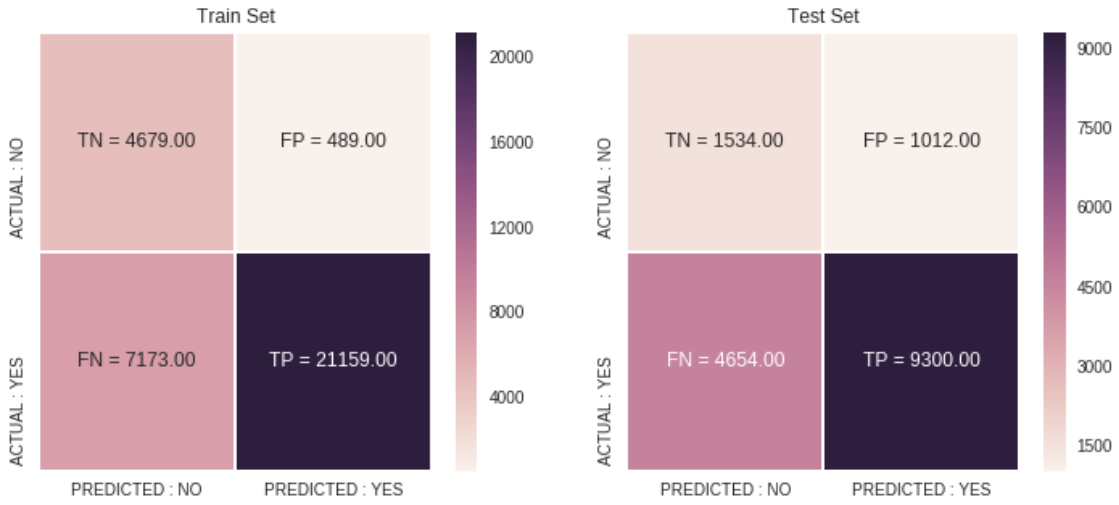
Confusion Matrix

```
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
```

```
In [92]: #https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
         import seaborn as sns; sns.set()

         con_m_train = confusion_matrix(y_train, predict(y_train_pred1, tr_thresholds1, train_fpr1, train_tpr
         con_m_test = confusion_matrix(y_test, predict(y_test_pred1, te_thresholds1, test_fpr1, test_tpr1))

         key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
```

```
fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_t

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytick
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytickl

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

plt.show()
```

the maximum value of tpr*(1-fpr) 0.7291350042770178 for threshold 0.922
the maximum value of tpr*(1-fpr) 0.40794818932051563 for threshold 0.911

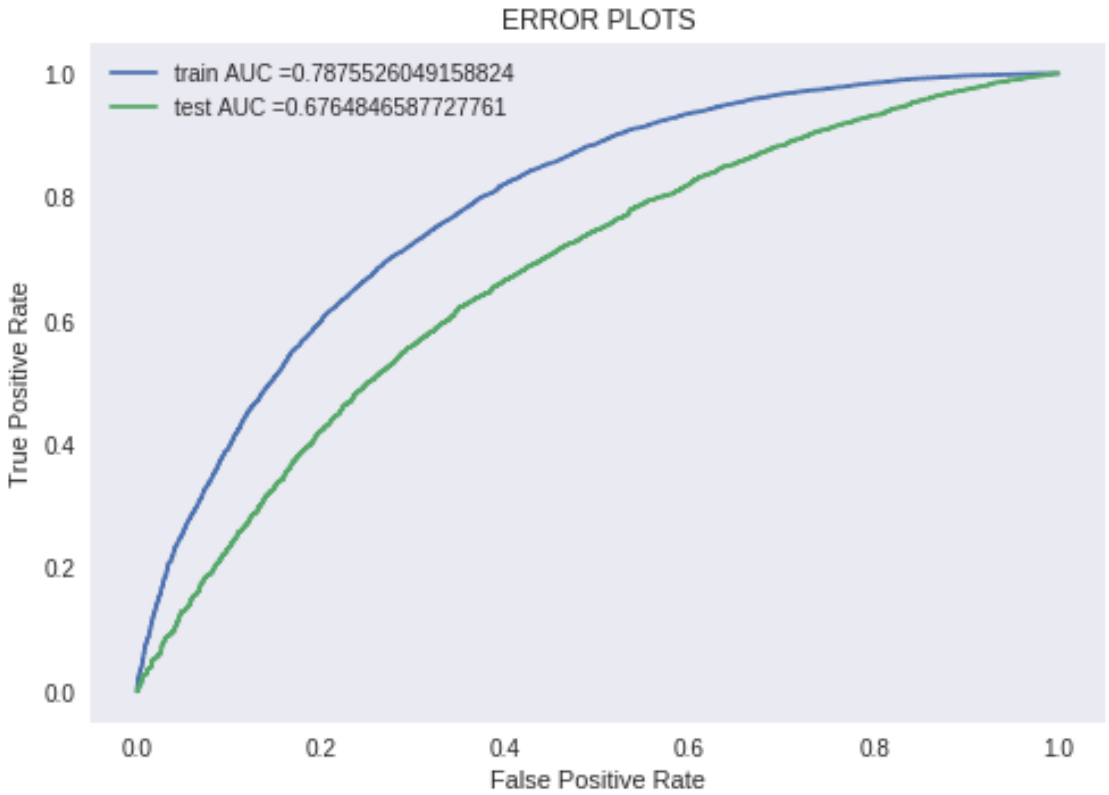


SET2

In [93]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.calibration import CalibratedClassifierCV



clf = SGDClassifier(loss='hinge', penalty='l2', alpha=10**-2)
```

```
clf.fit(X_tr2, y_train)

cc = CalibratedClassifierCV(base_estimator = clf, cv = 'prefit')
cc.fit(X_tr2, y_train)

y_train_pred2 = batch_predict(cc, X_tr2)
y_test_pred2 = batch_predict(cc, X_te2)

train_fpr2, train_tpr2, tr_thresholds2 = roc_curve(y_train, y_train_pred2)
test_fpr2, test_tpr2, te_thresholds2 = roc_curve(y_test, y_test_pred2)

plt.plot(train_fpr2, train_tpr2, label="train AUC ="+str(auc(train_fpr2, train_tpr2)))
plt.plot(test_fpr2, test_tpr2, label="test AUC ="+str(auc(test_fpr2, test_tpr2)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

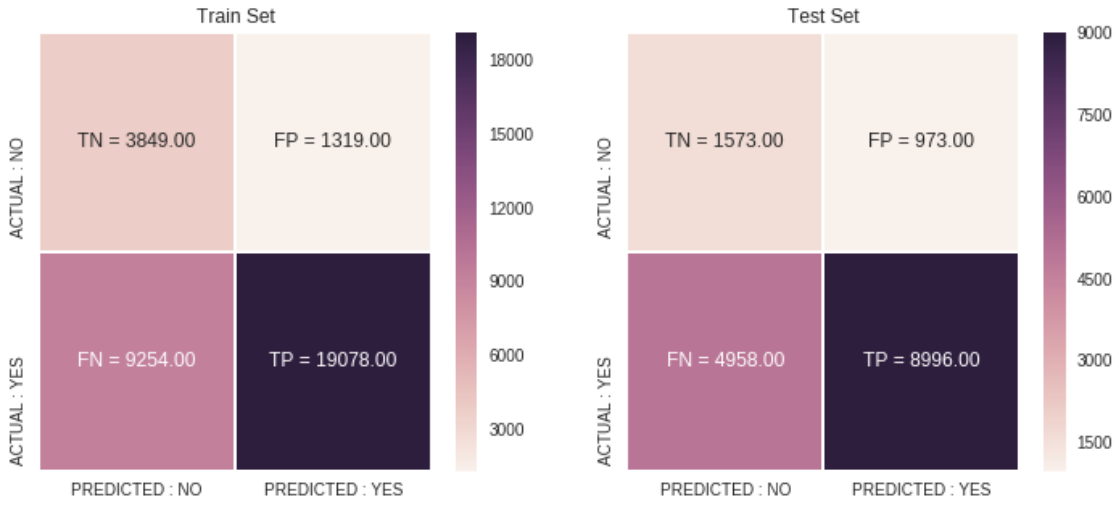


Confusion Matrix

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
```

In [95]:
```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()

con_m_train = confusion_matrix(y_train, predict(y_train_pred2, tr_thresholds2, train_fpr2, train_tpr
con_m_test = confusion_matrix(y_test, predict(y_test_pred2, te_thresholds2, test_fpr2, test_tpr2))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))

fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_t

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytick
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], yticklb

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

plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.5087373579918604 for threshold 0.864
the maximum value of tpr*(1-fpr) 0.40435294015653045 for threshold 0.861
```

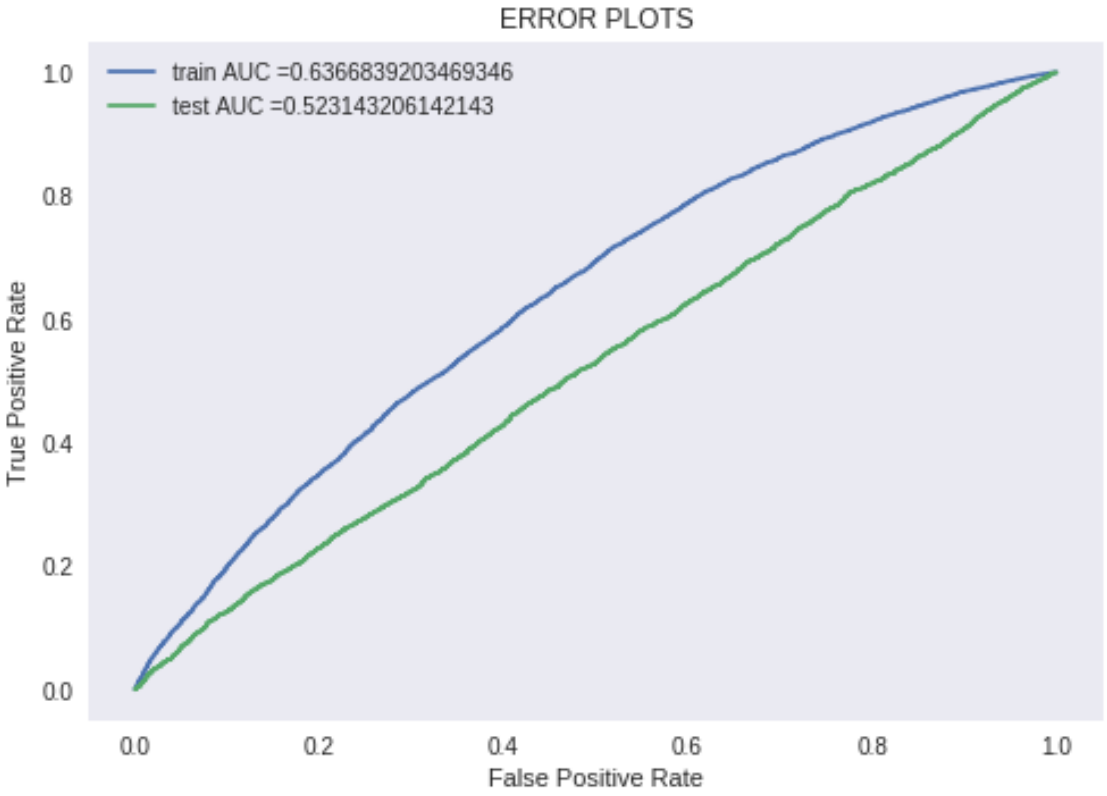


SET3

In [164]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.calibration import CalibratedClassifierCV



clf = SGDClassifier(loss='hinge', penalty='l2', alpha=10**-3)
clf.fit(X_tr3, y_train)

cc = CalibratedClassifierCV(base_estimator = clf, cv = 'prefit')
cc.fit(X_tr3, y_train)


y_train_pred3 = batch_predict(cc, X_tr3)
y_test_pred3 = batch_predict(cc, X_te3)

train_fpr3, train_tpr3, tr_thresholds3 = roc_curve(y_train, y_train_pred3)
test_fpr3, test_tpr3, te_thresholds3 = roc_curve(y_test, y_test_pred3)

plt.plot(train_fpr3, train_tpr3, label="train AUC ="+str(auc(train_fpr3, train_tpr3)))
plt.plot(test_fpr3, test_tpr3, label="test AUC ="+str(auc(test_fpr3, test_tpr3)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
```

```
plt.grid()
plt.show()
```



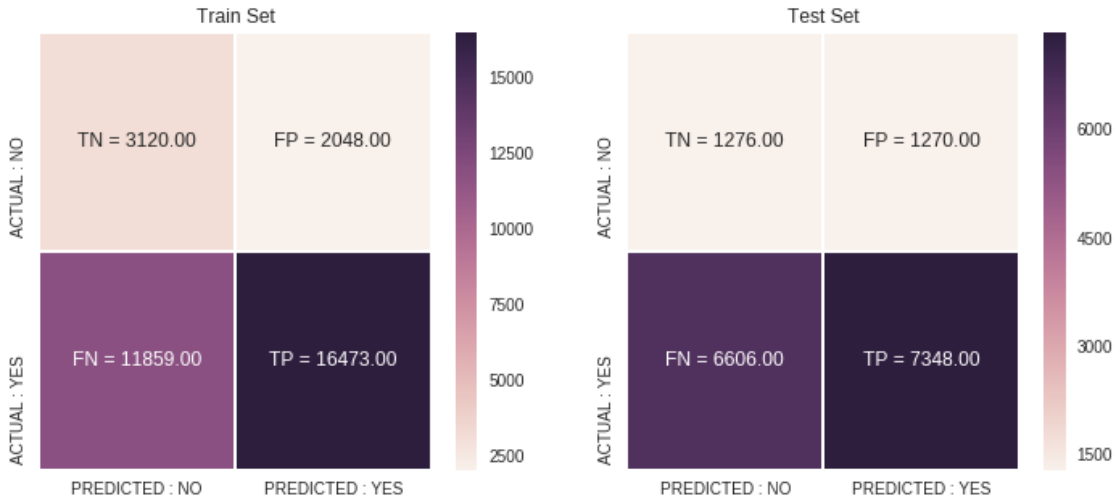
Confusion Matrix

```
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
```

```
In [166]: #https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
          import seaborn as sns; sns.set()
```

```
con_m_train = confusion_matrix(y_train, predict(y_train_pred3, tr_thresholds3, train_fpr3, train_tp
con_m_test = confusion_matrix(y_test, predict(y_test_pred3, te_thresholds3, test_fpr3, test_tpr3))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))

fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytic
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytick

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

plt.show()
```

the maximum value of tpr*(1-fpr) 0.35595388426219143 for threshold 0.848
the maximum value of tpr*(1-fpr) 0.26811020071447866 for threshold 0.983

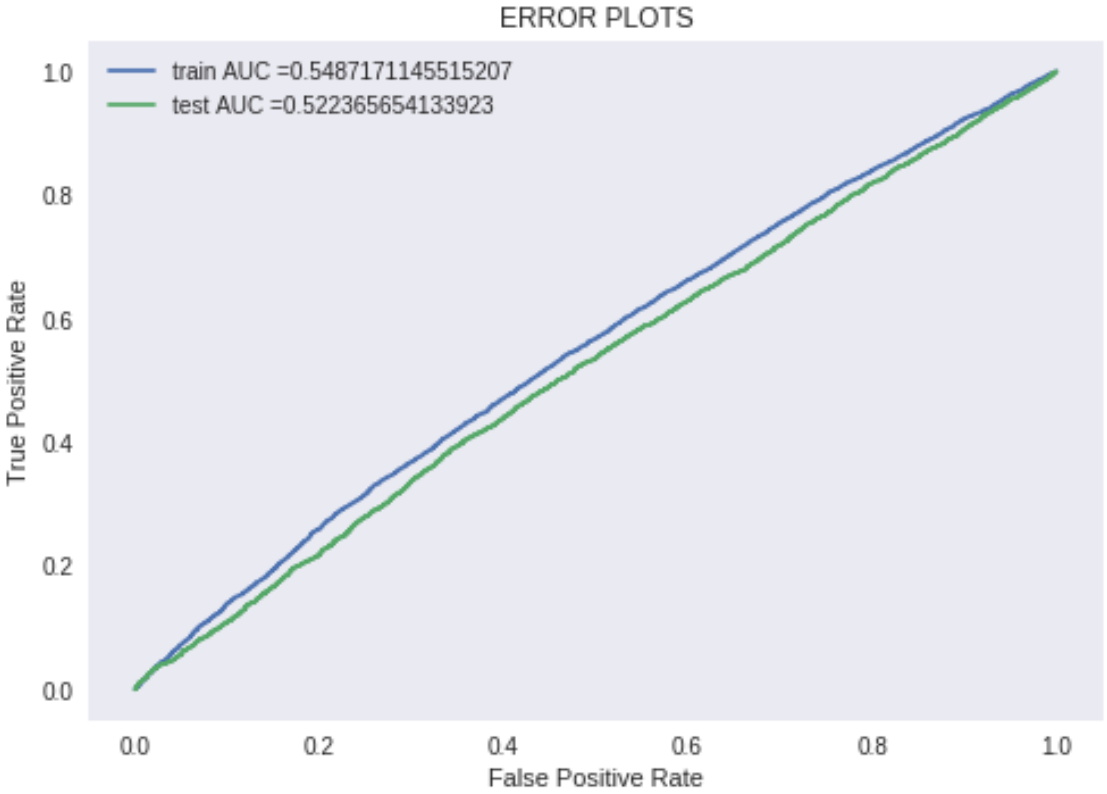


SET4

In [171]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.calibration import CalibratedClassifierCV
```

```
clf = SGDClassifier(loss='hinge', penalty='l2', alpha=10**-4)
clf.fit(X_tr4, y_train)

cc = CalibratedClassifierCV(base_estimator = clf, cv = 'prefit')
cc.fit(X_tr4, y_train)

y_train_pred4 = batch_predict(cc, X_tr4)
y_test_pred4 = batch_predict(cc, X_te4)

train_fpr4, train_tpr4, tr_thresholds4 = roc_curve(y_train, y_train_pred4)
test_fpr4, test_tpr4, te_thresholds4 = roc_curve(y_test, y_test_pred4)

plt.plot(train_fpr4, train_tpr4, label="train AUC ="+str(auc(train_fpr4, train_tpr4)))
plt.plot(test_fpr4, test_tpr4, label="test AUC ="+str(auc(test_fpr4, test_tpr4)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

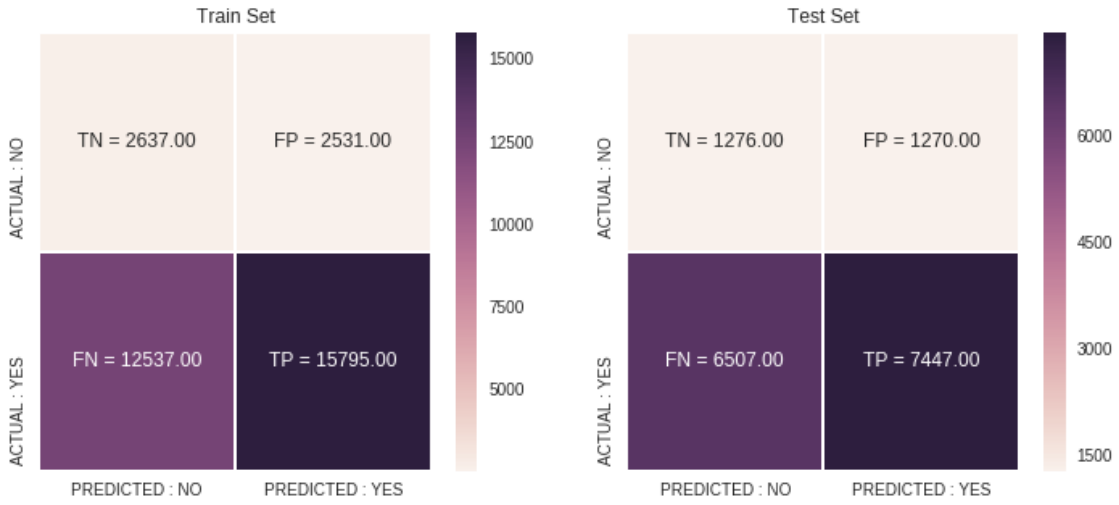


Confusion Matrix

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
```

In [172]:
```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()

con_m_train = confusion_matrix(y_train, predict(y_train_pred4, tr_thresholds4, train_fpr4, train_tp
con_m_test = confusion_matrix(y_test, predict(y_test_pred4, te_thresholds4, test_fpr4, test_tpr4))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))

fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytic
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytick

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

plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.288280327651915 for threshold 0.844
the maximum value of tpr*(1-fpr) 0.27151165297806584 for threshold 0.845
```



2.5 Logistic Regression with added Features Set 5

2.1 Encoding additional numeric features

In [102]: X_train.head(1)

Out[102]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_essay_1	project_essay_2	project_essay_3	...	clean_essay	clean_project_title	price	quantity	negative_sentiment_score	neutral_sentiment_score	positive_sentiment_score	compound_sentiment_score	num_words_essay	num_words_project_title
0	152779	p187939	f079e996501bdd78cff2dac333a50604	Ms.	PA	2016-11-27 17:36:22	Grades 3-5	This year I moved into a new position teaching...	Each day we meet as a class on the carpet/rug ...	NaN	...	year moved new position teaching supplemental ...	around world our classroom new rug	195.97	1	0.057	0.762	0.181	0.974	171	6

[1 rows x 25 columns]

Quantity

```
In [103]: from sklearn.preprocessing import Normalizer
          normalizerTr = Normalizer()
          # normalizer.fit(X_train['price'].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.
          normalizerTr.fit(X_train['quantity'].values.reshape(-1,1))

          X_train_q_norm = normalizerTr.transform(X_train['quantity'].values.reshape(-1,1))
          X_test_q_norm = normalizerTr.transform(X_test['quantity'].values.reshape(-1,1))

          print("After vectorizations")
          print(X_train_q_norm.shape, y_train.shape)
          print(X_test_q_norm.shape, y_test.shape)
          print("="*100)

After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

#Words in clean_project_title

```
In [104]: from sklearn.preprocessing import Normalizer
          normalizerTr = Normalizer()
          # normalizer.fit(X_train['price'].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.
          normalizerTr.fit(X_train['num_words_project_title'].values.reshape(-1,1))

          X_train_nwpt_norm = normalizerTr.transform(X_train['num_words_project_title'].values.reshape(-1,1
          X_test_nwpt_norm = normalizerTr.transform(X_test['num_words_project_title'].values.reshape(-1,1))

          print("After vectorizations")
          print(X_train_nwpt_norm.shape, y_train.shape)
          print(X_test_nwpt_norm.shape, y_test.shape)
          print("="*100)

After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

==========

#Words in clean_essay

```
In [105]: from sklearn.preprocessing import Normalizer
          normalizerTr = Normalizer()
          # normalizer.fit(X_train['price'].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.
          normalizerTr.fit(X_train['num_words_essay'].values.reshape(-1,1))

          X_train_nwe_norm = normalizerTr.transform(X_train['num_words_essay'].values.reshape(-1,1))
          X_test_nwe_norm = normalizerTr.transform(X_test['num_words_essay'].values.reshape(-1,1))

          print("After vectorizations")
          print(X_train_nwe_norm.shape, y_train.shape)
          print(X_test_nwe_norm.shape, y_test.shape)
          print("="*100)

After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

==========

negative_sentiment_score

```
In [106]: from sklearn.preprocessing import Normalizer
          normalizerTr = Normalizer()
          # normalizer.fit(X_train['price'].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.
          normalizerTr.fit(X_train['negative_sentiment_score'].values.reshape(-1,1))

          X_train_nss_norm = normalizerTr.transform(X_train['negative_sentiment_score'].values.reshape(-1,1))
          X_test_nss_norm = normalizerTr.transform(X_test['negative_sentiment_score'].values.reshape(-1,1))

          print("After vectorizations")
          print(X_train_nss_norm.shape, y_train.shape)
          print(X_test_nss_norm.shape, y_test.shape)
          print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
```

neutral_sentiment_score

```
In [107]: from sklearn.preprocessing import Normalizer
          normalizerTr = Normalizer()
          # normalizer.fit(X_train['price'].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.
          normalizerTr.fit(X_train['neutral_sentiment_score'].values.reshape(-1,1))

          X_train_ness_norm = normalizerTr.transform(X_train['neutral_sentiment_score'].values.reshape(-1,1))
          X_test_ness_norm = normalizerTr.transform(X_test['neutral_sentiment_score'].values.reshape(-1,1))

          print("After vectorizations")
          print(X_train_ness_norm.shape, y_train.shape)
          print(X_test_ness_norm.shape, y_test.shape)
          print("="*100)

After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
```

positive_sentiment_score

```
In [108]: from sklearn.preprocessing import Normalizer
          normalizerTr = Normalizer()
          # normalizer.fit(X_train['price'].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.
          normalizerTr.fit(X_train['positive_sentiment_score'].values.reshape(-1,1))

          X_train_pss_norm = normalizerTr.transform(X_train['positive_sentiment_score'].values.reshape(-1,1))
          X_test_pss_norm = normalizerTr.transform(X_test['positive_sentiment_score'].values.reshape(-1,1))

          print("After vectorizations")
```

```
print(X_train_pss_norm.shape, y_train.shape)
print(X_test_pss_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
```

compound_sentiment_score

```
In [109]: from sklearn.preprocessing import Normalizer
normalizerTr = Normalizer()
# normalizer.fit(X_train['price'].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.
normalizerTr.fit(X_train['compound_sentiment_score'].values.reshape(-1,1))

X_train_css_norm = normalizerTr.transform(X_train['compound_sentiment_score'].values.reshape(-1,1
X_test_css_norm = normalizerTr.transform(X_test['compound_sentiment_score'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_css_norm.shape, y_train.shape)
print(X_test_css_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
```

2.2 Elbow Method

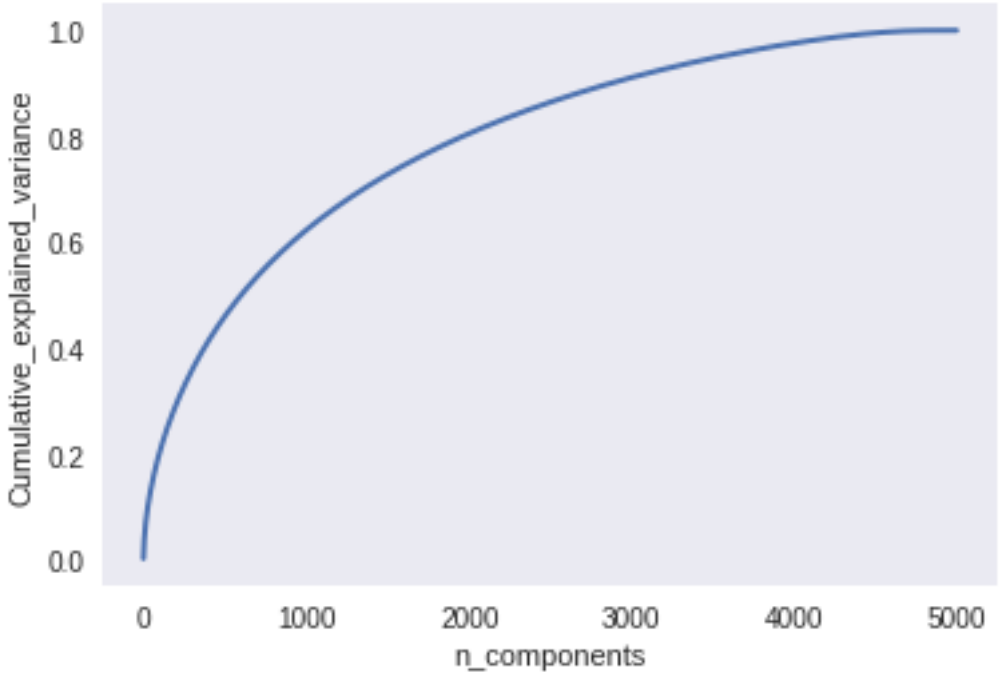
```
In [126]: #X_train_essay_tfidf

X_train_essay_tfidf.shape[1]-1
```

```
Out[126]: 4999
```

```
In [0]: from sklearn.decomposition import TruncatedSVD

tsvd = TruncatedSVD(n_components=X_train_essay_tfidf.shape[1]-1)
X_tsvd = tsvd.fit(X_train_essay_tfidf)
```

In [0]:
```
tsvd_var_ratios = tsvd.explained_variance_ratio_
```

In [129]:
```
percentage_var_explained = tsvd.explained_variance_ratio_ / np.sum(tsvd.explained_variance_ratio_
cum_var_explained = np.cumsum(percentage_var_explained)


plt.figure(1, figsize=(6, 4))

plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()
```



In [0]:
```
# SRC : https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_of_compon

# Create a function
def select_n_components(var_ratio, goal_var: float) -> int:
    # Set initial variance explained so far
    total_variance = 0.0

    # Set initial number of features
    n_components = 0
```

```
        # For the explained variance of each feature:
        for explained_variance in var_ratio:

            # Add the explained variance to the total
            total_variance += explained_variance

            # Add one to the number of components
            n_components += 1

            # If we reach our goal level of explained variance
            if total_variance >= goal_var:
                # End the loop
                break

        # Return the number of components
        return n_components
```

In [138]:
```
select_n_components(tsvd_var_ratios, 0.91)

    # 2990 components are explaining 91% of variance, therfore im reducing my dimensionality from 5000 to
```

Out[138]: 2990

In [0]:
```
tsvd_new = TruncatedSVD(n_components=2990)
X_tsvd_new = tsvd_new.fit_transform(X_train_essay_tfidf)
```

In [143]:
```
X_tsvd_new.shape
```

Out[143]: (33500, 2990)

In [0]:
```
X_test_tsvd_new = tsvd_new.transform(X_test_essay_tfidf)
```

In [145]:
```
X_test_tsvd_new.shape
```

Out[145]: (16500, 2990)

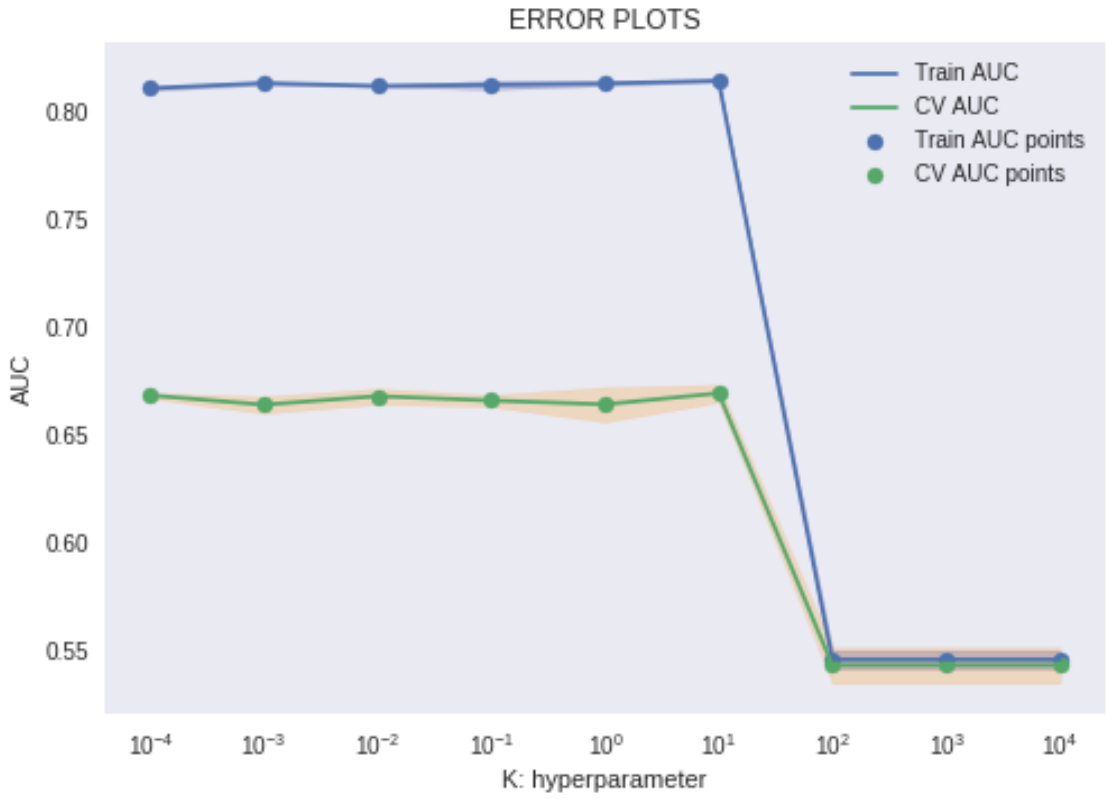
2.3 Merging all features with newly added features using hstack

In [146]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_new = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_grade_ohe, X_train_state_ohe, X
X_te_new = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_grade_ohe, X_test_state_ohe, X_tes

print("Final Data matrix")
print(X_tr_new.shape, y_train.shape)
print(X_te_new.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 3098) (33500,)
(16500, 3098) (16500,)
==========================================================================
```

2.4 Learning Curve (AUC)

```
In [147]: # https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
          from sklearn.model_selection import GridSearchCV
          from sklearn.linear_model import SGDClassifier

          sgc5 = SGDClassifier()

          parameters = {'alpha': [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10**4]}
          #parameters = {'alpha':[0.0001, 0.0025, 0.0005, 0.0075, 0.001, 0.025, .005, 0.075, 0.1, 0.25, 0.5, 0.75, 1]}

          clf = GridSearchCV(sgc5, parameters, cv=3, scoring='roc_auc')
          clr5 = clf.fit(X_tsvd_new, y_train)

          train_auc4= clf.cv_results_['mean_train_score']
          train_auc_std4= clf.cv_results_['std_train_score']
          cv_auc4 = clf.cv_results_['mean_test_score']
          cv_auc_std4 = clf.cv_results_['std_test_score']

          plt.plot(parameters['alpha'], train_auc4, label='Train AUC')
          # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
          plt.gca().fill_between(parameters['alpha'],train_auc4 - train_auc_std4,train_auc4 + train_auc_std4,alp

          plt.plot(parameters['alpha'], cv_auc4, label='CV AUC')
          # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
          plt.gca().fill_between(parameters['alpha'],cv_auc4 - cv_auc_std4, cv_auc4 + cv_auc_std4,alpha=0.2,co

          plt.scatter(parameters['alpha'], train_auc4, label='Train AUC points')
          plt.scatter(parameters['alpha'], cv_auc4, label='CV AUC points')

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

          print(train_auc4)
          print(cv_auc4)      #k_best = 10
```



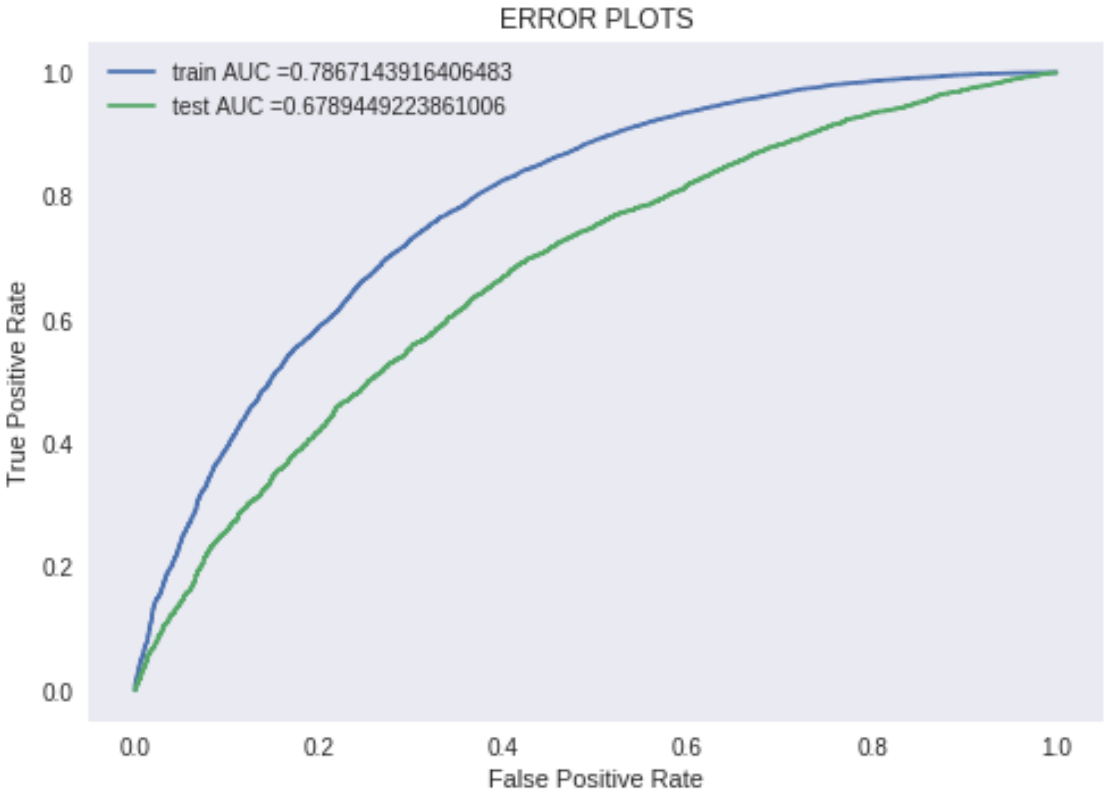
[0.81045782 0.81288338 0.8115571 0.81210058 0.81271945 0.81398927
 0.54560423 0.54560423 0.54560423]
[0.66809174 0.66373656 0.6677209 0.66568997 0.66390737 0.66907415
 0.54297383 0.54297383 0.54297383]

2.4.1 ROC Curve

In [150]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.calibration import CalibratedClassifierCV



clf = SGDClassifier(loss='hinge', penalty='l2', alpha=10**-2)
clf.fit(X_tsvd_new, y_train)

cc = CalibratedClassifierCV(base_estimator = clf, cv = 'prefit')
cc.fit(X_tsvd_new, y_train)
```

```
y_train_pred4 = batch_predict(cc, X_tsvd_new)
y_test_pred4 = batch_predict(cc, X_test_tsvd_new)

train_fpr4, train_tpr4, tr_thresholds4 = roc_curve(y_train, y_train_pred4)
test_fpr4, test_tpr4, te_thresholds4 = roc_curve(y_test, y_test_pred4)

plt.plot(train_fpr4, train_tpr4, label="train AUC ="+str(auc(train_fpr4, train_tpr4)))
plt.plot(test_fpr4, test_tpr4, label="test AUC ="+str(auc(test_fpr4, test_tpr4)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



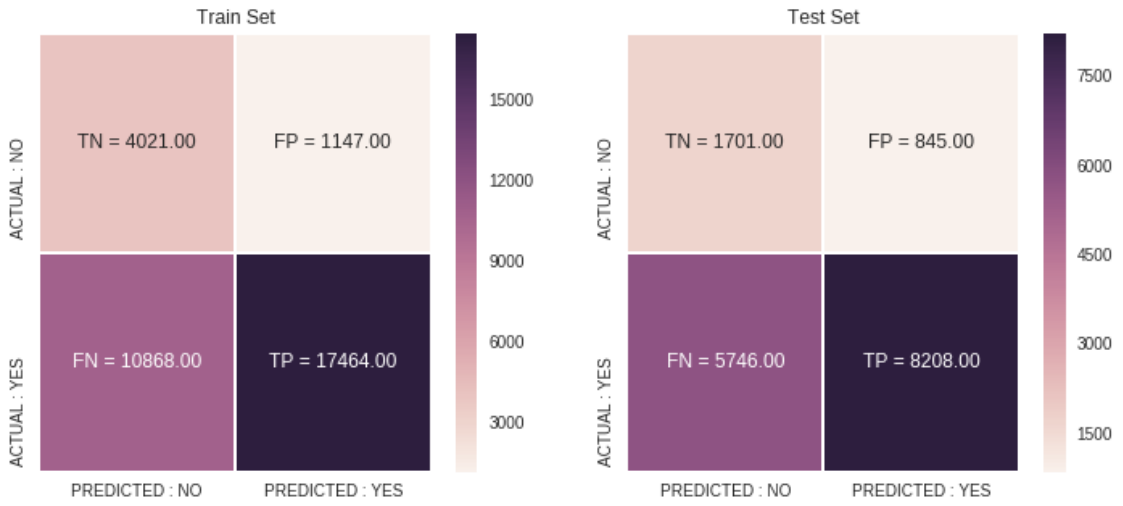
2.4.2 Confusion Matrix

```
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
```

In [152]:
```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()

con_m_train = confusion_matrix(y_train, predict(y_train_pred4, tr_thresholds4, train_fpr4, train_tp
con_m_test = confusion_matrix(y_test, predict(y_test_pred4, te_thresholds4, test_fpr4, test_tpr4))

key = (np.asarray([['TN','FP'], ['FN', 'TP']]))

fig, ax = plt.subplots(1,2, figsize=(12,5))

labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flatten(), con_m_

sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytic
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytick

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

plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.5124678103591689 for threshold 0.882
the maximum value of tpr*(1-fpr) 0.40269211338658356 for threshold 0.881
```



3. Conclusions

In [173]:
```
# Please compare all your models using Prettytable library
```

```
# Please compare all your models using Prettytable library

from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "HyperParameter", "AUC"]

x.add_row(["BOW",         "SVM", 10**-2, 0.685])
x.add_row(["TFIDF",       "SVM", 10**-2, 0.676])
x.add_row(["Avg W2V",     "SVM", 10**-3, 0.523])
x.add_row(["TFIDF-W2v",   "SVM", 10**-4, 0.522])
x.add_row(["SET5",        "SVM", 10**-2, 0.678])



print(x)
```

```
+------------+-------+---------------+-------+
| Vectorizer | Model | HyperParameter |  AUC  |
+------------+-------+---------------+-------+
|    BOW     |  SVM  |      0.01     | 0.685 |
|   TFIDF    |  SVM  |      0.01     | 0.676 |
|  Avg W2V   |  SVM  |     0.001     | 0.523 |
| TFIDF-W2v  |  SVM  |     0.0001    | 0.522 |
|    SET5    |  SVM  |      0.01     | 0.678 |
+------------+-------+---------------+-------+
```

2.4.3 Observations

As it can be seen from the above table, that the model is performing better than random model, from all the sets, TFIDF is working fairly well having AUC score of 0.676

2.4.4 Conclusions

I took **50000 datapoints** for my analysis and building my model

- I splitted the dataset into train, cv and test dataset
- Preprocessed all the text fetaures
- Vectorized all the text, categorical and numerical features, for text i used BOW & TFIDF
- Merged all features using hstack as instructed

- Using train dataset, i plotted my AUC curve using GridSearchCV using 3Fold Cross Validation for both categories
- from AUC curve, i picked best alpha. using best alpha, i plotted ROC curve on train and test data.
- Then i plotted my confusion matrix for both the sets.
- Atlast you can see my result in tabular format.
- **For SET5, i did dimensionality reduction of essay text using tSVD, and i got 2990 components from 5000 components explaining 91% of variance.**