

# Assignment 5: Apply Logistic Regression on Donors Choose dataset

This exercise is to apply Logistic regression on Donors Choose dataset and predict approval of a new project proposal.

Relevant Information : The dataset is divided into two files -

1. train.csv file which contains information regarding projects, schools and teachers who submitted the projects.
2. resources.csv which provides information about the resources required for each project.

**OBJECTIVE : The goal 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 using Logistic regression algorithm.**

```
In [2]: # Importing the required libraries
# Warning reference : https://stackoverflow.com/questions/41658568/chunkize-warning-while-installing-gensim

%matplotlib inline
import warnings
warnings.filterwarnings(action='ignore', category = UserWarning , module = 'gensim')

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
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. Reading the data

```
In [3]: project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')

print("\nNumber of data points in train data", project_data.shape)
print('-'*120)
print("The attributes of data :", project_data.columns.values)
```

Number of data points in train data (109248, 17)

```
-----
-
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]: `project_data.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109248 entries, 0 to 109247
Data columns (total 17 columns):
Unnamed: 0                109248 non-null int64
id                        109248 non-null object
teacher_id                109248 non-null object
teacher_prefix            109245 non-null object
school_state              109248 non-null object
project_submitted_datetime 109248 non-null object
project_grade_category    109248 non-null object
project_subject_categories 109248 non-null object
project_subject_subcategories 109248 non-null object
project_title             109248 non-null object
project_essay_1           109248 non-null object
project_essay_2           109248 non-null object
project_essay_3           3758 non-null object
project_essay_4           3758 non-null object
project_resource_summary  109248 non-null object
teacher_number_of_previously_posted_projects 109248 non-null int64
project_is_approved       109248 non-null int64
dtypes: int64(3), object(14)
memory usage: 14.2+ MB
```

## NOTE:

1. We have a total of 109248 datapoints and 17 columns.
2. Now we have to sort the data according to date and time so as to have a better prediction on the future data (Test data).
3. As we can see there are null points for project\_essay\_3 and project\_essay\_4. Only 3758 points are not null.
4. In teacher\_prefix there are 109245 points which means 3 points are null.

## Sorting according to date

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]

print(cols)
```

```
['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state', 'Date', 'project_grade_category', 'project_subject_categories', 'project_subject_subcategories', 'project_title', 'project_essay_1', 'project_essay_2', 'project_essay_3', 'project_essay_4', 'project_resource_summary', 'teacher_number_of_previously_posted_projects', 'project_is_approved']
```

In [6]: `# Dropping the Unnamed column`

```
project_data.drop('Unnamed: 0', axis=1, inplace=True)
```

## Adding the price and quantity column from resource\_data to the project\_data

In [7]: `# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in-one-step`  
`price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()`

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

```
# Deleting the resource data variable
del resource_data
```

## NOTE:

1. As we can see that the price and the quantity column has been added to the project\_data
2. This is where the preprocessing will start.
3. 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
- essay : text data
- quantity : numerical
- teacher\_number\_of\_previously\_posted\_projects : numerical
- price : numerical

## 2. Preprocessing Data

### project\_subject\_categories

```
In [8]: categories = 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 categories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the category based on space "Math & Science"=> "Math","&", "Sci
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

### project\_subject\_subcategories

```
In [9]: sub_categories = 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_categories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the category based on space "Math & Science"=> "Math","&", "Sci
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp +=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

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

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

## Essay

```
100%|██████████████████████████████████████████████████████████████████████████████| 109248/109248 [01:19<00:00, 1370.27it/s]
```

## project\_title

```
Out[13]: count          109248
          unique          100851
          top      Flexible Seating
          freq              234
          Name: project_title, dtype: object
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 109248/109248 [00:03<00:00, 30336.84it/s]
```

```
In [15]: # Adding processed essay columns in place of previous essays columns and dropping the previous columns
```

```
In [16]: ## Project_title  
  
# Adding processed project_title columns in place of previous project_title column and dropping the previous column  
  
project_data['clean_titles'] = preprocessed_titles  
  
project_data.drop(['project title'], axis=1, inplace=True)
```

```
In [17]: # Dropping NAN row
# https://stackoverflow.com/questions/46091924/python-how-to-drop-a-row-whose-particular-column-is-empty-nan

project_data.dropna(axis = 0, inplace = True, subset = ['teacher_prefix'])
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109245 entries, 0 to 109247
Data columns (total 15 columns):
id                                109245 non-null object
teacher_id                        109245 non-null object
teacher_prefix                    109245 non-null object
school_state                      109245 non-null object
Date                              109245 non-null datetime64[ns]
project_grade_category            109245 non-null object
project_resource_summary          109245 non-null object
teacher_number_of_previously_posted_projects 109245 non-null int64
project_is_approved              109245 non-null int64
price                             109245 non-null float64
quantity                          109245 non-null int64
clean_categories                  109245 non-null object
clean_subcategories               109245 non-null object
clean_essays                      109245 non-null object
clean_titles                      109245 non-null object
dtypes: datetime64[ns](1), float64(1), int64(3), object(10)
memory usage: 13.3+ MB
```

- **school state** : categorical data



- **clean\_categories** : categorical data
- **clean\_subcategories** : categorical data
- **project\_grade\_category** :categorical data
- **teacher\_prefix** : categorical data
- **quantity** : numerical data
- **teacher\_number\_of\_previously\_posted\_projects** : numerical data
- **price** : numerical data
- **sentiment score's of each of the essay** : numerical data
- **number of words in the title** : numerical data
- **number of words in the combine essays** : numerical data

## NOTE:

- Now we have to get:
  - number of words in the title
  - number of words in the combined essays
  - the sentiment score's of each of the essay
- WE ARE TAKING THE clean\_essay AND clean\_title TO FIND THE NUMBER OF WORDS IN EACH OF THEM, INSTEAD OF THE ORIGINAL ESSAYS AND TITLES

### - Number of words in the title

```
In [19]: # List to contain the number of words per sentence
words_in_clean_title = []

# Looping over each title and counting the words
for sent in project_data['clean_titles'].values:
    num_words = len(sent.split())
    words_in_clean_title.append(num_words)

# Adding a new column to the project_data
project_data['words_in_title'] = words_in_clean_title
```

```
In [20]: project_data['clean_titles'].head(3)
```

```
Out[20]: 0      engineering steam primary classroom
1              sensory tools focus
2  mobile learning mobile listening center
Name: clean_titles, dtype: object
```

```
In [21]: project_data['words_in_title'].head(3)
```

```
Out[21]: 0      4
1      3
2      5
Name: words_in_title, dtype: int64
```

## NOTE:

- The words in each of the clean\_titles are counted and are stored in the new column named : words\_in\_title

### - Number of words in the combined essays

```
In [22]: # List to contain the number of words per sentence
words_in_clean_essay = []

# Looping over each essay and counting the words
for sent in project_data['clean_essays'].values:
    num_words = len(sent.split())
    words_in_clean_essay.append(num_words)

# Adding a new column to the project_data
project_data['words_in_essay'] = words_in_clean_essay
```

```
In [23]: project_data['clean_essays'].head(3)
```

```
Out[23]: 0      fortunate enough use fairy tale stem kits clas...
1      imagine 8 9 years old third grade classroom se...
2      class 24 students comes diverse learners stude...
Name: clean_essays, dtype: object
```

```
In [24]: project_data['words_in_essay'].head(3)
```

```
Out[24]: 0      156
          1      159
          2     106
          Name: words_in_essay, dtype: int64
```

**NOTE:**

- The words in each of the `clean_essay` are counted and are stored in the new column named : `words_in_essay`

- Sentiment score's of each of the essay

```
In [25]: from nltk.sentiment.vader import SentimentIntensityAnalyzer

# List that contain all the scores (NEGATIVE : neg, NEUTRAL : neu, POSITIVE : pos, COMPOUND)
sentiment = []
for sent in tqdm(project_data['clean_essays'].values):

    sid = SentimentIntensityAnalyzer()
    sentiment.append(sid.polarity_scores(sent))
```

[illegible]

```
In [26]: sentiment[0:5]
```

```
Out[26]: [{'neg': 0.013, 'neu': 0.773, 'pos': 0.214, 'compound': 0.9867},
          {'neg': 0.078, 'neu': 0.65, 'pos': 0.272, 'compound': 0.9899},
          {'neg': 0.016, 'neu': 0.706, 'pos': 0.278, 'compound': 0.9864},
          {'neg': 0.031, 'neu': 0.775, 'pos': 0.194, 'compound': 0.9524},
          {'neg': 0.031, 'neu': 0.653, 'pos': 0.315, 'compound': 0.9873}]
```

## Converting the dictionary to the dataframe

```
In [25]: # Converting the dict to dataframe
sentiment_data = pd.DataFrame(sentiment)

#### Saving the data in csv file for further use (as it takes a lot of time)
#sentiment_data.to_csv('sentiment_data.csv', index=False)

### Reading from the scv file
#sentiment_data = pd.read_csv('sentiment_data.csv')
```

```
In [26]: # Resetting the indexes
# https://stackoverflow.com/questions/40339886/pandas-concat-generates-nan-values
sentiment_data.reset_index(drop=True, inplace=True)
project_data.reset_index(drop=True, inplace=True)

# concatenating the two data frames
project_data = pd.concat([project_data, sentiment_data], axis=1)
```

```
In [27]: sentiment_data.head(3)
```

```
Out[27]:
```

	<b>compound</b>	<b>neg</b>	<b>neu</b>	<b>pos</b>
<b>0</b>	0.9867	0.013	0.773	0.214
<b>1</b>	0.9899	0.078	0.650	0.272
<b>2</b>	0.9864	0.016	0.706	0.278

In [28]: `project_data.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109245 entries, 0 to 109244
Data columns (total 21 columns):
id                109245 non-null object
teacher_id        109245 non-null object
teacher_prefix    109245 non-null object
school_state      109245 non-null object
Date              109245 non-null datetime64[ns]
project_grade_category 109245 non-null object
project_resource_summary 109245 non-null object
teacher_number_of_previously_posted_projects 109245 non-null int64
project_is_approved 109245 non-null int64
price             109245 non-null float64
quantity          109245 non-null int64
clean_categories  109245 non-null object
clean_subcategories 109245 non-null object
clean_essays       109245 non-null object
clean_titles       109245 non-null object
words_in_title     109245 non-null int64
words_in_essay     109245 non-null int64
compound          109245 non-null float64
neg               109245 non-null float64
neu               109245 non-null float64
pos               109245 non-null float64
dtypes: datetime64[ns](1), float64(5), int64(5), object(10)
memory usage: 17.5+ MB
```

## NOTE:

- Till now we have preprocessed the data.
- Now we have to split the data and vectorize the data for BOW, TF-IDF, Avg W2V and TFIDF weighted W2Vec

\*\*\*\*\***ASSIGNMENT**\*\*\*\*\*

## 3. Splitting data into Train and cross validation(or test): Stratified Sampling

In [29]: `# Creating Label and feature data frame : Label- y, Features- X`

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

print(y.shape)
print(X.shape)

(109245,)
(109245, 20)
```

In [30]: `## train test cross-validation split`  
`# Referance : https://stackoverflow.com/questions/34842405/parameter-stratify-from-method-train-test-split-scikit-learn`

```
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)
```

## NOTE:

- This `stratify` parameter makes a split so that the proportion of values in the sample produced will be the same as the proportion of values provided to parameter stratify.
- For example, if variable y is a binary categorical variable with values 0 and 1 and there are 25% of zeros and 75% of ones, stratify=y will make sure that your random split has 25% of 0's and 75% of 1's.

In [31]: `## Shape of the matrices`

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

(49039, 20) (49039,)
(24155, 20) (24155,)
(36051, 20) (36051,)
```



## NOTE:

1. We will now use the train data for training our model, cv data to validate the model and perform testing on the test data

## 4. Make Data Model Ready: encoding numerical, categorical features

### Vectorizing Categorical Features

#### 1. clean\_categories

```
In [32]: # We use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer

vectorizer_clean_categories = CountVectorizer()

# We will fit the train data only
vectorizer_clean_categories.fit(X_train['clean_categories'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_category = vectorizer_clean_categories.transform(X_train['clean_categories'].values)
X_cv_clean_category = vectorizer_clean_categories.transform(X_cv['clean_categories'].values)
X_test_clean_category = vectorizer_clean_categories.transform(X_test['clean_categories'].values)

print("Clean categories are vectorized\n")
print(X_train_clean_category.shape, y_train.shape)
print(X_cv_clean_category.shape, y_cv.shape)
print(X_test_clean_category.shape, y_test.shape)

print(vectorizer_clean_categories.get_feature_names())
```

Clean categories are vectorized

```
(49039, 9) (49039,)
(24155, 9) (24155,)
(36051, 9) (36051,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_art
s', 'specialneeds', 'warmth']
```

#### 2. clean\_subcategories

```
In [33]: vectorizer_clean_subcategories = CountVectorizer()

# We will fit the train data only
vectorizer_clean_subcategories.fit(X_train['clean_subcategories'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategories = vectorizer_clean_subcategories.transform(X_train['clean_subcategories'].values)
X_cv_clean_subcategories = vectorizer_clean_subcategories.transform(X_cv['clean_subcategories'].values)
X_test_clean_subcategories = vectorizer_clean_subcategories.transform(X_test['clean_subcategories'].values)

print("Clean_subcategories are vectorized\n")
print(X_train_clean_subcategories.shape, y_train.shape)
print(X_cv_clean_subcategories.shape, y_cv.shape)
print(X_test_clean_subcategories.shape, y_test.shape)
print(vectorizer_clean_subcategories.get_feature_names())
```

Clean\_subcategories are vectorized

```
(49039, 30) (49039,)
(24155, 30) (24155,)
(36051, 30) (36051,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice',
'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlanguag
es', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'ma
thematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialsciences', 'specialne
eds', 'teamsports', 'visualarts', 'warmth']
```

#### 3. teacher\_prefix

```
In [34]: vectorizer_teacher_prefix = CountVectorizer()

# We will fit the train data only
vectorizer_teacher_prefix.fit(X_train['teacher_prefix'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix = vectorizer_teacher_prefix.transform(X_train['teacher_prefix'].values)
X_cv_teacher_prefix = vectorizer_teacher_prefix.transform(X_cv['teacher_prefix'].values)
X_test_teacher_prefix = vectorizer_teacher_prefix.transform(X_test['teacher_prefix'].values)

print("Teacher_prefix are vectorized\n")
print(X_train_teacher_prefix.shape, y_train.shape)
print(X_cv_teacher_prefix.shape, y_cv.shape)
print(X_test_teacher_prefix.shape, y_test.shape)
print(vectorizer_teacher_prefix.get_feature_names())
```

Teacher\_prefix are vectorized

```
(49039, 5) (49039,)
(24155, 5) (24155,)
(36051, 5) (36051,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

#### 4. school\_state

```
In [35]: vectorizer_school_state = CountVectorizer()

# We will fit the train data only
vectorizer_school_state.fit(X_train['school_state'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_school_state = vectorizer_school_state.transform(X_train['school_state'].values)
X_cv_school_state = vectorizer_school_state.transform(X_cv['school_state'].values)
X_test_school_state = vectorizer_school_state.transform(X_test['school_state'].values)

print("School_state are vectorized\n")
print(X_train_school_state.shape, y_train.shape)
print(X_cv_school_state.shape, y_cv.shape)
print(X_test_school_state.shape, y_test.shape)
print(vectorizer_school_state.get_feature_names())
```

School\_state are vectorized

```
(49039, 51) (49039,)
(24155, 51) (24155,)
(36051, 51) (36051,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'm',
a', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa',
'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
```

#### 5. project\_grade\_category

```
In [36]: #This step is to intialize a vectorizer with vocab from train data

# Creating the List of grades

grades = list(set(project_data['project_grade_category'].values))

# we use count vectorizer to convert the values into one hot encoded features
# We will fit the train data only
vectorizer_grade_category = CountVectorizer(vocabulary = grades, lowercase=False, binary=True)
vectorizer_grade_category.fit(X_train['project_grade_category'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade = vectorizer_grade_category.transform(X_train['project_grade_category'].values)
X_cv_project_grade = vectorizer_grade_category.transform(X_cv['project_grade_category'].values)
X_test_project_grade = vectorizer_grade_category.transform(X_test['project_grade_category'].values)

print("Project_grade_category are vectorized\n")
print(X_train_project_grade.shape, y_train.shape)
print(X_cv_project_grade.shape, y_cv.shape)
print(X_test_project_grade.shape, y_test.shape)
print(vectorizer_grade_category.get_feature_names())
```

Project\_grade\_category are vectorized

```
(49039, 4) (49039,)
(24155, 4) (24155,)
(36051, 4) (36051,)
['Grades PreK-2', 'Grades 3-5', 'Grades 9-12', 'Grades 6-8']
```

## Standardizing Numerical features

### 1. price

```
In [37]: # standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

from sklearn.preprocessing import StandardScaler

# 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.

price_scalar = StandardScaler(with_mean = False)

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

# Now standardize the data with above mean and variance.
X_train_price = price_scalar.transform(X_train['price'].values.reshape(-1,1))
X_cv_price = price_scalar.transform(X_cv['price'].values.reshape(-1,1))
X_test_price = price_scalar.transform(X_test['price'].values.reshape(-1,1))

print("Price is standardized\n")
print(X_train_price.shape, y_train.shape)
print(X_cv_price.shape, y_cv.shape)
print(X_test_price.shape, y_test.shape)
```

Mean : 298.7174079814026, Standard deviation : 367.01337903747947  
Price is standardized

```
(49039, 1) (49039,)
(24155, 1) (24155,)
(36051, 1) (36051,)
```

### 2. teacher\_number\_of\_previously\_posted\_projects

```
In [38]: # standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
# price_standardized = standardScalar.fit(project_data['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.

# https://stackoverflow.com/questions/29086398/sklearn-turning-off-warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

from sklearn.preprocessing import StandardScaler
previous_post_scalar = StandardScaler(with_mean = False)

# 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.

# We will fit the train data only
# finding the mean and standard deviation of this data
previous_post_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
print(f"Mean : {previous_post_scalar.mean_[0]}, Standard deviation : {np.sqrt(previous_post_scalar.var_[0])}")

X_train_previous_projects = previous_post_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].value
X_cv_previous_projects = previous_post_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.res
X_test_previous_projects = previous_post_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.

print("Teacher_number_of_previously_posted_projects is standardized\n")
print(X_train_previous_projects.shape, y_train.shape)
print(X_cv_previous_projects.shape, y_cv.shape)
print(X_test_previous_projects.shape, y_test.shape)
```

Mean : 11.121882583250066, Standard deviation : 27.675398832912535

Teacher\_number\_of\_previously\_posted\_projects is standardized

(49039, 1) (49039,)

(24155, 1) (24155,)

(36051, 1) (36051,)

### 3. quantity

```
In [39]: # standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
# price_standardized = standardScalar.fit(project_data['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.

# https://stackoverflow.com/questions/29086398/sklearn-turning-off-warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

from sklearn.preprocessing import StandardScaler
quantity_scalar = StandardScaler(with_mean = False)

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

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

print("quantity is standardized")
print(X_train_quantity.shape, y_train.shape)
print(X_cv_quantity.shape, y_cv.shape)
print(X_test_quantity.shape, y_test.shape)
```

Mean : 16.846204041681112, Standard deviation : 25.320236566773072

quantity is standardized

(49039, 1) (49039,)

(24155, 1) (24155,)

(36051, 1) (36051,)

## 4. words\_in\_title

```
In [40]: # standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

from sklearn.preprocessing import StandardScaler

# 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.

words_title_scaler = StandardScaler(with_mean = False)

# We will fit the train data only
words_title_scaler.fit(X_train['words_in_title'].values.reshape(-1,1)) # finding the mean and standard deviation of this
print(f"Mean : {words_title_scaler.mean_[0]}, Standard deviation : {np.sqrt(words_title_scaler.var_[0])}")

# Now standardize the data with above mean and variance.
X_train_words_in_title = words_title_scaler.transform(X_train['words_in_title'].values.reshape(-1,1))
X_cv_words_in_title = words_title_scaler.transform(X_cv['words_in_title'].values.reshape(-1,1))
X_test_words_in_title = words_title_scaler.transform(X_test['words_in_title'].values.reshape(-1,1))

print("Words_in_title are standardized\n")
print(X_train_words_in_title.shape, y_train.shape)
print(X_cv_words_in_title.shape, y_cv.shape)
print(X_test_words_in_title.shape, y_test.shape)
```

Mean : 4.337343746813761, Standard deviation : 1.7859310152045504  
Words\_in\_title are standardized

(49039, 1) (49039,)  
(24155, 1) (24155,)  
(36051, 1) (36051,)

## 5. words\_in\_essay

```
In [41]: # standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

# 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.

words_essay_scaler = StandardScaler(with_mean = False)

# We will fit the train data only
words_essay_scaler.fit(X_train['words_in_essay'].values.reshape(-1,1)) # finding the mean and standard deviation of this
print(f"Mean : {words_essay_scaler.mean_[0]}, Standard deviation : {np.sqrt(words_essay_scaler.var_[0])}")

# Now standardize the data with above mean and variance.
X_train_words_in_essay = words_essay_scaler.transform(X_train['words_in_essay'].values.reshape(-1,1))
X_cv_words_in_essay = words_essay_scaler.transform(X_cv['words_in_essay'].values.reshape(-1,1))
X_test_words_in_essay = words_essay_scaler.transform(X_test['words_in_essay'].values.reshape(-1,1))

print("Words_in_essay are standardized\n")
print(X_train_words_in_essay.shape, y_train.shape)
print(X_cv_words_in_essay.shape, y_cv.shape)
print(X_test_words_in_essay.shape, y_test.shape)
```

Mean : 138.03228042986194, Standard deviation : 36.25008240091709  
Words\_in\_essay are standardized

(49039, 1) (49039,)  
(24155, 1) (24155,)  
(36051, 1) (36051,)

- Sentiment Scores [NEGATIVE : neg, NEUTRAL : neu, POSITIVE : pos, COMPOUND]

## 6. NEGATIVE : neg



In [42]: [# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)

```
# 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.

neg_scalar = StandardScaler(with_mean = False)

# We will fit the train data only
neg_scalar.fit(X_train['neg'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {neg_scalar.mean_[0]}, Standard deviation : {np.sqrt(neg_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
X_train_neg = neg_scalar.transform(X_train['neg'].values.reshape(-1,1))
X_cv_neg = neg_scalar.transform(X_cv['neg'].values.reshape(-1,1))
X_test_neg = neg_scalar.transform(X_test['neg'].values.reshape(-1,1))

print("Sentiment score : NEGATIVE are standardized\n")
print(X_train_neg.shape, y_train.shape)
print(X_cv_neg.shape, y_cv.shape)
print(X_test_neg.shape, y_test.shape)
```

Mean : 0.047992679296070484, Standard deviation : 0.0356901834652085  
Sentiment score : NEGATIVE are standardized

(49039, 1) (49039,)  
(24155, 1) (24155,)  
(36051, 1) (36051,)

## 7. NEUTRAL : neu

In [43]: [# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)

```
# 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.

neu_scalar = StandardScaler(with_mean = False)

# We will fit the train data only
neu_scalar.fit(X_train['neu'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {neu_scalar.mean_[0]}, Standard deviation : {np.sqrt(neu_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
X_train_neu = neu_scalar.transform(X_train['neu'].values.reshape(-1,1))
X_cv_neu = neu_scalar.transform(X_cv['neu'].values.reshape(-1,1))
X_test_neu = neu_scalar.transform(X_test['neu'].values.reshape(-1,1))

print("Sentiment score : NEUTRAL are standardized\n")
print(X_train_neu.shape, y_train.shape)
print(X_cv_neu.shape, y_cv.shape)
print(X_test_neu.shape, y_test.shape)
```

Mean : 0.6707143701951507, Standard deviation : 0.0758049359008078  
Sentiment score : NEUTRAL are standardized

(49039, 1) (49039,)  
(24155, 1) (24155,)  
(36051, 1) (36051,)

## 8. POSITIVE : pos

In [44]: [# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)

```
# 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.

pos_scalar = StandardScaler(with_mean = False)

# We will fit the train data only
pos_scalar.fit(X_train['pos'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {pos_scalar.mean_[0]}, Standard deviation : {np.sqrt(pos_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
X_train_pos = pos_scalar.transform(X_train['pos'].values.reshape(-1,1))
X_cv_pos = pos_scalar.transform(X_cv['pos'].values.reshape(-1,1))
X_test_pos = pos_scalar.transform(X_test['pos'].values.reshape(-1,1))

print("Sentiment score : POSITIVE are standardized\n")
print(X_train_pos.shape, y_train.shape)
print(X_cv_pos.shape, y_cv.shape)
print(X_test_pos.shape, y_test.shape)
```

Mean : 0.28129221639919255, Standard deviation : 0.07797150498229787  
Sentiment score : POSITIVE are standardized

(49039, 1) (49039,)  
(24155, 1) (24155,)  
(36051, 1) (36051,)

## 9. COMPOUND

In [45]: [# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html)

```
# 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.

compound_scalar = StandardScaler(with_mean = False)

# We will fit the train data only
compound_scalar.fit(X_train['compound'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {compound_scalar.mean_[0]}, Standard deviation : {np.sqrt(compound_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
X_train_compound = compound_scalar.transform(X_train['compound'].values.reshape(-1,1))
X_cv_compound = compound_scalar.transform(X_cv['compound'].values.reshape(-1,1))
X_test_compound = compound_scalar.transform(X_test['compound'].values.reshape(-1,1))

print("Sentiment score : COMPOUND are standardized\n")
print(X_train_compound.shape, y_train.shape)
print(X_cv_compound.shape, y_cv.shape)
print(X_test_compound.shape, y_test.shape)
```

Mean : 0.9588067334162605, Standard deviation : 0.14995781122456714  
Sentiment score : COMPOUND are standardized

(49039, 1) (49039,)  
(24155, 1) (24155,)  
(36051, 1) (36051,)

## 5. Make Data Model Ready: encoding eassay, and project\_title

### BOW

#### 1. clean\_essay



```
In [48]: # Vectorizing the project_title column

# We are considering only the words which appeared in at least 10 documents(rows or projects).
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
# Taking top 5000 features

# Creating the vectorizer with bi-grams
vectorizer_bow_title = CountVectorizer(min_df=10, ngram_range=(2,2), max_features=5000)

# We will fit the train data only
vectorizer_bow_title.fit(X_train['clean_titles'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_bow = vectorizer_bow_title.transform(X_train['clean_titles'].values)
X_cv_titles_bow = vectorizer_bow_title.transform(X_cv['clean_titles'].values)
X_test_titles_bow = vectorizer_bow_title.transform(X_test['clean_titles'].values)

print("Project Titles vectorized")
print(X_train_titles_bow.shape, y_train.shape)
print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)
print(vectorizer_bow_title.get_feature_names())
```

```
Project Titles vectorized
(49039, 1706) (49039,)
(24155, 1706) (24155,)
(36051, 1706) (36051,)
```

## Finding the vocabulary

```
In [49]: # Getting the vocabulary
# https://stackoverflow.com/questions/52141785/sort-dict-by-values-in-python-3-6
vocab = vectorizer_bow_title.vocabulary_

vocab_sorted = {k: v for k, v in sorted(vocab.items(), key=lambda x: x[1], reverse=True)}

# vocabularies as per their indexes
vocab_sorted
```

```
Out[49]: {'your way': 1705,
'your seat': 1704,
'your own': 1703,
'your help': 1702,
'your feet': 1701,
'your brain': 1700,
'young scientists': 1699,
'young readers': 1698,
'young minds': 1697,
'young learners': 1696,
'young artists': 1695,
'you work': 1694,
'you will': 1693,
'you see': 1692,
'you read': 1691,
'you learn': 1690,
'you know': 1689,
'you help': 1688,
'you hear': 1687,
, , , 1686,
```

## TF-IDF

### 1. clean\_essay

```
Essay vectorized
(49039, 5000) (49039,)
(24155, 5000) (24155,)
(36051, 5000) (36051,)
```

## Finding the vocabulary

```
# vocabuaries as per their indexes
vocab sorted
```

```
'years students': 49
```

## 2. clean titles





◀ [REDACTED]

300

```
=====
=
Wall time: 41.5 s
```

```
In [47]: # Changing list to numpy arrays
train_w2v_vectors_essays = np.array(train_w2v_vectors_essays)
test_w2v_vectors_essays = np.array(test_w2v_vectors_essays)
cv_w2v_vectors_essays = np.array(cv_w2v_vectors_essays)

print("Essay vectorized")
print(train_w2v_vectors_essays.shape, y_train.shape)
print(cv_w2v_vectors_essays.shape, y_cv.shape)
print(test_w2v_vectors_essays.shape, y_test.shape)
```

```
Essay vectorized
(49039, 300) (49039,)
(24155, 300) (24155,)
(36051, 300) (36051,)
```

## 2. clean\_titles

```
100%|██████████████████████████████████████████████████████████████████████████| 49039/49039 [00:00<00:00, 51449.93it/s]
Train vector
49039
300
=====
=

100%|██████████████████████████████████████████████████████████████████████████| 36051/36051 [00:00<00:00, 50224.20it/s]
Test vec
36051
300
=====
=

100%|██████████████████████████████████████████████████████████████████████████| 24155/24155 [00:00<00:00, 49567.32it/s]
CV vec
24155
300
=====
=
Wall time: 2.22 s
```

```
In [49]: # Changing list to numpy arrays
train_w2v_vectors_titles = np.array(train_w2v_vectors_titles)
test_w2v_vectors_titles = np.array(test_w2v_vectors_titles)
cv_w2v_vectors_titles = np.array(cv_w2v_vectors_titles)

print("Title vectorized")
print(train_w2v_vectors_titles.shape, y_train.shape)
print(cv_w2v_vectors_titles.shape, y_cv.shape)
print(test_w2v_vectors_titles.shape, y_test.shape)
```

```
Title vectorized
(49039, 300) (49039,)
(24155, 300) (24155,)
(36051, 300) (36051,)
```

## TF-IDF weighted W2V

### 1. clean\_essay



```
Train matrix:
49039
300
=====
=
```

```
CV matrix:
24155
300
=====
=
```

```
Test matrix:
36051
300
=====
=
```

```
In [47]: # Changing list to numpy arrays
train_tfidf_w2v_essays = np.array(train_tfidf_w2v_essays)
test_tfidf_w2v_essays = np.array(test_tfidf_w2v_essays)
cv_tfidf_w2v_essays = np.array(cv_tfidf_w2v_essays)

print("Essay vectorized")
print(train_tfidf_w2v_essays.shape, y_train.shape)
print(cv_tfidf_w2v_essays.shape, y_cv.shape)
print(test_tfidf_w2v_essays.shape, y_test.shape)
```

## 2. clean\_titles

```
In [48]: %%time
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['clean_titles'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_tfidf_w2v_titles.append(vector)

print("Train matrix:")
print(len(train_tfidf_w2v_titles))
print(len(train_tfidf_w2v_titles[0]))
print('='*120)

cv_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_tfidf_w2v_titles.append(vector)

print("CV matrix:")
print(len(cv_tfidf_w2v_titles))
print(len(cv_tfidf_w2v_titles[0]))
print('='*120)

test_tfidf_w2v_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight = 0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_titles.append(vector)

print("Test matrix:")
print(len(test_tfidf_w2v_titles))
print(len(test_tfidf_w2v_titles[0]))
print('='*120)
```

[illegible]

Train matrix:

49039

300

$$=$$
$$=$$
[illegible]

CV matrix:

24155

300

$$=$$
$$=$$

[illegible]

Test matrix:

36051

300

---

$$=$$

Wall time: 5.13 s

```
In [49]: # Changing list to numpy arrays
train_tfidf_w2v_titles = np.array(train_tfidf_w2v_titles)
test_tfidf_w2v_titles = np.array(test_tfidf_w2v_titles)
cv_tfidf_w2v_titles = np.array(cv_tfidf_w2v_titles)

print("Title vectorized")
print(train_tfidf_w2v_titles.shape, y_train.shape)
print(cv_tfidf_w2v_titles.shape, y_cv.shape)
print(test_tfidf_w2v_titles.shape, y_test.shape)
```

```
Title vectorized
(49039, 300) (49039,)
(24155, 300) (24155,)
(36051, 300) (36051,)
```

### [Task-1] Applying Logistic Regression on different kind of featurization

## 1. Applying Logistic Regression on BOW, SET 1

## Merging the categorical, numerical and text features

```
In [50]: # merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# https://stackoverflow.com/questions/54226138/constructing-sparse-csr-matrix-directly-vs-using-coo-tocsr-scipy
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)

from scipy.sparse import hstack

# Training data
X_tr = hstack((X_train_essay_bow, X_train_titles_bow, X_train_clean_category, X_train_clean_subcategories,
               X_train_project_grade, X_train_school_state, X_train_teacher_prefix,
               X_train_previous_projects, X_train_price, X_train_quantity)).tocsr()

# CV data
X_cr = hstack((X_cv_essay_bow, X_cv_titles_bow, X_cv_clean_category, X_cv_clean_subcategories,
               X_cv_project_grade, X_cv_school_state, X_cv_teacher_prefix,
               X_cv_previous_projects, X_cv_price, X_cv_quantity)).tocsr()

# Test data
X_te = hstack((X_test_essay_bow, X_test_titles_bow, X_test_clean_category, X_test_clean_subcategories,
               X_test_project_grade, X_test_school_state, X_test_teacher_prefix,
               X_test_previous_projects, X_test_price, X_test_quantity)).tocsr()
```

```
In [51]: ## Print the final data matrix

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

```
Final Data matrix
(49039, 6808) (49039,)
(24155, 6808) (24155,)
(36051, 6808) (36051,)
```

### Dimensions of the hstacked features

```
In [52]: print('Training DATA\n')
print('ESSAY : ', X_train_essay_bow.shape)
print('Title : ', X_train_titles_bow.shape)
print('Categorical Data : ', (X_train_clean_category.shape + X_train_clean_subcategories.shape + X_train_project_grade.s
                             X_train_school_state.shape + X_train_teacher_prefix.shape))
print('Numerical Data : ', (X_train_previous_projects.shape + X_train_price.shape + X_train_quantity.shape))

print('\n','='*120)

print('CV DATA\n')
print('ESSAY : ', X_cv_essay_bow.shape)
print('Title : ', X_cv_titles_bow.shape)
print('Categorical Data : ', (X_cv_clean_category.shape + X_cv_clean_subcategories.shape + X_cv_project_grade.shape + \
                             X_cv_school_state.shape + X_cv_teacher_prefix.shape))
print('Numerical Data : ', (X_cv_previous_projects.shape + X_cv_price.shape + X_cv_quantity.shape))

print('\n','='*120)

print('Test DATA\n')
print('ESSAY : ', X_test_essay_bow.shape)
print('Title : ', X_test_titles_bow.shape)
print('Categorical Data : ', (X_test_clean_category.shape + X_test_clean_subcategories.shape + X_test_project_grade.shap
                             X_test_school_state.shape + X_test_teacher_prefix.shape))
print('Numerical Data : ', (X_test_previous_projects.shape + X_test_price.shape + X_test_quantity.shape))

print('\n','='*120)
```

Training DATA

```
ESSAY : (49039, 5000)
Title : (49039, 1706)
Categorical Data : (49039, 9, 49039, 30, 49039, 4, 49039, 51, 49039, 5)
Numerical Data : (49039, 1, 49039, 1, 49039, 1)
```

```
=====
==
CV DATA
```

```
ESSAY : (24155, 5000)
Title : (24155, 1706)
Categorical Data : (24155, 9, 24155, 30, 24155, 4, 24155, 51, 24155, 5)
Numerical Data : (24155, 1, 24155, 1, 24155, 1)
```

```
=====
==
Test DATA
```

```
ESSAY : (36051, 5000)
Title : (36051, 1706)
Categorical Data : (36051, 9, 36051, 30, 36051, 4, 36051, 51, 36051, 5)
Numerical Data : (36051, 1, 36051, 1, 36051, 1)
```

```
=====
==
```

## Hyper paramter tuning to find best $\alpha$ (alpha) (Using GridSearchCV)



```

In [55]: %%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# https://stackoverflow.com/questions/52640386/how-do-i-solve-the-future-warning-min-groups-self-n-splits-warning-in
# https://stackoverflow.com/questions/48643181/please-what-is-the-meaning-of-the-deprecation-warning-message

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc

# creating logistic regression classifier using SGDClassifier
classifier = SGDClassifier(loss='log', max_iter = 100000, tol = 1e-3)

# Lambda values
parameters = {'alpha':[0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001, 0.000001]}

# Finding the best parameter using gridsearchcv and 10-folds
clf = GridSearchCV(classifier, parameters, cv=10, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr, y_train)

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

# We use log(alpha) values so as to get a more distinguishable graph because log is monotonous function
# and it won't affect our results

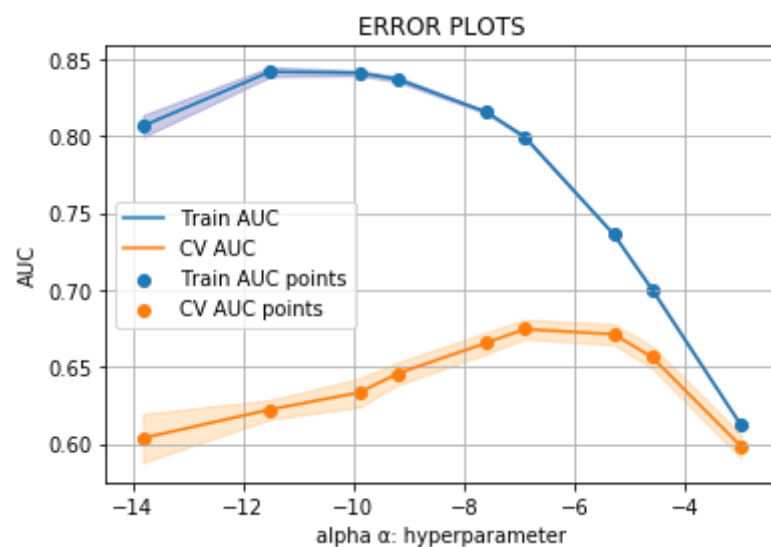
plt.plot(np.log(parameters['alpha']), train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(np.log(parameters['alpha']), train_auc - train_auc_std, train_auc + train_auc_std, alpha=0.2, color='blue')

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

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

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

```



Wall time: 1min 52s

### Now creating the model with best $\alpha$

```

In [56]: # From the error plot we choose α such that, we will have maximum AUC on cv data and gap between the train and cv is less
# Here we are choosing the best_α based on GridSearchCV results

best_α = 0.001

```

```
In [57]: %%time
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.linear_model import SGDClassifier

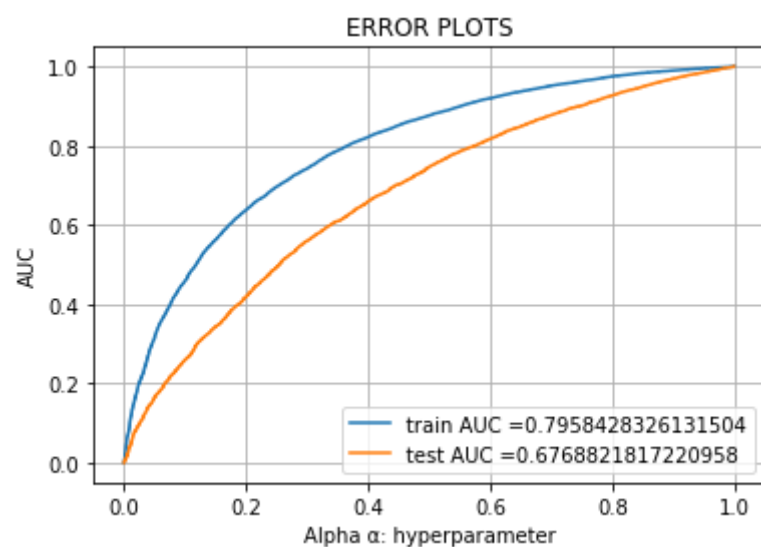
# Creating the classifier with best  $\alpha$ 
classifier = SGDClassifier(loss='log', alpha = best_α, max_iter = 1000, tol = 1e-3)
classifier.fit(X_tr, y_train)

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

# Getting the predicted probability scores for test and train values
y_train_pred = classifier.predict_proba(X_tr)[:,-1]
y_test_pred = classifier.predict_proba(X_te)[:,-1]

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

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Alpha  $\alpha$ : hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Wall time: 450 ms

## NOTE:

- As we can see from the graph. The AUC curve is lower for the test set than the train set.
- The AUC scores for the Train and Test data are : 79% and 67% respectively
- We choose the  $\alpha$  value equal to 0.001 because it has maximum AUC on the CV data

```
In [58]: # we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    #print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [59]: [# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification\\_report.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)

```
from sklearn.metrics import classification_report

y_pred_new = classifier.predict(X_te)
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred_new, target_names = target_names))
```

	precision	recall	f1-score	support
class 0	0.40	0.02	0.04	5459
class 1	0.85	0.99	0.92	30592
micro avg	0.85	0.85	0.85	36051
macro avg	0.63	0.51	0.48	36051
weighted avg	0.78	0.85	0.78	36051

In [60]: %%time  
[# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics)

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = classifier.predict(X_te)
print("Accuracy on test set: {}".format(accuracy_score(y_test, y_pred_new)))
print("Precision on test set: {}".format(precision_score(y_test, y_pred_new)))
print("Recall on test set: {}".format(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: {}".format(f1_score(y_test, y_pred_new)))
```

Accuracy on test set: 0.8472164433718898  
Precision on test set: 0.8503939207688439  
Recall on test set: 0.9949986924686193  
F1-Score on test set: 0.9170306992438165  
Wall time: 39.9 ms

In [61]: **from** sklearn.metrics **import** confusion\_matrix

```
print("="*120)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*120)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
=====
=
Train confusion matrix
[[ 5365  2060]
 [11474 30140]]
=====
=
Test confusion matrix
[[ 3247  2212]
 [10277 20315]]
```

## Function to create the confusion matrix

In [62]: [# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix](https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix)

```
def get_confusion_matrix(y_tr_ts, y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr):

    """Function to get heatmap confusion matrix"""

    # Creating the confusion matrix dataframe
    df_cm = pd.DataFrame(confusion_matrix(y_tr_ts, predict(y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr))
                        , range(2),range(2))

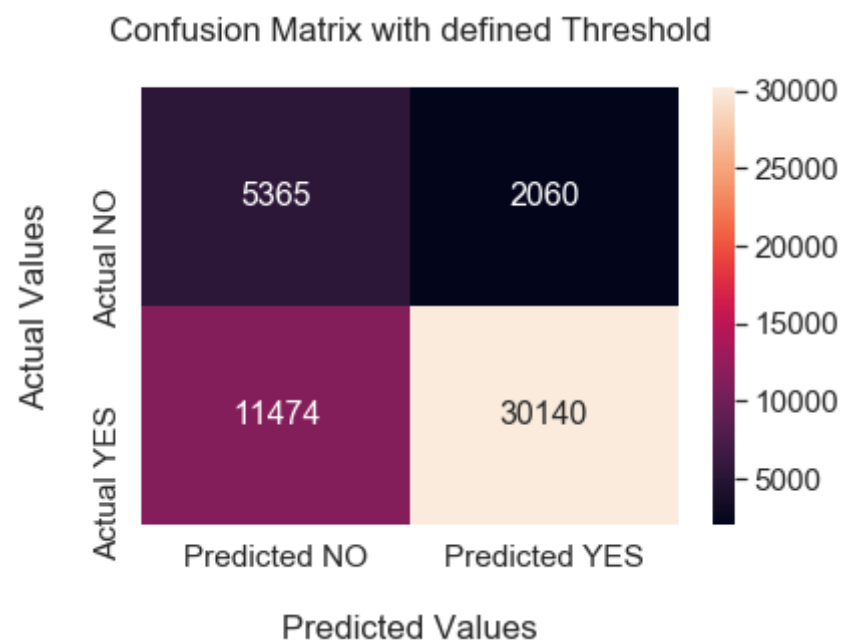
    df_cm.columns = ['Predicted NO', 'Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})

    plt.title('Confusion Matrix with defined Threshold\n ')

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

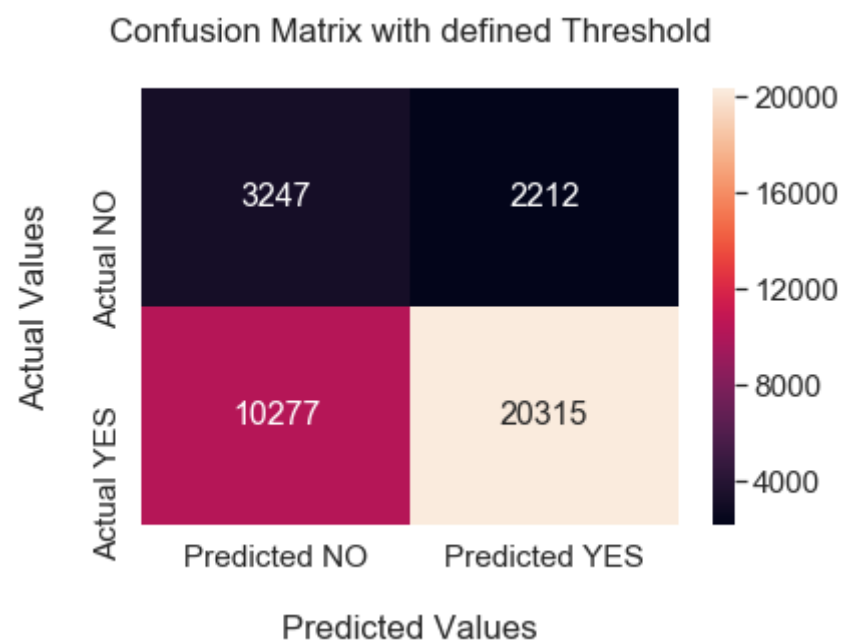
## Confusion Matrix on train data

```
In [64]: get_confusion_matrix(y_train, y_train_pred, tr_thresholds, train_fpr, train_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



### Confusion Matrix on test data

```
In [65]: get_confusion_matrix(y_test, y_test_pred, tr_thresholds, test_fpr, test_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



### NOTE:

1. The model predicts the test set correctly with a AUC score of 67%
2. The F1\_score obtained is 0.9170306992438165
3. The alpha value that we got after GridSearchCV is 0.001

## 2. Applying Logistic Regression on TF-IDF, SET 2

### Merging the categorical, numerical and text features

```
In [50]: # merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# https://stackoverflow.com/questions/54226138/constructing-sparse-csr-matrix-directly-vs-using-coo-tocsr-scipy
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)

from scipy.sparse import hstack

# Training data
X_tr = hstack((X_train_essay_tfidf, X_train_titles_tfidf, X_train_clean_category, X_train_clean_subcategories,
               X_train_project_grade, X_train_school_state, X_train_teacher_prefix,
               X_train_previous_projects, X_train_price, X_train_quantity)).tocsr()

# CV data
X_cr = hstack((X_cv_essay_tfidf, X_cv_titles_tfidf, X_cv_clean_category, X_cv_clean_subcategories,
               X_cv_project_grade, X_cv_school_state, X_cv_teacher_prefix,
               X_cv_previous_projects, X_cv_price, X_cv_quantity)).tocsr()

# Test data
X_te = hstack((X_test_essay_tfidf, X_test_titles_tfidf, X_test_clean_category, X_test_clean_subcategories,
               X_test_project_grade, X_test_school_state, X_test_teacher_prefix,
               X_test_previous_projects, X_test_price, X_test_quantity)).tocsr()

## Print the final data matrix

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

```
Final Data matrix
(49039, 6766) (49039,)
(24155, 6766) (24155,)
(36051, 6766) (36051,)
```

## Dimensions of the hstacked features

```
In [51]: print('Training DATA\n')
print('ESSAY : ', X_train_essay_tfidf.shape)
print('Title : ', X_train_titles_tfidf.shape)
print('Categorical Data : ', (X_train_clean_category.shape + X_train_clean_subcategories.shape + X_train_project_grade.s
                             X_train_school_state.shape + X_train_teacher_prefix.shape))
print('Numerical Data : ', (X_train_previous_projects.shape + X_train_price.shape + X_train_quantity.shape))

print('\n','='*120)

print('CV DATA\n')
print('ESSAY : ', X_cv_essay_tfidf.shape)
print('Title : ', X_cv_titles_tfidf.shape)
print('Categorical Data : ', (X_cv_clean_category.shape + X_cv_clean_subcategories.shape + X_cv_project_grade.shape + \
                             X_cv_school_state.shape + X_cv_teacher_prefix.shape))
print('Numerical Data : ', (X_cv_previous_projects.shape + X_cv_price.shape + X_cv_quantity.shape))

print('\n','='*120)

print('Test DATA\n')
print('ESSAY : ', X_test_essay_tfidf.shape)
print('Title : ', X_test_titles_tfidf.shape)
print('Categorical Data : ', (X_test_clean_category.shape + X_test_clean_subcategories.shape + X_test_project_grade.shap
                             X_test_school_state.shape + X_test_teacher_prefix.shape))
print('Numerical Data : ', (X_test_previous_projects.shape + X_test_price.shape + X_test_quantity.shape))

print('\n','='*120)
```

#### Training DATA

```
ESSAY : (49039, 5000)
Title : (49039, 1664)
Categorical Data : (49039, 9, 49039, 30, 49039, 4, 49039, 51, 49039, 5)
Numerical Data : (49039, 1, 49039, 1, 49039, 1)
```

```
=====
==
CV DATA
```

```
ESSAY : (24155, 5000)
Title : (24155, 1664)
Categorical Data : (24155, 9, 24155, 30, 24155, 4, 24155, 51, 24155, 5)
Numerical Data : (24155, 1, 24155, 1, 24155, 1)
```

```
=====
==
Test DATA
```

```
ESSAY : (36051, 5000)
Title : (36051, 1664)
Categorical Data : (36051, 9, 36051, 30, 36051, 4, 36051, 51, 36051, 5)
Numerical Data : (36051, 1, 36051, 1, 36051, 1)
```

```
=====
==
```

## Hyper paramter tuning to find best $\alpha$ (alpha) (Using GridSearchCV)



```

In [52]: %%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# https://stackoverflow.com/questions/52640386/how-do-i-solve-the-future-warning-min-groups-self-n-splits-warning-in
# https://stackoverflow.com/questions/48643181/please-what-is-the-meaning-of-the-deprecation-warning-message

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc

# creating logistic regression classifier
classifier = SGDClassifier(loss='log', max_iter = 100000, tol = 1e-3)

# Lambda values
parameters = {'alpha':[0.001, 0.0005, 0.0001, 0.00005, 0.00001, 0.000001]}

# Finding the best parameter using gridsearchcv and 10-folds
clf = GridSearchCV(classifier, parameters, cv=10, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr, y_train)

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

# We use log(alpha) values so as to get a more distinguishable graph because log is monotonous function
# and it won't affect our results

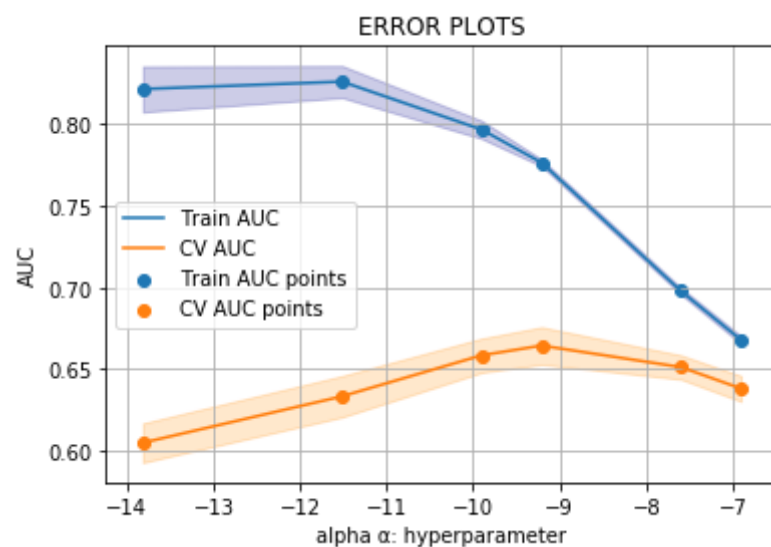
plt.plot(np.log(parameters['alpha']), train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(np.log(parameters['alpha']), train_auc - train_auc_std, train_auc + train_auc_std, alpha=0.2, color='blue')

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

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

plt.legend()
plt.xlabel("alpha  $\alpha$ : hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()

```



Wall time: 47.5 s

### Now creating the model with best $\alpha$

```

In [53]: # From the error plot we choose  $\alpha$  such that, we will have maximum AUC on cv data and gap between the train and cv is less
# Here we are choosing the best_α based on GridSearchCV results

best_α = 0.0001

```

```
In [54]: %%time
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.linear_model import SGDClassifier

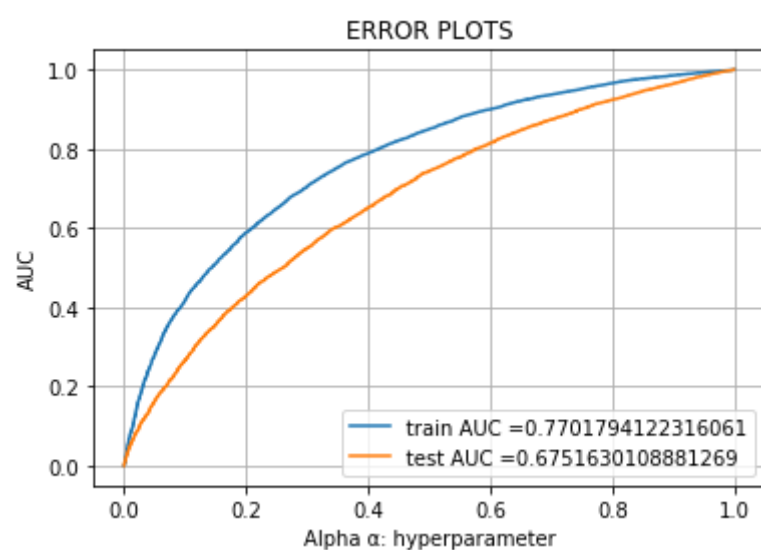
# Creating the classifier with best  $\alpha$ 
classifier = SGDClassifier(loss='log', alpha = best_α, max_iter = 100000, tol = 1e-3)
classifier.fit(X_tr, y_train)

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

# Getting the predicted probability scores for test and train values
y_train_pred = classifier.predict_proba(X_tr)[:,-1]
y_test_pred = classifier.predict_proba(X_te)[:,-1]

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

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Alpha  $\alpha$ : hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Wall time: 517 ms

## NOTE:

- As we can see from the graph. The AUC curve is lower for the test set than the train set.
- The AUC scores for the Train and Test data are : 77% and 67% respectively
- We choose the  $\alpha$  value equal to 0.0001 because it has maximum AUC on the CV data

```
In [55]: # we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    # print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i >= t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [57]: [# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification\\_report.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)

```
from sklearn.metrics import classification_report

y_pred_new = classifier.predict(X_te)
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred_new, target_names = target_names))
```

	precision	recall	f1-score	support
class 0	0.35	0.01	0.01	5459
class 1	0.85	1.00	0.92	30592
micro avg	0.85	0.85	0.85	36051
macro avg	0.60	0.50	0.47	36051
weighted avg	0.77	0.85	0.78	36051

In [58]: %%time  
[# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics)

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = classifier.predict(X_te)
print("Accuracy on test set: {}".format(accuracy_score(y_test, y_pred_new)))
print("Precision on test set: {}".format(precision_score(y_test, y_pred_new)))
print("Recall on test set: {}".format(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: {}".format(f1_score(y_test, y_pred_new)))
```

Accuracy on test set: 0.8476325205958226  
 Precision on test set: 0.8492083368116427  
 Recall on test set: 0.9975810669456067  
 F1-Score on test set: 0.9174345022471404  
 Wall time: 41.9 ms

In [59]: **from** sklearn.metrics **import** confusion\_matrix

```
print("="*120)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*120)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
=====
=
Train confusion matrix
[[ 5166  2259]
 [12056 29558]]
=====
=
Test confusion matrix
[[ 3488  1971]
 [11843 18749]]
```

## Function to create the confusion matrix

In [60]: [# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix](https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix)

```
def get_confusion_matrix(y_tr_ts, y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr):

    """Function to get heatmap confusion matrix"""

    # Creating the confusion matrix dataframe
    df_cm = pd.DataFrame(confusion_matrix(y_tr_ts, predict(y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr))
                        , range(2),range(2))

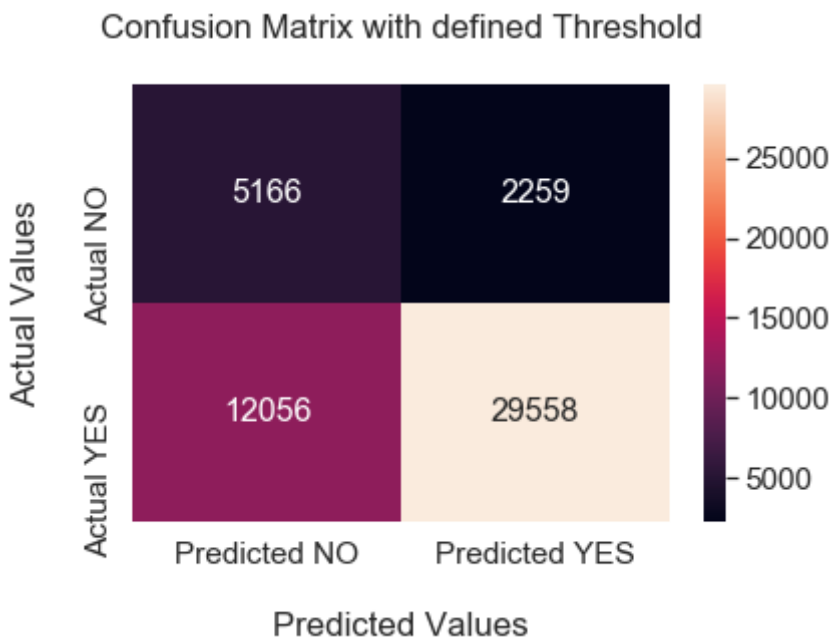
    df_cm.columns = ['Predicted NO', 'Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})

    plt.title('Confusion Matrix with defined Threshold\n ')

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

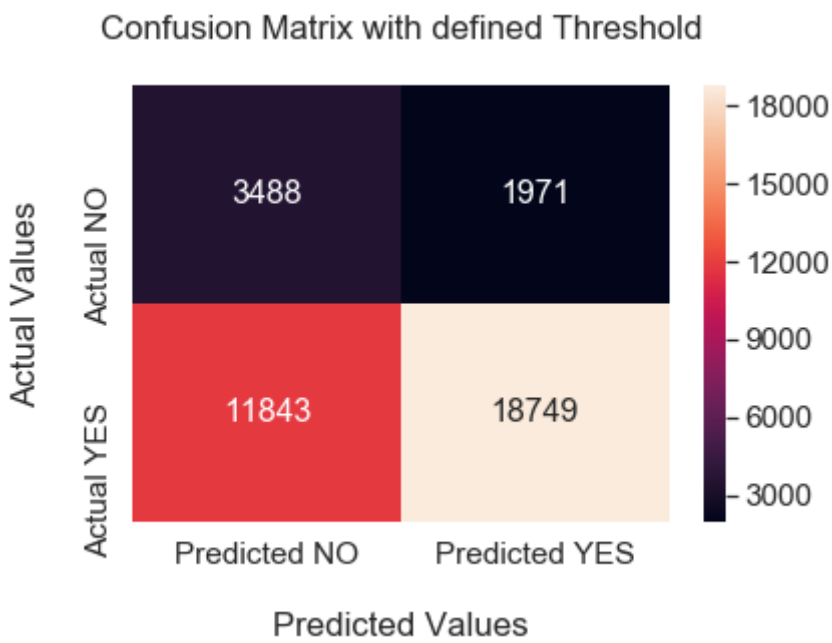
## Confusion Matrix on train data

```
In [62]: get_confusion_matrix(y_train, y_train_pred, tr_thresholds, train_fpr, train_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



Confusion Matrix on test data

```
In [63]: get_confusion_matrix(y_test, y_test_pred, tr_thresholds, test_fpr, test_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



NOTE:

- 1. The model predicts the test set correctly with a AUC score of 67%
- 2. The F1\_score obtained is 0.9174345022471404
- 3. The alpha value that we got after GridSearchCV is 0.0001

3. Applying Logistic Regression on AVG W2V, SET 3

Merging the categorical, numerical and text features

```
In [50]: # merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# https://stackoverflow.com/questions/54226138/constructing-sparse-csr-matrix-directly-vs-using-coo-tocsr-scipy
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)

from scipy.sparse import hstack

# Training data
X_tr = hstack((train_w2v_vectors_essays, train_w2v_vectors_titles, X_train_clean_category, X_train_clean_subcategories,
              X_train_project_grade, X_train_school_state, X_train_teacher_prefix,
              X_train_previous_projects, X_train_price, X_train_quantity)).tocsr()

# CV data
X_cr = hstack((cv_w2v_vectors_essays, cv_w2v_vectors_titles, X_cv_clean_category, X_cv_clean_subcategories,
              X_cv_project_grade, X_cv_school_state, X_cv_teacher_prefix,
              X_cv_previous_projects, X_cv_price, X_cv_quantity)).tocsr()

# Test data
X_te = hstack((test_w2v_vectors_essays, test_w2v_vectors_titles, X_test_clean_category, X_test_clean_subcategories,
              X_test_project_grade, X_test_school_state, X_test_teacher_prefix,
              X_test_previous_projects, X_test_price, X_test_quantity)).tocsr()

## Print the final data matrix

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

```
Final Data matrix
(49039, 702) (49039,)
(24155, 702) (24155,)
(36051, 702) (36051,)
```

## Dimensions of the hstacked features

```
In [51]: print('Training DATA\n')
print('ESSAY : ', train_w2v_vectors_essays.shape)
print('Title : ', train_w2v_vectors_titles.shape)
print('Categorical Data : ', (X_train_clean_category.shape + X_train_clean_subcategories.shape + X_train_project_grade.s
                             X_train_school_state.shape + X_train_teacher_prefix.shape))
print('Numerical Data : ', (X_train_previous_projects.shape + X_train_price.shape + X_train_quantity.shape))

print('\n','='*120)

print('CV DATA\n')
print('ESSAY : ', cv_w2v_vectors_essays.shape)
print('Title : ', cv_w2v_vectors_titles.shape)
print('Categorical Data : ', (X_cv_clean_category.shape + X_cv_clean_subcategories.shape + X_cv_project_grade.shape + \
                             X_cv_school_state.shape + X_cv_teacher_prefix.shape))
print('Numerical Data : ', (X_cv_previous_projects.shape + X_cv_price.shape + X_cv_quantity.shape))

print('\n','='*120)

print('Test DATA\n')
print('ESSAY : ', test_w2v_vectors_essays.shape)
print('Title : ', test_w2v_vectors_titles.shape)
print('Categorical Data : ', (X_test_clean_category.shape + X_test_clean_subcategories.shape + X_test_project_grade.shap
                             X_test_school_state.shape + X_test_teacher_prefix.shape))
print('Numerical Data : ', (X_test_previous_projects.shape + X_test_price.shape + X_test_quantity.shape))

print('\n','='*120)
```

#### Training DATA

```
ESSAY : (49039, 300)
Title : (49039, 300)
Categorical Data : (49039, 9, 49039, 30, 49039, 4, 49039, 51, 49039, 5)
Numerical Data : (49039, 1, 49039, 1, 49039, 1)
```

```
=====
==
CV DATA
```

```
ESSAY : (24155, 300)
Title : (24155, 300)
Categorical Data : (24155, 9, 24155, 30, 24155, 4, 24155, 51, 24155, 5)
Numerical Data : (24155, 1, 24155, 1, 24155, 1)
```

```
=====
==
Test DATA
```

```
ESSAY : (36051, 300)
Title : (36051, 300)
Categorical Data : (36051, 9, 36051, 30, 36051, 4, 36051, 51, 36051, 5)
Numerical Data : (36051, 1, 36051, 1, 36051, 1)
```

```
=====
==
```

## Hyper paramter tuning to find best $\alpha$ (alpha) (Using GridSearchCV)



```

In [52]: %%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# https://stackoverflow.com/questions/52640386/how-do-i-solve-the-future-warning-min-groups-self-n-splits-warning-in
# https://stackoverflow.com/questions/48643181/please-what-is-the-meaning-of-the-deprecation-warning-message

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc

# creating logistic regression classifier
classifier = SGDClassifier(loss='log', max_iter = 100000, tol = 1e-3)

# Lambda values
parameters = {'alpha':[0.001, 0.0005, 0.0001, 0.00005, 0.00001, 0.000005]}

# Finding the best parameter using gridsearchcv and 10-folds
clf = GridSearchCV(classifier, parameters, cv=10, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr, y_train)

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

# We use log(alpha) values so as to get a more distinguishable graph because log is monotonous function
# and it won't affect our results

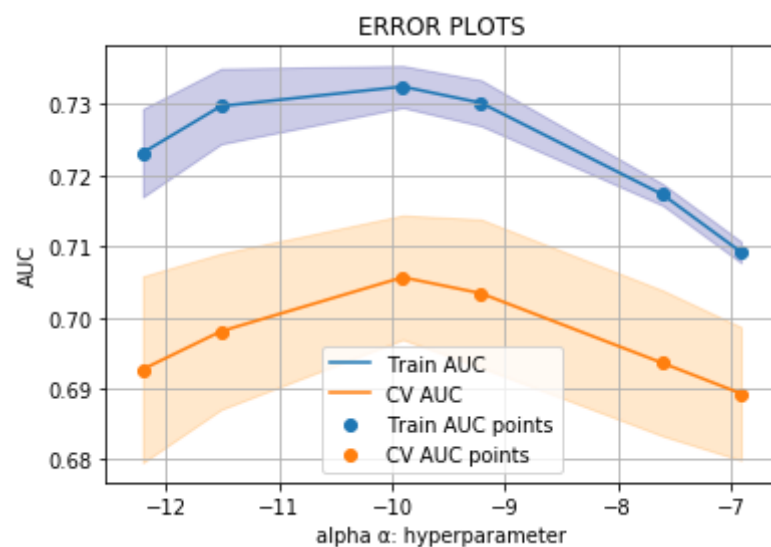
plt.plot(np.log(parameters['alpha']), train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(np.log(parameters['alpha']), train_auc - train_auc_std, train_auc + train_auc_std, alpha=0.2, color='blue')

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

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

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

```



Wall time: 6min 16s

### Now creating the model with best $\alpha$

```

In [53]: # From the error plot we choose α such that, we will have maximum AUC on cv data and gap between the train and cv is less
# Here we are choosing the best_α based on GridSearchCV results

best_α = 0.0001

```

```
In [54]: %%time
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.linear_model import SGDClassifier

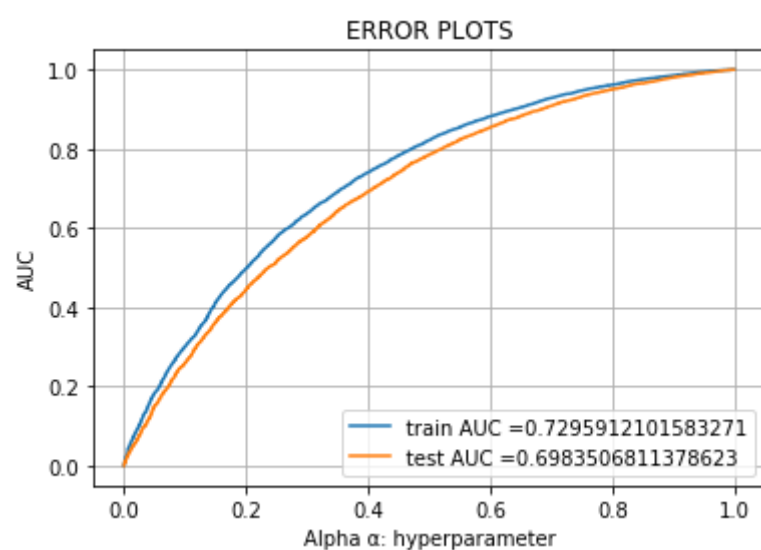
# Creating the classifier with best  $\alpha$ 
classifier = SGDClassifier(loss='log', alpha = best_α, max_iter = 100000, tol = 1e-3)
classifier.fit(X_tr, y_train)

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

# Getting the predicted probability scores for test and train values
y_train_pred = classifier.predict_proba(X_tr)[:,-1]
y_test_pred = classifier.predict_proba(X_te)[:,-1]

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

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Alpha  $\alpha$ : hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Wall time: 3.91 s

## NOTE:

- As we can see from the graph. The AUC curve is lower for the test set than the train set.
- The AUC scores for the Train and Test data are : 72% and 70% respectively
- We choose the  $\alpha$  value equal to 0.0001 because it has maximum AUC on the CV data

```
In [55]: # we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    # print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i >= t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [56]: [# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification\\_report.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)

```
from sklearn.metrics import classification_report

y_pred_new = classifier.predict(X_te)
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred_new, target_names = target_names))
```

	precision	recall	f1-score	support
class 0	0.49	0.07	0.12	5459
class 1	0.86	0.99	0.92	30592
micro avg	0.85	0.85	0.85	36051
macro avg	0.68	0.53	0.52	36051
weighted avg	0.80	0.85	0.80	36051

In [57]: `%%time`  
[# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics)

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = classifier.predict(X_te)
print("Accuracy on test set: {}".format(accuracy_score(y_test, y_pred_new)))
print("Precision on test set: {}".format(precision_score(y_test, y_pred_new)))
print("Recall on test set: {}".format(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: {}".format(f1_score(y_test, y_pred_new)))
```

Accuracy on test set: 0.848353721117306  
Precision on test set: 0.8555659335993886  
Recall on test set: 0.9881014644351465  
F1-Score on test set: 0.9170699149007174  
Wall time: 106 ms

In [58]: `from sklearn.metrics import confusion_matrix`

```
print("="*120)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*120)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
=====
=
Train confusion matrix
[[ 4784  2641]
 [12533 29081]]
=====
=
Test confusion matrix
[[ 3778  1681]
 [12586 18006]]
```

## Function to create the confusion matrix

In [59]: [# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix](https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix)

```
def get_confusion_matrix(y_tr_ts, y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr):

    """Function to get heatmap confusion matrix"""

    # Creating the confusion matrix dataframe
    df_cm = pd.DataFrame(confusion_matrix(y_tr_ts, predict(y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr))
                        , range(2),range(2))

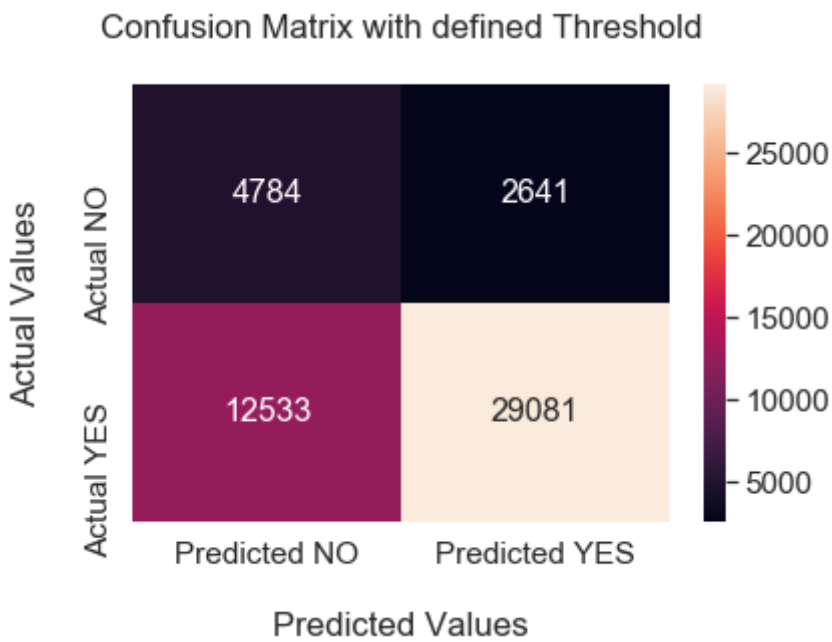
    df_cm.columns = ['Predicted NO', 'Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})

    plt.title('Confusion Matrix with defined Threshold\n ')

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

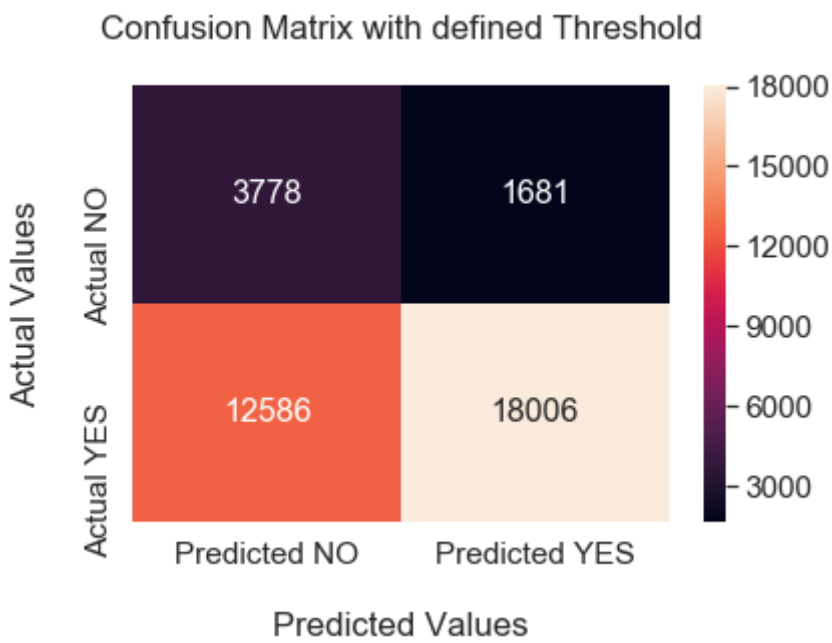
## Confusion Matrix on train data

```
In [61]: get_confusion_matrix(y_train, y_train_pred, tr_thresholds, train_fpr, train_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



Confusion Matrix on test data

```
In [62]: get_confusion_matrix(y_test, y_test_pred, tr_thresholds, test_fpr, test_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



NOTE:

- 1. The model predicts the test set correctly with a AUC score of 70%
- 2. The F1\_score obtained is 0.9170699149007174
- 3. The alpha value that we got after GridSearchCV is 0.0001

4. Applying Logistic Regression on TFIDF W2V, SET 4

Merging the categorical, numerical and text features

```
In [50]: # merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# https://stackoverflow.com/questions/54226138/constructing-sparse-csr-matrix-directly-vs-using-coo-to-csr-scipy
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)

from scipy.sparse import hstack

# Training data
X_tr = hstack((train_tfidf_w2v_essays, train_tfidf_w2v_titles, X_train_clean_category, X_train_clean_subcategories,
               X_train_project_grade, X_train_school_state, X_train_teacher_prefix,
               X_train_previous_projects, X_train_price, X_train_quantity)).tocsr()

# CV data
X_cr = hstack((cv_tfidf_w2v_essays, cv_tfidf_w2v_titles, X_cv_clean_category, X_cv_clean_subcategories,
               X_cv_project_grade, X_cv_school_state, X_cv_teacher_prefix,
               X_cv_previous_projects, X_cv_price, X_cv_quantity)).tocsr()

# Test data
X_te = hstack((test_tfidf_w2v_essays, test_tfidf_w2v_titles, X_test_clean_category, X_test_clean_subcategories,
               X_test_project_grade, X_test_school_state, X_test_teacher_prefix,
               X_test_previous_projects, X_test_price, X_test_quantity)).tocsr()

## Print the final data matrix

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

```
Final Data matrix
(49039, 702) (49039,)
(24155, 702) (24155,)
(36051, 702) (36051,)
```

## Dimensions of the hstacked features

```
In [51]: print('Training DATA\n')
print('ESSAY : ', train_tfidf_w2v_essays.shape)
print('Title : ', train_tfidf_w2v_titles.shape)
print('Categorical Data : ', (X_train_clean_category.shape + X_train_clean_subcategories.shape + X_train_project_grade.s
                             X_train_school_state.shape + X_train_teacher_prefix.shape))
print('Numerical Data : ', (X_train_previous_projects.shape + X_train_price.shape + X_train_quantity.shape))

print('\n','='*120)

print('CV DATA\n')
print('ESSAY : ', cv_tfidf_w2v_essays.shape)
print('Title : ', cv_tfidf_w2v_titles.shape)
print('Categorical Data : ', (X_cv_clean_category.shape + X_cv_clean_subcategories.shape + X_cv_project_grade.shape + \
                             X_cv_school_state.shape + X_cv_teacher_prefix.shape))
print('Numerical Data : ', (X_cv_previous_projects.shape + X_cv_price.shape + X_cv_quantity.shape))

print('\n','='*120)

print('Test DATA\n')
print('ESSAY : ', test_tfidf_w2v_essays.shape)
print('Title : ', test_tfidf_w2v_titles.shape)
print('Categorical Data : ', (X_test_clean_category.shape + X_test_clean_subcategories.shape + X_test_project_grade.shap
                             X_test_school_state.shape + X_test_teacher_prefix.shape))
print('Numerical Data : ', (X_test_previous_projects.shape + X_test_price.shape + X_test_quantity.shape))

print('\n','='*120)
```

Training DATA

```
ESSAY : (49039, 300)
Title : (49039, 300)
Categorical Data : (49039, 9, 49039, 30, 49039, 4, 49039, 51, 49039, 5)
Numerical Data : (49039, 1, 49039, 1, 49039, 1)
```

```
=====
==
CV DATA
```

```
ESSAY : (24155, 300)
Title : (24155, 300)
Categorical Data : (24155, 9, 24155, 30, 24155, 4, 24155, 51, 24155, 5)
Numerical Data : (24155, 1, 24155, 1, 24155, 1)
```

```
=====
==
Test DATA
```

```
ESSAY : (36051, 300)
Title : (36051, 300)
Categorical Data : (36051, 9, 36051, 30, 36051, 4, 36051, 51, 36051, 5)
Numerical Data : (36051, 1, 36051, 1, 36051, 1)
```

```
=====
==
```

## Hyper paramter tuning to find best $\alpha$ (alpha) (Using GridSearchCV)



```

In [52]: %%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# https://stackoverflow.com/questions/52640386/how-do-i-solve-the-future-warning-min-groups-self-n-splits-warning-in
# https://stackoverflow.com/questions/48643181/please-what-is-the-meaning-of-the-deprecation-warning-message

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc

# creating logistic regression classifier
classifier = SGDClassifier(loss='log', max_iter = 100000, tol = 1e-3)

# Lambda values
parameters = {'alpha':[0.001, 0.0005, 0.0001, 0.00005, 0.00001, 0.000005]}

# Finding the best parameter using gridsearchcv and 10-folds
clf = GridSearchCV(classifier, parameters, cv=10, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr, y_train)

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

# We use log(alpha) values so as to get a more distinguishable graph because log is monotonous function
# and it won't affect our results

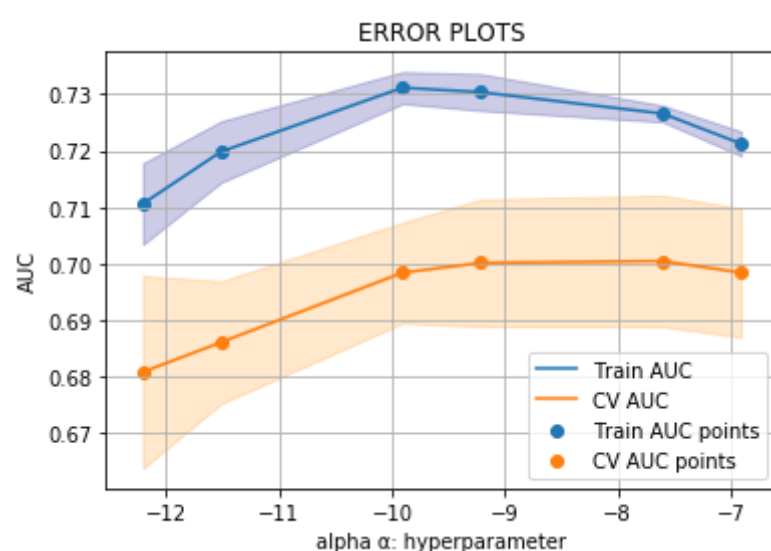
plt.plot(np.log(parameters['alpha']), train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(np.log(parameters['alpha']), train_auc - train_auc_std, train_auc + train_auc_std, alpha=0.2, color='blue')

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

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

plt.legend()
plt.xlabel("alpha  $\alpha$ : hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()

```



Wall time: 6min 2s

### Now creating the model with best $\alpha$

```

In [53]: # From the error plot we choose  $\alpha$  such that, we will have maximum AUC on cv data and gap between the train and cv is less
# Here we are choosing the best_α based on GridSearchCV results

best_α = 0.0001

```

```
In [54]: %%time
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.linear_model import SGDClassifier

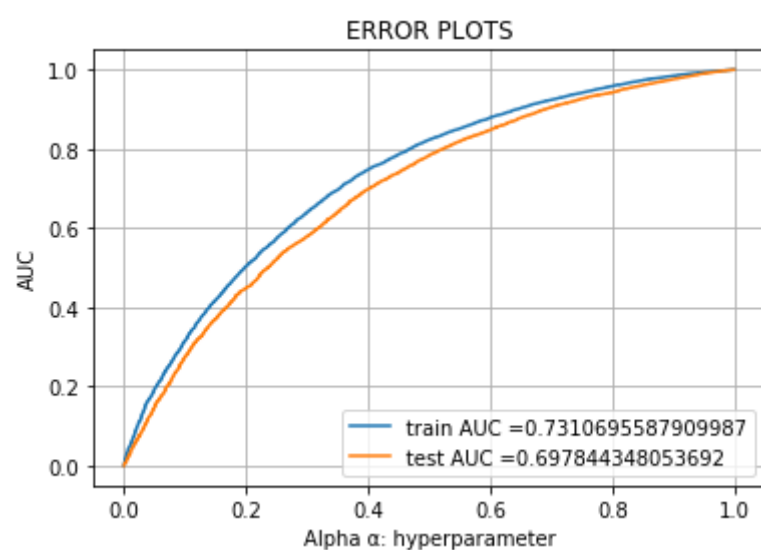
# Creating the classifier with best  $\alpha$ 
classifier = SGDClassifier(loss='log', alpha = best_α, max_iter = 100000, tol = 1e-3)
classifier.fit(X_tr, y_train)

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

# Getting the predicted probability scores for test and train values
y_train_pred = classifier.predict_proba(X_tr)[:,-1]
y_test_pred = classifier.predict_proba(X_te)[:,-1]

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

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Alpha  $\alpha$ : hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Wall time: 3.5 s

## NOTE:

- As we can see from the graph. The AUC curve is lower for the test set than the train set.
- The AUC scores for the Train and Test data are : 73% and 70% respectively
- We choose the  $\alpha$  value equal to 0.0001 because it has maximum AUC on the CV data

```
In [55]: # we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    # print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i >= t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [56]: [# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification\\_report.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)

```
from sklearn.metrics import classification_report

y_pred_new = classifier.predict(X_te)
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred_new, target_names = target_names))
```

	precision	recall	f1-score	support
class 0	0.49	0.04	0.07	5459
class 1	0.85	0.99	0.92	30592
micro avg	0.85	0.85	0.85	36051
macro avg	0.67	0.52	0.49	36051
weighted avg	0.80	0.85	0.79	36051

In [57]: `%%time`  
[# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics)

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = classifier.predict(X_te)
print("Accuracy on test set: {}".format(accuracy_score(y_test, y_pred_new)))
print("Precision on test set: {}".format(precision_score(y_test, y_pred_new)))
print("Recall on test set: {}".format(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: {}".format(f1_score(y_test, y_pred_new)))
```

Accuracy on test set: 0.8482150287093284  
Precision on test set: 0.8526504941599281  
Recall on test set: 0.9926778242677824  
F1-Score on test set: 0.9173513774770421  
Wall time: 107 ms

In [58]: `from sklearn.metrics import confusion_matrix`

```
print("="*120)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*120)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
=====
=
Train confusion matrix
[[ 4740  2685]
 [11992 29622]]
=====
=
Test confusion matrix
[[ 3776  1683]
 [12627 17965]]
```

## Function to create the confusion matrix

In [59]: [# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix](https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix)

```
def get_confusion_matrix(y_tr_ts, y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr):

    """Function to get heatmap confusion matrix"""

    # Creating the confusion matrix dataframe
    df_cm = pd.DataFrame(confusion_matrix(y_tr_ts, predict(y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr))
                        , range(2),range(2))

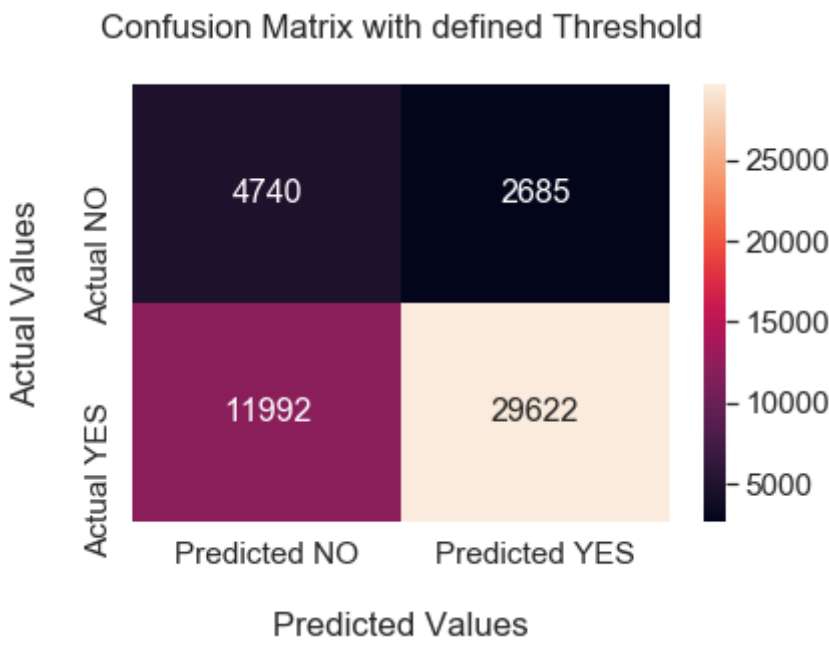
    df_cm.columns = ['Predicted NO', 'Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})

    plt.title('Confusion Matrix with defined Threshold\n ')

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

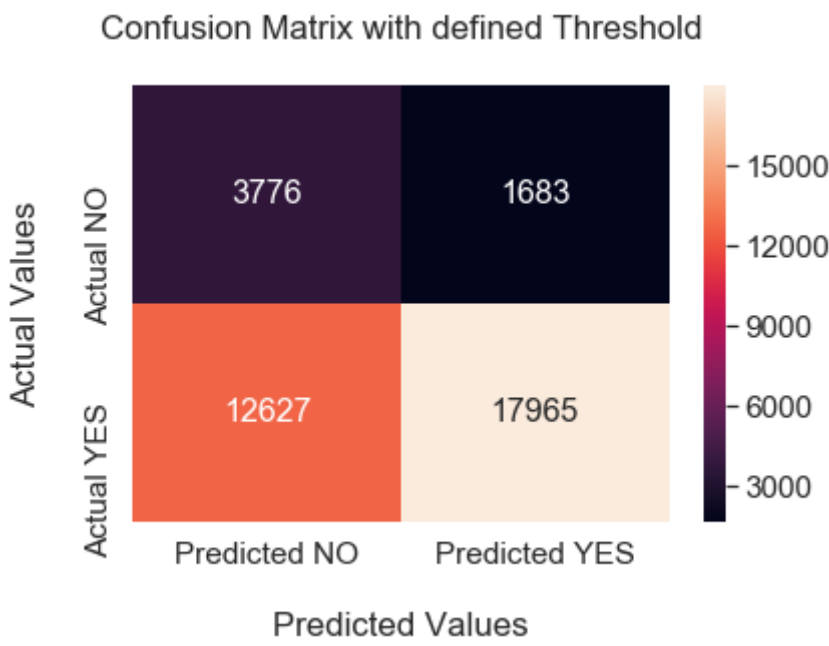
## Confusion Matrix on train data

```
In [61]: get_confusion_matrix(y_train, y_train_pred, tr_thresholds, train_fpr, train_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



Confusion Matrix on test data

```
In [62]: get_confusion_matrix(y_test, y_test_pred, tr_thresholds, test_fpr, test_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



NOTE:

- 1. The model predicts the test set correctly with a AUC score of 70%
- 2. The F1\_score obtained is 0.9173513774770421
- 3. The alpha value that we got after GridSearchCV is 0.0001

[Task-2] Apply Logistic Regression on the below feature set **Set 5**

- **school\_state** : categorical data
- **clean\_categories** : categorical data
- **clean\_subcategories** : categorical data
- **project\_grade\_category** :categorical data
- **teacher\_prefix** : categorical data
- **quantity** : numerical data
- **teacher\_number\_of\_previously\_posted\_projects** : numerical data
- **price** : numerical data
- **sentiment score's of each of the essay** : numerical data
- **number of words in the title** : numerical data
- **number of words in the combine essays** : numerical data

Merging the categorical, numerical and text features

```
In [46]: # merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# https://stackoverflow.com/questions/54226138/constructing-sparse-csr-matrix-directly-vs-using-coo-tocsr-scipy
# with the same hstack function we are concatenating a sparse matrix and a dense matrix :)

from scipy.sparse import hstack

# Training data
X_tr = hstack((X_train_words_in_essay, X_train_words_in_title, X_train_clean_category, X_train_clean_subcategories,
              X_train_project_grade, X_train_school_state, X_train_teacher_prefix,
              X_train_neg, X_train_pos, X_train_neu, X_train_compound,
              X_train_previous_projects, X_train_price, X_train_quantity)).tocsr()

# CV data
X_cr = hstack((X_cv_words_in_essay, X_cv_words_in_title, X_cv_clean_category, X_cv_clean_subcategories,
              X_cv_project_grade, X_cv_school_state, X_cv_teacher_prefix,
              X_cv_neg, X_cv_pos, X_cv_neu, X_cv_compound,
              X_cv_previous_projects, X_cv_price, X_cv_quantity)).tocsr()

# Test data
X_te = hstack((X_test_words_in_essay, X_test_words_in_title, X_test_clean_category, X_test_clean_subcategories,
              X_test_project_grade, X_test_school_state, X_test_teacher_prefix,
              X_test_neg, X_test_pos, X_test_neu, X_test_compound,
              X_test_previous_projects, X_test_price, X_test_quantity)).tocsr()

## Print the final data matrix

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

```
Final Data matrix
(49039, 108) (49039,)
(24155, 108) (24155,)
(36051, 108) (36051,)
```

## Dimensions of the hstacked features

```
In [47]: print('Training DATA\n')
print('ESSAY : ', X_train_words_in_essay.shape)
print('Title : ', X_train_words_in_title.shape)
print('Categorical Data : ', (X_train_clean_category.shape + X_train_clean_subcategories.shape + X_train_project_grade.s
                             X_train_school_state.shape + X_train_teacher_prefix.shape))
print('Numerical Data : ', (X_train_previous_projects.shape + X_train_price.shape + X_train_quantity.shape +\
                             X_train_pos.shape + X_train_neg.shape + X_train_neu.shape + X_train_compound.shape))

print('\n','='*120)

print('CV DATA\n')
print('ESSAY : ', X_cv_words_in_essay.shape)
print('Title : ', X_cv_words_in_title.shape)
print('Categorical Data : ', (X_cv_clean_category.shape + X_cv_clean_subcategories.shape + X_cv_project_grade.shape + \
                             X_cv_school_state.shape + X_cv_teacher_prefix.shape))
print('Numerical Data : ', (X_cv_previous_projects.shape + X_cv_price.shape + X_cv_quantity.shape +\
                             X_cv_pos.shape + X_cv_neg.shape + X_cv_neu.shape + X_cv_compound.shape))

print('\n','='*120)

print('Test DATA\n')
print('ESSAY : ', X_test_words_in_essay.shape)
print('Title : ', X_test_words_in_title.shape)
print('Categorical Data : ', (X_test_clean_category.shape + X_test_clean_subcategories.shape + X_test_project_grade.shap
                             X_test_school_state.shape + X_test_teacher_prefix.shape))
print('Numerical Data : ', (X_test_previous_projects.shape + X_test_price.shape + X_test_quantity.shape +\
                             X_test_pos.shape + X_test_neg.shape + X_test_neu.shape + X_test_compound.shape))

print('\n','='*120)
```

Training DATA

```
ESSAY : (49039, 1)
Title : (49039, 1)
Categorical Data : (49039, 9, 49039, 30, 49039, 4, 49039, 51, 49039, 5)
Numerical Data : (49039, 1, 49039, 1, 49039, 1, 49039, 1, 49039, 1, 49039, 1, 49039, 1)
```

=====

==

CV DATA

```
ESSAY : (24155, 1)
Title : (24155, 1)
Categorical Data : (24155, 9, 24155, 30, 24155, 4, 24155, 51, 24155, 5)
Numerical Data : (24155, 1, 24155, 1, 24155, 1, 24155, 1, 24155, 1, 24155, 1, 24155, 1)
```

=====

==

Test DATA

```
ESSAY : (36051, 1)
Title : (36051, 1)
Categorical Data : (36051, 9, 36051, 30, 36051, 4, 36051, 51, 36051, 5)
Numerical Data : (36051, 1, 36051, 1, 36051, 1, 36051, 1, 36051, 1, 36051, 1, 36051, 1)
```

=====

==

## Hyper paramter tuning to find best $\alpha$ (alpha) (Using GridSearchCV)



```
In [48]: %%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# https://stackoverflow.com/questions/52640386/how-do-i-solve-the-future-warning-min-groups-self-n-splits-warning-in
# https://stackoverflow.com/questions/48643181/please-what-is-the-meaning-of-the-deprecation-warning-message

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc

# creating logistic regression classifier
classifier = SGDClassifier(loss='log', max_iter = 100000, tol = 1e-3)

# Lambda values
parameters = {'alpha':[0.03, 0.015, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005]}

# Finding the best parameter using gridsearchcv and 10-folds
clf = GridSearchCV(classifier, parameters, cv=10, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr, y_train)

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

# We use log(alpha) values so as to get a more distinguishable graph because log is monotonous function
# and it won't affect our results

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

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

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

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



Wall time: 28.2 s

### Now creating the model with best $\alpha$

```
In [49]: # From the error plot we choose  $\alpha$  such that, we will have maximum AUC on cv data and gap between the train and cv is less
# Here we are choosing the best_alpha based on GridSearchCV results

best_alpha = 0.005
```

```
In [50]: %%time
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.linear_model import SGDClassifier

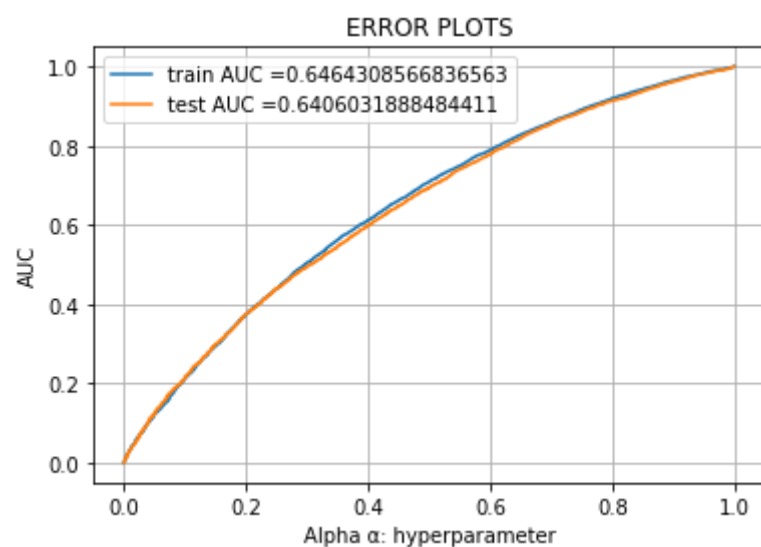
# Creating the classifier with best  $\alpha$ 
classifier = SGDClassifier(loss='log', alpha = best_α, max_iter = 100000, tol = 1e-3)
classifier.fit(X_tr, y_train)

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

# Getting the predicted probability scores for test and train values
y_train_pred = classifier.predict_proba(X_tr)[:,-1]
y_test_pred = classifier.predict_proba(X_te)[:,-1]

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

plt.plot(train_fpr, train_tpr, label="train AUC =" + str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC =" + str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Alpha  $\alpha$ : hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Wall time: 297 ms

## NOTE:

- As we can see from the graph. The difference in the AUC curve is very less between the train data and test data.
- The AUC scores for the Train and Test data are : 64% and 64% respectively
- We choose the  $\alpha$  value equal to 0.005 because it has maximum AUC on the CV data

```
In [51]: # we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    #print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i >= t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [52]: [# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification\\_report.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)

```
from sklearn.metrics import classification_report

y_pred_new = classifier.predict(X_te)
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred_new, target_names = target_names))
```

	precision	recall	f1-score	support
class 0	0.22	0.00	0.01	5459
class 1	0.85	1.00	0.92	30592
micro avg	0.85	0.85	0.85	36051
macro avg	0.53	0.50	0.46	36051
weighted avg	0.75	0.85	0.78	36051

In [53]: `%%time`  
[# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics)

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = classifier.predict(X_te)
print("Accuracy on test set: {}".format(accuracy_score(y_test, y_pred_new)))
print("Precision on test set: {}".format(precision_score(y_test, y_pred_new)))
print("Recall on test set: {}".format(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: {}".format(f1_score(y_test, y_pred_new)))
```

Accuracy on test set: 0.8473551357798674  
Precision on test set: 0.848719873238262  
Recall on test set: 0.9980060146443515  
F1-Score on test set: 0.917328926613085  
Wall time: 38.9 ms

In [54]: `from sklearn.metrics import confusion_matrix`

```
print("="*120)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*120)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
=====
=
Train confusion matrix
[[ 4551  2874]
 [16584 25030]]
=====
=
Test confusion matrix
[[ 3847  1612]
 [15619 14973]]
```

## Function to create the confusion matrix

In [55]: [# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix](https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix)

```
def get_confusion_matrix(y_tr_ts, y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr):

    """Function to get heatmap confusion matrix"""

    # Creating the confusion matrix dataframe
    df_cm = pd.DataFrame(confusion_matrix(y_tr_ts, predict(y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr))
                        , range(2),range(2))

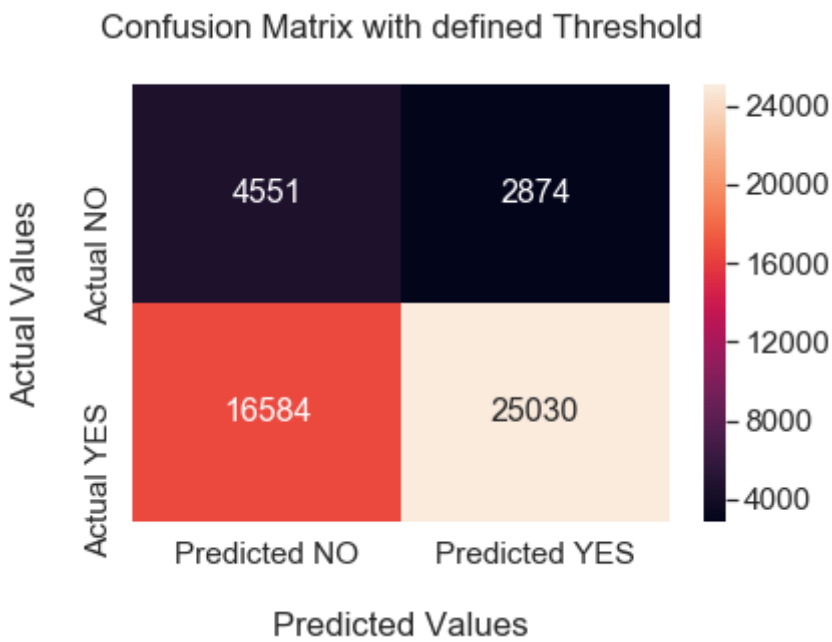
    df_cm.columns = ['Predicted NO', 'Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})

    plt.title('Confusion Matrix with defined Threshold\n ')

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

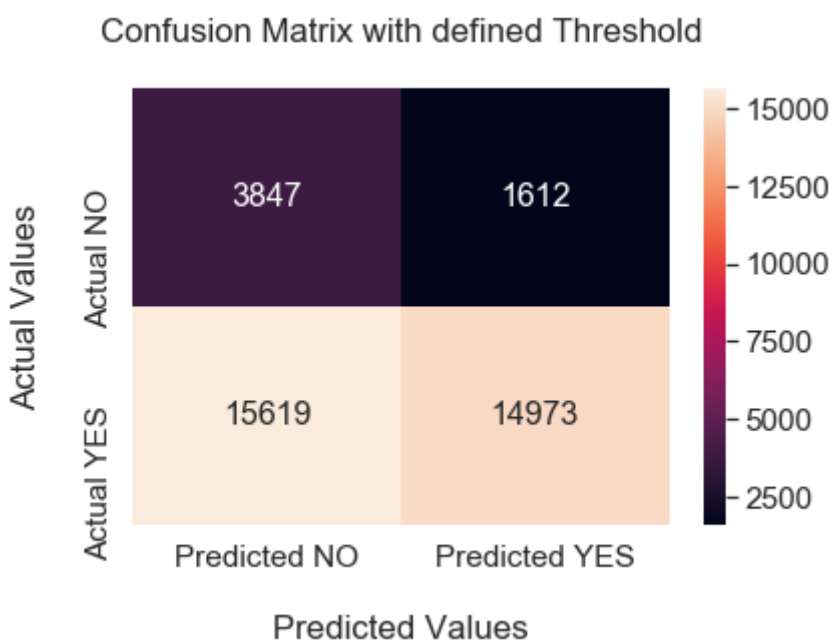
## Confusion Matrix on train data

```
In [58]: get_confusion_matrix(y_train, y_train_pred, tr_thresholds, train_fpr, train_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



Confusion Matrix on test data

```
In [57]: get_confusion_matrix(y_test, y_test_pred, tr_thresholds, test_fpr, test_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



- NOTE:**
- 1. The model predicts the test set correctly with a AUC score of 64%
  - 2. The F1\_score obtained is 0.917328926613085
  - 3. The alpha value that we got after GridSearchCV is 0.005

CONCLUSION

In [59]: `#http://zetcode.com/python/prettytable/  
from prettytable import PrettyTable  
  
x = PrettyTable()  
x.field_names = ["Vectorizer", "Model [Algorithm - 'log']", "Hyper parameter [α-value]", "AUC Score [train]", "AUC Score [test]"  
x.add_row(["Bag of Words", "SGDClassifier", 0.001, 0.79, 0.67])  
x.add_row(["TFIDF", "SGDClassifier", 0.0001, 0.77, 0.67])  
x.add_row(["AVG W2V", "SGDClassifier", 0.0001, 0.72, 0.70])  
x.add_row(["TFIDF weighted W2V", "SGDClassifier", 0.0001, 0.73, 0.70])  
x.add_row(["All Numerical Features", "SGDClassifier", 0.005, 0.64, 0.64])  
  
print(x)`

Vectorizer	Model [Algorithm - 'log']	Hyper parameter [α-value]	AUC Score [train]	AUC Score [test]
Bag of Words	SGDClassifier	0.001	0.79	0.67
TFIDF	SGDClassifier	0.0001	0.77	0.67
AVG W2V	SGDClassifier	0.0001	0.72	0.7
TFIDF weighted W2V	SGDClassifier	0.0001	0.73	0.7
All Numerical Features	SGDClassifier	0.005	0.64	0.64

NOTE:

1. As we can see from the results that, converting the test features (essays and titles) into numerical values doesn't make any upgrade in the AUC scores, instead we got less AUC scores for the Numerical feature (SET 5) than the other vectorizers.
2. It is clearly visible that Text data contained in the Essays and Essay Titles indeed play a major role in predicting the outcome of the project. Hence, it cannot be neglected as most of the models containing them proved to have a better AUC score.

In [ ]:

In [ ]:

In [ ]: