9_DonorsChoose_GBDT (1)

March 21, 2020

1 DonorsChoose

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

```
Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result

How to scale current manual processes and resources to screen 500,000 projects so that they ca
<li>How to increase the consistency of project vetting across different volunteers to improve
<li>How to focus volunteer time on the applications that need the most assistance</li>
</ul>
```

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

1.1 About the DonorsChoose Data Set

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

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

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

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

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

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

Literacy
Literature & Writing, Social Sciences

`project_resource_summary` | An explanation of the resources needed for the project. **Example:**

My students need hands on literacy materials to manage sensory needs!

`project_essay_1` | First application essay
`project_essay_2` | *Second application essay* `project_essay_3` | Third application essay
`project_essay_4` | *Fourth application essay* `project_submitted_datetime` | Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id` | A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix` | Teacher's title. One of the following enumerated values:

nan
Dr.
Mr.
Mrs.
Ms.
Teacher.

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

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

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

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

Feature	Description
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

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

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

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

1.1.1 Notes on the Essay Data

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

project_essay_1: "Introduce us to your classroom"

project_essay_2: "Tell us more about your students"

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

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

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

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

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

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

```
In [3]: %matplotlib inline
        import warnings
        warnings.filterwarnings("ignore")

        import sqlite3
        import pandas as pd
```

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

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

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

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

from tqdm import tqdm
import os


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

1.2 1.1 Reading Data

```
In [4]: from google.colab import drive
        drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6

Enter your authorization code:
ůůůůůůůůůů
Mounted at /content/drive
```

```
In [0]: project_data = pd.read_csv('/content/drive/My Drive/Assignments_DonorsChoose_2018/train
        resource_data = pd.read_csv('/content/drive/My Drive/Assignments_DonorsChoose_2018/reso
```

```
In [6]: print("Number of data points in train data", project_data.shape)
        print('-'*50)
        print("The attributes of data :", project_data.columns.values)
```

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

```
In [7]: print("Number of data points in train data", resource_data.shape)
        print(resource_data.columns.values)
        resource_data.head(2)
```

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

Out[7]:

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

1.3 1.2 preprocessing of `project_subject_categories`

```
In [0]: catogories = list(project_data['project_subject_categories'].values)
        # remove special characters from list of strings python: https://stackoverflow.com/a/4

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

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

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

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

1.4 1.3 preprocessing of project_subject_subcategories

```
In [0]: sub_catogories = list(project_data['project_subject_subcategories'].values)
        # remove special characters from list of strings python: https://stackoverflow.com/a/4

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

        sub_cat_list = []
        for i in sub_catogories:
            temp = ""
            # consider we have text like this "Math & Science, Warmth, Care & Hunger"
            for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmt
                if 'The' in j.split(): # this will split each of the catogory based on space "
                    j=j.replace('The','') # if we have the words "The" we are going to replace
                j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"
                temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing sp
                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/22898595/408403
        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]))
```

Preprocessing of project_grade_category

```
In [0]: project_grade = list(project_data['project_grade_category'].values)
        # remove special characters from list of strings python: https://stackoverflow.com/a/4

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

```
grade_cat_list = []
for i in project_grade:
    # consider we have text like this:
    for j in i.split(' '): #    # split by spae
        j=j.replace('Grades','')# clean grades from the row
    grade_cat_list.append(j.strip())



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

1.5 1.3 Text preprocessing

```
In [0]: # merge two column text dataframe:
        project_data["essay"] = project_data["project_essay_1"].map(str) +\
                                project_data["project_essay_2"].map(str) + \
                                project_data["project_essay_3"].map(str) + \
                                project_data["project_essay_4"].map(str)
```

```
In [12]: project_data.head(2)
```

```
Out[12]:    Unnamed: 0  ...                                              essay
         0      160221  ...  My students are English learners that are work...
         1      140945  ...  Our students arrive to our school eager to lea...

         [2 rows x 18 columns]
```

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

```
In [14]: # printing some random reviews
         print(project_data['essay'].values[0])
         print("="*50)
         print(project_data['essay'].values[150])
         print("="*50)
         print(project_data['essay'].values[1000])
         print("="*50)
         print(project_data['essay'].values[20000])
         print("="*50)
```

```
My students are English learners that are working on English as their second or third languages
==================================================
The 51 fifth grade students that will cycle through my classroom this year all love learning, a
==================================================
How do you remember your days of school? Was it in a sterile environment with plain walls, rows
==================================================
My kindergarten students have varied disabilities ranging from speech and language delays, cogn
==================================================
```

```
In [0]: # https://stackoverflow.com/a/47091490/4084039
        import re

        def decontracted(phrase):
            # specific
            phrase = re.sub(r"won't", "will not", phrase)
            phrase = re.sub(r"can\'t", "can not", phrase)

            # general
            phrase = re.sub(r"n\'t", " not", phrase)
            phrase = re.sub(r"\'re", " are", phrase)
            phrase = re.sub(r"\'s", " is", phrase)
            phrase = re.sub(r"\'d", " would", phrase)
            phrase = re.sub(r"\'ll", " will", phrase)
            phrase = re.sub(r"\'t", " not", phrase)
            phrase = re.sub(r"\'ve", " have", phrase)
            phrase = re.sub(r"\'m", " am", phrase)
            return phrase
```

```
In [16]: sent = decontracted(project_data['essay'].values[20000])
         print(sent)
         print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogn
==================================================
```

```
In [17]: # \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-
         sent = sent.replace('\\r', ' ')
         sent = sent.replace('\\"', ' ')
         sent = sent.replace('\\n', ' ')
         print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogn
```

```
In [18]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
         print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays cogni
```

```
In [0]: # https://gist.github.com/sebleier/554280
        # we are removing the words from the stop words list: 'no', 'nor', 'not'
        stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're
                    "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
                    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', '
                    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
```

```
                    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
                    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
                    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
                    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
                    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any
                    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
                    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
                    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
                    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
                    'won', "won't", 'wouldn', "wouldn't"]
```

```
In [20]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_essays = []
         # tqdm is for printing the status bar
         for sentance in tqdm(project_data['essay'].values):
             sent = decontracted(sentance)
             sent = sent.replace('\\r', ' ')
             sent = sent.replace('\\"', ' ')
             sent = sent.replace('\\n', ' ')
             sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
             # https://gist.github.com/sebleier/554280
             sent = ' '.join(e for e in sent.split() if e not in stopwords)
             preprocessed_essays.append(sent.lower().strip())
```

```
100%|| 109248/109248 [01:01<00:00, 1774.42it/s]
```

1.4 Preprocessing of `project_title`

```
In [0]: # https://stackoverflow.com/a/47091490/4084039
        import re

        def decontracted(phrase):
            # specific
            phrase = re.sub(r"won't", "will not", phrase)
            phrase = re.sub(r"can\'t", "can not", phrase)

            # general
            phrase = re.sub(r"n\'t", " not", phrase)
            phrase = re.sub(r"\'re", " are", phrase)
            phrase = re.sub(r"\'s", " is", phrase)
            phrase = re.sub(r"\'d", " would", phrase)
            phrase = re.sub(r"\'ll", " will", phrase)
            phrase = re.sub(r"\'t", " not", phrase)
            phrase = re.sub(r"\'ve", " have", phrase)
            phrase = re.sub(r"\'m", " am", phrase)
            return phrase
```

```
In [22]: sent = decontracted(project_data['project_title'].values[2000])
         print(sent)
         print("="*50)

Steady Stools for Active Learning
==================================================


In [23]: # \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-p
         sent = sent.replace('\\r', ' ')
         sent = sent.replace('\\"', ' ')
         sent = sent.replace('\\n', ' ')
         print(sent)

Steady Stools for Active Learning


In [24]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
         print(sent)

Steady Stools for Active Learning


In [0]: # https://gist.github.com/sebleier/554280
        # we are removing the words from the stop words list: 'no', 'nor', 'not'
        stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're
                    "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
                    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', '
                    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
                    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
                    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
                    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
                    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
                    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any
                    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
                    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
                    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
                    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
                    'won', "won't", 'wouldn', "wouldn't"]

In [26]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_titles = []
         # tqdm is for printing the status bar
         for sentance in tqdm(project_data['project_title'].values):
             sent = decontracted(sentance)
             sent = sent.replace('\\r', ' ')
```

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

```
100%|| 109248/109248 [00:02<00:00, 42072.19it/s]
```

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

Out[27]:

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

```
In [28]: price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_
         price_data.head(2)
```

Out[28]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

```
In [29]: project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')

         project_data.head(2)
```

Out[29]:

	Unnamed: 0	...	essay
0	160221	...	My students are English learners that are work...
1	140945	...	Our students arrive to our school eager to lea...

[2 rows x 18 columns]

```
In [0]: #Join train & Resource dataset
        # join two dataframes in python:

        data = pd.merge(project_data, price_data, on='id', how='left')
```

```
In [0]: approved_price = data[data['project_is_approved']==1]['price'].values

        rejected_price = data[data['project_is_approved']==0]['price'].values
```

```
In [32]: data.head(2)
```

Out[32]:

	Unnamed: 0	id	...	price	quantity
0	160221	p253737	...	154.6	23
1	140945	p258326	...	299.0	1

[2 rows x 20 columns]

Train Test split

```
In [33]: project_data = data.sample(n=50000)
         project_data.head(3)
```

```
Out[33]:        Unnamed: 0       id  ...   price quantity
         19203        1594  p072319  ...   13.90       16
         78217      178775  p119439  ...  610.44        4
         11500      162400  p244731  ...   48.20       34

         [3 rows x 20 columns]
```

```
In [34]: y = project_data['project_is_approved'].values
         X = project_data
         X.head(1)
```

```
Out[34]:        Unnamed: 0       id  ... price quantity
         19203        1594  p072319  ...  13.9       16

         [1 rows x 20 columns]
```

```
In [0]: y=project_data['project_is_approved'].values
```

```
In [0]: # train test split
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
```

```
In [37]: # Combining all the above stundents
         from tqdm import tqdm
         train_preprocessed_essays = []
         # tqdm is for printing the status bar
         for sentance in tqdm(X_train['essay'].values):
             sent = decontracted(sentance)
             sent = sent.replace('\\r', ' ')
             sent = sent.replace('\\"', ' ')
             sent = sent.replace('\\n', ' ')
             sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
             # https://gist.github.com/sebleier/554280
             sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
             train_preprocessed_essays.append(sent.lower().strip())
```

```
100%|| 40000/40000 [00:22<00:00, 1795.14it/s]
```

```
In [38]: # Combining all the above stundents
         from tqdm import tqdm
         test_preprocessed_essays = []
         # tqdm is for printing the status bar
         for sentance in tqdm(X_test['essay'].values):
```

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

```
100%|| 10000/10000 [00:05<00:00, 1798.30it/s]
```

```
In [39]: # after preprocesing
         preprocessed_essays[20000]
```

```
Out[39]: 'my kindergarten students varied disabilities ranging speech language delays cognitive
```

```
In [40]: # Combining all the above stundents
         from tqdm import tqdm
         train_preprocessed_titles = []
         # tqdm is for printing the status bar
         for sentance in tqdm(X_train['project_title'].values):
             sent = decontracted(sentance)
             sent = sent.replace('\\r', ' ')
             sent = sent.replace('\\"', ' ')
             sent = sent.replace('\\n', ' ')
             sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
             # https://gist.github.com/sebleier/554280
             sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
             train_preprocessed_titles.append(sent.lower().strip())
```

```
100%|| 40000/40000 [00:00<00:00, 41965.20it/s]
```

```
In [41]: # Combining all the above stundents
         from tqdm import tqdm
         test_preprocessed_titles = []
         # tqdm is for printing the status bar
         for sentance in tqdm(X_test['project_title'].values):
             sent = decontracted(sentance)
             sent = sent.replace('\\r', ' ')
             sent = sent.replace('\\"', ' ')
             sent = sent.replace('\\n', ' ')
             sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
             # https://gist.github.com/sebleier/554280
             sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
             test_preprocessed_titles.append(sent.lower().strip())
```

```
100%|| 10000/10000 [00:00<00:00, 40854.31it/s]
```

```
In [42]: preprocessed_titles[2000]
```

```
Out[42]: 'steady stools active learning'
```

1.6 1.5 Preparing data for models

```
In [43]: project_data.columns
```

```
Out[43]: Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
                'project_submitted_datetime', 'project_title', 'project_essay_1',
                'project_essay_2', 'project_essay_3', 'project_essay_4',
                'project_resource_summary',
                'teacher_number_of_previously_posted_projects', 'project_is_approved',
                'clean_categories', 'clean_subcategories', 'grade_cat_list', 'essay',
                'price', 'quantity'],
               dtype='object')
```

we are going to consider

```
- school_state : categorical data
- clean_categories : categorical data
- clean_subcategories : categorical data
- project_grade_category : categorical data
- teacher_prefix : categorical data

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

- quantity : numerical (optinal)
- teacher_number_of_previously_posted_projects : numerical
- price : numerical
```

2 Response coding for Categorical Data

```
In [0]: #http://www.saedsayad.com/encoding.htm
        #https://gist.github.com/marnixkoops/e68815d30474786e2b293682ed7cdb01
        #https://www.slideshare.net/0xdata/feature-engineering-83511751
```

3 Response Coding - School state

```
In [0]: # code for response coding with Laplace smoothing.
        # alpha : used for laplace smoothing

        def get_cat_fea_dict(alpha, feature, df):
            value_count = X_train[feature].value_counts()

            cat_dict = dict()
```

```
            # denominator will contain the number of time that particular feature occured in w
            for i, denominator in value_count.items():
                vec = []
                for k in range(0,2):

                    cls_cnt = X_train.loc[(X_train['project_is_approved']==k) & (X_train[featu

                    # cls_cnt.shape[0](numerator) will contain the number of time that particu
                    vec.append((cls_cnt.shape[0] + alpha*1)/ (denominator + 2*alpha))

                cat_dict[i]=vec
            return cat_dict


        def get_cat_feature(alpha, feature, df):
                cat_dict = get_cat_fea_dict(alpha, feature, df)

                value_count = X_train[feature].value_counts()

                cat_fea = []
                # for every feature values in the given data frame we will check if it is ther
                # if not we will add [1/2,1/2] to cat_fea
                for index, row in df.iterrows():
                    if row[feature] in dict(value_count).keys():
                        cat_fea.append(cat_dict[row[feature]])
                    else:
                        cat_fea.append([1/2,1/2])

                return cat_fea
```

In [0]:
```
#response-coding of the clean_categories
# alpha is used for laplace smoothing
alpha = 1

# train gene feature
tr_clean_categories = np.array(get_cat_feature(alpha, "clean_categories", X_train))

# test gene feature
te_clean_categories = np.array(get_cat_feature(alpha, "clean_categories", X_test))
```

In [46]:
```
print("The shape of train_clean_categories feature:", tr_clean_categories.shape)
print("The shape of test_clean_categories feature:", te_clean_categories.shape)
```

```
The shape of train_clean_categories feature: (40000, 2)
The shape of test_clean_categories feature: (10000, 2)
```

In [47]:
```
#response-coding of the clean_subcategories
# alpha is used for laplace smoothing
```

```
        alpha = 1

        # train gene feature
        tr_clean_subcategories = np.array(get_cat_feature(alpha, "clean_subcategories", X_tra

        # test gene feature
        te_clean_subcategories = np.array(get_cat_feature(alpha, "clean_subcategories", X_tes


        print("The shape of tr_clean_subcategories feature:", tr_clean_subcategories.shape)
        print("The shape of te_clean_subcategories feature:", te_clean_subcategories.shape)
```

```
The shape of tr_clean_subcategories feature: (40000, 2)
The shape of te_clean_subcategories feature: (10000, 2)
```

In [48]:
```
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')

        #response-coding of the clean_subcategories
        # alpha is used for laplace smoothing
        alpha = 1

        # train gene feature
        tr_prefix = np.array(get_cat_feature(alpha, "teacher_prefix", X_train))

        # test gene feature
        te_prefix = np.array(get_cat_feature(alpha, "teacher_prefix", X_test))


        print("The shape of train_teacher_prefix feature:", tr_prefix.shape)
        print("The shape of test_teacher_prefix feature:", te_prefix.shape)
```

```
The shape of train_teacher_prefix feature: (40000, 2)
The shape of test_teacher_prefix feature: (10000, 2)
```

In [49]:
```
#response-coding of the clean_subcategories
        # alpha is used for laplace smoothing
        alpha = 1

        # train gene feature
        tr_school_state = np.array(get_cat_feature(alpha, "school_state", X_train))

        # test gene feature
        te_school_state = np.array(get_cat_feature(alpha, "school_state", X_test))


        print("The shape of train_school_state feature:", tr_school_state.shape)
        print("The shape of test_school_state feature:", te_school_state.shape)
```

```
The shape of train_school_state feature: (40000, 2)
The shape of test_school_state feature: (10000, 2)
```

```
In [50]: #response-coding of the clean_subcategories
         # alpha is used for laplace smoothing
         alpha = 1

         # train gene feature
         tr_project_grade = np.array(get_cat_feature(alpha, "grade_cat_list", X_train))

         # test gene feature
         te_project_grade = np.array(get_cat_feature(alpha, "grade_cat_list", X_test))


         print("The shape of tr_clean_project_grade_category feature:", tr_project_grade.shape)
         print("The shape of te_clean_project_grade_category feature:", te_project_grade.shape)
```

```
The shape of tr_clean_project_grade_category feature: (40000, 2)
The shape of te_clean_project_grade_category feature: (10000, 2)
```

3.0.1 1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

```
In [51]: # We are considering only the words which appeared in at least 10 documents(rows or p
         vectorizer = CountVectorizer(min_df=10)
         text_bow = vectorizer.fit_transform(preprocessed_essays)
         print("Shape of matrix after one hot encodig ",text_bow.shape)
```

```
Shape of matrix after one hot encodig  (109248, 16623)
```

```
In [52]: # We are considering only the words which appeared in at least 10 documents(rows or p
         vectorizer = CountVectorizer(min_df=10)
         text_bow = vectorizer.fit_transform(preprocessed_titles)
         print("Shape of matrix after one hot encodig ",text_bow.shape)
```

```
Shape of matrix after one hot encodig  (109248, 3222)
```

1.5.2.2 TFIDF vectorizer

```
In [53]: from sklearn.feature_extraction.text import TfidfVectorizer
         vectorizer = TfidfVectorizer(min_df=10)
         text_tfidf = vectorizer.fit_transform(preprocessed_essays)
         print("Shape of matrix after one hot encodig ",text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (109248, 16623)
```

```
In [54]: from sklearn.feature_extraction.text import TfidfVectorizer
         vectorizer = TfidfVectorizer(min_df=10)
         text_tfidf = vectorizer.fit_transform(preprocessed_titles)
         print("Shape of matrix after one hot encodig ",text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (109248, 3222)
```

1.5.2.3 Using Pretrained Models: Avg W2V

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

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

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

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

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

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

        inter_words = set(model.keys()).intersection(words)
```

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

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


        # stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-

        import pickle
        with open('glove_vectors', 'wb') as f:
            pickle.dump(words_courpus, f)


        '''
```

Out[0]: '\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\nde

```
In [0]: # stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-
        # make sure you have the glove_vectors file
        with open('/content/drive/My Drive/Assignments_DonorsChoose_2018/glove_vectors', 'rb')
            model = pickle.load(f)
            glove_words =  set(model.keys())
```

```
In [56]: # average Word2Vec
         # compute average word2vec for each review.
         avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
         for sentence in tqdm(preprocessed_essays): # for each review/sentence
             vector = np.zeros(300) # as word vectors are of zero length
             cnt_words =0; # num of words with a valid vector in the sentence/review
             for word in sentence.split(): # for each word in a review/sentence
                 if word in glove_words:
                     vector += model[word]
                     cnt_words += 1
             if cnt_words != 0:
                 vector /= cnt_words
             avg_w2v_vectors.append(vector)

         print(len(avg_w2v_vectors))
         print(len(avg_w2v_vectors[0]))
```

```
100%|| 109248/109248 [00:35<00:00, 3119.76it/s]

109248
300
```

In [57]:
```
# average Word2Vec
# compute average word2vec for each review.
train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this
for sentence in tqdm(train_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_avg_w2v_vectors.append(vector)

print(len(train_avg_w2v_vectors))
print(len(train_avg_w2v_vectors[0]))
```

```
100%|| 40000/40000 [00:12<00:00, 3226.70it/s]
```

```
40000
300
```

In [58]:
```
# average Word2Vec
# compute average word2vec for each review.
test_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this l
for sentence in tqdm(test_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_avg_w2v_vectors.append(vector)

print(len(test_avg_w2v_vectors))
print(len(test_avg_w2v_vectors[0]))
```

```
100%|| 10000/10000 [00:03<00:00, 3251.80it/s]
```

```
10000
300
```

In [59]:
```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors1 = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors1.append(vector)

print(len(avg_w2v_vectors1))
print(len(avg_w2v_vectors1[0]))
```

```
100%|| 109248/109248 [00:01<00:00, 63255.67it/s]
```

```
109248
300
```

In [60]:
```
# average Word2Vec
# compute average word2vec for each review.
train_avg_w2v_vectors1 = []; # the avg-w2v for each sentence/review is stored in this
for sentence in tqdm(train_preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_avg_w2v_vectors1.append(vector)

print(len(train_avg_w2v_vectors1))
print(len(train_avg_w2v_vectors1[0]))
```

```
100%|| 40000/40000 [00:00<00:00, 59550.37it/s]
```

```
40000
300
```

```
In [61]: # average Word2Vec
         # compute average word2vec for each review.
         test_avg_w2v_vectors1 = []; # the avg-w2v for each sentence/review is stored in this
         for sentence in tqdm(test_preprocessed_titles): # for each review/sentence
             vector = np.zeros(300) # as word vectors are of zero length
             cnt_words =0; # num of words with a valid vector in the sentence/review
             for word in sentence.split(): # for each word in a review/sentence
                 if word in glove_words:
                     vector += model[word]
                     cnt_words += 1
             if cnt_words != 0:
                 vector /= cnt_words
             test_avg_w2v_vectors1.append(vector)

         print(len(test_avg_w2v_vectors1))
         print(len(test_avg_w2v_vectors1[0]))
```

```
100%|| 10000/10000 [00:00<00:00, 58808.99it/s]

10000
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

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

```
In [63]: # average Word2Vec
         # compute average word2vec for each review.
         tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
         for sentence in tqdm(preprocessed_essays): # 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((s
                     tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) #
```

```
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)

    print(len(tfidf_w2v_vectors))
    print(len(tfidf_w2v_vectors[0]))
```

```
100%|| 109248/109248 [03:41<00:00, 493.45it/s]

109248
300
```

In [64]:
```
# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in thi
for sentence in tqdm(train_preprocessed_essays): # 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((s
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) #
            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_vectors.append(vector)

print(len(train_tfidf_w2v_vectors))
print(len(train_tfidf_w2v_vectors[0]))
```

```
100%|| 40000/40000 [01:11<00:00, 558.02it/s]

40000
300
```

In [65]:
```
# average Word2Vec
# compute average word2vec for each review.
```

```
        test_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this
        for sentence in tqdm(test_preprocessed_essays): # 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((s
                    tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) #
                    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_vectors.append(vector)

        print(len(test_tfidf_w2v_vectors))
        print(len(test_tfidf_w2v_vectors[0]))
```

```
100%|| 10000/10000 [00:17<00:00, 557.79it/s]

10000
300
```

In [66]:
```
        # average Word2Vec
        # compute average word2vec for each review.
        tfidf_w2v_vectors1 = []; # the avg-w2v for each sentence/review is stored in this lis
        for sentence in tqdm(preprocessed_titles): # 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((s
                    tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) #
                    vector += (vec * tf_idf) # calculating tfidf weighted w2v
                    tf_idf_weight += tf_idf
            if tf_idf_weight != 0:
                vector /= tf_idf_weight
            tfidf_w2v_vectors1.append(vector)

        print(len(tfidf_w2v_vectors1))
        print(len(tfidf_w2v_vectors1[0]))
```

```
100%|| 109248/109248 [00:03<00:00, 28067.98it/s]
```

```
109248
300
```

In [67]:
```
# average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_vectors1 = []; # the avg-w2v for each sentence/review is stored in th
for sentence in tqdm(train_preprocessed_titles): # 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((s
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) #
            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_vectors1.append(vector)

print(len(train_tfidf_w2v_vectors1))
print(len(train_tfidf_w2v_vectors1[0]))
```

```
100%|| 40000/40000 [00:01<00:00, 31714.08it/s]

40000
300
```

In [68]:
```
# average Word2Vec
# compute average word2vec for each review.
test_tfidf_w2v_vectors1 = []; # the avg-w2v for each sentence/review is stored in thi
for sentence in tqdm(test_preprocessed_titles): # 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((s
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) #
            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_vectors1.append(vector)

        print(len(test_tfidf_w2v_vectors1))
        print(len(test_tfidf_w2v_vectors1[0]))
```

```
100%|| 10000/10000 [00:00<00:00, 32296.52it/s]

10000
300
```

3.0.2 1.5.3 Vectorizing Numerical features

```
In [69]: # check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
         # standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.
         from sklearn.preprocessing import StandardScaler

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

         price_scalar = StandardScaler()
         price_scalar.fit(project_data['price'].values.reshape(-1,1)) # finding the mean and s
         print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.va

         # Now standardize the data with above maen and variance.
         price_standardized = price_scalar.transform(project_data['price'].values.reshape(-1,
         tr_price_standardized = price_scalar.transform(X_train['price'].values.reshape(-1, 1))
         #cv_price_standardized = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
         te_price_standardized = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
```

```
Mean : 302.2916604, Standard deviation : 396.57463031408236
```

```
In [70]: price_standardized
```

```
Out[70]: array([[-0.72720653],
                [ 0.77702484],
                [-0.64071587],
                ...,
                [ 0.24756586],
                [-0.1223519 ],
                [-0.7328549 ]])
```

```
In [71]: # check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
         # standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.
         from sklearn.preprocessing import StandardScaler

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

         quantity_scalar = StandardScaler()
         quantity_scalar.fit(project_data['quantity'].values.reshape(-1,1)) # finding the mean
         print(f"Mean : {quantity_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scal

         # Now standardize the data with above maen and variance.
         quantity_standardized = quantity_scalar.transform(project_data['quantity'].values.resh
         tr_quantity_standardized = quantity_scalar.transform(X_train['quantity'].values.reshap
         #cv_quantity_standardized = quantity_scalar.transform(X_cv['quantity'].values.reshape
         te_quantity_standardized = quantity_scalar.transform(X_test['quantity'].values.reshape

Mean : 17.05354, Standard deviation : 26.34881920444254


In [72]: quantity_standardized

Out[72]: array([[-0.03998433],
                [-0.49541271],
                [ 0.64315823],
                ...,
                [-0.49541271],
                [-0.57131744],
                [ 1.25039607]])
```

__ Computing Sentiment Scores__

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

        # import nltk
        # nltk.download('vader_lexicon')

        sid = SentimentIntensityAnalyzer()

        for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest
        for learning my students learn in many different ways using all of our senses and multi
        of techniques to help all my students succeed students in my class come from a variety
        for wonderful sharing of experiences and cultures including native americans our school
        learners which can be seen through collaborative student project based learning in and
        in my class love to work with hands on materials and have many different opportunities
        mastered having the social skills to work cooperatively with friends is a crucial aspe
```

```
        montana is the perfect place to learn about agriculture and nutrition my students love
        in the early childhood classroom i have had several kids ask me can we try cooking with
        and create common core cooking lessons where we learn important math and writing concep
        food for snack time my students will have a grounded appreciation for the work that wen
        of where the ingredients came from as well as how it is healthy for their bodies this p
        nutrition and agricultural cooking recipes by having us peel our own apples to make hom
        and mix up healthy plants from our classroom garden in the spring we will also create o
        shared with families students will gain math and literature skills as well as a life lo
        nannan'
        ss = sid.polarity_scores(for_sentiment)

        for k in ss:
            print('{0}: {1}, '.format(k, ss[k]), end='')

        # we can use these 4 things as features/attributes (neg, neu, pos, compound)
        # neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
D:\installed\Anaconda3\lib\site-packages\nltk\twitter\__init__.py:20: UserWarning:

The twython library has not been installed. Some functionality from the twitter package will no
```

```
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

4 Assignment 9: RF and GBDT

Response Coding: Example

> The response tabel is built only on train dataset. For a category which is not there in train data and present in test data, we will encode them with default values Ex: in our test data if have State: D then we encode it as [0.5, 0.5]

```
<li><strong>Apply both Random Forrest and GBDT on these feature sets</strong>
    <ul>
        <li><font color='red'>Set 1</font>: categorical(instead of one hot encoding, try <a hre
        <li><font color='red'>Set 2</font>: categorical(instead of one hot encoding, try <a hre
        <li><font color='red'>Set 3</font>: categorical(instead of one hot encoding, try <a hre
        <li><font color='red'>Set 4</font>: categorical(instead of one hot encoding, try <a hre
</li>
<br>
<li><strong>The hyper paramter tuning (Consider any two hyper parameters preferably n_estimato
    <ul>
      <li> Consider the following range for hyperparameters <b>n_estimators</b> = [10, 50, 100
        <b>max_depth</b> = [2, 3, 4, 5, 6, 7, 8, 9, 10] </li>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Find the best hyper paramter using simple cross validation data</li>
<li>You can write your own for loops to do this task</li>
```

```
    </ul>
</li>
<br>
<li>
<strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='3d_plot.JPG' width=500px> with X-axis as <strong>n_estimators</strong>, Y-axis as <st
        <p style="text-align:center;font-size:30px;color:red;"><strong>or</strong></p> <br>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='heat_map.JPG' width=300px> <a href='https://seaborn.pydata.org/generated/seaborn.heat
<li>You can choose either of the plotting techniques: 3d plot or heat map</li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
        </ul>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table for
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

Note: Data Leakage

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

2. Random Forest and GBDT

2.2 Make Data Model Ready: encoding numerical, categorical features

5 Set1 - BoW

In [73]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = CountVectorizer(min_df=10, ngram_range=(1,4), max_features=10000)
vectorizer.fit(train_preprocessed_essays) # fit has to happen only on train data
```

```
        # we use the fitted CountVectorizer to convert the text to vector
        X_train_essay_bow = vectorizer.transform(train_preprocessed_essays)
        X_test_essay_bow = vectorizer.transform(test_preprocessed_essays)

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


        # print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
        # vectorizer = CountVectorizer()
        # x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
        # x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
        # x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

        # print(x_train_bow.shape, y_train.shape)
        # print(x_cv_bow.shape, y_cv.shape)
        # print(x_test_bow.shape, y_test.shape)

        print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(40000, 20) (40000,)
(10000, 20) (10000,)
====================================================================================================
After vectorizations
(40000, 10000) (40000,)
(10000, 10000) (10000,)
====================================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

```
In [74]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)


         vectorizer = CountVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
         vectorizer.fit(train_preprocessed_titles) # fit has to happen only on train data

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_title_bow = vectorizer.transform(train_preprocessed_titles)
         X_test_title_bow = vectorizer.transform(test_preprocessed_titles)

         print("After vectorizations")
         print(X_train_title_bow.shape, y_train.shape)
```

```
print(X_test_title_bow.shape, y_test.shape)
print("="*100)


# print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
# vectorizer = CountVectorizer()
# x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
# x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
# x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

# print(x_train_bow.shape, y_train.shape)
# print(x_cv_bow.shape, y_cv.shape)
# print(x_test_bow.shape, y_test.shape)

print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(40000, 20) (40000,)
(10000, 20) (10000,)
==========================================================================================
After vectorizations
(40000, 2840) (40000,)
(10000, 2840) (10000,)
==========================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

6 Set2 - TfIdf

In [75]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = TfidfVectorizer(min_df=10, max_features=10000)
vectorizer.fit(train_preprocessed_essays) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tf = vectorizer.transform(train_preprocessed_essays)
X_test_essay_tf = vectorizer.transform(test_preprocessed_essays)

print("After vectorizations")
print(X_train_essay_tf.shape, y_train.shape)
print(X_test_essay_tf.shape, y_test.shape)
print("="*100)
```

```
        # print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
        # vectorizer = CountVectorizer()
        # x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
        # x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
        # x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

        # print(x_train_bow.shape, y_train.shape)
        # print(x_cv_bow.shape, y_cv.shape)
        # print(x_test_bow.shape, y_test.shape)

        print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(40000, 20) (40000,)
(10000, 20) (10000,)
==========================================================================================
After vectorizations
(40000, 10000) (40000,)
(10000, 10000) (10000,)
==========================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

```
In [76]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)


         vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=10000)
         vectorizer.fit(train_preprocessed_titles) # fit has to happen only on train data

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_title_tf = vectorizer.transform(train_preprocessed_titles)
         X_test_title_tf = vectorizer.transform(test_preprocessed_titles)

         print("After vectorizations")
         print(X_train_title_tf.shape, y_train.shape)
         print(X_test_title_tf.shape, y_test.shape)
         print("="*100)


         # print("YOU SHOULD NOT DO SOMETHING LIKE THIS")
         # vectorizer = CountVectorizer()
         # x_train_bow = vectorizer.fit_transform(X_train['essay'].values)
         # x_cv_bow = vectorizer.fit_transform(X_cv['essay'].values)
         # x_test_bow = vectorizer.fit_transform(X_test['essay'].values)

         # print(x_train_bow.shape, y_train.shape)
```

```
        # print(x_cv_bow.shape, y_cv.shape)
        # print(x_test_bow.shape, y_test.shape)

        print("NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME")
```

```
(40000, 20) (40000,)
(10000, 20) (10000,)
==========================================================================================
After vectorizations
(40000, 2840) (40000,)
(10000, 2840) (10000,)
==========================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

7 Set3 - Avg Word 2 Vec

```
In [0]: #https://stackoverflow.com/questions/21015674/list-object-has-no-attribute-shape
        #List to Numpy array
        #for Essays

        X_train_essay_avgw2v = np.array(train_avg_w2v_vectors)
        #X_cv_essay_avgw2v = np.array(cv_avg_w2v_vectors)
        X_test_essay_avgw2v = np.array(test_avg_w2v_vectors)

        #similarly, we are doing it for titles

        X_train_title_avgw2v = np.array(train_avg_w2v_vectors1)
        #X_cv_title_avgw2v = np.array(cv_avg_w2v_vectors1)
        X_test_title_avgw2v = np.array(test_avg_w2v_vectors1)
```

```
In [78]: #For Essays - Avgw2v
         print(X_train.shape, y_train.shape)
         #print(X_cv.shape, y_cv.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)

         print("After vectorizations")
         print(X_train_essay_avgw2v.shape, y_train.shape)
         #print(X_cv_essay_avgw2v.shape, y_cv.shape)
         print(X_test_essay_avgw2v.shape, y_test.shape)
         print("="*100)
```

```
(40000, 20) (40000,)
(10000, 20) (10000,)
==========================================================================================
After vectorizations
```

```
(40000, 300) (40000,)
(10000, 300) (10000,)
==========================================================================================
```

```
In [79]: #For Titles - Avgw2v
         print(X_train.shape, y_train.shape)
         #print(X_cv.shape, y_cv.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)

         print("After vectorizations")
         print(X_train_title_avgw2v.shape, y_train.shape)
         #print(X_cv_title_avgw2v.shape, y_cv.shape)
         print(X_test_title_avgw2v.shape, y_test.shape)
         print("="*100)
```

```
(40000, 20) (40000,)
(10000, 20) (10000,)
==========================================================================================
After vectorizations
(40000, 300) (40000,)
(10000, 300) (10000,)
==========================================================================================
```

8 Set4 -TfIdf Weighted W2vec

```
In [0]: #https://stackoverflow.com/questions/21015674/list-object-has-no-attribute-shape
        #List to Numpy array
        #for Essays

        X_train_es_tfidf_w2v = np.array(train_tfidf_w2v_vectors)
        #X_cv_es_tfidf_w2v = np.array(cv_tfidf_w2v_vectors)
        X_test_es_tfidf_w2v = np.array(test_tfidf_w2v_vectors)

        #similarly, we are doing it for titles

        X_train_title_tfidf_w2v = np.array(train_tfidf_w2v_vectors1)
        #X_cv_title_tfidf_w2v = np.array(cv_tfidf_w2v_vectors1)
        X_test_title_tfidf_w2v = np.array(test_tfidf_w2v_vectors1)
```

```
In [81]: #For Essays - TfIdf weighted W2vec
         print(X_train.shape, y_train.shape)
         #print(X_cv.shape, y_cv.shape)
         print(X_test.shape, y_test.shape)
```

```
print("="*100)

print("After vectorizations")
print(X_train_es_tfidf_w2v.shape, y_train.shape)
#print(X_cv_es_tfidf_w2v.shape, y_cv.shape)
print(X_test_es_tfidf_w2v.shape, y_test.shape)
print("="*100)
```

```
(40000, 20) (40000,)
(10000, 20) (10000,)
====================================================================================================
After vectorizations
(40000, 300) (40000,)
(10000, 300) (10000,)
====================================================================================================
```

In [82]:
```
#For Titles - TfIdf Weighted W2Vec
print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

print("After vectorizations")
print(X_train_title_tfidf_w2v.shape, y_train.shape)
#print(X_cv_title_tfidf_w2v.shape, y_cv.shape)
print(X_test_title_tfidf_w2v.shape, y_test.shape)
print("="*100)
```

```
(40000, 20) (40000,)
(10000, 20) (10000,)
====================================================================================================
After vectorizations
(40000, 300) (40000,)
(10000, 300) (10000,)
====================================================================================================
```

In [0]:
```
# Concatinating all the features
```

In [0]:
```
#BoW
```

In [83]:
```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense mati
X_Bow_train = hstack((X_train_essay_bow, X_train_title_bow, tr_price_standardized, tr
                      tr_prefix, tr_school_state, tr_project_grade, tr_clean_subcateg


print(X_Bow_train.shape, y_train.shape)
```

```
(40000, 12852) (40000,)
```

```
In [84]: X_Bow_test = hstack((X_test_essay_bow, X_test_title_bow, te_price_standardized, te_qua
                              te_prefix, te_school_state, te_project_grade, te_clean_subcategor

         print(X_Bow_test.shape, y_test.shape)
```

```
(10000, 12852) (10000,)
```

```
In [0]: #tfIdf
```

```
In [85]: from scipy.sparse import hstack
         # with the same hstack function we are concatinating a sparse matrix and a dense mati
         X_tf_train = hstack((X_train_essay_tf, X_train_title_tf, tr_price_standardized, tr_qua
                              tr_prefix, tr_school_state, tr_project_grade, tr_clean_subcategor


         print(X_tf_train.shape, y_train.shape)
```

```
(40000, 12852) (40000,)
```

```
In [86]: X_tf_test = hstack((X_test_essay_tf, X_test_title_tf, te_price_standardized, te_quanti
                             te_prefix, te_school_state, te_project_grade, te_clean_subcategori

         print(X_tf_test.shape, y_test.shape)
```

```
(10000, 12852) (10000,)
```

```
In [0]: #Avg W2V
```

```
In [87]: from scipy.sparse import hstack
         # with the same hstack function we are concatinating a sparse matrix and a dense mati
         X_avg_w2v_train = np.hstack((X_train_essay_avgw2v, X_train_title_avgw2v, tr_price_star
                                      tr_prefix, tr_school_state, tr_project_grade, tr_clean_subca


         print(X_avg_w2v_train.shape, y_train.shape)
```

```
(40000, 612) (40000,)
```

```
In [88]: X_avg_w2v_test = np.hstack((X_test_essay_avgw2v, X_test_title_avgw2v, te_price_standar
                                    te_prefix, te_school_state, te_project_grade, te_clean_subcat

         print(X_avg_w2v_test.shape, y_test.shape)
```

```
(10000, 612) (10000,)
```

```
In [0]: #TfIdf W2V
```

```
In [89]: from scipy.sparse import hstack
         # with the same hstack function we are concatinating a sparse matrix and a dense mati
         X_tf_w2v_train = np.hstack((X_train_es_tfidf_w2v, X_train_title_tfidf_w2v, tr_price_st
                                     tr_prefix, tr_school_state, tr_project_grade, tr_clean_subcat


         print(X_tf_w2v_train.shape, y_train.shape)
```

```
(40000, 612) (40000,)
```

```
In [90]: X_tf_w2v_test = np.hstack((X_test_es_tfidf_w2v, X_test_title_tfidf_w2v, te_price_stand
                                    te_prefix, te_school_state, te_project_grade, te_clean_subcate

         print(X_tf_w2v_test.shape, y_test.shape)
```

```
(10000, 612) (10000,)
```

2.5 Applying LightGBM

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

8.0.1 2.4.1 Applying LightGBM on BOW, SET 1

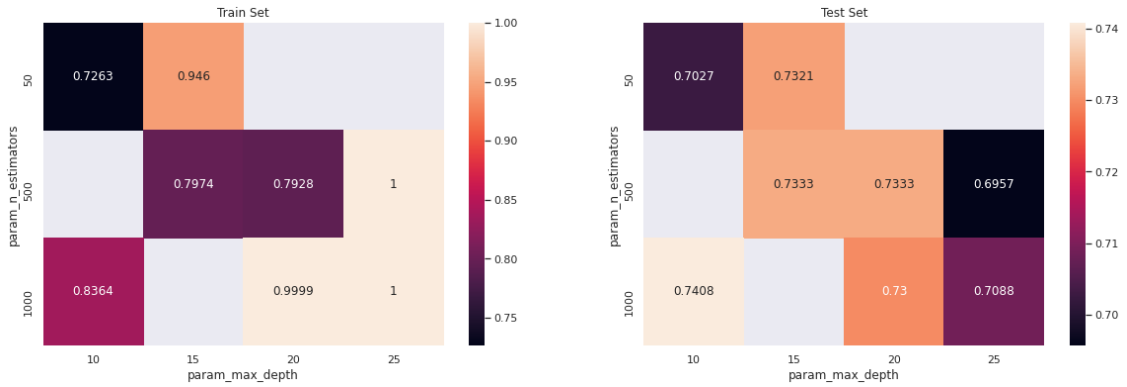
```
In [91]: from lightgbm import LGBMClassifier
         from sklearn.model_selection import RandomizedSearchCV
         param = {'n_estimators':  [50,100,200,500,1000] ,
                     'max_depth' :  [10,15,20,25] ,
                     'reg_lambda':  [0.05,0.5,0,1,2] ,
                     'reg_alpha' :  [0.05,0.5,0,1,2] ,
                     'learning_rate':  [0.005,0.05,0.5,0.1]}


         estimator = LGBMClassifier(objective = "binary" ,eval_metric= 'auc',class_weight = "ba
         clf= RandomizedSearchCV(estimator,param_distributions=param,scoring='roc_auc', cv=5, r
         clf.fit(X_Bow_train,y_train)
         clf.best_params_, clf.best_score_
```

```
Out[91]: ({'learning_rate': 0.005,
           'max_depth': 10,
           'n_estimators': 1000,
           'reg_alpha': 2,
           'reg_lambda': 1},
          0.7408027631164583)
```

In [92]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearch
a=clf.best_params_['n_estimators']
p = clf.best_params_['max_depth']
q = clf.best_params_['reg_lambda']
r = clf.best_params_['reg_alpha']
s = clf.best_params_['learning_rate']
print(clf.best_score_)
print(a)
print(p)
print(q)
print(r)
print(s)
```

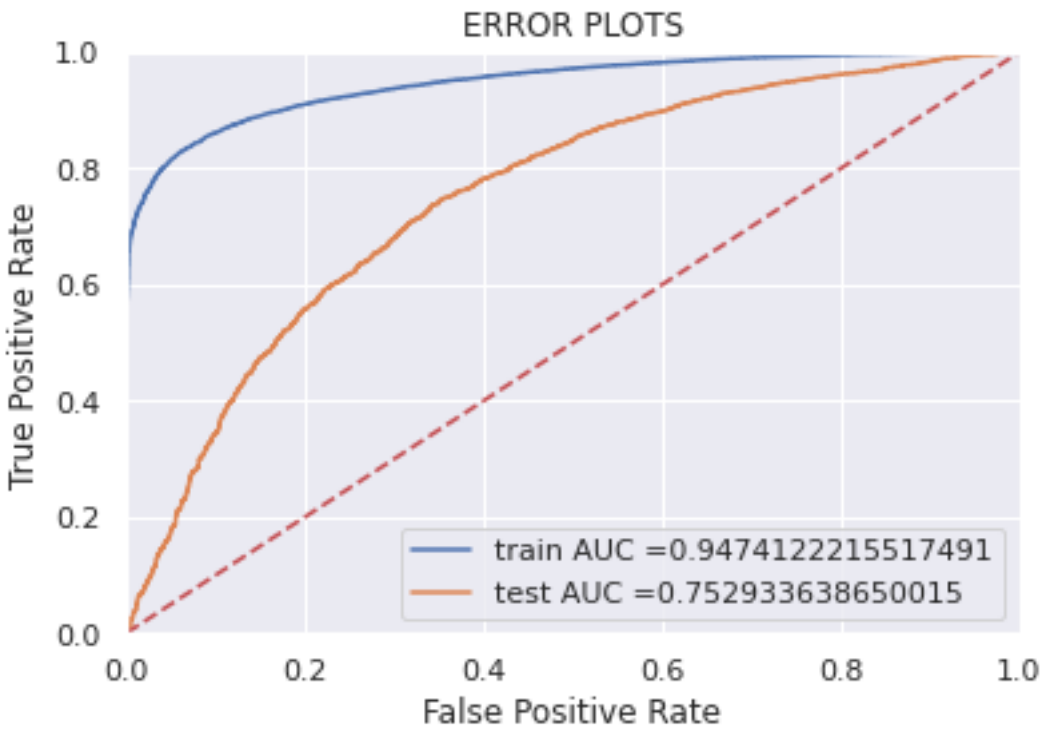
```
0.7408027631164583
1000
10
1
2
0.005
```

In [94]:
```
#https://towardsdatascience.com/using-3d-visualizations-to-tune-hyperparameters-of-ml
#https://github.com/xoelop/Medium-posts/blob/master/3d%20cross%20validation/ML%206%20
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
plt.show()
```



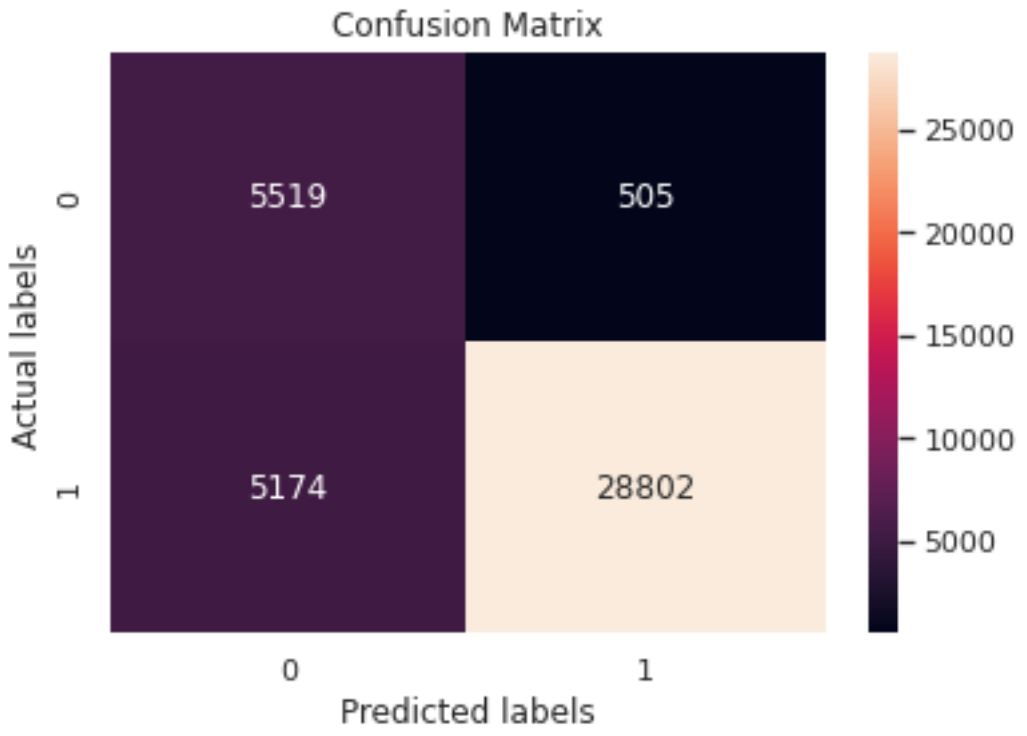
In [0]:
```
# Train new model
LG_BoW = LGBMClassifier(max_depth=p, n_estimators=a, learning_rate = s, reg_lambda = q
LG_BoW.fit(X_Bow_train,y_train)
```

```
Out[0]: LGBMClassifier(boosting_type='gbdt', class_weight='balanced',
                       colsample_bytree=1.0, importance_type='split',
                       learning_rate=0.05, max_depth=25, min_child_samples=20,
                       min_child_weight=0.001, min_split_gain=0.0, n_estimators=500,
                       n_jobs=-1, num_leaves=31, objective=None, random_state=None,
                       reg_alpha=2, reg_lambda=2, silent=True, subsample=1.0,
                       subsample_for_bin=200000, subsample_freq=0)

In [0]: #https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.h
        y_train_pred1 = LG_BoW.predict_proba(X_Bow_train) [:,1]
        y_test_pred1 = LG_BoW.predict_proba(X_Bow_test) [:,1]

        train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
        test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)
        plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)))
        plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))

        plt.plot([0, 1], [0, 1],'r--')
        plt.xlim([0, 1])
        plt.ylim([0, 1])

        plt.legend()
        plt.xlabel("False Positive Rate")
        plt.ylabel("True Positive Rate")
        plt.title("ERROR PLOTS")
        plt.grid(True)
        plt.show()
```



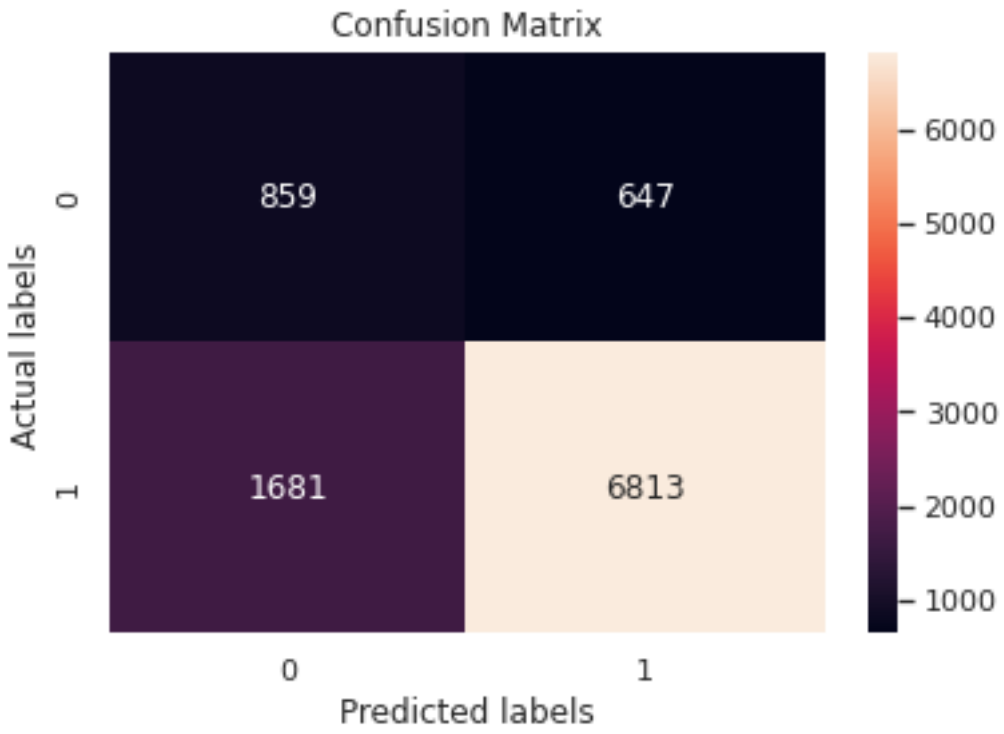
In [0]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#Train dataset - COnfusion Matrix
import seaborn as sns
import matplotlib.pyplot as plt
print("Confusin Matrix On train")
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, LG_BoW.predict(X_Bow_train)), annot=True, ax = ax

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('Actual labels');
ax.set_title('Confusion Matrix');
```

```
Confusin Matrix On train
```



In [0]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#Train dataset - COnfusion Matrix
import seaborn as sns
import matplotlib.pyplot as plt
print("Confusin Matrix On test")
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, LG_BoW.predict(X_Bow_test)), annot=True, ax = ax,

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('Actual labels');
ax.set_title('Confusion Matrix');
```

```
Confusin Matrix On test
```



8.0.2 2.4.2 Applying LightGBM on TFIDF, SET 2

```
In [0]: # Please write all the code with proper documentation
```

```
In [96]: from lightgbm import LGBMClassifier
         from sklearn.model_selection import RandomizedSearchCV
         param1 = {'n_estimators':  [50,100,200,500,1000] ,
                   'max_depth' :  [10,15,20,25] ,
                   'reg_lambda':  [0.05,0.5,0,1,2] ,
                   'reg_alpha' :  [0.05,0.5,0,1,2] ,
                   'learning_rate':  [0.005,0.05,0.5,0.1]}


         estimator1 = LGBMClassifier(objective = "binary" ,eval_metric= 'auc',class_weight = "
         clf1= RandomizedSearchCV(estimator1, param_distributions=param1, scoring='roc_auc', cv
         clf1.fit(X_tf_train,y_train)
```

```
Out[96]: RandomizedSearchCV(cv=5, error_score=nan,
                            estimator=LGBMClassifier(boosting_type='gbdt',
                                                     class_weight='balanced',
                                                     colsample_bytree=1.0,
                                                     eval_metric='auc',
                                                     importance_type='split',
```

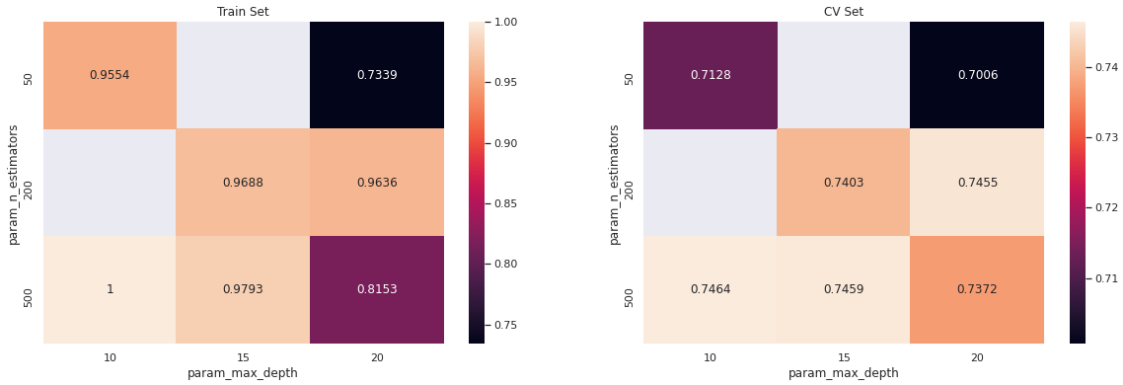
```
                                        learning_rate=0.1, max_depth=-1,
                                        min_child_samples=20,
                                        min_child_weight=0.001,
                                        min_split_gain=0.0,
                                        n_estimators=100, n_jobs=-1,
                                        num_leaves=31, objective='binary',
                                        random_state=None, reg_a...
                                        subsample_for_bin=200000,
                                        subsample_freq=0),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'learning_rate': [0.005, 0.05, 0.5,
                                                          0.1],
                                        'max_depth': [10, 15, 20, 25],
                                        'n_estimators': [50, 100, 200, 500,
                                                         1000],
                                        'reg_alpha': [0.05, 0.5, 0, 1, 2],
                                        'reg_lambda': [0.05, 0.5, 0, 1, 2]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='roc_auc', verbose=0)
```

In [97]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearch
a1=clf.best_params_['n_estimators']
p1 = clf.best_params_['max_depth']
q1 = clf.best_params_['reg_lambda']
r1 = clf.best_params_['reg_alpha']
s1 = clf.best_params_['learning_rate']
print(clf.best_score_)
print(a1)
print(p1)
print(q1)
print(r1)
print(s1)
```

```
0.7408027631164583
1000
10
1
2
0.005
```

In [98]:
```
#https://towardsdatascience.com/using-3d-visualizations-to-tune-hyperparameters-of-ml
#https://github.com/xoelop/Medium-posts/blob/master/3d%20cross%20validation/ML%206%20
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf1.cv_results_).groupby(['param_n_estimators', 'param_ma
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
```

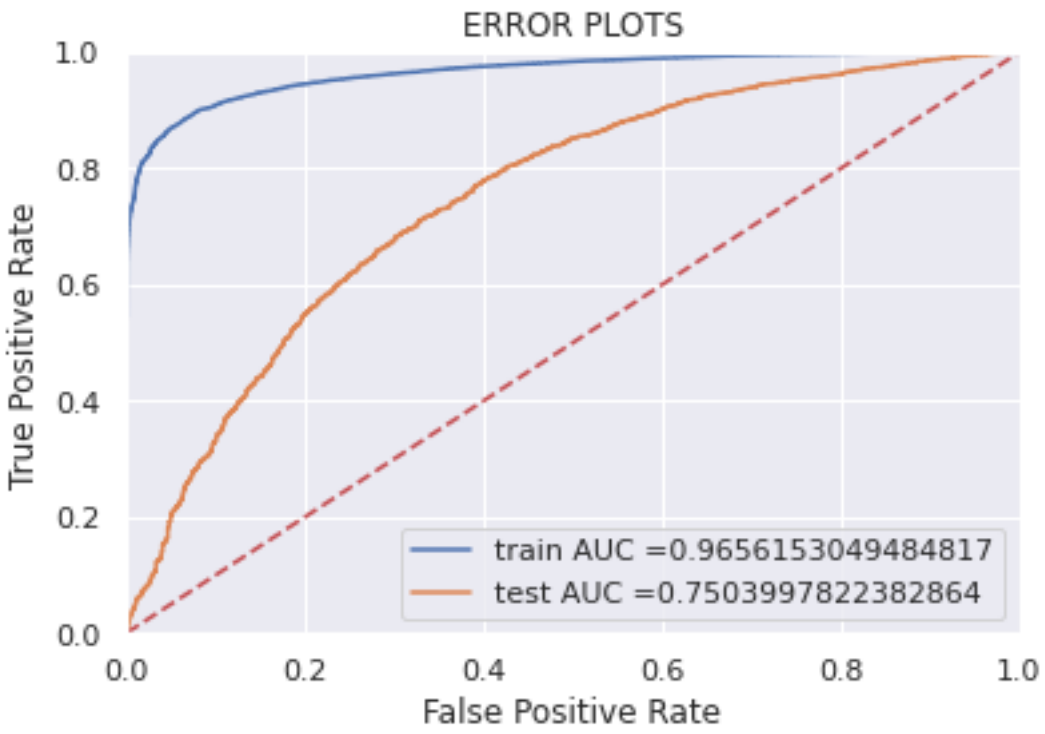
```
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [0]:
```
# Train new model
LG_tf = LGBMClassifier(max_depth=p1, n_estimators=a1, learning_rate = s1, reg_lambda =
LG_tf.fit(X_tf_train,y_train)
```

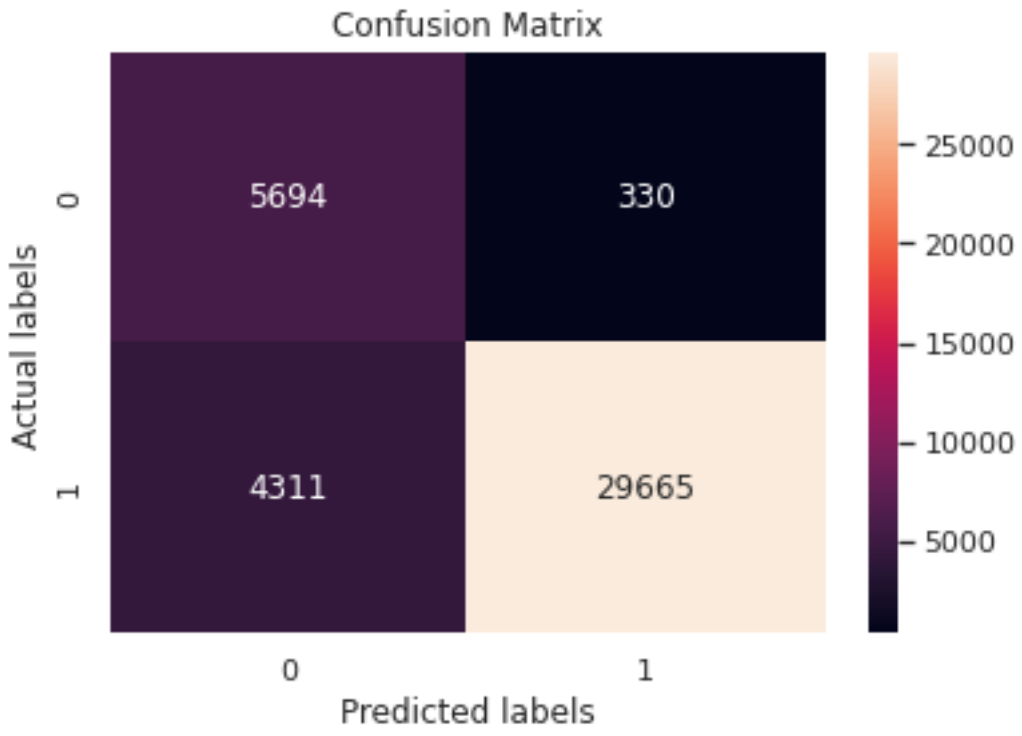
Out[0]:
```
LGBMClassifier(boosting_type='gbdt', class_weight='balanced',
               colsample_bytree=1.0, importance_type='split',
               learning_rate=0.05, max_depth=25, min_child_samples=20,
               min_child_weight=0.001, min_split_gain=0.0, n_estimators=500,
               n_jobs=-1, num_leaves=31, objective=None, random_state=None,
               reg_alpha=2, reg_lambda=2, silent=True, subsample=1.0,
               subsample_for_bin=200000, subsample_freq=0)
```

In [0]:
```
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.h
y_train_pred1 = LG_tf.predict_proba(X_tf_train) [:,1]
y_test_pred1 = LG_tf.predict_proba(X_tf_test) [:,1]

train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)
plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))
plt.legend()

plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid(True)
plt.show()
```



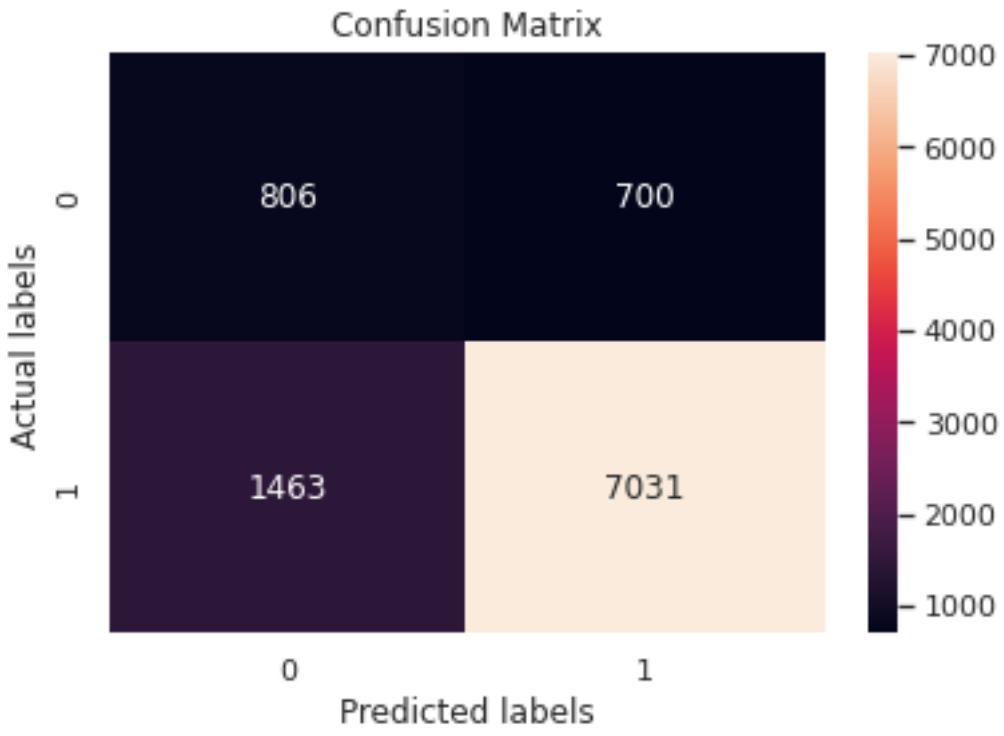
In [0]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#Train dataset - COnfusion Matrix
import seaborn as sns
import matplotlib.pyplot as plt
print("Confusin Matrix On train")
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, LG_tf.predict(X_tf_train)), annot=True, ax = ax,

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('Actual labels');
ax.set_title('Confusion Matrix');
```

```
Confusin Matrix On train
```



In [0]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#Train dataset - COnfusion Matrix
import seaborn as sns
import matplotlib.pyplot as plt
print("Confusin Matrix On test")
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, LG_tf.predict(X_tf_test)), annot=True, ax = ax,fm

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('Actual labels');
ax.set_title('Confusion Matrix');
```

```
Confusin Matrix On test
```



2.4.3 Applying LightGBM on Avg Word 2 Vec, SET 3

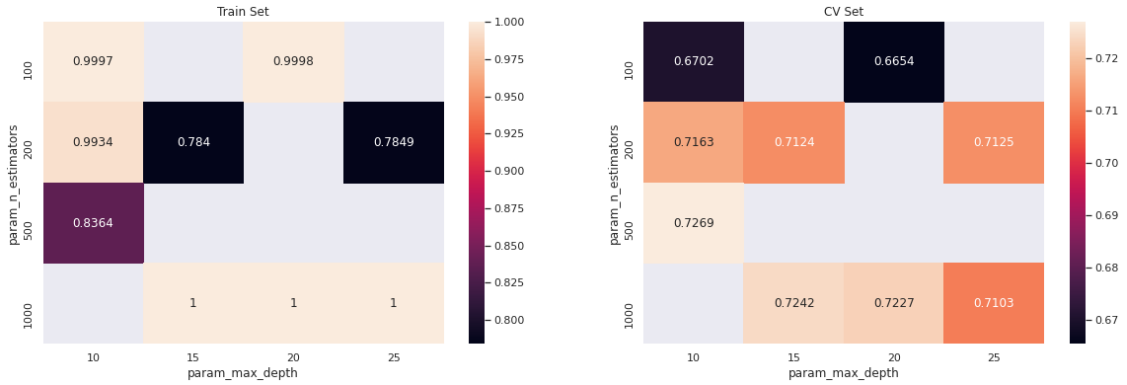
```
In [99]: from lightgbm import LGBMClassifier
         from sklearn.model_selection import RandomizedSearchCV
         param2 = {'n_estimators':  [50,100,200,500,1000] ,
                   'max_depth' :  [10,15,20,25] ,
                   'reg_lambda':  [0.05,0.5,0,1,2] ,
                   'reg_alpha' :  [0.05,0.5,0,1,2] ,
                   'learning_rate':  [0.005,0.05,0.5,0.1]}


         estimator2 = LGBMClassifier(objective = "binary" ,eval_metric= 'auc',class_weight = "
         clf2= RandomizedSearchCV(estimator2, param_distributions=param2, scoring='roc_auc', cv
         clf2.fit(X_avg_w2v_train,y_train)

Out[99]: RandomizedSearchCV(cv=5, error_score=nan,
                            estimator=LGBMClassifier(boosting_type='gbdt',
                                                     class_weight='balanced',
                                                     colsample_bytree=1.0,
                                                     eval_metric='auc',
                                                     importance_type='split',
                                                     learning_rate=0.1, max_depth=-1,
                                                     min_child_samples=20,
                                                     min_child_weight=0.001,
```

```
                                        min_split_gain=0.0,
                                        n_estimators=100, n_jobs=-1,
                                        num_leaves=31, objective='binary',
                                        random_state=None, reg_a...
                                        subsample_for_bin=200000,
                                        subsample_freq=0),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'learning_rate': [0.005, 0.05, 0.5,
                                                          0.1],
                                       'max_depth': [10, 15, 20, 25],
                                       'n_estimators': [50, 100, 200, 500,
                                                        1000],
                                       'reg_alpha': [0.05, 0.5, 0, 1, 2],
                                       'reg_lambda': [0.05, 0.5, 0, 1, 2]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='roc_auc', verbose=0)
```

In [106]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearc
a2=clf2.best_params_['n_estimators']
p2 = clf2.best_params_['max_depth']
q2 = clf2.best_params_['reg_lambda']
r2 = clf2.best_params_['reg_alpha']
s2 = clf2.best_params_['learning_rate']
print(clf2.best_score_)
print(a2)
print(p2)
print(q2)
print(r2)
print(s2)
```

```
0.7269405015853195
500
10
0
0.5
0.005
```

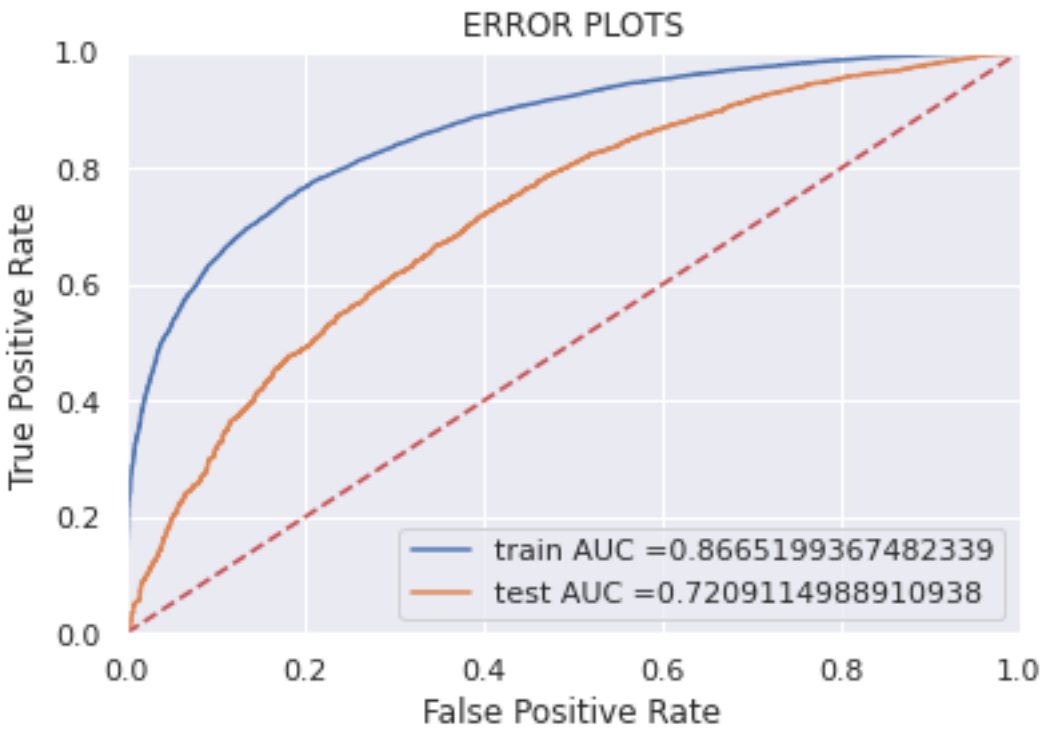
In [101]:
```
#https://towardsdatascience.com/using-3d-visualizations-to-tune-hyperparameters-of-m
#https://github.com/xoelop/Medium-posts/blob/master/3d%20cross%20validation/ML%206%2
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf2.cv_results_).groupby(['param_n_estimators', 'param_ma
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [102]:
```
# Train new model
LG_avg_w2v = LGBMClassifier(max_depth=p2, n_estimators=a2, learning_rate = s2, reg_la
LG_avg_w2v.fit(X_avg_w2v_train,y_train)
```

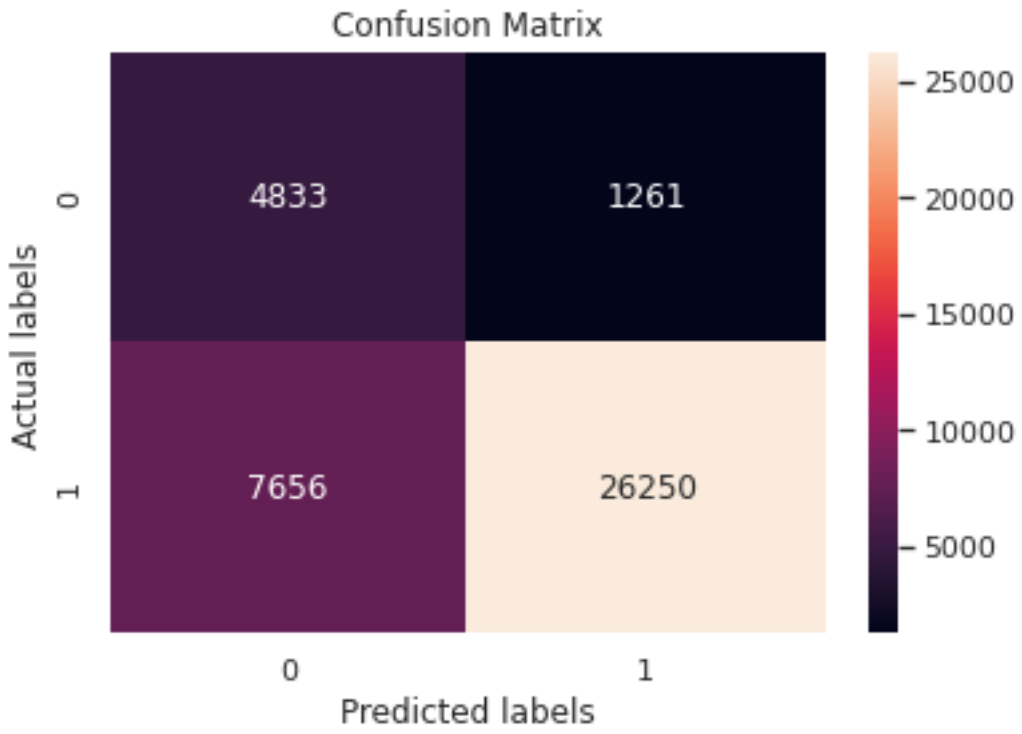
Out[102]:
```
LGBMClassifier(boosting_type='gbdt', class_weight='balanced',
               colsample_bytree=1.0, importance_type='split',
               learning_rate=0.005, max_depth=10, min_child_samples=20,
               min_child_weight=0.001, min_split_gain=0.0, n_estimators=1000,
               n_jobs=-1, num_leaves=31, objective=None, random_state=None,
               reg_alpha=2, reg_lambda=1, silent=True, subsample=1.0,
               subsample_for_bin=200000, subsample_freq=0)
```

In [103]:
```
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier
y_train_pred1 = LG_avg_w2v.predict_proba(X_avg_w2v_train) [:,1]
y_test_pred1 = LG_avg_w2v.predict_proba(X_avg_w2v_test) [:,1]

train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)
plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))
plt.legend()

plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid(True)
plt.show()
```



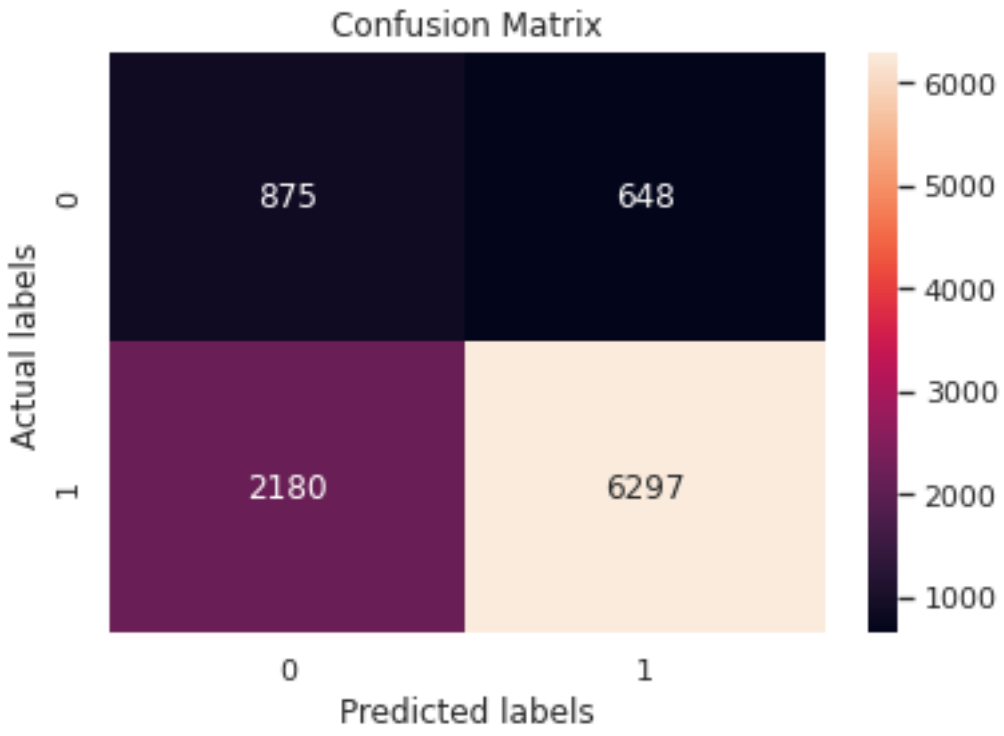
In [104]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#Train dataset - COnfusion Matrix
import seaborn as sns
import matplotlib.pyplot as plt
print("Confusin Matrix On train")
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, LG_avg_w2v.predict(X_avg_w2v_train)), annot=Tru

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('Actual labels');
ax.set_title('Confusion Matrix');
```

```
Confusin Matrix On train
```



In [105]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#Train dataset - COnfusion Matrix
import seaborn as sns
import matplotlib.pyplot as plt
print("Confusin Matrix On test")
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, LG_avg_w2v.predict(X_avg_w2v_test)), annot=True

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('Actual labels');
ax.set_title('Confusion Matrix');
```

```
Confusin Matrix On test
```



2.4.4 Applying LightGBM on weighted tfidf Word 2 Vec, SET 4

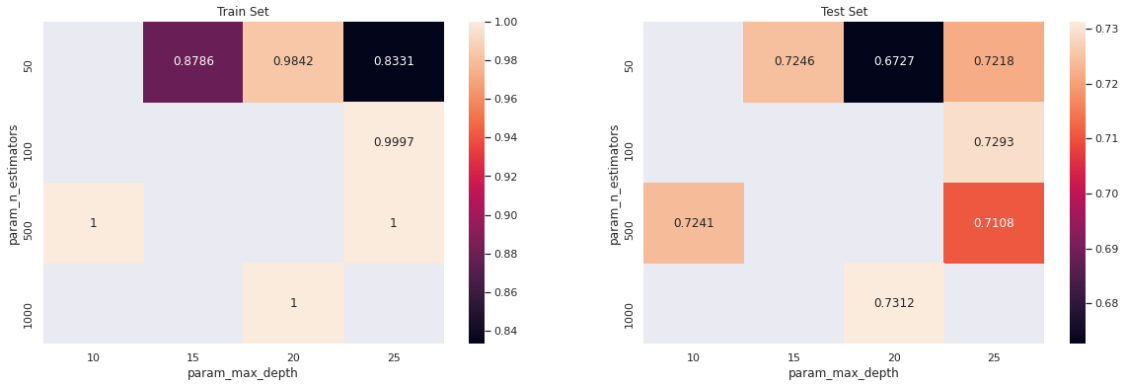
```
In [107]: from lightgbm import LGBMClassifier
          from sklearn.model_selection import RandomizedSearchCV
          param3 = {'n_estimators':  [50,100,200,500,1000] ,
                    'max_depth' :  [10,15,20,25] ,
                    'reg_lambda':  [0.05,0.5,0,1,2] ,
                    'reg_alpha' :  [0.05,0.5,0,1,2] ,
                    'learning_rate':  [0.005,0.05,0.5,0.1]}


          estimator3 = LGBMClassifier(objective = "binary" ,eval_metric= 'auc',class_weight = '
          clf3= RandomizedSearchCV(estimator3, param_distributions=param3, scoring='roc_auc', c
          clf3.fit(X_tf_w2v_train,y_train)

Out[107]: RandomizedSearchCV(cv=5, error_score=nan,
                             estimator=LGBMClassifier(boosting_type='gbdt',
                                                      class_weight='balanced',
                                                      colsample_bytree=1.0,
                                                      eval_metric='auc',
                                                      importance_type='split',
                                                      learning_rate=0.1, max_depth=-1,
                                                      min_child_samples=20,
                                                      min_child_weight=0.001,
```

```
                                        min_split_gain=0.0,
                                        n_estimators=100, n_jobs=-1,
                                        num_leaves=31, objective='binary',
                                        random_state=None, reg_a...
                                        subsample_for_bin=200000,
                                        subsample_freq=0),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'learning_rate': [0.005, 0.05, 0.5,
                                                          0.1],
                                       'max_depth': [10, 15, 20, 25],
                                       'n_estimators': [50, 100, 200, 500,
                                                        1000],
                                       'reg_alpha': [0.05, 0.5, 0, 1, 2],
                                       'reg_lambda': [0.05, 0.5, 0, 1, 2]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='roc_auc', verbose=0)
```

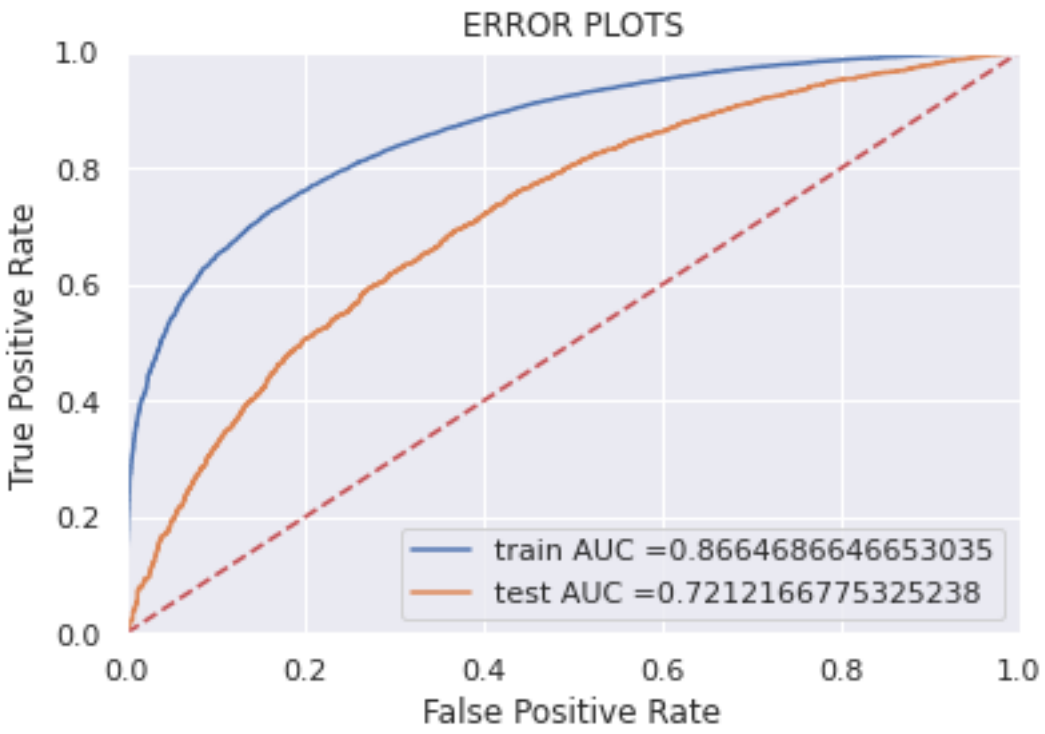
In [108]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearc
a3=clf3.best_params_['n_estimators']
p3 = clf3.best_params_['max_depth']
q3 = clf3.best_params_['reg_lambda']
r3 = clf3.best_params_['reg_alpha']
s3 = clf3.best_params_['learning_rate']
print(clf3.best_score_)
print(a3)
print(p3)
print(q3)
print(r3)
print(s3)
```

```
0.7312320934778613
1000
20
2
0.05
0.005
```

In [109]:
```
#https://towardsdatascience.com/using-3d-visualizations-to-tune-hyperparameters-of-m
#https://github.com/xoelop/Medium-posts/blob/master/3d%20cross%20validation/ML%206%2
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf3.cv_results_).groupby(['param_n_estimators', 'param_ma
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
plt.show()
```

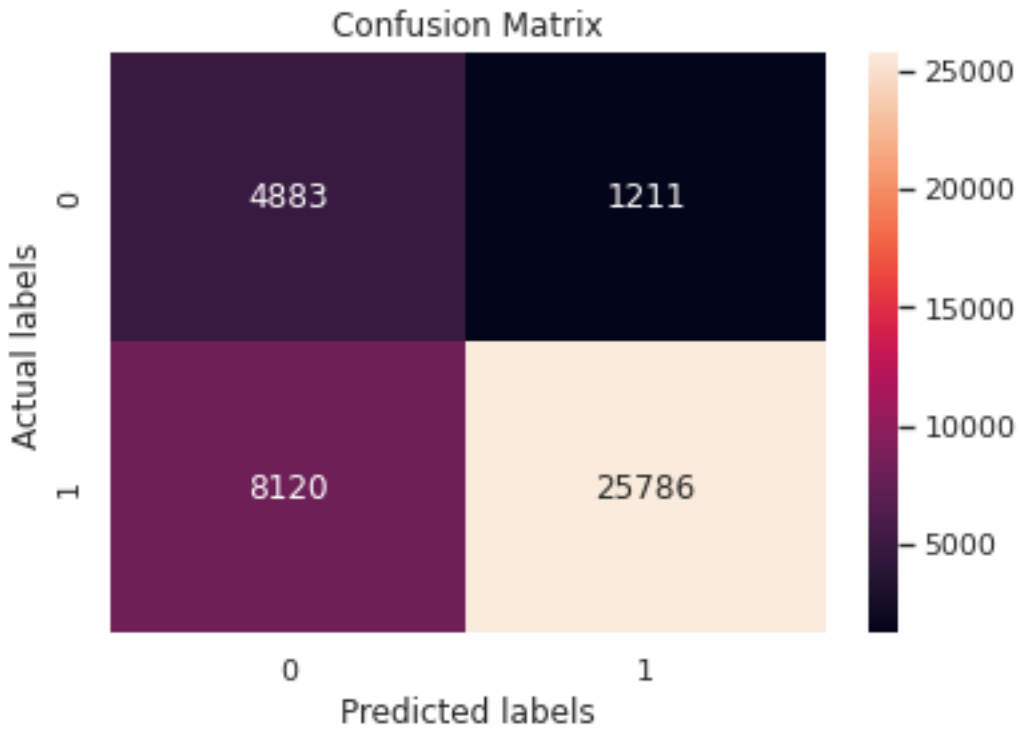



```
In [110]: # Train new model
          LG_tf_w2v = LGBMClassifier(max_depth=p3, n_estimators=a3, learning_rate = s3, reg_la
          LG_tf_w2v.fit(X_tf_w2v_train,y_train)

Out[110]: LGBMClassifier(boosting_type='gbdt', class_weight='balanced',
                         colsample_bytree=1.0, importance_type='split',
                         learning_rate=0.005, max_depth=20, min_child_samples=20,
                         min_child_weight=0.001, min_split_gain=0.0, n_estimators=1000,
                         n_jobs=-1, num_leaves=31, objective=None, random_state=None,
                         reg_alpha=0.05, reg_lambda=2, silent=True, subsample=1.0,
                         subsample_for_bin=200000, subsample_freq=0)

In [111]: #https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier
          y_train_pred1 = LG_tf_w2v.predict_proba(X_tf_w2v_train) [:,1]
          y_test_pred1 = LG_tf_w2v.predict_proba(X_tf_w2v_test) [:,1]

          train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
          test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)
          plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)
          plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))
          plt.legend()

          plt.plot([0, 1], [0, 1],'r--')
          plt.xlim([0, 1])
          plt.ylim([0, 1])

          plt.xlabel("False Positive Rate")
          plt.ylabel("True Positive Rate")
          plt.title("ERROR PLOTS")
          plt.grid(True)
          plt.show()
```



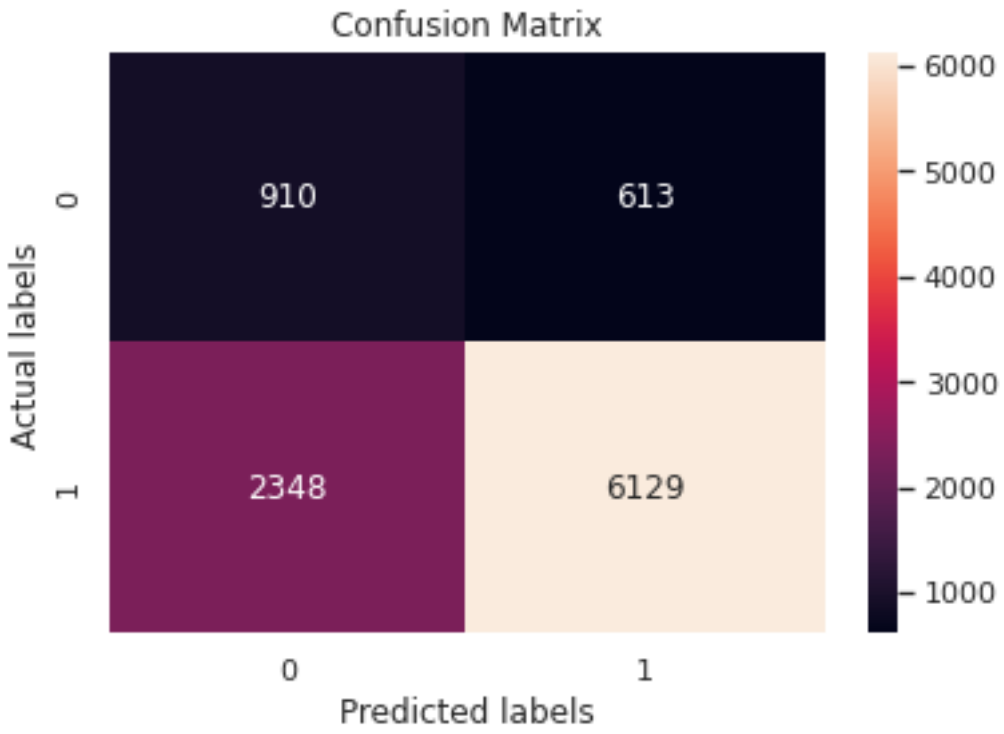
In [112]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#Train dataset - COnfusion Matrix
import seaborn as sns
import matplotlib.pyplot as plt
print("Confusin Matrix On train")
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, LG_tf_w2v.predict(X_tf_w2v_train)), annot=True

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('Actual labels');
ax.set_title('Confusion Matrix');
```

```
Confusin Matrix On train
```



In [113]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#Train dataset - COnfusion Matrix
import seaborn as sns
import matplotlib.pyplot as plt
print("Confusin Matrix On test")
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, LG_tf_w2v.predict(X_tf_w2v_test)), annot=True, a

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('Actual labels');
ax.set_title('Confusion Matrix');
```

```
Confusin Matrix On test
```



3. Conclusion

```
In [0]: # Please compare all your models using Prettytable library
```

```
In [124]: # Please compare all your models using Prettytable library
          from prettytable import PrettyTable

          p = PrettyTable()

          p.field_names = ["Vetorizer", "Model", "n_estimators","max_depth","Test AUC"]

          p.add_row(["SET 1- Bow","LGBM","1000","10","0.752"])
          p.add_row(["SET 2- TF_IDF","LGBM","1000","10","0.750"])
          p.add_row(["SET 3- AVG w2v","LGBM","500","10","0.720" ])
          p.add_row(["SET 4- TF_IDF w2v","LGBM","1000","20","0.721" ])

          print(p)
```

```
+-------------------+-------+--------------+-----------+----------+
|     Vetorizer     | Model | n_estimators | max_depth | Test AUC |
+-------------------+-------+--------------+-----------+----------+
|     SET 1- Bow    |  LGBM |     1000     |     10    |  0.752   |
|   SET 2- TF_IDF   |  LGBM |     1000     |     10    |  0.750   |
|   SET 3- AVG w2v  |  LGBM |     500      |     10    |  0.720   |
```

SET 4- TF_IDF w2v	LGBM	1000	20	0.721