7_DonorsChoose_SVM

March 16, 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 [1]: %matplotlib inline
        import warnings
        warnings.filterwarnings("ignore")

        import sqlite3
        import pandas as pd
```

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

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

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

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

from tqdm import tqdm
import os


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

```
In [2]: 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
```

1.2 1.1 Reading Data

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

```
Number of data points in train data (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 [5]: 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[5]:

	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)
```

Join train & Resource dataset

```
In [9]: price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_
        project_data = pd.merge(project_data, price_data, on='id', how='left')
        project_data.head(2)
```

```
Out[9]:    Unnamed: 0       id  ...  price quantity
        0      160221  p253737  ...  154.6       23
        1      140945  p258326  ...  299.0        1

        [2 rows x 19 columns]
```

Train Test split

```
In [139]: y = project_data['project_is_approved'].values
          X = project_data.drop(['project_is_approved'], axis=1)
          X.head(1)
```

```
Out[139]:    Unnamed: 0       id  ... essay_len title_len
          0      160221  p253737  ...      1121        41

          [1 rows x 27 columns]
```

```
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)
```

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

```
Out[13]:    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 20 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)
         print(project_data['essay'].values[99999])
         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
==================================================
The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates. The g
==================================================
```

```
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-p
         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 [01:00<00:00, 1805.78it/s]
```

```
In [60]: project_data['preprocessed_essays'] = preprocessed_essays
```

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

```
Out[60]:    Unnamed: 0  ...                                preprocessed_essays
         0      160221  ...  my students english learners working english s...
         1      140945  ...  our students arrive school eager learn they po...

         [2 rows x 21 columns]
```

```
In [24]: # 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%|| 87398/87398 [00:47<00:00, 1824.62it/s]
```

```
In [25]: # 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%|| 21850/21850 [00:12<00:00, 1815.42it/s]
```

```
In [26]: # after preprocesing
         preprocessed_essays[20000]
```

```
Out[26]: 'my kindergarten students varied disabilities ranging speech language delays cognitive
```

1.4 Preprocessing of project_title

```
In [0]: # similarly you can preprocess the titles also
```

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

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

```
In [29]: # \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)
```

```
Steady Stools for Active Learning
```

```
In [30]: #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 [32]: # 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, 42844.04it/s]
```

```
In [61]: project_data['preprocessed_titles'] = preprocessed_titles

         project_data.head(2)
```

```
Out[61]:    Unnamed: 0  ...                        preprocessed_titles
         0      160221  ...  educational support english learners home
         1      140945  ...           wanted projector hungry learners

         [2 rows x 22 columns]
```

```
In [33]: # 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%|| 87398/87398 [00:02<00:00, 43132.20it/s]
```

```
In [34]: # 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%|| 21850/21850 [00:00<00:00, 42357.41it/s]
```

```
In [35]: preprocessed_titles[2000]
```

```
Out[35]: 'steady stools active learning'
```

1.6 1.5 Preparing data for models

```
In [36]: project_data.columns
```

```
Out[36]: 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', 'price',
                'quantity', 'essay'],
               dtype='object')
```

we are going to consider

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

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

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

1.6.1 1.5.1 Vectorizing Categorical data

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

```
In [37]: # we use count vectorizer to convert the values into one
         from sklearn.feature_extraction.text import CountVectorizer
         vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False
         categories_one_hot = vectorizer.fit_transform(project_data['clean_categories'].values)
         print(vectorizer.get_feature_names())
         print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'I
Shape of matrix after one hot encodig  (109248, 9)
```

```
In [38]: # we use count vectorizer to convert the values into one
         vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=Fa
         sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subcategories']
         print(vectorizer.get_feature_names())
         print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
Shape of matrix after one hot encodig  (109248, 30)
```

```
In [0]: # you can do the similar thing with state, teacher_prefix and project_grade_category a

In [0]: # count of all the words in corpus python: https://stackoverflow.com/a/22898595/408403
        my_counter = Counter()
        for word in project_data['school_state'].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]))

In [40]: # we use count vectorizer to convert the values into one
         from sklearn.feature_extraction.text import CountVectorizer
         vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=Fa
         categories_one_hot = vectorizer.fit_transform(project_data['school_state'].values)
         print(vectorizer.get_feature_names())
         print("Shape of matrix after one hot encodig ",categories_one_hot.shape)

['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'WV', 'ME', 'HI', 'DC', 'NM', 'KS
Shape of matrix after one hot encodig  (109248, 51)


In [0]: # count of all the words in corpus python: https://stackoverflow.com/a/22898595/408403
        project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(" ")
        my_counter = Counter()
        for word in project_data['teacher_prefix'].values.astype('str'):  #https://stackoverfl
            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]))

In [42]: # we use count vectorizer to convert the values into one
         from sklearn.feature_extraction.text import CountVectorizer
         vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=Fa
         prefix_one_hot = vectorizer.fit_transform(project_data['teacher_prefix'].values.astype
         print(vectorizer.get_feature_names())
         print("Shape of matrix after one hot encodig ",prefix_one_hot.shape)

['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encodig  (109248, 5)


In [0]: # count of all the words in corpus python: https://stackoverflow.com/a/22898595/408403
        my_counter = Counter()
        for word in project_data['grade_cat_list'].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]))
```

```
In [44]: # we use count vectorizer to convert the values into one
         from sklearn.feature_extraction.text import CountVectorizer
         vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=Fa
         categories_one_hot = vectorizer.fit_transform(project_data['grade_cat_list'].values)
         print(vectorizer.get_feature_names())
         print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

```
['9-12', '6-8', '3-5', 'PreK-2']
Shape of matrix after one hot encodig  (109248, 4)
```

1.6.2 1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

```
In [45]: # 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 [0]: # you can vectorize the title also
        # before you vectorize the title make sure you preprocess it
```

```
In [46]: # 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 [47]: 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)
```

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 [49]: # 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 [00:33<00:00, 3254.18it/s]
```

```
109248
300
```

```
In [50]: # 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%|| 87398/87398 [00:24<00:00, 3522.89it/s]

87398
300
```

In [51]:
```
# 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%|| 21850/21850 [00:05<00:00, 3642.77it/s]

21850
300
```

In [52]:
```
# 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, 67513.69it/s]

109248
300
```

In [53]:
```
# 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%|| 87398/87398 [00:01<00:00, 69335.97it/s]

87398
300
```

In [54]:
```
# 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%|| 21850/21850 [00:00<00:00, 66311.55it/s]

21850
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(X_train['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 [56]: # 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 [03:32<00:00, 513.80it/s]

109248
300
```

```
In [57]: # 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%|| 87398/87398 [02:36<00:00, 557.12it/s]

87398
300
```

```
In [58]: # 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%|| 21850/21850 [00:39<00:00, 557.88it/s]

21850
300
```

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

```
In [0]: # Similarly you can vectorize for title also
    tfidf_model2 = TfidfVectorizer()
    tfidf_model2.fit(X_train['preprocessed_titles'])
    # we are converting a dictionary with word as a key, and the idf as a value
    dictionary = dict(zip(tfidf_model2.get_feature_names(), list(tfidf_model2.idf_)))
    tfidf_words = set(tfidf_model2.get_feature_names())
```

```
In [66]: # 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%|| 87398/87398 [00:02<00:00, 35310.50it/s]
```

```
87398
300
```

```
In [67]: # 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%|| 21850/21850 [00:00<00:00, 29331.68it/s]

21850
300
```

1.6.3 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))
        te_price_standardized = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
```

```
Mean : 298.1193425966608, Standard deviation : 367.49634838483496
```

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

```
Out[70]: array([[-0.3905327 ],
                [ 0.00239637],
                [ 0.59519138],
                ...,
                [-0.15825829],
                [-0.61243967],
                [-0.51216657]])
```

```
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.resha
         te_quantity_standardized = quantity_scalar.transform(X_test['quantity'].values.reshape
```

```
Mean : 16.965610354422964, Standard deviation : 26.182821919093175
```

```
In [72]: quantity_standardized
```

```
Out[72]: array([[ 0.23047132],
                [-0.60977424],
                [ 0.19227834],
                ...,
                [-0.4951953 ],
                [-0.03687954],
                [-0.45700232]])
```

```
In [73]: # 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)

         number_projects_scalar = StandardScaler()
         number_projects_scalar.fit(project_data['teacher_number_of_previously_posted_projects
         print(f"Mean : {number_projects_scalar.mean_[0]}, Standard deviation : {np.sqrt(numbe

         # Now standardize the data with above mean and variance.
         number_projects_standardized = number_projects_scalar.transform(project_data['teacher
         tr_number_projects_standardized = number_projects_scalar.transform(X_train['teacher_nu
         te_number_projects_standardized = number_projects_scalar.transform(X_test['teacher_nu

Mean : 11.153165275336848, Standard deviation : 27.77702641477403
```

__ 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 smalles
        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 schoo
        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 wit
        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 we
        of where the ingredients came from as well as how it is healthy for their bodies this
        nutrition and agricultural cooking recipes by having us peel our own apples to make ho
        and mix up healthy plants from our classroom garden in the spring we will also create
        shared with families students will gain math and literature skills as well as a life l
        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,
```

```
In [74]: import nltk
         nltk.download('vader_lexicon')

         from nltk.sentiment.vader import SentimentIntensityAnalyzer

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

         sid = SentimentIntensityAnalyzer()

         # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_essays = []
         neg=[]
         neut=[]
         pos=[]
         comp=[]
         # 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())
             ss = sid.polarity_scores(sent.lower().strip())
             neg.append(ss['neg'])
             neut.append(ss['neu'])
             pos.append(ss['pos'])
             comp.append(ss['compound'])
```

```
  0%|          | 0/109248 [00:00<?, ?it/s]
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...


100%|| 109248/109248 [04:30<00:00, 404.51it/s]


In [75]: # after preprocesing
         preprocessed_essays[20000]
         ss

Out[75]: {'compound': 0.9868, 'neg': 0.059, 'neu': 0.693, 'pos': 0.248}

In [0]: essays_len=[]
        titles_len=[]
        for x in range(len(preprocessed_essays)):  # we are using lenght of preprocessed essay
            essays_len.append(len(preprocessed_essays[x]))
            titles_len.append(len(preprocessed_titles[x]))

In [0]: project_data['neg']=neg
        project_data['neut']=neut
        project_data['pos']=pos
        project_data['comp']=comp
        project_data['essay_len']=essays_len
        project_data['title_len']=titles_len

In [78]: project_data.head(5)

Out[78]:    Unnamed: 0       id  ... essay_len title_len
         0      160221  p253737  ...      1121        41
         1      140945  p258326  ...       814        32
         2       21895  p182444  ...      1441        47
         3          45  p246581  ...       861        22
         4      172407  p104768  ...       812        22

         [5 rows x 28 columns]
```

2 Assignment 7: SVM

```
<li><strong>[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) o
    <ul>
        <li><font color='red'>Set 1</font>: categorical, numerical features + project_title(BOW
        <li><font color='red'>Set 2</font>: categorical, numerical features + project_title(TFI
        <li><font color='red'>Set 3</font>: categorical, numerical features + project_title(AVG
        <li><font color='red'>Set 4</font>: categorical, numerical features + project_title(TFI
</li>
<br>
<li><strong>The hyper paramter tuning (best alpha in range [10^-4 to 10^4], and the best penalt
    <ul>
```

```
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </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='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and 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>
</li>
<br>
<li><strong>[Task-2] Apply the Support Vector Machines on these features by finding the best hy
    <ul>
<li> Consider these set of features <font color='red'> Set 5 :</font>
        <ul>
            <li><strong>school_state</strong> : categorical data</li>
            <li><strong>clean_categories</strong> : categorical data</li>
            <li><strong>clean_subcategories</strong> : categorical data</li>
            <li><strong>project_grade_category</strong> :categorical data</li>
            <li><strong>teacher_prefix</strong> : categorical data</li>
            <li><strong>quantity</strong> : numerical data</li>
            <li><strong>teacher_number_of_previously_posted_projects</strong> : numerical data
            <li><strong>price</strong> : numerical data</li>
            <li><strong>sentiment score's of each of the essay</strong> : numerical data</li>
            <li><strong>number of words in the title</strong> : numerical data</li>
            <li><strong>number of words in the combine essays</strong> : numerical data</li>
            <li><strong>Apply <a href='http://scikit-learn.org/stable/modules/generated/sklearn
        </ul>
     <br>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table 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-leakage, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

2. Support Vector Machines

3 Encoding - Categorical

```
In [79]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)


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

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_cl_categories_ohe = vectorizer.transform(X_train['clean_categories'].values)
         X_test_cl_categories_ohe = vectorizer.transform(X_test['clean_categories'].values)

         print("After vectorizations")
         print(X_train_cl_categories_ohe.shape, y_train.shape)
         print(X_test_cl_categories_ohe.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")
```

```
(87398, 21) (87398,)
(21850, 21) (21850,)
====================================================================================================
After vectorizations
(87398, 9) (87398,)
(21850, 9) (21850,)
====================================================================================================
```

NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME

In [80]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on tra

# we use the fitted CountVectorizer to convert the text to vector
X_train_cl_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].va
X_test_cl_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].valu

print("After vectorizations")
print(X_train_cl_subcategories_ohe.shape, y_train.shape)
print(X_test_cl_subcategories_ohe.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")
```

```
(87398, 21) (87398,)
(21850, 21) (21850,)
====================================================================================================
After vectorizations
(87398, 30) (87398,)
(21850, 30) (21850,)
====================================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

In [81]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print("="*100)
```

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

        # we use the fitted CountVectorizer to convert the text to vector
        X_train_school_state_ohe = vectorizer.transform(X_train['school_state'].values)
        X_test_school_state_ohe = vectorizer.transform(X_test['school_state'].values)

        print("After vectorizations")
        print(X_train_school_state_ohe.shape, y_train.shape)
        print(X_test_school_state_ohe.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")
```

```
(87398, 21) (87398,)
(21850, 21) (21850,)
====================================================================================================
After vectorizations
(87398, 51) (87398,)
(21850, 51) (21850,)
====================================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

```
In [82]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)


         vectorizer = CountVectorizer()
         vectorizer.fit(X_train['teacher_prefix'].values.astype('U')) # fit has to happen only
         #https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueer
         # we use the fitted CountVectorizer to convert the text to vector
         X_train_teacher_prefix_ohe = vectorizer.transform(X_train['teacher_prefix'].values.as
```

```
X_test_teacher_prefix_ohe = vectorizer.transform(X_test['teacher_prefix'].values.asty

print("After vectorizations")
print(X_train_teacher_prefix_ohe.shape, y_train.shape)
print(X_test_teacher_prefix_ohe.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")
```

```
(87398, 21) (87398,)
(21850, 21) (21850,)
====================================================================================================
After vectorizations
(87398, 5) (87398,)
(21850, 5) (21850,)
====================================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

In [84]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=Fa
vectorizer.fit(X_train['grade_cat_list'].values) # fit has to happen only on train da

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

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.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")
```

```
(87398, 21) (87398,)
(21850, 21) (21850,)
==========================================================================================
After vectorizations
(87398, 4) (87398,)
(21850, 4) (21850,)
==========================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

2.2 Make Data Model Ready: encoding numerical, categorical features

4 Text - BOW

```
In [85]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)


         vectorizer = CountVectorizer(ngram_range=(2,2), min_df=10, max_features=5000)
         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")
```

```
(87398, 21) (87398,)
(21850, 21) (21850,)
==================================================================================================
After vectorizations
(87398, 5000) (87398,)
(21850, 5000) (21850,)
==================================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

```
In [86]: words_essay = vectorizer.get_feature_names()
         print(len(words_essay))
```

```
5000
```

```
In [87]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)


         vectorizer = CountVectorizer(min_df=10, 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")
```

```
(87398, 21) (87398,)
(21850, 21) (21850,)
====================================================================================================
After vectorizations
(87398, 2827) (87398,)
(21850, 2827) (21850,)
====================================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

```
In [88]: words_title = vectorizer.get_feature_names()
         print(len(words_title))
```

```
2827
```

5 Text - TfIdf

```
In [89]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)


         vectorizer = TfidfVectorizer(ngram_range=(2,2), min_df=10, max_features=5000)
         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")
```

```
(87398, 21) (87398,)
(21850, 21) (21850,)
==========================================================================================
After vectorizations
(87398, 5000) (87398,)
(21850, 5000) (21850,)
==========================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

```
In [90]: print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)


         vectorizer = TfidfVectorizer(min_df=10, 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_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")
```

```
(87398, 21) (87398,)
(21850, 21) (21850,)
==========================================================================================
After vectorizations
(87398, 2827) (87398,)
(21850, 2827) (21850,)
==========================================================================================
NOTE: THE NUMBER OF COLUMNS IN EACH OF THE VECTOR WONT BE SAME
```

6 Text - 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_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_test_title_avgw2v = np.array(test_avg_w2v_vectors1)
```

```
In [92]: #For Essays - Avgw2v
         print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)

         print("="*100)

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

```
(87398, 21) (87398,)
(21850, 21) (21850,)
==========================================================================================
After vectorizations
(87398, 300) (87398,)
(21850, 300) (21850,)
==========================================================================================
```

```
In [93]: #For Titles - Avgw2v
         print(X_train.shape, y_train.shape)
         print(X_test.shape, y_test.shape)
```

```
        print("="*100)

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

```
(87398, 21) (87398,)
(21850, 21) (21850,)
====================================================================================================
After vectorizations
(87398, 300) (87398,)
(21850, 300) (21850,)
====================================================================================================
```

7 Text - 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_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_test_title_tfidf_w2v = np.array(test_tfidf_w2v_vectors1)
```

```
In [95]: #For Essays - TfIdf weighted W2vec
         print(X_train.shape, y_train.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_test_es_tfidf_w2v.shape, y_test.shape)
         print("="*100)
```

```
(87398, 21) (87398,)
(21850, 21) (21850,)
====================================================================================================
After vectorizations
(87398, 300) (87398,)
(21850, 300) (21850,)
====================================================================================================
```

```
In [96]: #For Titles - TfIdf Weighted W2Vec
         print(X_train.shape, y_train.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_test_title_tfidf_w2v.shape, y_test.shape)
         print("="*100)
```

```
(87398, 21) (87398,)
(21850, 21) (21850,)
====================================================================================================
After vectorizations
(87398, 300) (87398,)
(21850, 300) (21850,)
====================================================================================================
```

8 Concatinating all the features

9 Bow

```
In [97]: 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
                               X_train_cl_subcategories_ohe, X_train_school_state_ohe, X_train_

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

```
(87398, 7928) (87398,)
```

```
In [98]: X_Bow_test = hstack((X_test_essay_bow, X_test_title_bow, te_price_standardized, te_qua
                             X_test_cl_subcategories_ohe, X_test_school_state_ohe, X_test_tea

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

```
(21850, 7928) (21850,)
```

10 TfIdf

```
In [99]: 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
                     X_train_cl_subcategories_ohe, X_train_school_state_ohe, X_train_t
```

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

```
(87398, 7928) (87398,)
```

```
In [100]: X_tf_test = hstack((X_test_essay_tf, X_test_title_tf, te_price_standardized, te_quant
                            X_test_cl_subcategories_ohe, X_test_school_state_ohe, X_test_tea
```

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

```
(21850, 7928) (21850,)
```

11 Avg W2Vec

```
In [101]: from scipy.sparse import hstack
          # with the same hstack function we are concatinating a sparse matrix and a dense mat
          X_avg_w2v_train = hstack((X_train_essay_avgw2v, X_train_title_avgw2v, tr_price_standa
                            X_train_cl_subcategories_ohe, X_train_school_state_ohe, X_train_
```

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

```
(87398, 701) (87398,)
```

```
In [102]: X_avg_w2v_test = hstack((X_test_essay_avgw2v, X_test_title_avgw2v, te_price_standardi
                            X_test_cl_subcategories_ohe, X_test_school_state_ohe, X_test_tea
```

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

```
(21850, 701) (21850,)
```

12 TfIdf weighted W2V

```
In [103]: from scipy.sparse import hstack
          # with the same hstack function we are concatinating a sparse matrix and a dense mat
          X_tf_w2v_train = hstack((X_train_es_tfidf_w2v, X_train_title_tfidf_w2v, tr_price_star
                            X_train_cl_subcategories_ohe, X_train_school_state_ohe, X_train_
```

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

```
(87398, 701) (87398,)
```

```
In [104]: X_avg_w2v_test = hstack((X_test_es_tfidf_w2v, X_test_title_tfidf_w2v, te_price_standa
                                   X_test_cl_subcategories_ohe, X_test_school_state_ohe, X_test_tea

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

```
(21850, 701) (21850,)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

13 Set1 - BOW

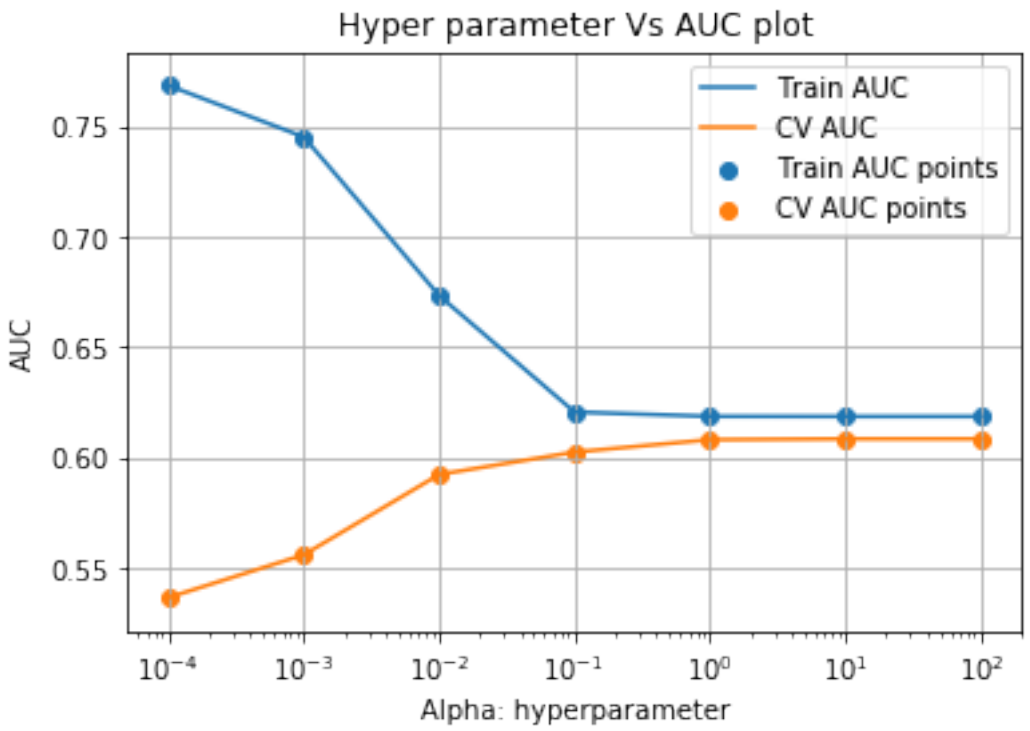
```
In [105]: from sklearn.linear_model import SGDClassifier
          from sklearn.model_selection import GridSearchCV

          parameters1={'alpha': [10**x for x in range(-4,3)] ,
                       'penalty' : ['l2']}

          clf_NB = SGDClassifier(loss = 'hinge',random_state=11,class_weight='balanced')

          clf1=GridSearchCV(clf_NB ,param_grid = parameters1, scoring="roc_auc", cv=5, return_t
          clf1.fit(X_Bow_train,y_train)
```

```
Out[105]: GridSearchCV(cv=5, error_score=nan,
                       estimator=SGDClassifier(alpha=0.0001, average=False,
                                               class_weight='balanced',
                                               early_stopping=False, epsilon=0.1,
                                               eta0=0.0, fit_intercept=True,
                                               l1_ratio=0.15, learning_rate='optimal',
                                               loss='hinge', max_iter=1000,
                                               n_iter_no_change=5, n_jobs=None,
                                               penalty='l2', power_t=0.5, random_state=11,
                                               shuffle=True, tol=0.001,
                                               validation_fraction=0.1, verbose=0,
                                               warm_start=False),
                       iid='deprecated', n_jobs=None,
                       param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],
                                   'penalty': ['l2']},
                       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                       scoring='roc_auc', verbose=0)
```

```
In [106]: a=clf1.best_params_['alpha']
          p= clf1.best_params_['penalty']
          print(clf1.best_score_)
          print(a)
          print(p)
```

```
0.6083523495337333
100
l2
```

In [107]:
```
train_auc= clf1.cv_results_['mean_train_score'][clf1.cv_results_['param_penalty']==p
train_auc_std= clf1.cv_results_['std_train_score'][clf1.cv_results_['param_penalty']=
cv_auc = clf1.cv_results_['mean_test_score'][clf1.cv_results_['param_penalty']==p]
cv_auc_std= clf1.cv_results_['std_test_score'][clf1.cv_results_['param_penalty']==p]

plt.plot(parameters1['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alph

plt.plot(parameters1['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color=

plt.scatter(parameters1['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters1['alpha'], cv_auc, label='CV AUC points')

plt.xscale('log')# we take the log in the x axis
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

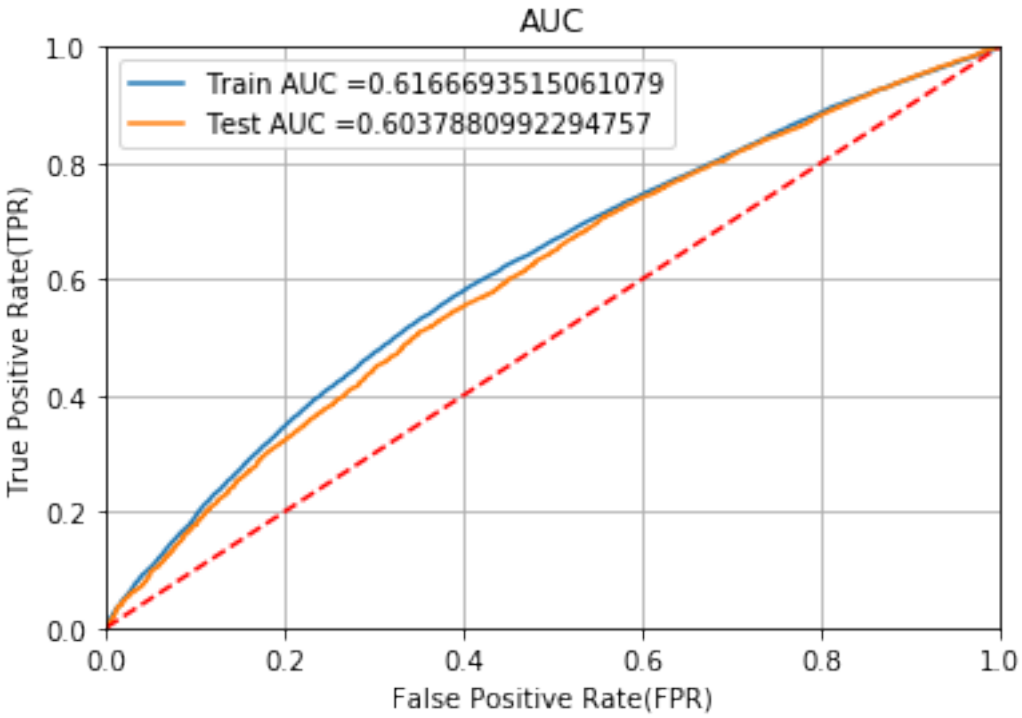


In [108]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#s
from sklearn.metrics import roc_curve, auc

SVM = SGDClassifier(loss = 'hinge',alpha=a, penalty=p, class_weight='balanced') # n_
SVM.fit(X_Bow_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
# not the predicted outputs
```

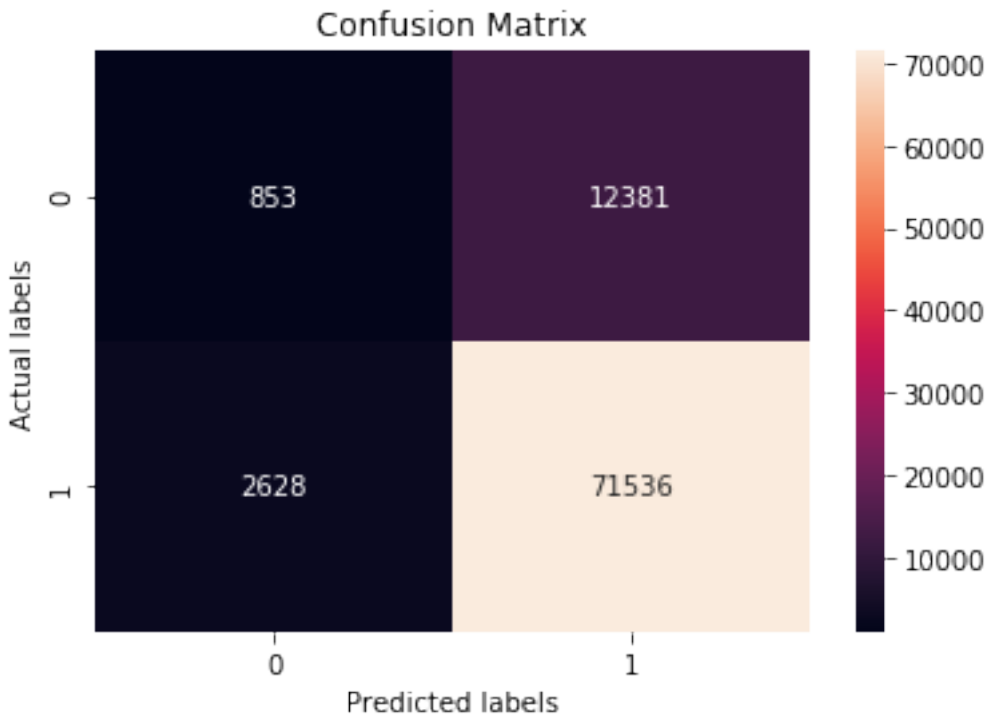
Out[108]:
```
SGDClassifier(alpha=100, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

In [109]:
```
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier
y_train_pred = SVM.decision_function(X_Bow_train)
y_test_pred = SVM.decision_function(X_Bow_test)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
```

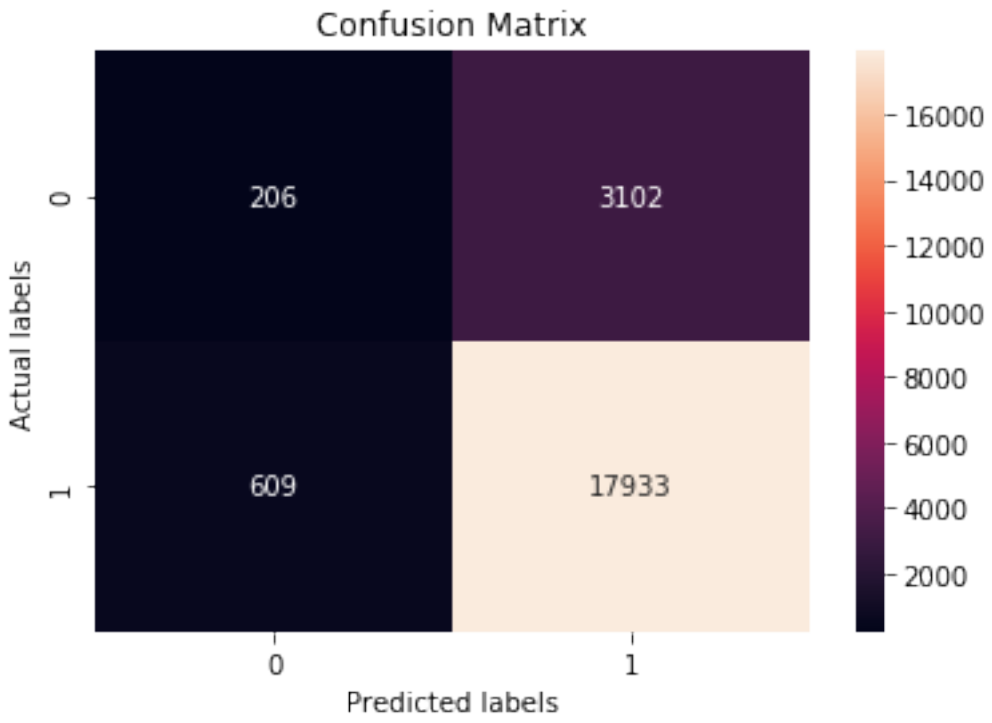
```
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))

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

plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [110]:
```
#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
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, SVM.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');
```



In [111]:
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
#Test dataset - confusion Matrix
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, SVM.predict(X_Bow_test)), annot=True, ax = ax,fr

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



14 Set2 - TfIdf

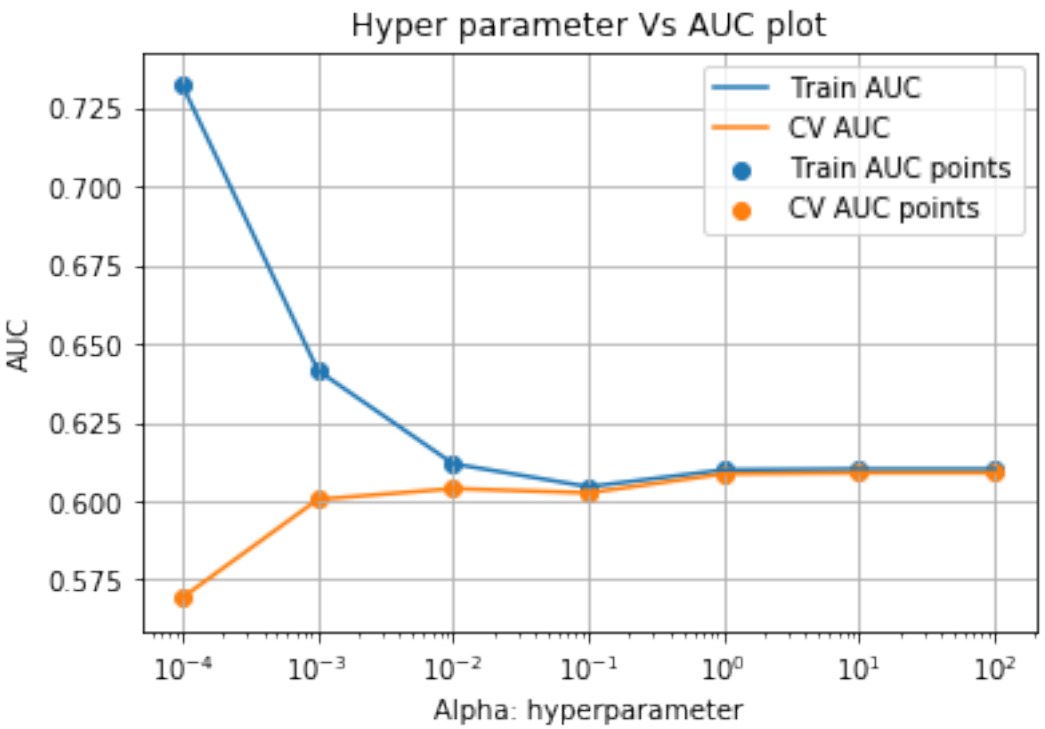
```
In [112]: from sklearn.linear_model import SGDClassifier
          from sklearn.model_selection import GridSearchCV

          parameters2={'alpha': [10**x for x in range(-4,3)] ,
                       'penalty' : ['l2']}

          clf_SVM1 = SGDClassifier(loss = 'hinge',random_state=11, class_weight='balanced')

          clf2=GridSearchCV(clf_SVM1 ,param_grid = parameters1, scoring="roc_auc", return_trai
          clf2.fit(X_tf_train,y_train)

Out[112]: GridSearchCV(cv=None, error_score=nan,
                       estimator=SGDClassifier(alpha=0.0001, average=False,
                                               class_weight='balanced',
                                               early_stopping=False, epsilon=0.1,
                                               eta0=0.0, fit_intercept=True,
                                               l1_ratio=0.15, learning_rate='optimal',
                                               loss='hinge', max_iter=1000,
                                               n_iter_no_change=5, n_jobs=None,
                                               penalty='l2', power_t=0.5, random_state=11,
                                               shuffle=True, tol=0.001,
```

```
                                        validation_fraction=0.1, verbose=0,
                                        warm_start=False),
                   iid='deprecated', n_jobs=None,
                   param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],
                               'penalty': ['l2']},
                   pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                   scoring='roc_auc', verbose=0)
```

In [113]:
```
a1=clf2.best_params_['alpha']
p1= clf2.best_params_['penalty']
print(clf2.best_score_)
print(a1)
print(p1)
```

```
0.6089595309958281
10
l2
```

In [114]:
```
train_auc= clf2.cv_results_['mean_train_score'][clf2.cv_results_['param_penalty']==p1
train_auc_std= clf2.cv_results_['std_train_score'][clf2.cv_results_['param_penalty']=
cv_auc = clf2.cv_results_['mean_test_score'][clf2.cv_results_['param_penalty']==p1]
cv_auc_std= clf2.cv_results_['std_test_score'][clf2.cv_results_['param_penalty']==p1]

plt.plot(parameters2['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha

plt.plot(parameters2['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color=

plt.scatter(parameters2['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters2['alpha'], cv_auc, label='CV AUC points')

plt.xscale('log')# we take the log in the x axis
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

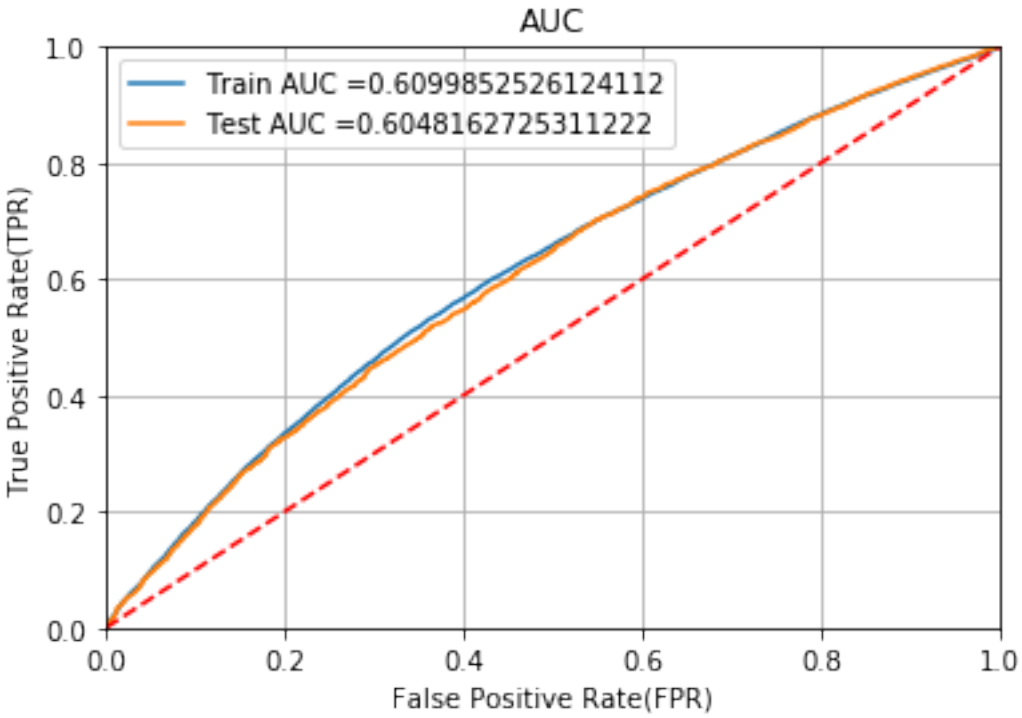


In [115]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#s
from sklearn.metrics import roc_curve, auc

SVM1 = SGDClassifier(loss = 'hinge',alpha=a, penalty=p, class_weight='balanced') # n
SVM1.fit(X_tf_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
# not the predicted outputs
```

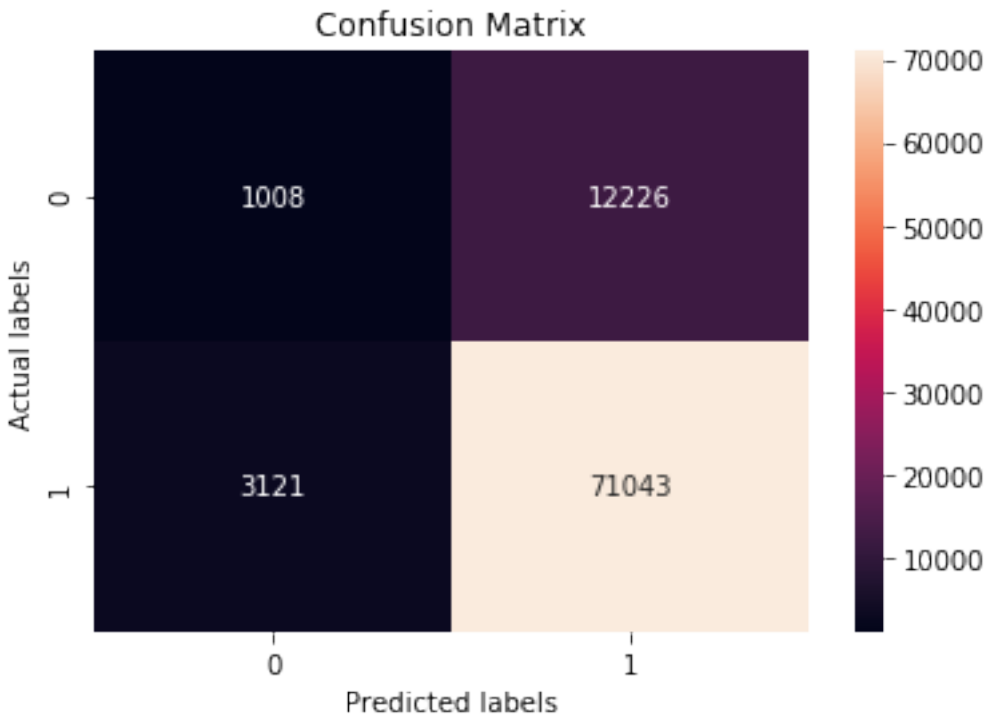
Out[115]:
```
SGDClassifier(alpha=100, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

In [116]:
```
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier
y_train_pred = SVM1.decision_function(X_tf_train)
y_test_pred = SVM1.decision_function(X_tf_test)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
```

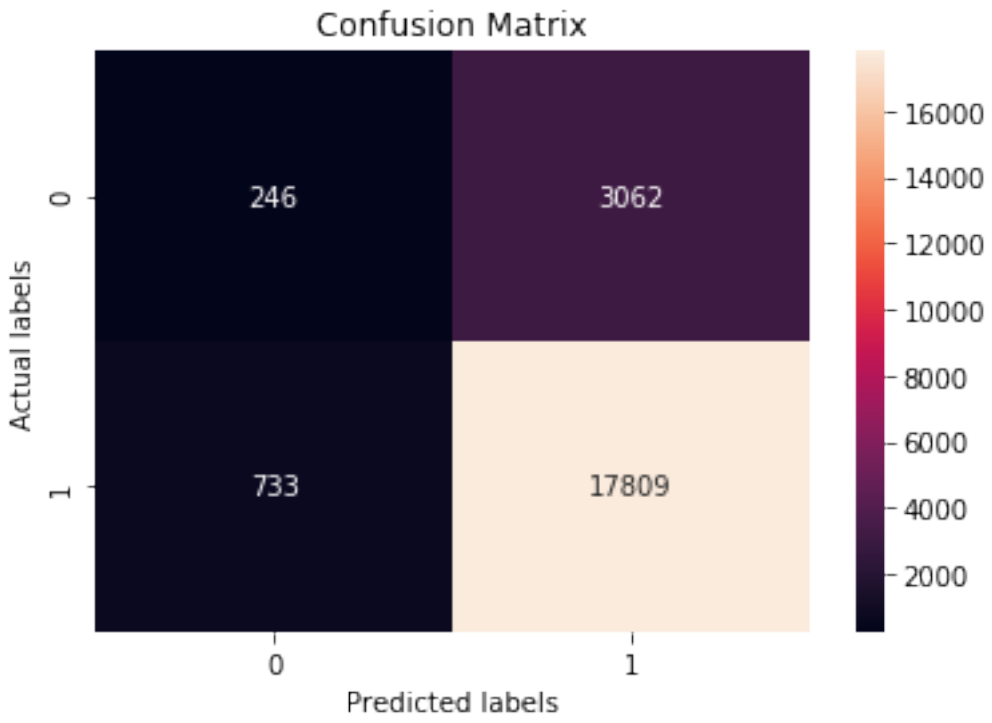
```
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))

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

plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [117]:
```
#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
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, SVM1.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');
```



In [118]:
```
#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
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, SVM1.predict(X_tf_test)), annot=True, ax = ax,fr

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



15 Set3 - Avg W2v

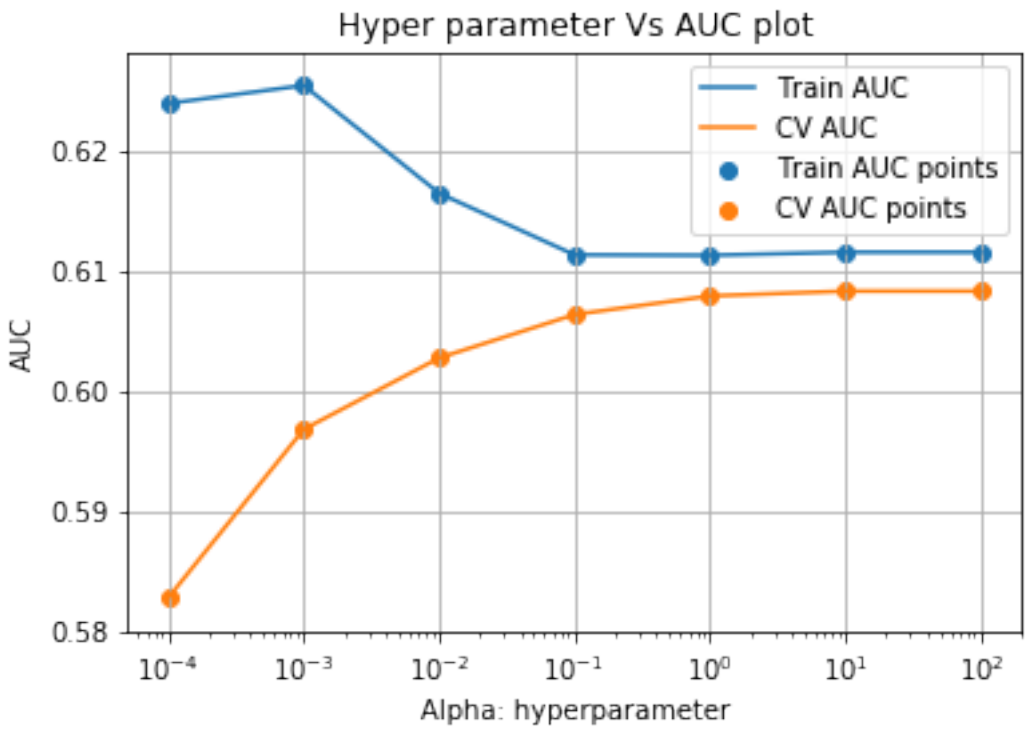
```
In [119]: from sklearn.linear_model import SGDClassifier
          from sklearn.model_selection import GridSearchCV

          parameters3={'alpha': [10**x for x in range(-4,3)] ,
                       'penalty' : ['l2']}

          clf_SVM2 = SGDClassifier(loss = 'hinge', random_state=11, penalty='l2', class_weight=

          clf3=GridSearchCV(clf_SVM2 ,param_grid = parameters1, scoring="roc_auc", return_train
          clf3.fit(X_avg_w2v_train,y_train)

Out[119]: GridSearchCV(cv=None, error_score=nan,
                       estimator=SGDClassifier(alpha=0.0001, average=False,
                                               class_weight='balanced',
                                               early_stopping=False, epsilon=0.1,
                                               eta0=0.0, fit_intercept=True,
                                               l1_ratio=0.15, learning_rate='optimal',
                                               loss='hinge', max_iter=1000,
                                               n_iter_no_change=5, n_jobs=None,
                                               penalty='l2', power_t=0.5, random_state=11,
                                               shuffle=True, tol=0.001,
```

```
                                        validation_fraction=0.1, verbose=0,
                                        warm_start=False),
                   iid='deprecated', n_jobs=None,
                   param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],
                               'penalty': ['l2']},
                   pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                   scoring='roc_auc', verbose=0)
```

In [120]:
```
a2=clf3.best_params_['alpha']
p2= clf3.best_params_['penalty']
print(clf3.best_score_)
print(a2)
print(p2)
```

```
0.6082794302920311
100
l2
```

In [121]:
```
train_auc= clf3.cv_results_['mean_train_score'][clf3.cv_results_['param_penalty']==p2
train_auc_std= clf3.cv_results_['std_train_score'][clf3.cv_results_['param_penalty']=
cv_auc = clf3.cv_results_['mean_test_score'][clf3.cv_results_['param_penalty']==p2]
cv_auc_std= clf3.cv_results_['std_test_score'][clf3.cv_results_['param_penalty']==p2]

plt.plot(parameters3['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha

plt.plot(parameters3['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color=

plt.scatter(parameters3['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters3['alpha'], cv_auc, label='CV AUC points')

plt.xscale('log')# we take the log in the x axis
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

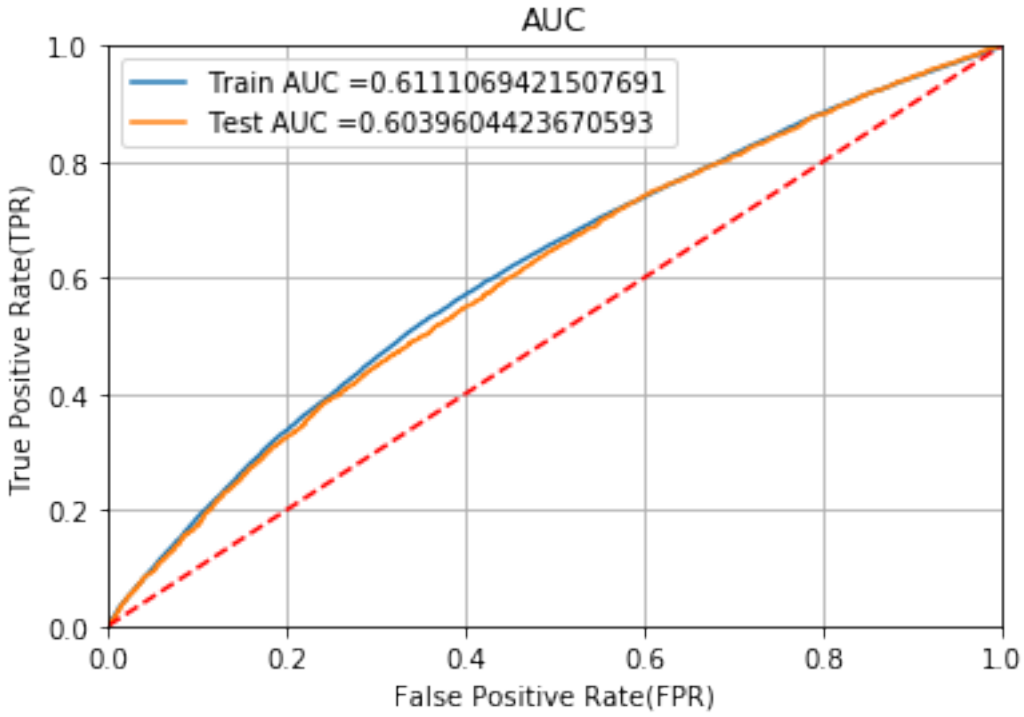


In [122]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#s
from sklearn.metrics import roc_curve, auc

SVM2 = SGDClassifier(loss = 'hinge',alpha=a2, penalty='l2', class_weight='balanced')
SVM2.fit(X_avg_w2v_train, y_train)
```

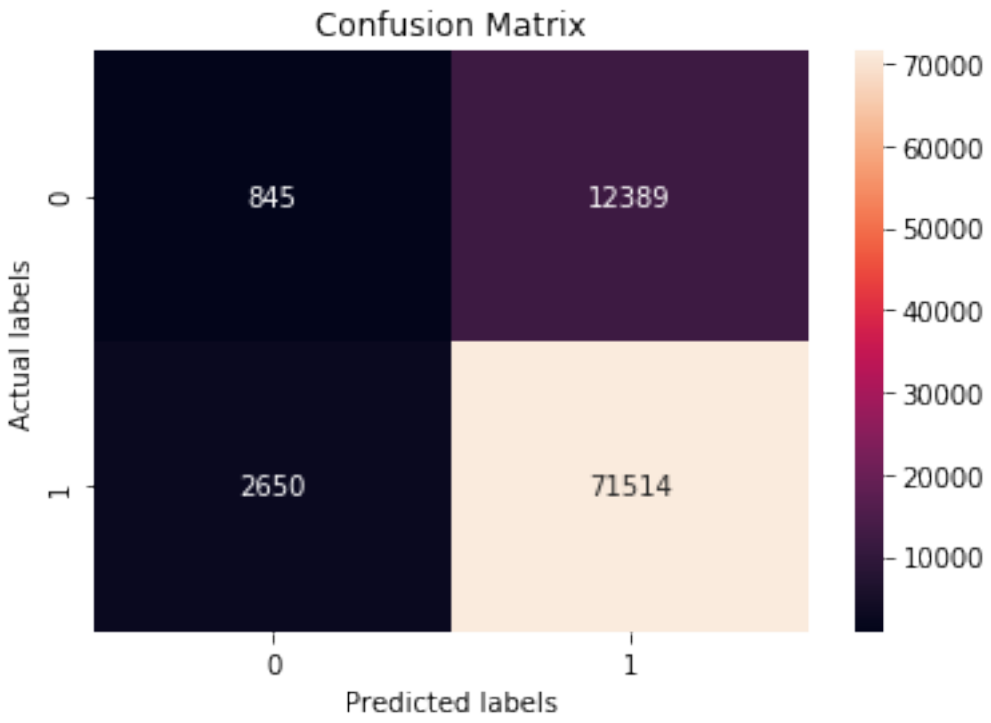
Out[122]:
```
SGDClassifier(alpha=100, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

In [123]:
```
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier
y_train_pred = SVM2.decision_function(X_avg_w2v_train)
y_test_pred = SVM2.decision_function(X_avg_w2v_test)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
```

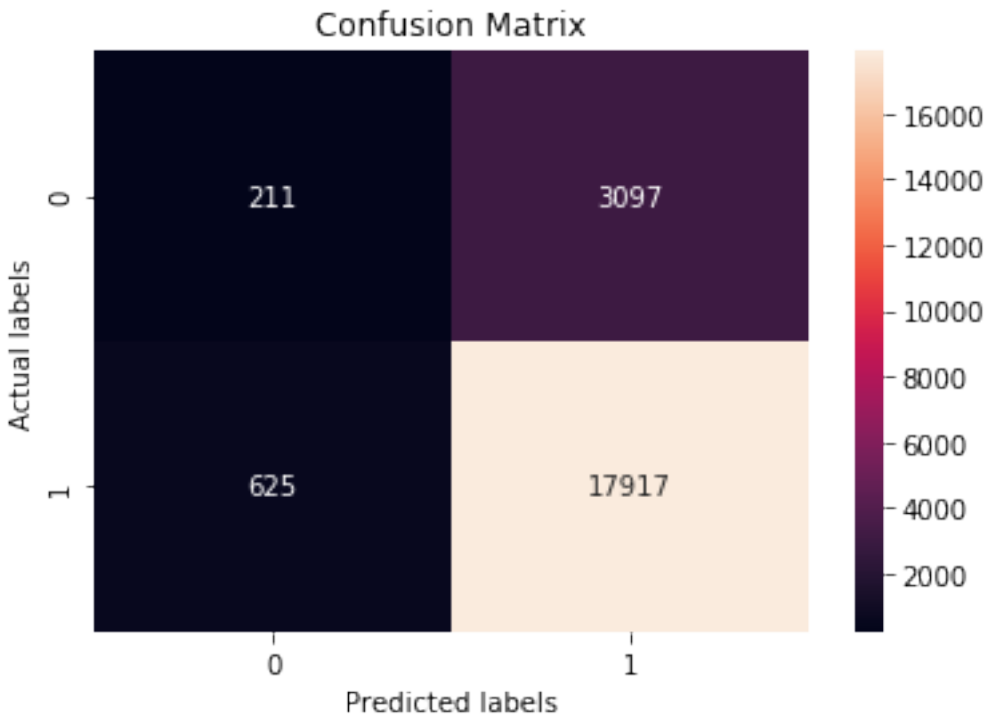
```
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])

plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [124]:
```
#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
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, SVM2.predict(X_avg_w2v_train)), annot=True, ax

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



In [125]:
```
#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
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, SVM2.predict(X_avg_w2v_test)), annot=True, ax =

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



16 Set4 - tfidf W2v

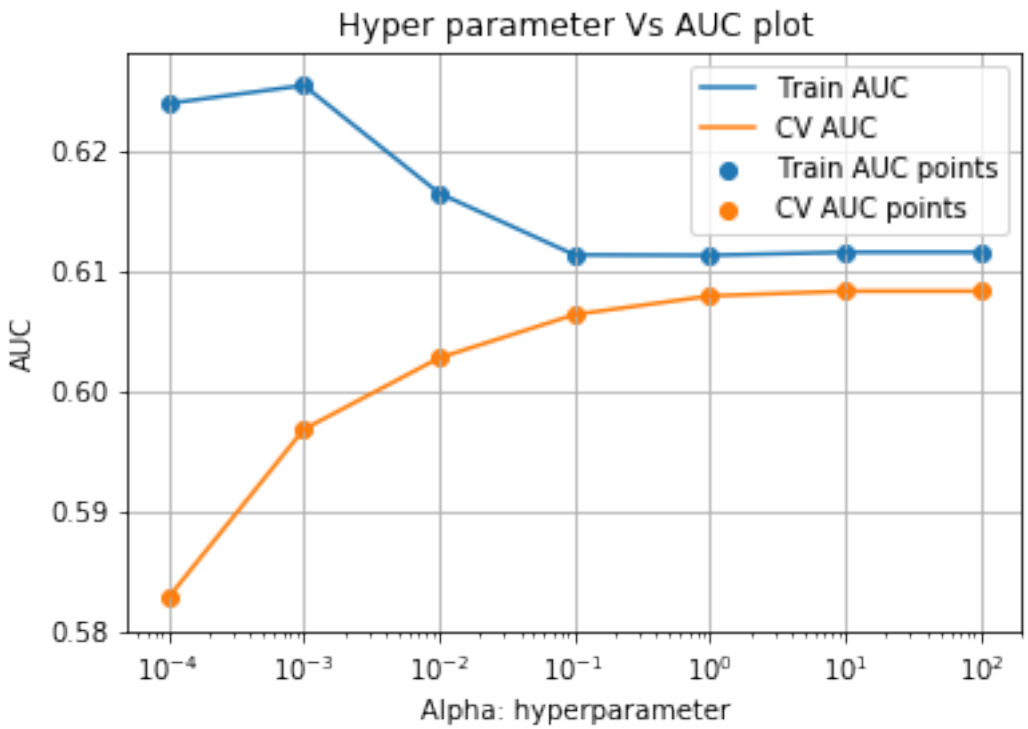
```
In [126]: from sklearn.linear_model import SGDClassifier
          from sklearn.model_selection import GridSearchCV

          parameters4={'alpha': [10**x for x in range(-4,3)] ,
                       'penalty' : ['l2']}

          clf_SVM4 = SGDClassifier(loss = 'hinge', random_state=11, class_weight='balanced')

          clf4=GridSearchCV(clf_SVM4 ,param_grid = parameters4, scoring="roc_auc", return_trai
          clf4.fit(X_avg_w2v_train,y_train)

Out[126]: GridSearchCV(cv=None, error_score=nan,
                       estimator=SGDClassifier(alpha=0.0001, average=False,
                                               class_weight='balanced',
                                               early_stopping=False, epsilon=0.1,
                                               eta0=0.0, fit_intercept=True,
                                               l1_ratio=0.15, learning_rate='optimal',
                                               loss='hinge', max_iter=1000,
                                               n_iter_no_change=5, n_jobs=None,
                                               penalty='l2', power_t=0.5, random_state=11,
                                               shuffle=True, tol=0.001,
```

```
                                        validation_fraction=0.1, verbose=0,
                                        warm_start=False),
                   iid='deprecated', n_jobs=None,
                   param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],
                               'penalty': ['l2']},
                   pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                   scoring='roc_auc', verbose=0)
```

In [127]:
```
a3=clf4.best_params_['alpha']
p3= clf4.best_params_['penalty']
print(clf4.best_score_)
print(a3)
print(p3)
```

```
0.6082794302920311
100
l2
```

In [128]:
```
train_auc= clf4.cv_results_['mean_train_score'][clf4.cv_results_['param_penalty']==p3
train_auc_std= clf4.cv_results_['std_train_score'][clf4.cv_results_['param_penalty']=
cv_auc = clf4.cv_results_['mean_test_score'][clf4.cv_results_['param_penalty']==p3]
cv_auc_std= clf4.cv_results_['std_test_score'][clf4.cv_results_['param_penalty']==p3]

plt.plot(parameters4['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alph

plt.plot(parameters4['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color=

plt.scatter(parameters4['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters4['alpha'], cv_auc, label='CV AUC points')

plt.xscale('log')# we take the log in the x axis
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

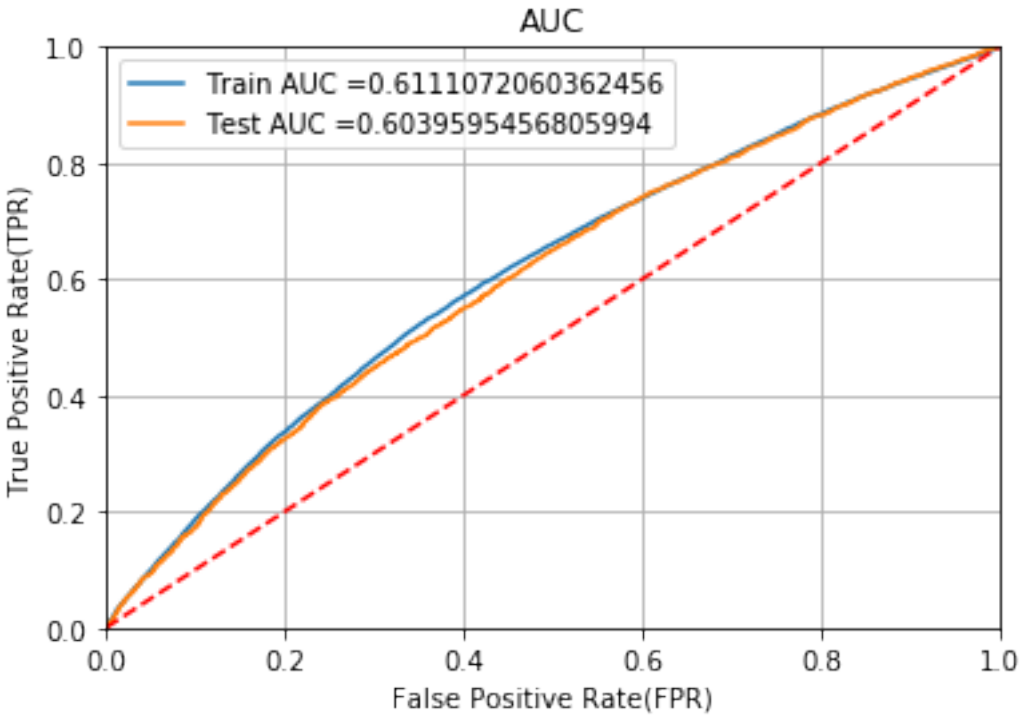


In [129]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#s
from sklearn.metrics import roc_curve, auc

SVM4 = SGDClassifier(loss = 'hinge', alpha=a3, penalty='l2', class_weight='balanced'
SVM4.fit(X_avg_w2v_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
# not the predicted outputs
```

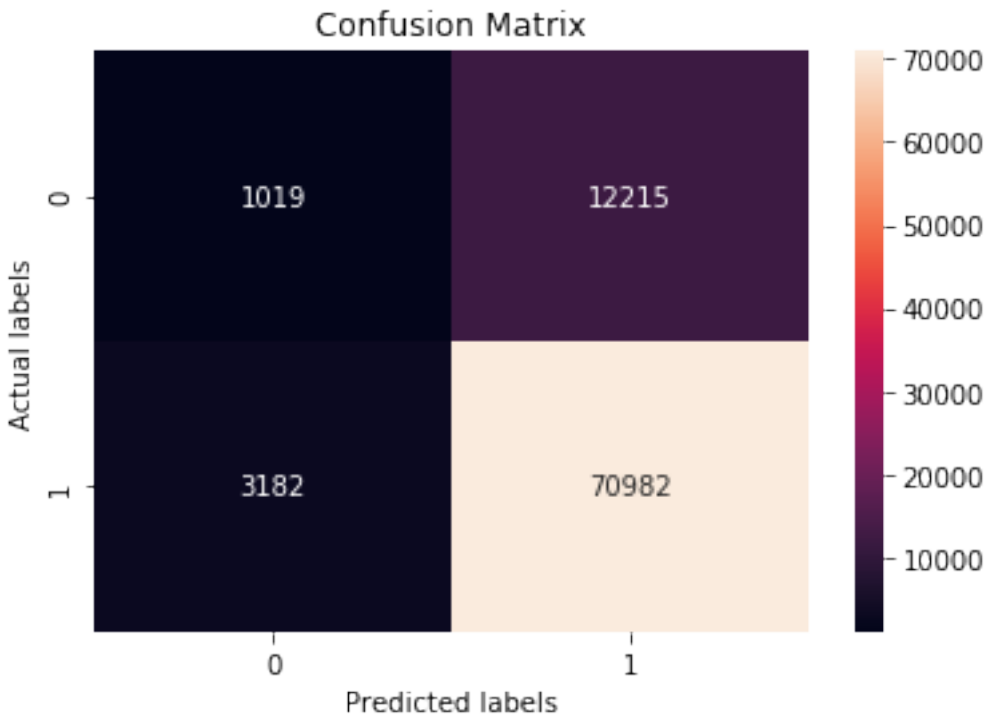
Out[129]:
```
SGDClassifier(alpha=100, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

In [130]:
```
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier
y_train_pred = SVM4.decision_function(X_avg_w2v_train)
y_test_pred = SVM4.decision_function(X_avg_w2v_test)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
```

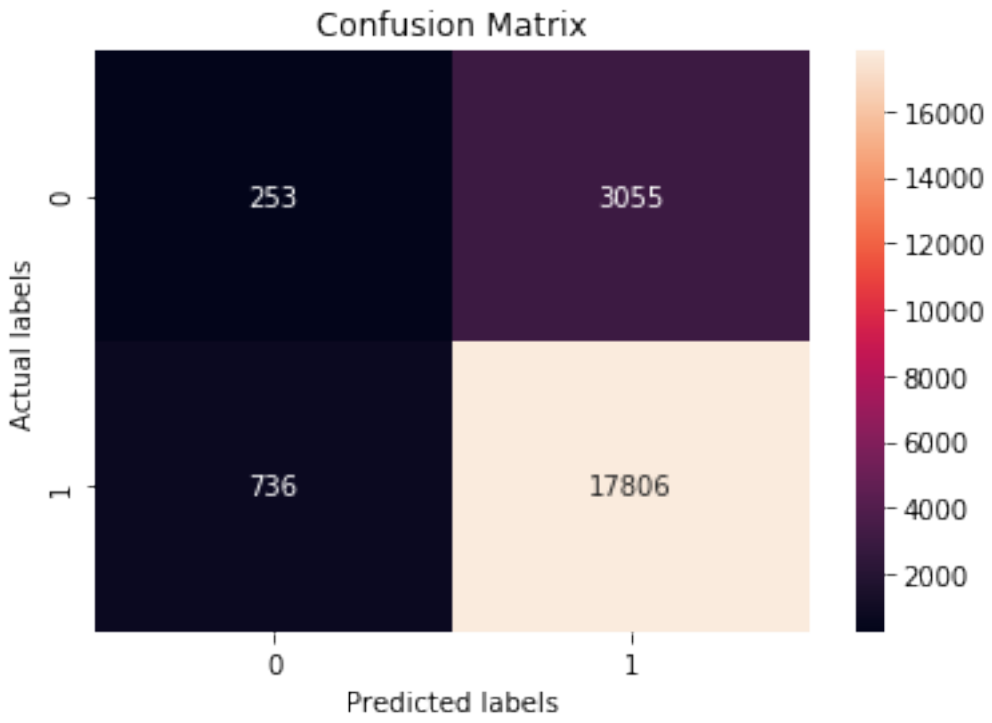
```
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))

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

plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [131]:
```
#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
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, SVM4.predict(X_avg_w2v_train)), annot=True, ax

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



In [132]:
```
#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
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, SVM4.predict(X_avg_w2v_test)), annot=True, ax =

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



2.4 Appling Support Vector Machines on different kind of featurization as mentioned in the instructions

Apply Support Vector Machines 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

17 Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (n_components) using elbow method : numerical data

```
In [133]: #https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_of_c
          #https://stackoverflow.com/questions/48424084/number-of-components-trucated-svd
          #https://www.youtube.com/watch?v=xvfJXNGCQBM

          from sklearn.decomposition import TruncatedSVD
          svd = TruncatedSVD(n_components=4000, algorithm='randomized', n_iter=5, random_state=
          svd.fit(X_train_essay_tf)

Out[133]: TruncatedSVD(algorithm='randomized', n_components=4000, n_iter=5,
                       random_state=None, tol=0.0001)

In [134]: # List of explained variances
          tsvd_var_ratios = svd.explained_variance_ratio_
          np.cumsum(tsvd_var_ratios)
```

```
Out[134]: array([0.00112121, 0.00607297, 0.00939448, ..., 0.92654402, 0.92662604,
                 0.92670799])
```

```
In [0]: # Create a function for find best no. of components
        def select_n_components(var_ratio, goal_var: float):
            # Set initial variance explained so far
            total_variance = 0.0

            # Set initial number of features
            n_components = 0

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


                total_variance += explained_variance


                n_components += 1


                if total_variance >= goal_var:

                    break


            return n_components
```

```
In [136]: # we are taking variance as 95%, one can either chose 90% or 95%
          #https://www.youtube.com/watch?v=xvfJXNGCQBM

          n_components=select_n_components(tsvd_var_ratios, 0.95)
          n_components
```

```
Out[136]: 4000
```

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

        svd_trun = TruncatedSVD(n_components=4000, algorithm='randomized', n_iter=5, random_sta
        svd_train = svd_trun.fit_transform(X_train_essay_tf)
        svd_test = svd_trun.transform(X_test_essay_tf)
```

```
In [0]: #Now, we are adding the TSVD to task-2
```

2.5 Support Vector Machines with added Features `Set 5`

18 Concatinating the features - Task 2

```
In [0]: #Number of words in Title
        num_words_scaler = StandardScaler()
```

```
        num_words_scaler.fit(X_train['title_len'].values.reshape(-1,1)) # finding the mean and

        # Now standardize the data with above maen and variance.
        tr_title_standardized = num_words_scaler.transform(X_train['title_len'].values.reshape
        te_title_standardized = num_words_scaler.transform(X_test['title_len'].values.reshape(

In [0]: #Number of words in Essay
        num_essay_scaler = StandardScaler()
        num_essay_scaler.fit(X_train['essay_len'].values.reshape(-1,1)) # finding the mean and

        # Now standardize the data with above maen and variance.
        tr_essay_standardized = num_essay_scaler.transform(X_train['essay_len'].values.reshape
        te_essay_standardized = num_essay_scaler.transform(X_test['essay_len'].values.reshape(

In [0]: #Computing Sentiment score - Essays - Neg
        essay_neg = StandardScaler()
        essay_neg.fit(X_train['neg'].values.reshape(-1,1)) # finding the mean and standard dev

        # Now standardize the data with above maen and variance.
        tr_essay_neg = essay_neg.transform(X_train['neg'].values.reshape(-1, 1))
        te_essay_neg = essay_neg.transform(X_test['neg'].values.reshape(-1, 1))

In [0]: #Computing Sentiment score - Essays - Neut
        essay_neut = StandardScaler()
        essay_neut.fit(X_train['neut'].values.reshape(-1,1)) # finding the mean and standard d

        # Now standardize the data with above maen and variance.
        tr_essay_neut = essay_neut.transform(X_train['neut'].values.reshape(-1, 1))
        te_essay_neut = essay_neut.transform(X_test['neut'].values.reshape(-1, 1))

In [0]: #Computing Sentiment score - Essays - pos
        essay_pos = StandardScaler()
        essay_pos.fit(X_train['pos'].values.reshape(-1,1)) # finding the mean and standard dev

        # Now standardize the data with above maen and variance.
        tr_essay_pos = essay_pos.transform(X_train['pos'].values.reshape(-1, 1))
        te_essay_pos = essay_pos.transform(X_test['pos'].values.reshape(-1, 1))

In [0]: #Computing Sentiment score - Essays - comp
        essay_comp = StandardScaler()
        essay_comp.fit(X_train['comp'].values.reshape(-1,1)) # finding the mean and standard d

        # Now standardize the data with above maen and variance.
        tr_essay_comp = essay_comp.transform(X_train['comp'].values.reshape(-1, 1))
        te_essay_comp = essay_comp.transform(X_test['comp'].values.reshape(-1, 1))

In [147]: X_train_data = hstack((svd_train, tr_essay_comp, tr_essay_neut, tr_essay_pos, tr_essa
                                 tr_price_standardized, tr_quantity_standardized, tr_number_pro
                                 X_train_teacher_prefix_ohe, X_train_school_state_ohe, X_train_
```

```
                              X_train_cl_categories_ohe)).tocsr()
          X_train_data.shape
```

```
Out[147]: (87398, 4108)
```

```
In [148]: X_test_data = hstack((svd_test, te_essay_comp, te_essay_neut, te_essay_pos, te_essay
                              te_price_standardized, te_quantity_standardized, te_number_pro
                              X_test_teacher_prefix_ohe, X_test_school_state_ohe, X_test_cl_
                              X_test_cl_categories_ohe)).tocsr()
          X_test_data.shape
```

```
Out[148]: (21850, 4108)
```

```
In [149]: from sklearn.linear_model import SGDClassifier
          from sklearn.model_selection import GridSearchCV

          parameters5={'alpha': [10**x for x in range(-4,3)] ,
                       'penalty' : ['l2']}

          clf_SVM5 = SGDClassifier(loss = 'hinge', random_state=11, class_weight='balanced')

          clf5=GridSearchCV(clf_SVM5, param_grid = parameters5, scoring="roc_auc", return_trai
          clf5.fit(X_train_data, y_train)
```

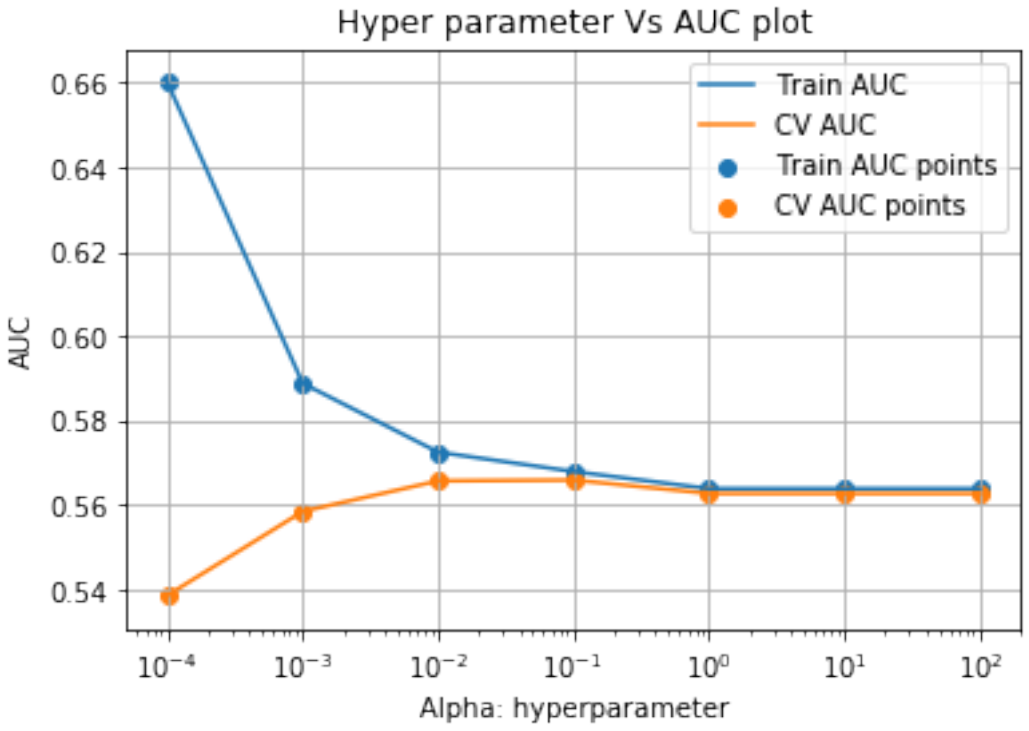
```
Out[149]: GridSearchCV(cv=None, error_score=nan,
                       estimator=SGDClassifier(alpha=0.0001, average=False,
                                               class_weight='balanced',
                                               early_stopping=False, epsilon=0.1,
                                               eta0=0.0, fit_intercept=True,
                                               l1_ratio=0.15, learning_rate='optimal',
                                               loss='hinge', max_iter=1000,
                                               n_iter_no_change=5, n_jobs=None,
                                               penalty='l2', power_t=0.5, random_state=11,
                                               shuffle=True, tol=0.001,
                                               validation_fraction=0.1, verbose=0,
                                               warm_start=False),
                       iid='deprecated', n_jobs=None,
                       param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],
                                   'penalty': ['l2']},
                       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                       scoring='roc_auc', verbose=0)
```

```
In [150]: a4=clf5.best_params_['alpha']
          p4= clf5.best_params_['penalty']
          print(clf5.best_score_)
          print(a4)
          print(p4)
```

```
0.5659003090238606
0.1
```

12

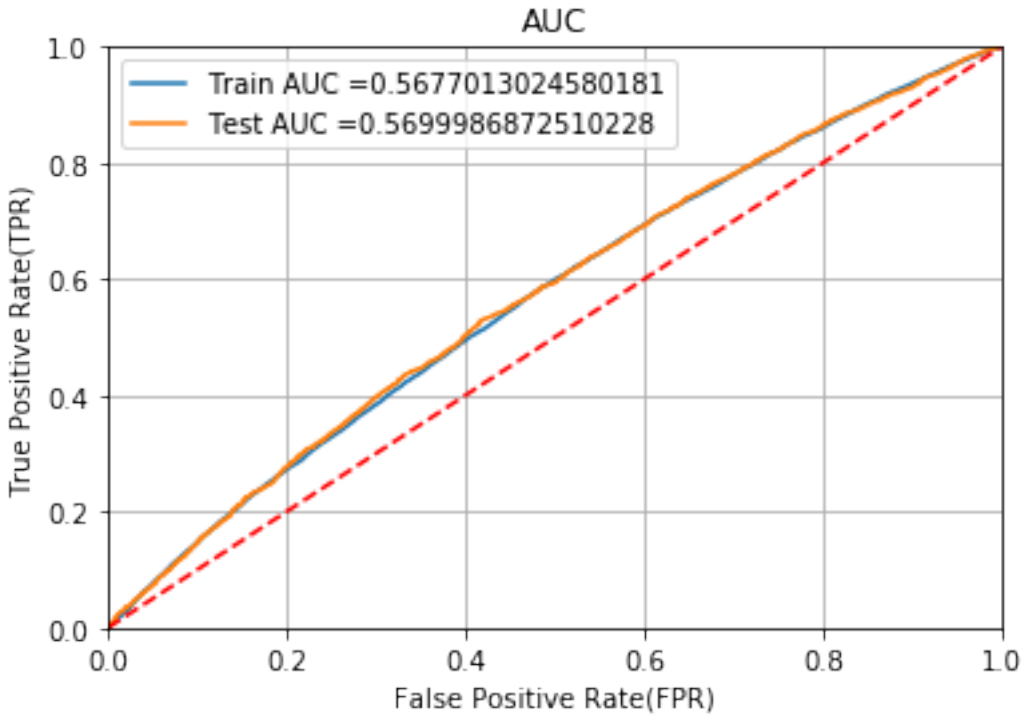
In [151]:
```
train_auc= clf5.cv_results_['mean_train_score'][clf5.cv_results_['param_penalty']==p4
train_auc_std= clf5.cv_results_['std_train_score'][clf5.cv_results_['param_penalty']=
cv_auc = clf5.cv_results_['mean_test_score'][clf5.cv_results_['param_penalty']==p4]
cv_auc_std= clf5.cv_results_['std_test_score'][clf5.cv_results_['param_penalty']==p4]

plt.plot(parameters5['alpha'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha

plt.plot(parameters5['alpha'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color=

plt.scatter(parameters5['alpha'], train_auc, label='Train AUC points')
plt.scatter(parameters5['alpha'], cv_auc, label='CV AUC points')

plt.xscale('log')# we take the log in the x axis
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



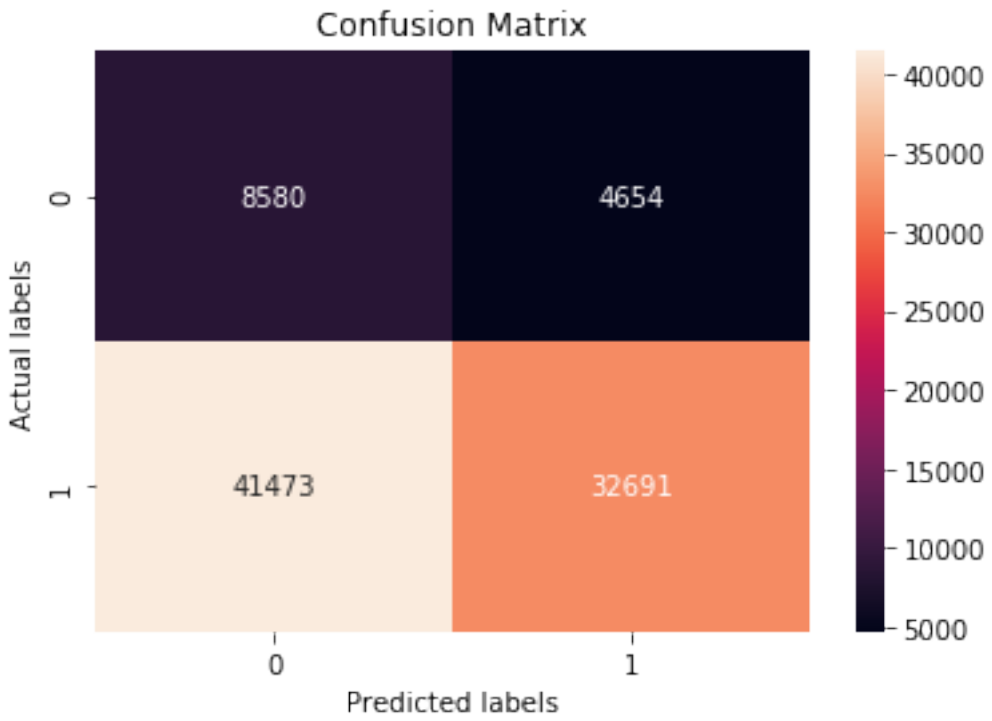
In [152]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#s
from sklearn.metrics import roc_curve, auc

SVM5 = SGDClassifier(loss = 'hinge', alpha=a4, penalty='l2', class_weight='balanced'
SVM5.fit(X_train_data, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
# not the predicted outputs
```

Out[152]:
```
SGDClassifier(alpha=0.1, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

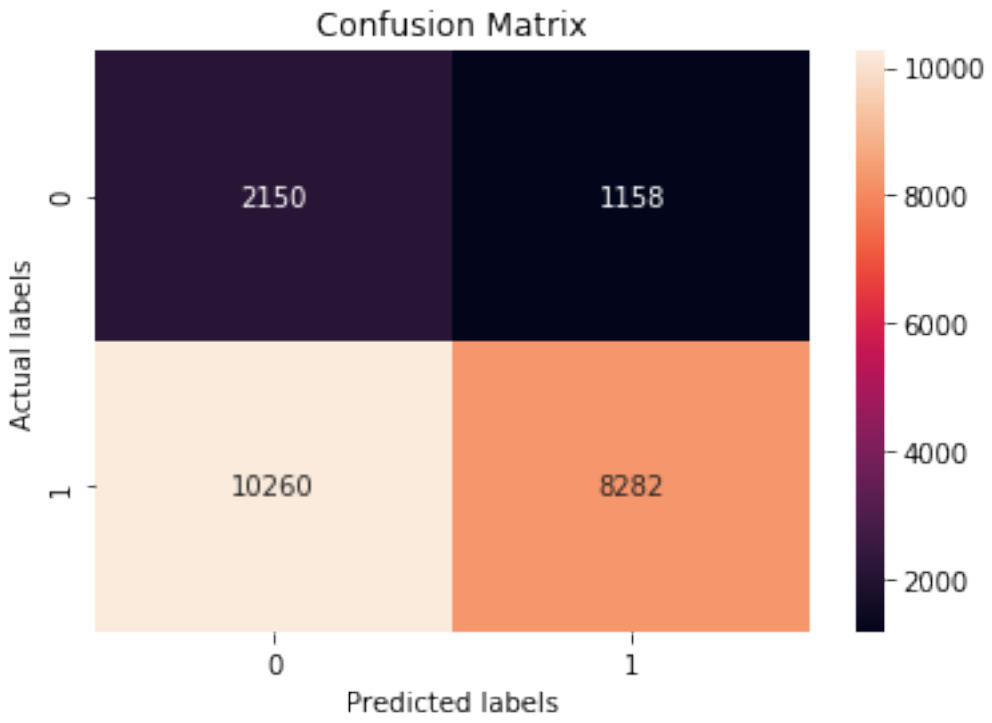
In [153]:
```
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier
y_train_pred = SVM5.decision_function(X_train_data)
y_test_pred = SVM5.decision_function(X_test_data)

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))

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

plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



In [154]:
```
#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
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, SVM5.predict(X_train_data)), annot=True, ax = a

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




```
In [155]: #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
          ax= plt.subplot()
          sns.heatmap(confusion_matrix(y_test, SVM5.predict(X_test_data)), annot=True, ax = ax

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



3. Conclusion

```
In [157]: # Please compare all your models using Prettytable library
          #how to use pretty table http://zetcode.com/python/prettytable/
          from prettytable import PrettyTable
          tb = PrettyTable()
          tb.field_names= ("Vectorizer", " Alpha ", " AUC ")
          tb.add_row(["BOW ", a, '0.603'])
          tb.add_row(["Tf - Idf ", a1, '0.604'])
          tb.add_row(["AVG - W2V", a2, '0.603'])
          tb.add_row(["AVG - Tf - Idf", a3, '0.603'])
          tb.add_row(["Truncated SVD", a4 , '0.569'])
          print(tb.get_string(titles = "SVM- Observations"))
```

```
+----------------+---------+-------+
|   Vectorizer   |  Alpha  |  AUC  |
+----------------+---------+-------+
|      BOW       |   100   | 0.603 |
|   Tf - Idf     |    10   | 0.604 |
|   AVG - W2V    |   100   | 0.603 |
| AVG - Tf - Idf |   100   | 0.603 |
| Truncated SVD  |   0.1   | 0.569 |
+----------------+---------+-------+
```