Importing Libraries

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os

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

from collections import Counter
from prettytable import PrettyTable

from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import math
```

Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv',nrows=50000)
resource_data =pd.read_csv('resources.csv', nrows =50000)
```

In [3]:

```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 17 columns):
 #   Column                                        Non-Null Count  Dtype 
---  ------                                        --------------  ----- 
 0   Unnamed: 0                                    50000 non-null  int64 
 1   id                                            50000 non-null  object
 2   teacher_id                                    50000 non-null  object
 3   teacher_prefix                                49998 non-null  object
 4   school_state                                  50000 non-null  object
 5   project_submitted_datetime                    50000 non-null  object
 6   project_grade_category                        50000 non-null  object
 7   project_subject_categories                    50000 non-null  object
 8   project_subject_subcategories                 50000 non-null  object
 9   project_title                                 50000 non-null  object
 10  project_essay_1                               50000 non-null  object
 11  project_essay_2                               50000 non-null  object
 12  project_essay_3                               1685 non-null   object
 13  project_essay_4                               1685 non-null   object
 14  project_resource_summary                      50000 non-null  object
 15  teacher_number_of_previously_posted_projects  50000 non-null  int64 
 16  project_is_approved                           50000 non-null  int64 
dtypes: int64(3), object(14)
memory usage: 6.5+ MB
```

In [4]:

```
resource_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   id           50000 non-null  object 
 1   description  49995 non-null  object 
 2   quantity     50000 non-null  int64  
 3   price        50000 non-null  float64
dtypes: float64(1), int64(1), object(2)
memory usage: 1.5+ MB
```

In [5]:

```
# replacing nan with most occuring element
project_data['teacher_prefix']= project_data['teacher_prefix'].fillna(project_data['tea
cher_prefix'].mode().iloc[0])
```

In [6]:

```
# 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)
```

preprocessing subject categories

In [7]:

```
categories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-stri
ng
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-pyth
on
cat_list = []
for i in categories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmt
h", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "M
ath & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace
 it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"M
ath & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spa
ces
        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]))
```

preprocessing subject subcategories

In [8]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

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

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
h", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "M
ath & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace
 it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"M
ath & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spa
ces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

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

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

In [9]:

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

Out[9]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_sta
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN

Spliting the data into test and train

In [10]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, strat
ify=y_train)
```

Text Preprocessing

For Essay

In [11]:

```
# 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 [12]:

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

```
My kindergarten students have varied disabilities ranging from speech and
language delays, cognitive delays, gross/fine motor delays, to autism. The
y are eager beavers and always strive to work their hardest working past t
heir limitations. \r\n\r\nThe materials we have are the ones I seek out fo
r my students. I teach in a Title I school where most of the students rece
ive free or reduced price lunch.  Despite their disabilities and limitatio
ns, my students love coming to school and come eager to learn and explore.
Have you ever felt like you had ants in your pants and you needed to groov
e and move as you were in a meeting? This is how my kids feel all the tim
e. The want to be able to move as they learn or so they say.Wobble chairs
are the answer and I love then because they develop their core, which enha
nces gross motor and in Turn fine motor skills. \r\nThey also want to lear
n through games, my kids do not want to sit and do worksheets. They want t
o learn to count by jumping and playing. Physical engagement is the key to
our success. The number toss and color and shape mats can make that happe
n. My students will forget they are doing work and just have the fun a 6 y
ear old deserves.nannan
==================================================
```

In [13]:

```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-py
thon/
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, cognitive delays, gross/fine motor delays, to autism. The
y are eager beavers and always strive to work their hardest working past t
heir limitations.     The materials we have are the ones I seek out for my
students. I teach in a Title I school where most of the students receive f
ree or reduced price lunch.  Despite their disabilities and limitations, m
y students love coming to school and come eager to learn and explore.Have
you ever felt like you had ants in your pants and you needed to groove and
move as you were in a meeting? This is how my kids feel all the time. The
want to be able to move as they learn or so they say.Wobble chairs are the
answer and I love then because they develop their core, which enhances gro
ss motor and in Turn fine motor skills.   They also want to learn through
games, my kids do not want to sit and do worksheets. They want to learn to
count by jumping and playing. Physical engagement is the key to our succes
s. The number toss and color and shape mats can make that happen. My stude
nts will forget they are doing work and just have the fun a 6 year old des
erves.nannan
```

In [14]:

```
#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 cognitive delays gross fine motor delays to autism They ar
e eager beavers and always strive to work their hardest working past their
limitations The materials we have are the ones I seek out for my students
I teach in a Title I school where most of the students receive free or red
uced price lunch Despite their disabilities and limitations my students lo
ve coming to school and come eager to learn and explore Have you ever felt
like you had ants in your pants and you needed to groove and move as you w
ere in a meeting This is how my kids feel all the time The want to be able
to move as they learn or so they say Wobble chairs are the answer and I lo
ve then because they develop their core which enhances gross motor and in
Turn fine motor skills They also want to learn through games my kids do no
t want to sit and do worksheets They want to learn to count by jumping and
playing Physical engagement is the key to our success The number toss and
color and shape mats can make that happen My students will forget they are
doing work and just have the fun a 6 year old deserves nannan
```

In [15]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [16]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_train_essay = []
# 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)
    preprocessed_train_essay.append(sent.lower().strip())
```

100%|██████████| 22445/22445 [00:23<00:00, 960.00it/s]

In [17]:

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

100%|██████████| 11055/11055 [00:11<00:00, 992.68it/s]

In [18]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_test_essay = []
# 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)
    preprocessed_test_essay.append(sent.lower().strip())
```

100%|██████████| 16500/16500 [00:16<00:00, 980.66it/s]

For Titles

In [19]:

```
# 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 [20]:

```
sent = decontracted(project_data['project_title'].values[37000])
print(sent)
print('='*50)
```

```
Focus our CORE!
==================================================
```

In [21]:

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

```
Focus our CORE!
```

In [22]:

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

```
Focus our CORE
```

In [23]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [24]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_train_title = []
# 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)
    preprocessed_train_title.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████
████| 22445/22445 [00:01<00:00, 21696.79it/s]
```

In [25]:

```
preprocessed_cv_title = []
# tqdm is for printing the status bar
for sentance in tqdm(X_cv['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_cv_title.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████
████| 11055/11055 [00:00<00:00, 20752.94it/s]
```

In [26]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_test_title = []
# 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)
    preprocessed_test_title.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████
████| 16500/16500 [00:00<00:00, 21722.89it/s]
```

Vectorizing categorical data

Categories

In [27]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vec1 = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary
=True)
X_train_cat_ohe = vec1.fit_transform(X_train['clean_categories'].values)
X_cv_cat_ohe = vec1.transform(X_cv['clean_categories'].values)
X_test_cat_ohe = vec1.transform(X_test['clean_categories'].values)
```

In [28]:

```
print("Shape of X_train after one hot encodig ",X_train_cat_ohe.shape, y_train.shape)
print("Shape of X_cv after one hot encodig ",X_cv_cat_ohe.shape, y_cv.shape)
print("Shape of X_test after one hot encodig ",X_test_cat_ohe.shape, y_test.shape)
```

```
Shape of X_train after one hot encodig  (22445, 9) (22445,)
Shape of X_cv after one hot encodig  (11055, 9) (11055,)
Shape of X_test after one hot encodig  (16500, 9) (16500,)
```

Subcategories

In [29]:

```
vec2 = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, bi
nary=True)
X_train_sub_cat_ohe = vec2.fit_transform(X_train['clean_subcategories'].values)
X_cv_sub_cat_ohe = vec2.transform(X_cv['clean_subcategories'].values)
X_test_sub_cat_ohe = vec2.transform(X_test['clean_subcategories'].values)
```

In [30]:

```
print("Shape of X_train after one hot encodig ",X_train_sub_cat_ohe.shape, y_train.shap
e)
print("Shape of X_cv after one hot encodig ",X_cv_sub_cat_ohe.shape, y_cv.shape)
print("Shape of X_test after one hot encodig ",X_test_sub_cat_ohe.shape, y_test.shape)
```

```
Shape of X_train after one hot encodig  (22445, 30) (22445,)
Shape of X_cv after one hot encodig  (11055, 30) (11055,)
Shape of X_test after one hot encodig  (16500, 30) (16500,)
```

School state

In [31]:

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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
school_dict = dict(my_counter)
sorted_school_dict = dict(sorted(school_dict.items(), key=lambda kv: kv[1]))
```

In [32]:

```
vec3 = CountVectorizer(vocabulary=list(sorted_school_dict.keys()), lowercase=False, bin
ary=True)
X_train_state_ohe = vec3.fit_transform(X_train['school_state'].values)
X_cv_state_ohe = vec3.transform(X_cv['school_state'].values)
X_test_state_ohe = vec3.transform(X_test['school_state'].values)
```

In [33]:

```
print("Shape of X_train after one hot encodig ",X_train_state_ohe.shape, y_train.shape)
print("Shape of X_cv after one hot encodig ",X_cv_state_ohe.shape, y_cv.shape)
print("Shape of X_test after one hot encodig ",X_test_state_ohe.shape, y_test.shape)
```

```
Shape of X_train after one hot encodig  (22445, 51) (22445,)
Shape of X_cv after one hot encodig  (11055, 51) (11055,)
Shape of X_test after one hot encodig  (16500, 51) (16500,)
```

Teacher prefix

In [34]:

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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
teacher_dict = dict(my_counter)
sorted_teacher_dict = dict(sorted(teacher_dict.items(), key=lambda kv: kv[1]))
```

In [35]:

```
vec4 = CountVectorizer(vocabulary=list(sorted_teacher_dict.keys()), lowercase=False, bi
nary=True)
X_train_teacher_ohe = vec4.fit_transform(X_train['teacher_prefix'].values) # fit has to
happen only on train data
X_cv_teacher_ohe = vec4.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vec4.transform(X_test['teacher_prefix'].values)
```

In [36]:

```
print("Shape of X_train after one hot encodig ",X_train_teacher_ohe.shape, y_train.shap
e)
print("Shape of X_cv after one hot encodig ",X_cv_teacher_ohe.shape, y_cv.shape)
print("Shape of X_test after one hot encodig ",X_test_teacher_ohe.shape, y_test.shape)
```

```
Shape of X_train after one hot encodig  (22445, 5) (22445,)
Shape of X_cv after one hot encodig  (11055, 5) (11055,)
Shape of X_test after one hot encodig  (16500, 5) (16500,)
```

Grade category

In [37]:

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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
grade_dict = dict(my_counter)
sorted_grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))
```

In [38]:

```
vec5 = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=False, bina
ry=True)
X_train_grade_ohe = vec5.fit_transform(X_train['project_grade_category'].values) # fit
 has to happen only on train data
X_cv_grade_ohe = vec5.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vec5.transform(X_test['project_grade_category'].values)
```

In [39]:

```
print("Shape of X_train after one hot encodig ",X_train_grade_ohe.shape, y_train.shape)
print("Shape of X_cv after one hot encodig ",X_cv_grade_ohe.shape, y_cv.shape)
print("Shape of X_test after one hot encodig ",X_test_grade_ohe.shape, y_test.shape)
```

```
Shape of X_train after one hot encodig  (22445, 5) (22445,)
Shape of X_cv after one hot encodig  (11055, 5) (11055,)
Shape of X_test after one hot encodig  (16500, 5) (16500,)
```

Bag of words for essay

In [40]:

```
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
vec6 = CountVectorizer(min_df=10)

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vec6.fit_transform(preprocessed_train_essay) # fit has to happen on
ly on train data
X_cv_essay_bow = vec6.transform(preprocessed_cv_essay)
X_test_essay_bow = vec6.transform(preprocessed_test_essay)
```

In [41]:

```
print("Shape of matrix after one hot encodig",X_train_essay_bow.shape, y_train.shape)
print("Shape of matrix after one hot encodig",X_cv_essay_bow.shape, y_cv.shape)
print("Shape of matrix after one hot encodig",X_test_essay_bow.shape, y_test.shape)
```

```
Shape of matrix after one hot encodig (22445, 8775) (22445,)
Shape of matrix after one hot encodig (11055, 8775) (11055,)
Shape of matrix after one hot encodig (16500, 8775) (16500,)
```

Bag of words for titles

In [42]:

```
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
from sklearn.feature_extraction.text import TfidfVectorizer
vec7 = CountVectorizer(min_df=10)


# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vec7.fit_transform(preprocessed_train_title)# fit has to happen onl
y on train data
X_cv_title_bow = vec7.transform(preprocessed_cv_title)
X_test_title_bow = vec7.transform(preprocessed_test_title)
```

In [43]:

```
print("Shape of matrix after one hot encodig",X_train_title_bow.shape, y_train.shape)
print("Shape of matrix after one hot encodig",X_cv_title_bow.shape, y_cv.shape)
print("Shape of matrix after one hot encodig",X_test_title_bow.shape, y_test.shape)
```

```
Shape of matrix after one hot encodig (22445, 1149) (22445,)
Shape of matrix after one hot encodig (11055, 1149) (11055,)
Shape of matrix after one hot encodig (16500, 1149) (16500,)
```

tfidf featuring for essay

In [44]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vec8 = TfidfVectorizer(min_df=10)

# we use the fitted Tfidf Vectorizer to convert the text to vector
X_train_ess_tfidf = vec8.fit_transform(preprocessed_train_essay)# fit has to happen onl
y on train data
X_cv_ess_tfidf = vec8.transform(preprocessed_cv_essay)
X_test_ess_tfidf = vec8.transform(preprocessed_test_essay)
```

In [45]:

```
print("Shape of matrix after one hot encodig",X_train_ess_tfidf.shape, y_train.shape)
print("Shape of matrix after one hot encodig",X_cv_ess_tfidf.shape, y_cv.shape)
print("Shape of matrix after one hot encodig",X_test_ess_tfidf.shape, y_test.shape)
```

```
Shape of matrix after one hot encodig (22445, 8775) (22445,)
Shape of matrix after one hot encodig (11055, 8775) (11055,)
Shape of matrix after one hot encodig (16500, 8775) (16500,)
```

tfidf featuring for titles

In [46]:

```
vec9 = TfidfVectorizer(min_df=10)
# we use the fitted Tfidf Vectorizer to convert the text to vector

X_train_title_tfidf = vec9.fit_transform(preprocessed_train_title)# fit has to happen o
nly on train data
X_cv_title_tfidf = vec9.transform(preprocessed_cv_title)
X_test_title_tfidf = vec9.transform(preprocessed_test_title)
```

In [47]:

```
print("Shape of matrix after one hot encodig",X_train_title_tfidf.shape, y_train.shape)
print("Shape of matrix after one hot encodig",X_cv_title_tfidf.shape, y_cv.shape)
print("Shape of matrix after one hot encodig",X_test_title_tfidf.shape, y_test.shape)
```

```
Shape of matrix after one hot encodig (22445, 1149) (22445,)
Shape of matrix after one hot encodig (11055, 1149) (11055,)
Shape of matrix after one hot encodig (16500, 1149) (16500,)
```

Using Pretrained Model: avgw2v

In [48]:

```
# 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
```

In [49]:

```
model = loadGloveModel('glove.42B.300d.txt')
```

```
144it [00:00, 1426.63it/s]

Loading Glove Model

1917494it [07:52, 4060.62it/s]

Done. 1917494  words loaded!
```

In [50]:

```
glove_words = set(model.keys())
```

In [51]:

```
# compute average word2vec for each review.
def func(wordlist):

    train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in thi
s list
    for sentence in tqdm(wordlist): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length    # we are taking
 the 300 dimensions  very large
        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]))
    return train_avg_w2v_vectors
```

avgw2v for essay

In [52]:

```
X_train_avg_w2v_ess=func(preprocessed_train_essay)
X_cv_avg_w2v_ess=func(preprocessed_cv_essay)
X_test_avg_w2v_ess=func(preprocessed_test_essay)
```

```
100%|████████████████████████████████████████████████████████████████
█████| 22445/22445 [00:11<00:00, 1922.43it/s]
  7%|████▉
| 746/11055 [00:00<00:05, 1848.84it/s]

22445
300

100%|████████████████████████████████████████████████████████████████
█████| 11055/11055 [00:05<00:00, 1912.07it/s]
  5%|███
| 774/16500 [00:00<00:08, 1900.10it/s]

11055
300

100%|████████████████████████████████████████████████████████████████
█████| 16500/16500 [00:08<00:00, 1978.83it/s]

16500
300
```

avgw2v for titles

In [53]:

```
# FOR TITLES
X_train_avg_w2v_title=func(preprocessed_train_title)
X_cv_avg_w2v_title=func(preprocessed_cv_title)
X_test_avg_w2v_title=func(preprocessed_test_title)
```

```
100%|████████████████████████████████████████████████████████████████
████| 22445/22445 [00:00<00:00, 30764.15it/s]
 28%|████████████████████
| 3044/11055 [00:00<00:00, 30156.75it/s]

22445
300

100%|████████████████████████████████████████████████████████████████
████| 11055/11055 [00:00<00:00, 30388.25it/s]
 36%|████████████████████████
| 6020/16500 [00:00<00:00, 27407.12it/s]

11055
300

100%|████████████████████████████████████████████████████████████████
████| 16500/16500 [00:00<00:00, 33151.49it/s]

16500
300
```

In [54]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_train_essay)
# 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 [55]:

```
# tfidf Word2Vec
# compute tfidf word2vec for each review.
def tf_idf_done(word_list):
    train_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is store
d in this list
    for sentence in tqdm(word_list): # 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():#.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words):
                vec = model[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and the tf value
((sentence.count(word)/len(sentence.split())))
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
# getting the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
                if tf_idf_weight != 0:
                    vector /= tf_idf_weight
                    train_title_tfidf_w2v_vectors.append(vector)

    print(len(train_title_tfidf_w2v_vectors))
    print(len(train_title_tfidf_w2v_vectors[0]))
    return train_title_tfidf_w2v_vectors
```

tf-idf- w2v for essay

In [56]:

```
#For essays
X_train_tfidf_w2v_ess =tf_idf_done(preprocessed_train_essay)
X_test_tfidf_w2v_ess =tf_idf_done(preprocessed_test_essay)
X_cv_tfidf_w2v_ess =tf_idf_done(preprocessed_cv_essay)
```

```
100%|████████████████████████████████████████████████████████████████████
██████| 22445/22445 [01:32<00:00, 241.38it/s]
  1%|▉
| 150/16500 [00:00<01:07, 242.09it/s]

3082893
300

100%|████████████████████████████████████████████████████████████████████
██████| 16500/16500 [01:08<00:00, 240.12it/s]
  4%|███
| 390/11055 [00:01<00:42, 253.49it/s]

2252139
300

100%|████████████████████████████████████████████████████████████████████
██████| 11055/11055 [00:45<00:00, 245.05it/s]

1508499
300
```

tf-idf-w2v for titles

In [57]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_train_title)
# 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 [58]:

```
# tfidf Word2Vec
# compute tfidf word2vec for each review.
def tf_idf_done(word_list):
    train_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is store
d in this list
    for sentence in tqdm(word_list): # 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():#.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words):
                vec = model[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and the tf value
((sentence.count(word)/len(sentence.split())))
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
# getting the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
                if tf_idf_weight != 0:
                    vector /= tf_idf_weight
                    train_title_tfidf_w2v_vectors.append(vector)

    print(len(train_title_tfidf_w2v_vectors))
    print(len(train_title_tfidf_w2v_vectors[0]))
    return train_title_tfidf_w2v_vectors
```

In [59]:

```
# For Titles
X_train_tfidf_w2v_title=tf_idf_done(preprocessed_train_title)
X_test_tfidf_w2v_title=tf_idf_done(preprocessed_test_title)
X_cv_tfidf_w2v_title=tf_idf_done(preprocessed_cv_title)
```

```
100%|████████████████████████████████████████████████████████████
████| 22445/22445 [00:02<00:00, 10208.09it/s]
 13%|████████
| 2137/16500 [00:00<00:01, 10789.90it/s]

81908
300

100%|████████████████████████████████████████████████████████████
████| 16500/16500 [00:01<00:00, 11157.97it/s]
 20%|████████████
| 2219/11055 [00:00<00:00, 11218.00it/s]

57090
300

100%|████████████████████████████████████████████████████████████
████| 11055/11055 [00:01<00:00, 10435.20it/s]


38604
300
```

Vectorizing numerical data

In [60]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-al
l-groups-in-one-step
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_i
ndex()
price_data.head(2)
```

Out[60]:

	id	price	quantity
0	p000027	782.13	15
1	p000052	114.98	2

In [61]:

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

In [62]:

```
# https://stackoverflow.com/questions/32617811/imputation-of-missing-values-for-categor
ies-in-pandas
#replacing nan with most frequently occuring element
project_data['price'] = project_data['price'].fillna(project_data['price'].mode().iloc[
0])
```

In [63]:

```
X_train =pd.merge(X_train,price_data, how ='left', on = 'id')
X_cv =pd.merge(X_cv,price_data, how ='left', on ='id')
X_test = pd.merge(X_test,price_data,how ='left',on='id')
```

In [64]:

```
# https://stackoverflow.com/questions/32617811/imputation-of-missing-values-for-categor
ies-in-pandas
#replacing nan with most frequently occuring element
X_train['price'] = X_train['price'].fillna(X_train['price'].mode().iloc[0])
X_cv['price'] = X_cv['price'].fillna(X_train['price'].mode().iloc[0])
X_test['price'] = X_test['price'].fillna(X_test['price'].mode().iloc[0])
print(X_train['price'].isnull().sum())
print(X_cv['price'].isnull().sum())
print(X_test['price'].isnull().sum())
```

```
0
0
0
```

Standardizing price, quantity, previous projects

In [65]:

```
# price
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.pr
eprocessing.StandardScaler.html
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.   ...
 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()

X_train_price_stndrd = price_scalar.fit_transform(X_train['price'].values.reshape(-1,1
))
X_cv_price_stndrd = price_scalar.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_stndrd = price_scalar.transform(X_test['price'].values.reshape(-1,1))
```

In [66]:

```
project_data['quantity'] = project_data['quantity'].fillna(project_data['quantity'].mod
e().iloc[0])
```

In [67]:

```
# https://stackoverflow.com/questions/32617811/imputation-of-missing-values-for-categor
ies-in-pandas
#replacing nan with most frequently occuring element

X_train['quantity'] = X_train['quantity'].fillna(X_train['quantity'].mode().iloc[0])
X_cv['quantity'] = X_cv['quantity'].fillna(X_cv['quantity'].mode().iloc[0])
X_test['quantity'] = X_test['quantity'].fillna(X_test['quantity'].mode().iloc[0])
```

In [68]:

```
# quantity
quantity_scalar = StandardScaler()
X_train_quantity_stndrd = quantity_scalar.fit_transform(X_train['quantity'].values.resh
ape(-1,1))

X_cv_quantity_stndrd = quantity_scalar.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity_stndrd = quantity_scalar.transform(X_test['quantity'].values.reshape(-1
,1))
```

In [69]:

```
# previous_year_projects
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape
(-1,1)) # finding the mean and standard deviation of this data
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var
_[0])}")
# Now standardize the data with above mean and variance.
X_train_prev_proj_stndrd =price_scalar.transform(X_train['teacher_number_of_previously_
posted_projects'].values.reshape(-1,1))
# Now standardize the data with above mean and variance.
X_test_prev_proj_stndrd =price_scalar.transform(X_test['teacher_number_of_previously_po
sted_projects'].values.reshape(-1, 1))
# Now standardize the data with above mean and variance.
X_cv_prev_proj_stndrd = price_scalar.transform(X_cv['teacher_number_of_previously_poste
d_projects'].values.reshape(-1, 1))
```

```
C:\Users\SUBHODAYA KUMAR\Anaconda3\lib\site-packages\sklearn\utils\validat
ion.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\SUBHODAYA KUMAR\Anaconda3\lib\site-packages\sklearn\utils\validat
ion.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\SUBHODAYA KUMAR\Anaconda3\lib\site-packages\sklearn\utils\validat
ion.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\SUBHODAYA KUMAR\Anaconda3\lib\site-packages\sklearn\utils\validat
ion.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

In [70]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_train_bow = hstack((X_train_title_bow,X_train_essay_bow,X_train_teacher_ohe,X_train_c
at_ohe,X_train_sub_cat_ohe,
                      X_train_grade_ohe,X_train_state_ohe,X_train_price_stndrd, X_train
_quantity_stndrd,X_train_prev_proj_stndrd))

# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_cv_bow = hstack((X_cv_title_bow,X_cv_essay_bow,X_cv_teacher_ohe,X_cv_cat_ohe,X_cv_sub
_cat_ohe,
                      X_cv_grade_ohe,X_cv_state_ohe,X_cv_price_stndrd, X_cv_quantity_st
ndrd,X_cv_prev_proj_stndrd))

# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_test_bow = hstack((X_test_title_bow,X_test_essay_bow,X_test_teacher_ohe,X_test_cat_oh
e,X_test_sub_cat_ohe,
                      X_test_grade_ohe,X_test_state_ohe,X_test_price_stndrd, X_test_qua
ntity_stndrd,X_test_prev_proj_stndrd))
```

In [71]:

```
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_train.shape)
```

```
(22445, 10027) (22445,)
(11055, 10027) (11055,)
(16500, 10027) (22445,)
```

In [72]:

```
X_train_bow = X_train_bow.tocsr()[0:22445]
X_cv_bow = X_cv_bow.tocsr()[0:11055]
X_test_bow = X_test_bow.tocsr()[0:16500]
```

In [73]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_train_tfidf = hstack((X_train_title_tfidf,X_train_ess_tfidf,X_train_teacher_ohe,X_tra
in_cat_ohe,X_train_sub_cat_ohe,
                        X_train_grade_ohe,X_train_state_ohe,X_train_price_stndrd, X_tra
in_quantity_stndrd,X_train_prev_proj_stndrd))

# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_cv_tfidf = hstack((X_cv_title_tfidf,X_cv_ess_tfidf,X_cv_teacher_ohe,X_cv_cat_ohe,X_cv
_sub_cat_ohe,
                      X_cv_grade_ohe,X_cv_state_ohe,X_cv_price_stndrd, X_cv_quantity_st
ndrd,X_cv_prev_proj_stndrd))

# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_test_tfidf = hstack((X_test_title_tfidf,X_test_ess_tfidf,X_test_teacher_ohe,X_test_ca
t_ohe,X_test_sub_cat_ohe,
                      X_test_grade_ohe,X_test_state_ohe,X_test_price_stndrd, X_test_qua
ntity_stndrd,X_test_prev_proj_stndrd))
```

In [74]:

```
print(X_train_tfidf.shape, y_train.shape)
print(X_cv_tfidf.shape, y_cv.shape)
print(X_test_tfidf.shape, y_test.shape)
```

```
(22445, 10027) (22445,)
(11055, 10027) (11055,)
(16500, 10027) (16500,)
```

In [75]:

```
X_train_tfidf = X_train_tfidf.tocsr()[0:22445]
X_cv_tfidf = X_cv_tfidf.tocsr()[0:11055]
X_test_tfidf = X_test_tfidf.tocsr()[0:16500]
```

In [76]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_train_avg_w2v = hstack((X_train_avg_w2v_title,X_train_avg_w2v_ess,X_train_teacher_ohe
,X_train_cat_ohe,X_train_sub_cat_ohe,
                          X_train_grade_ohe,X_train_state_ohe,X_train_price_stndrd, X_t
rain_quantity_stndrd))

# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_cv_avg_w2v = hstack((X_cv_avg_w2v_title,X_cv_avg_w2v_ess,X_cv_teacher_ohe,X_cv_cat_oh
e,X_cv_sub_cat_ohe,
                       X_cv_grade_ohe,X_cv_state_ohe,X_cv_price_stndrd, X_cv_quantity_s
tndrd))

# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_test_avg_w2v = hstack((X_test_avg_w2v_title,X_test_avg_w2v_ess,X_test_teacher_ohe,X_t
est_cat_ohe,X_test_sub_cat_ohe,
                         X_test_grade_ohe,X_test_state_ohe,X_test_price_stndrd, X_test_
quantity_stndrd))
```

In [77]:

```
print(X_train_avg_w2v.shape, y_train.shape)
print(X_cv_avg_w2v.shape, y_cv.shape)
print(X_test_avg_w2v.shape, y_test.shape)
```

```
(22445, 702) (22445,)
(11055, 702) (11055,)
(16500, 702) (16500,)
```

In [78]:

```
X_train_avg_w2v = X_train_avg_w2v.tocsr()[0:22445]
X_cv_avg_w2v = X_cv_avg_w2v.tocsr()[0:11055]
X_test_avg_w2v = X_test_avg_w2v.tocsr()[0:16500]
```

I'm having memory constrain so reduced the size while performing tfidf_w2v

In [79]:

```
X_train_tfidf_w2v_title_1 =X_train_tfidf_w2v_title[0:5000]
X_train_tfidf_w2v_ess_1 = X_train_cat_ohe[0:5000]
X_train_teacher_ohe_1= X_train_teacher_ohe[0:5000]
X_train_cat_ohe_1 =X_train_tfidf_w2v_ess[0:5000]
X_train_sub_cat_ohe_1 =X_train_sub_cat_ohe[0:5000]
X_train_grade_ohe_1 =  X_train_grade_ohe[0:5000]
X_train_state_ohe_1 = X_train_state_ohe[0:5000]
X_train_price_stndrd_1 = X_train_price_stndrd[0:5000]
X_train_quantity_stndrd_1 =X_train_quantity_stndrd[0:5000]
```

In [80]:

```
X_cv_tfidf_w2v_title_1 =X_cv_tfidf_w2v_title[0:5000]
X_cv_tfidf_w2v_ess_1 = X_cv_cat_ohe[0:5000]
X_cv_teacher_ohe_1= X_cv_teacher_ohe[0:5000]
X_cv_cat_ohe_1 =X_cv_tfidf_w2v_ess[0:5000]
X_cv_sub_cat_ohe_1 =X_cv_sub_cat_ohe[0:5000]
X_cv_grade_ohe_1 =  X_cv_grade_ohe[0:5000]
X_cv_state_ohe_1 = X_cv_state_ohe[0:5000]
X_cv_price_stndrd_1 = X_cv_price_stndrd[0:5000]
X_cv_quantity_stndrd_1 =X_cv_quantity_stndrd[0:5000]
```

In [81]:

```
X_test_tfidf_w2v_title_1 =X_test_tfidf_w2v_title[0:5000]
X_test_tfidf_w2v_ess_1 = X_test_cat_ohe[0:5000]
X_test_teacher_ohe_1= X_test_teacher_ohe[0:5000]
X_test_cat_ohe_1 =X_test_tfidf_w2v_ess[0:5000]
X_test_sub_cat_ohe_1 =X_test_sub_cat_ohe[0:5000]
X_test_grade_ohe_1 =  X_test_grade_ohe[0:5000]
X_test_state_ohe_1 = X_test_state_ohe[0:5000]
X_test_price_stndrd_1 = X_test_price_stndrd[0:5000]
X_test_quantity_stndrd_1 =X_test_quantity_stndrd[0:5000]
```

In [82]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_train_tfidf_w2v = hstack((X_train_tfidf_w2v_title_1,X_train_tfidf_w2v_ess_1,X_train_t
eacher_ohe_1,X_train_cat_ohe_1,X_train_sub_cat_ohe_1,
                            X_train_grade_ohe_1,X_train_state_ohe_1,X_train_price_stndr
d_1, X_train_quantity_stndrd_1))


# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_cv_tfidf_w2v = hstack((X_cv_tfidf_w2v_title_1,X_cv_tfidf_w2v_ess_1,X_cv_teacher_ohe_1
,X_cv_cat_ohe_1,X_cv_sub_cat_ohe_1,
                         X_cv_grade_ohe_1,X_cv_state_ohe_1,X_cv_price_stndrd_1, X_cv_qu
antity_stndrd_1))


# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_test_tfidf_w2v = hstack((X_test_tfidf_w2v_title_1,X_test_tfidf_w2v_ess_1,X_test_teach
er_ohe_1,X_test_cat_ohe_1,X_test_sub_cat_ohe_1,
                            X_test_grade_ohe_1,X_test_state_ohe_1,X_test_price_stndrd_1,
X_test_quantity_stndrd_1))
```

In [83]:

```
print(X_train_tfidf_w2v.shape, y_train.shape)
print(X_cv_tfidf_w2v.shape, y_cv.shape)
print(X_test_tfidf_w2v.shape, y_test.shape)
```

```
(5000, 702) (22445,)
(5000, 702) (11055,)
(5000, 702) (16500,)
```

In [84]:

```
X_train_tfidf_w2v = X_train_tfidf_w2v.tocsr()[0:5000]
X_cv_tfidf_w2v = X_cv_tfidf_w2v.tocsr()[0:5000]
X_test_tfidf_w2v = X_test_tfidf_w2v.tocsr()[0:5000]
```

In [85]:

```
y_train_1 = y_train[0:5000]
y_cv_1 = y_cv[0:5000]
y_test_1 = y_test[0:5000]
```

In [86]:

```
print(X_train_tfidf_w2v.shape, y_train_1.shape)
print(X_cv_tfidf_w2v.shape, y_cv_1.shape)
print(X_test_tfidf_w2v.shape, y_test_1.shape)
print('='*50)
```

```
(5000, 702) (5000,)
(5000, 702) (5000,)
(5000, 702) (5000,)
==================================================
```

SVM on Bag of words

In [87]:

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 =
 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [88]:

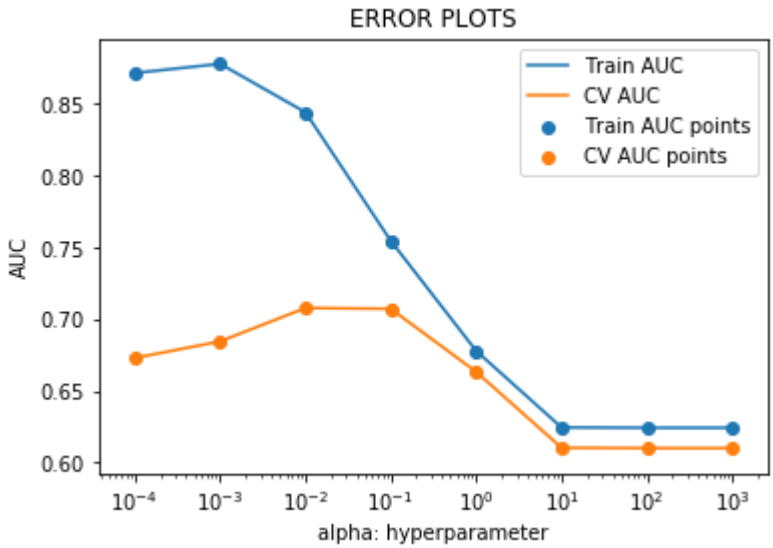
```
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn import svm
from sklearn.metrics import roc_auc_score
import math


train_auc = []
cv_auc = []
alpha =[0.0001,0.001,0.01,0.1,1,10,100,1000]

# hyperparameter tuning with l2 reg
for i  in   tqdm(alpha):
    sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced',alpha
= i)
    svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
    svm.fit(X_train_bow, y_train)# fit the model

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    y_train_pred=batch_predict(svm,X_train_bow)
    y_cv_pred=batch_predict(svm,X_cv_bow)
# roc curve
#Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from predicti
on scores.
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
plt.xscale('log')# we take the log in the x axis
plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|████████████████████████████████████████████████████████████| 8/8 [00:08<00:00,  1.08s/it]
```



In [89]:

```
optimal_alpha= alpha[cv_auc.index(max(cv_auc))]
alpha_values=[math.log(x) for x in alpha]
print('optimal alpha for which auc is maximum : ',optimal_alpha)
```

```
optimal alpha for which auc is maximum :  0.01
```

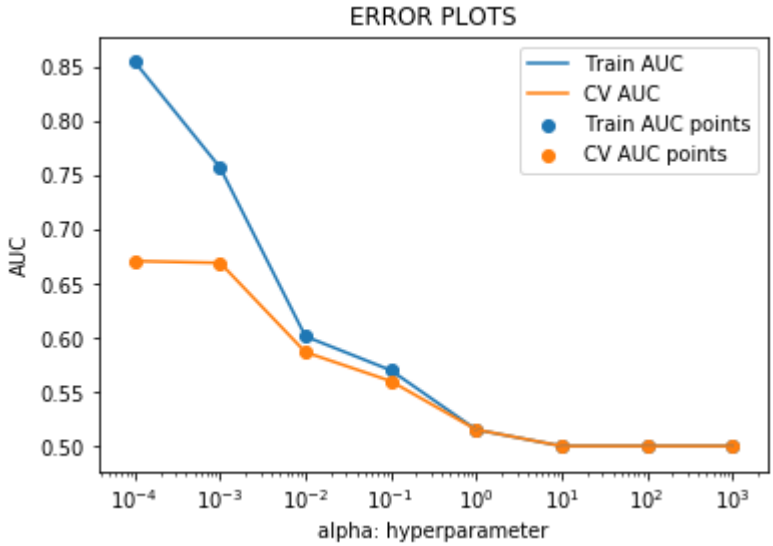
In [90]:

```
train_auc = []
cv_auc = []
alpha =[0.0001,0.001,0.01,0.1,1,10,100,1000]

# hyperparameter tuning with l2 reg
for i  in   tqdm(alpha):
    sd = SGDClassifier(loss = 'hinge',alpha = i, penalty = 'l1', class_weight = 'balanc
ed')
    svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
    svm.fit(X_train_bow, y_train)# fit the model

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    y_train_pred=batch_predict(svm,X_train_bow)
    y_cv_pred=batch_predict(svm,X_cv_bow)
# roc curve
#Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from predicti
on scores.
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
plt.xscale('log')# we take the log in the x axis
plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|██████████| 8/8 [00:11<00:00,  1.44s/it]
```



Hyperparameter tuning

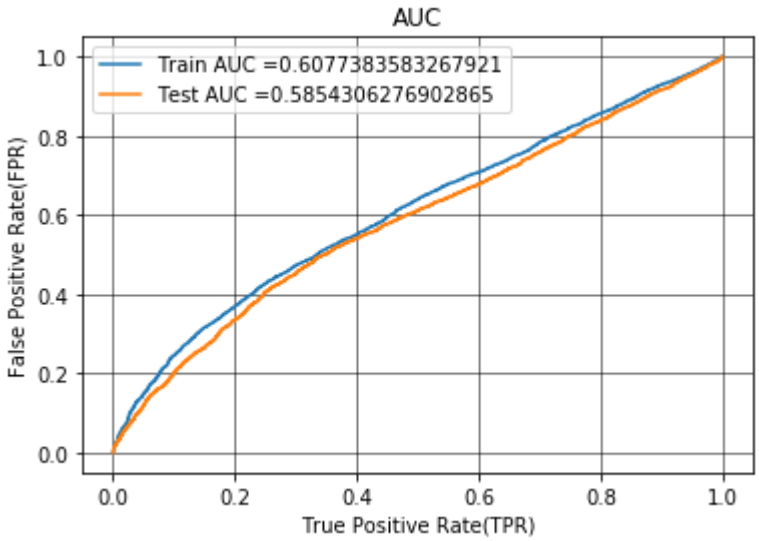
In [91]:

```
sd = SGDClassifier(penalty = 'l1', class_weight = 'balanced',alpha = optimal_alpha)
svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
svm.fit(X_train_bow, y_train)


y_train_pred = batch_predict(svm,X_train_bow)
y_test_pred = batch_predict(svm,X_test_bow)


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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



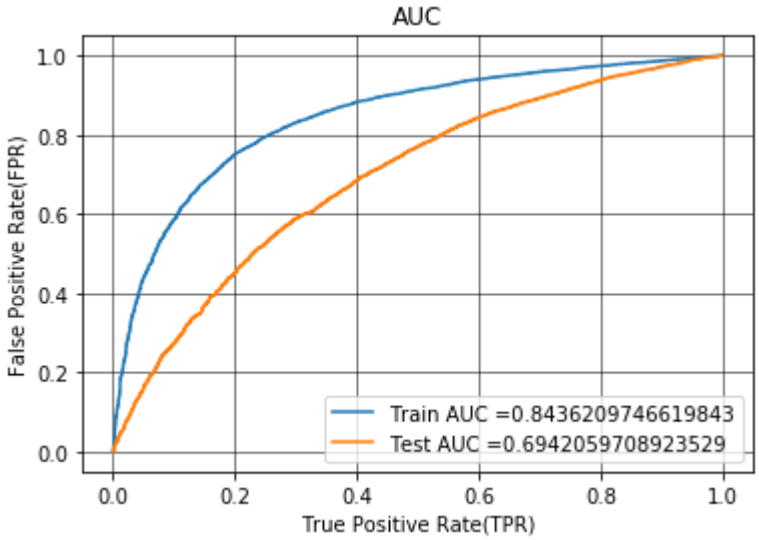
In [92]:

```
sd = SGDClassifier(penalty = 'l2', class_weight = 'balanced',alpha = optimal_alpha)
svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
svm.fit(X_train_bow, y_train)


y_train_pred = batch_predict(svm,X_train_bow)
y_test_pred = batch_predict(svm,X_test_bow)


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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



Preicting best Threshold

In [93]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou
nd(t,3))
    return t

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

Implementing Confusion Matrix

In [94]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
print('='*100)
```

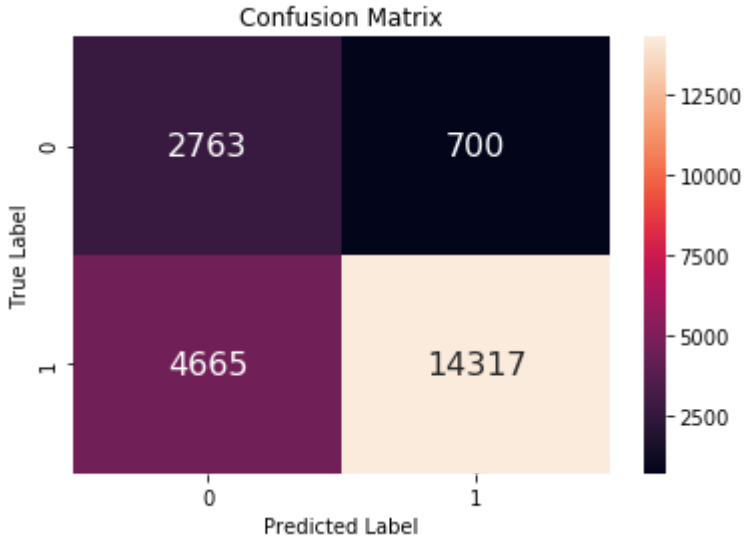
```
the maximum value of tpr*(1-fpr) 0.6017809689639254 for threshold 0.824
Train confusion matrix
[[ 2763   700]
 [ 4665 14317]]
Test confusion matrix
[[1542 1004]
 [4490 9464]]
====================================================================================
==========================
```

In [95]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-mix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[95]:

```
Text(0.5, 1, 'Confusion Matrix')
```

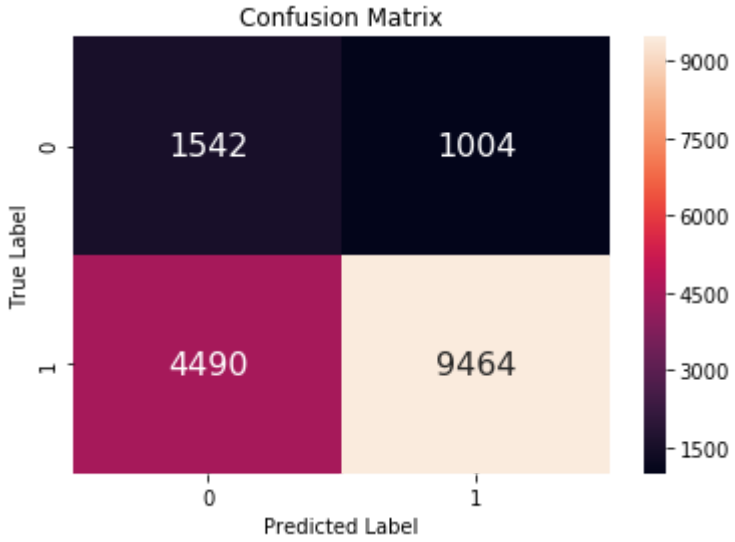


In [96]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-mix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[96]:

```
Text(0.5, 1, 'Confusion Matrix')
```



SVM on tfidf

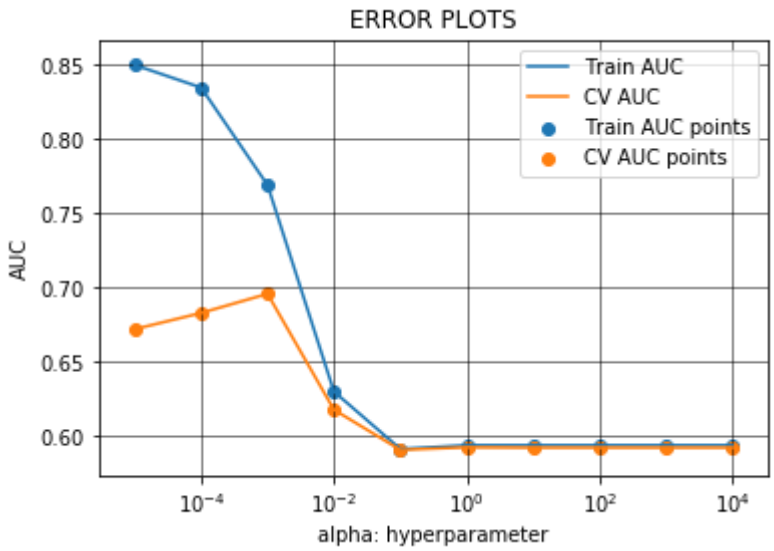
In [97]:

```
train_auc = []
cv_auc = []
alpha =[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

# hyperparameter tuning with l2 reg
for i  in   tqdm(alpha):
    sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced',alpha
= i)
    svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
    svm.fit(X_train_tfidf, y_train)# fit the model

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    y_train_pred=batch_predict(svm,X_train_tfidf)
    y_cv_pred=batch_predict(svm,X_cv_tfidf)
# roc curve
#Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from predicti
on scores.
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
plt.xscale('log')# we take the log in the x axis
plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

```
100%|██████████| 10/10 [00:10<00:00,  1.04s/it]
```



In [98]:

```
optimal_alpha= alpha[cv_auc.index(max(cv_auc))]
alpha_values=[math.log(x) for x in alpha]
print('optimal alpha for which auc is maximum : ',optimal_alpha)
```

```
optimal alpha for which auc is maximum :  0.001
```

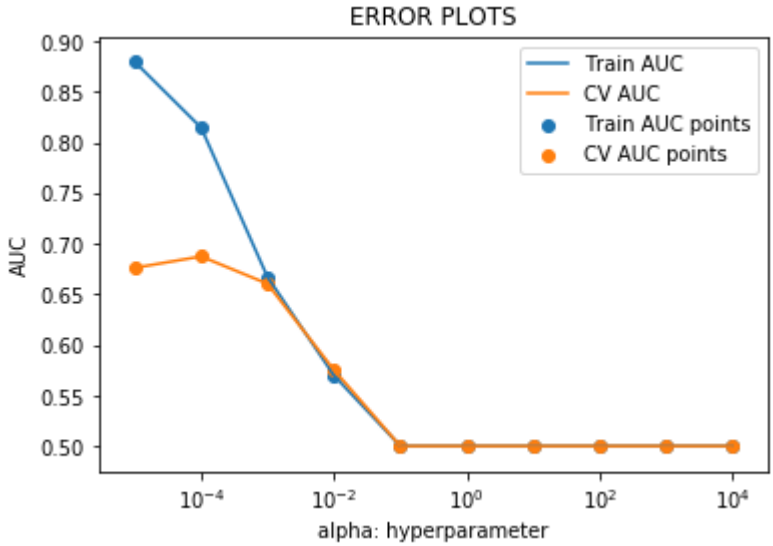
In [99]:

```
train_auc = []
cv_auc = []
alpha =[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

# hyperparameter tuning with l2 reg
for i  in   tqdm(alpha):
    sd = SGDClassifier(loss = 'hinge',alpha = i, penalty = 'l1', class_weight = 'balanc
ed')
    svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
    svm.fit(X_train_tfidf, y_train)# fit the model

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    y_train_pred=batch_predict(svm,X_train_tfidf)
    y_cv_pred=batch_predict(svm,X_cv_tfidf)
# roc curve
#Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from predicti
on scores.
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
plt.xscale('log')# we take the log in the x axis
plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
100%|████████████████████████████████████████████████████████████████████████████| 10/10 [00:14<00:00,  1.42s/it]
```



Hyperparameter tuning

In [100]:

```
sd = SGDClassifier(penalty = 'l1', class_weight = 'balanced',alpha = 10)
svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
svm.fit(X_train_tfidf, y_train)


y_train_pred = batch_predict(svm,X_train_tfidf)
y_test_pred = batch_predict(svm,X_test_tfidf)


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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

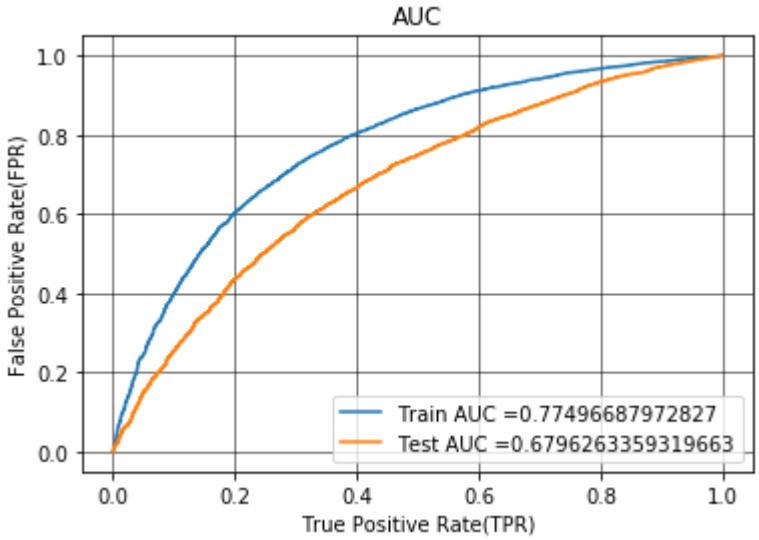
In [101]:

```
sd = SGDClassifier(penalty = 'l2', class_weight = 'balanced',alpha = optimal_alpha)
svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
svm.fit(X_train_tfidf, y_train)


y_train_pred = batch_predict(svm,X_train_tfidf)
y_test_pred = batch_predict(svm,X_test_tfidf)


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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



Predicting best threshold

In [102]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou
nd(t,3))
    return t

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

Implementing Confusion Matrix

In [103]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
print('='*100)
```

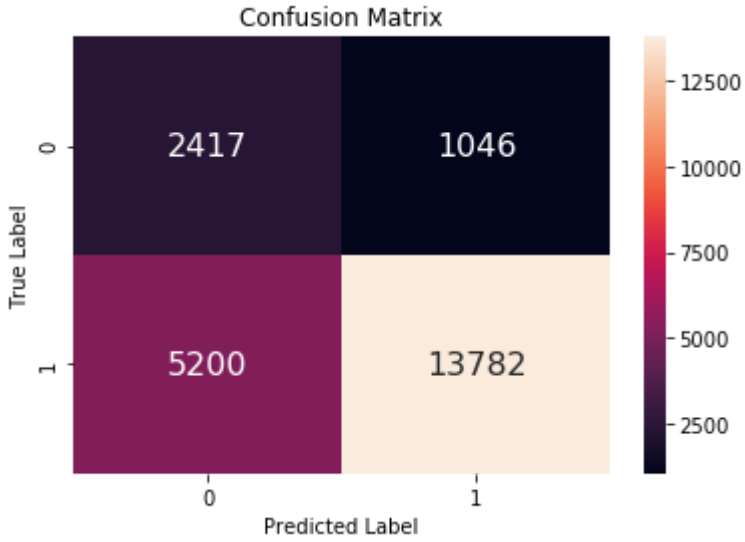
```
the maximum value of tpr*(1-fpr) 0.50675079112747 for threshold 0.827
Train confusion matrix
[[ 2417  1046]
 [ 5200 13782]]
Test confusion matrix
[[2063  483]
 [8118 5836]]
====================================================================================
==========================
```

In [104]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-mix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[104]:

```
Text(0.5, 1, 'Confusion Matrix')
```

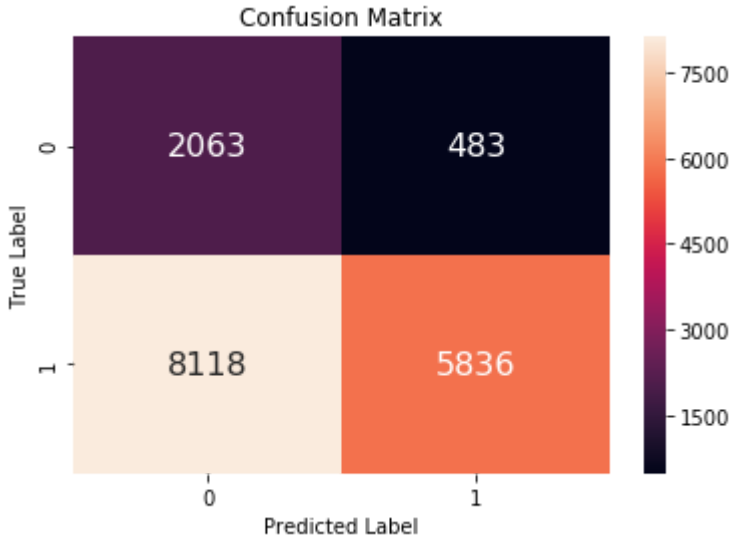


In [105]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-mix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[105]:

```
Text(0.5, 1, 'Confusion Matrix')
```



SVM on avgw2v

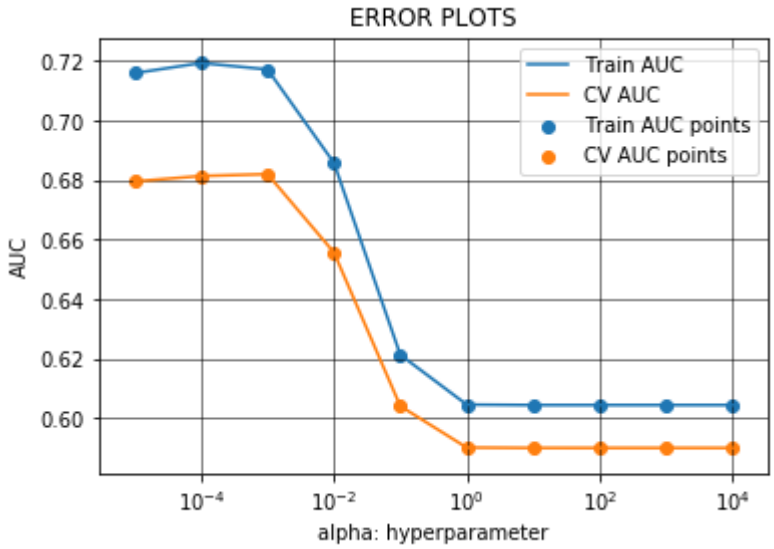
In [106]:

```
train_auc = []
cv_auc = []
alpha =[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

# hyperparameter tuning with l2 reg
for i  in   tqdm(alpha):
    sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced',alpha
= i)# takes the alpha from the i th list value
    svm = CalibratedClassifierCV(sd, cv= 5)
    svm.fit(X_train_avg_w2v, y_train)# fit the model

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    y_train_pred=batch_predict(svm,X_train_avg_w2v)
    y_cv_pred=batch_predict(svm,X_cv_avg_w2v)
# roc curve
#Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from predicti
on scores.
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
plt.xscale('log')# we take the log in the x axis
plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

```
100%|██████████| 10/10 [00:32<00:00,  3.25s/it]
```



In [107]:

```
optimal_alpha= alpha[cv_auc.index(max(cv_auc))]
alpha_values=[math.log(x) for x in alpha]
print('optimal alpha for which auc is maximum : ',optimal_alpha)
```

```
optimal alpha for which auc is maximum :  0.001
```

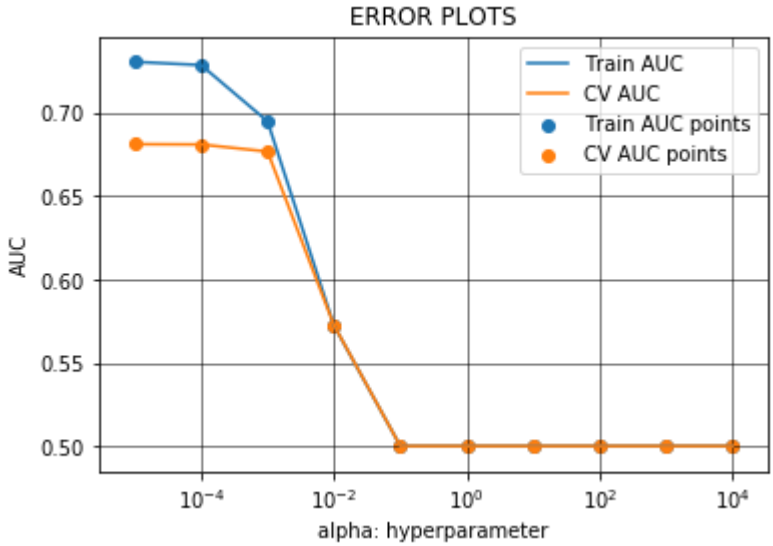
In [108]:

```
train_auc = []
cv_auc = []
alpha =[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

# hyperparameter tuning with l2 reg
for i  in  tqdm(alpha):
    sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced',alpha
= i)# takes the alpha from the i th list value
    svm = CalibratedClassifierCV(sd, cv= 5)
    svm.fit(X_train_avg_w2v, y_train)# fit the model

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    y_train_pred=batch_predict(svm,X_train_avg_w2v)
    y_cv_pred=batch_predict(svm,X_cv_avg_w2v)
# roc curve
#Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from predicti
on scores.
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
plt.xscale('log')# we take the log in the x axis
plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

100%|██| 10/10 [01:05<00:00, 6.59s/it]



In [109]:

```
sd = SGDClassifier(penalty = 'l1', class_weight = 'balanced',alpha = 10)# takes the alp
ha from the i th list value
svm = CalibratedClassifierCV(sd, cv= 5)
svm.fit(X_train_avg_w2v, y_train)


y_train_pred = batch_predict(svm,X_train_avg_w2v)
y_test_pred = batch_predict(svm,X_test_avg_w2v)


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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

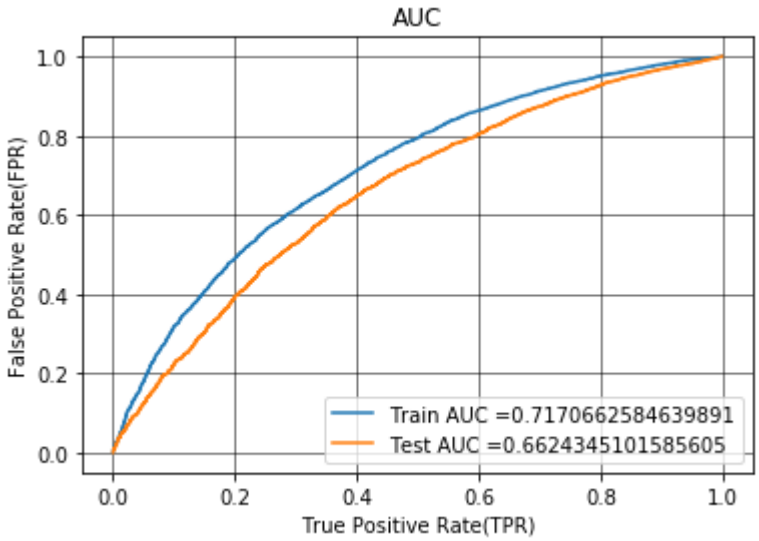
In [110]:

```
sd = SGDClassifier(penalty = 'l2', class_weight = 'balanced',alpha = optimal_alpha)
svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
svm.fit(X_train_avg_w2v, y_train)


y_train_pred = batch_predict(svm,X_train_avg_w2v)
y_test_pred = batch_predict(svm,X_test_avg_w2v)


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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



Predicting best threshold

In [111]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou
nd(t,3))
    return t

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

Implementing Confusion matrix

In [112]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
print('='*100)
```

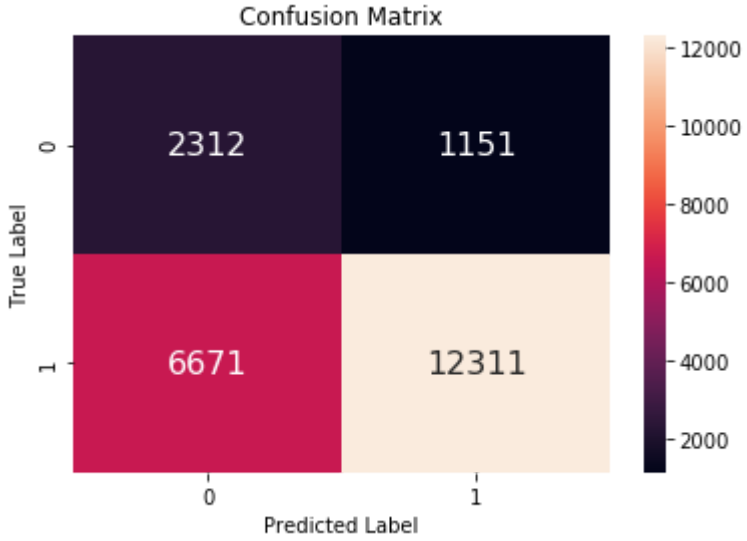
```
the maximum value of tpr*(1-fpr) 0.4329988076610901 for threshold 0.841
Train confusion matrix
[[ 2312  1151]
 [ 6671 12311]]
Test confusion matrix
[[1742  804]
 [6294 7660]]
====================================================================================
==========================
```

In [113]:

```
matrix = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[113]:

Text(0.5, 1, 'Confusion Matrix')

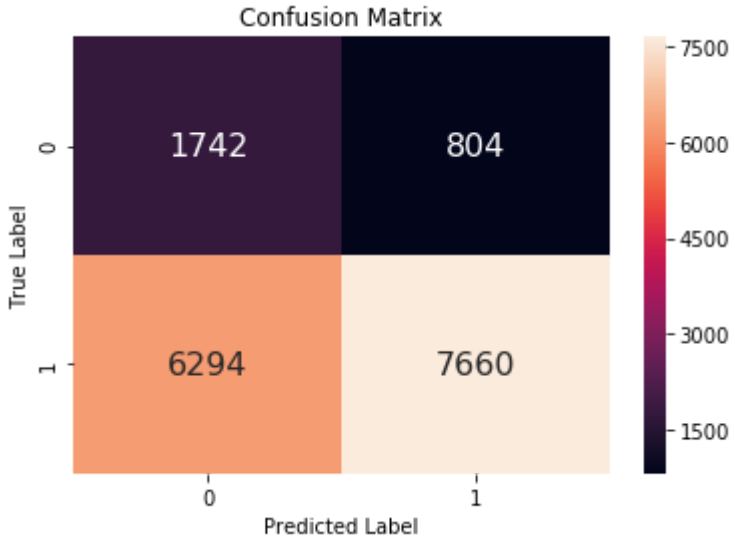


In [114]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-mix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[114]:

```
Text(0.5, 1, 'Confusion Matrix')
```



SVM on tfidf w2v

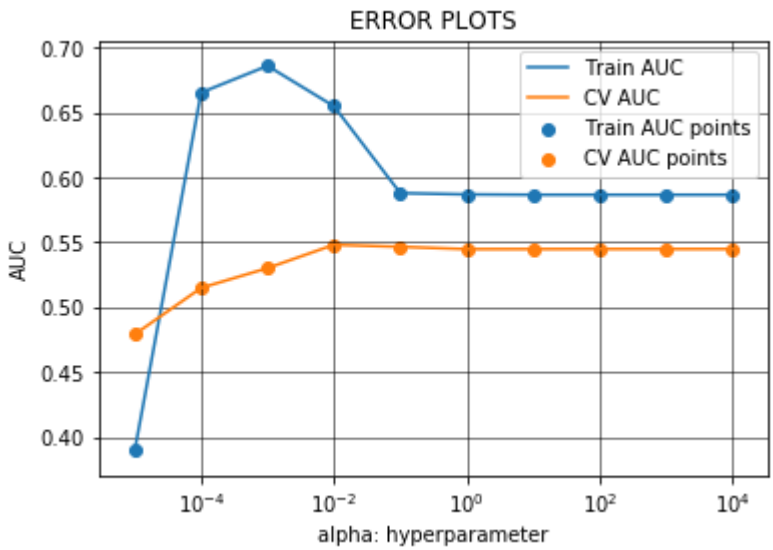
In [115]:

```
train_auc = []
cv_auc = []
alpha =[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

# hyperparameter tuning with l2 reg
for i  in   tqdm(alpha):
    sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced',alpha
= i)
    svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
    svm.fit(X_train_tfidf_w2v, y_train_1)# fit the model

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    y_train_pred=batch_predict(svm,X_train_tfidf_w2v)
    y_cv_pred=batch_predict(svm,X_cv_tfidf_w2v)
# roc curve
#Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from predicti
on scores.
    train_auc.append(roc_auc_score(y_train_1,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv_1, y_cv_pred))
plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
plt.xscale('log')# we take the log in the x axis
plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

```
100%|██████████| 10/10 [00:08<00:00,  1.23it/s]
```



In [116]:

```
optimal_alpha= alpha[cv_auc.index(max(cv_auc))]
alpha_values=[math.log(x) for x in alpha]
print('optimal alpha for which auc is maximum : ',optimal_alpha)
```

```
optimal alpha for which auc is maximum :  0.01
```

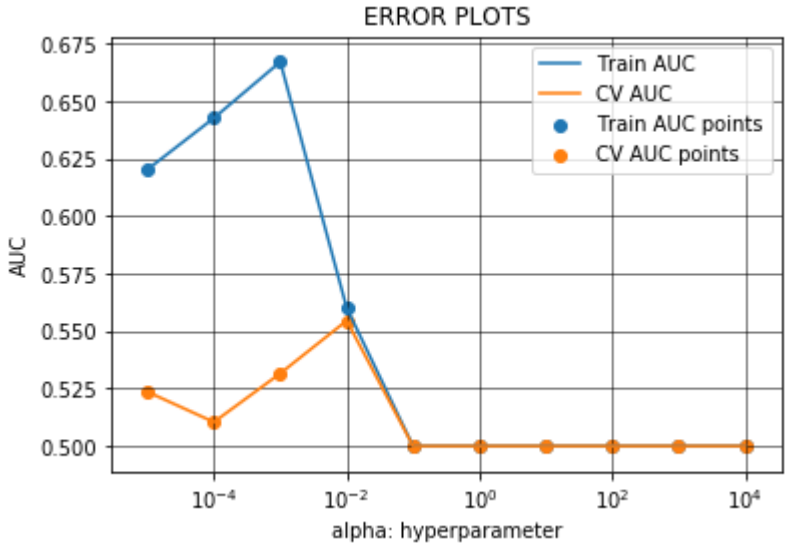
In [117]:

```
train_auc = []
cv_auc = []
alpha =[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

# hyperparameter tuning with l2 reg
for i  in   tqdm(alpha):
    sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced',alpha
= i)
    svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
    svm.fit(X_train_tfidf_w2v, y_train_1)# fit the model

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    y_train_pred=batch_predict(svm,X_train_tfidf_w2v)
    y_cv_pred=batch_predict(svm,X_cv_tfidf_w2v)
# roc curve
#Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from predicti
on scores.
    train_auc.append(roc_auc_score(y_train_1,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv_1, y_cv_pred))
plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
plt.xscale('log')# we take the log in the x axis
plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

```
100%|██████████| 10/10 [00:14<00:00,  1.44s/it]
```



In [118]:

```
sd = SGDClassifier(penalty = 'l1', class_weight = 'balanced',alpha = 0.1)
svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
svm.fit(X_train_tfidf_w2v, y_train_1)


y_train_pred = batch_predict(svm,X_train_tfidf_w2v)
y_test_pred = batch_predict(svm,X_test_tfidf_w2v)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_1, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_1, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

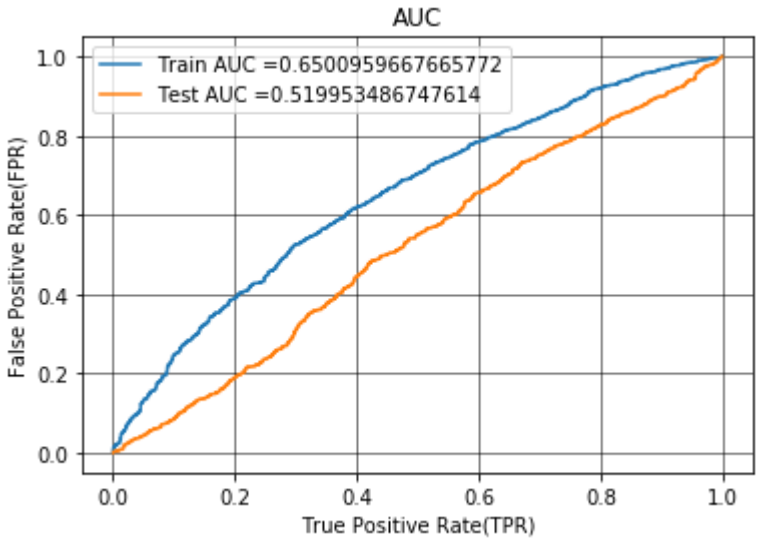
In [119]:

```
sd = SGDClassifier(penalty = 'l2', class_weight = 'balanced',alpha = optimal_alpha)
svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
svm.fit(X_train_tfidf_w2v, y_train_1)


y_train_pred = batch_predict(svm,X_train_tfidf_w2v)
y_test_pred = batch_predict(svm,X_test_tfidf_w2v)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_1, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_1, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



Predicting Threshold

In [120]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou
nd(t,3))
    return t

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

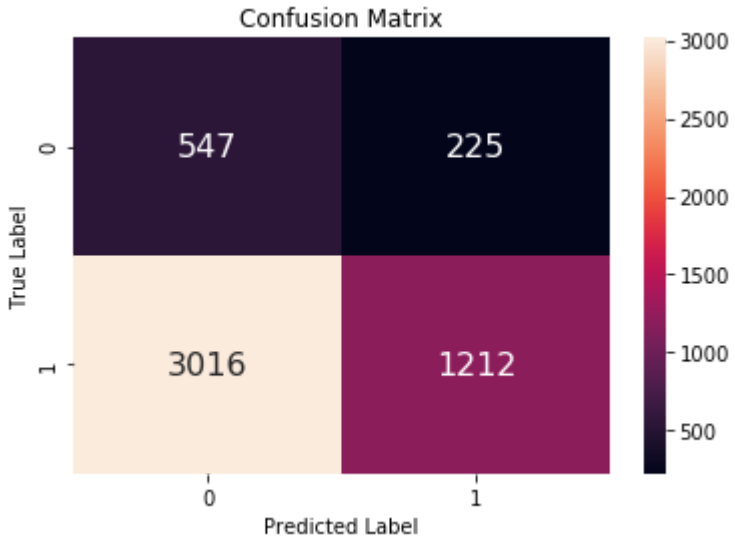
Implementing Confusion matrix

In [121]:

```
matrix = confusion_matrix(y_test_1, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[121]:

```
Text(0.5, 1, 'Confusion Matrix')
```

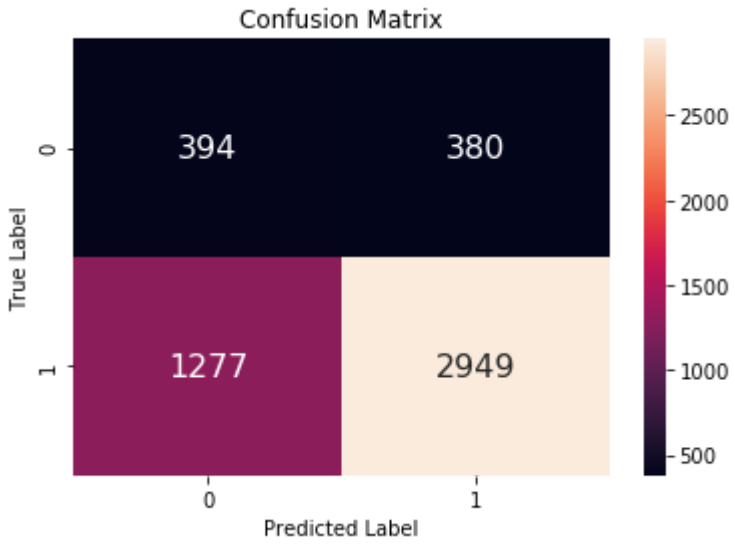


In [122]:

```
matrix = confusion_matrix(y_train_1, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[122]:

```
Text(0.5, 1, 'Confusion Matrix')
```



Number of words in title

In [123]:

```
# For train data
title_length_train=[]
for i in range(0,22445):
    title_length_train.append(len(X_train["project_title"][i].split()))

title_length_train=np.array(title_length_train)
X_train['title_length'] = title_length_train

#for test data titles
title_length_test=[]
for i in range(0,16500):
    title_length_test.append(len(X_test["project_title"][i].split()))
X_test['title_length'] = title_length_test

title_length_test=np.array(title_length_test)
#for cv data titles
title_length_cv=[]
for i in range(0,11055):
    title_length_cv.append(len(X_cv["project_title"][i].split()))

title_length_cv=np.array(title_length_cv)
X_cv['title_length'] = title_length_cv
```

Standardizing essay, title words

In [124]:

```
# title
title_scalar = StandardScaler()
X_train_title_stndrd = title_scalar.fit_transform(X_train['title_length'].values.reshap
e(-1,1))
X_cv_title_stndrd = title_scalar.transform(X_cv['title_length'].values.reshape(-1,1))
X_test_title_stndrd = title_scalar.transform(X_test['title_length'].values.reshape(-1,1
))
```

```
C:\Users\SUBHODAYA KUMAR\Anaconda3\lib\site-packages\sklearn\utils\validat
ion.py:595: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.

C:\Users\SUBHODAYA KUMAR\Anaconda3\lib\site-packages\sklearn\utils\validat
ion.py:595: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.

C:\Users\SUBHODAYA KUMAR\Anaconda3\lib\site-packages\sklearn\utils\validat
ion.py:595: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.

C:\Users\SUBHODAYA KUMAR\Anaconda3\lib\site-packages\sklearn\utils\validat
ion.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

Number of words in essay

In [125]:

```
# For train data
essay_length_train=[]
for i in range(0,22445):
    essay_length_train.append(len(X_train["essay"][i].split()))

essay_length_train=np.array(essay_length_train)
X_train['essay_length'] = essay_length_train

#for test data essays
essay_length_test=[]
for i in range(0,16500):
    essay_length_test.append(len(X_test["essay"][i].split()))
X_test['essay_length'] = essay_length_test

essay_length_test=np.array(essay_length_test)

#for cv data essays
essay_length_cv=[]
for i in range(0,11055):
    essay_length_cv.append(len(X_cv["essay"][i].split()))

essay_length_cv=np.array(essay_length_cv)
X_cv['essay_length'] = essay_length_cv
```

In [126]:

```
# essay
essay_scalar = StandardScaler()
X_train_essay_stndrd = essay_scalar.fit_transform(X_train['essay_length'].values.reshap
e(-1,1))
X_cv_essay_stndrd = essay_scalar.transform(X_cv['essay_length'].values.reshape(-1,1))
X_test_essay_stndrd = essay_scalar.transform(X_test['essay_length'].values.reshape(-1,1
))
```

```
C:\Users\SUBHODAYA KUMAR\Anaconda3\lib\site-packages\sklearn\utils\validat
ion.py:595: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.

C:\Users\SUBHODAYA KUMAR\Anaconda3\lib\site-packages\sklearn\utils\validat
ion.py:595: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.

C:\Users\SUBHODAYA KUMAR\Anaconda3\lib\site-packages\sklearn\utils\validat
ion.py:595: DataConversionWarning:

Data with input dtype int32 was converted to float64 by StandardScaler.

C:\Users\SUBHODAYA KUMAR\Anaconda3\lib\site-packages\sklearn\utils\validat
ion.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

Sentiment Analysis

In [127]:

```
# http://t-redactyl.io/blog/2017/04/applying-sentiment-analysis-with-vader-and-the-twitter-api.html
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

tr_compound = []
tr_pos = []
tr_neu = []
tr_neg = []

analyzer = SentimentIntensityAnalyzer()

for i in tqdm(X_train['essay']):
    tr_pos.append(analyzer.polarity_scores(i)['pos'])
    tr_neg.append(analyzer.polarity_scores(i)['neg'])
    tr_neu.append(analyzer.polarity_scores(i)['neu'])
    tr_compound.append(analyzer.polarity_scores(i)['compound'])
X_train['pos'] = tr_pos
X_train['neg'] = tr_neg
X_train['neu'] = tr_neu
X_train['comp'] = tr_compound
```

```
C:\Users\SUBHODAYA KUMAR\Anaconda3\lib\site-packages\nltk\twitter\__init_
_.py:20: UserWarning:

The twython library has not been installed. Some functionality from the tw
itter package will not be available.

100%|██████████████████████████████████████████████████████████████████
███████| 22445/22445 [10:06<00:00, 37.01it/s]
```

In [128]:

```
X_train.head()
```

Out[128]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_s
0	149746	p154132	dc236e9634b34b01534c4f9655c03aab	Ms.	CA
1	159233	p214777	adea34a8c5655a887d4c79c8d0dde87d	Mrs.	GA
2	20200	p211399	bbae1451babde02914f6b655bc9c2f4d	Mr.	TN
3	56437	p231930	59d56b4f92ac901a1b8d817dc472f869	Mrs.	TX
4	75500	p032379	97bcc5f892481a13a9ac2dfaa3b1cc44	Ms.	IL

5 rows × 25 columns

In [129]:

```
# http://t-redactyl.io/blog/2017/04/applying-sentiment-analysis-with-vader-and-the-twitter-api.html
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

ts_compound = []
ts_pos = []
ts_neu = []
ts_neg = []

analyzer = SentimentIntensityAnalyzer()

for i in tqdm(X_test['essay']):
    ts_pos.append(analyzer.polarity_scores(i)['pos'])
    ts_neg.append(analyzer.polarity_scores(i)['neg'])
    ts_neu.append(analyzer.polarity_scores(i)['neu'])
    ts_compound.append(analyzer.polarity_scores(i)['compound'])
X_test['pos'] = ts_pos
X_test['neg'] = ts_neg
X_test['neu'] = ts_neu
X_test['comp'] = ts_compound
```

```
100%|████████████████████████████████████████████████████████████████████
███████| 16500/16500 [06:43<00:00, 40.94it/s]
```

In [130]:

```
cv_compound = []
cv_pos = []
cv_neu = []
cv_neg = []
for i in tqdm(X_cv['essay']):
    cv_pos.append(analyzer.polarity_scores(i)['pos'])
    cv_neg.append(analyzer.polarity_scores(i)['neg'])
    cv_neu.append(analyzer.polarity_scores(i)['neu'])
    cv_compound.append(analyzer.polarity_scores(i)['compound'])
X_cv['pos'] = cv_pos
X_cv['neg'] = cv_neg
X_cv['neu'] = cv_neu
X_cv['comp'] = cv_compound
```

```
100%|████████████████████████████████████████████████████████████████████
███████| 11055/11055 [03:41<00:00, 49.91it/s]
```

Standardizing positive, negative, neutral, compound

In [131]:

```
# positive
pos_scalar = StandardScaler()
X_train_pos_stndrd = pos_scalar.fit_transform(X_train['pos'].values.reshape(-1,1))
X_cv_pos_stndrd = pos_scalar.transform(X_cv['pos'].values.reshape(-1,1))
X_test_pos_stndrd = pos_scalar.transform(X_test['pos'].values.reshape(-1,1))
```

In [132]:

```
# negative
neg_scalar = StandardScaler()
X_train_neg_stndrd = neg_scalar.fit_transform(X_train['neg'].values.reshape(-1,1))
X_cv_neg_stndrd = neg_scalar.transform(X_cv['neg'].values.reshape(-1,1))
X_test_neg_stndrd = neg_scalar.transform(X_test['neg'].values.reshape(-1,1))
```

In [133]:

```
# neutral
neu_scalar = StandardScaler()
X_train_neu_stndrd = neu_scalar.fit_transform(X_train['neu'].values.reshape(-1,1))
X_cv_neu_stndrd = neu_scalar.transform(X_cv['neu'].values.reshape(-1,1))
X_test_neu_stndrd = neu_scalar.transform(X_test['neu'].values.reshape(-1,1))
```

In [134]:

```
# compound
comp_scalar = StandardScaler()
X_train_comp_stndrd = comp_scalar.fit_transform(X_train['comp'].values.reshape(-1,1))
X_cv_comp_stndrd = comp_scalar.transform(X_cv['comp'].values.reshape(-1,1))
X_test_comp_stndrd = comp_scalar.transform(X_test['comp'].values.reshape(-1,1))
```

In [135]:

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx

# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_train_tfidf_1 = hstack((X_train_title_tfidf,X_train_ess_tfidf,X_train_teacher_ohe,X_t
rain_cat_ohe,X_train_sub_cat_ohe,
                        X_train_grade_ohe,X_train_state_ohe,X_train_price_stndrd, X_trai
n_quantity_stndrd,X_train_prev_proj_stndrd,
                         X_train_title_stndrd, X_train_essay_stndrd,X_train_pos_stndrd,X
_train_neu_stndrd,X_train_neg_stndrd,X_train_comp_stndrd))

# with the same hstack function we are concatinating a sparse matrix and a dense matirx
X_cv_tfidf_1 = hstack((X_cv_title_tfidf,X_cv_ess_tfidf,X_cv_teacher_ohe,X_cv_cat_ohe,X_
cv_sub_cat_ohe,
                       X_cv_grade_ohe,X_cv_state_ohe,X_cv_price_stndrd, X_cv_quantity_s
tndrd,X_cv_prev_proj_stndrd,
                      X_cv_essay_stndrd, X_cv_title_stndrd,X_cv_pos_stndrd,X_cv_neu_stnd
rd,X_cv_neg_stndrd,X_cv_comp_stndrd))

X_test_tfidf_1 = hstack((X_test_title_tfidf,X_test_ess_tfidf,X_test_teacher_ohe,X_test_
cat_ohe,X_test_sub_cat_ohe,X_test_grade_ohe,
                         X_test_state_ohe,X_test_price_stndrd, X_test_quantity_stndrd,X_
test_prev_proj_stndrd,
                         X_test_essay_stndrd, X_test_title_stndrd,X_test_pos_stndrd,X_te
st_neu_stndrd,X_test_neg_stndrd,X_test_comp_stndrd))
```

In [136]:

```
print(X_train_tfidf_1.shape, y_train.shape)
print(X_test_tfidf_1.shape, y_test.shape)
print(X_cv_tfidf_1.shape, y_cv.shape)
```

```
(22445, 10033) (22445,)
(16500, 10033) (16500,)
(11055, 10033) (11055,)
```

In [137]:

```
X_train_tfidf_1 = X_train_tfidf_1.tocsr()[0:5000]
X_cv_tfidf_1 = X_cv_tfidf_1.tocsr()[0:5000]
X_test_tfidf_1 = X_test_tfidf_1.tocsr()[0:5000]
```

In [138]:

```
X_train_tfidf_1.shape
```

Out[138]:

```
(5000, 10033)
```

TruncatedSVD

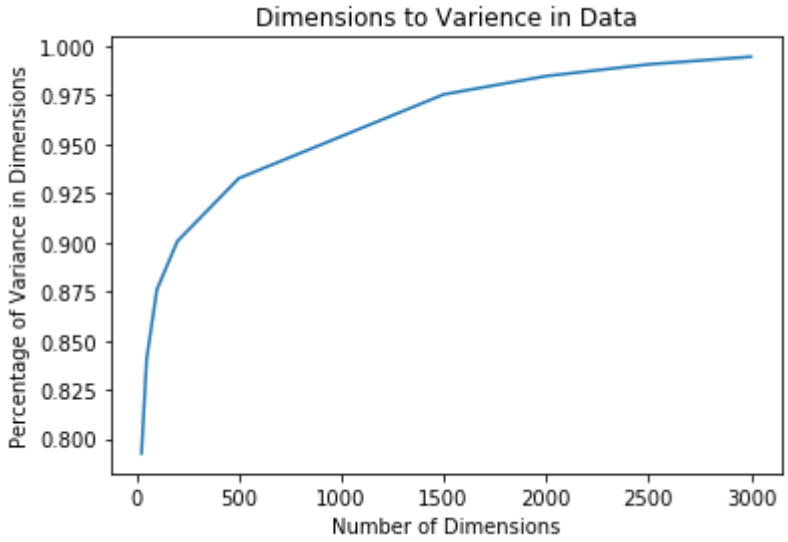
In [139]:

```
from sklearn.decomposition import TruncatedSVD
#https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
#declaring index as Dimensions in train_text_tfidf
D1 = [25,50,100,200,500,1500,2000,2500,3000]
Varience_sum = []
for i in tqdm(D1):
    svd = TruncatedSVD(n_components = i, random_state = 42)
    svd.fit(X_train_tfidf_1)
    Varience_sum.append(svd.explained_variance_ratio_.sum())
```

```
100%|████████████████████████████████████████████████████████████████████████████| 9/9 [04:07<00:00, 27.48s/it]
```

In [140]:

```
plt.xlabel("Number of Dimensions")
plt.ylabel("Percentage of Variance in Dimensions")
plt.title("Dimensions to Varience in Data")
plt.plot(D1,Varience_sum)
plt.show()
```



In [141]:

```
svd = TruncatedSVD(n_components= 2000)
svd.fit(X_train_tfidf_1)
#Transforms:
#Train SVD
X_train_tfidf_1= svd.transform(X_train_tfidf_1 )
#Test SVD
X_test_tfidf_1 = svd.transform(X_test_tfidf_1 )
#CV SVD
X_cv_tfidf_1 = svd.transform(X_cv_tfidf_1 )
```

SVM on new tfidf

In [142]:

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 =
 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [143]:

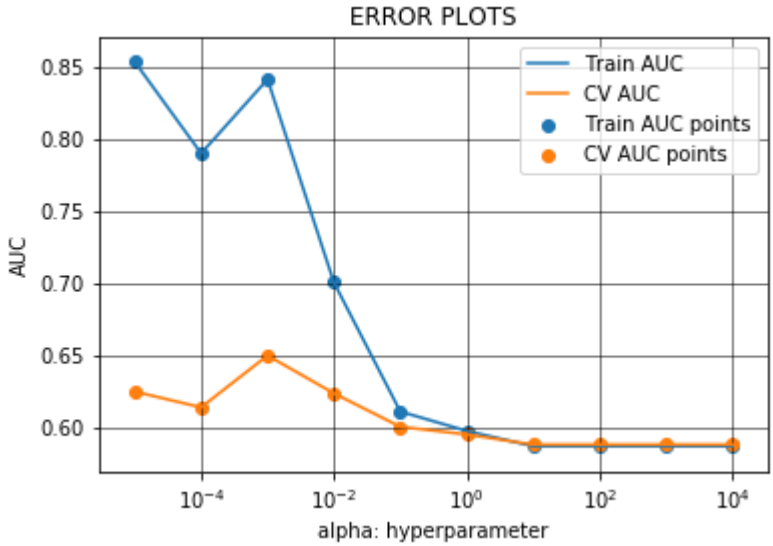
```
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn import svm
from sklearn.metrics import roc_auc_score
import math

train_auc = []
cv_auc = []
alpha =[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

# hyperparameter tuning with l2 reg
for i  in   tqdm(alpha):
    sd = SGDClassifier(loss = 'hinge', penalty = 'l2', class_weight = 'balanced',alpha
= i)# takes the alpha from the i th list value
    svm = CalibratedClassifierCV(sd, cv= 5)
    svm.fit(X_train_tfidf_1, y_train_1)# fit the model

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    y_train_pred=batch_predict(svm,X_train_tfidf_1)
    y_cv_pred=batch_predict(svm,X_cv_tfidf_1)
# roc curve
#Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from predicti
on scores.
    train_auc.append(roc_auc_score(y_train_1,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv_1, y_cv_pred))
plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
plt.xscale('log')# we take the log in the x axis
plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

```
100%|██████████| 10/10 [00:14<00:00,  1.43s/it]
```



In [144]:

```
optimal_alpha= alpha[cv_auc.index(max(cv_auc))]
alpha_values=[math.log(x) for x in alpha]
print('optimal alpha for which auc is maximum : ',optimal_alpha)
```

```
optimal alpha for which auc is maximum :  0.001
```

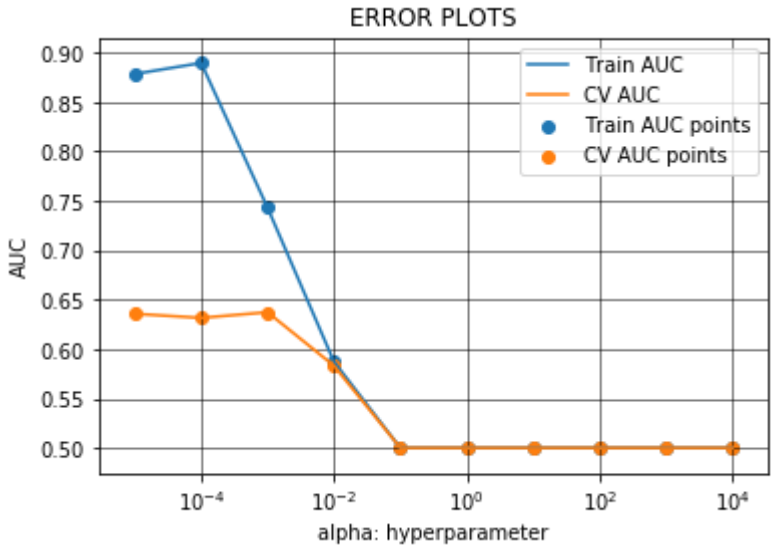
In [145]:

```
train_auc = []
cv_auc = []
alpha =[0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

# hyperparameter tuning with l2 reg
for i  in   tqdm(alpha):
    sd = SGDClassifier(loss = 'hinge', penalty = 'l1', class_weight = 'balanced',alpha
= i)# takes the alpha from the i th list value
    svm = CalibratedClassifierCV(sd, cv= 5)
    svm.fit(X_train_tfidf_1, y_train_1)# fit the model

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    y_train_pred=batch_predict(svm,X_train_tfidf_1)
    y_cv_pred=batch_predict(svm,X_cv_tfidf_1)
# roc curve
#Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from predicti
on scores.
    train_auc.append(roc_auc_score(y_train_1,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv_1, y_cv_pred))
plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
plt.xscale('log')# we take the log in the x axis
plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')
plt.xscale('log')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

```
100%|██████████| 10/10 [00:36<00:00,  3.62s/it]
```



In [146]:

```
sd = SGDClassifier(penalty = 'l1', class_weight = 'balanced',alpha = 0.1)
svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
svm.fit(X_train_tfidf_1, y_train_1)


y_train_pred = batch_predict(svm,X_train_tfidf_1)
y_test_pred = batch_predict(svm,X_test_tfidf_1)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_1, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_1, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

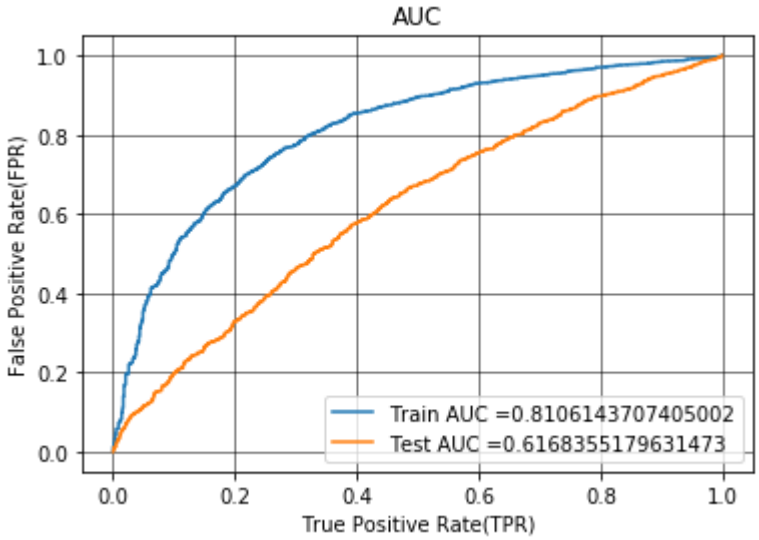
In [147]:

```
sd = SGDClassifier(penalty = 'l2', class_weight = 'balanced',alpha = optimal_alpha)
svm = CalibratedClassifierCV(sd, cv= 5)# takes the alpha from the i th list value
svm.fit(X_train_tfidf_1, y_train_1)


y_train_pred = batch_predict(svm,X_train_tfidf_1)
y_test_pred = batch_predict(svm,X_test_tfidf_1)


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_1, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test_1, y_test_pred)

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



Predicting best threshold

In [148]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.rou
nd(t,3))
    return t

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

Implementing Confusion Matrix

In [149]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train_1, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test_1, predict_with_best_t(y_test_pred, best_t)))
print('='*100)
```

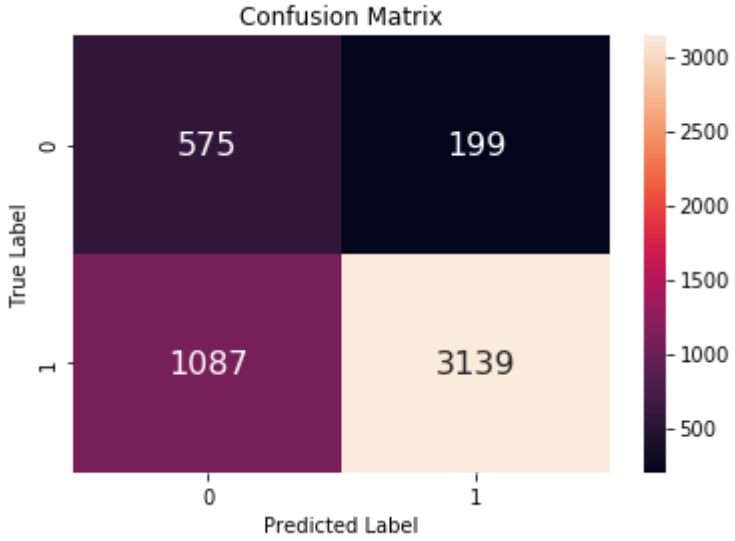
```
the maximum value of tpr*(1-fpr) 0.5518089078193196 for threshold 0.835
Train confusion matrix
[[ 575  199]
 [1087 3139]]
Test confusion matrix
[[ 762   10]
 [4023  205]]
====================================================================================
==========================
```

In [150]:

```
matrix = confusion_matrix(y_train_1, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[150]:

Text(0.5, 1, 'Confusion Matrix')

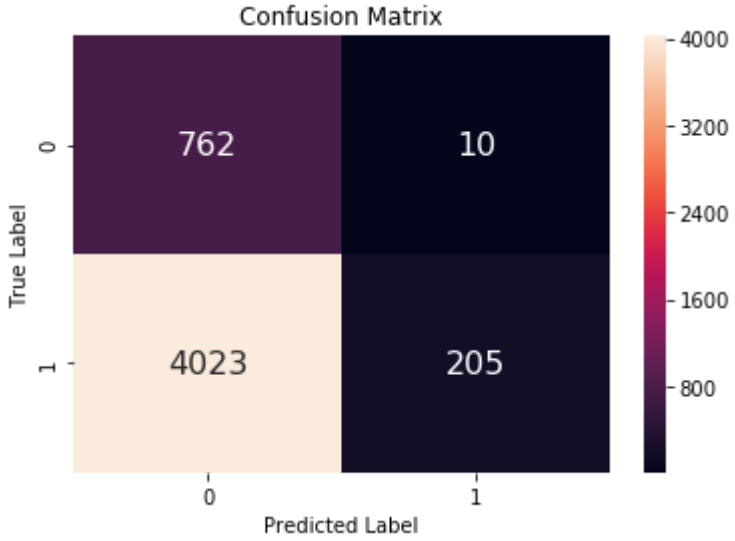


In [151]:

```
matrix = confusion_matrix(y_test_1, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(matrix, annot=True, annot_kws={'size':16}, fmt='g')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

Out[151]:

```
Text(0.5, 1, 'Confusion Matrix')
```



In [154]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

ptable = PrettyTable()
ptable.title = 'Classification Report'

ptable.field_names = ["Vectorization", "Model", "alpha(l2)", "AUC"]

ptable.add_row(["BOW","SGD Classifier",10,69.42])
ptable.add_row(["tf-idf", "SGD Classifier",0.1,67.96])
ptable.add_row(["avg-w2v", "SGDD Classifier",1,66.24])
ptable.add_row(["tf-idf-w2v", "SGD Classifier",0.1,51.99])
ptable.add_row(["tf-idf", "TruncatedSVD", 0.1,61.68])

print(ptable)
```

```
+---------------+-----------------+-----------+-------+
| Vectorization |      Model      | alpha(l2) |  AUC  |
+---------------+-----------------+-----------+-------+
|      BOW      |  SGD Classifier |     10    | 69.42 |
|     tf-idf    |  SGD Classifier |    0.1    | 67.96 |
|    avg-w2v    | SGDD Classifier |     1     | 66.24 |
|   tf-idf-w2v  |  SGD Classifier |    0.1    | 51.99 |
|     tf-idf    |   TruncatedSVD  |    0.1    | 61.68 |
+---------------+-----------------+-----------+-------+
```

Summary:

Initially after loading the dataset if any null values exists replace them with most occuring element then split the data into training, validation, testing data and preprocessed the data to avoid the leakage. Applied bag of words, tfidf, avg_w2v, tfidf_w2v featurising on the data. After concatenating all the features applied SGD Classifier on each.

In all cases l2 is better than l1 reguralization.

Reference:

Applied AI Course

Stackoverflow

geekforgeeks

some other websites in case of any doubts.