Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import nltk
import string
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 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

import warnings
warnings.filterwarnings("ignore")
```

Reading the data

In [2]:

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

In [3]:

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

```
Number of data points in train data (50000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix'
'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
print(resource_data.head(2))
```

```
Number of data points in train data (50000, 4)
['id' 'description' 'quantity' 'price']
        id                                      description  quantity  \
0  p233245  LC652 - Lakeshore Double-Space Mobile Drying Rack         1
1  p069063        Bouncy Bands for Desks (Blue support pipes)         3

    price
0  149.00
1   14.95
```

In [5]:

```
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 category

In [7]:

```
categories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

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

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

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

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

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]))
```

preprocessing grade category

In [9]:

```
grade_categories = list(project_data['project_grade_category'].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

grade_cat_list = []
for i in grade_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('&','_')
    grade_cat_list.append(temp.strip())

project_data['grade_categories'] = grade_cat_list
project_data.drop(['project_grade_category'], 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['grade_categories'].values:
    my_counter.update(word.split())

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

In [10]:

```
y = project_data['project_is_approved'].values
X = project_data.drop(['project_is_approved','Unnamed: 0','teacher_id','project_submitt
ed_datetime'], axis=1)
X.head(1)
```

Out[10]:

	id	teacher_prefix	school_state	project_title	project_essay_1	project_essay
0	p253737	Mrs.	IN	Educational Support for English Learners at Home	My students are English learners that are work...	\"The limits of your language are the limits c

Train- test spliting

In [11]:

```
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)
```

preprocessing essays

In [12]:

```
# 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 [13]:

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

```
Describing my students is not an easy task.  Many would say that they are
inspirational, creative, and hard-working.  They are all unique - unique i
n their interests, their learning, their abilities, and so much more.  Wha
t they all have in common is their desire to learn each day, despite diffi
culties that they encounter.  \r\nOur classroom is amazing - because we un
derstand that everyone learns at their own pace.  As the teacher, I pride
myself in making sure my students are always engaged, motivated, and inspi
red to create their own learning! \r\nThis project is to help my students
choose seating that is more appropriate for them, developmentally.  Many s
tudents tire of sitting in chairs during lessons, and having different sea
ts available helps to keep them engaged and learning.\r\nFlexible seating
is important in our classroom, as many of our students struggle with atten
tion, focus, and engagement.  We currently have stability balls for seatin
g, as well as regular chairs, but these stools will help students who have
trouble with balance, or find it difficult to sit on a stability ball for
a long period of time.  We are excited to try these stools as a part of ou
r engaging classroom community!nannan
==================================================
```

In [14]:

```
# \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)
```

```
Describing my students is not an easy task.  Many would say that they are
inspirational, creative, and hard-working.  They are all unique - unique i
n their interests, their learning, their abilities, and so much more.  Wha
t they all have in common is their desire to learn each day, despite diffi
culties that they encounter.    Our classroom is amazing - because we unde
rstand that everyone learns at their own pace.  As the teacher, I pride my
self in making sure my students are always engaged, motivated, and inspire
d to create their own learning!   This project is to help my students choo
se seating that is more appropriate for them, developmentally.  Many stude
nts tire of sitting in chairs during lessons, and having different seats a
vailable helps to keep them engaged and learning.  Flexible seating is imp
ortant in our classroom, as many of our students struggle with attention,
focus, and engagement.  We currently have stability balls for seating, as
well as regular chairs, but these stools will help students who have troub
le with balance, or find it difficult to sit on a stability ball for a lon
g period of time.  We are excited to try these stools as a part of our eng
aging classroom community!nannan
```

In [15]:

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

```
Describing my students is not an easy task Many would say that they are in
spirational creative and hard working They are all unique unique in their
interests their learning their abilities and so much more What they all ha
ve in common is their desire to learn each day despite difficulties that t
hey encounter Our classroom is amazing because we understand that everyone
learns at their own pace As the teacher I pride myself in making sure my s
tudents are always engaged motivated and inspired to create their own lear
ning This project is to help my students choose seating that is more appro
priate for them developmentally Many students tire of sitting in chairs du
ring lessons and having different seats available helps to keep them engag
ed and learning Flexible seating is important in our classroom as many of
our students struggle with attention focus and engagement We currently hav
e stability balls for seating as well as regular chairs but these stools w
ill help students who have trouble with balance or find it difficult to si
t on a stability ball for a long period of time We are excited to try thes
e stools as a part of our engaging classroom community nannan
```

In [16]:

```
# 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'r
e", "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', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', '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", 'no
w', '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', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [17]:

```
# 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 [00:24<00:00, 929.65it/s]
```

In [18]:

```
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 [00:21<00:00, 517.00it/s]
```

In [19]:

```
# 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 [00:23<00:00, 699.58it/s]
```

preprocessing titles

In [20]:

```
# 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 [21]:

```
sent = decontracted(project_data['project_title'].values[3700])
print(sent)
print('='*50)
```

```
Chromebooks for the Classroom: Technology Integration
==================================================
```

In [22]:

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

```
Chromebooks for the Classroom Technology Integration
```

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'r
e", "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', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', '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', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', '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", 'no
w', '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', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "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, 18862.02it/s]
```

In [25]:

```
# Combining all the above stundents
from tqdm import tqdm
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, 18938.62it/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, 20354.74it/s]
```

One hot encoding

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,)
```

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,)
```

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]:

```
# 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 [34]:

```
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.astype('U'))
# fit has to happen only on train data
X_cv_teacher_ohe = vec4.transform(X_cv['teacher_prefix'].values.astype('U'))
X_test_teacher_ohe = vec4.transform(X_test['teacher_prefix'].values.astype('U'))
```

In [35]:

```
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,)
```

In [36]:

```
# 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['grade_categories'].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 [37]:

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

In [38]:

```
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, 4) (22445,)
Shape of X_cv after one hot encodig  (11055, 4) (11055,)
Shape of X_test after one hot encodig  (16500, 4) (16500,)
```

In [39]:

```
vec6 = CountVectorizer(min_df=10,ngram_range = (1,2), max_features=5000)
 # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vec6.fit_transform(preprocessed_train_essay)
X_cv_essay_bow = vec6.transform(preprocessed_cv_essay)
X_test_essay_bow = vec6.transform(preprocessed_test_essay)

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

```
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
```

In [40]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vec7 = CountVectorizer(min_df=10,ngram_range = (1,2), max_features=5000)

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

print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
```

```
After vectorizations
(22445, 1608) (22445,)
(11055, 1608) (11055,)
(16500, 1608) (16500,)
```

In [41]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vec8 = TfidfVectorizer(min_df=10, ngram_range = (1,2), max_features =5000)

# fit has to happen only on train data
X_train_ess_tfidf = vec8.fit_transform(preprocessed_train_essay)

# we use the fitted Tfidf Vectorizer to convert the text to vector
X_cv_ess_tfidf = vec8.transform(preprocessed_cv_essay)
X_test_ess_tfidf = vec8.transform(preprocessed_test_essay)

print("After vectorizations")
print(X_train_ess_tfidf.shape, y_train.shape)
print(X_cv_ess_tfidf.shape, y_cv.shape)
print(X_test_ess_tfidf.shape, y_test.shape)
```

```
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
```

In [42]:

```
vec9 = TfidfVectorizer(min_df=10, ngram_range = (1,2), max_features = 5000)
X_train_title_tfidf = vec9.fit_transform(preprocessed_train_title)

# we use the fitted Tfidf Vectorizer to convert the text to vector
X_cv_title_tfidf = vec9.transform(preprocessed_cv_title)
X_test_title_tfidf = vec9.transform(preprocessed_test_title)

print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
```

```
After vectorizations
(22445, 1608) (22445,)
(11055, 1608) (11055,)
(16500, 1608) (16500,)
```

In [43]:

```
# 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()
```

In [44]:

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

In [45]:

```
# 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])
```

Standardising price, quantity, previous projects

In [46]:

```
#for train
# 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()

price_stndrd = price_scalar.fit_transform(project_data['price'].values.reshape(-1,1)) #
finding the mean and standard deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_
[0])}")
```

```
Mean : 299.62610439999986, Standard deviation : 62.66017637441105
```

In [47]:

```
#merging
# we also have to do this in train,test and cv
# so also merge the resource data with the trian,cv and test

X_train = pd.merge(X_train, price_data, on = "id", how = "left")
#print(x_train.columns)
X_test = pd.merge(X_test, price_data, on = "id", how = "left")
X_cv = pd.merge(X_cv, price_data, on = "id", how = "left")
```

In [48]:

```
# 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_cv['price'].mode().iloc[0])
X_test['price'] = X_test['price'].fillna(X_test['price'].mode().iloc[0])
```

In [49]:

```
# 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 [50]:

```
X_train_price_stndrd
```

Out[50]:

```
array([[-0.05307197],
       [-0.05307197],
       [-0.05307197],
       ...,
       [-0.05307197],
       [-0.05307197],
       [-0.05307197]])
```

In [51]:

```
X_cv_price_stndrd
```

Out[51]:

```
array([[2.07598928],
       [2.07598928],
       [2.07598928],
       ...,
       [2.07598928],
       [2.07598928],
       [2.07598928]])
```

In [52]:

```
X_test_price_stndrd
```

Out[52]:

```
array([[2.07598928],
       [2.07598928],
       [2.07598928],
       ...,
       [2.07598928],
       [2.07598928],
       [2.07598928]])
```

In [53]:

```
# 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 [54]:

```
# 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 [55]:

```
X_train_quantity_stndrd
```

Out[55]:

```
array([[-0.09204458],
       [-0.09204458],
       [-0.09204458],
       ...,
       [-0.09204458],
       [-0.09204458],
       [-0.09204458]])
```

In [56]:

```
X_cv_quantity_stndrd
```

Out[56]:

```
array([[-0.09204458],
       [-0.09204458],
       [-0.09204458],
       ...,
       [-0.09204458],
       [-0.09204458],
       [-0.09204458]])
```

In [57]:

```
X_test_quantity_stndrd
```

Out[57]:

```
array([[-0.09204458],
       [-0.09204458],
       [-0.09204458],
       ...,
       [-0.09204458],
       [-0.09204458],
       [-0.09204458]])
```

In [58]:

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

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

In [59]:

```
# prev_proj
prev_proj_scalar = StandardScaler()
X_train_prev_proj_stndrd = prev_proj_scalar.fit_transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_cv_prev_proj_stndrd = prev_proj_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_test_prev_proj_stndrd = prev_proj_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
```

In [60]:

```
X_train_prev_proj_stndrd
```

Out[60]:

```
array([[ 0.16582248],
       [-0.36662087],
       [ 5.56124837],
       ...,
       [-0.36662087],
       [ 0.02383758],
       [-0.33112465]])
```

In [61]:

```
X_cv_prev_proj_stndrd
```

Out[61]:

```
array([[ 0.94673938],
       [-0.33112465],
       [-0.33112465],
       ...,
       [-0.29562842],
       [-0.40211709],
       [-0.04715486]])
```

In [62]:

```
X_test_prev_proj_stndrd
```

Out[62]:

```
array([[-0.08265109],
       [-0.36662087],
       [-0.40211709],
       ...,
       [-0.40211709],
       [-0.36662087],
       [ 0.2013187 ]])
```

In [63]:

```
X_train.head()
```

Out[63]:

	id	teacher_prefix	school_state	project_title	project_essay_1	project_es
0	p029657	Mrs.	WA	Help Medically Fragile Students Communicate an...	My students arrive with huge smiles, and an ea...	I have two students w classified a medi...
1	p029818	Mrs.	AZ	Differentiated Learning with Technology!	Our west-Phoenix students come from bilingual ...	Having the of Chromel will allow fc
2	p256848	Ms.	NC	Water, Water Everywhere! Watercolors Are Great !	High School Visual Arts are not just about cr...	I like using watercolor in all levels
3	p236308	Mrs.	NE	Savvy Seats in Sixth	My students are in 6th grade at an suburban el...	The seat s will keep th sixth grade org...
4	p190054	Mrs.	ID	My Rockin Mathematicians!	\"The function of education is to teach one to...	Ready beg how we sta each morn They...

concatenating bow and tfidf

In [64]:

```
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))


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

```
(22445, 6710) (22445,)
```

In [65]:

```
from scipy.sparse import hstack
# 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))


print(X_cv_bow.shape, y_cv.shape)
```

```
(11055, 6710) (11055,)
```

In [66]:

```
from scipy.sparse import hstack
# 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))


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

```
(16500, 6710) (16500,)
```

In [95]:

```
X_train_bow = X_train_bow.tocsr()
X_cv_bow = X_cv_bow.tocsr()
X_test_bow = X_test_bow.tocsr()
```

In [67]:

```
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_train
_quantity_stndrd,X_train_prev_proj_stndrd))


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

```
(22445, 6710) (22445,)
```

In [68]:

```
from scipy.sparse import hstack
# 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))


print(X_cv_tfidf.shape, y_cv.shape)
```

```
(11055, 6710) (11055,)
```

In [69]:

```
from scipy.sparse import hstack
# 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))


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

```
(16500, 6710) (16500,)
```

In [96]:

```
X_train_tfidf = X_train_tfidf.tocsr()
X_cv_tfidf = X_cv_tfidf.tocsr()
X_test_tfidf = X_test_tfidf.tocsr()
```

Using Pretrained Model: Avgw2v

In [70]:

```
# 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 [71]:

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

```
725it [00:00, 3514.64it/s]

Loading Glove Model

1917494it [09:26, 3384.93it/s]

Done. 1917494  words loaded!
```

In [72]:

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

In [73]:

```
# average Word2Vec
# 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
```

In [74]:

```
# FOR ESSAYS
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 [00:09<00:00, 2347.89it/s]
  4%|███
| 457/11055 [00:00<00:04, 2276.89it/s]

22445
300

100%|██████████████████████████████████████████████████████████████████████
█████| 11055/11055 [00:04<00:00, 2309.86it/s]
  3%|██
| 420/16500 [00:00<00:07, 2023.95it/s]

11055
300

100%|██████████████████████████████████████████████████████████████████████
█████| 16500/16500 [00:06<00:00, 2369.32it/s]

16500
300
```

In [75]:

```
# 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, 42100.26it/s]
 40%|█████████████████████████████▏
| 4368/11055 [00:00<00:00, 39935.04it/s]

22445
300

100%|█████████████████████████████████████████████████████████████████████
████| 11055/11055 [00:00<00:00, 44215.74it/s]
 24%|█████████████████▌
| 3927/16500 [00:00<00:00, 35907.13it/s]

11055
300

100%|█████████████████████████████████████████████████████████████████████
████| 16500/16500 [00:00<00:00, 37589.29it/s]

16500
300
```

Using Pretrained Model: tfidf_w2v

In [76]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
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 [77]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_tfidf_w2v_essay = []; # the avg-w2v for each sentence/review is stored in this
 list
for sentence in tqdm(preprocessed_train_essay): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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
    X_train_tfidf_w2v_essay.append(vector)

print(len(X_train_tfidf_w2v_essay))
print(len(X_train_tfidf_w2v_essay[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████
██████| 22445/22445 [01:25<00:00, 263.10it/s]

22445
300
```

In [78]:

```
# average Word2Vec
# compute average word2vec for each review.
X_cv_tfidf_w2v_essay = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(preprocessed_cv_essay): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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
    X_cv_tfidf_w2v_essay.append(vector)

print(len(X_cv_tfidf_w2v_essay))
print(len(X_cv_tfidf_w2v_essay[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 11055/11055 [00:41<00:00, 268.25it/s]

11055
300
```

In [79]:

```
# average Word2Vec
# compute average word2vec for each review.
X_test_tfidf_w2v_essay = []; # the avg-w2v for each sentence/review is stored in this l
ist
for sentence in tqdm(preprocessed_test_essay): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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
    X_test_tfidf_w2v_essay.append(vector)

print(len(X_test_tfidf_w2v_essay))
print(len(X_test_tfidf_w2v_essay[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████
██████| 16500/16500 [01:09<00:00, 237.51it/s]

16500
300
```

In [80]:

```
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 [81]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_tfidf_w2v_title = []; # the avg-w2v for each sentence/review is stored in this
 list
for sentence in tqdm(preprocessed_train_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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
    X_train_tfidf_w2v_title.append(vector)

print(len(X_train_tfidf_w2v_title))
print(len(X_train_tfidf_w2v_title[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████
████| 22445/22445 [00:02<00:00, 10669.48it/s]

22445
300
```

In [82]:

```
# average Word2Vec
# compute average word2vec for each review.
X_cv_tfidf_w2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_cv_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.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
    X_cv_tfidf_w2v_title.append(vector)

print(len(X_cv_tfidf_w2v_title))
print(len(X_cv_tfidf_w2v_title[0]))
```

```
100%|█████████████████████████████████████████████████████████████████████
████| 11055/11055 [00:00<00:00, 11194.81it/s]

11055
300
```

In [83]:

```
# average Word2Vec
# compute average word2vec for each review.
X_test_tfidf_w2v_title = []; # the avg-w2v for each sentence/review is stored in this l
ist
for sentence in tqdm(preprocessed_test_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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
    X_test_tfidf_w2v_title.append(vector)

print(len(X_test_tfidf_w2v_title))
print(len(X_test_tfidf_w2v_title[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████| 16500/16500 [00:01<00:00, 10875.04it/s]

16500
300
```

concatenating avg_w2v, tfidf_w2v

In [84]:

```
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_train
_quantity_stndrd,X_train_prev_proj_stndrd)).tocsr()


print(X_train_avg_w2v.shape, y_train.shape)
```

```
(22445, 702) (22445,)
```

In [85]:

```
from scipy.sparse import hstack
# 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_st
ndrd,X_cv_prev_proj_stndrd)).tocsr()


print(X_cv_avg_w2v.shape, y_cv.shape)
```

```
(11055, 702) (11055,)
```

In [86]:

```
from scipy.sparse import hstack
# 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_qua
ntity_stndrd,X_test_prev_proj_stndrd)).tocsr()


print(X_test_avg_w2v.shape, y_test.shape)
```

```
(16500, 702) (16500,)
```

In [87]:

```
# List to np.array
X_train_avg_w2v_ess=np.array(X_train_avg_w2v_ess)
X_cv_avg_w2v_ess=np.array(X_cv_avg_w2v_ess)
X_test_avg_w2v_ess=np.array(X_test_avg_w2v_ess)

X_train_avg_w2v_title=np.array(X_train_avg_w2v_title)
X_cv_avg_w2v_title=np.array(X_cv_avg_w2v_title)
X_test_avg_w2v_title=np.array(X_test_avg_w2v_title)
```

In [88]:

```
print("After vectorizations")
print(X_train_avg_w2v_ess.shape, y_train.shape)
print(X_cv_avg_w2v_ess.shape, y_cv.shape)
print(X_test_avg_w2v_ess.shape, y_test.shape)
print('='*50)
```

```
After vectorizations
(22445, 300) (22445,)
(11055, 300) (11055,)
(16500, 300) (16500,)
==================================================
```

In [89]:

```
print("After vectorizations")
print(X_train_avg_w2v_title.shape, y_train.shape)
print(X_cv_avg_w2v_title.shape, y_cv.shape)
print(X_test_avg_w2v_title.shape, y_test.shape)
print('='*50)
```

```
After vectorizations
(22445, 300) (22445,)
(11055, 300) (11055,)
(16500, 300) (16500,)
==================================================
```

In [90]:

```
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,X_train_tfidf_w2v_essay,X_train_tea
cher_ohe,X_train_cat_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)).tocsr()


print(X_train_tfidf_w2v.shape, y_train.shape)
```

```
(22445, 702) (22445,)
```

In [91]:

```
from scipy.sparse import hstack
# 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,X_cv_tfidf_w2v_essay,X_cv_teacher_ohe,X_c
v_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)).tocsr()


print(X_cv_tfidf_w2v.shape, y_cv.shape)
```

```
(11055, 702) (11055,)
```

In [92]:

```
from scipy.sparse import hstack
# 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,X_test_tfidf_w2v_essay,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_qua
ntity_stndrd,X_test_prev_proj_stndrd)).tocsr()


print(X_test_tfidf_w2v.shape, y_test.shape)
```

```
(16500, 702) (16500,)
```

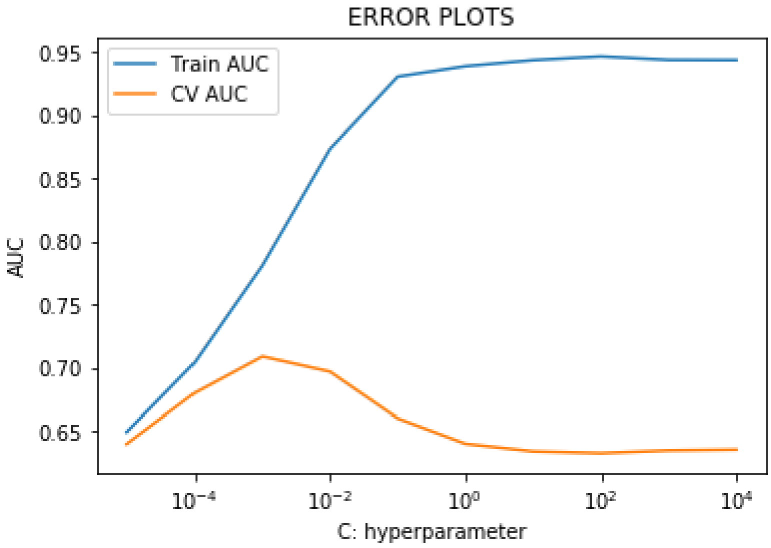
In [93]:

```
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
```

Logistic Regression on Bag of words

In [97]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
C = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
for i in C:
    clf = LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    clf.fit(X_train_bow, y_train)


    y_train_pred=batch_predict(clf,X_train_bow)
    y_cv_pred=batch_predict(clf,X_cv_bow)

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')
plt.xscale('log')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



best C

In [98]:

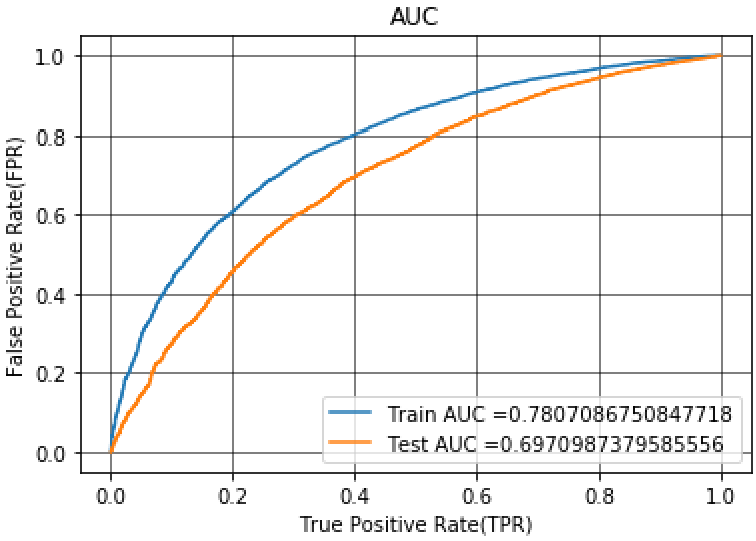
```
optimal_C= C[cv_auc.index(max(cv_auc))]
C_values=[math.log(x) for x in C]
print('optimal alpha for which auc is maximum : ',optimal_C)
```

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

Hyperparameter tuning

In [99]:

```
from sklearn.metrics import roc_auc_score

import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

model = LogisticRegression(C = optimal_C ,class_weight='balanced')

model.fit(X_train_bow, y_train)

# 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(model, X_train_bow)
y_test_pred = batch_predict(model, 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()
```



best threshold

In [100]:

```
# 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
```

Confusion Matrix

In [101]:

```
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)
```

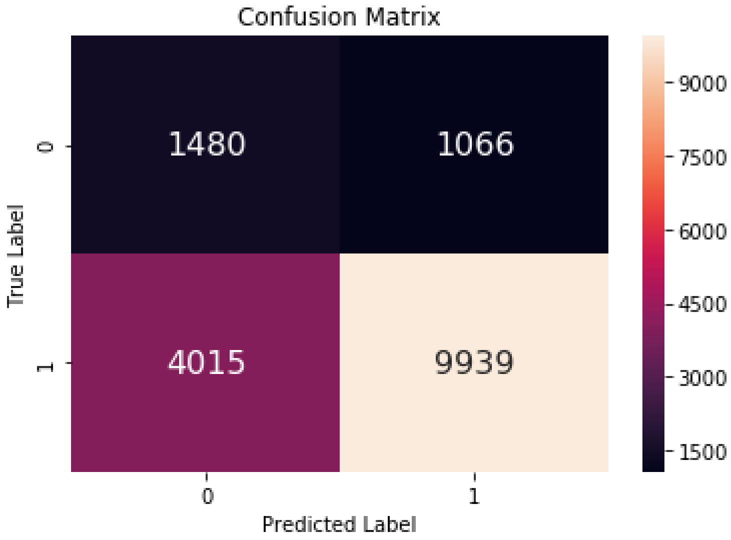
```
the maximum value of tpr*(1-fpr) 0.5090654602245945 for threshold 0.486
Train confusion matrix
[[ 2444  1019]
 [ 5290 13692]]
Test confusion matrix
[[1480 1066]
 [4015 9939]]
==========================================================================
==========================
```

In [102]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-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[102]:

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

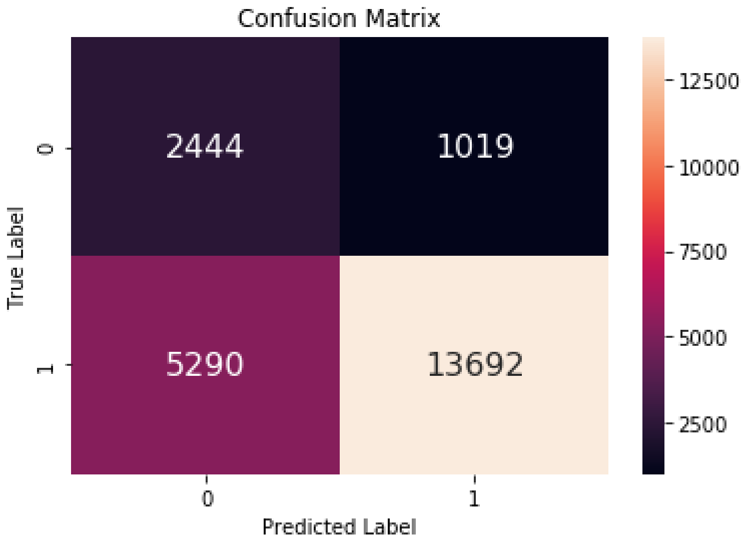


In [103]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-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[103]:

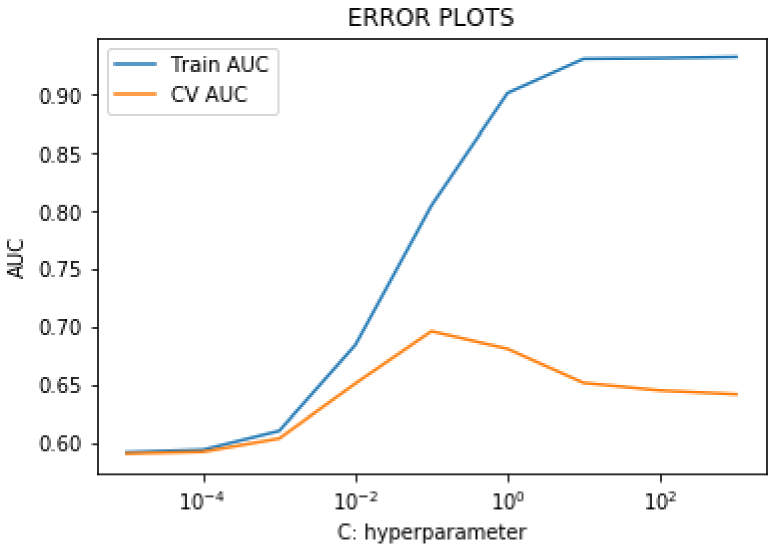
```
Text(0.5, 1, 'Confusion Matrix')
```



Logistic Regression on tfidf

In [104]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
C = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]
for i in C:
    clf = LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    clf.fit(X_train_tfidf, y_train)


    y_train_pred=batch_predict(clf,X_train_tfidf)
    y_cv_pred=batch_predict(clf,X_cv_tfidf)

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')
plt.xscale('log')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



best C

In [105]:

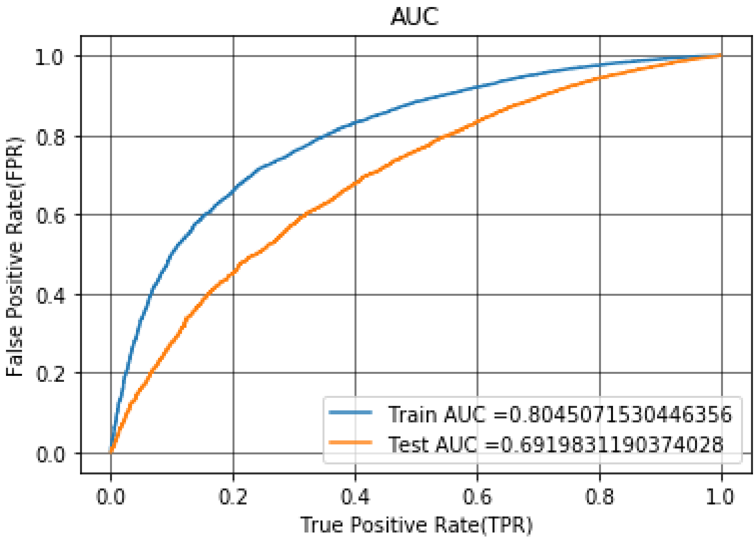
```
optimal_C= C[cv_auc.index(max(cv_auc))]
C_values=[math.log(x) for x in C]
print('optimal alpha for which auc is maximum : ',optimal_C)
```

```
optimal alpha for which auc is maximum :  0.1
```

Hyperparameter tuning

In [106]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

model = LogisticRegression(C = optimal_C,class_weight='balanced')

model.fit(X_train_tfidf, y_train)

# 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(model, X_train_tfidf)
y_test_pred = batch_predict(model, 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()
```



best threshold

In [107]:

```
# 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
```

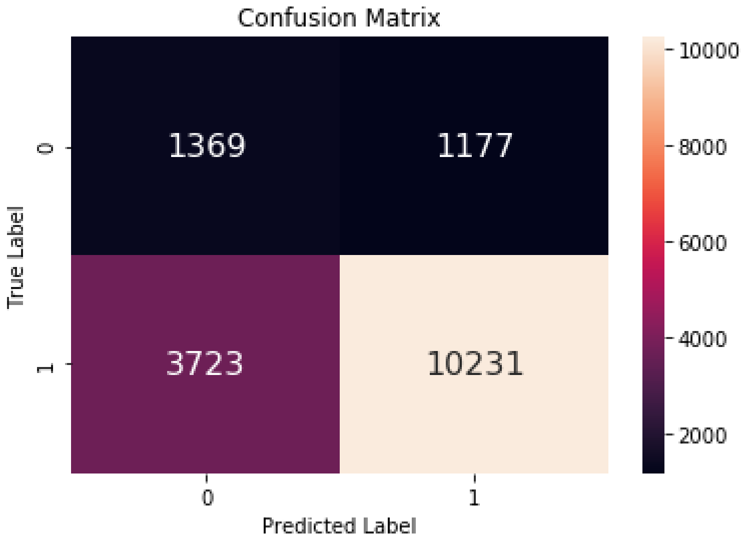
Confusion Matrix

In [108]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-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[108]:

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

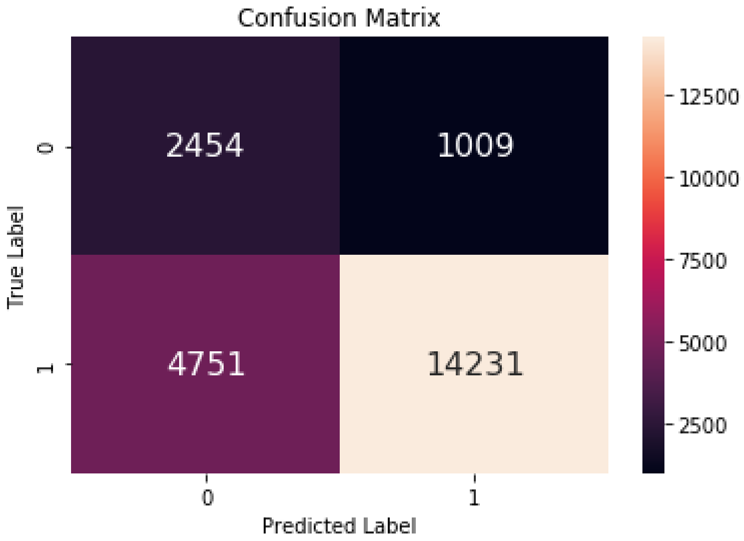


In [109]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-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[109]:

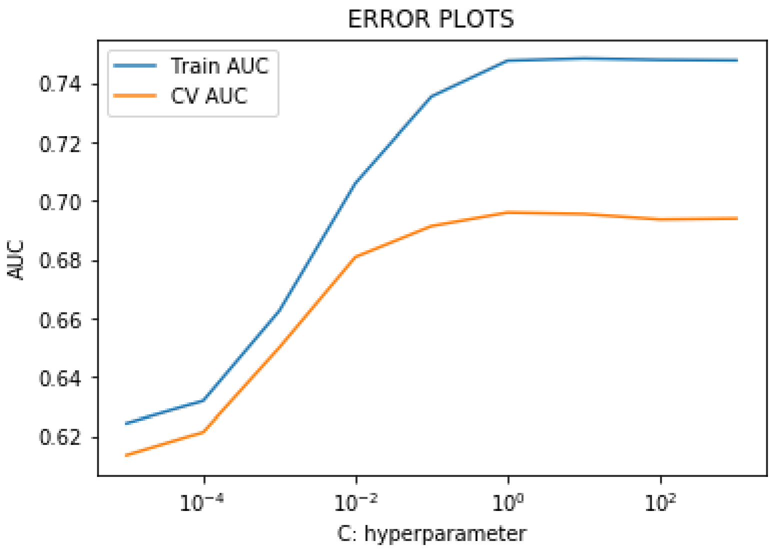
```
Text(0.5, 1, 'Confusion Matrix')
```



Logistic Regression on avg_w2v

In [110]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
C = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]
for i in C:
    clf = LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    clf.fit(X_train_avg_w2v, y_train)


    y_train_pred=batch_predict(clf,X_train_avg_w2v)
    y_cv_pred=batch_predict(clf,X_cv_avg_w2v)

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')
plt.xscale('log')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



best C

In [111]:

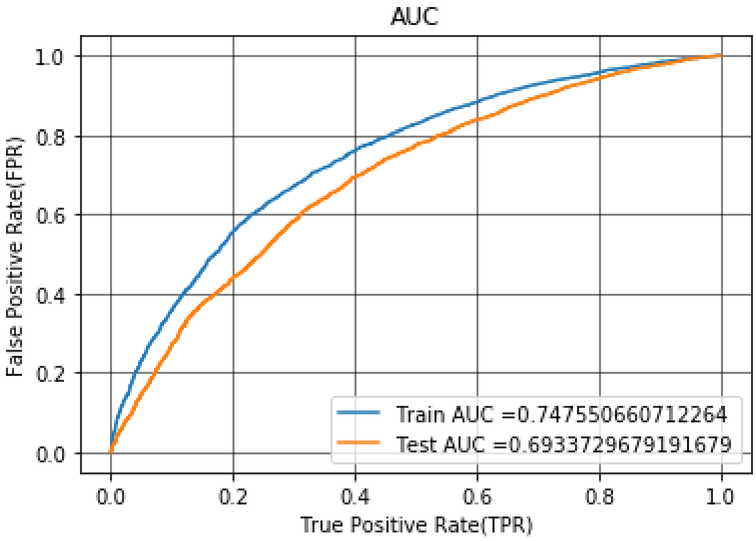
```
optimal_C= C[cv_auc.index(max(cv_auc))]
C_values=[math.log(x) for x in C]
print('optimal alpha for which auc is maximum : ',optimal_C)
```

```
optimal alpha for which auc is maximum :  1
```

Hyperparameter tuning

In [112]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

model = LogisticRegression(C = optimal_C, class_weight='balanced')

model.fit(X_train_avg_w2v, y_train)

# 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(model, X_train_avg_w2v)
y_test_pred = batch_predict(model, 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()
```



best threshold

In [113]:

```
# 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
```

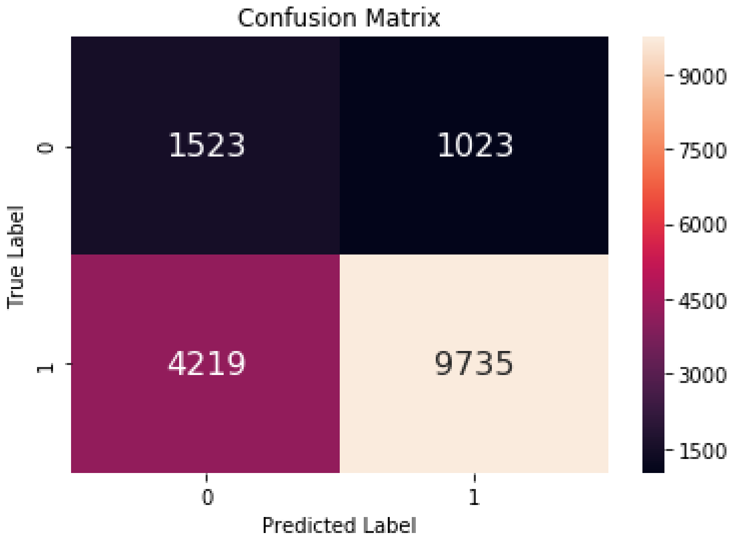
Confusion Matrix

In [114]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-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')
```

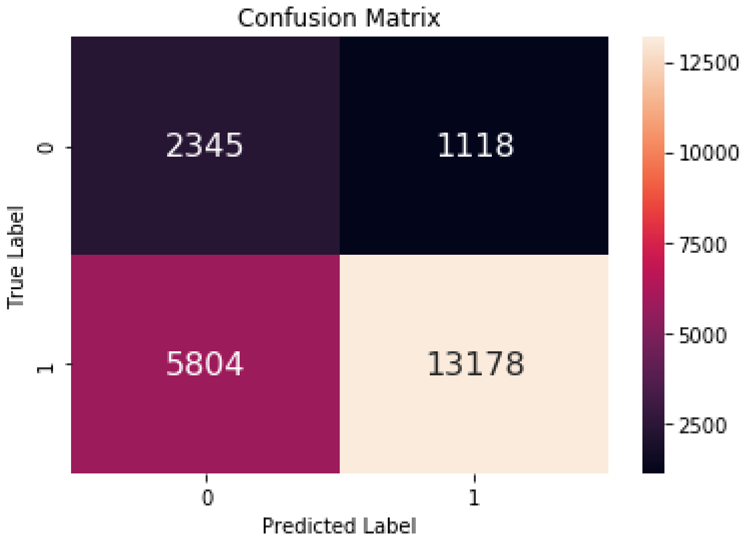


In [115]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-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[115]:

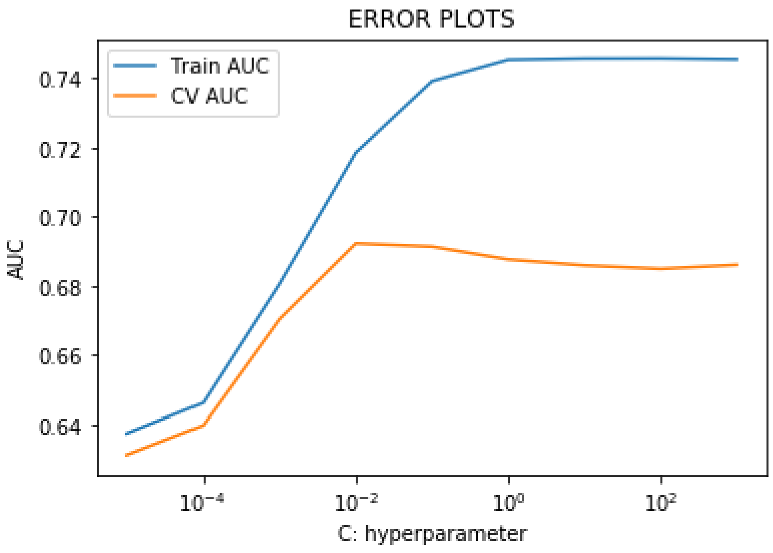
```
Text(0.5, 1, 'Confusion Matrix')
```



Logistic Regression on tfidf_w2v

In [116]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
C = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]
for i in C:
    clf = LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    clf.fit(X_train_tfidf_w2v, y_train)


    y_train_pred=batch_predict(clf,X_train_tfidf_w2v)
    y_cv_pred=batch_predict(clf,X_cv_tfidf_w2v)

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')
plt.xscale('log')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



best C

In [117]:

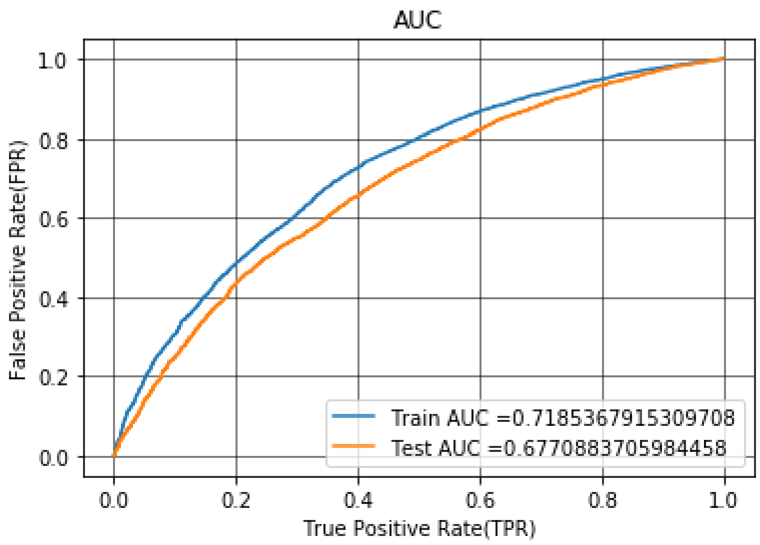
```
optimal_C= C[cv_auc.index(max(cv_auc))]
C_values=[math.log(x) for x in C]
print('optimal alpha for which auc is maximum : ',optimal_C)
```

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

Hyperparameter tuning

In [118]:

```
import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

model = LogisticRegression(C = optimal_C, class_weight='balanced')

model.fit(X_train_tfidf_w2v, y_train)

# 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(model, X_train_tfidf_w2v)
y_test_pred = batch_predict(model, X_test_tfidf_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()
```



best threshold

In [119]:

```
# 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
```

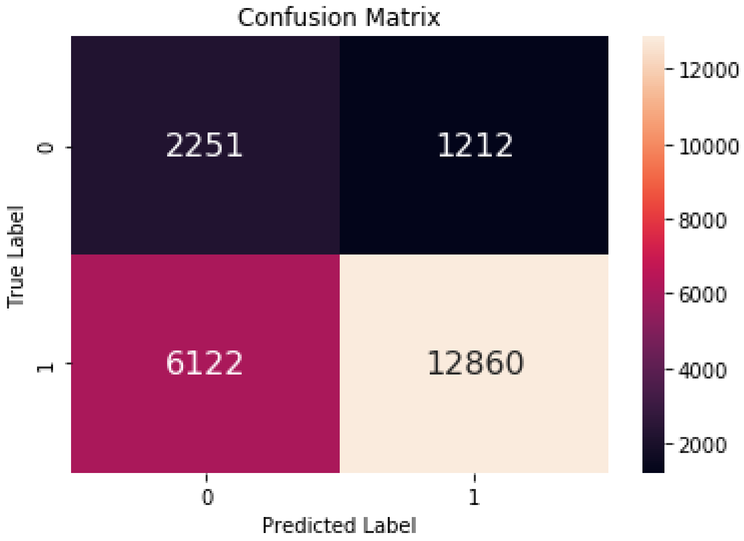
Confusion Matrix

In [120]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-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[120]:

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

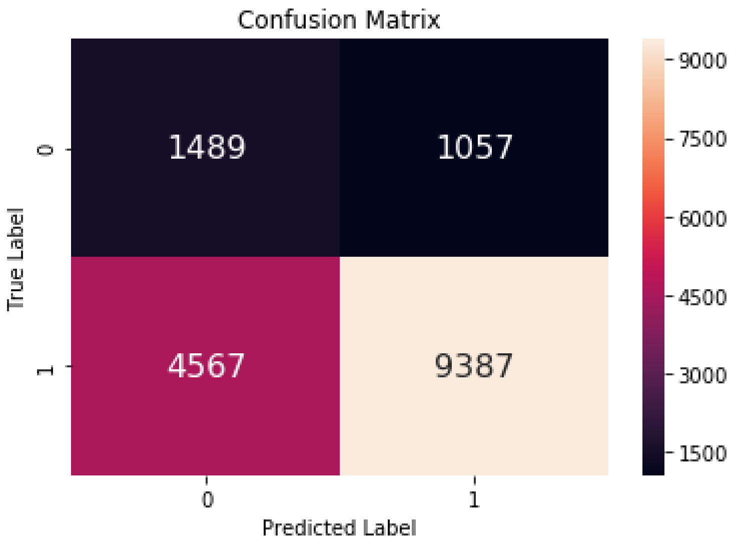


In [121]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-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[121]:

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



Number of words in essay

In [122]:

```
# For train data
essay_length_train=[]
for i in range(0,len(X_train)):
    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,len(X_test)):
    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,len(X_cv)):
    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 [123]:

```
# 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
))
```

In [124]:

```
X_train_essay_stndrd
```

Out[124]:

```
array([[-1.02185678],
       [-0.18667222],
       [-0.63519726],
       ...,
       [-0.66613002],
       [ 0.77224338],
       [-0.37226879]])
```

In [125]:

```
X_cv_essay_stndrd
```

Out[125]:

```
array([[ 3.47886   ],
       [ 1.49916327],
       [ 0.52478129],
       ...,
       [-0.6815964 ],
       [ 1.17436928],
       [-0.77439469]])
```

In [126]:

```
X_test_essay_stndrd
```

Out[126]:

```
array([[-1.0063904 ],
       [-0.99092402],
       [ 1.83942365],
       ...,
       [-0.89812573],
       [-1.02185678],
       [ 2.27248231]])
```

Number of words in titles

In [127]:

```
# For train data
title_length_train=[]
for i in range(0,len(X_train)):
    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,len(X_test)):
    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,len(X_cv)):
    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
```

In [128]:

```
# 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
))
```

In [129]:

```
X_train_title_stndrd
```

Out[129]:

```
array([[ 2.29963256],
       [-0.5464306 ],
       [ 0.87660098],
       ...,
       [-0.5464306 ],
       [ 1.8252887 ],
       [-1.02077446]])
```

In [130]:

```
X_cv_title_stndrd
```

Out[130]:

```
array([[-0.07208674],
       [-0.07208674],
       [-0.5464306 ],
       ...,
       [ 0.87660098],
       [-0.07208674],
       [ 1.8252887 ]])
```

In [131]:

```
X_test_title_stndrd
```

Out[131]:

```
array([[ 0.40225712],
       [-0.5464306 ],
       [ 0.87660098],
       ...,
       [ 1.8252887 ],
       [-1.02077446],
       [-0.07208674]])
```

Sentimental scores

In [132]:

```
# http://t-redactyl.io/blog/2017/04/applying-sentiment-analysis-with-vader-and-the-twit
ter-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
```

```
100%|█████████████████████████████████████████████████████████████████████
███████| 22445/22445 [07:07<00:00, 52.56it/s]
```

In [133]:

```
# http://t-redactyl.io/blog/2017/04/applying-sentiment-analysis-with-vader-and-the-twit
ter-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 [05:31<00:00, 49.83it/s]
```

In [134]:

```
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 [03:38<00:00, 50.56it/s]
```

Standardizing positive, negative, neutral, compound

In [135]:

```
# 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 [136]:

```
# 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 [137]:

```
# 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 [138]:

```
# 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))
```

Concatenating categorical values, numerical values, sentimental scores

In [139]:

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

X_train = hstack((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_train_quantity_stndrd,X_tra
in_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 = hstack((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_stndrd,X_cv_prev_proj_stndrd,X_cv_essay
_stndrd,
                X_cv_title_stndrd,X_cv_pos_stndrd,X_cv_neu_stndrd,X_cv_neg_stndrd,X_cv_
comp_stndrd))

X_test = hstack((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_test_neu_stndrd,X_test_neg_st
ndrd,X_test_comp_stndrd))
```

In [140]:

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

```
(22445, 108) (22445,)
(16500, 108) (16500,)
(11055, 108) (11055,)
```

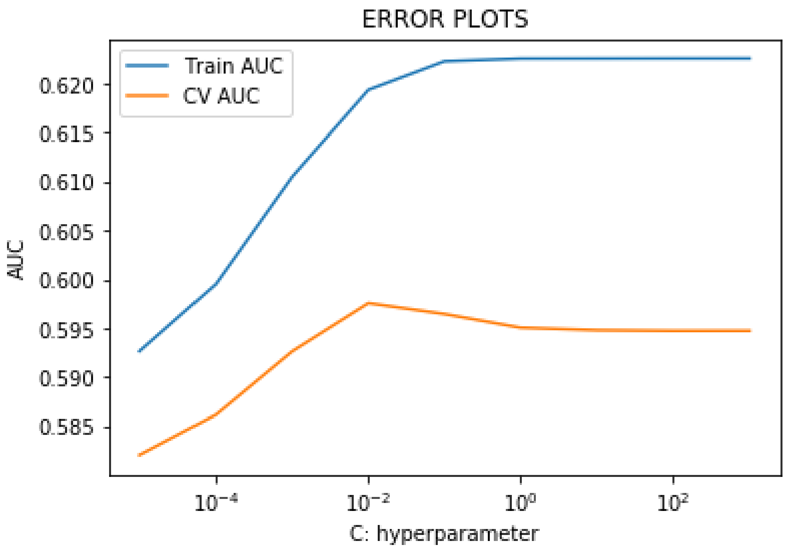
In [141]:

```
X_train = X_train.tocsr()
X_cv = X_cv.tocsr()
X_test =X_test.tocsr()
```

Logistic regression without text features

In [142]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
C = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000]
for i in C:
    clf = LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    clf.fit(X_train, y_train)


    y_train_pred=batch_predict(clf,X_train)
    y_cv_pred=batch_predict(clf,X_cv)

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(C, train_auc, label='Train AUC')
plt.plot(C, cv_auc, label='CV AUC')
plt.xscale('log')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



best C

In [143]:

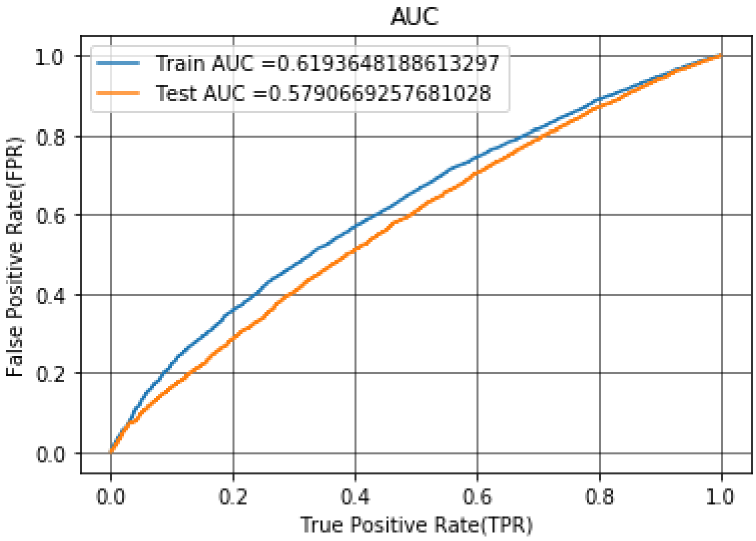
```
optimal_C= C[cv_auc.index(max(cv_auc))]
C_values=[math.log(x) for x in C]
print('optimal alpha for which auc is maximum : ',optimal_C)
```

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

Hyperparameter tuning

In [144]:

```
import matplotlib.pyplot as plt

"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence va
lues, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

model = LogisticRegression(C = optimal_C, class_weight='balanced')

model.fit(X_train, y_train)

# 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(model, X_train)
y_test_pred = batch_predict(model, X_test)


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

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



best threshold

In [145]:

```
# 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.round(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
```

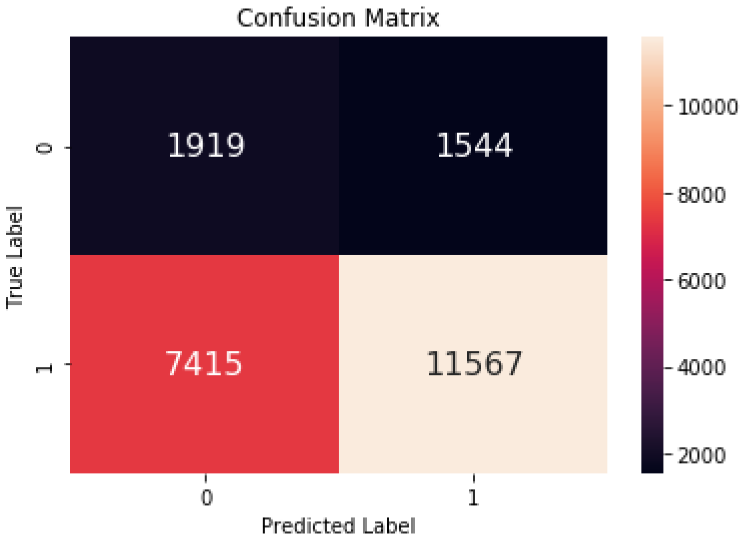
Confusion Matrix

In [146]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-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[146]:

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

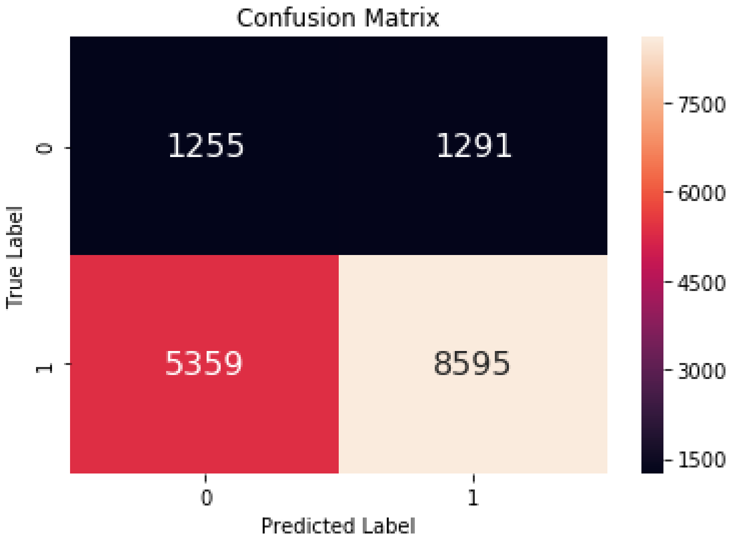


In [147]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-m
ix-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[147]:

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



In [149]:

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

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

ptable.field_names = ["Vectorization", "Model", "C", "AUC"]

ptable.add_row(["BOW","Logistic Regression",0.001,69.70])
ptable.add_row(["tf-idf", "Logistic Regression",0.1,69.19])
ptable.add_row(["avg-w2v", "Logistic Regression",1,69.33])
ptable.add_row(["tf-idf-w2v", "Logistic Regression",0.01,67.77])
ptable.add_row(["Without text", "Logistic Regression", 0.01,57.90])

print(ptable)
```

```
+---------------+---------------------+-------+-------+
| Vectorization |        Model        |   C   |  AUC  |
+---------------+---------------------+-------+-------+
|      BOW      | Logistic Regression | 0.001 |  69.7 |
|     tf-idf    | Logistic Regression |  0.1  | 69.19 |
|    avg-w2v    | Logistic Regression |   1   | 69.33 |
|   tf-idf-w2v  | Logistic Regression |  0.01 | 67.77 |
|  Without text | Logistic Regression |  0.01 |  57.9 |
+---------------+---------------------+-------+-------+
```

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 Logistic regression on each.

Considered logistic regression without text data in the last set of modelling.

Reference:

Applied AI Course

Stackoverflow

geekforgeeks

some other websites in case of any doubts