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 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
```

Loading 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.head()
```

Out[3]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_s
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX

In [4]:

```
resource_data.head()
```

Out[4]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95
2	p069063	Cory Stories: A Kid's Book About Living With Adhd	1	8.45
3	p069063	Dixon Ticonderoga Wood-Cased #2 HB Pencils, Bo...	2	13.59
4	p069063	EDUCATIONAL INSIGHTS FLUORESCENT LIGHT FILTERS...	3	24.95

In [5]:

```
project_data.isnull().any()
```

Out[5]:

```
Unnamed: 0                                      False
id                                              False
teacher_id                                      False
teacher_prefix                                   True
school_state                                    False
project_submitted_datetime                      False
project_grade_category                          False
project_subject_categories                      False
project_subject_subcategories                   False
project_title                                   False
project_essay_1                                 False
project_essay_2                                 False
project_essay_3                                  True
project_essay_4                                  True
project_resource_summary                        False
teacher_number_of_previously_posted_projects    False
project_is_approved                             False
dtype: bool
```

In [6]:

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

```
Number of data points in train data (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 [7]:

```
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 [8]:

```
project_data['teacher_prefix']= project_data['teacher_prefix'].fillna(project_data['tea
cher_prefix'].mode().iloc[0])
```

In [9]:

```
# 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 [10]:

```
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 subcategory

In [11]:

```
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 [12]:

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

Out[12]:

	id	teacher_prefix	school_state	project_grade_category	project_title	projec
0	p253737	Mrs.	IN	Grades PreK-2	Educational Support for English Learners at Home	My stu Englis that ar

Spliting the data into train and test

In [13]:

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

Text preprocessing

For Essay

In [14]:

```
# 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 [15]:

```
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 [16]:

```
# \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 [17]:

```
#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 [18]:

```
# 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 [19]:

```
# 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, 933.60it/s]

In [20]:

```
# 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, 966.56it/s]

In [21]:

```
# 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 [00:17<00:00, 926.82it/s]

For Title

In [22]:

```
# 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 [23]:

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

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

In [24]:

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

```
Focus our CORE!
```

In [25]:

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

```
Focus our CORE
```

In [26]:

```
# 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 [27]:

```
# 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, 22235.54it/s]
```

In [28]:

```
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, 21688.89it/s]
```

In [29]:

```
# 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, 19537.86it/s]
```

One hot encoding

Categories

In [30]:

```
# 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()), 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)

print("After Vectorizations")
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)
```

```
After Vectorizations
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 [31]:

```
vec2 = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), binary=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)

print("After Vectorizations")
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)
```

```
After Vectorizations
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 [32]:

```
# 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 [33]:

```
vec3 = CountVectorizer(vocabulary=list(sorted_school_dict.keys()), binary=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)

print("After vectorizations")
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)
```

```
After vectorizations
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()), binary=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)

print("After vectorizations")
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)
```

```
After vectorizations
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,)
```

Project grade category

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['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 [37]:

```
vec5 = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), binary=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 [38]:

```
print('After Vectorizations')
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)
```

```
After Vectorizations
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 on essay

In [39]:

```
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 onl
y on train data
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, 8789) (22445,)
(11055, 8789) (11055,)
(16500, 8789) (16500,)
```

Bag of words on title

In [40]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vec7 = CountVectorizer(min_df=10)

# 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, 1155) (22445,)
(11055, 1155) (11055,)
(16500, 1155) (16500,)
```

tfidf on essay

In [41]:

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

# 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, 8789) (22445,)
(11055, 8789) (11055,)
(16500, 8789) (16500,)
```

tfidf on title

In [42]:

```
vec9 = TfidfVectorizer(min_df=10)
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, 1155) (22445,)
(11055, 1155) (11055,)
(16500, 1155) (16500,)
```

Loading glove file

In [43]:

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

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

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

# ==============================
```

```
Loading Glove Model

1917494it [09:56, 3214.51it/s]

Done. 1917494  words loaded!
```

In [44]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-p
ickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

Using Pretrained Models: Average weighted W2V

on essay

In [45]:

```
# average Word2Vec
# compute average word2vec for each review.
train_avg_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
    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_essay.append(vector)

print(len(train_avg_w2v_essay))
print(len(train_avg_w2v_essay[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
█████| 22445/22445 [00:12<00:00, 1805.73it/s]

22445
300
```

In [46]:

```
# average Word2Vec
# compute average word2vec for each review.
cv_avg_w2v_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_cv_essay): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_avg_w2v_essay.append(vector)

print(len(cv_avg_w2v_essay))
print(len(cv_avg_w2v_essay[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█████| 11055/11055 [00:05<00:00, 2044.23it/s]

11055
300
```

In [47]:

```
# average Word2Vec
# compute average word2vec for each review.
test_avg_w2v_essay = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_test_essay): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_avg_w2v_essay.append(vector)

print(len(test_avg_w2v_essay))
print(len(test_avg_w2v_essay[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█████| 16500/16500 [00:08<00:00, 2011.33it/s]

16500
300
```

on title

In [48]:

```
# average Word2Vec
# compute average word2vec for each review.
train_avg_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
    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_title.append(vector)

print(len(train_avg_w2v_title))
print(len(train_avg_w2v_title[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████| 22445/22445 [00:00<00:00, 33369.79it/s]

22445
300
```

In [49]:

```
# average Word2Vec
# compute average word2vec for each review.
cv_avg_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
    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
    cv_avg_w2v_title.append(vector)

print(len(cv_avg_w2v_title))
print(len(cv_avg_w2v_title[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████| 11055/11055 [00:00<00:00, 31246.82it/s]

11055
300
```

In [50]:

```
# average Word2Vec
# compute average word2vec for each review.
test_avg_w2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_test_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_avg_w2v_title.append(vector)

print(len(test_avg_w2v_title))
print(len(test_avg_w2v_title[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
████| 16500/16500 [00:00<00:00, 35733.53it/s]

16500
300
```

Using Pretrained Models: TFIDF weighted W2V

on essay

In [51]:

```
# 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_ess = set(tfidf_model.get_feature_names())
```

In [52]:

```
# tfidf Word2Vec
# compute tfidf word2vec for each review.
train_tfidf_w2v_essay = []; # the avg-w2v for each sentence/review is stored in this li
st
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_ess):
            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
    train_tfidf_w2v_essay.append(vector)

print(len(train_tfidf_w2v_essay))
print(len(train_tfidf_w2v_essay[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
██████| 22445/22445 [01:25<00:00, 262.09it/s]

22445
300
```

In [53]:

```
# tfidf Word2Vec
# compute tfidf word2vec for each review.
cv_tfidf_w2v_essay = []; # the avg-w2v for each sentence/review is stored in this list
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_ess):
            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
    cv_tfidf_w2v_essay.append(vector)

print(len(cv_tfidf_w2v_essay))
print(len(cv_tfidf_w2v_essay[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██████| 11055/11055 [00:38<00:00, 286.42it/s]

11055
300
```

In [54]:

```
# tfidf Word2Vec
# compute tfidf word2vec for each review.
test_tfidf_w2v_essay = []; # the avg-w2v for each sentence/review is stored in this lis
t
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_ess):
            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
    test_tfidf_w2v_essay.append(vector)

print(len(test_tfidf_w2v_essay))
print(len(test_tfidf_w2v_essay[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██████| 16500/16500 [01:09<00:00, 237.27it/s]

16500
300
```

on title

In [55]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
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_title = set(tfidf_model.get_feature_names())
```

In [56]:

```
# tfidf Word2Vec
# compute tfidf word2vec for each review.
train_tfidf_w2v_title = []; # the avg-w2v for each sentence/review is stored in this li
st
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_title):
            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
    train_tfidf_w2v_title.append(vector)

print(len(train_tfidf_w2v_title))
print(len(train_tfidf_w2v_title[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████| 22445/22445 [00:02<00:00, 10875.48it/s]

22445
300
```

In [57]:

```
# tfidf Word2Vec
# compute tfidf word2vec for each review.
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_title):
            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
    cv_tfidf_w2v_title.append(vector)

print(len(cv_tfidf_w2v_title))
print(len(cv_tfidf_w2v_title[0]))
```

```
100%|████████████████████████████████████████████████████████████
████| 11055/11055 [00:01<00:00, 10718.34it/s]

11055
300
```

In [58]:

```
# tfidf Word2Vec
# compute tfidf word2vec for each review.
test_tfidf_w2v_title = []; # the avg-w2v for each sentence/review is stored in this lis
t
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_title):
            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
    test_tfidf_w2v_title.append(vector)

print(len(test_tfidf_w2v_title))
print(len(test_tfidf_w2v_title[0]))
```

```
100%|████████████████████████████████████████████████████████████████
████| 16500/16500 [00:01<00:00, 11354.52it/s]

16500
300
```

price

In [59]:

```
# 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()
```

Out[59]:

	id	price	quantity
0	p000027	782.13	15
1	p000052	114.98	2
2	p000147	13.13	25
3	p000157	3508.32	9
4	p000169	573.89	3

In [60]:

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

Out[60]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_s
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX

In [61]:

```
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 [62]:

```
# 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
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
# 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 = Normalizer()

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]:

```
X_train_price_stndrd = X_train_price_stndrd.reshape(-1,1)
X_cv_price_stndrd = X_cv_price_stndrd.reshape(-1,1)
X_test_price_stndrd = X_test_price_stndrd.reshape(-1,1)
```

In [67]:

```
X_train_price_stndrd
```

Out[67]:

```
array([[0.00605875],
       [0.00605875],
       [0.00605875],
       ...,
       [0.00605875],
       [0.00605875],
       [0.00605875]])
```

In [68]:

```
X_cv_price_stndrd
```

Out[68]:

```
array([[0.00793744],
       [0.00508771],
       [0.00793744],
       ...,
       [0.00704838],
       [0.00793744],
       [0.00793744]])
```

In [69]:

```
X_test_price_stndrd
```

Out[69]:

```
array([[0.00768376],
       [0.00768376],
       [0.00768376],
       ...,
       [0.00768376],
       [0.00768376],
       [0.00768376]])
```

quantity

In [70]:

```
# 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 [72]:

```
# quantity
quantity_scalar = Normalizer()
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 [73]:

```
X_train_quantity_stndrd = X_train_quantity_stndrd.reshape(-1,1)
X_cv_quantity_stndrd = X_cv_quantity_stndrd.reshape(-1,1)
X_test_quantity_stndrd = X_test_quantity_stndrd.reshape(-1,1)
```

In [74]:

```
X_train_quantity_stndrd
```

Out[74]:

```
array([[1.],
       [1.],
       [1.],
       ...,
       [1.],
       [1.],
       [1.]])
```

In [75]:

```
X_cv_quantity_stndrd
```

Out[75]:

```
array([[1.],
       [1.],
       [1.],
       ...,
       [1.],
       [1.],
       [1.]])
```

In [76]:

```
X_test_quantity_stndrd
```

Out[76]:

```
array([[1.],
       [1.],
       [1.],
       ...,
       [1.],
       [1.],
       [1.]])
```

In [77]:

```
# 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 [78]:

```
# 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))
```

In [79]:

```
X_train_prev_proj_stndrd = X_train_prev_proj_stndrd.reshape(-1,1)
X_cv_prev_proj_stndrd = X_cv_prev_proj_stndrd.reshape(-1,1)
X_test_prev_proj_stndrd = X_test_prev_proj_stndrd.reshape(-1,1)
```

In [80]:

```
X_train_prev_proj_stndrd
```

Out[80]:

```
array([[0.],
       [1.],
       [0.],
       ...,
       [1.],
       [1.],
       [1.]])
```

In [81]:

```
X_cv_prev_proj_stndrd
```

Out[81]:

```
array([[0.],
       [1.],
       [0.],
       ...,
       [1.],
       [1.],
       [1.]])
```

In [82]:

```
X_test_prev_proj_stndrd
```

Out[82]:

```
array([[1.],
       [1.],
       [0.],
       ...,
       [1.],
       [1.],
       [1.]])
```

In [83]:

```
X_train.head()
```

Out[83]:

	id	teacher_prefix	school_state	project_grade_category	project_title	projec
0	p180158	Mrs.	GA	Grades PreK-2	Store It!	As an a low- incom povert
1	p074924	Mrs.	NJ	Grades 6-8	Back to School Essentials	Every my stu off the
2	p259471	Mrs.	WA	Grades PreK-2	Moving with Mrs. March	I want my far they a
3	p191716	Mr.	PA	Grades 9-12	Needed materials	My stu come major distr...
4	p041322	Mr.	NY	Grades 9-12	Photography Class; Our American Stories: Immig...	We ar Title I school Brookl

In [84]:

```
print(X_train_cat_ohe.shape)
print(X_train_sub_cat_ohe.shape)
print(X_train_state_ohe.shape)
print(X_train_teacher_ohe.shape)
print(X_train_grade_ohe.shape)
print(X_train_essay_bow.shape)
print(X_train_ess_tfidf.shape)
print(X_train_title_bow.shape)
print(X_train_title_tfidf.shape)
print(X_train_price_stndrd.shape)
print(X_train_quantity_stndrd.shape)
print(X_train_prev_proj_stndrd.shape)
```

```
(22445, 9)
(22445, 30)
(22445, 51)
(22445, 5)
(22445, 5)
(22445, 8789)
(22445, 8789)
(22445, 1155)
(22445, 1155)
(22445, 1)
(22445, 1)
(22445, 1)
```

Concatenating all the features

In [85]:

```
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_trai
n_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)).tocsr()
```

In [86]:

```
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)).tocsr()
```

In [87]:

```
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)).tocsr()
```

In [88]:

```
print("Final Data matrix")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 10047) (22445,)
(11055, 10047) (11055,)
(16500, 10047) (16500,)
====================================================================================================
```

In [89]:

```
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_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()
```

In [90]:

```
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_stndrd, X_cv_prev_proj_stndrd)).tocsr()
```

In [91]:

```
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_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)).tocsr()
```

In [92]:

```
print('Final data matrix')
print(X_train_tfidf.shape, y_train.shape)
print(X_cv_tfidf.shape, y_cv.shape)
print(X_test_tfidf.shape, y_test.shape)
print("="*100)
```

```
Final data matrix
(22445, 10047) (22445,)
(11055, 10047) (11055,)
(16500, 10047) (16500,)
====================================================================================================
```

In [93]:

```
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((train_avg_w2v_essay,train_avg_w2v_title,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, X_train_prev_proj_stndrd)).tocsr()
```

In [94]:

```
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((cv_avg_w2v_essay,cv_avg_w2v_title,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)).tocsr()
```

In [95]:

```
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((test_avg_w2v_essay,test_avg_w2v_title,X_test_teacher_ohe,X_tes
t_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()
```

In [96]:

```
print('Final data matrix')
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)
print("="*100)
```

```
Final data matrix
(22445, 703) (22445,)
(11055, 703) (11055,)
(16500, 703) (16500,)
====================================================================================
==========================
```

In [97]:

```
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((train_tfidf_w2v_essay,train_tfidf_w2v_title,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().toarray()
```

In [98]:

```
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((cv_tfidf_w2v_essay,cv_tfidf_w2v_title,X_cv_teacher_ohe,X_cv_ca
t_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()
```

In [99]:

```
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((test_tfidf_w2v_essay,test_tfidf_w2v_title,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()
```

In [100]:

```
print('Final data matrix')
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)
print("="*100)
```

```
Final data matrix
(22445, 703) (22445,)
(11055, 703) (11055,)
(16500, 703) (16500,)
====================================================================================
==========================
```

KNN on BOW featurization

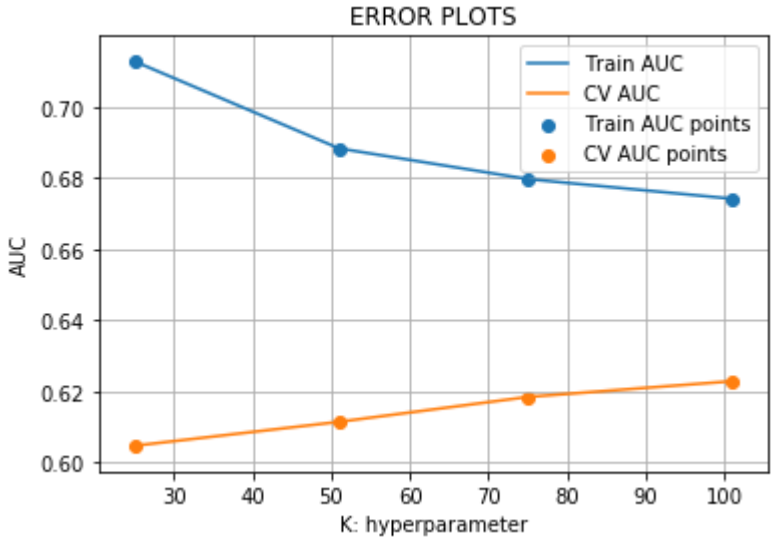
In [101]:

```
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 [102]:

```
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.

"""

train_auc = []
cv_auc = []
K = [25,51,75,101]# min k causes overfitting, max k causes underfitting
#K = range(1,50,4)
for i in  tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')# takes the k from the
i th list value
    neigh.fit(X_train_bow, y_train)# fit the model


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

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


    # roc curve
    #Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from pred
iction 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(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
100%|████████████████████████████████████████████████████████████████████████████| 4/4 [08:14<00:00, 123.58s/it]
```



best alpha

In [103]:

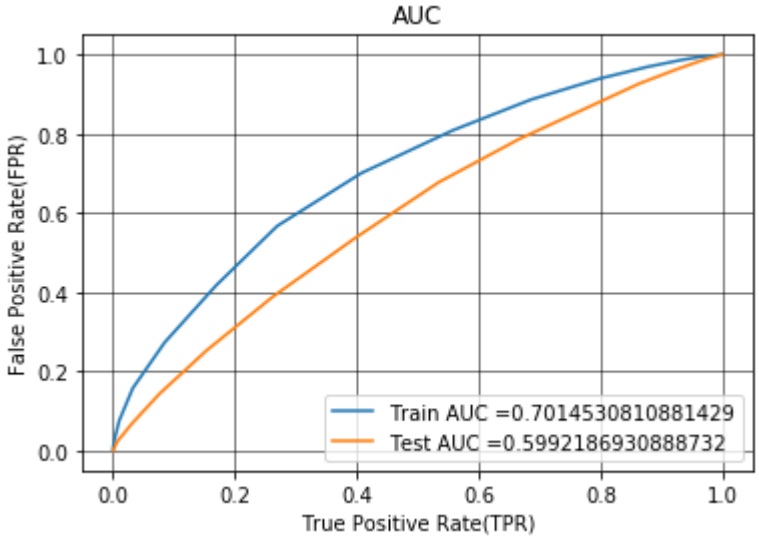
```
optimal_k= K[cv_auc.index(max(cv_auc))]
k_values=[math.log(x) for x in K]
print('optimal k for which auc is maximum : ',optimal_k)
```

```
optimal k for which auc is maximum :  101
```

Hyperparameter tuning

In [104]:

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


knn = KNeighborsClassifier(n_neighbors=32,algorithm='brute')
knn.fit(X_train_bow ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(knn, X_train_bow)
y_test_pred = batch_predict(knn, 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 [105]:

```
# 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 [106]:

```
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)
```

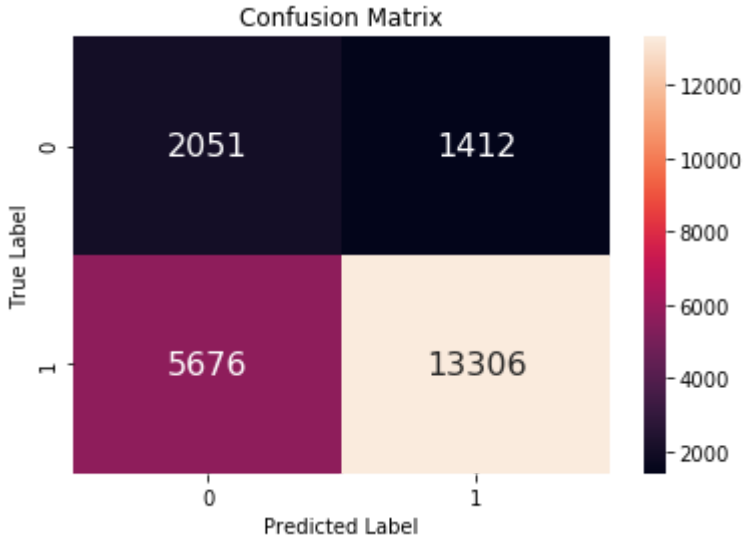
```
the maximum value of tpr*(1-fpr) 0.41516307392510365 for threshold 0.781
Train confusion matrix
[[ 2051  1412]
 [ 5676 13306]]
Test confusion matrix
[[1186 1360]
 [4502 9452]]
====================================================================================
==========================
```

In [107]:

```
#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[107]:

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

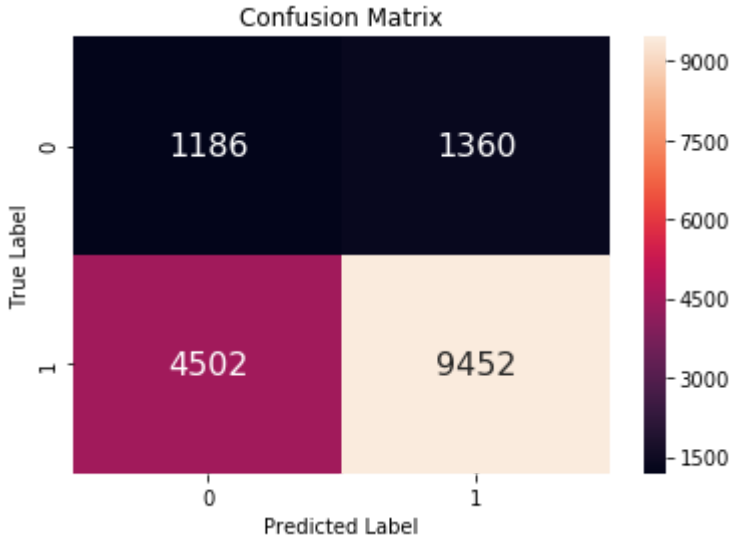


In [108]:

```
#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[108]:

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

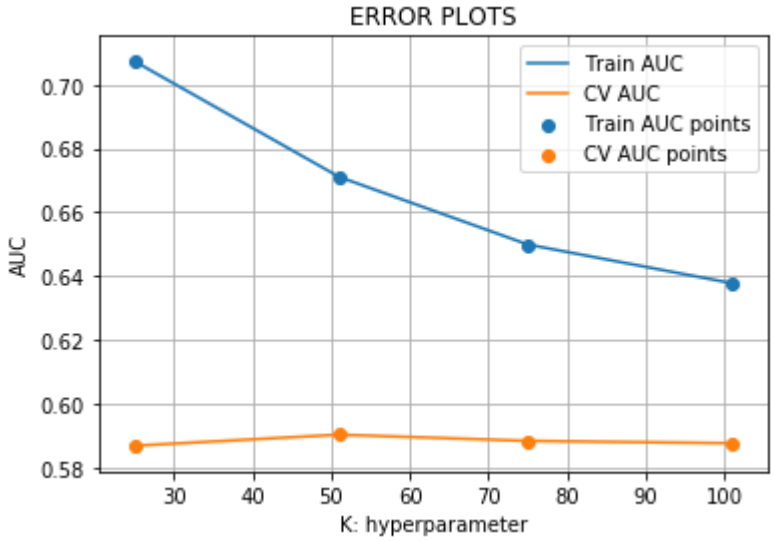


KNN on tfidf featurization

In [109]:

```
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.

"""

train_auc = []
cv_auc = []
K = [25,51,75,101]# min k causes overfitting, max k causes underfitting
#K = range(1,50,4)
for i in  tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')# takes the k from the
i th list value
    neigh.fit(X_train_tfidf, y_train)# fit the model


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

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


    # roc curve
    #Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from pred
iction 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(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
100%|██████████| 4/4 [09:04<00:00, 136.12s/it]
```



best alpha

In [110]:

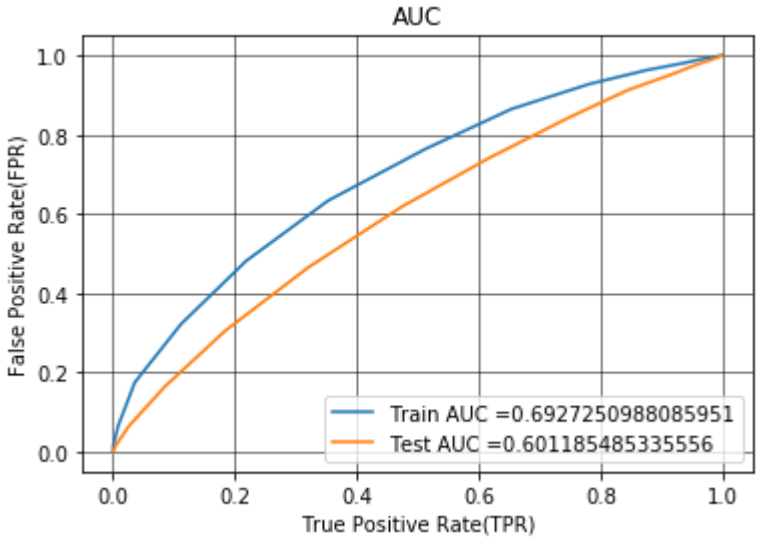
```
optimal_K= K[cv_auc.index(max(cv_auc))]
K_values=[math.log(x) for x in K]
print('optimal k for which auc is maximum : ',optimal_K)
```

```
optimal k for which auc is maximum :  51
```

Hyperparameter tuning

In [111]:

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


knn = KNeighborsClassifier(n_neighbors=32,algorithm='brute')
knn.fit(X_train_tfidf ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(knn, X_train_tfidf)
y_test_pred = batch_predict(knn, 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 [112]:

```
# 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
```

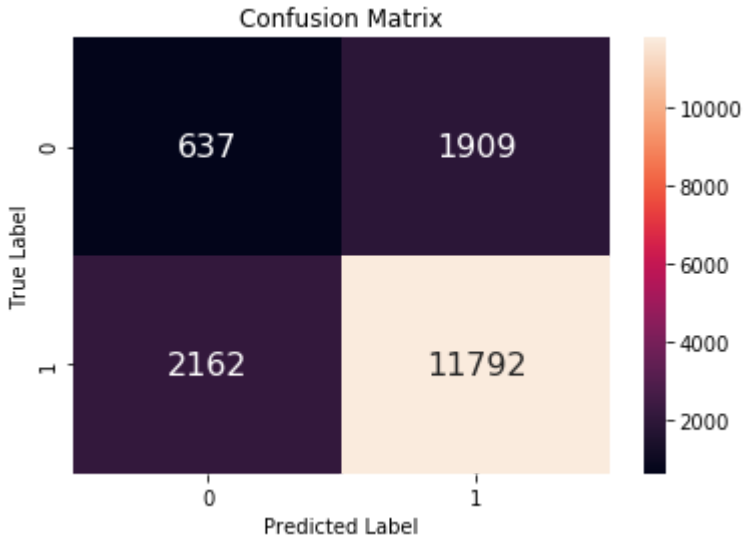
Confusion Matrix

In [113]:

```
#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[113]:

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

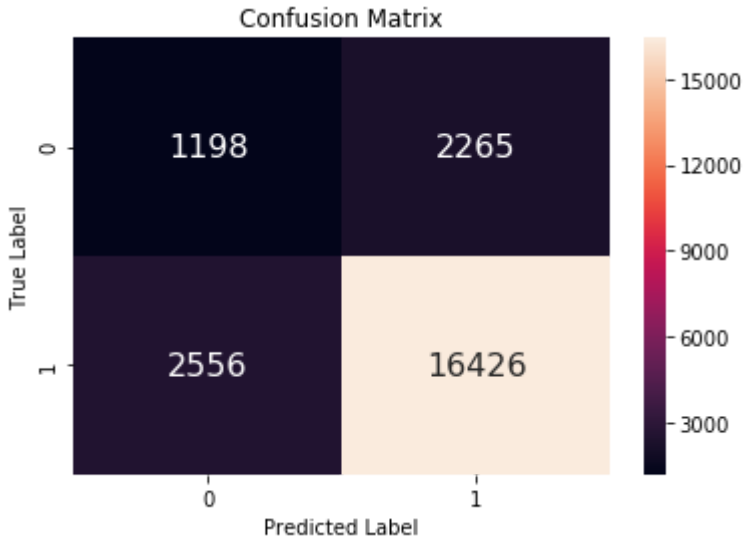


In [114]:

```
#stackoverflow.com/questions/54018742/valueerror-classification-metrics-cant-handle-a-mix-of-unknown-and-binary-targ
matrix = confusion_matrix(y_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[114]:

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

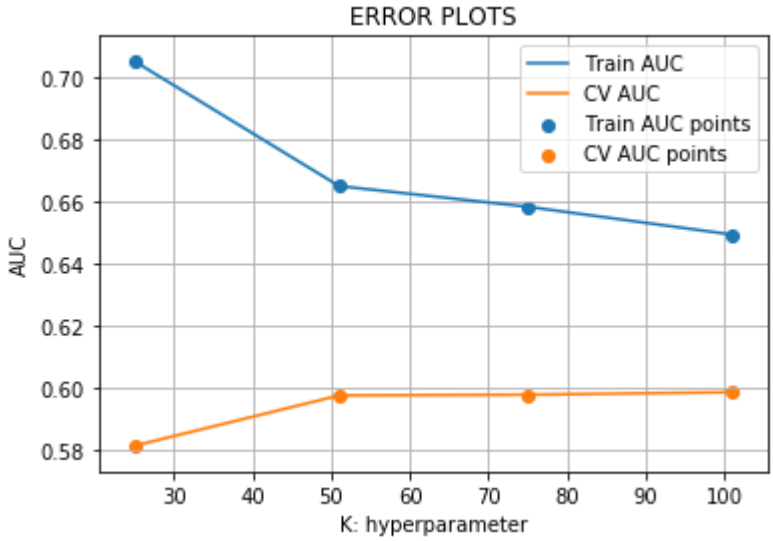


KNN on avg_w2v

In [115]:

```
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.

"""

train_auc = []
cv_auc = []
K = [25,51,75,101]# min k causes overfitting, max k causes underfitting
#K = range(1,50,4)
for i in  tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')# takes the k from the
i th list value
    neigh.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 the positive class
    # not the predicted outputs

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


    # roc curve
    #Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from pred
iction 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(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
100%|██████████| 4/4 [1:40:45<00:00, 1511.42s/it]
```



best alpha

In [116]:

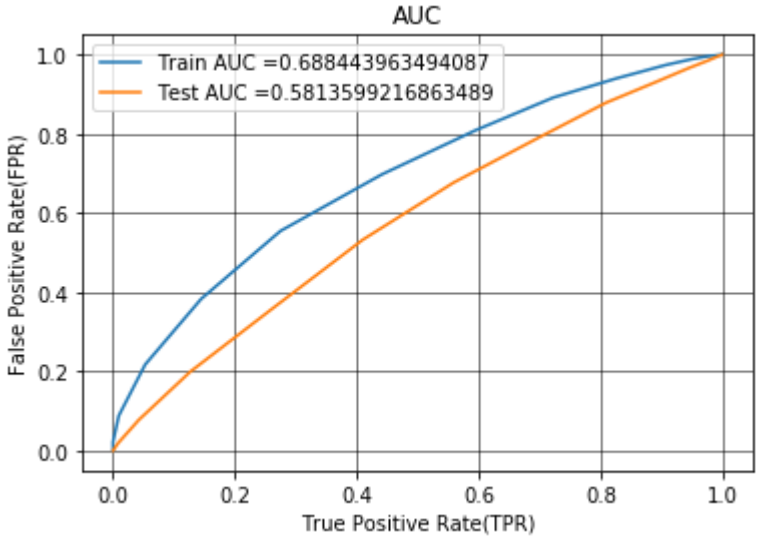
```
optimal_K= K[cv_auc.index(max(cv_auc))]
K_values=[math.log(x) for x in K]
print('optimal k for which auc is maximum : ',optimal_K)
```

```
optimal k for which auc is maximum :  101
```

Hyperparameter tuning

In [117]:

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


knn = KNeighborsClassifier(n_neighbors=32,algorithm='brute')
knn.fit(X_train_avg_w2v ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(knn, X_train_avg_w2v)
y_test_pred = batch_predict(knn, 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 [118]:

```
# 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
```

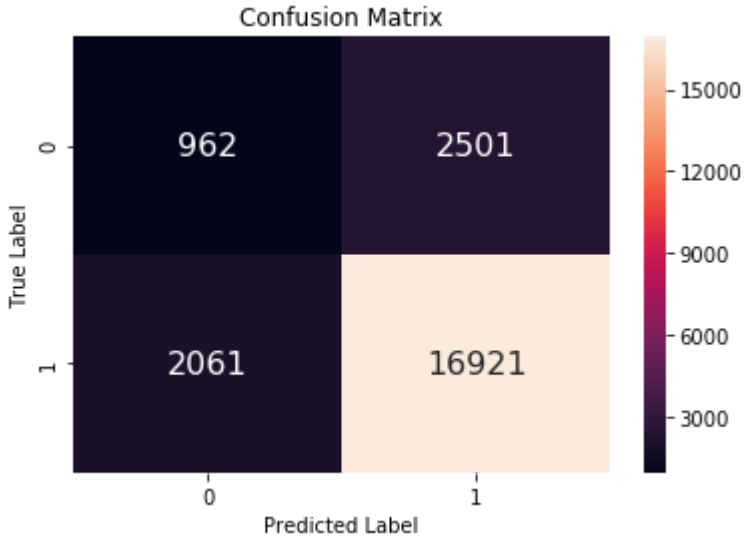
Confusion Matrix

In [119]:

```
#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[119]:

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

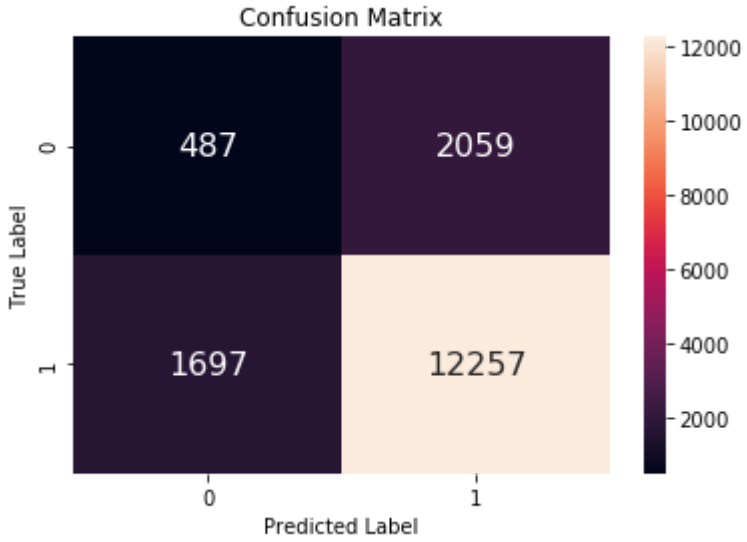


In [120]:

```
#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[120]:

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

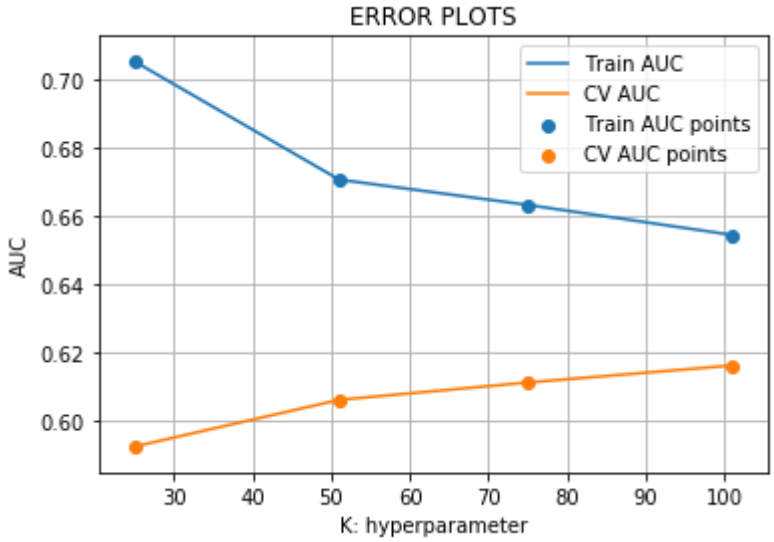


KNN on tfidf_w2v

In [137]:

```
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.

"""

train_auc = []
cv_auc = []
K = [25,51,75,101]# min k causes overfitting, max k causes underfitting
#K = range(1,50,4)
for i in  tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')# takes the k from the
i th list value
    neigh.fit(X_train_tfidf_w2v, y_train)# fit the model


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

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


    # roc curve
    #Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from pred
iction 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(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

```
  0%|
| 0/4 [00:00<?, ?it/s]
 25%|
| 1/4 [03:43<11:09, 223.01s/it]
 50%|
| 2/4 [07:30<07:28, 224.39s/it]
 75%|
| 3/4 [11:06<03:41, 221.78s/it]
100%|
| 4/4 [14:49<00:00, 222.40s/it]
```



best alpha

In [138]:

```
optimal_K= K[cv_auc.index(max(cv_auc))]
K_values=[math.log(x) for x in K]
print('optimal k for which auc is maximum : ',optimal_K)
```

```
optimal k for which auc is maximum :  101
```

Hyperparameter tuning

In [139]:

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


knn = KNeighborsClassifier(n_neighbors=32,algorithm='brute')
knn.fit(X_train_tfidf_w2v ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(knn, X_train_tfidf_w2v)
y_test_pred = batch_predict(knn, 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 [140]:

```
# 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
```

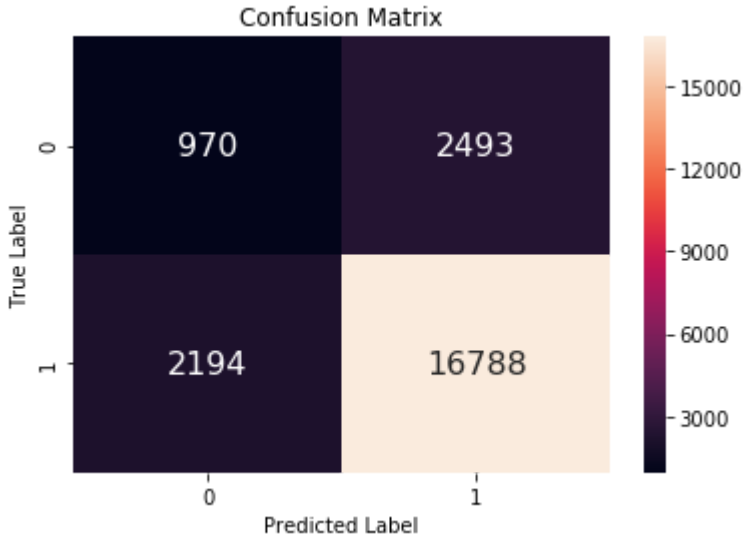
Confusion Matrix

In [141]:

```
#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[141]:

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

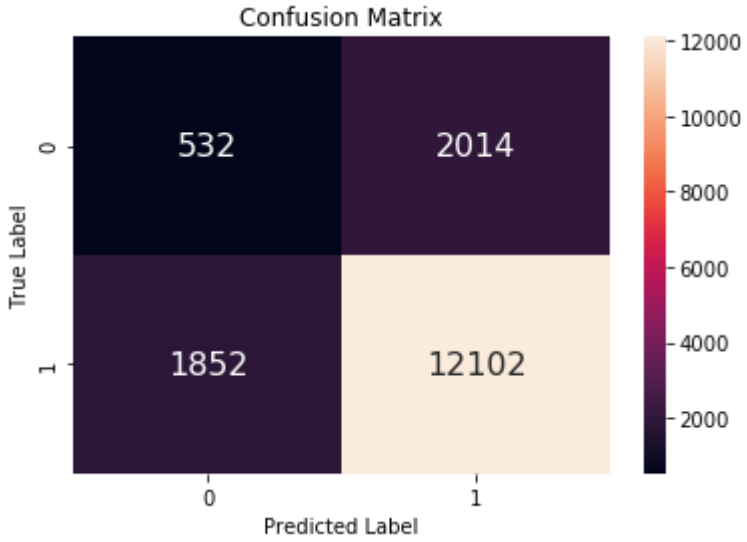


In [142]:

```
#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[142]:

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



top 2000 features from tfidf using selectkBest

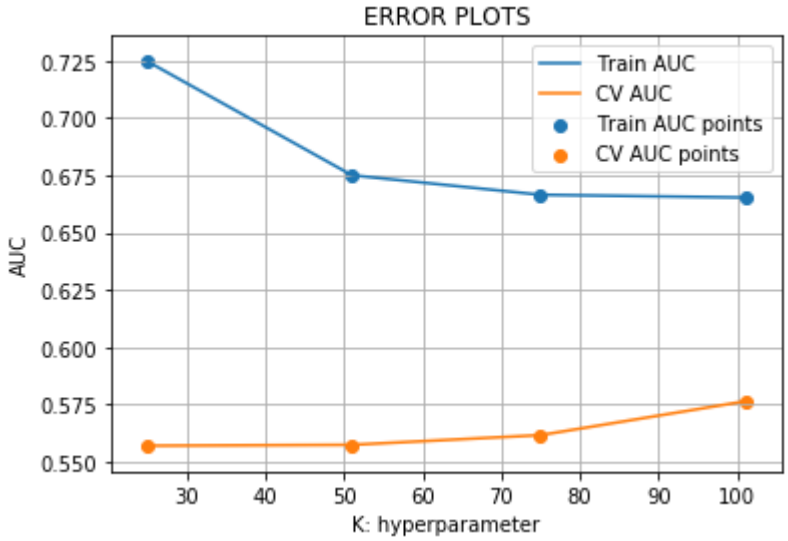
In [127]:

```
from sklearn.feature_selection import SelectKBest, chi2
t = SelectKBest(chi2,k=2000).fit(X_train_tfidf, y_train)
X_train = t.transform(X_train_tfidf)
X_cv = t.transform(X_cv_tfidf)
X_test = t.transform(X_test_tfidf)
```

In [128]:

```
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.

"""

train_auc = []
cv_auc = []
K = [25,51,75,101]# min k causes overfitting, max k causes underfitting
#K = range(1,50,4)
for i in  tqdm(K):

    neigh = KNeighborsClassifier(n_neighbors=i,algorithm='brute')# takes the k from the
i th list value
    neigh.fit(X_train, y_train)# fit the model


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

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


    # roc curve
    #Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from pred
iction 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(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

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

100%|██| 4/4 [04:40<00:00, 70.07s/it]



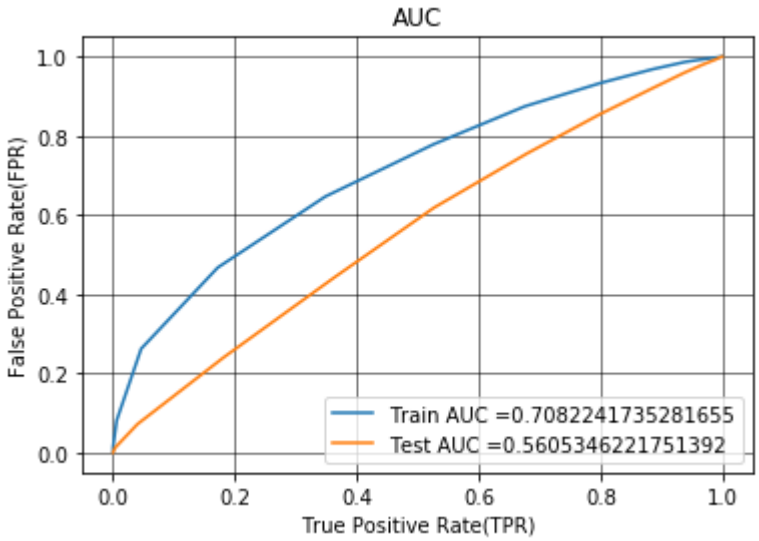
In [129]:

```
optimal_K= K[cv_auc.index(max(cv_auc))]
K_values=[math.log(x) for x in K]
print('optimal k for which auc is maximum : ',optimal_K)
```

optimal k for which auc is maximum : 101

In [130]:

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


knn = KNeighborsClassifier(n_neighbors=32,algorithm='brute')
knn.fit(X_train ,y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batch_predict(knn, X_train)
y_test_pred = batch_predict(knn, 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()
```



In [131]:

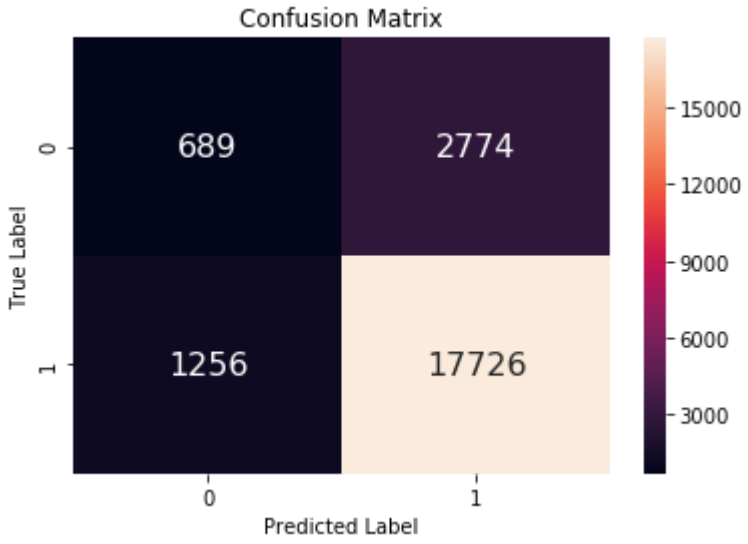
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
 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

In [132]:

```
#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[132]:

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

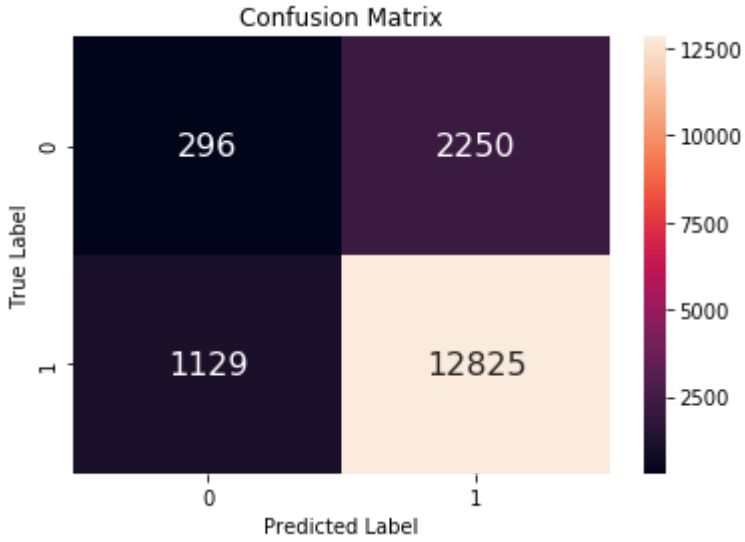


In [133]:

```
#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[133]:

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



In [2]:

```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = 'Classification Report'

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

ptable.add_row(["BOW","brute",101,59.92])
ptable.add_row(["tf-idf", "brute",101,60.11])
ptable.add_row(["avg_w2v","brute",101,58.13])
ptable.add_row(["tfidf_w2v","brute",101,59.00])
ptable.add_row(["top 2000","brute",101,56.05])

print(ptable)
```

```
+---------------+-------+-----+-------+
| Vectorization | Model |  K  |  AUC  |
+---------------+-------+-----+-------+
|      BOW      | brute | 101 | 59.92 |
|     tf-idf    | brute | 101 | 60.11 |
|    avg_w2v    | brute | 101 | 58.13 |
|   tfidf_w2v   | brute | 101 |  59.0 |
|    top 2000   | brute | 101 | 56.05 |
+---------------+-------+-----+-------+
```

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 for featurizing the data. After concatenating the data applied KNN classifier.