DonorsChoose

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

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example**
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targete enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories f following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal ((https://en.wikipedia.org/wiki/List_of_U.S._state_abbr **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategori **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the projec • `My students need hands on literacy mater sensory needs!`

Feature	Description
project_essay_1	First application essay*
project_essay_2	Second application essay*
project_essay_3	Third application essay*
project_essay_4	Fourth application essay*
project_submitted_datetime	Datetime when project application was submitted. **Ex** 12:43:56.245
teacher_id	A unique identifier for the teacher of the proposed pro bdf8baa8fedef6bfeec7ae4ff1c15c56
teacher_prefix	Teacher's title. One of the following enumerated value • nan • Dr. • Mr. • Mrs. • Ms. • Teacher.
teacher_number_of_previously_posted_projects	Number of project applications previously submitted b **Example:** 2

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

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

Feature	Description
id	A project_id value from the train.csv file. **Example:** p036502
description	Desciption of the resource. **Example:** Tenor Saxophone Reeds, Box of 25
quantity	Quantity of the resource required. **Example:** 3
price	Price of the resource required. **Example:** 9.95

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

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

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

Notes on the Essay Data

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

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_3:__ "Close by sharing why your project will make a difference"

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

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [2]:

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

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

In [4]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.col
umns)]


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/40840
39
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]


project_data.head(2)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	scho
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

In [5]:

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

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

Out[5]:

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

1.2 preprocessing of `project_subject_categories`

In [6]:

```
catogories = 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 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('&','_') # 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)
#project_data.head(2)

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

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

1.3 preprocessing of `project_subject_subcategories`

In [7]:

```
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 PROJECT GRADE CATEGORY

In [8]:

```
grade_categories=list(project_data['project_grade_category'].values)
clean_grades=[]
for i in grade_categories:
    temp=""
    for j in i.split(','):
        j=j.replace(' ','_')
        j=j.replace('-','_')
        temp+=j
        clean_grades.append(temp)
project_data['clean_grades']=clean_grades
project_data.drop(['project_grade_category'],axis=1,inplace=True)
```

1.3 Text preprocessing

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)
```

In [10]:

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

Out[10]:

	Unnamed: 0	id	teacher_id	teacher_prefix	scho
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

In [11]:

```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [12]:

```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

I have been fortunate enough to use the Fairy Tale STEM kits in my classro
om as well as the STEM journals, which my students really enjoyed. I woul
d love to implement more of the Lakeshore STEM kits in my classroom for th
e next school year as they provide excellent and engaging STEM lessons.My
students come from a variety of backgrounds, including language and socioe
conomic status. Many of them don't have a lot of experience in science an
d engineering and these kits give me the materials to provide these exciti
ng opportunities for my students.Each month I try to do several science or
STEM/STEAM projects. I would use the kits and robot to help guide my scie
nce instruction in engaging and meaningful ways. I can adapt the kits to
my current language arts pacing guide where we already teach some of the m
aterial in the kits like tall tales (Paul Bunyan) or Johnny Appleseed. Th
e following units will be taught in the next school year where I will impl
ement these kits: magnets, motion, sink vs. float, robots. I often get to
these units and don't know If I am teaching the right way or using the rig
ht materials.   The kits will give me additional ideas, strategies, and l
essons to prepare my students in science.It is challenging to develop high
quality science activities. These kits give me the materials I need to pr
ovide my students with science activities that will go along with the curr
iculum in my classroom. Although I have some things (like magnets) in my
classroom, I don't know how to use them effectively. The kits will provid
e me with the right amount of materials and show me how to use them in an
appropriate way.

==========

I teach high school English to students with learning and behavioral disab
ilities. My students all vary in their ability level. However, the ultimat
e goal is to increase all students literacy levels. This includes their re
ading, writing, and communication levels.I teach a really dynamic group of
students. However, my students face a lot of challenges. My students all l
ive in poverty and in a dangerous neighborhood. Despite these challenges,
I have students who have the the desire to defeat these challenges. My stu
dents all have learning disabilities and currently all are performing belo
w grade level. My students are visual learners and will benefit from a cla
ssroom that fulfills their preferred learning style.The materials I am req
uesting will allow my students to be prepared for the classroom with the n
ecessary supplies. Too often I am challenged with students who come to sc
hool unprepared for class due to economic challenges. I want my students
to be able to focus on learning and not how they will be able to get schoo
l supplies. The supplies will last all year. Students will be able to co
mplete written assignments and maintain a classroom journal. The chart pa
per will be used to make learning more visual in class and to create poste
rs to aid students in their learning. The students have access to a class
room printer. The toner will be used to print student work that is comple
ted on the classroom Chromebooks.I want to try and remove all barriers for
the students learning and create opportunities for learning. One of the bi
ggest barriers is the students not having the resources to get pens, pape
r, and folders. My students will be able to increase their literacy skills
because of this project.

==========

\"Life moves pretty fast. If you don't stop and look around once in awhil
e, you could miss it.\" from the movie, Ferris Bueller's Day Off. Think
back...what do you remember about your grandparents? How amazing would it
be to be able to flip through a book to see a day in their lives?My second
graders are voracious readers! They love to read both fiction and nonficti
on books. Their favorite characters include Pete the Cat, Fly Guy, Piggie
and Elephant, and Mercy Watson. They also love to read about insects, spac
e and plants. My students are hungry bookworms! My students are eager to l
earn and read about the world around them. My kids love to be at school an
d are like little sponges absorbing everything around them. Their parents
work long hours and usually do not see their children. My students are usu
ally cared for by their grandparents or a family friend. Most of my studen

ts do not have someone who speaks English at home. Thus it is difficult fo
r my students to acquire language.Now think forward... wouldn't it mean a
lot to your kids, nieces or nephews or grandchildren, to be able to see a
day in your life today 30 years from now? Memories are so precious to us a
nd being able to share these memories with future generations will be a re
warding experience. As part of our social studies curriculum, students wi
ll be learning about changes over time. Students will be studying photos
to learn about how their community has changed over time. In particular,
we will look at photos to study how the land, buildings, clothing, and sch
ools have changed over time. As a culminating activity, my students will
capture a slice of their history and preserve it through scrap booking. Ke
y important events in their young lives will be documented with the date,
location, and names. Students will be using photos from home and from sc
hool to create their second grade memories. Their scrap books will prese
rve their unique stories for future generations to enjoy.Your donation to
this project will provide my second graders with an opportunity to learn a
bout social studies in a fun and creative manner. Through their scrapbook
s, children will share their story with others and have a historical docum
ent for the rest of their lives.

==========

\"A person's a person, no matter how small.\" (Dr.Seuss) I teach the small
est students with the biggest enthusiasm for learning. My students learn i
n many different ways using all of our senses and multiple intelligences.
I use a wide range of techniques to help all my students succeed. \r\nStud
ents in my class come from a variety of different backgrounds which makes
for wonderful sharing of experiences and cultures, including Native Americ
ans.\r\nOur school is a caring community of successful learners which can
be seen through collaborative student project based learning in and out of
the classroom. Kindergarteners in my class love to work with hands-on mate
rials and have many different opportunities to practice a skill before it
is mastered. Having the social skills to work cooperatively with friends i
s a crucial aspect of the kindergarten curriculum.Montana is the perfect p
lace to learn about agriculture and nutrition. My students love to role pl
ay in our pretend kitchen in the early childhood classroom. I have had sev
eral kids ask me, \"Can we try cooking with REAL food?\" I will take their
idea and create \"Common Core Cooking Lessons\" where we learn important m
ath and writing concepts while cooking delicious healthy food for snack ti
me. My students will have a grounded appreciation for the work that went i
nto making the food and knowledge of where the ingredients came from as we
ll as how it's healthy for their bodies. This project would expand our lea
rning of nutrition and agricultural cooking recipes by having us peel our
own apples to make homemade applesauce, make our own bread, and mix up hea
lthy plants from our classroom garden in the spring. We will also create o
ur own cookbooks to be printed and shared with families. \r\nStudents will
gain math and literature skills as well as a life long enjoyment for healt
hy cooking.nannan

==========

My classroom consists of twenty-two amazing sixth graders from different c
ultures and backgrounds. They are a social bunch who enjoy working in part
ners and working with groups. They are hard-working and eager to head to m
iddle school next year. My job is to get them ready to make this transitio
n and make it as smooth as possible. In order to do this, my students need
to come to school every day and feel safe and ready to learn. Because they
are getting ready to head to middle school, I give them lots of choice- ch
oice on where to sit and work, the order to complete assignments, choice o
f projects, etc. Part of the students feeling safe is the ability for them
to come into a welcoming, encouraging environment. My room is colorful and
the atmosphere is casual. I want them to take ownership of the classroom b
ecause we ALL share it together. Because my time with them is limited, I w
ant to ensure they get the most of this time and enjoy it to the best of t
heir abilities.Currently, we have twenty-two desks of differing sizes, yet

```
the desks are similar to the ones the students will use in middle school.
We also have a kidney table with crates for seating. I allow my students t
o choose their own spots while they are working independently or in group
s. More often than not, most of them move out of their desks and onto the
crates. Believe it or not, this has proven to be more successful than maki
ng them stay at their desks! It is because of this that I am looking towar
d the “Flexible Seating” option for my classroom.\r\n The students look fo
rward to their work time so they can move around the room. I would like to
get rid of the constricting desks and move toward more “fun” seating optio
ns. I am requesting various seating so my students have more options to si
t. Currently, I have a stool and a papasan chair I inherited from the prev
ious sixth-grade teacher as well as five milk crate seats I made, but I wo
uld like to give them more options and reduce the competition for the “goo
d seats”. I am also requesting two rugs as not only more seating options b
ut to make the classroom more welcoming and appealing. In order for my stu
dents to be able to write and complete work without desks, I am requesting
a class set of clipboards. Finally, due to curriculum that requires groups
to work together, I am requesting tables that we can fold up when we are n
ot using them to leave more room for our flexible seating options.\r\nI kn
ow that with more seating options, they will be that much more excited abo
ut coming to school! Thank you for your support in making my classroom one
students will remember forever!nannan
==================================================
```

In [13]:

```
# 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 [14]:

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

```
\"A person is a person, no matter how small.\" (Dr.Seuss) I teach the smal
lest students with the biggest enthusiasm for learning. My students learn
in many different ways using all of our senses and multiple intelligences.
I use a wide range of techniques to help all my students succeed. \r\nStud
ents in my class come from a variety of different backgrounds which makes
for wonderful sharing of experiences and cultures, including Native Americ
ans.\r\nOur school is a caring community of successful learners which can
be seen through collaborative student project based learning in and out of
the classroom. Kindergarteners in my class love to work with hands-on mate
rials and have many different opportunities to practice a skill before it
is mastered. Having the social skills to work cooperatively with friends i
s a crucial aspect of the kindergarten curriculum.Montana is the perfect p
lace to learn about agriculture and nutrition. My students love to role pl
ay in our pretend kitchen in the early childhood classroom. I have had sev
eral kids ask me, \"Can we try cooking with REAL food?\" I will take their
idea and create \"Common Core Cooking Lessons\" where we learn important m
ath and writing concepts while cooking delicious healthy food for snack ti
me. My students will have a grounded appreciation for the work that went i
nto making the food and knowledge of where the ingredients came from as we
ll as how it is healthy for their bodies. This project would expand our le
arning of nutrition and agricultural cooking recipes by having us peel our
own apples to make homemade applesauce, make our own bread, and mix up hea
lthy plants from our classroom garden in the spring. We will also create o
ur own cookbooks to be printed and shared with families. \r\nStudents will
gain math and literature skills as well as a life long enjoyment for healt
hy cooking.nannan
==================================================
```

In [15]:

```
# \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)
```

```
 A person is a person, no matter how small.  (Dr.Seuss) I teach the smalle
st students with the biggest enthusiasm for learning. My students learn in
many different ways using all of our senses and multiple intelligences. I
use a wide range of techniques to help all my students succeed.   Students
in my class come from a variety of different backgrounds which makes for w
onderful sharing of experiences and cultures, including Native Americans.
Our school is a caring community of successful learners which can be seen
through collaborative student project based learning in and out of the cla
ssroom. Kindergarteners in my class love to work with hands-on materials a
nd have many different opportunities to practice a skill before it is mast
ered. Having the social skills to work cooperatively with friends is a cru
cial aspect of the kindergarten curriculum.Montana is the perfect place to
learn about agriculture and nutrition. My students love to role play in ou
r pretend kitchen in the early childhood classroom. I have had several kid
s ask me,  Can we try cooking with REAL food?  I will take their idea and
create  Common Core Cooking Lessons  where we learn important math and wri
ting concepts while cooking delicious healthy food for snack time. My stud
ents will have a grounded appreciation for the work that went into making
the food and knowledge of where the ingredients came from as well as how i
t is healthy for their bodies. This project would expand our learning of n
utrition and agricultural cooking recipes by having us peel our own apples
to make homemade applesauce, make our own bread, and mix up healthy plants
from our classroom garden in the spring. We will also create our own cookb
ooks to be printed and shared with families.   Students will gain math and
literature skills as well as a life long enjoyment for healthy cooking.nan
nan
```

In [16]:

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

```
 A person is a person no matter how small Dr Seuss I teach the smallest st
udents with the biggest enthusiasm for learning My students learn in many
different ways using all of our senses and multiple intelligences I use a
wide range of techniques to help all my students succeed Students in my cl
ass come from a variety of different backgrounds which makes for wonderful
sharing of experiences and cultures including Native Americans Our school
is a caring community of successful learners which can be seen through col
laborative student project based learning in and out of the classroom Kind
ergarteners in my class love to work with hands on materials and have many
different opportunities to practice a skill before it is mastered Having t
he social skills to work cooperatively with friends is a crucial aspect of
the kindergarten curriculum Montana is the perfect place to learn about ag
riculture and nutrition My students love to role play in our pretend kitch
en in the early childhood classroom I have had several kids ask me Can we
try cooking with REAL food I will take their idea and create Common Core C
ooking Lessons where we learn important math and writing concepts while co
oking delicious healthy food for snack time My students will have a ground
ed appreciation for the work that went into making the food and knowledge
of where the ingredients came from as well as how it is healthy for their
bodies This project would expand our learning of nutrition and agricultura
l cooking recipes by having us peel our own apples to make homemade apples
auce make our own bread and mix up healthy plants from our classroom garde
n in the spring We will also create our own cookbooks to be printed and sh
ared with families Students will gain math and literature skills as well a
s a life long enjoyment for healthy cooking nannan
```

In [17]:

```
# 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 [18]:

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

```
100%|██████████████████████████████████████████████████████████████████
███| 109248/109248 [01:42<00:00, 1069.30it/s]
```

In [19]:

```
# after preprocesing
preprocessed_essays[20000]
```

Out[19]:

```
'person person no matter small dr seuss teach smallest students biggest en
thusiasm learning students learn many different ways using senses multiple
intelligences use wide range techniques help students succeed students cla
ss come variety different backgrounds makes wonderful sharing experiences
cultures including native americans school caring community successful lea
rners seen collaborative student project based learning classroom kinderga
rteners class love work hands materials many different opportunities pract
ice skill mastered social skills work cooperatively friends crucial aspect
kindergarten curriculum montana perfect place learn agriculture nutrition
students love role play pretend kitchen early childhood classroom several
kids ask try cooking real food take idea create common core cooking lesson
s learn important math writing concepts cooking delicious healthy food sna
ck time students grounded appreciation work went making food knowledge ing
redients came well healthy bodies project would expand learning nutrition
agricultural cooking recipes us peel apples make homemade applesauce make
bread mix healthy plants classroom garden spring also create cookbooks pri
nted shared families students gain math literature skills well life long e
njoyment healthy cooking nannan'
```

1.4 Preprocessing of `project_title`

In [20]:

```
# similarly you can preprocess the titles also
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████
██| 109248/109248 [00:04<00:00, 23556.14it/s]
```

1.5 Preparing data for models

In [21]:

```
project_data.columns
```

Out[21]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', '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_approve
d',
       'clean_categories', 'clean_subcategories', 'clean_grades', 'essa
y'],
      dtype='object')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

In [22]:

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

```
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'lit
eracy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
Shape of matrix after one hot encodig  (109248, 9)
```

In [23]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer()
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subcategories'].v
alues)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)
```

```
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_governmen
t', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economi
cs', 'environmentalscience', 'esl', 'extracurricular', 'financialliterac
y', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_welln
ess', 'history_geography', 'literacy', 'literature_writing', 'mathematic
s', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performi
ngarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'wa
rmth']
Shape of matrix after one hot encodig  (109248, 30)
```

In [24]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category al
so
```

In [25]:

```
vectorizer = CountVectorizer()
school_state_one_hot = vectorizer.fit_transform(project_data['school_state'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",school_state_one_hot.shape)
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi',
'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'm
o', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'o
k', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'w
i', 'wv', 'wy']
Shape of matrix after one hot encodig  (109248, 51)
```

In [26]:

```
x=project_data['teacher_prefix'].fillna('')
vectorizer = CountVectorizer()

teacher_prefix_one_hot = vectorizer.fit_transform(x.values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",teacher_prefix_one_hot.shape)
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
Shape of matrix after one hot encodig  (109248, 5)
```

In [27]:

```
vectorizer = CountVectorizer()
project_grade_one_hot = vectorizer.fit_transform(project_data['clean_grades'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",project_grade_one_hot.shape)
```

```
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
Shape of matrix after one hot encodig  (109248, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [28]:

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

```
Shape of matrix after one hot encodig  (109248, 16512)
```

In [29]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer=CountVectorizer(min_df=10)
title_bow=vectorizer.fit_transform(preprocessed_titles)
print('the shape of matrix of title bow',title_bow.shape)
```

```
the shape of matrix of title bow (109248, 3222)
```

1.5.2.2 TFIDF vectorizer

In [30]:

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

```
Shape of matrix after one hot encodig  (109248, 16512)
```

In [31]:

```
vectorizer=TfidfVectorizer(min_df=10)
title_tfidf=vectorizer.fit_transform(preprocessed_titles)
print('the shape of the matrix of title tfidf',title_tfidf.shape)
```

```
the shape of the matrix of title tfidf (109248, 3222)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [34]:

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

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

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

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

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

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

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


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-p
ickle-to-save-and-load-variables-in-python/

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


'''
```

Out[34]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/3823034
9/4084039\ndef loadGloveModel(gloveFile):\n    print ("Loading Glove Mode
l")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    model = {}\n    f
or line in tqdm(f):\n        splitLine = line.split()\n        word = spli
tLine[0]\n        embedding = np.array([float(val) for val in splitLine
[1:]])\n        model[word] = embedding\n    print ("Done.",len(model)," w
ords loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.
txt\')\n# ============================\nOutput:\n    \nLoading Glove Model
\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n# ======
======================\n\nwords = []\nfor i in preprocessed_essays:\n    w
ords.extend(i.split(\' \'))\n\nfor i in preprocessed_titles:\n    words.ex
tend(i.split(\' \'))\nprint("all the words in the corpus", len(words))\nwo
rds = set(words)\nprint("the unique words in the corpus", len(words))\n\ni
nter_words = set(model.keys()).intersection(words)\nprint("The number of w
ords that are present in both glove vectors and our coupus",       len(int
er_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_c
ourpus = {}\nwords_glove = set(model.keys())\nfor i in words:\n    if i in
words_glove:\n        words_courpus[i] = model[i]\nprint("word 2 vec lengt
h", len(words_courpus))\n\n\n# stronging variables into pickle files pytho
n: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables
-in-python/\n\nimport pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n
pickle.dump(words_courpus, f)\n\n\n'
```

In [32]:

```
# 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())
```

In [33]:

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

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

```
100%|████████████████████████████████████████████████████████████████
███| 109248/109248 [00:39<00:00, 2761.04it/s]

109248
300
```

In [35]:

```
avg_w2v_title_vectors = []; # the avg-w2v for each sentence/review is stored in this li
st
for sentence in tqdm(preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_title_vectors.append(vector)

print(len(avg_w2v_title_vectors))
print(len(avg_w2v_title_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██| 109248/109248 [00:02<00:00, 45793.83it/s]

109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [36]:

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

In [37]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((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
    tfidf_w2v_vectors.append(vector)

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

```
100%|████████████████████████████████████████████████████████████████
████| 109248/109248 [04:23<00:00, 414.52it/s]

109248
300
```

In [38]:

```
# Similarly you can vectorize for title also
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_titles)
# 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 [39]:

```
# compute average word2vec for each review.
tfidf_w2v_titles_vectors = []; # the avg-w2v for each sentence/review is stored in this
list
for sentence in tqdm(preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    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
    tfidf_w2v_titles_vectors.append(vector)

print(len(tfidf_w2v_titles_vectors))
print(len(tfidf_w2v_titles_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██| 109248/109248 [00:04<00:00, 22100.79it/s]

109248
300
```

1.5.3 Vectorizing Numerical features

In [40]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_i
ndex()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [41]:

```
data1=project_data.drop(['id','teacher_id','project_essay_1','project_essay_2','project
_essay_3','project_essay_4','project_is_approved'],axis=1)
data1.head(2)
data=data1[0:20000]
data[0:2]
```

Out[41]:

	Unnamed: 0	teacher_prefix	school_state	Date	project_title	project_resource_s
0	8393	Mrs.	CA	2016-04-27 00:27:36	Engineering STEAM into the Primary Classroom	My students need ST to learn critical s...
1	37728	Ms.	UT	2016-04-27 00:31:25	Sensory Tools for Focus	My students need Bc Boards for quiet sens

In [42]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.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_scalar.fit(project_data['price'].values.reshape(-1,1)) # finding the mean and sta
ndard 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 maen and variance.
price_standardized = price_scalar.transform(project_data['price'].values.reshape(-1, 1
))
```

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

In [43]:

```
price_standardized
```

Out[43]:

```
array([[ 1.16172762],
       [-0.23153793],
       [ 0.08402983],
       ...,
       [ 0.27450792],
       [-0.0282706 ],
       [-0.79625102]])
```

In [44]:

```
#vectorizing previously posted projects
projects_scalar = StandardScaler()
projects_scalar.fit(project_data['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 maen and variance.
projects_standardized = projects_scalar.transform(project_data['teacher_number_of_previ
ously_posted_projects'].values.reshape(-1, 1))
projects_standardized
```

```
C:\Users\HP\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\ut
ils\validation.py:475: DataConversionWarning:

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


Mean : 298.1193425966608, Standard deviation : 367.49634838483496

C:\Users\HP\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\ut
ils\validation.py:475: DataConversionWarning:

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

Out[44]:

```
array([[ 1.5065268 ],
       [-0.25752092],
       [-0.04151507],
       ...,
       [-0.40152481],
       [-0.36552384],
       [-0.32952286]])
```

In [45]:

```
project_data.columns
```

Out[45]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', '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_approve
d',
       'clean_categories', 'clean_subcategories', 'clean_grades', 'essay',
       'price', 'quantity'],
      dtype='object')
```

1.5.4 Merging all the above features

- we need to merge all the numerical vectors i.e catogorical, text, numerical vectors

In [46]:

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(teacher_prefix_one_hot.shape)
print(school_state_one_hot.shape)
print(project_grade_one_hot.shape)
print(title_bow.shape)
print(text_bow.shape)
print(price_standardized.shape)
print(projects_standardized.shape)
```

```
(109248, 9)
(109248, 30)
(109248, 5)
(109248, 51)
(109248, 4)
(109248, 3222)
(109248, 16512)
(109248, 1)
(109248, 1)
```

In [47]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx
:)
X = hstack((categories_one_hot, sub_categories_one_hot,teacher_prefix_one_hot,school_st
ate_one_hot,project_grade_one_hot,title_bow,text_bow, price_standardized,projects_stand
ardized))
X.shape
```

Out[47]:

```
(109248, 19835)
```

In [48]:

```
y1=project_data['project_is_approved']
print(y1.shape)
y=y1[0:20000]
```

```
(109248,)
```

In [49]:

```
print(data.head(2))
```

```
   Unnamed: 0 teacher_prefix school_state                Date  \
0        8393           Mrs.           CA 2016-04-27 00:27:36
1       37728            Ms.           UT 2016-04-27 00:31:25

                                project_title  \
0  Engineering STEAM into the Primary Classroom
1                       Sensory Tools for Focus

                           project_resource_summary  \
0  My students need STEM kits to learn critical s...
1  My students need Boogie Boards for quiet senso...

   teacher_number_of_previously_posted_projects clean_categories  \
0                                            53     Math_Science
1                                             4     SpecialNeeds

                 clean_subcategories   clean_grades  \
0  AppliedSciences Health_LifeScience  Grades_PreK_2
1                       SpecialNeeds     Grades_3_5

                                              essay   price  quantity
0  I have been fortunate enough to use the Fairy ...  725.05         4
1  Imagine being 8-9 years old. You're in your th...  213.03         8
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**

   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

2. **Hyper paramter tuning to find best K**

   - Find the best hyper parameter which results in the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
   - Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task

3. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure

   - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.

   - Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

4. **[Task-2]**

   - Select top 2000 features from feature Set 2 using `SelectKBest` (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) and then apply KNN on top of these features

   -
```
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest,
chi2
X, y = load_digits(return_X_y=True)
X.shape
X_new = SelectKBest(chi2, k=20).fit_transform(X,
y)
X_new.shape
========
output:
(1797, 64)
(1797, 20)
```

   - Repeat the steps 2 and 3 on the data matrix after feature selection

5. **Conclusion**

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link (http://zetcode.com/python/prettytable/)

Note: Data Leakage

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

2. K Nearest Neighbor

2.1 Splitting data into Train and cross validation(or test): Stratified Sampling

In [50]:

```
# please write all the code with proper documentation, and proper titles for each subse
ction
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your
code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(data,y,test_size=0.30,stratify=y)
X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.3,stratify=y_tra
in)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [51]:

```
# please write all the code with proper documentation, and proper titles for each subse
ction
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your
code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

vectorizer=CountVectorizer()
vectorizer.fit(X_train['school_state'].values)

X_train_state_encoded=vectorizer.transform(X_train['school_state'].values)
X_cv_state_encoded=vectorizer.transform(X_cv['school_state'].values)
X_test_state_encoded=vectorizer.transform(X_test['school_state'].values)
print('AFTER VECTORIZATION')
print('='*50)
print(X_train_state_encoded.shape,y_train.shape)
print(X_cv_state_encoded.shape,y_cv.shape)
print(X_test_state_encoded.shape,y_test.shape)
print(vectorizer.get_feature_names())
print('='*50)
```

```
AFTER VECTORIZATION
==================================================
(9800, 51) (9800,)
(4200, 51) (4200,)
(6000, 51) (6000,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi',
'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'm
o', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'o
k', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'w
i', 'wv', 'wy']
==================================================
```

In [52]:

```
#encoding categorical feature teacher prefix
X_train_encoding=X_train['teacher_prefix'].fillna('No prefix',inplace=True)
X_cv_encoding=X_cv['teacher_prefix'].fillna('No prefix',inplace=True)
X_test_encoding=X_test["teacher_prefix"].fillna('No prefix',inplace=True)
```

In [53]:

```
print(X_train["teacher_prefix"].unique())
```

```
['Mrs.' 'Ms.' 'Mr.' 'Teacher' 'Dr.' 'No prefix']
```

In [54]:

```
vectorizer=CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values)
X_train_prefix_encoded=vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_prefix_encoded=vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_prefix_encoded=vectorizer.transform(X_test['teacher_prefix'].values)

print('AFTER VECTORIZATION')
print('='*50)
print(X_train_prefix_encoded.shape,y_train.shape)
print(X_cv_prefix_encoded.shape,y_cv.shape)
print(X_test_prefix_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
(9800, 7) (9800,)
(4200, 7) (4200,)
(6000, 7) (6000,)
```

In [55]:

```
#encoding project grade category
vectorizer=CountVectorizer()
vectorizer.fit(X_train['clean_grades'].values)
X_train_grade_encoded=vectorizer.transform(X_train['clean_grades'].values)
X_cv_grade_encoded=vectorizer.transform(X_cv['clean_grades'].values)
X_test_grade_encoded=vectorizer.transform(X_test['clean_grades'].values)

print('AFTER VECTORIZATION')
print('='*50)
print(X_train_grade_encoded.shape,y_train.shape)
print(X_cv_grade_encoded.shape,y_cv.shape)
print(X_test_grade_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
(9800, 4) (9800,)
(4200, 4) (4200,)
(6000, 4) (6000,)
```

In [56]:

```
#encoding clean categories
vectorizer=CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values)
X_train_categories_encoded=vectorizer.transform(X_train['clean_categories'].values)
X_cv_categories_encoded=vectorizer.transform(X_cv['clean_categories'].values)
X_test_categories_encoded=vectorizer.transform(X_test['clean_categories'].values)
print("AFTER VECTORIZATION")
print('='*50)
print(X_train_categories_encoded.shape,y_train.shape)
print(X_cv_categories_encoded.shape,y_cv.shape)
print(X_test_categories_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
(9800, 7) (9800,)
(4200, 7) (4200,)
(6000, 7) (6000,)
```

In [57]:

```
#encoding subcategories
vectorizer=CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values)
X_train_subcategories_encoded=vectorizer.transform(X_train['clean_subcategories'].value
s)
X_cv_subcategories_encoded=vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcategories_encoded=vectorizer.transform(X_test['clean_subcategories'].values)
print("AFTER VECTORIZATION")
print('='*50)
print(X_train_subcategories_encoded.shape,y_train.shape)
print(X_cv_subcategories_encoded.shape,y_cv.shape)
print(X_test_subcategories_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
(9800, 28) (9800,)
(4200, 28) (4200,)
(6000, 28) (6000,)
```

In [58]:

```
X_train.columns
```

Out[58]:

```
Index(['Unnamed: 0', 'teacher_prefix', 'school_state', 'Date', 'project_ti
tle',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'clean_grades', 'essay', 'price', 'quantit
y'],
      dtype='object')
```

In [59]:

```
#encoding numerical categories---price
from sklearn.preprocessing import Normalizer
normalizer=Normalizer()
normalizer.fit(X_train['price'].values.reshape(-1,1))

X_train_price_norm=normalizer.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_norm=normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm=normalizer.transform(X_test['price'].values.reshape(-1,1))

print("after vectorization")
print(X_train_price_norm.shape,y_train.shape)
print(X_cv_price_norm.shape,y_cv.shape)
print(X_test_price_norm.shape,y_test.shape)
```

```
after vectorization
(9800, 1) (9800,)
(4200, 1) (4200,)
(6000, 1) (6000,)
```

In [60]:

```
#encoding previous projects posted by teachers
normalizer=Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-
1,1))

X_train_projects_norm=normalizer.transform(X_train['teacher_number_of_previously_posted
_projects'].values.reshape(-1,1))
X_cv_projects_norm=normalizer.transform(X_cv['teacher_number_of_previously_posted_proje
cts'].values.reshape(-1,1))
X_test_projects_norm=normalizer.transform(X_test['teacher_number_of_previously_posted_p
rojects'].values.reshape(-1,1))

print("after vectorization")
print(X_train_projects_norm.shape,y_train.shape)
print(X_cv_projects_norm.shape,y_cv.shape)
print(X_test_projects_norm.shape,y_test.shape)
```

```
after vectorization
(9800, 1) (9800,)
(4200, 1) (4200,)
(6000, 1) (6000,)
```

In [61]:

```
#encoding numerical category quantity
normalizer=Normalizer()
normalizer.fit(X_train['quantity'].values.reshape(-1,1))

X_train_quantity_norm=normalizer.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_quantity_norm=normalizer.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity_norm=normalizer.transform(X_test['quantity'].values.reshape(-1,1))

print('after vectorization')
print(X_train_quantity_norm.shape,y_train.shape)
print(X_cv_quantity_norm.shape,y_cv.shape)
print(X_test_quantity_norm.shape,y_test.shape)
```

```
after vectorization
(9800, 1) (9800,)
(4200, 1) (4200,)
(6000, 1) (6000,)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [62]:

```
# please write all the code with proper documentation, and proper titles for each subse
ction
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your
code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

print(X_train.shape)
print(y_train.shape)
print(X_cv.shape)
print(y_cv.shape)
print(X_test.shape)
print(y_test.shape)
print('='*50)


vectorizer=CountVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)
vectorizer.fit(X_train['essay'].values)
X_train_essay_bow=vectorizer.transform(X_train['essay'].values)
#print(X_train_essay_bow.shape)
X_cv_essay_bow=vectorizer.transform(X_cv["essay"].values)
X_test_essay_bow=vectorizer.transform(X_test['essay'].values)

print('AFTER VECTORIZATION')
print('='*50)
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)
```

```
(9800, 13)
(9800,)
(4200, 13)
(4200,)
(6000, 13)
(6000,)
==================================================
AFTER VECTORIZATION
==================================================
(9800, 5000) (9800,)
(4200, 5000) (4200,)
(6000, 5000) (6000,)
```

In [63]:

```
#encoding project title
vectorizer=CountVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)
vectorizer.fit(X_train['project_title'].values)
X_train_title_bow=vectorizer.transform(X_train['project_title'].values)
X_cv_title_bow=vectorizer.transform(X_cv['project_title'].values)
X_test_title_bow=vectorizer.transform(X_test['project_title'].values)
print("after vectorization")
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 vectorization
(9800, 1190) (9800,)
(4200, 1190) (4200,)
(6000, 1190) (6000,)
```

2.4 Appling KNN on different kind of featurization as mentioned in the instructions

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

In [64]:

```
# please write all the code with proper documentation, and proper titles for each subse
ction
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your
code

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2.4.1 Applying KNN brute force on BOW, SET 1

In [65]:

```
#creating final data matrix
from scipy.sparse import hstack
final_train_bow=hstack((X_train_state_encoded,X_train_prefix_encoded,X_train_grade_enco
ded,X_train_categories_encoded,X_train_subcategories_encoded,X_train_price_norm,X_train
_projects_norm,X_train_quantity_norm,X_train_essay_bow,X_train_title_bow)).tocsr()
final_cv_bow=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_cat
egories_encoded,X_cv_subcategories_encoded,X_cv_price_norm,X_cv_projects_norm,X_cv_quan
tity_norm,X_cv_essay_bow,X_cv_title_bow)).tocsr()
final_test_bow=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encoded,
X_test_categories_encoded,X_test_subcategories_encoded,X_test_price_norm,X_test_project
s_norm,X_test_quantity_norm,X_test_essay_bow,X_test_title_bow)).tocsr()
print(final_train_bow.shape,y_train.shape)
print(final_cv_bow.shape,y_cv.shape)
print(final_test_bow.shape,y_test.shape)
```

```
(9800, 6290) (9800,)
(4200, 6290) (4200,)
(6000, 6290) (6000,)
```

In [66]:

```
# Please write all the code with proper documentation
def batch_predict(clf,data):


    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000

    for i in range(0,tr_loop ,1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])

    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [67]:

```
y_train.shape
```

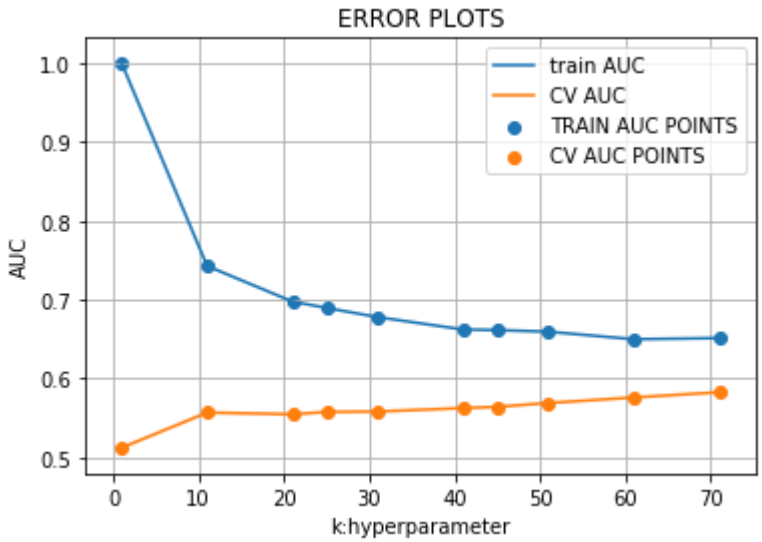
Out[67]:

```
(9800,)
```

In [68]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc=[]
cv_auc=[]
k=[1,11,21,25,31,41,45,51,61,71]
for i in tqdm(k):
    neigh=KNeighborsClassifier(n_neighbors=i)
    neigh.fit(final_train_bow,y_train)


    y_tr_pred=batch_predict(neigh,final_train_bow)
    y_cv_pred=batch_predict(neigh,final_cv_bow)

    train_auc.append(roc_auc_score(y_train,y_tr_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%|██████████| 10/10 [05:32<00:00, 33.38s/it]
```



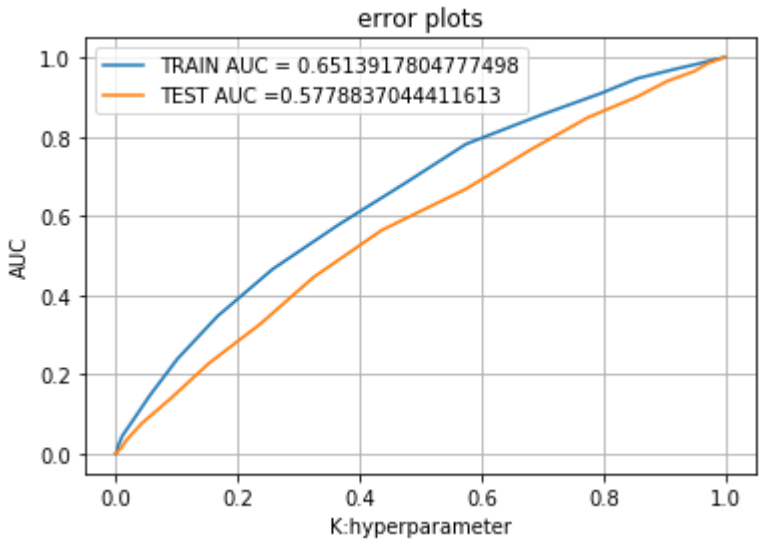
OBSERVATIONS: From plotting k values we choose our best k to be 71.

In [ ]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

neigh = KNeighborsClassifier(n_jobs=-1)
parameters = {'n_neighbors':sp_randint(50, 100)}
clf = RandomizedSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(final_train_bow, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_n_neighbors'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_n_neighbors']

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

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

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("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head(5)
```

In [71]:

```
best_k=71
from sklearn.metrics import roc_curve,auc

neigh=KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(final_train_bow,y_train)

y_train_pred=batch_predict(neigh,final_train_bow)
y_test_pred=batch_predict(neigh,final_test_bow)

train_fpr,train_tpr,tr_threshold=roc_curve(y_train,y_train_pred)
test_fpr,test_tpr,test_threshold=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('K:hyperparameter')
plt.ylabel('AUC')
plt.title("error plots")
plt.grid()
plt.show()
```



OBSERVATIONS: For k=71 we got auc of 65.13% for train data. FOr k=71 we got auc of 57.78% for test data.

In [72]:

```
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 [73]:

```
#printing confusion matrix
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,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)))
```

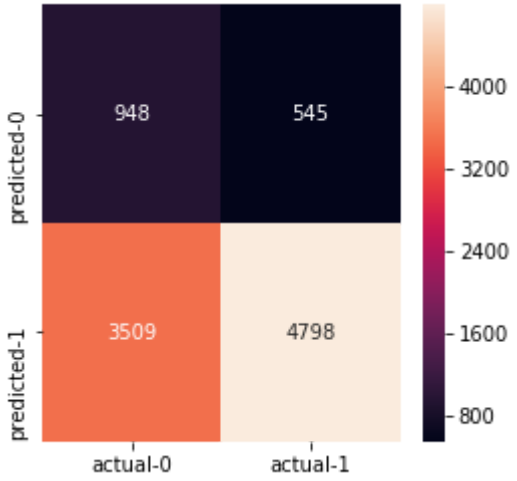
```
====================================================================================
==========================
the maximum value of tpr*(1-fpr) 0.36674530498290203 for threshold 0.845
TRAIN CONFUSION MATRIX
[[ 948  545]
 [3509 4798]]
test confusion matrix
[[ 515  399]
 [2215 2871]]
```

In [74]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[948,545],[3509,4798]]

train=pd.DataFrame(array,index=['predicted-0','predicted-1'],columns=['actual-0','actua
l-1'])
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[74]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18db349a908>
```

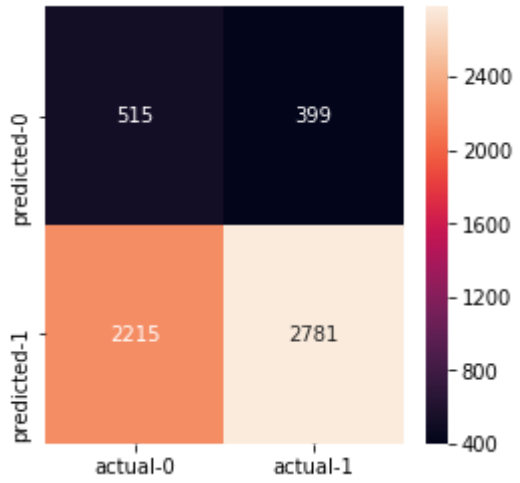


In [75]:

```
#printing heatmap for test confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

testarray=[[515,399],[2215,2781]]

test=pd.DataFrame(testarray,index=['predicted-0','predicted-1'],columns=['actual-0','ac
tual-1'])
plt.figure(figsize=(4,4))
sn.heatmap(test,annot=True,fmt='d')
```

Out[75]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18db342ab00>
```



2.4.2 Applying KNN brute force on TFIDF, SET 2

In [76]:

```
#tfidf encoding  of text
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['essay'].values)
train_tfidf=vectorizer.transform(X_train['essay'].values)
cv_tfidf=vectorizer.transform(X_cv['essay'].values)
test_tfidf=vectorizer.transform(X_test['essay'].values)
print("Shape of matrix after one hot encodig ",train_tfidf.shape)
print("Shape of matrix after one hot encodig ",cv_tfidf.shape)
print("Shape of matrix after one hot encodig ",test_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (9800, 6247)
Shape of matrix after one hot encodig  (4200, 6247)
Shape of matrix after one hot encodig  (6000, 6247)
```

In [77]:

```
#tfidf encoding of title
vectorizer=TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['project_title'].values)
title_train_tfidf=vectorizer.transform(X_train['project_title'].values)
title_cv_tfidf=vectorizer.transform(X_cv['project_title'].values)
title_test_tfidf=vectorizer.transform(X_test['project_title'].values)

print("Shape of matrix after one hot encodig ",title_train_tfidf.shape)
print("Shape of matrix after one hot encodig ",title_cv_tfidf.shape)
print("Shape of matrix after one hot encodig ",title_test_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (9800, 650)
Shape of matrix after one hot encodig  (4200, 650)
Shape of matrix after one hot encodig  (6000, 650)
```

In [78]:

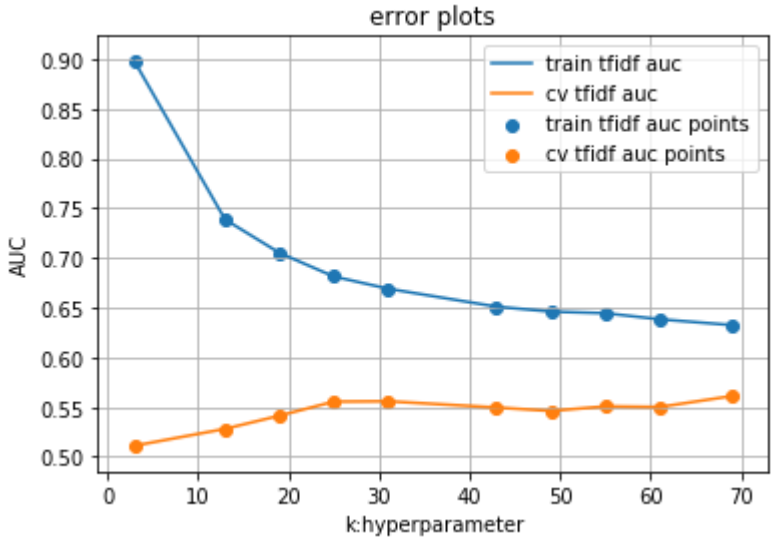
```
#creating final data matrix
#creating final data matrix
from scipy.sparse import hstack
final_train_tfidf=hstack((X_train_state_encoded,X_train_prefix_encoded,X_train_grade_en
coded,X_train_categories_encoded,X_train_subcategories_encoded,X_train_price_norm,X_tra
in_projects_norm,X_train_quantity_norm,train_tfidf,title_train_tfidf)).tocsr()
final_cv_tfidf=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_c
ategories_encoded,X_cv_subcategories_encoded,X_cv_price_norm,X_cv_projects_norm,X_cv_qu
antity_norm,cv_tfidf,title_cv_tfidf)).tocsr()
final_test_tfidf=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encode
d,X_test_categories_encoded,X_test_subcategories_encoded,X_test_price_norm,X_test_proje
cts_norm,X_test_quantity_norm,test_tfidf,title_test_tfidf)).tocsr()
print(final_train_tfidf.shape,y_train.shape)
print(final_cv_tfidf.shape,y_cv.shape)
print(final_test_tfidf.shape,y_test.shape)
```

```
(9800, 6997) (9800,)
(4200, 6997) (4200,)
(6000, 6997) (6000,)
```

In [80]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
tfidf_train_auc=[]
tfidf_cv_auc=[]
k=[3,13,19,25,31,43,49,55,61,69]
for i in tqdm(k):
    neigh=KNeighborsClassifier(n_neighbors=i)
    neigh.fit(final_train_tfidf,y_train)

    y_train_pred=batch_predict(neigh,final_train_tfidf)
    y_cv_pred=batch_predict(neigh,final_cv_tfidf)

    tfidf_train_auc.append(roc_auc_score(y_train,y_train_pred))
    tfidf_cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

plt.plot(k,tfidf_train_auc,label='train tfidf auc')
plt.plot(k,tfidf_cv_auc,label='cv tfidf auc')

plt.scatter(k,tfidf_train_auc,label='train tfidf auc points')
plt.scatter(k,tfidf_cv_auc,label='cv tfidf auc points')
plt.legend()
plt.xlabel("k:hyperparameter")
plt.ylabel('AUC')
plt.title('error plots')
plt.grid()
plt.show()
```

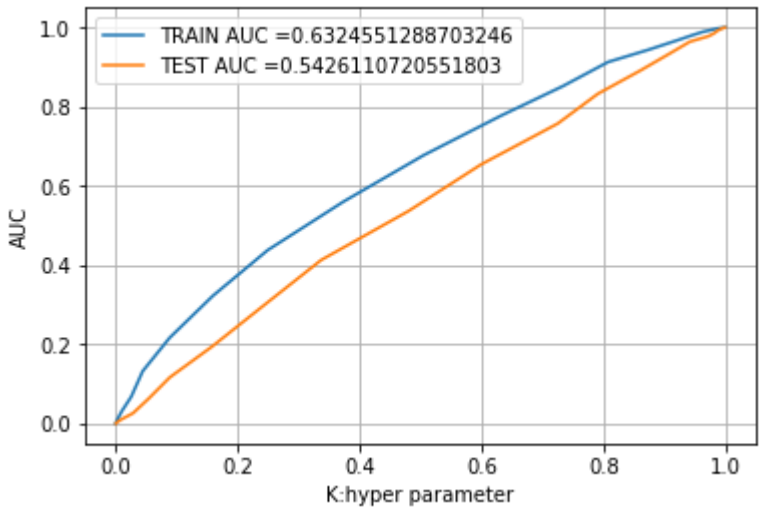
```
  0%|
| 0/10 [00:00<?, ?it/s]
 10%|
| 1/10 [00:24<03:41, 24.60s/it]
 20%|
| 2/10 [00:50<03:19, 24.96s/it]
 30%|
| 3/10 [01:16<02:57, 25.35s/it]
 40%|
| 4/10 [01:43<02:34, 25.71s/it]
 50%|
| 5/10 [02:09<02:08, 25.77s/it]
 60%|
| 6/10 [02:37<01:45, 26.44s/it]
 70%|
| 7/10 [03:03<01:19, 26.35s/it]
 80%|
| 8/10 [03:29<00:52, 26.30s/it]
 90%|
          | 9/10 [03:55<00:26, 26.19s/it]
100%|
          | 10/10 [04:21<00:00, 26.12s/it]
```



OBSERVATIONS: By using tfidf model and plotting error plots for 10 values of k we choose our best k to be 69.

In [81]:

```
#plotting auc curves
from sklearn.metrics import roc_curve, auc

best_k=69
neigh=KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(final_train_tfidf,y_train)

y_train_pred=batch_predict(neigh,final_train_tfidf)
y_test_pred=batch_predict(neigh,final_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('K:hyper parameter')
plt.ylabel('AUC')
plt.grid()
plt.show()
```



OBSERVATIONS: By using tfidf model and best k to be 69 we got a train auc as 63.24% and test auc as 54.26%

In [82]:

```
#printing confusion matrix
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,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)))
```

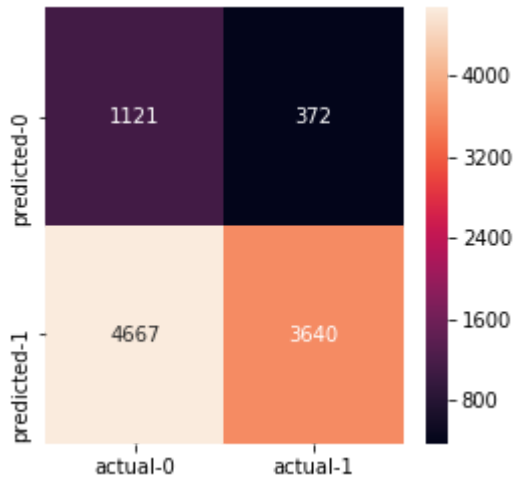
```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.3512370356233266 for threshold 0.859
TRAIN CONFUSION MATRIX
[[1121  372]
 [4667 3640]]
test confusion matrix
[[ 606  308]
 [2984 2102]]
```

In [83]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[1121,372],[4667,3640]]

train=pd.DataFrame(array,index=['predicted-0','predicted-1'],columns=['actual-0','actua
l-1'])
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[83]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18db4280160>
```

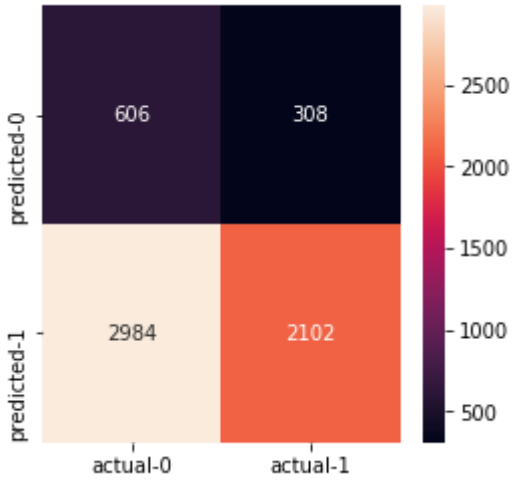


In [84]:

```
#printing heatmap for test confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

testarray=[[606,308],[2984,2102]]

test=pd.DataFrame(testarray,index=['predicted-0','predicted-1'],columns=['actual-0','ac
tual-1'])
plt.figure(figsize=(4,4))
sn.heatmap(test,annot=True,fmt='d')
```

Out[84]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18db353e898>
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [85]:

```
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [86]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_train.append(vector)

print(len(avg_w2v_train))
print(len(avg_w2v_train[0]))
print(avg_w2v_train[0])
```

```
  0%|
| 0/9800 [00:00<?, ?it/s]
  3%|
| 263/9800 [00:00<00:03, 2529.46it/s]
  5%|
| 485/9800 [00:00<00:03, 2396.68it/s]
  8%|
| 763/9800 [00:00<00:03, 2473.58it/s]
 11%|
| 1058/9800 [00:00<00:03, 2572.51it/s]
 14%|
| 1332/9800 [00:00<00:03, 2591.01it/s]
 17%|
| 1643/9800 [00:00<00:03, 2699.33it/s]
 20%|
| 1928/9800 [00:00<00:02, 2711.72it/s]
 23%|
| 2218/9800 [00:00<00:02, 2734.52it/s]
 25%|
| 2486/9800 [00:00<00:02, 2685.33it/s]
 28%|
| 2778/9800 [00:01<00:02, 2721.10it/s]
 31%|
| 3044/9800 [00:01<00:02, 2261.60it/s]
 34%|
| 3329/9800 [00:01<00:02, 2386.85it/s]
 37%|
| 3606/9800 [00:01<00:02, 2463.86it/s]
 40%|
| 3890/9800 [00:01<00:02, 2538.48it/s]
 43%|
| 4182/9800 [00:01<00:02, 2613.85it/s]
 46%|
| 4487/9800 [00:01<00:01, 2702.10it/s]
 49%|
| 4761/9800 [00:01<00:01, 2620.24it/s]
 52%|
| 5066/9800 [00:01<00:01, 2706.95it/s]
 54%|
| 5340/9800 [00:02<00:01, 2623.25it/s]
 57%|
| 5605/9800 [00:02<00:01, 2570.26it/s]
 60%|
| 5890/9800 [00:02<00:01, 2619.18it/s]
 63%|
| 6154/9800 [00:02<00:01, 2534.90it/s]
 65%|
| 6417/9800 [00:02<00:01, 2533.30it/s]
 68%|
| 6679/9800 [00:02<00:01, 2529.21it/s]
 71%|
| 6980/9800 [00:02<00:01, 2628.89it/s]
 74%|
| 7280/9800 [00:02<00:00, 2700.89it/s]
 77%|
| 7552/9800 [00:02<00:00, 2583.43it/s]
 80%|
| 7830/9800 [00:03<00:00, 2609.89it/s]
 83%|
| 8108/9800 [00:03<00:00, 2628.61it/s]
 85%|
```

```
| 8373/9800 [00:03<00:00, 2573.90it/s]
 88%|████████████████████████████████████████████████████████████████▉
| 8632/9800 [00:03<00:00, 2489.56it/s]
 91%|████████████████████████████████████████████████████████████████
|       | 8917/9800 [00:03<00:00, 2560.14it/s]
 94%|████████████████████████████████████████████████████████████████
██▏     | 9205/9800 [00:03<00:00, 2619.59it/s]
 97%|████████████████████████████████████████████████████████████████
████▊   | 9469/9800 [00:03<00:00, 2594.99it/s]
100%|████████████████████████████████████████████████████████████████
███████ | 9766/9800 [00:03<00:00, 2668.28it/s]
100%|████████████████████████████████████████████████████████████████
███████| 9800/9800 [00:03<00:00, 2598.74it/s]
```

```
9800
300
[ 3.13226607e-02 -5.48201517e-02  3.71754310e-02 -1.59735749e-01
  1.99387024e-02 -5.94057024e-02 -2.77799233e+00  8.23461679e-02
  1.00625226e-02  1.44799924e-01 -5.43037143e-02  1.76421548e-02
  2.07811190e-02 -2.46976221e-01 -4.42968393e-02 -1.20448223e-01
  8.07539286e-04  1.36040536e-02  8.39286036e-02 -3.52078917e-02
 -3.39257762e-02  2.42672857e-02 -5.32155000e-02 -2.90530845e-02
  2.13343833e-03 -3.41664833e-02  1.75127643e-01 -8.08869964e-02
 -8.66474905e-02 -9.68939524e-02 -2.91740726e-01 -1.18508179e-01
 -1.00567419e-01  8.78193512e-02 -6.26992932e-02 -8.40579405e-02
 -5.87448345e-02  1.77535714e-02  3.53999585e-02  2.53908238e-02
  5.76291226e-03  8.88359333e-02  3.97734881e-02 -1.20784448e-01
  9.43547488e-02 -5.61288607e-02  8.62582500e-02 -2.81716461e-02
 -8.06280357e-03 -1.74838382e-01 -4.99149524e-03  6.07143190e-02
 -1.29598571e-02  4.32688940e-02  1.18194476e-02 -8.76751607e-02
 -5.57009524e-03 -2.03979058e-02 -9.16633667e-02  6.98412905e-02
 -1.04404848e-01  2.56772024e-02  1.00418607e-02 -6.29297357e-02
 -1.12030825e-01  5.61919167e-02  2.04371894e-02 -6.95057333e-02
  1.81929067e-01 -1.43444957e-01 -3.22277274e-02 -5.66177262e-02
 -2.72510471e-02 -6.98088702e-02 -3.41120762e-02 -2.39624799e-01
 -4.94336310e-03 -5.75782619e-02  3.67425298e-02  1.82151667e-02
  2.05361619e-02 -2.28009333e-01  8.60961512e-02 -8.36454726e-02
 -1.61137321e-01  5.45540635e-02  1.83138640e-01 -4.07666950e-02
  1.50084494e-01  1.26157262e-02  1.39280869e-02  3.45297429e-02
  6.58588095e-03  3.42278440e-02 -1.27751383e-02 -1.27354761e-01
 -2.19050185e+00  5.60336083e-02  3.88152500e-02  1.85078699e-01
 -1.14775405e-01 -2.79842976e-02  1.64769966e-01 -2.52402917e-02
  1.03971845e-01 -4.93317595e-02 -1.66012024e-02 -1.74356440e-01
  4.61837631e-02  6.42568095e-02 -9.74762190e-02  7.05118214e-04
  7.76077024e-02  2.93270690e-01  1.07189580e-01 -2.66392631e-02
 -3.96891227e-01  3.12369133e-02  1.35952500e-01 -3.85690369e-02
 -1.69252500e-03  2.74722262e-02  2.92851090e-02 -1.56504147e-01
  1.75334862e-01  7.76664850e-02  4.98670605e-02  4.45271774e-02
 -2.36163012e-02  1.61280430e-01  1.21396296e-01 -2.56696619e-02
  1.85890833e-03 -1.12075610e-01 -5.44592821e-02 -7.19105190e-02
  1.39096393e-02 -2.04607250e-02  4.61684429e-02  1.56428389e-01
  8.09334655e-02 -3.99770393e-02  8.51520679e-02 -1.41342500e-02
 -7.49462507e-02  1.15068920e-01  6.02966643e-02 -6.88279262e-02
  5.88665119e-02 -7.70265500e-02 -2.80731369e-02 -7.55715333e-02
  6.00616310e-02  5.38038202e-02  2.05781190e-02  1.22989944e-01
  5.30735476e-02 -4.00385357e-02 -6.87305143e-02  2.40688929e-03
  3.59212262e-02 -1.14100036e-02  5.07839048e-02 -7.13203190e-02
 -4.54160667e-02  2.37199571e-02 -1.26744489e-01  1.20789460e-01
 -2.93261274e-02 -2.05486012e-02 -6.57253083e-02 -3.46318048e-02
 -4.04989286e-02 -1.15333250e-01  4.87699393e-02 -3.88798702e-02
  3.48965381e-02  9.70784548e-02 -1.88058063e-01  2.30808083e-02
  3.13839876e-02  2.34441857e-01  8.18070476e-02 -1.49989179e-02
 -5.87425476e-02  3.86129762e-05 -5.71814842e-02  1.96571226e-02
 -1.59404048e-02 -2.79301560e-02 -7.60652707e-02 -6.37027774e-02
 -2.29063357e-01  4.26371429e-03  4.30534012e-02 -4.07523464e-02
 -1.16826793e-01  1.21928008e-01 -5.08611556e-02  3.58137321e-02
  2.59705179e-01 -1.01228512e-02 -9.14888163e-02  4.34184821e-02
 -9.19269869e-02  5.46607774e-02  1.07584411e-01 -1.95468117e-01
  8.70556498e-02  1.28944643e-02 -2.95773595e-02  2.81142024e-03
  3.80582905e-02 -1.25264864e-01 -1.20813998e-01  6.92058058e-02
 -1.16573690e-01 -3.73906217e-02 -2.22965917e-02 -1.04909274e-02
 -1.27238881e-01 -4.18567536e-02 -1.79543462e-01 -5.92927512e-02
 -1.74634119e+00 -4.11334702e-02 -6.50599133e-02  5.32518929e-02
 -6.29677345e-02 -1.41300724e-01  5.75828214e-02 -8.95870000e-02
 -1.68831226e-01 -4.70325143e-02 -1.12209036e-01 -8.26619405e-02
```

```
 9.45442429e-02 -7.12164762e-02  6.55876286e-02  2.17046124e-01
-3.38935583e-02  2.45048695e-02 -1.00648761e-01 -4.17641325e-02
-5.25807310e-02  2.50742321e-02  4.80930107e-02 -5.33492762e-02
-1.09854988e-02 -4.44404940e-02 -3.34581155e-03  1.07964376e-01
 1.08321570e-01 -1.36483715e-01  9.90867633e-02  1.25630082e-01
 6.88205254e-02 -5.82053571e-02  1.86476250e-01 -1.92672487e-01
 3.85947976e-02  5.06778690e-02 -3.36646155e-02  6.38012267e-02
 4.69852333e-02 -1.91335905e-02 -7.32826282e-02 -1.40663571e-03
 4.31002643e-02  9.29893226e-02 -1.61963095e-03  3.53602060e-02
-7.38128286e-02  1.08706620e-01 -5.50468583e-02  1.11310098e-01
 4.44741226e-02 -1.20040881e-02 -6.67131333e-02  1.08346139e-01
 8.24933587e-02  3.31892857e-04 -8.78588536e-02  3.13450333e-02
-4.95507619e-03  1.64336662e-01  6.41110714e-02  1.74457100e-02
 1.01204548e-01  3.36845357e-02  7.46442857e-03 -2.21982202e-02
-9.68419500e-02 -7.42114250e-02  7.08839821e-02  1.28998702e-02
-2.34553440e-02  2.23613560e-01  1.35062256e-01  5.06844238e-02]
```

In [87]:

```
avg_w2v_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_cv.append(vector)

print(len(avg_w2v_cv))
print(len(avg_w2v_cv[0]))
print(avg_w2v_cv[0])
```

```
  0%|
| 0/4200 [00:00<?, ?it/s]
  6%|▌
| 263/4200 [00:00<00:01, 2529.49it/s]
 12%|▉
| 491/4200 [00:00<00:01, 2418.12it/s]
 17%|▏
| 725/4200 [00:00<00:01, 2365.31it/s]
 23%|▏
| 962/4200 [00:00<00:01, 2338.87it/s]
 29%|▎
| 1233/4200 [00:00<00:01, 2413.19it/s]
 36%|▍
| 1507/4200 [00:00<00:01, 2475.66it/s]
 42%|▍
| 1746/4200 [00:00<00:01, 2419.89it/s]
 49%|▍
| 2070/4200 [00:00<00:00, 2593.74it/s]
 56%|▌
| 2351/4200 [00:00<00:00, 2625.46it/s]
 62%|▋
| 2624/4200 [00:01<00:00, 2625.53it/s]
 69%|▋
| 2882/4200 [00:01<00:00, 2580.57it/s]
 76%|▊
| 3177/4200 [00:01<00:00, 2652.57it/s]
 82%|▊
| 3441/4200 [00:01<00:00, 2556.66it/s]
 89%|▉
| 3731/4200 [00:01<00:00, 2622.21it/s]
 95%|▉
▌    | 3994/4200 [00:01<00:00, 2533.74it/s]
100%|▉
▉| 4200/4200 [00:01<00:00, 2536.87it/s]
```

```
4200
300
[ 1.22430250e-03  9.39480275e-02  6.00137675e-02 -1.02484221e-01
 -9.80715750e-02 -5.86601413e-02 -3.05463713e+00  1.52713128e-01
  9.05223025e-02  1.51056053e-01 -6.65017750e-02  1.87820875e-02
  1.03322375e-02 -1.36477475e-01  4.15815487e-02  6.21131625e-03
  5.12600625e-02 -1.28125563e-01  2.81779912e-02 -5.58456538e-02
 -4.16117050e-02 -3.13034537e-02 -4.03840050e-02 -1.82894250e-02
 -1.27536450e-02 -3.33210250e-03  1.97241134e-01 -3.61637200e-02
 -1.51508128e-01 -1.08259535e-01 -2.69567775e-01 -1.35954382e-01
  2.82051946e-02  7.11319613e-02  3.34460750e-02 -1.07450200e-01
 -9.12472094e-02 -1.02959228e-01 -3.35025000e-02 -6.23700750e-03
 -6.26996500e-02  1.85191250e-02  1.16111641e-01 -1.39023837e-01
  3.16367358e-02 -8.53047500e-02  2.12257500e-02 -1.27580200e-01
  1.53122737e-02 -1.39264395e-01  1.17114750e-02 -7.04225750e-02
 -2.52140625e-02  3.06148750e-02  1.14695427e-01 -8.47156838e-02
 -3.04247212e-02  5.54788238e-02 -1.33413790e-01  3.77132175e-02
 -3.00575525e-02 -5.20882000e-03  5.03186188e-02 -1.43296555e-01
 -1.00009920e-01  8.67569000e-02  4.90623975e-02 -1.40017804e-01
  1.82839000e-01 -7.34921925e-02 -5.09250000e-02 -3.76648875e-02
 -9.09034903e-02 -1.06476484e-01 -7.15566375e-02 -1.47084953e-01
 -3.99518512e-02 -1.69406250e-03  4.87153600e-02 -3.90341760e-02
 -1.47891463e-02 -2.77861100e-01 -3.69730750e-02 -1.05508040e-02
 -1.41284405e-01  9.37938675e-02  1.37230938e-01 -5.23492162e-02
  1.90838877e-01  5.50214500e-02  3.99599650e-02  4.22914000e-03
  2.10295625e-02  8.41513050e-02  5.25815150e-02 -1.20732764e-01
 -2.21279637e+00  1.76104772e-01  1.04359137e-01  1.44792228e-01
 -1.93084077e-01 -1.34094125e-03  1.59453700e-01 -8.17114450e-02
  1.71370407e-01  6.68245000e-02 -3.28593750e-02 -1.14483437e-01
  9.40022329e-02  3.01272425e-02 -5.30276250e-02 -9.39775500e-03
 -9.80906875e-03  2.69721112e-01  8.81059000e-02 -9.29372250e-02
 -4.29669700e-01 -1.96250912e-02  5.24026425e-02 -5.10531516e-02
  8.57364200e-02  6.96373925e-02 -1.77569405e-02 -9.46207225e-02
  1.87804513e-02  8.13023263e-02  8.54548575e-02 -6.89748750e-04
 -5.77440375e-03  2.71656588e-02  1.21952555e-01  2.99552250e-02
  3.69807750e-03 -2.00638188e-02  6.38720000e-03 -9.67322463e-02
  6.44376525e-02 -3.48341100e-02  1.09608275e-02  1.45765801e-01
  5.39702313e-02  5.83151875e-02  2.53320875e-03 -1.35364000e-02
 -1.08760617e-01  1.65108885e-01  4.74391575e-02 -9.81684750e-03
  1.54462963e-01 -2.79331381e-02 -3.35766217e-02 -4.30468300e-02
  9.01049500e-02 -9.72305125e-02 -6.71538125e-02  1.79110011e-01
  1.04082720e-01 -2.57190750e-03 -1.88839987e-02 -4.39861175e-02
  6.84596312e-02  2.02225000e-02  8.47792750e-03 -1.01761409e-01
 -3.05675313e-02  8.89676463e-02 -1.12480345e-01  5.60348386e-02
  1.01492111e-01 -6.75450375e-02 -1.38726971e-01  3.46536075e-02
 -3.46957238e-02 -1.62132835e-01  2.33684037e-02 -1.17469944e-01
 -5.82563050e-02  8.34400987e-02 -2.32586850e-01 -2.33023887e-02
  5.17754860e-02  2.89213821e-01  7.75243625e-02  4.83742475e-02
 -8.89356875e-02 -2.11594425e-02  2.78775437e-02  6.79009125e-02
 -3.96221375e-02 -1.29865275e-02 -9.59667896e-02 -3.05761000e-02
 -1.92231244e-01  1.34380375e-02 -9.10253750e-03 -1.72099682e-01
 -6.07001500e-03  9.75264437e-02  4.49134375e-02  1.14608738e-01
  1.91415074e-01  6.29272625e-03  3.34904425e-02  4.41033250e-02
 -1.30691767e-01  5.89883750e-03  8.88201550e-02 -3.15345388e-02
  8.03298425e-02 -1.22249150e-02 -1.10777120e-01 -5.67869475e-02
  2.30784538e-02 -1.74863591e-01 -1.34654280e-01  1.31753838e-02
 -2.53715750e-02 -5.98089663e-02  2.00554550e-03 -2.16592750e-03
 -1.86666687e-01 -1.25672075e-01 -1.72721188e-01 -6.19042888e-02
 -1.57482301e+00  5.43930500e-02  1.35482388e-02  4.53950000e-02
 -7.48338713e-02 -6.55338375e-02  3.47320625e-02 -1.59803450e-02
 -1.10142085e-01 -9.77295075e-02 -6.49295438e-02 -2.47913625e-02
```

```
 1.09020362e-01 -7.11046425e-02  4.06347950e-02  1.32084572e-01
-1.18642875e-02  3.76360650e-02 -1.59324913e-01 -1.66173615e-01
-1.74058975e-02  5.88872313e-02 -2.37593975e-02 -1.06327357e-01
 6.21134125e-03 -1.10382262e-01  3.66019359e-02  6.51446662e-02
 4.74146525e-02 -1.25100016e-01  1.59067552e-01  9.64307625e-03
 3.68243125e-02 -1.05185050e-01  1.58904400e-01 -1.59233513e-01
 2.91483563e-02 -1.78149250e-02 -4.55703250e-03  3.25261625e-02
 2.79222387e-02 -3.95685350e-02 -9.13929825e-02  2.35592163e-02
 1.29799925e-01  7.71673625e-02 -6.77723712e-02 -1.78125000e-02
-1.34178994e-01  1.00918979e-01 -2.96912425e-02  1.24618664e-01
 2.94751475e-02  4.08787938e-02 -2.01702500e-03  4.40543855e-02
-1.92744375e-02 -8.86272837e-02 -6.47904605e-02 -2.18912500e-04
 8.34750000e-05  1.51091367e-01 -7.22685775e-02  3.32277500e-03
 7.81656500e-02 -4.28477000e-02  8.84117875e-02 -8.19893625e-02
-7.22229143e-02 -9.76124775e-02 -2.77479125e-02 -1.16841452e-01
 8.45205625e-03  2.22733569e-01  1.64497832e-01 -2.70796300e-02]
```

In [88]:

```
avg_w2v_test =[];
for sentence in tqdm(X_test['essay'].values):
    vector=np.zeros(300)
    cnt_words = 0;
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words +=1
    if cnt_words != 0:
            vector /= cnt_words
    avg_w2v_test.append(vector)
print(len(avg_w2v_test))
```

```
  0%|
| 0/6000 [00:00<?, ?it/s]
  4%|
| 258/6000 [00:00<00:02, 2481.42it/s]
  8%|
| 499/6000 [00:00<00:02, 2429.96it/s]
 12%|
| 725/6000 [00:00<00:02, 2346.95it/s]
 16%|
| 979/6000 [00:00<00:02, 2374.87it/s]
 21%|
| 1274/6000 [00:00<00:01, 2496.91it/s]
 26%|
| 1532/6000 [00:00<00:01, 2492.23it/s]
 31%|
| 1848/6000 [00:00<00:01, 2634.58it/s]
 36%|
| 2150/6000 [00:00<00:01, 2710.15it/s]
 41%|
| 2443/6000 [00:00<00:01, 2741.65it/s]
 45%|
| 2719/6000 [00:01<00:01, 2714.92it/s]
 50%|
| 2985/6000 [00:01<00:01, 2665.82it/s]
 55%|
| 3280/6000 [00:01<00:01, 2715.14it/s]
 59%|
| 3550/6000 [00:01<00:00, 2647.03it/s]
 64%|
| 3852/6000 [00:01<00:00, 2719.30it/s]
 69%|
| 4127/6000 [00:01<00:00, 2696.50it/s]
 74%|
| 4421/6000 [00:01<00:00, 2734.54it/s]
 78%|
| 4695/6000 [00:01<00:00, 2641.45it/s]
 83%|
| 4960/6000 [00:01<00:00, 2612.94it/s]
 87%|
| 5246/6000 [00:01<00:00, 2652.89it/s]
 92%|
       | 5512/6000 [00:02<00:00, 2593.00it/s]
 97%|
    | 5797/6000 [00:02<00:00, 2635.75it/s]
100%|
       | 6000/6000 [00:02<00:00, 2613.89it/s]

6000
```

In [89]:

```
title_train_avgw2v=[]
for sentence in tqdm(X_train['project_title'].values):
    vector=np.zeros(300)
    cnt_words=0;
    for word in sentence.split():
        if word in glove_words:
            vector+=model[word]
            cnt_words+=1
    if cnt_words !=0:
        vector/=cnt_words
    title_train_avgw2v.append(vector)
print(len(title_train_avgw2v))
```

```
  0%|
| 0/9800 [00:00<?, ?it/s]
100%|
| 9800/9800 [00:00<00:00, 94259.21it/s]
```

```
9800
```

In [90]:

```
title_cv_avgw2v=[]
for sentence in tqdm(X_cv['project_title'].values):
    vector=np.zeros(300)
    cnt_words=0;
    for word in sentence.split():
        if word in glove_words:
            vector+=model[word]
            cnt_words+=1
    if cnt_words !=0:
        vector/=cnt_words
    title_cv_avgw2v.append(vector)
print(len(title_cv_avgw2v))
```

```
  0%|
| 0/4200 [00:00<?, ?it/s]
100%|
| 4200/4200 [00:00<00:00, 80789.90it/s]
```

```
4200
```

In [91]:

```
title_test_avgw2v=[]
for sentence in tqdm(X_test['project_title'].values):
    vector=np.zeros(300)
    cnt_words=0;
    for word in sentence.split():
        if word in glove_words:
            vector+=model[word]
            cnt_words+=1
    if cnt_words !=0:
        vector/=cnt_words
    title_test_avgw2v.append(vector)
print(len(title_test_avgw2v))
```

```
  0%|
| 0/6000 [00:00<?, ?it/s]
100%|
| 6000/6000 [00:00<00:00, 115409.91it/s]
```

6000

In [92]:

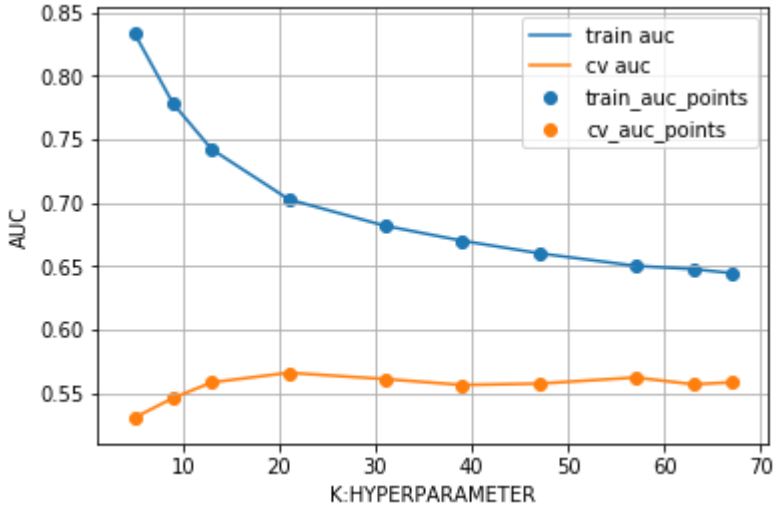
```
from scipy.sparse import hstack
final_train_avgw2v=hstack((X_train_state_encoded,X_train_prefix_encoded,X_train_grade_e
ncoded,X_train_categories_encoded,X_train_subcategories_encoded,X_train_price_norm,X_tr
ain_projects_norm,X_train_quantity_norm,avg_w2v_train,title_train_avgw2v)).tocsr()
final_cv_avgw2v=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_
categories_encoded,X_cv_subcategories_encoded,X_cv_price_norm,X_cv_projects_norm,X_cv_q
uantity_norm,avg_w2v_cv,title_cv_avgw2v)).tocsr()
final_test_avgw2v=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encod
ed,X_test_categories_encoded,X_test_subcategories_encoded,X_test_price_norm,X_test_proj
ects_norm,X_test_quantity_norm,avg_w2v_test,title_test_avgw2v)).tocsr()
print(final_train_avgw2v.shape,y_train.shape)
print(final_cv_avgw2v.shape,y_cv.shape)
print(final_test_avgw2v.shape,y_test.shape)
```

```
(9800, 700) (9800,)
(4200, 700) (4200,)
(6000, 700) (6000,)
```

In [93]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.neighbors import KNeighborsClassifier

avg_w2v_train=[]
avg_w2v_cv=[]
k=[5,9,13,21,31,39,47,57,63,67]
for i in tqdm(k):
    neigh=KNeighborsClassifier(n_neighbors=i)
    neigh.fit(final_train_avgw2v,y_train)

    y_train_pred=batch_predict(neigh,final_train_avgw2v)
    y_cv_pred=batch_predict(neigh,final_cv_avgw2v)

    avg_w2v_train.append(roc_auc_score(y_train,y_train_pred))
    avg_w2v_cv.append(roc_auc_score(y_cv,y_cv_pred))

plt.plot(k,avg_w2v_train,label='train auc')
plt.plot(k,avg_w2v_cv,label='cv auc')

plt.scatter(k,avg_w2v_train,label='train_auc_points')
plt.scatter(k,avg_w2v_cv,label="cv_auc_points")

plt.legend()
plt.xlabel('K:HYPERPARAMETER')
plt.ylabel('AUC')
plt.grid()
plt.show()
```

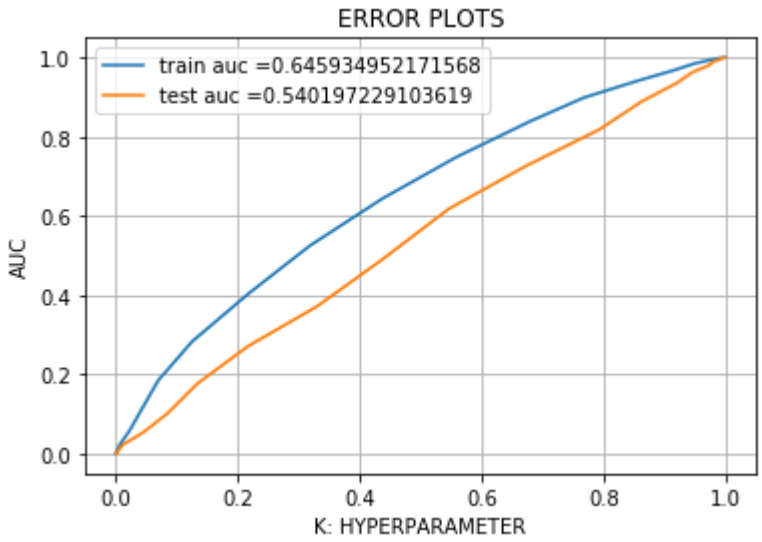
```
  0%|
| 0/10 [00:00<?, ?it/s]
 10%|
| 1/10 [01:57<17:39, 117.70s/it]
 20%|
| 2/10 [03:55<15:41, 117.67s/it]
 30%|
| 3/10 [05:52<13:43, 117.59s/it]
 40%|
| 4/10 [07:51<11:46, 117.81s/it]
 50%|
| 5/10 [09:49<09:49, 117.93s/it]
 60%|
| 6/10 [11:46<07:51, 117.77s/it]
 70%|
| 7/10 [13:43<05:52, 117.60s/it]
 80%|
| 8/10 [15:41<03:55, 117.58s/it]
 90%|
          | 9/10 [17:39<01:57, 117.65s/it]
100%|
          | 10/10 [19:36<00:00, 117.60s/it]
```



OBSERVATIONS: By using avg w2v model and plotting for 10 different values of k we choose best k to be 65.

In [94]:

```
best_k=65
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve,auc

neigh=KNeighborsClassifier(n_neighbors=best_k,n_jobs=-1)
neigh.fit(final_train_avgw2v,y_train)

y_train_pred=batch_predict(neigh,final_train_avgw2v)
y_test_pred=batch_predict(neigh,final_test_avgw2v)

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.title('ERROR PLOTS')
plt.xlabel('K: HYPERPARAMETER')
plt.ylabel('AUC')
plt.grid()
plt.show()
```



OBSERVATIONS: By using avg w2v model and k=65 we got train auc as 64.9% and test auc as 54.01%.

In [95]:

```
#printing the confusion matrix
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,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)))
```

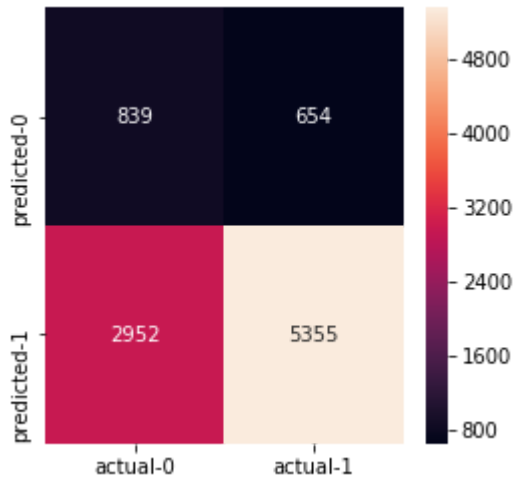
```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.36225752681890716 for threshold 0.845
TRAIN CONFUSION MATRIX
[[ 839  654]
 [2952 5355]]
test confusion matrix
[[ 415  499]
 [1944 3142]]
```

In [96]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[839,654],[2952,5355]]

train=pd.DataFrame(array,index=['predicted-0','predicted-1'],columns=['actual-0','actua
l-1'])
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[96]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18e3208d9b0>
```

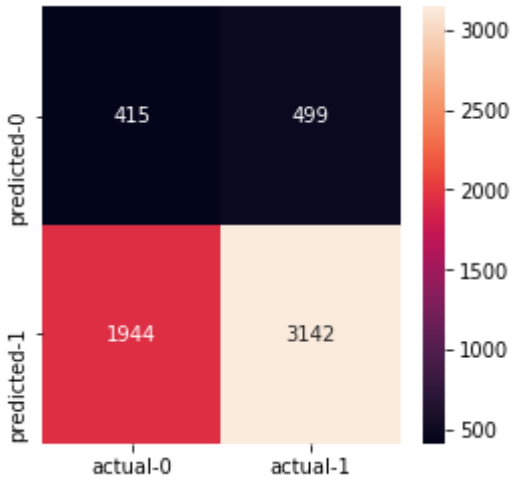


In [97]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

testarray=[[415,499],[1944,3142]]

test=pd.DataFrame(testarray,index=['predicted-0','predicted-1'],columns=['actual-0','ac
tual-1'])
plt.figure(figsize=(4,4))
sn.heatmap(test,annot=True,fmt='d')
```

Out[97]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18db4270c88>
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [89]:

```
# Please write all the code with proper documentation
```

In [98]:

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

In [99]:

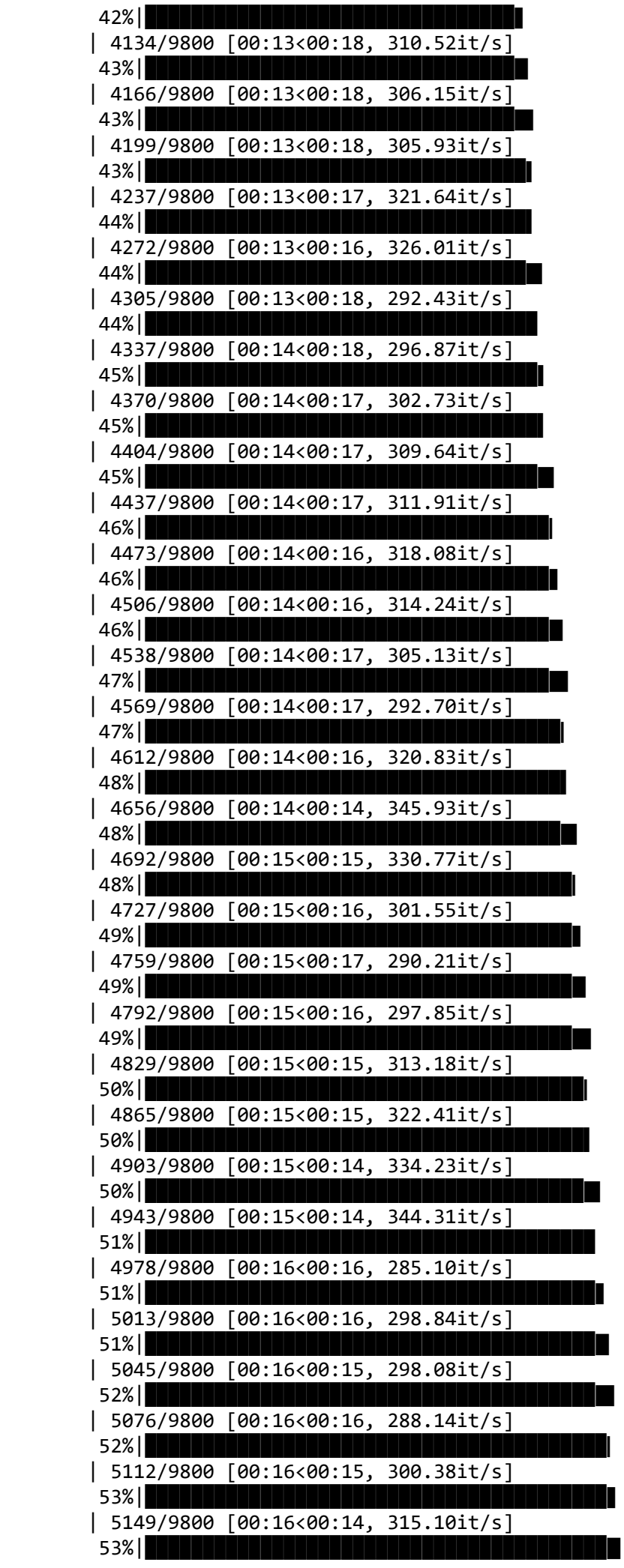
```
# compute average word2vec for each review.
tfidf_w2v_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with 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
    tfidf_w2v_train.append(vector)

print(len(tfidf_w2v_train))
print(len(tfidf_w2v_train[0]))
```

```
  0%|
| 0/9800 [00:00<?, ?it/s]
  0%||
| 22/9800 [00:00<00:46, 211.59it/s]
  0%|▏
| 48/9800 [00:00<00:43, 221.83it/s]
  1%|▏
| 68/9800 [00:00<00:45, 212.08it/s]
  1%|▏
| 93/9800 [00:00<00:44, 219.86it/s]
  1%|▏
| 119/9800 [00:00<00:42, 228.12it/s]
  2%|▏
| 155/9800 [00:00<00:38, 252.00it/s]
  2%|▏
| 191/9800 [00:00<00:35, 274.41it/s]
  2%|▏
| 229/9800 [00:00<00:32, 296.57it/s]
  3%|▎
| 259/9800 [00:00<00:32, 290.71it/s]
  3%|▎
| 289/9800 [00:01<00:33, 286.73it/s]
  3%|▎
| 322/9800 [00:01<00:32, 292.15it/s]
  4%|▎
| 353/9800 [00:01<00:32, 293.92it/s]
  4%|▎
| 388/9800 [00:01<00:30, 305.55it/s]
  4%|▍
| 425/9800 [00:01<00:29, 319.08it/s]
  5%|▍
| 460/9800 [00:01<00:28, 324.14it/s]
  5%|▍
| 494/9800 [00:01<00:28, 325.01it/s]
  5%|▌
| 531/9800 [00:01<00:27, 333.68it/s]
  6%|▌
| 567/9800 [00:01<00:27, 337.35it/s]
  6%|▌
| 601/9800 [00:01<00:27, 330.29it/s]
  6%|▌
| 635/9800 [00:02<00:28, 321.80it/s]
  7%|▋
| 671/9800 [00:02<00:27, 328.76it/s]
  7%|▋
| 705/9800 [00:02<00:28, 313.72it/s]
  8%|▊
| 740/9800 [00:02<00:28, 320.25it/s]
  8%|▊
| 774/9800 [00:02<00:28, 322.25it/s]
  8%|▊
| 809/9800 [00:02<00:27, 326.43it/s]
  9%|▊
| 847/9800 [00:02<00:26, 337.24it/s]
  9%|▉
| 881/9800 [00:02<00:26, 334.09it/s]
  9%|▉
| 915/9800 [00:02<00:27, 328.09it/s]
 10%|▉
| 948/9800 [00:03<00:27, 324.82it/s]
 10%|▉
```

```
| 984/9800 [00:03<00:26, 330.95it/s]
 10%|
| 1018/9800 [00:03<00:27, 318.63it/s]
 11%|
| 1051/9800 [00:03<00:27, 318.27it/s]
 11%|
| 1084/9800 [00:03<00:27, 318.01it/s]
 11%|
| 1121/9800 [00:03<00:26, 325.01it/s]
 12%|
| 1154/9800 [00:03<00:27, 311.72it/s]
 12%|
| 1186/9800 [00:03<00:27, 310.53it/s]
 12%|
| 1218/9800 [00:03<00:28, 306.14it/s]
 13%|
| 1249/9800 [00:04<00:28, 296.72it/s]
 13%|
| 1286/9800 [00:04<00:27, 312.30it/s]
 13%|
| 1318/9800 [00:04<00:27, 303.84it/s]
 14%|
| 1349/9800 [00:04<00:30, 279.26it/s]
 14%|
| 1378/9800 [00:04<00:31, 266.82it/s]
 14%|
| 1406/9800 [00:04<00:32, 258.67it/s]
 15%|
| 1433/9800 [00:04<00:33, 247.58it/s]
 15%|
| 1459/9800 [00:04<00:35, 234.86it/s]
 15%|
| 1483/9800 [00:04<00:37, 220.75it/s]
 15%|
| 1518/9800 [00:05<00:33, 246.17it/s]
 16%|
| 1559/9800 [00:05<00:29, 277.44it/s]
 16%|
| 1597/9800 [00:05<00:27, 299.05it/s]
 17%|
| 1641/9800 [00:05<00:24, 327.91it/s]
 17%|
| 1682/9800 [00:05<00:23, 345.36it/s]
 18%|
| 1723/9800 [00:05<00:22, 358.73it/s]
 18%|
| 1763/9800 [00:05<00:21, 366.14it/s]
 18%|
| 1801/9800 [00:05<00:23, 342.22it/s]
 19%|
| 1837/9800 [00:05<00:24, 324.83it/s]
 19%|
| 1871/9800 [00:06<00:25, 307.80it/s]
 19%|
| 1903/9800 [00:06<00:26, 300.84it/s]
 20%|
| 1938/9800 [00:06<00:25, 310.76it/s]
 20%|
| 1970/9800 [00:06<00:25, 309.86it/s]
 20%|
| 2002/9800 [00:06<00:26, 292.29it/s]
```

```
 21%|
| 2035/9800 [00:06<00:25, 299.39it/s]
 21%|
| 2066/9800 [00:06<00:26, 288.97it/s]
 21%|
| 2096/9800 [00:06<00:27, 282.33it/s]
 22%|
| 2135/9800 [00:06<00:25, 304.96it/s]
 22%|
| 2167/9800 [00:07<00:24, 305.79it/s]
 22%|
| 2199/9800 [00:07<00:26, 286.62it/s]
 23%|
| 2232/9800 [00:07<00:25, 295.21it/s]
 23%|
| 2264/9800 [00:07<00:25, 298.86it/s]
 23%|
| 2301/9800 [00:07<00:23, 313.95it/s]
 24%|
| 2334/9800 [00:07<00:23, 311.42it/s]
 24%|
| 2368/9800 [00:07<00:23, 315.94it/s]
 25%|
| 2405/9800 [00:07<00:22, 326.94it/s]
 25%|
| 2438/9800 [00:07<00:24, 299.32it/s]
 25%|
| 2469/9800 [00:08<00:24, 295.56it/s]
 26%|
| 2500/9800 [00:08<00:24, 292.96it/s]
 26%|
| 2530/9800 [00:08<00:24, 291.62it/s]
 26%|
| 2565/9800 [00:08<00:23, 303.80it/s]
 27%|
| 2600/9800 [00:08<00:23, 312.96it/s]
 27%|
| 2638/9800 [00:08<00:21, 327.07it/s]
 27%|
| 2672/9800 [00:08<00:21, 327.04it/s]
 28%|
| 2705/9800 [00:08<00:23, 306.07it/s]
 28%|
| 2744/9800 [00:08<00:21, 323.95it/s]
 28%|
| 2778/9800 [00:09<00:21, 324.85it/s]
 29%|
| 2811/9800 [00:09<00:22, 304.72it/s]
 29%|
| 2843/9800 [00:09<00:22, 305.63it/s]
 29%|
| 2874/9800 [00:09<00:24, 286.52it/s]
 30%|
| 2915/9800 [00:09<00:22, 312.12it/s]
 30%|
| 2953/9800 [00:09<00:20, 326.41it/s]
 30%|
| 2987/9800 [00:09<00:21, 315.69it/s]
 31%|
| 3020/9800 [00:09<00:22, 302.30it/s]
 31%|
```

```
| 3051/9800 [00:09<00:22, 301.03it/s]
 31%|
| 3082/9800 [00:10<00:22, 296.72it/s]
 32%|
| 3114/9800 [00:10<00:22, 299.94it/s]
 32%|
| 3145/9800 [00:10<00:22, 292.62it/s]
 32%|
| 3175/9800 [00:10<00:22, 291.39it/s]
 33%|
| 3205/9800 [00:10<00:23, 283.94it/s]
 33%|
| 3238/9800 [00:10<00:22, 293.21it/s]
 33%|
| 3270/9800 [00:10<00:21, 297.43it/s]
 34%|
| 3305/9800 [00:10<00:21, 304.97it/s]
 34%|
| 3336/9800 [00:10<00:23, 271.12it/s]
 34%|
| 3365/9800 [00:11<00:23, 270.34it/s]
 35%|
| 3402/9800 [00:11<00:21, 291.34it/s]
 35%|
| 3434/9800 [00:11<00:21, 296.08it/s]
 35%|
| 3467/9800 [00:11<00:20, 302.18it/s]
 36%|
| 3498/9800 [00:11<00:21, 297.48it/s]
 36%|
| 3535/9800 [00:11<00:20, 312.89it/s]
 36%|
| 3572/9800 [00:11<00:19, 324.65it/s]
 37%|
| 3610/9800 [00:11<00:18, 335.90it/s]
 37%|
| 3649/9800 [00:11<00:17, 346.77it/s]
 38%|
| 3685/9800 [00:11<00:17, 346.63it/s]
 38%|
| 3726/9800 [00:12<00:16, 359.68it/s]
 38%|
| 3763/9800 [00:12<00:18, 331.52it/s]
 39%|
| 3797/9800 [00:12<00:19, 311.99it/s]
 39%|
| 3834/9800 [00:12<00:18, 323.99it/s]
 40%|
| 3871/9800 [00:12<00:17, 332.93it/s]
 40%|
| 3911/9800 [00:12<00:16, 346.94it/s]
 40%|
| 3954/9800 [00:12<00:16, 364.56it/s]
 41%|
| 3992/9800 [00:12<00:16, 356.60it/s]
 41%|
| 4029/9800 [00:12<00:16, 348.35it/s]
 41%|
| 4065/9800 [00:13<00:16, 347.71it/s]
 42%|
| 4101/9800 [00:13<00:17, 317.84it/s]
```


42%| | 4134/9800 [00:13<00:18, 310.52it/s]
43%| | 4166/9800 [00:13<00:18, 306.15it/s]
43%| | 4199/9800 [00:13<00:18, 305.93it/s]
43%| | 4237/9800 [00:13<00:17, 321.64it/s]
44%| | 4272/9800 [00:13<00:16, 326.01it/s]
44%| | 4305/9800 [00:13<00:18, 292.43it/s]
44%| | 4337/9800 [00:14<00:18, 296.87it/s]
45%| | 4370/9800 [00:14<00:17, 302.73it/s]
45%| | 4404/9800 [00:14<00:17, 309.64it/s]
45%| | 4437/9800 [00:14<00:17, 311.91it/s]
46%| | 4473/9800 [00:14<00:16, 318.08it/s]
46%| | 4506/9800 [00:14<00:16, 314.24it/s]
46%| | 4538/9800 [00:14<00:17, 305.13it/s]
47%| | 4569/9800 [00:14<00:17, 292.70it/s]
47%| | 4612/9800 [00:14<00:16, 320.83it/s]
48%| | 4656/9800 [00:14<00:14, 345.93it/s]
48%| | 4692/9800 [00:15<00:15, 330.77it/s]
48%| | 4727/9800 [00:15<00:16, 301.55it/s]
49%| | 4759/9800 [00:15<00:17, 290.21it/s]
49%| | 4792/9800 [00:15<00:16, 297.85it/s]
49%| | 4829/9800 [00:15<00:15, 313.18it/s]
50%| | 4865/9800 [00:15<00:15, 322.41it/s]
50%| | 4903/9800 [00:15<00:14, 334.23it/s]
50%| | 4943/9800 [00:15<00:14, 344.31it/s]
51%| | 4978/9800 [00:16<00:16, 285.10it/s]
51%| | 5013/9800 [00:16<00:16, 298.84it/s]
51%| | 5045/9800 [00:16<00:15, 298.08it/s]
52%| | 5076/9800 [00:16<00:16, 288.14it/s]
52%| | 5112/9800 [00:16<00:15, 300.38it/s]
53%| | 5149/9800 [00:16<00:14, 315.10it/s]
53%|

```
| 5182/9800 [00:16<00:14, 315.78it/s]
 53%|
| 5224/9800 [00:16<00:13, 337.92it/s]
 54%|
| 5259/9800 [00:16<00:13, 337.53it/s]
 54%|
| 5295/9800 [00:16<00:13, 340.08it/s]
 54%|
| 5330/9800 [00:17<00:13, 339.05it/s]
 55%|
| 5365/9800 [00:17<00:13, 330.65it/s]
 55%|
| 5404/9800 [00:17<00:12, 342.84it/s]
 56%|
| 5439/9800 [00:17<00:13, 325.71it/s]
 56%|
| 5477/9800 [00:17<00:12, 336.71it/s]
 56%|
| 5512/9800 [00:17<00:17, 241.80it/s]
 57%|
| 5541/9800 [00:17<00:17, 246.72it/s]
 57%|
| 5576/9800 [00:17<00:15, 268.20it/s]
 57%|
| 5606/9800 [00:18<00:15, 271.02it/s]
 58%|
| 5646/9800 [00:18<00:13, 297.40it/s]
 58%|
| 5679/9800 [00:18<00:13, 303.13it/s]
 58%|
| 5712/9800 [00:18<00:13, 307.27it/s]
 59%|
| 5744/9800 [00:18<00:13, 303.90it/s]
 59%|
| 5776/9800 [00:18<00:13, 291.72it/s]
 59%|
| 5806/9800 [00:18<00:15, 263.21it/s]
 60%|
| 5837/9800 [00:18<00:14, 269.94it/s]
 60%|
| 5865/9800 [00:19<00:14, 263.67it/s]
 60%|
| 5896/9800 [00:19<00:14, 273.14it/s]
 60%|
| 5924/9800 [00:19<00:15, 256.99it/s]
 61%|
| 5956/9800 [00:19<00:14, 270.38it/s]
 61%|
| 5988/9800 [00:19<00:13, 280.62it/s]
 61%|
| 6024/9800 [00:19<00:12, 297.52it/s]
 62%|
| 6055/9800 [00:19<00:12, 297.72it/s]
 62%|
| 6090/9800 [00:19<00:12, 308.41it/s]
 63%|
| 6127/9800 [00:19<00:11, 321.25it/s]
 63%|
| 6160/9800 [00:19<00:11, 309.30it/s]
 63%|
| 6197/9800 [00:20<00:11, 321.92it/s]
```

64%|
| 6242/9800 [00:20<00:10, 348.74it/s]
64%|
| 6278/9800 [00:20<00:10, 340.09it/s]
64%|
| 6314/9800 [00:20<00:10, 341.90it/s]
65%|
| 6349/9800 [00:20<00:10, 325.14it/s]
65%|
| 6390/9800 [00:20<00:09, 343.20it/s]
66%|
| 6432/9800 [00:20<00:09, 359.41it/s]
66%|
| 6473/9800 [00:20<00:09, 369.22it/s]
66%|
| 6511/9800 [00:20<00:09, 363.86it/s]
67%|
| 6548/9800 [00:21<00:09, 349.16it/s]
67%|
| 6591/9800 [00:21<00:08, 362.57it/s]
68%|
| 6630/9800 [00:21<00:08, 366.24it/s]
68%|
| 6668/9800 [00:21<00:08, 366.01it/s]
68%|
| 6705/9800 [00:21<00:10, 291.00it/s]
69%|
| 6737/9800 [00:21<00:11, 256.07it/s]
69%|
| 6766/9800 [00:21<00:11, 259.70it/s]
69%|
| 6794/9800 [00:21<00:12, 245.91it/s]
70%|
| 6824/9800 [00:22<00:11, 257.32it/s]
70%|
| 6851/9800 [00:22<00:12, 238.85it/s]
70%|
| 6876/9800 [00:22<00:12, 231.36it/s]
71%|
| 6910/9800 [00:22<00:11, 253.61it/s]
71%|
| 6940/9800 [00:22<00:10, 263.17it/s]
71%|
| 6975/9800 [00:22<00:10, 281.59it/s]
71%|
| 7005/9800 [00:22<00:09, 283.65it/s]
72%|
| 7035/9800 [00:22<00:10, 266.84it/s]
72%|
| 7063/9800 [00:22<00:10, 261.58it/s]
72%|
| 7091/9800 [00:23<00:10, 263.83it/s]
73%|
| 7118/9800 [00:23<00:10, 253.70it/s]
73%|
| 7146/9800 [00:23<00:10, 252.61it/s]
73%|
| 7177/9800 [00:23<00:10, 262.05it/s]
74%|
| 7212/9800 [00:23<00:09, 280.70it/s]
74%|

```
| 7241/9800 [00:23<00:09, 264.83it/s]
 74%|
| 7279/9800 [00:23<00:08, 288.67it/s]
 75%|
| 7315/9800 [00:23<00:08, 300.79it/s]
 75%|
| 7351/9800 [00:23<00:07, 313.11it/s]
 75%|
| 7384/9800 [00:24<00:07, 303.96it/s]
 76%|
| 7418/9800 [00:24<00:07, 310.54it/s]
 76%|
| 7450/9800 [00:24<00:08, 286.40it/s]
 76%|
| 7480/9800 [00:24<00:08, 280.61it/s]
 77%|
| 7516/9800 [00:24<00:07, 297.54it/s]
 77%|
| 7547/9800 [00:24<00:07, 287.69it/s]
 77%|
| 7586/9800 [00:24<00:07, 309.31it/s]
 78%|
| 7621/9800 [00:24<00:06, 317.05it/s]
 78%|
| 7657/9800 [00:24<00:06, 321.79it/s]
 78%|
| 7691/9800 [00:25<00:06, 323.33it/s]
 79%|
| 7724/9800 [00:25<00:06, 307.15it/s]
 79%|
| 7756/9800 [00:25<00:06, 303.85it/s]
 79%|
| 7787/9800 [00:25<00:06, 291.88it/s]
 80%|
| 7817/9800 [00:25<00:06, 290.87it/s]
 80%|
| 7847/9800 [00:25<00:07, 274.25it/s]
 80%|
| 7875/9800 [00:25<00:07, 257.69it/s]
 81%|
| 7902/9800 [00:25<00:07, 258.29it/s]
 81%|
| 7943/9800 [00:25<00:06, 288.10it/s]
 81%|
| 7977/9800 [00:26<00:06, 298.77it/s]
 82%|
| 8013/9800 [00:26<00:05, 311.57it/s]
 82%|
| 8046/9800 [00:26<00:05, 306.33it/s]
 82%|
| 8082/9800 [00:26<00:05, 317.30it/s]
 83%|
| 8115/9800 [00:26<00:05, 296.78it/s]
 83%|
| 8157/9800 [00:26<00:05, 322.43it/s]
 84%|
| 8200/9800 [00:26<00:04, 345.27it/s]
 84%|
| 8236/9800 [00:26<00:04, 337.78it/s]
 84%|
| 8276/9800 [00:26<00:04, 350.61it/s]
```

```
 85%|
| 8314/9800 [00:27<00:04, 351.01it/s]
 85%|
| 8350/9800 [00:27<00:04, 316.39it/s]
 86%|
| 8383/9800 [00:27<00:04, 309.56it/s]
 86%|
| 8415/9800 [00:27<00:04, 282.81it/s]
 86%|
| 8454/9800 [00:27<00:04, 305.35it/s]
 87%|
| 8489/9800 [00:27<00:04, 314.10it/s]
 87%|
| 8527/9800 [00:27<00:03, 327.93it/s]
 87%|
| 8565/9800 [00:27<00:03, 338.36it/s]
 88%|
| 8600/9800 [00:27<00:03, 309.19it/s]
 88%|
| 8632/9800 [00:28<00:03, 301.78it/s]
 88%|
| 8663/9800 [00:28<00:03, 300.68it/s]
 89%|
| 8698/9800 [00:28<00:03, 310.63it/s]
 89%|
| 8732/9800 [00:28<00:03, 315.37it/s]
 89%|
| 8764/9800 [00:28<00:03, 313.05it/s]
 90%|
| 8797/9800 [00:28<00:03, 314.32it/s]
 90%|
|         | 8835/9800 [00:28<00:02, 328.11it/s]
 90%|
|         | 8869/9800 [00:28<00:02, 320.37it/s]
 91%|
|         | 8902/9800 [00:28<00:02, 305.29it/s]
 91%|
|         | 8933/9800 [00:29<00:02, 296.15it/s]
 91%|
|         | 8966/9800 [00:29<00:02, 302.23it/s]
 92%|
|         | 9001/9800 [00:29<00:02, 308.49it/s]
 92%|
|       | 9037/9800 [00:29<00:02, 318.92it/s]
 93%|
|         | 9070/9800 [00:29<00:02, 311.24it/s]
 93%|
|       | 9103/9800 [00:29<00:02, 313.07it/s]
 93%|
|         | 9135/9800 [00:29<00:02, 307.87it/s]
 94%|
|       | 9169/9800 [00:29<00:02, 313.37it/s]
 94%|
|         | 9204/9800 [00:29<00:01, 319.99it/s]
 94%|
|       | 9237/9800 [00:30<00:01, 315.56it/s]
 95%|
|         | 9269/9800 [00:30<00:01, 302.52it/s]
 95%|
|       | 9306/9800 [00:30<00:01, 316.77it/s]
 95%|
```

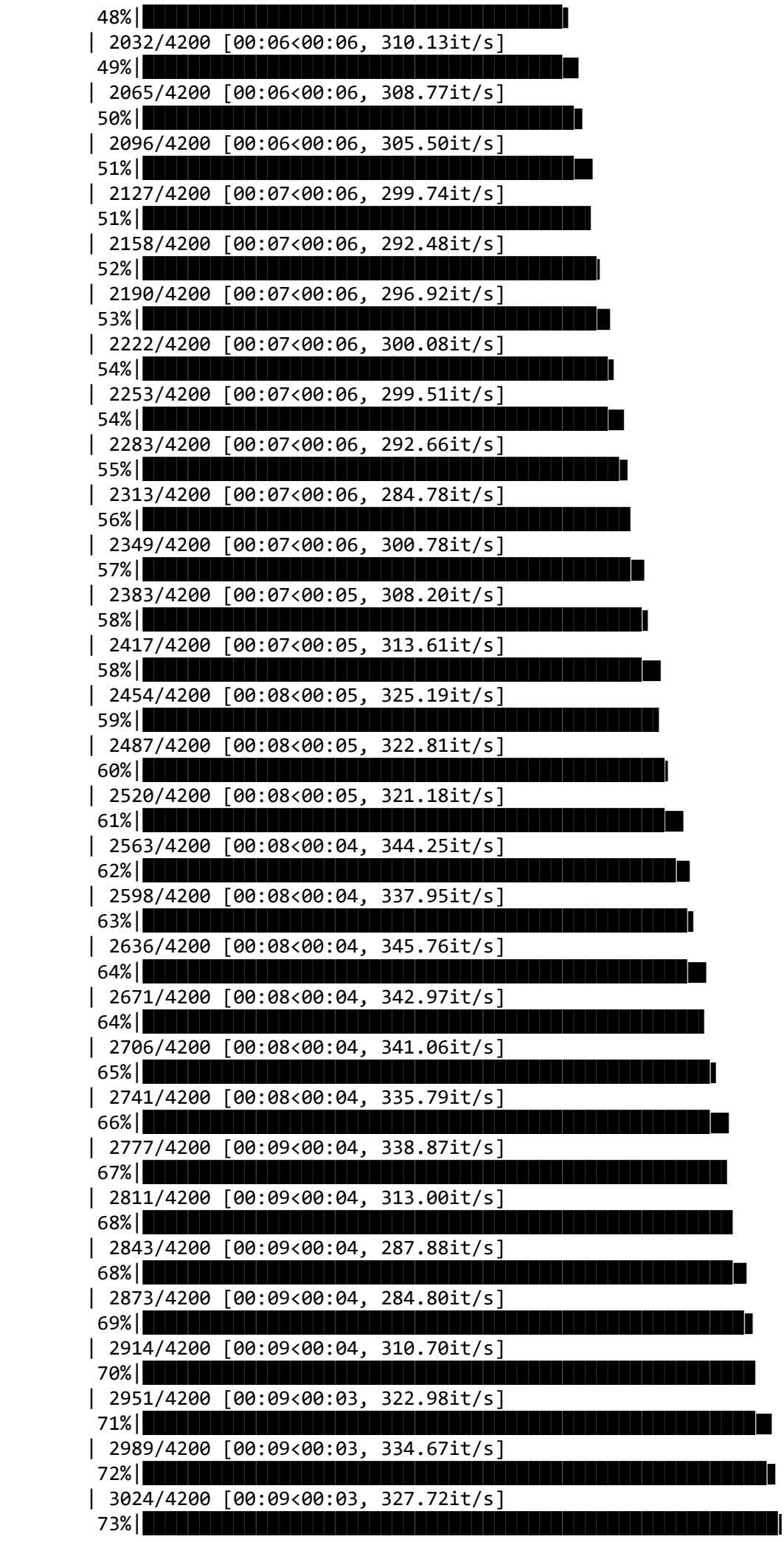
```
 96%|                                                                                | 9338/9800 [00:30<00:01, 314.01it/s]
 96%|                                                                                | 9378/9800 [00:30<00:01, 332.34it/s]
 96%|                                                                                | 9412/9800 [00:30<00:01, 287.14it/s]
 96%|                                                                                | 9443/9800 [00:30<00:01, 287.14it/s]
 97%|                                                                                | 9473/9800 [00:30<00:01, 287.46it/s]
 97%|                                                                                | 9510/9800 [00:30<00:00, 305.05it/s]
 97%|                                                                                | 9547/9800 [00:31<00:00, 318.70it/s]
 98%|                                                                                | 9580/9800 [00:31<00:00, 304.22it/s]
 98%|                                                                                | 9619/9800 [00:31<00:00, 322.50it/s]
 98%|                                                                                | 9652/9800 [00:31<00:00, 310.08it/s]
 99%|                                                                                | 9690/9800 [00:31<00:00, 324.85it/s]
 99%|                                                                                | 9728/9800 [00:31<00:00, 332.54it/s]
100%|                                                                                | 9762/9800 [00:31<00:00, 316.10it/s]
100%|                                                                                | 9800/9800 [00:31<00:00, 308.29it/s]

9800
300
```

In [100]:

```
tfidf_w2v_cv=[]
for sentence in tqdm(X_cv['essay']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if(word in glove_words) and (word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector+=(vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight!=0:
        vector /=tf_idf_weight
    tfidf_w2v_cv.append(vector)

print(len(tfidf_w2v_cv))
print(len(tfidf_w2v_cv[0]))
```

```
  0%|
| 0/4200 [00:00<?, ?it/s]
  1%|▏
| 35/4200 [00:00<00:12, 336.57it/s]
  2%|▏
| 64/4200 [00:00<00:13, 312.83it/s]
  2%|▏▏
| 98/4200 [00:00<00:12, 316.96it/s]
  3%|▏▏
| 128/4200 [00:00<00:13, 307.86it/s]
  4%|▏▏
| 161/4200 [00:00<00:13, 310.64it/s]
  5%|▏▏▏
| 197/4200 [00:00<00:12, 320.53it/s]
  5%|▏▏▏
| 226/4200 [00:00<00:13, 302.97it/s]
  6%|▏▏▏▏
| 258/4200 [00:00<00:12, 304.38it/s]
  7%|▏▏▏▏
| 287/4200 [00:00<00:14, 279.17it/s]
  8%|▏▏▏▏▏
| 321/4200 [00:01<00:13, 291.97it/s]
  8%|▏▏▏▏▏
| 350/4200 [00:01<00:13, 284.54it/s]
  9%|▏▏▏▏▏
| 387/4200 [00:01<00:12, 302.74it/s]
 10%|▏▏▏▏▏▏
| 418/4200 [00:01<00:12, 291.16it/s]
 11%|▏▏▏▏▏▏
| 449/4200 [00:01<00:12, 293.22it/s]
 11%|▏▏▏▏▏▏▏
| 479/4200 [00:01<00:14, 255.96it/s]
 12%|▏▏▏▏▏▏▏
| 510/4200 [00:01<00:13, 267.31it/s]
 13%|▏▏▏▏▏▏▏▏
| 549/4200 [00:01<00:12, 292.53it/s]
 14%|▏▏▏▏▏▏▏▏▏
| 589/4200 [00:01<00:11, 315.20it/s]
 15%|▏▏▏▏▏▏▏▏▏
| 622/4200 [00:02<00:11, 301.98it/s]
 16%|▏▏▏▏▏▏▏▏▏▏
| 654/4200 [00:02<00:12, 287.32it/s]
 16%|▏▏▏▏▏▏▏▏▏▏
| 684/4200 [00:02<00:12, 278.10it/s]
 17%|▏▏▏▏▏▏▏▏▏▏▏
| 721/4200 [00:02<00:11, 297.60it/s]
 18%|▏▏▏▏▏▏▏▏▏▏▏▏
| 754/4200 [00:02<00:11, 303.28it/s]
 19%|▏▏▏▏▏▏▏▏▏▏▏▏
| 792/4200 [00:02<00:10, 319.59it/s]
 20%|▏▏▏▏▏▏▏▏▏▏▏▏▏
| 829/4200 [00:02<00:10, 329.67it/s]
 21%|▏▏▏▏▏▏▏▏▏▏▏▏▏
| 867/4200 [00:02<00:09, 339.65it/s]
 21%|▏▏▏▏▏▏▏▏▏▏▏▏▏▏
| 902/4200 [00:02<00:10, 327.35it/s]
 22%|▏▏▏▏▏▏▏▏▏▏▏▏▏▏
| 936/4200 [00:03<00:10, 319.86it/s]
 23%|▏▏▏▏▏▏▏▏▏▏▏▏▏▏▏
| 969/4200 [00:03<00:10, 311.86it/s]
 24%|▏▏▏▏▏▏▏▏▏▏▏▏▏▏▏
```

```
| 1002/4200 [00:03<00:10, 309.97it/s]
 25%|
| 1035/4200 [00:03<00:10, 312.16it/s]
 25%|
| 1067/4200 [00:03<00:10, 310.83it/s]
 26%|
| 1102/4200 [00:03<00:09, 318.14it/s]
 27%|
| 1134/4200 [00:03<00:10, 294.13it/s]
 28%|
| 1169/4200 [00:03<00:09, 305.71it/s]
 29%|
| 1200/4200 [00:03<00:10, 299.87it/s]
 30%|
| 1242/4200 [00:04<00:09, 324.99it/s]
 30%|
| 1278/4200 [00:04<00:08, 331.08it/s]
 31%|
| 1313/4200 [00:04<00:08, 332.74it/s]
 32%|
| 1347/4200 [00:04<00:09, 302.70it/s]
 33%|
| 1379/4200 [00:04<00:09, 300.79it/s]
 34%|
| 1414/4200 [00:04<00:08, 310.71it/s]
 34%|
| 1446/4200 [00:04<00:09, 289.64it/s]
 35%|
| 1479/4200 [00:04<00:09, 297.44it/s]
 36%|
| 1510/4200 [00:04<00:09, 294.26it/s]
 37%|
| 1545/4200 [00:05<00:08, 305.81it/s]
 38%|
| 1581/4200 [00:05<00:08, 316.91it/s]
 38%|
| 1615/4200 [00:05<00:08, 319.86it/s]
 39%|
| 1648/4200 [00:05<00:08, 308.38it/s]
 40%|
| 1680/4200 [00:05<00:08, 291.36it/s]
 41%|
| 1710/4200 [00:05<00:09, 271.59it/s]
 41%|
| 1740/4200 [00:05<00:08, 276.46it/s]
 42%|
| 1769/4200 [00:05<00:08, 270.98it/s]
 43%|
| 1800/4200 [00:05<00:08, 278.60it/s]
 44%|
| 1834/4200 [00:06<00:08, 291.53it/s]
 44%|
| 1867/4200 [00:06<00:07, 298.85it/s]
 45%|
| 1898/4200 [00:06<00:07, 288.63it/s]
 46%|
| 1931/4200 [00:06<00:07, 296.68it/s]
 47%|
| 1967/4200 [00:06<00:07, 306.83it/s]
 48%|
| 1999/4200 [00:06<00:07, 307.12it/s]
```


48%| | 2032/4200 [00:06<00:06, 310.13it/s]
49%| | 2065/4200 [00:06<00:06, 308.77it/s]
50%| | 2096/4200 [00:06<00:06, 305.50it/s]
51%| | 2127/4200 [00:07<00:06, 299.74it/s]
51%| | 2158/4200 [00:07<00:06, 292.48it/s]
52%| | 2190/4200 [00:07<00:06, 296.92it/s]
53%| | 2222/4200 [00:07<00:06, 300.08it/s]
54%| | 2253/4200 [00:07<00:06, 299.51it/s]
54%| | 2283/4200 [00:07<00:06, 292.66it/s]
55%| | 2313/4200 [00:07<00:06, 284.78it/s]
56%| | 2349/4200 [00:07<00:06, 300.78it/s]
57%| | 2383/4200 [00:07<00:05, 308.20it/s]
58%| | 2417/4200 [00:07<00:05, 313.61it/s]
58%| | 2454/4200 [00:08<00:05, 325.19it/s]
59%| | 2487/4200 [00:08<00:05, 322.81it/s]
60%| | 2520/4200 [00:08<00:05, 321.18it/s]
61%| | 2563/4200 [00:08<00:04, 344.25it/s]
62%| | 2598/4200 [00:08<00:04, 337.95it/s]
63%| | 2636/4200 [00:08<00:04, 345.76it/s]
64%| | 2671/4200 [00:08<00:04, 342.97it/s]
64%| | 2706/4200 [00:08<00:04, 341.06it/s]
65%| | 2741/4200 [00:08<00:04, 335.79it/s]
66%| | 2777/4200 [00:09<00:04, 338.87it/s]
67%| | 2811/4200 [00:09<00:04, 313.00it/s]
68%| | 2843/4200 [00:09<00:04, 287.88it/s]
68%| | 2873/4200 [00:09<00:04, 284.80it/s]
69%| | 2914/4200 [00:09<00:04, 310.70it/s]
70%| | 2951/4200 [00:09<00:03, 322.98it/s]
71%| | 2989/4200 [00:09<00:03, 334.67it/s]
72%| | 3024/4200 [00:09<00:03, 327.72it/s]
73%|

```
| 3062/4200 [00:09<00:03, 338.13it/s]
 74%|
| 3097/4200 [00:10<00:03, 330.03it/s]
 75%|
| 3131/4200 [00:10<00:03, 314.51it/s]
 75%|
| 3163/4200 [00:10<00:03, 305.30it/s]
 76%|
| 3194/4200 [00:10<00:03, 296.17it/s]
 77%|
| 3224/4200 [00:10<00:03, 280.65it/s]
 78%|
| 3255/4200 [00:10<00:03, 285.68it/s]
 78%|
| 3286/4200 [00:10<00:03, 286.11it/s]
 79%|
| 3317/4200 [00:10<00:03, 289.62it/s]
 80%|
| 3347/4200 [00:10<00:02, 285.98it/s]
 81%|
| 3385/4200 [00:11<00:02, 305.94it/s]
 81%|
| 3417/4200 [00:11<00:02, 303.00it/s]
 82%|
| 3450/4200 [00:11<00:02, 307.19it/s]
 83%|
| 3485/4200 [00:11<00:02, 315.47it/s]
 84%|
| 3521/4200 [00:11<00:02, 324.11it/s]
 85%|
| 3554/4200 [00:11<00:02, 311.12it/s]
 86%|
| 3593/4200 [00:11<00:01, 327.91it/s]
 86%|
| 3627/4200 [00:11<00:01, 323.89it/s]
 87%|
| 3666/4200 [00:11<00:01, 337.72it/s]
 88%|
| 3703/4200 [00:11<00:01, 342.95it/s]
 89%|
| 3739/4200 [00:12<00:01, 343.95it/s]
 90%|
|         | 3779/4200 [00:12<00:01, 355.24it/s]
 91%|
|         | 3815/4200 [00:12<00:01, 344.40it/s]
 92%|
|       | 3850/4200 [00:12<00:01, 345.36it/s]
 93%|
|       | 3888/4200 [00:12<00:00, 351.17it/s]
 94%|
|      | 3931/4200 [00:12<00:00, 367.82it/s]
 94%|
|      | 3969/4200 [00:12<00:00, 362.90it/s]
 96%|
|    | 4015/4200 [00:12<00:00, 379.79it/s]
 97%|
|    | 4054/4200 [00:12<00:00, 378.37it/s]
 97%|
|   | 4093/4200 [00:13<00:00, 373.03it/s]
 98%|
|  | 4131/4200 [00:13<00:00, 362.25it/s]
```

```
 99%|████████████████████████████████████████████████████████████
████████ | 4168/4200 [00:13<00:00, 352.08it/s]
100%|████████████████████████████████████████████████████████████
████████| 4200/4200 [00:13<00:00, 314.62it/s]

4200
300
```

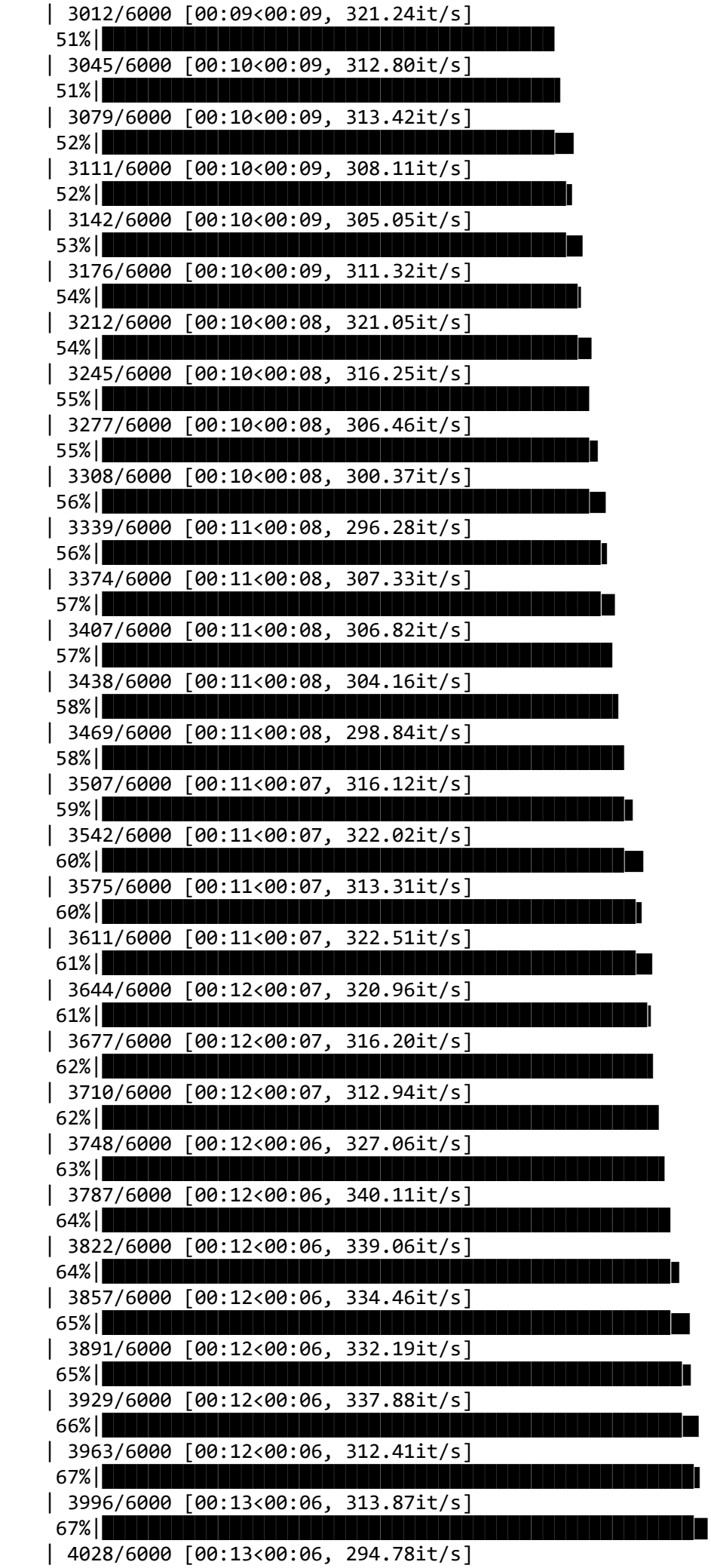
In [101]:

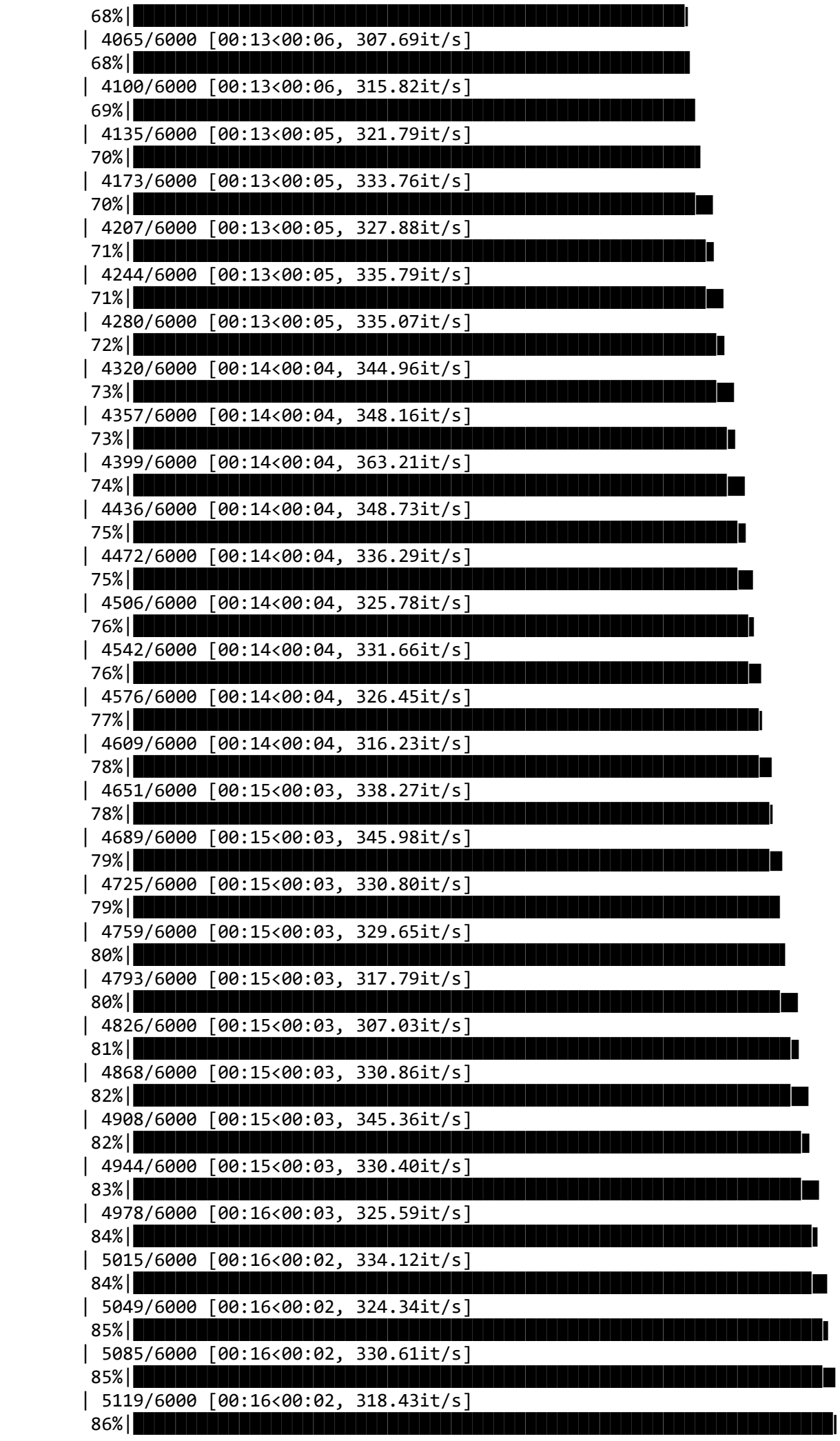
```
tfidf_w2v_test=[]
for sentence in tqdm(X_test['essay']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if (word in glove_words)and(word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector+=(vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    tfidf_w2v_test.append(vector)
print(len(tfidf_w2v_test))
print(len(tfidf_w2v_test[0]))
```

```
  0%|
| 0/6000 [00:00<?, ?it/s]
  1%|▏
| 32/6000 [00:00<00:19, 307.76it/s]
  1%|▏
| 62/6000 [00:00<00:19, 301.73it/s]
  1%|▏
| 87/6000 [00:00<00:21, 280.30it/s]
  2%|▏
| 124/6000 [00:00<00:19, 296.49it/s]
  3%|▏
| 158/6000 [00:00<00:19, 305.03it/s]
  3%|▎
| 197/6000 [00:00<00:17, 323.14it/s]
  4%|▎
| 232/6000 [00:00<00:17, 327.05it/s]
  4%|▍
| 264/6000 [00:00<00:18, 317.20it/s]
  5%|▍
| 301/6000 [00:00<00:17, 327.89it/s]
  6%|▍
| 333/6000 [00:01<00:18, 306.80it/s]
  6%|▌
| 368/6000 [00:01<00:17, 315.18it/s]
  7%|▋
| 404/6000 [00:01<00:17, 323.88it/s]
  7%|▋
| 437/6000 [00:01<00:17, 318.18it/s]
  8%|▊
| 469/6000 [00:01<00:17, 307.73it/s]
  8%|▊
| 502/6000 [00:01<00:17, 310.57it/s]
  9%|▊
| 535/6000 [00:01<00:17, 312.58it/s]
  9%|▉
| 567/6000 [00:01<00:21, 257.12it/s]
 10%|▉
| 595/6000 [00:02<00:22, 241.75it/s]
 10%|▉
| 621/6000 [00:02<00:24, 219.47it/s]
 11%|▉
| 656/6000 [00:02<00:21, 245.05it/s]
 12%|▉
| 692/6000 [00:02<00:19, 268.60it/s]
 12%|▉
| 733/6000 [00:02<00:17, 297.01it/s]
 13%|▉
| 772/6000 [00:02<00:16, 316.80it/s]
 13%|▉
| 806/6000 [00:02<00:18, 273.51it/s]
 14%|▉
| 838/6000 [00:02<00:18, 282.95it/s]
 15%|▉
| 872/6000 [00:02<00:17, 291.84it/s]
 15%|▉
| 905/6000 [00:03<00:17, 299.05it/s]
 16%|▉
| 939/6000 [00:03<00:16, 303.65it/s]
 16%|▉
| 971/6000 [00:03<00:17, 282.29it/s]
 17%|▉
```

```
| 1002/6000 [00:03<00:17, 286.86it/s]
 17%|
| 1034/6000 [00:03<00:16, 292.82it/s]
 18%|
| 1069/6000 [00:03<00:16, 304.73it/s]
 18%|
| 1106/6000 [00:03<00:15, 318.46it/s]
 19%|
| 1139/6000 [00:03<00:16, 300.74it/s]
 20%|
| 1172/6000 [00:03<00:15, 305.54it/s]
 20%|
| 1203/6000 [00:04<00:16, 299.78it/s]
 21%|
| 1236/6000 [00:04<00:15, 304.84it/s]
 21%|
| 1275/6000 [00:04<00:14, 323.00it/s]
 22%|
| 1308/6000 [00:04<00:14, 313.95it/s]
 22%|
| 1340/6000 [00:04<00:15, 301.49it/s]
 23%|
| 1374/6000 [00:04<00:14, 308.73it/s]
 23%|
| 1407/6000 [00:04<00:14, 311.26it/s]
 24%|
| 1442/6000 [00:04<00:14, 318.46it/s]
 25%|
| 1475/6000 [00:04<00:16, 271.11it/s]
 25%|
| 1504/6000 [00:05<00:16, 267.35it/s]
 26%|
| 1532/6000 [00:05<00:16, 267.94it/s]
 26%|
| 1564/6000 [00:05<00:15, 278.75it/s]
 27%|
| 1593/6000 [00:05<00:15, 278.81it/s]
 27%|
| 1625/6000 [00:05<00:15, 286.91it/s]
 28%|
| 1655/6000 [00:05<00:15, 287.40it/s]
 28%|
| 1685/6000 [00:05<00:14, 287.72it/s]
 29%|
| 1714/6000 [00:05<00:15, 285.04it/s]
 29%|
| 1746/6000 [00:05<00:14, 291.48it/s]
 30%|
| 1784/6000 [00:05<00:13, 310.33it/s]
 30%|
| 1816/6000 [00:06<00:13, 306.02it/s]
 31%|
| 1847/6000 [00:06<00:13, 300.09it/s]
 31%|
| 1880/6000 [00:06<00:13, 305.07it/s]
 32%|
| 1911/6000 [00:06<00:13, 292.68it/s]
 32%|
| 1942/6000 [00:06<00:13, 294.30it/s]
 33%|
| 1982/6000 [00:06<00:12, 316.61it/s]
```

```
 34%|
| 2019/6000 [00:06<00:12, 327.46it/s]
 34%|
| 2053/6000 [00:06<00:12, 323.59it/s]
 35%|
| 2092/6000 [00:06<00:11, 337.49it/s]
 35%|
| 2127/6000 [00:07<00:12, 322.33it/s]
 36%|
| 2161/6000 [00:07<00:11, 323.70it/s]
 37%|
| 2194/6000 [00:07<00:12, 314.44it/s]
 37%|
| 2226/6000 [00:07<00:12, 301.79it/s]
 38%|
| 2260/6000 [00:07<00:12, 308.94it/s]
 38%|
| 2292/6000 [00:07<00:12, 298.25it/s]
 39%|
| 2326/6000 [00:07<00:12, 303.05it/s]
 39%|
| 2363/6000 [00:07<00:11, 317.17it/s]
 40%|
| 2396/6000 [00:07<00:11, 317.22it/s]
 40%|
| 2428/6000 [00:08<00:11, 300.18it/s]
 41%|
| 2459/6000 [00:08<00:11, 296.15it/s]
 42%|
| 2492/6000 [00:08<00:11, 302.20it/s]
 42%|
| 2523/6000 [00:08<00:11, 290.83it/s]
 43%|
| 2556/6000 [00:08<00:11, 298.32it/s]
 43%|
| 2587/6000 [00:08<00:11, 285.09it/s]
 44%|
| 2624/6000 [00:08<00:11, 300.23it/s]
 44%|
| 2657/6000 [00:08<00:10, 305.19it/s]
 45%|
| 2688/6000 [00:08<00:11, 296.45it/s]
 45%|
| 2719/6000 [00:09<00:11, 296.96it/s]
 46%|
| 2749/6000 [00:09<00:11, 287.62it/s]
 46%|
| 2780/6000 [00:09<00:11, 290.70it/s]
 47%|
| 2810/6000 [00:09<00:10, 290.04it/s]
 47%|
| 2846/6000 [00:09<00:10, 304.89it/s]
 48%|
| 2877/6000 [00:09<00:10, 302.83it/s]
 48%|
| 2910/6000 [00:09<00:10, 307.00it/s]
 49%|
| 2942/6000 [00:09<00:09, 310.64it/s]
 50%|
| 2979/6000 [00:09<00:09, 322.95it/s]
 50%|
```


| 3012/6000 [00:09<00:09, 321.24it/s]
51%|
| 3045/6000 [00:10<00:09, 312.80it/s]
51%|
| 3079/6000 [00:10<00:09, 313.42it/s]
52%|
| 3111/6000 [00:10<00:09, 308.11it/s]
52%|
| 3142/6000 [00:10<00:09, 305.05it/s]
53%|
| 3176/6000 [00:10<00:09, 311.32it/s]
54%|
| 3212/6000 [00:10<00:08, 321.05it/s]
54%|
| 3245/6000 [00:10<00:08, 316.25it/s]
55%|
| 3277/6000 [00:10<00:08, 306.46it/s]
55%|
| 3308/6000 [00:10<00:08, 300.37it/s]
56%|
| 3339/6000 [00:11<00:08, 296.28it/s]
56%|
| 3374/6000 [00:11<00:08, 307.33it/s]
57%|
| 3407/6000 [00:11<00:08, 306.82it/s]
57%|
| 3438/6000 [00:11<00:08, 304.16it/s]
58%|
| 3469/6000 [00:11<00:08, 298.84it/s]
58%|
| 3507/6000 [00:11<00:07, 316.12it/s]
59%|
| 3542/6000 [00:11<00:07, 322.02it/s]
60%|
| 3575/6000 [00:11<00:07, 313.31it/s]
60%|
| 3611/6000 [00:11<00:07, 322.51it/s]
61%|
| 3644/6000 [00:12<00:07, 320.96it/s]
61%|
| 3677/6000 [00:12<00:07, 316.20it/s]
62%|
| 3710/6000 [00:12<00:07, 312.94it/s]
62%|
| 3748/6000 [00:12<00:06, 327.06it/s]
63%|
| 3787/6000 [00:12<00:06, 340.11it/s]
64%|
| 3822/6000 [00:12<00:06, 339.06it/s]
64%|
| 3857/6000 [00:12<00:06, 334.46it/s]
65%|
| 3891/6000 [00:12<00:06, 332.19it/s]
65%|
| 3929/6000 [00:12<00:06, 337.88it/s]
66%|
| 3963/6000 [00:12<00:06, 312.41it/s]
67%|
| 3996/6000 [00:13<00:06, 313.87it/s]
67%|
| 4028/6000 [00:13<00:06, 294.78it/s]


68%| | 4065/6000 [00:13<00:06, 307.69it/s]
68%| | 4100/6000 [00:13<00:06, 315.82it/s]
69%| | 4135/6000 [00:13<00:05, 321.79it/s]
70%| | 4173/6000 [00:13<00:05, 333.76it/s]
70%| | 4207/6000 [00:13<00:05, 327.88it/s]
71%| | 4244/6000 [00:13<00:05, 335.79it/s]
71%| | 4280/6000 [00:13<00:05, 335.07it/s]
72%| | 4320/6000 [00:14<00:04, 344.96it/s]
73%| | 4357/6000 [00:14<00:04, 348.16it/s]
73%| | 4399/6000 [00:14<00:04, 363.21it/s]
74%| | 4436/6000 [00:14<00:04, 348.73it/s]
75%| | 4472/6000 [00:14<00:04, 336.29it/s]
75%| | 4506/6000 [00:14<00:04, 325.78it/s]
76%| | 4542/6000 [00:14<00:04, 331.66it/s]
76%| | 4576/6000 [00:14<00:04, 326.45it/s]
77%| | 4609/6000 [00:14<00:04, 316.23it/s]
78%| | 4651/6000 [00:15<00:03, 338.27it/s]
78%| | 4689/6000 [00:15<00:03, 345.98it/s]
79%| | 4725/6000 [00:15<00:03, 330.80it/s]
79%| | 4759/6000 [00:15<00:03, 329.65it/s]
80%| | 4793/6000 [00:15<00:03, 317.79it/s]
80%| | 4826/6000 [00:15<00:03, 307.03it/s]
81%| | 4868/6000 [00:15<00:03, 330.86it/s]
82%| | 4908/6000 [00:15<00:03, 345.36it/s]
82%| | 4944/6000 [00:15<00:03, 330.40it/s]
83%| | 4978/6000 [00:16<00:03, 325.59it/s]
84%| | 5015/6000 [00:16<00:02, 334.12it/s]
84%| | 5049/6000 [00:16<00:02, 324.34it/s]
85%| | 5085/6000 [00:16<00:02, 330.61it/s]
85%| | 5119/6000 [00:16<00:02, 318.43it/s]
86%|

```
| 5161/6000 [00:16<00:02, 340.01it/s]
 87%|
| 5202/6000 [00:16<00:02, 354.67it/s]
 87%|
| 5239/6000 [00:16<00:02, 343.19it/s]
 88%|
| 5274/6000 [00:16<00:02, 315.37it/s]
 88%|
| 5307/6000 [00:17<00:03, 227.22it/s]
 89%|
| 5334/6000 [00:17<00:02, 226.58it/s]
 90%|
| 5373/6000 [00:17<00:02, 257.11it/s]
 90%|
          | 5412/6000 [00:17<00:02, 283.90it/s]
 91%|
          | 5448/6000 [00:17<00:01, 300.11it/s]
 91%|
          | 5482/6000 [00:17<00:01, 307.70it/s]
 92%|
          | 5515/6000 [00:17<00:01, 310.56it/s]
 92%|
          | 5549/6000 [00:17<00:01, 315.32it/s]
 93%|
          | 5584/6000 [00:18<00:01, 321.41it/s]
 94%|
          | 5623/6000 [00:18<00:01, 335.84it/s]
 94%|
       | 5658/6000 [00:18<00:01, 332.25it/s]
 95%|
       | 5692/6000 [00:18<00:00, 326.84it/s]
 95%|
       | 5726/6000 [00:18<00:00, 315.95it/s]
 96%|
       | 5758/6000 [00:18<00:00, 313.46it/s]
 97%|
       | 5796/6000 [00:18<00:00, 327.43it/s]
 97%|
       | 5843/6000 [00:18<00:00, 356.96it/s]
 98%|
       | 5882/6000 [00:18<00:00, 362.21it/s]
 99%|
       | 5919/6000 [00:18<00:00, 356.11it/s]
 99%|
       | 5956/6000 [00:19<00:00, 348.01it/s]
100%|
       | 5993/6000 [00:19<00:00, 350.32it/s]
100%|
       | 6000/6000 [00:19<00:00, 312.19it/s]

6000
300
```

In [102]:

```
#tfidf weighted w2v vectorizing project title
title_train_tfidfw2v=[]
for sentence in tqdm(X_train['project_title']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if(word in glove_words)and(word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector+=(vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    title_train_tfidfw2v.append(vector)
print(len(title_train_tfidfw2v))
print(len(title_train_tfidfw2v[0]))
```

```
  0%|
| 0/9800 [00:00<?, ?it/s]
 75%|
| 7379/9800 [00:00<00:00, 70969.27it/s]
100%|
| 9800/9800 [00:00<00:00, 68072.55it/s]

9800
300
```

In [103]:

```
title_cv_tfidfw2v=[]
for sentence in tqdm(X_cv['project_title']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector+=(vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    title_cv_tfidfw2v.append(vector)
print(len(title_cv_tfidfw2v))
print(len(title_cv_tfidfw2v[0]))
```

```
  0%|
| 0/4200 [00:00<?, ?it/s]
100%|
| 4200/4200 [00:00<00:00, 65640.27it/s]

4200
300
```

In [104]:

```
title_test_tfidfw2v=[]
for sentence in tqdm(X_test['project_title']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if (word in glove_words)and(word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector+=(vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    title_test_tfidfw2v.append(vector)

print(len(title_test_tfidfw2v))
print(len(title_test_tfidfw2v[0]))
```

```
  0%|
| 0/6000 [00:00<?, ?it/s]
100%|
| 6000/6000 [00:00<00:00, 83357.87it/s]

6000
300
```

In [105]:

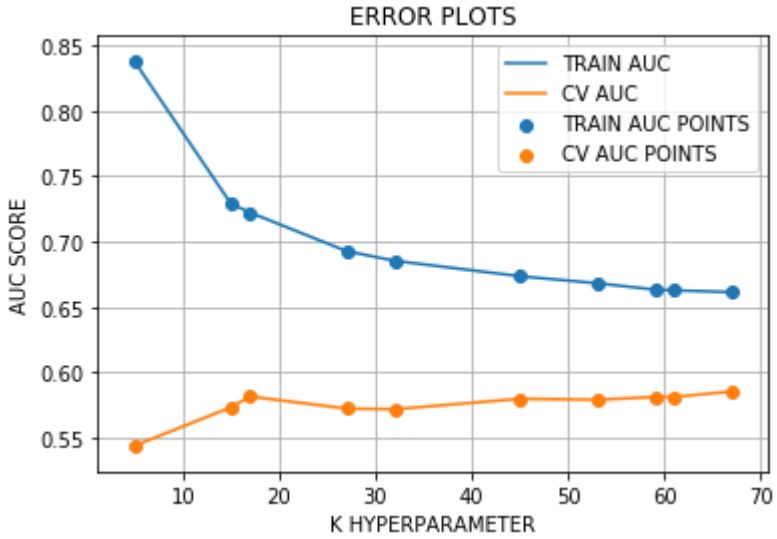
```
#creating data matrix
from scipy.sparse import hstack
final_train_tfidfw2v=hstack((X_train_state_encoded,X_train_prefix_encoded,X_train_grade
_encoded,X_train_categories_encoded,X_train_subcategories_encoded,X_train_price_norm,X_
train_projects_norm,X_train_quantity_norm,tfidf_w2v_train,title_train_tfidfw2v)).tocsr
()
final_cv_tfidfw2v=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_c
v_categories_encoded,X_cv_subcategories_encoded,X_cv_price_norm,X_cv_projects_norm,X_cv
_quantity_norm,tfidf_w2v_cv,title_cv_tfidfw2v)).tocsr()
final_test_tfidfw2v=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_enc
oded,X_test_categories_encoded,X_test_subcategories_encoded,X_test_price_norm,X_test_pr
ojects_norm,X_test_quantity_norm,tfidf_w2v_test,title_test_tfidfw2v)).tocsr()
print(final_train_tfidfw2v.shape,y_train.shape)
print(final_cv_tfidfw2v.shape,y_cv.shape)
print(final_test_tfidfw2v.shape,y_test.shape)
```

```
(9800, 700) (9800,)
(4200, 700) (4200,)
(6000, 700) (6000,)
```

In [106]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc=[]
cv_auc=[]
k=[5,15,17,27,32,45,53,59,61,67]
for i in tqdm(k):
    neigh=KNeighborsClassifier(n_neighbors=i)
    neigh.fit(final_train_tfidfw2v,y_train)

    y_train_pred=batch_predict(neigh,final_train_tfidfw2v)
    y_cv_pred=batch_predict(neigh,final_cv_tfidfw2v)

    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 SCORE')
plt.title('ERROR PLOTS')
plt.grid()
plt.show()
```

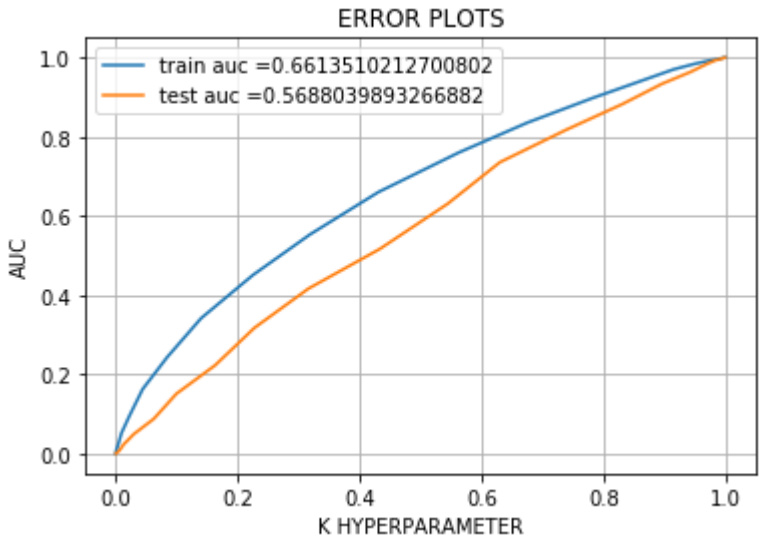
```
  0%|
| 0/10 [00:00<?, ?it/s]
 10%|
| 1/10 [01:57<17:37, 117.48s/it]
 20%|
| 2/10 [03:54<15:39, 117.38s/it]
 30%|
| 3/10 [05:52<13:41, 117.42s/it]
 40%|
| 4/10 [07:49<11:44, 117.40s/it]
 50%|
| 5/10 [09:46<09:46, 117.38s/it]
 60%|
| 6/10 [11:44<07:49, 117.48s/it]
 70%|
| 7/10 [13:42<05:52, 117.52s/it]
 80%|
| 8/10 [15:39<03:55, 117.59s/it]
 90%|
     | 9/10 [17:37<01:57, 117.56s/it]
100%|
          | 10/10 [19:34<00:00, 117.48s/it]
```



OBSERVATIONS: BY using tfidf weighted avgw2v model and plotting error plots we choose best k to be 67.

In [107]:

```
best_k=67
from sklearn.metrics import roc_curve,auc

neigh=KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(final_train_tfidfw2v,y_train)

y_train_pred=batch_predict(neigh,final_train_tfidfw2v)
y_test_pred=batch_predict(neigh,final_test_tfidfw2v)

train_fpr,train_tpr,tr_threshold=roc_curve(y_train,y_train_pred)
test_fpr,test_tpr,te_threshold=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('K HYPERPARAMETER')
plt.ylabel('AUC')
plt.title('ERROR PLOTS')
plt.grid()
plt.show()
```



OBSERVATIONS: For k=67 we got train auc score of 66.13% and test auc score of 56.80%.

In [108]:

```
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,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)))
```

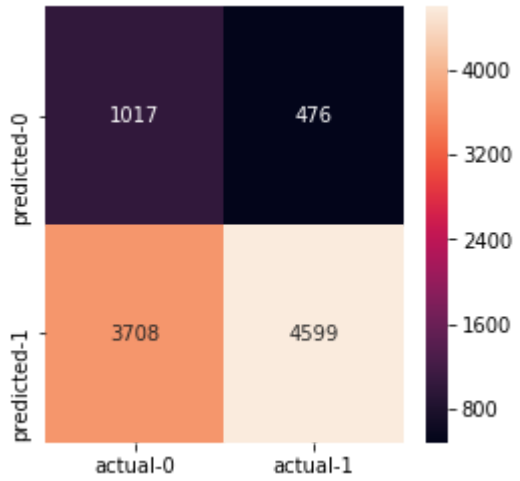
```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.3771206765555982 for threshold 0.851
TRAIN CONFUSION MATRIX
[[1017  476]
 [3708 4599]]
test confusion matrix
[[ 518  396]
 [2457 2629]]
```

In [109]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[1017,476],[3708,4599]]

train=pd.DataFrame(array,index=['predicted-0','predicted-1'],columns=['actual-0','actua
l-1'])
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[109]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18db4c7f240>
```

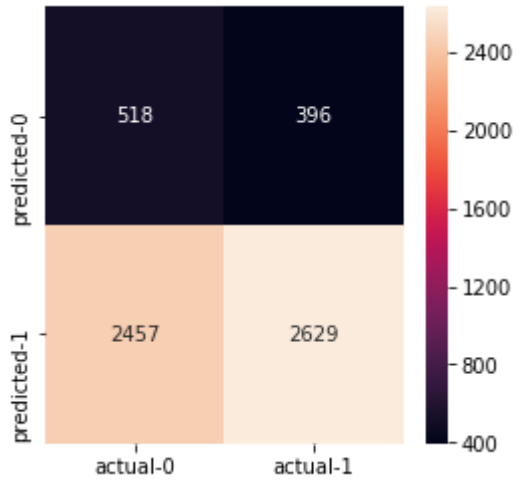


In [110]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

testarray=[[518,396],[2457,2629]]

test=pd.DataFrame(testarray,index=['predicted-0','predicted-1'],columns=['actual-0','ac
tual-1'])
plt.figure(figsize=(4,4))
sn.heatmap(test,annot=True,fmt='d')
```

Out[110]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18db4098ba8>
```



2.5 Feature selection with `SelectKBest`

In [0]:

```
# please write all the code with proper documentation, and proper titles for each subse
ction
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your
code

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [ ]:

```
#selecting 2000 best features from tfidf model.
```

In [111]:

```
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, chi2

X_train_2000=SelectKBest(score_func=chi2,k=2000).fit_transform(final_train_tfidf,y_trai
n)
X_train_2000.shape

X_cv_2000=SelectKBest(score_func=chi2,k=2000).fit_transform(final_cv_tfidf,y_cv)
X_cv_2000.shape

X_test_2000=SelectKBest(score_func=chi2,k=2000).fit_transform(final_test_tfidf,y_test)
X_test_2000.shape
```

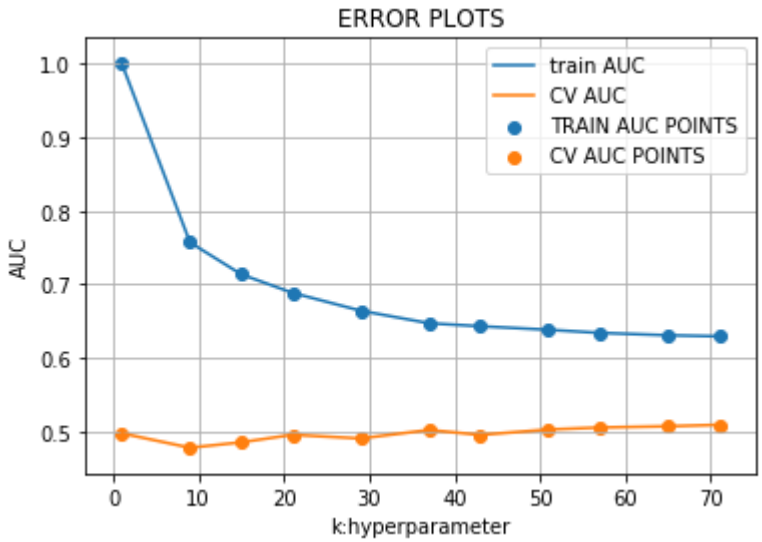
Out[111]:

```
(6000, 2000)
```

In [113]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc=[]
cv_auc=[]
k=[1,9,15,21,29,37,43,51,57,65,71]
for i in tqdm(k):
    neigh=KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train_2000,y_train)


    y_tr_pred=batch_predict(neigh,X_train_2000)
    y_cv_pred=batch_predict(neigh,X_cv_2000)

    train_auc.append(roc_auc_score(y_train,y_tr_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()
```

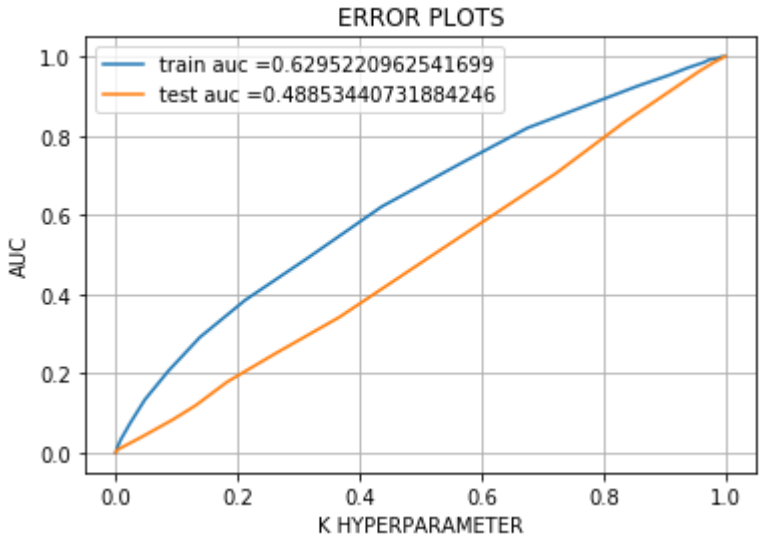
```
  0%|
| 0/11 [00:00<?, ?it/s]

  9%|
| 1/11 [00:08<01:26,  8.66s/it]

 18%|
| 2/11 [00:18<01:21,  9.07s/it]

 27%|
| 3/11 [00:28<01:14,  9.37s/it]

 36%|
| 4/11 [00:38<01:07,  9.60s/it]

 45%|
| 5/11 [00:48<00:58,  9.75s/it]

 55%|
| 6/11 [00:59<00:49,  9.86s/it]

 64%|
| 7/11 [01:09<00:39,  9.95s/it]

 73%|
| 8/11 [01:19<00:30, 10.11s/it]

 82%|
| 9/11 [01:30<00:20, 10.23s/it]

 91%|
      | 10/11 [01:40<00:10, 10.25s/it]

100%|
           | 11/11 [01:50<00:00, 10.26s/it]
```



OBSERVATIONS: We chose the best 2000 features of tfidf model. For those features after plotting error plots we choose our best k to be 71.

In [114]:

```
best_k=71
from sklearn.metrics import roc_curve,auc

neigh=KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_train_2000,y_train)

y_train_pred=batch_predict(neigh,X_train_2000)
y_test_pred=batch_predict(neigh,X_test_2000)

train_fpr,train_tpr,tr_threshold=roc_curve(y_train,y_train_pred)
test_fpr,test_tpr,te_threshold=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('K HYPERPARAMETER')
plt.ylabel('AUC')
plt.title('ERROR PLOTS')
plt.grid()
plt.show()
```



OBSERVATIONS: For k=71 we got a train auc of 62.95% and test auc of 48.85%.

In [115]:

```
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,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)))
```

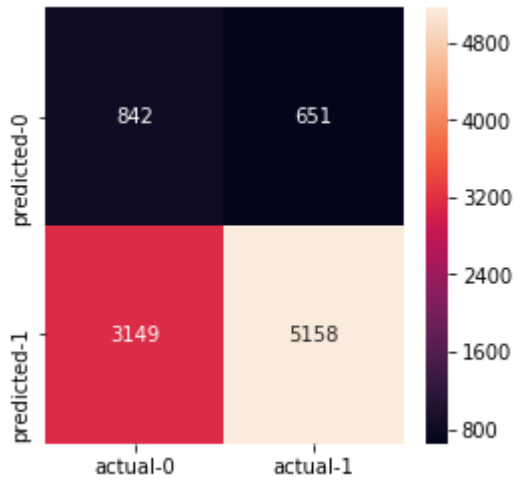
```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.3501784460059227 for threshold 0.845
TRAIN CONFUSION MATRIX
[[ 842  651]
 [3149 5158]]
test confusion matrix
[[ 579  335]
 [3348 1738]]
```

In [116]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[842,651],[3149,5158]]

train=pd.DataFrame(array,index=['predicted-0','predicted-1'],columns=['actual-0','actua
l-1'])
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[116]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18db4c0d0b8>
```

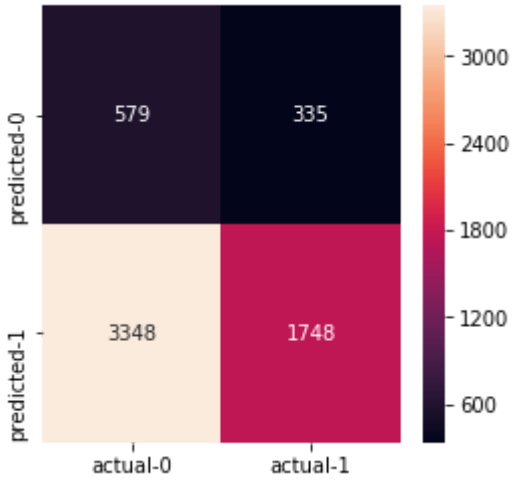


In [117]:

```
#printing heatmap for test confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

testarray=[[579,335],[3348,1748]]

test=pd.DataFrame(testarray,index=['predicted-0','predicted-1'],columns=['actual-0','ac
tual-1'])
plt.figure(figsize=(4,4))
sn.heatmap(test,annot=True,fmt='d')
```

Out[117]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x18db42cb978>
```



3. Conclusions

In [118]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x=PrettyTable(['vectorizer','best_k','train_auc','test_auc'])
x.add_row(["bag of words",71,0.651391,0.577883])
x.add_row(["avgw2v",65,0.645934,0.540197])
x.add_row(["tfidf",69,0.632455,0.542611])
x.add_row(["tfidf_w2v",67,0.661351,0.568803])
x.add_row(['tfidf top 2000',71,0.629522,0.488534])

print(x.get_string(start=0,end=7))
```

```
+----------------+--------+-----------+----------+
|   vectorizer   | best_k | train_auc | test_auc |
+----------------+--------+-----------+----------+
|  bag of words  |   71   |  0.651391 | 0.577883 |
|     avgw2v     |   65   |  0.645934 | 0.540197 |
|     tfidf      |   69   |  0.632455 | 0.542611 |
|   tfidf_w2v    |   67   |  0.661351 | 0.568803 |
| tfidf top 2000 |   71   |  0.629522 | 0.488534 |
+----------------+--------+-----------+----------+
```

CONCLUSIONS: WE PLOTTED 5 MODELS FOR DONORS CHOOSE DATASET. From the summary table we conclude all the vectorizers have approximately similar train and test auc scores. From all the models we plotted bag of words vectorizer has better train auc and test auc scores.