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:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!</code`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p036502`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`
`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 approve

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.

Import some useful Libraries

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")



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

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

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

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

from tqdm import tqdm
import os
```

```
C:\Users\Tarun Makkar\Anaconda3\lib\site-packages\smart_open\ssh.py:34: UserWarning: paramiko missing, opening SSH/SCP/
SFTP paths will be disabled.  `pip install paramiko` to suppress
  warnings.warn('paramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install paramiko` to suppress')
C:\Users\Tarun Makkar\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows; aliasing chunkiz
e to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

1.1 Reading Data

In [2]:
```
import pandas as pd

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

In [3]:
```
project_data.head(3)
```

Out[3]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	I
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Histo
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Grades 6-8	

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

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

In [5]:
```
print("Number of data points in resources data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

```
Number of data points in resources 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

In [6]:
```
# Print some train dataframe

project_data.head(3)
```

Out[6]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	I
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Histo
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Grades 6-8	

1.2 preprocessing of `project_subject_categories`

In [7]:
```
catogories = list(project_data["project_subject_categories"].values)

# remove special characters from list of strings

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

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

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

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

project_data.head()
```

```
{'Warmth': 1388, 'Care_Hunger': 1388, 'History_Civics': 5914, 'Music_Arts': 10293, 'AppliedLearning': 12135, 'SpecialNe
eds': 13642, 'Health_Sports': 14223, 'Math_Science': 41421, 'Literacy_Language': 52239}
```

Out[7]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Civi
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Grades 6-8	Health &
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	2016-10-06 21:16:17	Grades PreK-2	
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	2016-07-11 01:10:09	Grades PreK-2	

1.3 preprocessing of `project_subject_subcategories`

In [8]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

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

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

print(sorted_sub_cat_dict)

project_data.head()
```

```
{'Economics': 269, 'CommunityService': 441, 'FinancialLiteracy': 568, 'ParentInvolvement': 677, 'Extracurricular': 810,
'Civics_Government': 815, 'ForeignLanguages': 890, 'NutritionEducation': 1355, 'Warmth': 1388, 'Care_Hunger': 1388, 'So
cialSciences': 1920, 'PerformingArts': 1961, 'CharacterEducation': 2065, 'TeamSports': 2192, 'Other': 2372, 'College_Ca
reerPrep': 2568, 'Music': 3145, 'History_Geography': 3171, 'Health_LifeScience': 4235, 'EarlyDevelopment': 4254, 'ESL':
4367, 'Gym_Fitness': 4509, 'EnvironmentalScience': 5591, 'VisualArts': 6278, 'Health_Wellness': 10234, 'AppliedScience
s': 10816, 'SpecialNeeds': 13642, 'Literature_Writing': 22179, 'Mathematics': 28074, 'Literacy': 33700}
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	proje
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	Educ Sup Lear
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	V Proje Le
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Grades 6-8	Equipn AWE Middle
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	2016-10-06 21:16:17	Grades PreK-2	Kinderga
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	2016-07-11 01:10:09	Grades PreK-2	Interactiv

1.3 Text preprocessing

[1.3.1] Essays

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	school_state	project_submitted_datetime	project_grade_category	project_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	Educati Suppor Eng Learner Hc
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Wan Projecto Hur Learı

In [11]:
```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

```
My students are English learners that are working on English as their second or third languages. We are a melting pot o
f refugees, immigrants, and native-born Americans bringing the gift of language to our school. \r\n\r\n We have over 24
languages represented in our English Learner program with students at every level of mastery.  We also have over 40 cou
ntries represented with the families within our school.  Each student brings a wealth of knowledge and experiences to u
s that open our eyes to new cultures, beliefs, and respect.\"The limits of your language are the limits of your worl
d.\"-Ludwig Wittgenstein  Our English learner's have a strong support system at home that begs for more resources.  Man
y times our parents are learning to read and speak English along side of their children.  Sometimes this creates barrie
rs for parents to be able to help their child learn phonetics, letter recognition, and other reading skills.\r\n\r\nBy
providing these dvd's and players, students are able to continue their mastery of the English language even if no one a
t home is able to assist.  All families with students within the Level 1 proficiency status, will be a offered to be a
part of this program.  These educational videos will be specially chosen by the English Learner Teacher and will be sen
t home regularly to watch.  The videos are to help the child develop early reading skills.\r\n\r\nParents that do not h
ave access to a dvd player will have the opportunity to check out a dvd player to use for the year.  The plan is to use
these videos and educational dvd's for the years to come for other EL students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this year all love learning, at least most of the tim
e. At our school, 97.3% of the students receive free or reduced price lunch. Of the 560 students, 97.3% are minority st
udents. \r\nThe school has a vibrant community that loves to get together and celebrate. Around Halloween there is a wh
ole school parade to show off the beautiful costumes that students wear. On Cinco de Mayo we put on a big festival with
crafts made by the students, dances, and games. At the end of the year the school hosts a carnival to celebrate the har
d work put in during the school year, with a dunk tank being the most popular activity.My students will use these five
brightly colored Hokki stools in place of regular, stationary, 4-legged chairs. As I will only have a total of ten in t
he classroom and not enough for each student to have an individual one, they will be used in a variety of ways. During
independent reading time they will be used as special chairs students will each use on occasion. I will utilize them in
place of chairs at my small group tables during math and reading times. The rest of the day they will be used by the st
udents who need the highest amount of movement in their life in order to stay focused on school.\r\n\r\nWhenever asked
what the classroom is missing, my students always say more Hokki Stools. They can't get their fill of the 5 stools we a
lready have. When the students are sitting in group with me on the Hokki Stools, they are always moving, but at the sam
e time doing their work. Anytime the students get to pick where they can sit, the Hokki Stools are the first to be take
n. There are always students who head over to the kidney table to get one of the stools who are disappointed as there a
re not enough of them. \r\n\r\nWe ask a lot of students to sit for 7 hours a day. The Hokki stools will be a compromise
that allow my students to do desk work and move at the same time. These stools will help students to meet their 60 minu
tes a day of movement by allowing them to activate their core muscles for balance while they sit. For many of my studen
ts, these chairs will take away the barrier that exists in schools for a child who can't sit still.nannan
==================================================
How do you remember your days of school? Was it in a sterile environment with plain walls, rows of desks, and a teacher
in front of the room? A typical day in our room is nothing like that. I work hard to create a warm inviting themed room
for my students look forward to coming to each day.\r\n\r\nMy class is made up of 28 wonderfully unique boys and girls
of mixed races in Arkansas.\r\nThey attend a Title I school, which means there is a high enough percentage of free and
reduced-price lunch to qualify. Our school is an \"open classroom\" concept, which is very unique as there are no walls
separating the classrooms. These 9 and 10 year-old students are very eager learners; they are like sponges, absorbing a
ll the information and experiences and keep on wanting more.With these resources such as the comfy red throw pillows an
d the whimsical nautical hanging decor and the blue fish nets, I will be able to help create the mood in our classroom
setting to be one of a themed nautical environment. Creating a classroom environment is very important in the success i
n each and every child's education. The nautical photo props will be used with each child as they step foot into our cl
assroom for the first time on Meet the Teacher evening. I'll take pictures of each child with them, have them develope
d, and then hung in our classroom ready for their first day of 4th grade.  This kind gesture will set the tone before e
ven the first day of school! The nautical thank you cards will be used throughout the year by the students as they crea
te thank you cards to their team groups.\r\n\r\nYour generous donations will help me to help make our classroom a fun,
inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of my own pocket on resources to get our
classroom ready. Please consider helping with this project to make our new school year a very successful one. Thank yo
u!nannan
==================================================
My kindergarten students have varied disabilities ranging from speech and language delays, cognitive delays, gross/fine
motor delays, to autism. They are eager beavers and always strive to work their hardest working past their limitations.
\r\n\r\nThe materials we have are the ones I seek out for my students. I teach in a Title I school where most of the st
udents receive free or reduced price lunch.  Despite their disabilities and limitations, my students love coming to sch
ool and come eager to learn and explore.Have you ever felt like you had ants in your pants and you needed to groove and
move as you were in a meeting? This is how my kids feel all the time. The want to be able to move as they learn or so t
hey say.Wobble chairs are the answer and I love then because they develop their core, which enhances gross motor and in
Turn fine motor skills. \r\nThey also want to learn through games, my kids don't want to sit and do worksheets. They wa
nt to learn to count by jumping and playing. Physical engagement is the key to our success. The number toss and color a
nd shape mats can make that happen. My students will forget they are doing work and just have the fun a 6 year old dese
rves.nannan
==================================================

The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates. The great teacher inspires. -
William A. Ward\r\n\r\nMy school has 803 students which is makeup is 97.6% African-American, making up the largest segm
ent of the student body. A typical school in Dallas is made up of 23.2% African-American students. Most of the students
are on free or reduced lunch. We aren't receiving doctors, lawyers, or engineers children from rich backgrounds or neig
hborhoods. As an educator I am inspiring minds of young children and we focus not only on academics but one smart, effe
ctive, efficient, and disciplined students with good character.In our classroom we can utilize the Bluetooth for swift
transitions during class. I use a speaker which doesn't amplify the sound enough to receive the message. Due to the vol
ume of my speaker my students can't hear videos or books clearly and it isn't making the lessons as meaningful. But wit
h the bluetooth speaker my students will be able to hear and I can stop, pause and replay it at any time.\r\nThe cart w
ill allow me to have more room for storage of things that are needed for the day and has an extra part to it I can use.
```

```
The table top chart has all of the letter, words and pictures for students to learn about different letters and it is m
ore accessible.nannan
==================================================
```

In [12]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [13]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

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

In [14]:
```
# \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)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays, cognitive delays, gross/fine
motor delays, to autism. They are eager beavers and always strive to work their hardest working past their limitations.
The materials we have are the ones I seek out for my students. I teach in a Title I school where most of the students r
eceive free or reduced price lunch.  Despite their disabilities and limitations, my students love coming to school and
come eager to learn and explore.Have you ever felt like you had ants in your pants and you needed to groove and move as
you were in a meeting? This is how my kids feel all the time. The want to be able to move as they learn or so they say.
Wobble chairs are the answer and I love then because they develop their core, which enhances gross motor and in Turn fi
ne motor skills.   They also want to learn through games, my kids do not want to sit and do worksheets. They want to le
arn to count by jumping and playing. Physical engagement is the key to our success. The number toss and color and shape
mats can make that happen. My students will forget they are doing work and just have the fun a 6 year old deserves.nann
an
```

In [15]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays cognitive delays gross fine m
otor delays to autism They are eager beavers and always strive to work their hardest working past their limitations The
materials we have are the ones I seek out for my students I teach in a Title I school where most of the students receiv
e free or reduced price lunch Despite their disabilities and limitations my students love coming to school and come eag
er to learn and explore Have you ever felt like you had ants in your pants and you needed to groove and move as you wer
e in a meeting This is how my kids feel all the time The want to be able to move as they learn or so they say Wobble ch
airs are the answer and I love then because they develop their core which enhances gross motor and in Turn fine motor s
kills They also want to learn through games my kids do not want to sit and do worksheets They want to learn to count by
jumping and playing Physical engagement is the key to our success The number toss and color and shape mats can make tha
t happen My students will forget they are doing work and just have the fun a 6 year old deserves nannan
```

In [16]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'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 [17]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████| 109248/109248 [00:47<00:00, 2306.32it/s]
```

In [18]:
```
project_data["cleaned_essay"] = preprocessed_essays
```

[1.3.2] Title

In [19]:
```


# Data preprocessing on title text
from tqdm import tqdm
import re
import string
from bs4 import BeautifulSoup
preprocessed_title_text = []
# 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)
    sent = re.sub("\S*\d\S*", "", sent).strip()
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_title_text.append(sent.lower().strip())
```

```
100%|██████████| 109248/109248 [00:02<00:00, 39835.12it/s]
```

In [20]:
```
project_data = pd.DataFrame(project_data)
project_data['cleaned_title_text'] = preprocessed_title_text

project_data.head(2)
```

Out[20]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	Educati Suppor Eng Learner Hc
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Wan Projecto Hur Learı

[1.4] Process Resource Data

In [21]:
```
# we get the cost of the project using resource.csv file
resource_data.head(2)
```

Out[21]:

	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

In [22]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)
```

Out[22]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [23]:
```
# Check for Null values in price data
price_data.isnull().any().any()
```

Out[23]:
```
False
```

In [24]:
```
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')

project_data.head(2)
```

Out[24]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	Educati Suppor Eng Learner H
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Wan Projecto Hur Learı

Join train & Resource dataset

In [25]:
```
# join two dataframes in python:
data = pd.merge(project_data, price_data, on='id', how='left')
```

In [26]:
```
approved_price = data[data['project_is_approved']==1]['price'].values

rejected_price = data[data['project_is_approved']==0]['price'].values
```

In [27]:
```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
import numpy as np


t = PrettyTable()
t.field_names = ["Percentile", "Approved Projects", "Not Approved Projects"]

for i in range(0,101,5):
    t.add_row([i,np.round(np.percentile(approved_price,i), 3), np.round(np.percentile(rejected_price,i), 3)])
print(t)
```

```
+------------+-------------------+-----------------------+
| Percentile | Approved Projects | Not Approved Projects |
+------------+-------------------+-----------------------+
|     0      |        0.66       |          1.97         |
|     5      |       13.59       |          41.9         |
|     10     |       33.88       |         73.67         |
|     15     |        58.0       |         99.109        |
|     20     |       77.38       |         118.56        |
|     25     |       99.95       |        140.892        |
|     30     |       116.68      |         162.23        |
|     35     |      137.232      |        184.014        |
|     40     |       157.0       |        208.632        |
|     45     |      178.265      |        235.106        |
|     50     |       198.99      |        263.145        |
|     55     |       223.99      |         292.61        |
|     60     |       255.63      |        325.144        |
|     65     |       285.412     |         362.39        |
|     70     |       321.225     |         399.99        |
|     75     |       366.075     |        449.945        |
|     80     |       411.67      |        519.282        |
|     85     |       479.0       |        618.276        |
|     90     |       593.11      |        739.356        |
|     95     |       801.598     |        992.486        |
|    100     |       9999.0      |         9999.0        |
+------------+-------------------+-----------------------+
```

In [28]:
```
data.head(2)
```

Out[28]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	Educati Suppor Eng Learne Hc
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Wan Projecto Hur Learı

2 rows × 22 columns

Train Test split

In [29]:
```
print("Shape of data is : ",data.shape)
project_data["project_is_approved"].value_counts()
```

```
Shape of data is :  (109248, 22)
```

Out[29]:
```
1    92706
0    16542
Name: project_is_approved, dtype: int64
```

In [30]:
```
data = data.sample(n=50000)
```

In [31]:
```
# Define x & y for splitting

y=data['project_is_approved'].values
data.drop(['project_is_approved'], axis=1, inplace=True)      # drop project is approved columns

x=data
```

In [32]:
```
# break in train test

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.2,random_state=2,stratify = y)

# now break trainig data further in train and cv
#x_train,x_cv,y_train,y_cv= train_test_split(x_train, y_train, test_size=0.3 ,random_state=2,stratify=y_train)
```

One Hot Encoding of Categorical Data

In [33]:
```
# OHE of subject category
from sklearn.feature_extraction.text import CountVectorizer
vectorizer1 = CountVectorizer()
vectorizer1.fit(x_train['clean_categories'].values) # fit has to happen only on train data


# we use the fitted CountVectorizer to convert the text to vector
x_train_clean_cat_ohe = vectorizer1.transform(x_train['clean_categories'].values)
#x_cv_clean_cat_ohe = vectorizer.transform(x_cv['clean_categories'].values)
x_test_clean_cat_ohe = vectorizer1.transform(x_test['clean_categories'].values)

print("After vectorizations")
print(x_train_clean_cat_ohe.shape, y_train.shape)
#print(x_cv_clean_cat_ohe.shape, y_cv.shape)
print(x_test_clean_cat_ohe.shape, y_test.shape)
print(vectorizer1.get_feature_names())
print("="*100)
```

```
After vectorizations
(40000, 9) (40000,)
(10000, 9) (10000,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_art
s', 'specialneeds', 'warmth']
====================================================================================================
```

In [34]:
```
# ONE of subject subcategory

vectorizer2 = CountVectorizer()
vectorizer2.fit(x_train['clean_subcategories'].values) # fit has to happen only on train data


# we use the fitted CountVectorizer to convert the text to vector
x_train_clean_subcat_ohe = vectorizer2.transform(x_train['clean_subcategories'].values)
#x_cv_clean_subcat_ohe = vectorizer.transform(x_cv['clean_subcategories'].values)
x_test_clean_subcat_ohe = vectorizer2.transform(x_test['clean_subcategories'].values)

print("After vectorizations")
print(x_train_clean_subcat_ohe.shape, y_train.shape)
#print(x_cv_clean_cat_ohe.shape, y_cv.shape)
print(x_test_clean_subcat_ohe.shape, y_test.shape)
print(vectorizer2.get_feature_names())
print("="*100)
```

```
After vectorizations
(40000, 30) (40000,)
(10000, 30) (10000,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice',
'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlanguag
es', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'ma
thematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialsciences', 'specialne
eds', 'teamsports', 'visualarts', 'warmth']
====================================================================================================
```

In [35]:
```
# one hot encoding the catogorical features: categorical_categories
# teacher_prefix

vectorizer3 = CountVectorizer()
vectorizer3.fit(x_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
x_train_teacher_pre = vectorizer3.transform(x_train['teacher_prefix'].values)

x_test_teacher_pre = vectorizer3.transform(x_test['teacher_prefix'].values)

print("After vectorizations")
print(x_train_teacher_pre.shape, y_train.shape)
#print(x_cv_teacher_pre.shape, y_cv.shape)
print(x_test_teacher_pre.shape, y_test.shape)
print(vectorizer3.get_feature_names())
print("="*100)
```

```
After vectorizations
(40000, 6) (40000,)
(10000, 6) (10000,)
['dr', 'mr', 'mrs', 'ms', 'null', 'teacher']
====================================================================================================
```

In [36]:
```
# school_state

vectorizer4 = CountVectorizer()
vectorizer4.fit(x_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
x_train_state_ohe = vectorizer4.transform(x_train['school_state'].values)
#x_cv_state_ohe = vectorizer.transform(x_cv['school_state'].values)
x_test_state_ohe = vectorizer4.transform(x_test['school_state'].values)

print("After vectorizations")
print(x_train_state_ohe.shape, y_train.shape)
#print(x_cv_state_ohe.shape, y_cv.shape)
print(x_test_state_ohe.shape, y_test.shape)
print(vectorizer4.get_feature_names())
print("="*100)
```

```
After vectorizations
(40000, 51) (40000,)
(10000, 51) (10000,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'm
a', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa',
'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
====================================================================================================
```

In [37]:
```
project_grade_category= x_train['project_grade_category'].unique()
```

In [38]:
```
vectorizer5 = CountVectorizer(vocabulary=list(project_grade_category), lowercase=False, binary=True)
vectorizer5.fit(x_train['project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
x_train_grade_ohe = vectorizer5.transform(x_train['project_grade_category'].values)
#x_cv_grade_ohe = vectorizer.transform(x_cv['project_grade_category'].values)
x_test_grade_ohe = vectorizer5.transform(x_test['project_grade_category'].values)

print("After vectorizations")
print(x_train_grade_ohe.shape, y_train.shape)
#print(x_cv_grade_ohe.shape, y_cv.shape)
print(x_test_grade_ohe.shape, y_test.shape)
print(vectorizer5.get_feature_names())
print("="*100)
```

```
After vectorizations
(40000, 4) (40000,)
(10000, 4) (10000,)
['Grades 3-5', 'Grades 9-12', 'Grades 6-8', 'Grades PreK-2']
====================================================================================================
```

Standardize Numerical data

In [39]:
```
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_test_price_norm = normalizer.transform(x_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(x_train_price_norm.shape, y_train.shape)

print(x_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(40000, 1) (40000,)
(10000, 1) (10000,)
====================================================================================================
```

In [40]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(x_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

x_train_teacher_previously_norm = normalizer.transform(x_train['teacher_number_of_previously_posted_projects'].values.re

x_test_teacher_previously_norm = normalizer.transform(x_test['teacher_number_of_previously_posted_projects'].values.resh

print("After vectorizations")
print(x_train_teacher_previously_norm.shape, y_train.shape)

print(x_test_teacher_previously_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(40000, 1) (40000,)
(10000, 1) (10000,)
====================================================================================================
```

In [41]:
```
from sklearn.preprocessing import Normalizer
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_test_quantity_norm = normalizer.transform(x_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(x_train_quantity_norm.shape, y_train.shape)

print(x_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(40000, 1) (40000,)
(10000, 1) (10000,)
====================================================================================================
```

Set 1 : Apply BOW

In [42]:
```
from sklearn.feature_extraction.text import CountVectorizer
# Vectorizing text data
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer7 = CountVectorizer(min_df=10,ngram_range=(1,2),max_features=10000)
vectorizer7.fit(x_train["cleaned_essay"].values)

x_train_essay_bow = vectorizer7.transform(x_train['cleaned_essay'].values)
#x_cv_essay_bow = vectorizer.transform(x_cv['cleaned_essays'].values)
x_test_essay_bow = vectorizer7.transform(x_test['cleaned_essay'].values)

print("After vectorizations")
print(x_train_essay_bow.shape, y_train.shape)
#print(x_cv_essay_bow.shape, y_cv.shape)
print(x_test_essay_bow.shape, y_test.shape)
print("="*100)
print(vectorizer7.get_feature_names())
```

```
After vectorizations
(40000, 10000) (40000,)
(10000, 10000) (10000,)
====================================================================================================
['00', '000', '000 students', '10', '10 students', '10 years', '100', '100 free', '100 percent', '100 students', '10
th', '11', '11th', '12', '120', '12th', '12th grade', '13', '14', '15', '15 minutes', '150', '16', '17', '18', '19',
'1st', '1st grade', '1st graders', '20', '20 minutes', '20 students', '20 years', '200', '200 students', '2015', '20
16', '2016 2017', '2017', '2017 school', '21', '21 students', '21st', '21st century', '22', '22 students', '23', '23
students', '24', '24 students', '25', '25 students', '26', '27', '28', '29', '2nd', '2nd grade', '2nd graders', '3
0', '30 minutes', '30 students', '300', '300 students', '32', '35', '3d', '3d printer', '3d printing', '3doodler',
'3rd', '3rd 4th', '3rd grade', '3rd graders', '40', '40 students', '400', '400 students', '45', '45 minutes', '450',
'4th', '4th 5th', '4th grade', '4th graders', '50', '50 students', '500', '500 students', '5th', '5th 6th', '5th gra
de', '5th graders', '60', '60 minutes', '60 students', '600', '600 students', '65', '6th', '6th 7th', '6th grade',
'6th graders', '70', '70 students', '700', '700 students', '75', '75 students', '7th', '7th 8th', '7th grade', '7th
graders', '80', '80 students', '800', '800 students', '85', '85 students', '8th', '8th grade', '8th graders', '90',
'90 students', '900', '900 students', '95', '95 students', '96', '97', '98', '98 students', '99', '9th', '9th grad
e', 'abilities', 'ability', 'ability focus', 'ability learn', 'ability levels', 'ability move', 'ability read', 'abi
lity use', 'ability work', 'able', 'able access', 'able afford', 'able apply', 'able better', 'able bring', 'able bu
ild', 'able choose', 'able come', 'able communicate', 'able complete', 'able concentrate', 'able continue', 'able co
```

In [43]:
```
# BOW on clean_titles

from sklearn.feature_extraction.text import CountVectorizer
vectorizer8 = CountVectorizer(min_df=10,ngram_range=(1,2),max_features=10000)
vectorizer8.fit(x_train['cleaned_title_text'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
x_train_titles_bow = vectorizer8.transform(x_train['cleaned_title_text'].values)
#x_cv_titles_bow = vectorizer.transform(x_cv['cleaned_title_text'].values)
x_test_titles_bow = vectorizer8.transform(x_test['cleaned_title_text'].values)

print("After vectorizations")
print(x_train_titles_bow.shape, y_train.shape)
#print(x_cv_titles_bow.shape, y_cv.shape)
print(x_test_titles_bow.shape, y_test.shape)
print("="*100)
print(vectorizer8.get_feature_names())
```

```
'but', 'butterflies', 'butterfly', 'by', 'ca', 'ca not', 'calculating', 'calculator', 'calculators', 'call', 'callin
g', 'calling all', 'calm', 'calming', 'camera', 'camera action', 'cameras', 'can', 'can be', 'can code', 'can do',
'can hear', 'can help', 'can learn', 'can read', 'can see', 'can we', 'can you', 'canvas', 'captivating', 'capture',
'capturing', 'cardboard', 'care', 'career', 'careers', 'caring', 'carpet', 'carpet ride', 'carpet time', 'cart', 'ca
se', 'cases', 'catch', 'catching', 'cause', 'cd', 'celebrate', 'celebration', 'center', 'center time', 'centered',
'centered classroom', 'centered learning', 'centers', 'century', 'century classroom', 'century learners', 'century l
earning', 'century skills', 'century students', 'century technology', 'ceramics', 'chair', 'chair pockets', 'chair
s', 'challenge', 'challenges', 'challenging', 'change', 'change world', 'changes', 'changing', 'chaos', 'chapter',
'chapter books', 'character', 'characters', 'charge', 'charged', 'charging', 'chart', 'charts', 'check', 'cheer', 'c
heese', 'chemistry', 'chess', 'chevron', 'child', 'childhood', 'children', 'choice', 'choice seating', 'choices', 'c
hoose', 'choose your', 'christmas', 'chrome', 'chrome book', 'chrome books', 'chromebook', 'chromebook for', 'chrome
books', 'chromebooks all', 'chromebooks century', 'chromebooks classroom', 'chromebooks create', 'chromebooks for',
'chromebooks make', 'chromebooks needed', 'chromebooks our', 'chromebooks part', 'circle', 'circle time', 'circles',
'circuits', 'citizens', 'city', 'civil', 'class', 'class library', 'class needs', 'class set', 'classes', 'classic',
'classics', 'classroom', 'classroom carpet', 'classroom chromebooks', 'classroom community', 'classroom environmen
t', 'classroom essentials', 'classroom learning', 'classroom library', 'classroom materials', 'classroom needs', 'cl
assroom project', 'classroom rug', 'classroom seating', 'classroom supplies', 'classroom technology', 'classroom wit
h', 'classrooms', 'clay', 'clean', 'cleaning', 'clear', 'clearly', 'click', 'close', 'close reading', 'closer', 'clo
sing', 'club', 'clubs', 'clutter', 'code', 'coders', 'coding', 'coding our', 'coding with', 'coffee', 'cold', 'colla
borate', 'collaborating', 'collaboration', 'collaboration station', 'collaborative', 'collaborative learning', 'coll
```

In [44]:
```
# CONCATINATE all features of BOW


from scipy.sparse import hstack
X_train_bow = hstack((x_train_essay_bow,x_train_titles_bow,x_train_clean_cat_ohe,x_train_clean_subcat_ohe, x_train_state

X_test_bow = hstack((x_test_essay_bow,x_test_titles_bow,x_test_clean_cat_ohe,x_test_clean_subcat_ohe, x_test_state_ohe,

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

```
Final Data matrix
(40000, 13239) (40000,)
(10000, 13239) (10000,)
====================================================================================================
```

In [45]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

params ={ 'n_neighbors': [1, 5, 10, 15, 21, 31, 41, 51, 60]}

estimator1 = KNeighborsClassifier()
Research1 =GridSearchCV(estimator1,param_grid = params,cv= 5,scoring = 'roc_auc',verbose=22,n_jobs=4)
Research1.fit(X_train_bow,y_train)
```

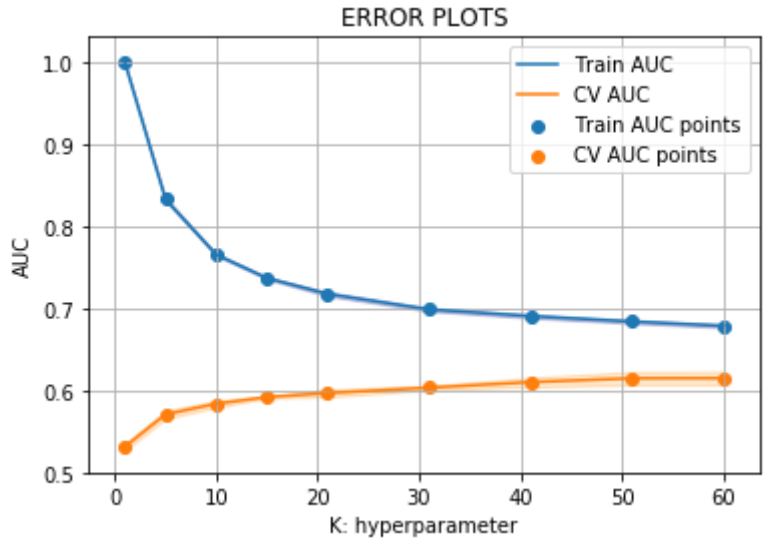
```
Fitting 5 folds for each of 9 candidates, totalling 45 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   1 tasks      | elapsed:  3.9min
[Parallel(n_jobs=4)]: Done   2 tasks      | elapsed:  4.0min
[Parallel(n_jobs=4)]: Done   3 tasks      | elapsed:  4.1min
[Parallel(n_jobs=4)]: Done   4 tasks      | elapsed:  4.1min
[Parallel(n_jobs=4)]: Done   5 tasks      | elapsed:  7.2min
[Parallel(n_jobs=4)]: Done   6 tasks      | elapsed:  7.4min
[Parallel(n_jobs=4)]: Done   7 tasks      | elapsed:  7.5min
[Parallel(n_jobs=4)]: Done   8 tasks      | elapsed:  7.6min
[Parallel(n_jobs=4)]: Done   9 tasks      | elapsed: 10.5min
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed: 10.8min
[Parallel(n_jobs=4)]: Done  11 tasks      | elapsed: 10.9min
[Parallel(n_jobs=4)]: Done  12 tasks      | elapsed: 11.0min
[Parallel(n_jobs=4)]: Done  13 tasks      | elapsed: 13.9min
[Parallel(n_jobs=4)]: Done  14 tasks      | elapsed: 14.1min
[Parallel(n_jobs=4)]: Done  15 tasks      | elapsed: 14.2min
[Parallel(n_jobs=4)]: Done  16 tasks      | elapsed: 14.3min
[Parallel(n_jobs=4)]: Done  17 tasks      | elapsed: 17.4min
```

In [47]:
```
print(Research1.best_score_)
n1=Research1.best_params_['n_neighbors']
print('best  k = ',n1)
```

```
0.6158171196398929
best  k =  60
```

Performance Plot

In [48]:
```
train_auc1= Research1.cv_results_['mean_train_score']
train_auc_std1= Research1.cv_results_['std_train_score']
cv_auc1 = Research1.cv_results_['mean_test_score']
cv_auc_std1= Research1.cv_results_['std_test_score']

plt.plot(params['n_neighbors'], train_auc1, label='Train AUC')

# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(params['n_neighbors'],train_auc1 - train_auc_std1,train_auc1 + train_auc_std1,alpha=0.2,color='da
# create a shaded area between [mean - std, mean + std]

plt.plot(params['n_neighbors'], cv_auc1, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(params['n_neighbors'],cv_auc1 - cv_auc_std1,cv_auc1 + cv_auc_std1,alpha=0.2,color='darkorange')

plt.scatter(params['n_neighbors'], train_auc1, label='Train AUC points')
plt.scatter(params['n_neighbors'], cv_auc1, label='CV AUC points')


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



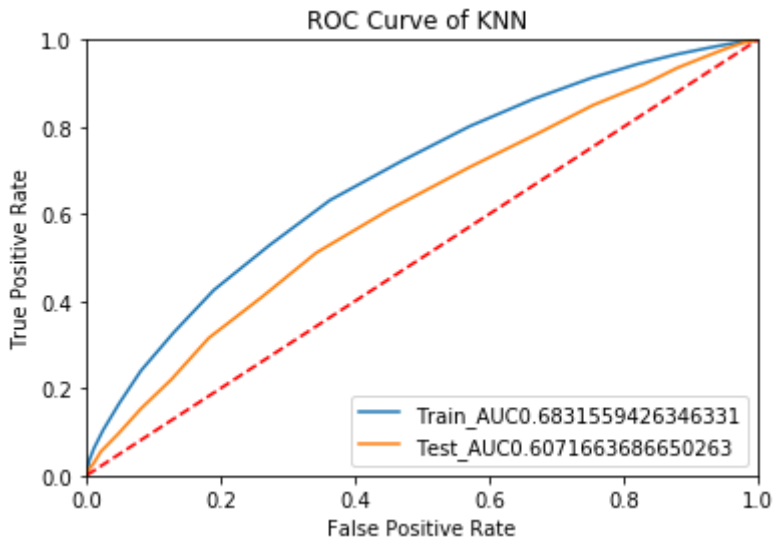
Train new model on best params

In [47]:
```
from sklearn.neighbors import KNeighborsClassifier

model_new1 = KNeighborsClassifier(n_neighbors = n1)
model_new1.fit(X_train_bow,y_train)
```

Out[47]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=60, p=2,
           weights='uniform')
```

ROC Curve

In [48]:
```
from  sklearn.metrics import roc_curve
from sklearn.metrics import auc
import matplotlib.pyplot as plt


score_roc_train = model_new1.predict_proba(X_train_bow)
fpr_train, tpr_train, threshold_train = roc_curve(y_train, score_roc_train[:,1])
roc_auc_train = auc(fpr_train, tpr_train)

score_roc_test = model_new1.predict_proba(X_test_bow)
fpr_test, tpr_test, threshold_test = roc_curve(y_test, score_roc_test[:,1])
roc_auc_test = auc(fpr_test, tpr_test)


plt.plot(fpr_train, tpr_train, label = "Train_AUC"+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label = "Test_AUC"+str(auc(fpr_test, tpr_test)))
plt.legend(loc = 'lower right')

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

plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of KNN ')
plt.show()
```



Confusion_Matrix

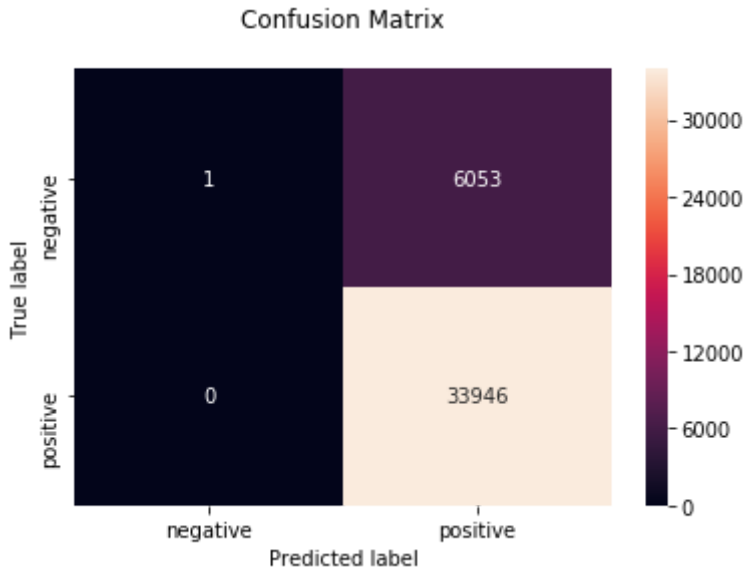
In [49]:
```
y_train_pred = model_new1.predict(X_train_bow)

y_test_pred = model_new1.predict(X_test_bow)
```

In [50]:
```
# we are defining are own function for use probabilities to plot confusion matrix
# we have to plot confusion matrix at least fpr and high tpr values

def predict(proba,threshold,fpr,tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))


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

In [51]:
```
from sklearn.metrics import confusion_matrix

ax = plt.subplot()

print("Train confusion matrix")
cnn_train = confusion_matrix(y_train, predict(y_train_pred, threshold_train, fpr_train, tpr_train))
sns.heatmap(cnn_train,annot = True,ax=ax,fmt='d')




ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.xaxis.set_ticklabels(['negative','positive'])
ax.yaxis.set_ticklabels(['negative','positive'])
plt.title('Confusion Matrix\n')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.4025260897956219 for threshold 0.8
```

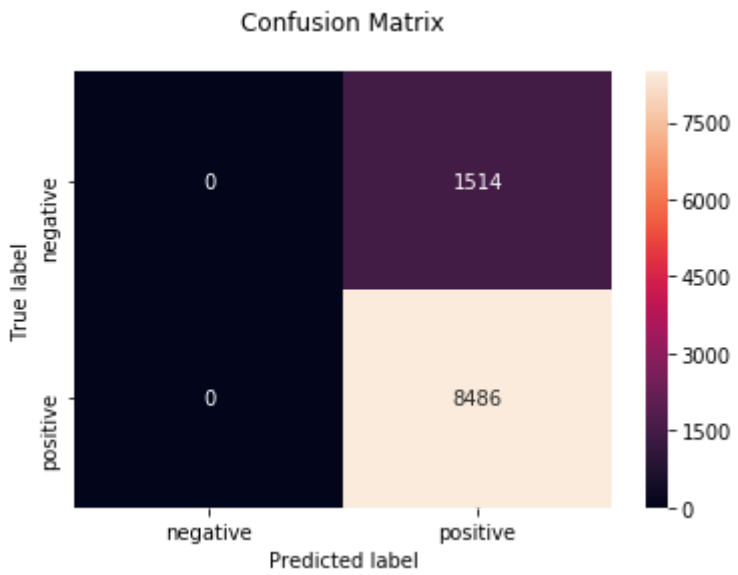
Out[51]:
```
Text(0.5, 1.0, 'Confusion Matrix\n')
```



In [52]:
```
ax = plt.subplot()

print("Test confusion matrix")
cnn_test = confusion_matrix(y_test, predict(y_test_pred, threshold_test, fpr_test, tpr_test))
sns.heatmap(cnn_test,annot = True,ax=ax,fmt='d')

ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.xaxis.set_ticklabels(['negative','positive'])
ax.yaxis.set_ticklabels(['negative','positive'])
plt.title('Confusion Matrix\n')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3360621005737634 for threshold 0.817
```

Out[52]:
```
Text(0.5, 1.0, 'Confusion Matrix\n')
```



Classification Report

In [53]:
```
from sklearn.metrics import classification_report
print("_" * 101)
print("Classification Report: \n")
print(classification_report(y_test,y_test_pred))
print("_" * 101)
```

```
_____________________________________________________________________________________________________
Classification Report:


C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)

             precision    recall  f1-score   support

          0       0.00      0.00      0.00      1514
          1       0.85      1.00      0.92      8486

avg / total       0.85      0.85      0.85     10000
   micro avg       0.85      0.85      0.85     10000
   macro avg       0.42      0.50      0.46     10000
weighted avg       0.72      0.85      0.78     10000

_____________________________________________________________________________________________________
```

SET 2 : TF-IDF

In [54]:
```
# On Clean Essay

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer8 = TfidfVectorizer(min_df=10,ngram_range = (1,2),max_features=10000)
preprocessed_essays_xtr_tfidf = vectorizer8.fit_transform(x_train['cleaned_essay'])
print("Shape of matrix after one hot encodig ",preprocessed_essays_xtr_tfidf.shape)

preprocessed_essays_xtest_tfidf = vectorizer8.transform(x_test['cleaned_essay'])
print("Shape of matrix after one hot encodig ",preprocessed_essays_xtest_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (40000, 10000)
Shape of matrix after one hot encodig  (10000, 10000)
```

In [55]:
```
# On Clean_title

vectorizer9 = TfidfVectorizer(min_df=10,ngram_range = (1,2),max_features=10000)
preprocessed_title_xtr_tfidf = vectorizer9.fit_transform(x_train['cleaned_title_text'])
print("Shape of matrix after one hot encodig ",preprocessed_title_xtr_tfidf.shape)

preprocessed_title_xtest_tfidf = vectorizer9.transform(x_test['cleaned_title_text'])
print("Shape of matrix after one hot encodig ",preprocessed_title_xtest_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (40000, 3136)
Shape of matrix after one hot encodig  (10000, 3136)
```

In [93]:
```
# Concatenate TFIDF

X_train_tfidf=hstack((preprocessed_essays_xtr_tfidf,preprocessed_title_xtr_tfidf,x_train_clean_cat_ohe,x_train_clean_sub
                      ,x_train_quantity_norm )).tocsr()
#X_cv_tfidf=hstack((preprocessed_essays_xcv_tfidf,preprocessed_title_xcv_tfidf,x_cv_clean_cat_ohe,x_cv_clean_subcat_ohe,

X_test_tfidf=hstack((preprocessed_essays_xtest_tfidf,preprocessed_title_xtest_tfidf,x_test_clean_cat_ohe,x_test_clean_su

                    ,x_test_quantity_norm   )).tocsr()
```

In [49]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

params ={ 'n_neighbors': [1, 5, 10, 15, 21, 31, 41, 51, 60]}

estimator2 = KNeighborsClassifier()
Research2 =GridSearchCV(estimator2 ,param_grid = params,cv= 5 ,scoring = 'roc_auc',verbose=22,n_jobs=3)
Research2.fit(X_train_tfidf,y_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits

[Parallel(n_jobs=3)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=3)]: Done   1 tasks      | elapsed:  3.5min
[Parallel(n_jobs=3)]: Done   2 tasks      | elapsed:  3.6min
[Parallel(n_jobs=3)]: Done   3 tasks      | elapsed:  3.6min
[Parallel(n_jobs=3)]: Done   4 tasks      | elapsed:  6.3min
[Parallel(n_jobs=3)]: Done   5 tasks      | elapsed:  6.4min
[Parallel(n_jobs=3)]: Done   6 tasks      | elapsed:  6.6min
[Parallel(n_jobs=3)]: Done   7 tasks      | elapsed:  9.0min
[Parallel(n_jobs=3)]: Done   8 tasks      | elapsed:  9.1min
[Parallel(n_jobs=3)]: Done   9 tasks      | elapsed:  9.3min
[Parallel(n_jobs=3)]: Done  10 tasks      | elapsed: 12.1min
[Parallel(n_jobs=3)]: Done  11 tasks      | elapsed: 12.2min
[Parallel(n_jobs=3)]: Done  12 tasks      | elapsed: 12.3min
[Parallel(n_jobs=3)]: Done  13 tasks      | elapsed: 15.1min
[Parallel(n_jobs=3)]: Done  14 tasks      | elapsed: 15.2min
[Parallel(n_jobs=3)]: Done  15 tasks      | elapsed: 15.3min
[Parallel(n_jobs=3)]: Done  16 tasks      | elapsed: 18.1min
[Parallel(n_jobs=3)]: Done  17 tasks      | elapsed: 18.1min
[Parallel(n_jobs=3)]: Done  18 tasks      | elapsed: 18.3min
[Parallel(n_jobs=3)]: Done  19 tasks      | elapsed: 21.1min
[Parallel(n_jobs=3)]: Done  20 tasks      | elapsed: 21.2min
[Parallel(n_jobs=3)]: Done  21 tasks      | elapsed: 21.3min
[Parallel(n_jobs=3)]: Done  22 tasks      | elapsed: 24.2min
[Parallel(n_jobs=3)]: Done  23 tasks      | elapsed: 24.3min
[Parallel(n_jobs=3)]: Done  24 tasks      | elapsed: 24.4min
[Parallel(n_jobs=3)]: Done  25 tasks      | elapsed: 27.2min
[Parallel(n_jobs=3)]: Done  26 tasks      | elapsed: 27.2min
[Parallel(n_jobs=3)]: Done  27 tasks      | elapsed: 27.3min
[Parallel(n_jobs=3)]: Done  28 tasks      | elapsed: 29.5min
[Parallel(n_jobs=3)]: Done  29 tasks      | elapsed: 29.6min
[Parallel(n_jobs=3)]: Done  30 tasks      | elapsed: 29.7min
[Parallel(n_jobs=3)]: Done  31 tasks      | elapsed: 31.8min
[Parallel(n_jobs=3)]: Done  32 tasks      | elapsed: 31.9min
[Parallel(n_jobs=3)]: Done  33 tasks      | elapsed: 32.0min
[Parallel(n_jobs=3)]: Done  34 tasks      | elapsed: 34.2min
[Parallel(n_jobs=3)]: Done  35 tasks      | elapsed: 34.2min
[Parallel(n_jobs=3)]: Done  36 tasks      | elapsed: 34.3min
[Parallel(n_jobs=3)]: Done  37 tasks      | elapsed: 36.5min
[Parallel(n_jobs=3)]: Done  38 tasks      | elapsed: 36.5min
[Parallel(n_jobs=3)]: Done  39 tasks      | elapsed: 36.6min
[Parallel(n_jobs=3)]: Done  40 tasks      | elapsed: 38.8min
[Parallel(n_jobs=3)]: Done  43 out of  45 | elapsed: 41.3min remaining:  1.9min
[Parallel(n_jobs=3)]: Done  45 out of  45 | elapsed: 41.4min finished
```

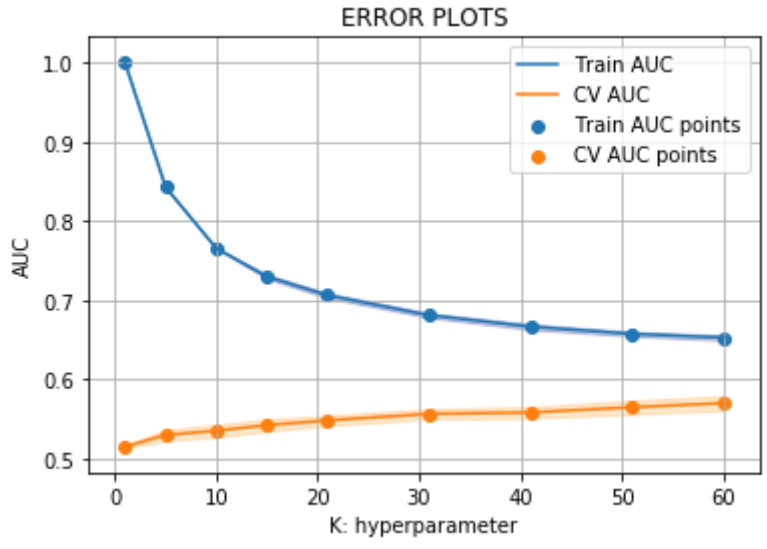
Out[49]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid='warn', n_jobs=3,
       param_grid={'n_neighbors': [1, 5, 10, 15, 21, 31, 41, 51, 60]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=22)
```

In [50]:
```
print(Research2.best_score_)
n2=Research2.best_params_['n_neighbors']
print('best  k = ',n2)
```

```
0.5701250908593548
best  k =  60
```

Performance Plot

In [51]:
```
train_auc2= Research2.cv_results_['mean_train_score']
train_auc_std2= Research2.cv_results_['std_train_score']
cv_auc2 = Research2.cv_results_['mean_test_score']
cv_auc_std2 = Research2.cv_results_['std_test_score']

plt.plot(params['n_neighbors'], train_auc2, label='Train AUC')

# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(params['n_neighbors'],train_auc2 - train_auc_std2,train_auc2 + train_auc_std2,alpha=0.2,color='da
# create a shaded area between [mean - std, mean + std]

plt.plot(params['n_neighbors'], cv_auc2, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(params['n_neighbors'],cv_auc2 - cv_auc_std2,cv_auc2 + cv_auc_std2,alpha=0.2,color='darkorange')

plt.scatter(params['n_neighbors'], train_auc2, label='Train AUC points')
plt.scatter(params['n_neighbors'], cv_auc2, label='CV AUC points')


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



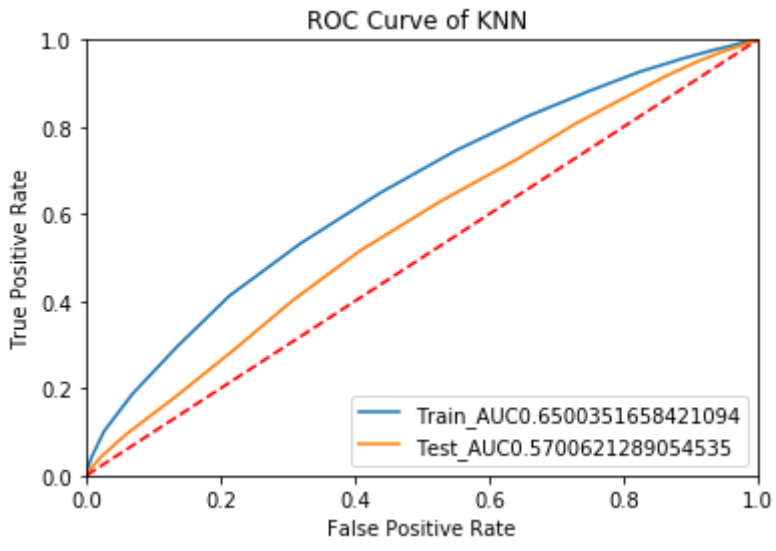
Train new model on best params

In [57]:
```
model_new_tfidf = KNeighborsClassifier(n_neighbors = n2)
model_new_tfidf.fit(X_train_tfidf,y_train)
```

Out[57]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=60, p=2,
           weights='uniform')
```

ROC curve

In [58]:
```
from  sklearn.metrics import roc_curve
from sklearn.metrics import auc
import matplotlib.pyplot as plt


score_roc_train = model_new_tfidf.predict_proba(X_train_tfidf)
fpr_train, tpr_train, threshold_train = roc_curve(y_train, score_roc_train[:,1])
roc_auc_train = auc(fpr_train, tpr_train)

score_roc_test = model_new_tfidf.predict_proba(X_test_tfidf)
fpr_test, tpr_test, threshold_test = roc_curve(y_test, score_roc_test[:,1])
roc_auc_test = auc(fpr_test, tpr_test)


plt.plot(fpr_train, tpr_train, label = "Train_AUC"+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label = "Test_AUC"+str(auc(fpr_test, tpr_test)))
plt.legend(loc = 'lower right')

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

plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of KNN ')
plt.show()
```



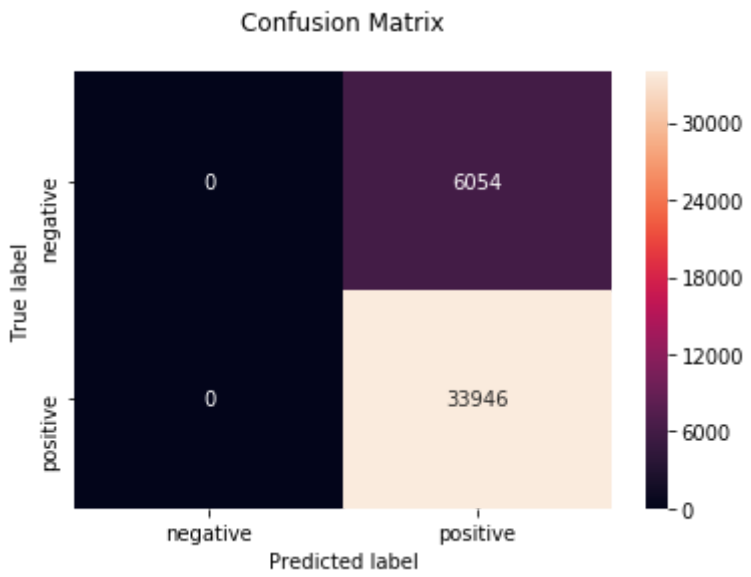
Confusion_Matrix

In [59]:
```
y_train_pred_tfidf = model_new_tfidf.predict(X_train_tfidf)

y_test_pred_tfidf = model_new_tfidf.predict(X_test_tfidf)
```

In [60]:
```
from sklearn.metrics import confusion_matrix

ax = plt.subplot()

print("Train confusion matrix")
cnn_train = confusion_matrix(y_train, predict(y_train_pred_tfidf, threshold_train, fpr_train, tpr_train))
sns.heatmap(cnn_train,annot = True,ax=ax,fmt='d')




ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.xaxis.set_ticklabels(['negative','positive'])
ax.yaxis.set_ticklabels(['negative','positive'])
plt.title('Confusion Matrix\n')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.36479879400367526 for threshold 0.85
```

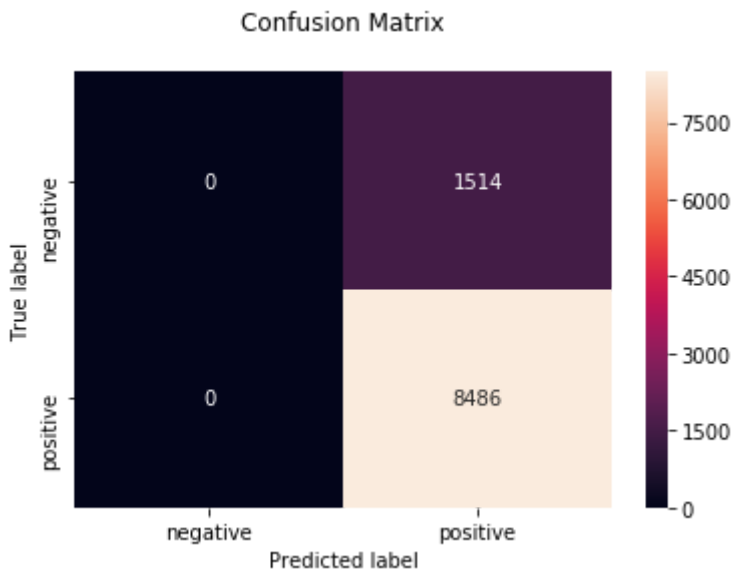
Out[60]:
```
Text(0.5, 1.0, 'Confusion Matrix\n')
```



In [61]:
```
ax = plt.subplot()

print("Test confusion matrix")
cnn_test = confusion_matrix(y_test, predict(y_test_pred_tfidf, threshold_test, fpr_test, tpr_test))
sns.heatmap(cnn_test,annot = True,ax=ax,fmt='d')

ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.xaxis.set_ticklabels(['negative','positive'])
ax.yaxis.set_ticklabels(['negative','positive'])
plt.title('Confusion Matrix\n')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.30551197698844096 for threshold 0.867
```

Out[61]:
```
Text(0.5, 1.0, 'Confusion Matrix\n')
```



Classification Report

In [62]:
```
from sklearn.metrics import classification_report
print("_" * 101)
print("Classification Report: \n")
print(classification_report(y_test,y_test_pred_tfidf))
print("_" * 101)
```

```
_____________________________________________________________________________________________________
Classification Report:


C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)

             precision    recall  f1-score   support

          0       0.00      0.00      0.00      1514
          1       0.85      1.00      0.92      8486

   micro avg       0.85      0.85      0.85     10000
   macro avg       0.42      0.50      0.46     10000
weighted avg       0.72      0.85      0.78     10000

_____________________________________________________________________________________________________
```

SET : 3 [AVG -W2V]

In [63]:
```
list_preprocessed_essays_xtr = []
for e in x_train['cleaned_essay'].values:
    list_preprocessed_essays_xtr.append(e.split())

from gensim.models import Word2Vec
preprocessed_essays_xtr_w2v=Word2Vec(list_preprocessed_essays_xtr,min_count=10,size=100,workers = 8)
```

In [64]:
```
# average Word2Vec
# compute average word2vec for each review.
preprocessed_essays_xtr_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train['cleaned_essay']): # for each review/sentence
    vector = np.zeros(100) # 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 preprocessed_essays_xtr_w2v.wv.vocab:
            vector += preprocessed_essays_xtr_w2v[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    preprocessed_essays_xtr_avg_w2v_vectors.append(vector)

print(len(preprocessed_essays_xtr_avg_w2v_vectors))
print(len(preprocessed_essays_xtr_avg_w2v_vectors[0]))
```

```
100%|██████████| 40000/40000 [00:46<00:00, 855.02it/s]

40000
100
```

In [65]:
```
preprocessed_essays_xtest_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_test['cleaned_essay']): # for each review/sentence
    vector = np.zeros(100) # 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 preprocessed_essays_xtr_w2v.wv.vocab:
            vector += preprocessed_essays_xtr_w2v[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    preprocessed_essays_xtest_avg_w2v_vectors.append(vector)

print(len(preprocessed_essays_xtest_avg_w2v_vectors))
print(len(preprocessed_essays_xtest_avg_w2v_vectors[0]))
```

```
100%|██████████| 10000/10000 [00:11<00:00, 844.20it/s]

10000
100
```

In [66]:
```
list_preprocessed_title_xtr = []
for e in x_train['cleaned_title_text'].values:
    list_preprocessed_title_xtr.append(e.split())
```

In [67]:
```
preprocessed_title_xtr_w2v=Word2Vec(list_preprocessed_title_xtr,min_count=10,size=100,workers = 8)
```

In [68]:
```
preprocessed_title_xtr_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train['cleaned_title_text']): # for each review/sentence
    vector = np.zeros(100) # 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 preprocessed_title_xtr_w2v.wv.vocab:
            vector += preprocessed_title_xtr_w2v[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    preprocessed_title_xtr_avg_w2v_vectors.append(vector)

print(len(preprocessed_title_xtr_avg_w2v_vectors))
print(len(preprocessed_title_xtr_avg_w2v_vectors[0]))
```

```
100%|██████████| 40000/40000 [00:01<00:00, 27462.60it/s]

40000
100
```

In [69]:
```
preprocessed_title_xtest_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_test['cleaned_title_text']): # for each review/sentence
    vector = np.zeros(100) # 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 preprocessed_title_xtr_w2v.wv.vocab:
            vector += preprocessed_title_xtr_w2v[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    preprocessed_title_xtest_avg_w2v_vectors.append(vector)

print(len(preprocessed_title_xtest_avg_w2v_vectors))
print(len(preprocessed_title_xtest_avg_w2v_vectors[0]))
```

```
100%|██████████| 10000/10000 [00:00<00:00, 26203.67it/s]

10000
100
```

In [70]:
```
X_train_w2v=hstack((preprocessed_essays_xtr_avg_w2v_vectors,preprocessed_title_xtr_avg_w2v_vectors,x_train_clean_cat_ohe
                    ,x_train_quantity_norm ))
#X_cv_tfidf=hstack((preprocessed_essays_xcv_tfidf,preprocessed_title_xcv_tfidf,x_cv_clean_cat_ohe,x_cv_clean_subcat_ohe,

X_test_w2v=hstack((preprocessed_essays_xtest_avg_w2v_vectors,preprocessed_essays_xtest_avg_w2v_vectors,x_test_clean_cat_

                    ,x_test_quantity_norm))
```

In [70]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

params ={ 'n_neighbors': [1, 5, 10, 15, 21, 31, 41, 51, 60]}

estimator3 = KNeighborsClassifier()
Research3 =GridSearchCV(estimator3 ,param_grid = params,cv= 5 ,scoring = 'roc_auc',verbose=22,n_jobs=3)
Research3.fit(X_train_w2v,y_train)
```

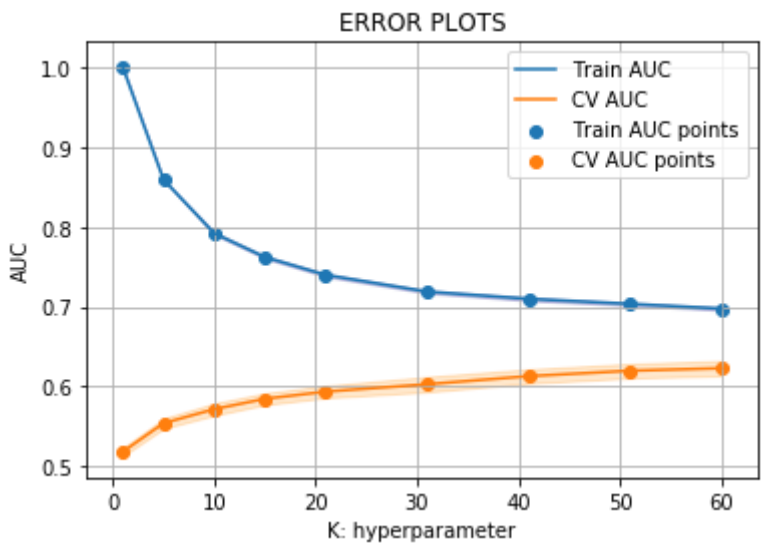
```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

In [71]:
```
print(Research3.best_score_)
n3=Research3.best_params_['n_neighbors']
print('best  k = ',n3)
```

```
0.623057528437459
best  k =  60
```

Performance Plot

In [72]:
```
train_auc3= Research3.cv_results_['mean_train_score']
train_auc_std3= Research3.cv_results_['std_train_score']
cv_auc3 = Research3.cv_results_['mean_test_score']
cv_auc_std3 = Research3.cv_results_['std_test_score']

plt.plot(params['n_neighbors'], train_auc3, label='Train AUC')

# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(params['n_neighbors'],train_auc3 - train_auc_std3,train_auc3 + train_auc_std3,alpha=0.2,color='da
# create a shaded area between [mean - std, mean + std]

plt.plot(params['n_neighbors'], cv_auc3, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(params['n_neighbors'],cv_auc3 - cv_auc_std3,cv_auc3 + cv_auc_std3,alpha=0.2,color='darkorange')

plt.scatter(params['n_neighbors'], train_auc3, label='Train AUC points')
plt.scatter(params['n_neighbors'], cv_auc3, label='CV AUC points')


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



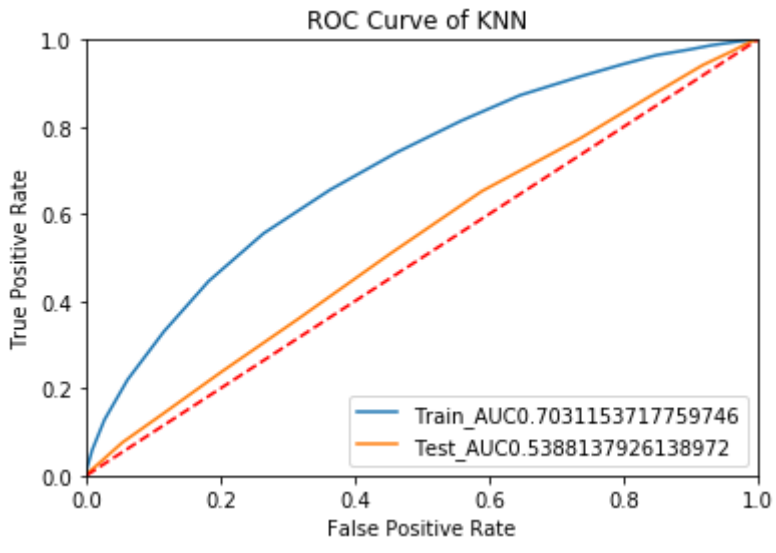
Train new model on best params

In [71]:
```
model_new_w2v = KNeighborsClassifier(n_neighbors = n3)
model_new_w2v.fit(X_train_w2v,y_train)
```

Out[71]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=60, p=2,
           weights='uniform')
```

Roc Curve

In [72]:
```
from  sklearn.metrics import roc_curve
from sklearn.metrics import auc
import matplotlib.pyplot as plt


score_roc_train = model_new_w2v.predict_proba(X_train_w2v)
fpr_train, tpr_train, threshold_train = roc_curve(y_train, score_roc_train[:,1])
roc_auc_train = auc(fpr_train, tpr_train)

score_roc_test = model_new_w2v.predict_proba(X_test_w2v)
fpr_test, tpr_test, threshold_test = roc_curve(y_test, score_roc_test[:,1])
roc_auc_test = auc(fpr_test, tpr_test)


plt.plot(fpr_train, tpr_train, label = "Train_AUC"+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label = "Test_AUC"+str(auc(fpr_test, tpr_test)))
plt.legend(loc = 'lower right')

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


plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of KNN ')
plt.show()
```



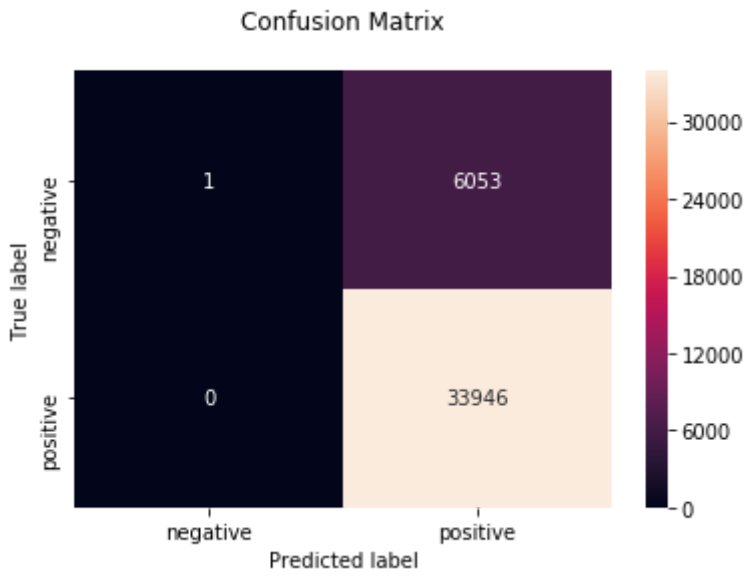
Confusion Matrix

In [73]:
```
y_train_pred_w2v = model_new_w2v.predict(X_train_w2v)

y_test_pred_w2v = model_new_w2v.predict(X_test_w2v)
```

In [74]:
```
from sklearn.metrics import confusion_matrix

ax = plt.subplot()

print("Train confusion matrix")
cnn_train = confusion_matrix(y_train, predict(y_train_pred_w2v, threshold_train, fpr_train, tpr_train))
sns.heatmap(cnn_train,annot = True,ax=ax,fmt='d')




ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.xaxis.set_ticklabels(['negative','positive'])
ax.yaxis.set_ticklabels(['negative','positive'])
plt.title('Confusion Matrix\n')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.4176636688235154 for threshold 0.867
```

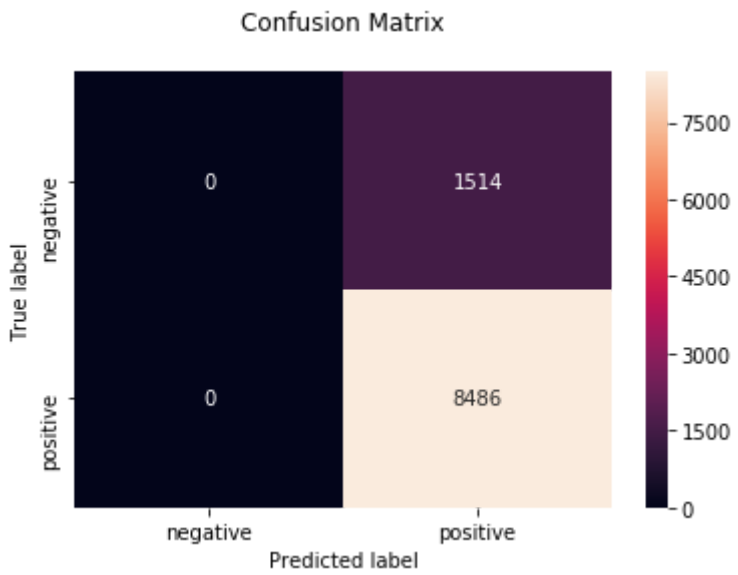
Out[74]:
```
Text(0.5, 1.0, 'Confusion Matrix\n')
```



In [75]:
```
ax = plt.subplot()

print("Test confusion matrix")
cnn_test = confusion_matrix(y_test, predict(y_test_pred_w2v, threshold_test, fpr_test, tpr_test))
sns.heatmap(cnn_test,annot = True,ax=ax,fmt='d')

ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.xaxis.set_ticklabels(['negative','positive'])
ax.yaxis.set_ticklabels(['negative','positive'])
plt.title('Confusion Matrix\n')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.27856589343984384 for threshold 0.867
```

Out[75]:
```
Text(0.5, 1.0, 'Confusion Matrix\n')
```



Classification report

In [76]:
```
from sklearn.metrics import classification_report
print("_" * 101)
print("Classification Report: \n")
print(classification_report(y_test,y_test_pred_w2v))
print("_" * 101)
```

```
_____________________________________________________________________________________________________
Classification Report:


C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)

             precision    recall  f1-score   support

          0       0.00      0.00      0.00      1514
          1       0.85      1.00      0.92      8486

   micro avg       0.85      0.85      0.85     10000
   macro avg       0.42      0.50      0.46     10000
weighted avg       0.72      0.85      0.78     10000

_____________________________________________________________________________________________________
```

SET 4 : [TFIDF-W2V]

In [77]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model1 = TfidfVectorizer()
tfidf_model1.fit(x_train['cleaned_essay'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model1.get_feature_names(), list(tfidf_model1.idf_)))
tfidf_words = set(tfidf_model1.get_feature_names())
```

In [78]:
```
# average Word2Vec
# compute average word2vec for each review.
preprocessed_essays_xtr_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train['cleaned_essay']): # for each review/sentence
    vector = np.zeros(100) # 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 list(preprocessed_essays_xtr_w2v.wv.vocab)) and (word in tfidf_words):
            vec = preprocessed_essays_xtr_w2v[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    preprocessed_essays_xtr_tfidf_w2v_vectors.append(vector)

print(len(preprocessed_essays_xtr_tfidf_w2v_vectors))
print(len(preprocessed_essays_xtr_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 40000/40000 [18:11<00:00, 36.66it/s]

40000
100
```

In [79]:
```
preprocessed_essays_xtest_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_test['cleaned_essay']): # for each review/sentence
    vector = np.zeros(100) # 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 list(preprocessed_essays_xtr_w2v.wv.vocab)) and (word in tfidf_words):
            vec = preprocessed_essays_xtr_w2v[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    preprocessed_essays_xtest_tfidf_w2v_vectors.append(vector)

print(len(preprocessed_essays_xtest_tfidf_w2v_vectors))
print(len(preprocessed_essays_xtest_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 10000/10000 [04:40<00:00, 38.59it/s]

10000
100
```

In [80]:
```
# Similarly you can vectorize for title also
tfidf_model2 = TfidfVectorizer()
tfidf_model2.fit(x_train['cleaned_title_text'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model2.get_feature_names(), list(tfidf_model2.idf_)))
tfidf_words = set(tfidf_model2.get_feature_names())
```

In [81]:
```
preprocessed_title_xtr_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_train['cleaned_title_text']): # for each review/sentence
    vector = np.zeros(100) # 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 list(preprocessed_title_xtr_w2v.wv.vocab)) and (word in tfidf_words):
            vec = preprocessed_title_xtr_w2v[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    preprocessed_title_xtr_tfidf_w2v_vectors.append(vector)

print(len(preprocessed_title_xtr_tfidf_w2v_vectors))
print(len(preprocessed_title_xtr_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 40000/40000 [00:05<00:00, 6762.57it/s]

40000
100
```

In [82]:
```
preprocessed_title_xtest_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(x_test['cleaned_title_text']): # for each review/sentence
    vector = np.zeros(100) # 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 list(preprocessed_title_xtr_w2v.wv.vocab)) and (word in tfidf_words):
            vec = preprocessed_title_xtr_w2v[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.sp
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each wo
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    preprocessed_title_xtest_tfidf_w2v_vectors.append(vector)

print(len(preprocessed_title_xtest_tfidf_w2v_vectors))
print(len(preprocessed_title_xtest_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 10000/10000 [00:01<00:00, 6919.69it/s]

10000
100
```

In [83]:
```
from scipy.sparse import hstack
X_train_tfidf_w2v=hstack((preprocessed_essays_xtr_tfidf_w2v_vectors,preprocessed_title_xtr_tfidf_w2v_vectors,x_train_cle
                    ,x_train_quantity_norm))
#X_cv_tfidf=hstack((preprocessed_essays_xcv_tfidf,preprocessed_title_xcv_tfidf,x_cv_clean_cat_ohe,x_cv_clean_subcat_ohe,

X_test_tfidf_w2v=hstack((preprocessed_essays_xtest_tfidf_w2v_vectors,preprocessed_title_xtest_tfidf_w2v_vectors,x_test_c

                   ,x_test_quantity_norm ))
```

In [86]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

params ={ 'n_neighbors': [1, 5, 10, 15, 21, 31, 41, 51, 60]}

estimator4 = KNeighborsClassifier()
Research4 =GridSearchCV(estimator4 ,param_grid = params,cv= 5 ,scoring = 'roc_auc',verbose=22,n_jobs=3)
Research4.fit(X_train_tfidf_w2v,y_train)
```

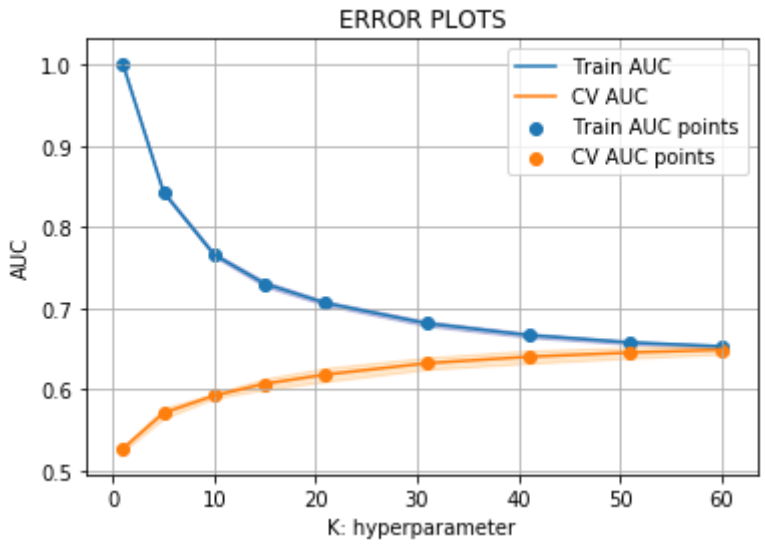
```
Fitting 5 folds for each of 9 candidates, totalling 45 fits

[Parallel(n_jobs=3)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=3)]: Done   1 tasks      | elapsed: 17.7min
[Parallel(n_jobs=3)]: Done   2 tasks      | elapsed: 17.7min
[Parallel(n_jobs=3)]: Done   3 tasks      | elapsed: 17.8min
[Parallel(n_jobs=3)]: Done   4 tasks      | elapsed: 34.4min
[Parallel(n_jobs=3)]: Done   5 tasks      | elapsed: 34.4min
[Parallel(n_jobs=3)]: Done   6 tasks      | elapsed: 34.4min
[Parallel(n_jobs=3)]: Done   7 tasks      | elapsed: 48.8min
[Parallel(n_jobs=3)]: Done   8 tasks      | elapsed: 48.9min
[Parallel(n_jobs=3)]: Done   9 tasks      | elapsed: 48.9min
[Parallel(n_jobs=3)]: Done  10 tasks      | elapsed: 63.0min
[Parallel(n_jobs=3)]: Done  11 tasks      | elapsed: 63.3min
[Parallel(n_jobs=3)]: Done  12 tasks      | elapsed: 63.4min
[Parallel(n_jobs=3)]: Done  13 tasks      | elapsed: 76.7min
[Parallel(n_jobs=3)]: Done  14 tasks      | elapsed: 77.7min
[Parallel(n_jobs=3)]: Done  15 tasks      | elapsed: 77.8min
[Parallel(n_jobs=3)]: Done  16 tasks      | elapsed: 90.9min
[Parallel(n_jobs=3)]: Done  17 tasks      | elapsed: 91.9min
```

In [87]:
```
print(Research4.best_score_)
n4=Research4.best_params_['n_neighbors']
print('best  k = ',n4)
```

```
0.6485102176773646
best  k =  60
```

Performance Plot

In [88]:
```
train_auc4 = Research2.cv_results_['mean_train_score']
train_auc_std4 = Research4.cv_results_['std_train_score']
cv_auc4 = Research4.cv_results_['mean_test_score']
cv_auc_std4 = Research4.cv_results_['std_test_score']

plt.plot(params['n_neighbors'], train_auc4, label='Train AUC')

# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(params['n_neighbors'],train_auc4 - train_auc_std4,train_auc4 + train_auc_std4,alpha=0.2,color='da
# create a shaded area between [mean - std, mean + std]

plt.plot(params['n_neighbors'], cv_auc4, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(params['n_neighbors'],cv_auc4 - cv_auc_std4,cv_auc4 + cv_auc_std4,alpha=0.2,color='darkorange')

plt.scatter(params['n_neighbors'], train_auc4, label='Train AUC points')
plt.scatter(params['n_neighbors'], cv_auc4, label='CV AUC points')


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



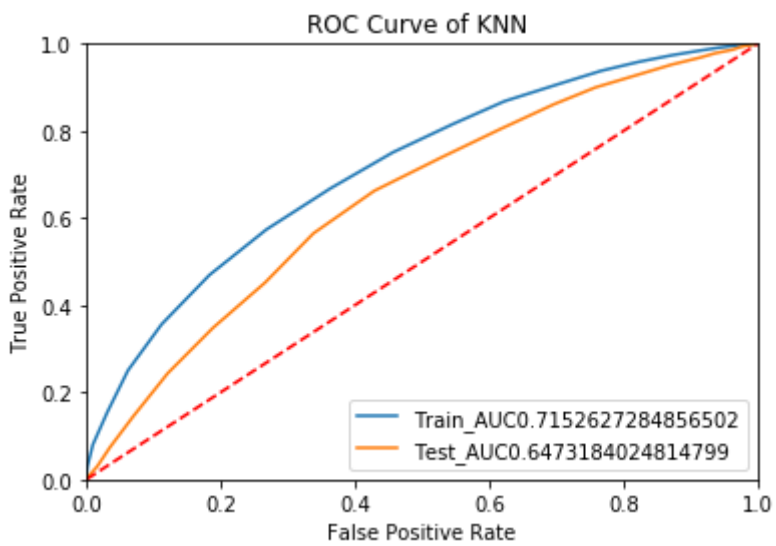
Train new model on best params

In [89]:
```
model_new__tfidf_w2v = KNeighborsClassifier(n_neighbors = n4)
model_new__tfidf_w2v.fit(X_train_tfidf_w2v,y_train)
```

Out[89]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=60, p=2,
           weights='uniform')
```

ROC Curve

In [90]:
```
from  sklearn.metrics import roc_curve
from sklearn.metrics import auc
import matplotlib.pyplot as plt


score_roc_train = model_new__tfidf_w2v.predict_proba(X_train_tfidf_w2v)
fpr_train, tpr_train, threshold_train = roc_curve(y_train, score_roc_train[:,1])
roc_auc_train = auc(fpr_train, tpr_train)

score_roc_test = model_new__tfidf_w2v.predict_proba(X_test_tfidf_w2v)
fpr_test, tpr_test, threshold_test = roc_curve(y_test, score_roc_test[:,1])
roc_auc_test = auc(fpr_test, tpr_test)


plt.plot(fpr_train, tpr_train, label = "Train_AUC"+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label = "Test_AUC"+str(auc(fpr_test, tpr_test)))
plt.legend(loc = 'lower right')

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

plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of KNN ')
plt.show()
```



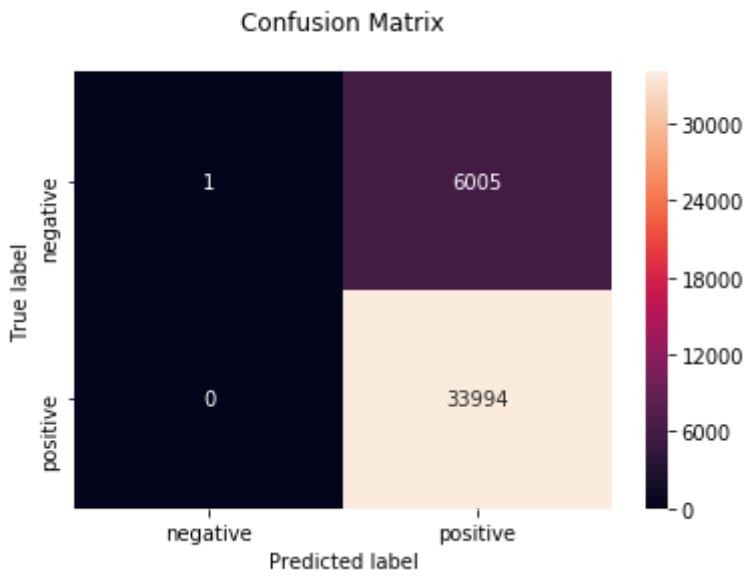
Confusion Matrix

In [91]:
```
y_train_pred_tfidf_w2v = model_new__tfidf_w2v.predict(X_train_tfidf_w2v)

y_test_pred_tfidf_w2v = model_new__tfidf_w2v.predict(X_test_tfidf_w2v)
```

In [92]:
```
from sklearn.metrics import confusion_matrix

ax = plt.subplot()

print("Train confusion matrix")
cnn_train = confusion_matrix(y_train, predict(y_train_pred_tfidf_w2v, threshold_train, fpr_train, tpr_train))
sns.heatmap(cnn_train,annot = True,ax=ax,fmt='d')




ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.xaxis.set_ticklabels(['negative','positive'])
ax.yaxis.set_ticklabels(['negative','positive'])
plt.title('Confusion Matrix\n')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.4252198547662453 for threshold 0.867
```

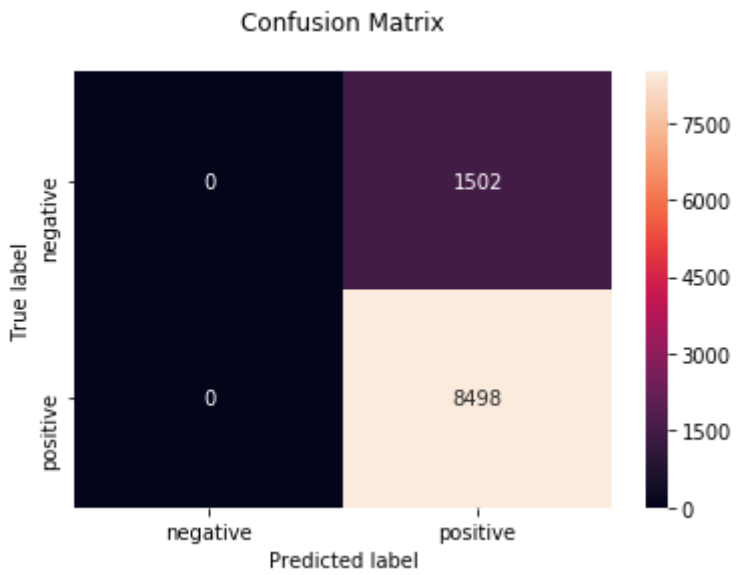
Out[92]:
```
Text(0.5, 1.0, 'Confusion Matrix\n')
```



In [93]:
```
ax = plt.subplot()

print("Test confusion matrix")
cnn_test = confusion_matrix(y_test, predict(y_test_pred_tfidf_w2v, threshold_test, fpr_test, tpr_test))
sns.heatmap(cnn_test,annot = True,ax=ax,fmt='d')

ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.xaxis.set_ticklabels(['negative','positive'])
ax.yaxis.set_ticklabels(['negative','positive'])
plt.title('Confusion Matrix\n')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.37838322732160057 for threshold 0.867
```

Out[93]:
```
Text(0.5, 1.0, 'Confusion Matrix\n')
```



Classification Report

In [94]:
```
from sklearn.metrics import classification_report
print("_" * 101)
print("Classification Report: \n")
print(classification_report(y_test,y_test_pred_tfidf_w2v))
print("_" * 101)
```

```
_____________________________________________________________________________________________________
Classification Report:


C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1502
           1       0.85      1.00      0.92      8498

   micro avg       0.85      0.85      0.85     10000
   macro avg       0.42      0.50      0.46     10000
weighted avg       0.72      0.85      0.78     10000

_____________________________________________________________________________________________________
```

TASK - 2 @ SET-2

Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features

In [101]:
```
from sklearn.feature_selection import SelectKBest, chi2

Selector = SelectKBest(chi2,k=2000)
Selector.fit(X_train_tfidf,y_train)

X_train_new = Selector.transform(X_train_tfidf)
X_test_new  = Selector.transform(X_test_tfidf)
```

In [114]:
```
print(X_train_new.shape)
print(X_test_new.shape)
```

```
(40000, 2000)
(10000, 2000)
```

Hyper-Parameter Tunning

In [102]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

params ={ 'n_neighbors': [1, 5, 10, 15, 21, 31, 41, 51, 60]}

estimator5 = KNeighborsClassifier()
Research5 =GridSearchCV(estimator5 ,param_grid = params,cv= 5 ,scoring = 'roc_auc',verbose=22,n_jobs=4)
Research5.fit(X_train_new,y_train)
```

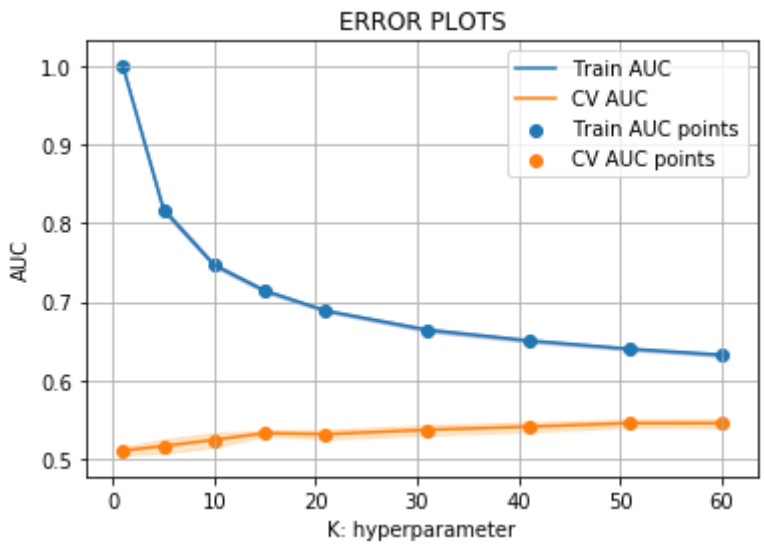
```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

In [105]:
```
print(Research5.best_score_)
n5=Research5.best_params_['n_neighbors']
print('best  k = ',n5)
```

```
0.5454695568857282
best  k =  51
```

Performance Plot

In [106]:
```
train_auc5 = Research5.cv_results_['mean_train_score']
train_auc_std5 = Research5.cv_results_['std_train_score']
cv_auc5 = Research5.cv_results_['mean_test_score']
cv_auc_std5 = Research5.cv_results_['std_test_score']

plt.plot(params['n_neighbors'], train_auc5, label='Train AUC')

# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(params['n_neighbors'],train_auc5 - train_auc_std5,train_auc5 + train_auc_std5,alpha=0.2,color='da
# create a shaded area between [mean - std, mean + std]

plt.plot(params['n_neighbors'], cv_auc5, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(params['n_neighbors'],cv_auc5 - cv_auc_std5,cv_auc5 + cv_auc_std5,alpha=0.2,color='darkorange')

plt.scatter(params['n_neighbors'], train_auc5, label='Train AUC points')
plt.scatter(params['n_neighbors'], cv_auc5, label='CV AUC points')



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



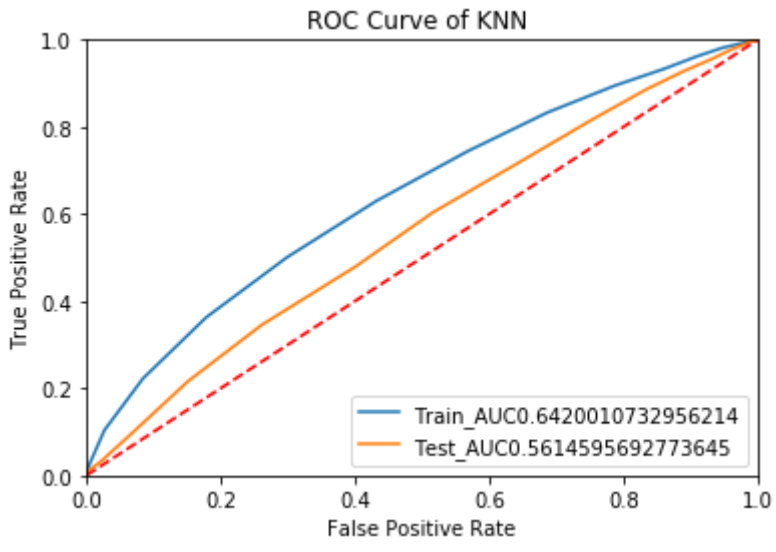
Train Model on best params

In [107]:
```
model_new5 = KNeighborsClassifier(n_neighbors = n5)
model_new5.fit(X_train_new,y_train)
```

Out[107]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=51, p=2,
           weights='uniform')
```

ROC Curve

In [108]:
```
from  sklearn.metrics import roc_curve
from sklearn.metrics import auc
import matplotlib.pyplot as plt


score_roc_train = model_new5.predict_proba(X_train_new)
fpr_train, tpr_train, threshold_train = roc_curve(y_train, score_roc_train[:,1])
roc_auc_train = auc(fpr_train, tpr_train)

score_roc_test = model_new5.predict_proba(X_test_new)
fpr_test, tpr_test, threshold_test = roc_curve(y_test, score_roc_test[:,1])
roc_auc_test = auc(fpr_test, tpr_test)


plt.plot(fpr_train, tpr_train, label = "Train_AUC"+str(auc(fpr_train, tpr_train)))
plt.plot(fpr_test, tpr_test, label = "Test_AUC"+str(auc(fpr_test, tpr_test)))
plt.legend(loc = 'lower right')

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

plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve of KNN ')
plt.show()
```



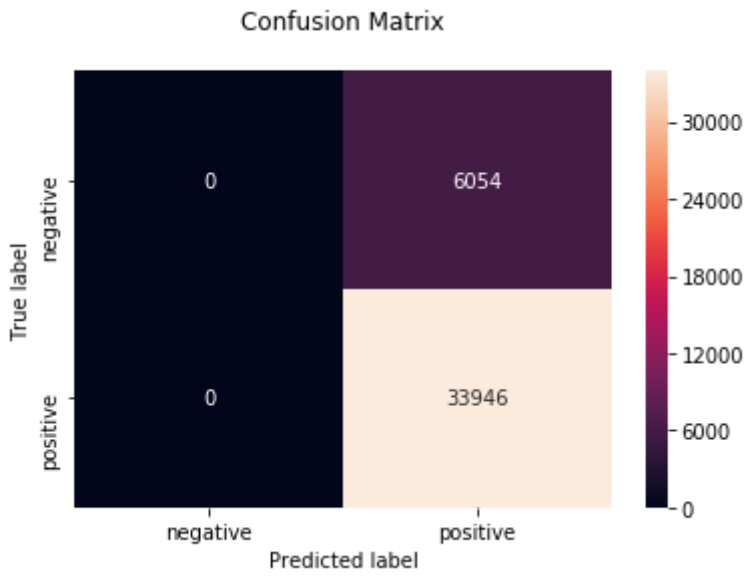
Confusion Matrix

In [109]:
```
y_train_pred_new = model_new5.predict(X_train_new)

y_test_pred_new = model_new5.predict(X_test_new)
```

In [110]:
```
from sklearn.metrics import confusion_matrix

ax = plt.subplot()

print("Train confusion matrix")
cnn_train = confusion_matrix(y_train, predict(y_train_pred_new, threshold_train, fpr_train, tpr_train))
sns.heatmap(cnn_train,annot = True,ax=ax,fmt='d')




ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.xaxis.set_ticklabels(['negative','positive'])
ax.yaxis.set_ticklabels(['negative','positive'])
plt.title('Confusion Matrix\n')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.3578837566129194 for threshold 0.843
```

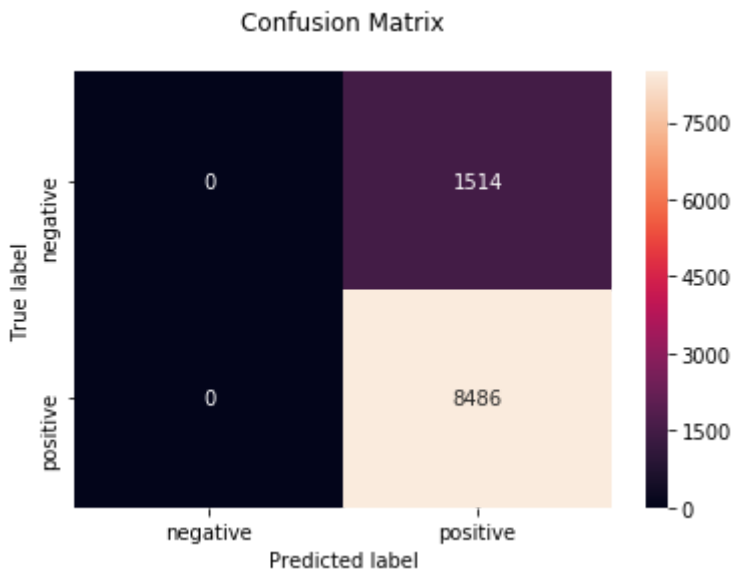
Out[110]:
```
Text(0.5, 1.0, 'Confusion Matrix\n')
```



In [111]:
```
ax = plt.subplot()

print("Test confusion matrix")
cnn_test = confusion_matrix(y_test, predict(y_test_pred_new, threshold_test, fpr_test, tpr_test))
sns.heatmap(cnn_test,annot = True,ax=ax,fmt='d')

ax.set_xlabel('Predicted label')
ax.set_ylabel('True label')
ax.xaxis.set_ticklabels(['negative','positive'])
ax.yaxis.set_ticklabels(['negative','positive'])
plt.title('Confusion Matrix\n')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2921093752675555 for threshold 0.843
```

Out[111]:
```
Text(0.5, 1.0, 'Confusion Matrix\n')
```



Classification Report

In [112]:
```
from sklearn.metrics import classification_report
print("_" * 101)
print("Classification Report: \n")
print(classification_report(y_test,y_test_pred_new))
print("_" * 101)
```

```
_____________________________________________________________________________________________________
Classification Report:


C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\MERCER\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1143: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)

             precision    recall  f1-score   support

          0       0.00      0.00      0.00      1514
          1       0.85      1.00      0.92      8486

   micro avg       0.85      0.85      0.85     10000
   macro avg       0.42      0.50      0.46     10000
weighted avg       0.72      0.85      0.78     10000

_____________________________________________________________________________________________________
```

Conclusion

In [115]:
```
from prettytable import PrettyTable

pretty = PrettyTable()

pretty.field_names = ['Vectorizer','Hyperparameter_n_neighbors','AUC_train','AUC_test']

pretty.add_row(['BOW','60','0.68','0.60'])
pretty.add_row(['TF-IDF','60','0.65','0.57'])
pretty.add_row(['AVG-W2V','60','0.70','0.53'])
pretty.add_row(['TF-IDF WEIGHTED','60','71','0.64'])
pretty.add_row(['Task 2','51','0.64','0.56'])


print(pretty)
```

```
+-----------------+----------------------------+-----------+----------+
|    Vectorizer   | Hyperparameter_n_neighbors | AUC_train | AUC_test |
+-----------------+----------------------------+-----------+----------+
|       BOW       |             60             |    0.68   |   0.60   |
|      TF-IDF     |             60             |    0.65   |   0.57   |
|     AVG-W2V     |             60             |    0.70   |   0.53   |
| TF-IDF WEIGHTED |             60             |     71    |   0.64   |
|      Task 2     |             51             |    0.64   |   0.56   |
+-----------------+----------------------------+-----------+----------+
```