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). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy`

Feature	Description
	• `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!`
`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 approved.

Notes on the Essay Data

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

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

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

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."

your neighborhood, and your school are all helpful.

- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [2]:

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

In [3]:

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

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

In [4]:

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


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


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


project_data.head(2)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_categ
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5

In [5]:

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

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

Out[5]:

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

In [6]:

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

Out[6]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2

In [7]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
project_data.head(5)
```

Out[7]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
0	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
1	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5
2	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Grades 3-5
3	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Grades PreK-2
4	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	IL	2016-04-27 07:19:44	Grades PreK-2

In [8]:

```
X=project_data
X.head(5)
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
0	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
1	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5
2	146723	p099708	c0a28c79fe8ad5810da49de47b3fb491	Mrs.	CA	2016-04-27 01:10:09	Grades 3-5
3	72317	p087808	598621c141cda5fb184ee7e8ccdd3fcc	Ms.	CA	2016-04-27 02:04:15	Grades PreK-2
4	57854	p099430	4000cfe0c8b2df75a218347c1765e283	Ms.	IL	2016-04-27 07:19:44	Grades PreK-2

	Unnamed:							

1.2 preprocessing of `project_subject_categories`

In [9]:

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

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [10]:

```
sub_catogories = list(X['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 & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
```

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

1.3 Text preprocessing

In [11]:

```
# merge two column text dataframe:
X["essay"] = X["project_essay_1"].map(str) +\
                        X["project_essay_2"].map(str) + \
                        X["project_essay_3"].map(str) + \
                        X["project_essay_4"].map(str)
```

In [12]:

```
X.head(2)
```

Out[12]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
0	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2
1	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5

In [13]:

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

In [14]:

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

```
I recently read an article about giving students a choice about how they learn. We already set goa
ls; why not let them choose where to sit, and give them options of what to sit on?I teach at a low
-income (Title 1) school. Every year, I have a class with a range of abilities, yet they are all t
he same age. They learn differently, and they have different interests. Some have ADHD, and some a
re fast learners. Yet they are eager and active learners that want and need to be able to move aro
und the room, yet have a place that they can be comfortable to complete their work.We need a class
room rug that we can use as a class for reading time, and students can use during other learning t
imes. I have also requested four Kore Kids wobble chairs and four Back Jack padded portable chairs
so that students can still move during whole group lessons without disrupting the class. Having th
ese areas will provide these little ones with a way to wiggle while working.Benjamin Franklin once
said, \"Tell me and I forget, teach me and I may remember, involve me and I learn.\" I want these
```

children to be involved in their learning by having a choice on where to sit and how to learn, all by giving them options for comfortable flexible seating.
==
At the beginning of every class we start out with a Math Application problem to help students see the relevance of topics in math. We are always in groups and do a lot of cooperative activities. W e also use lots of technology in our class. I love seeing my students grow and love math!I have a very diverse population of students from all different races, SES, and experiences. My students lo ve school and are starting to embrace the hard work it takes to be a fifth grader. My school is a 5th/6th grade school only and is considered a school for the middle grades. It is located in a sub urban area. It is now more diverse than it has been in many years.I am in an inclusion setting and many of my students have disabilities. It is hard for them to see the board because our resources are old and outdated. A new document camera for our classroom will allow our students to see the board more clearly during instructional times and will create a classroom environment where lots o f movement isn't necessary just because my students cannot see the board.It's frustrating to teach a lesson when many of my students can't see the board because the resources I have are old and out dated. Oftentimes students will tell me to wait before moving on because it takes them forever to write notes because they cannot see the materials. I want students to enjoy coming to my class to learn math and not feel frustrated because they cannot see the board.
==
My students love coming to school and they love learning. I strive daily to make our classroom a r elaxed, comfortable and welcoming environment where all learners will excel and grow in their lear ning. And a new rug will make our days even brighter!My 2nd grade classroom is filled with 20 amaz ing young learners. These students fill my heart everyday with their passion for learning new thin gs. Working with these students and how engaged they are in each subject matter is so much fun. We are small elementary school in mid-Missouri and we have an 80 percent free and reduced lunch rate. I have a wide range of learners in my classroom, and all of my students learn in different ways. S o it is important to provide a learning environment that meets all students.A beautiful new carpet will be the focal point of our classroom. The carpet will be full of students all day long. It wil l be a clean and comfortable place where my students will find comfort in learning. Students will be sitting in small groups, laying and reading a book or even dancing on the carpet for brain brea ks during the day. A carpet in an elementary classroom is the heart of where learning takes place!Thank you for donating or considering a donation to this project. I want to make my 2nd grad e classroom as comfortable and inviting as Starbucks or as cozy as a grandma's living room! This b eautiful carpet will be a perfect addition to a classroom the is filled with so much excitement an d enthusiasm!
==
I teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school pr ovides free breakfast for all students. I am a Special Education certified teacher and I teach Kin dergarten in a general education setting with my class that consists 52% students with special nee ds. The disabilities include Autism Spectrum Disorder, Speech Impaired, Language Impaired, Other H ealth Impaired (ADHD), and Developmentally Delayed. I also have about 42% of my students who are E nglish Language Learners.\r\n\r\n\"Self-motivated learners\" is a synonym of \"my students\". The y love to learn and they possess a positive outlook and attitude in school. Almost everyday, my st udents would ask me, \"Ms. Perez, what are we going to learn today?\" I could not ask for a better greeting from my students.This project will greatly impact my students' learning on a daily basis. The wobble chairs will provide assistance for my students who have difficulties focusing and atten ding during lessons and discussions. Despite the fact that students participate in physical activi ties in P.E., Recess, and GoNoodle (dance videos) sessions in our classroom, students still have e nergy to stand or wiggle from their seats during lessons. Due to these special needs that are beyond the students' control, there is a lot of distraction and student learning is not really ach ieved at its full potential. The lack of appropriate stimulation hinders them to focus and learn i n class. Students with special needs will be able to sit on the wobble chairs during whole group/small group lessons. This will enable their little active bodies to move while "sitting stil l" without disrupting other students. As a result, all students will improve focus and increase st udent attention in learning all content areas. In addition, the visual timer will help my students to actually see the allotted time for activities. This will benefit especially ELL students and st udents with special needs. Whenever we do independent classwork or work in our centers, the studen ts can refer to it and self-monitor their progress in completing assignments. It will encourage th em to use their time wisely and finish tasks on time. It will also help the students have a smooth er transition from one activity to another. \r\nBy donating to this project, you will significantly help students with special needs have an equal opportunity to learn with their peers. Behavior issues will be greatly minimized and classroom management will be optimized. Help me set all students for success! I am looking forward to seeing my students become active listener s and engaged learners, and always happy to go to school!\r\nnannan
==

In [15]:

```
# 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 [16]:

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

```
I teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school pr
ovides free breakfast for all students. I am a Special Education certified teacher and I teach Kin
dergarten in a general education setting with my class that consists 52% students with special nee
ds. The disabilities include Autism Spectrum Disorder, Speech Impaired, Language Impaired, Other H
ealth Impaired (ADHD), and Developmentally Delayed. I also have about 42% of my students who are E
nglish Language Learners.\r\n\r\n\"Self-motivated learners\" is a synonym of \"my students\".  The
y love to learn and they possess a positive outlook and attitude in school. Almost everyday, my st
udents would ask me, \"Ms. Perez, what are we going to learn today?\" I could not ask for a better
greeting from my students.This project will greatly impact my students' learning on a daily basis.
The wobble chairs will provide assistance for my students who have difficulties focusing and atten
ding during lessons and discussions. Despite the fact that students participate in physical activi
ties in P.E., Recess, and GoNoodle (dance videos) sessions in our classroom, students still have e
nergy to stand or wiggle from their seats during lessons. Due to these special needs that are
beyond the students' control, there is a lot of distraction and student learning is not really ach
ieved at its full potential. The lack of appropriate stimulation hinders them to focus and learn i
n class. Students with special needs will be able to sit on the wobble chairs during whole
group/small group lessons. This will enable their little active bodies to move while "sitting stil
l" without disrupting other students. As a result, all students will improve focus and increase st
udent attention in learning all content areas. In addition, the visual timer will help my students
to actually see the allotted time for activities. This will benefit especially ELL students and st
udents with special needs. Whenever we do independent classwork or work in our centers, the studen
ts can refer to it and self-monitor their progress in completing assignments. It will encourage th
em to use their time wisely and finish tasks on time. It will also help the students have a smooth
er transition from one activity to another. \r\nBy donating to this project, you will
significantly help students with special needs have an equal opportunity to learn with their
peers. Behavior issues will be greatly minimized and classroom management will be optimized. Help
me set all students for success! I am looking forward to seeing my students become active listener
s and engaged learners, and always happy to go to school!\r\nnannan
==================================================
```

In [17]:

```
# \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)
```

```
I teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school pr
ovides free breakfast for all students. I am a Special Education certified teacher and I teach Kin
dergarten in a general education setting with my class that consists 52% students with special nee
ds. The disabilities include Autism Spectrum Disorder, Speech Impaired, Language Impaired, Other H
ealth Impaired (ADHD), and Developmentally Delayed. I also have about 42% of my students who are E
nglish Language Learners.     Self-motivated learners  is a synonym of  my students .  They love
to learn and they possess a positive outlook and attitude in school. Almost everyday, my students
would ask me,  Ms. Perez, what are we going to learn today?  I could not ask for a better greeting
from my students.This project will greatly impact my students' learning on a daily basis. The wobb
le chairs will provide assistance for my students who have difficulties focusing and attending dur
ing lessons and discussions. Despite the fact that students participate in physical activities in
P.E., Recess, and GoNoodle (dance videos) sessions in our classroom, students still have energy to
stand or wiggle from their seats during lessons. Due to these special needs that are beyond the st
udents' control, there is a lot of distraction and student learning is not really achieved at its
full potential. The lack of appropriate stimulation hinders them to focus and learn in class. Stud
ents with special needs will be able to sit on the wobble chairs during whole group/small group le
ssons. This will enable their little active bodies to move while "sitting still" without
disrupting other students. As a result, all students will improve focus and increase student atten
tion in learning all content areas. In addition, the visual timer will help my students to actuall
```

y see the allotted time for activities. This will benefit especially ELL students and students wit
h special needs. Whenever we do independent classwork or work in our centers, the students can ref
er to it and self-monitor their progress in completing assignments. It will encourage them to use
their time wisely and finish tasks on time. It will also help the students have a smoother transit
ion from one activity to another. By donating to this project, you will significantly help stude
nts with special needs have an equal opportunity to learn with their peers. Behavior issues will b
e greatly minimized and classroom management will be optimized. Help me set all students for succe
ss! I am looking forward to seeing my students become active listeners and engaged learners, and a
lways happy to go to school! nannan

In [18]:

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

I teach at a Title 1 school with 73 of my students who receive free reduced lunch Our school
provides free breakfast for all students I am a Special Education certified teacher and I teach Ki
ndergarten in a general education setting with my class that consists 52 students with special nee
ds The disabilities include Autism Spectrum Disorder Speech Impaired Language Impaired Other Healt
h Impaired ADHD and Developmentally Delayed I also have about 42 of my students who are English La
nguage Learners Self motivated learners is a synonym of my students They love to learn and they po
ssess a positive outlook and attitude in school Almost everyday my students would ask me Ms Perez
what are we going to learn today I could not ask for a better greeting from my students This proje
ct will greatly impact my students learning on a daily basis The wobble chairs will provide assist
ance for my students who have difficulties focusing and attending during lessons and discussions D
espite the fact that students participate in physical activities in P E Recess and GoNoodle dance
videos sessions in our classroom students still have energy to stand or wiggle from their seats du
ring lessons Due to these special needs that are beyond the students control there is a lot of dis
traction and student learning is not really achieved at its full potential The lack of appropriate
stimulation hinders them to focus and learn in class Students with special needs will be able to
sit on the wobble chairs during whole group small group lessons This will enable their little
active bodies to move while sitting still without disrupting other students As a result all
students will improve focus and increase student attention in learning all content areas In
addition the visual timer will help my students to actually see the allotted time for activities T
his will benefit especially ELL students and students with special needs Whenever we do
independent classwork or work in our centers the students can refer to it and self monitor their p
rogress in completing assignments It will encourage them to use their time wisely and finish tasks
on time It will also help the students have a smoother transition from one activity to another By
donating to this project you will significantly help students with special needs have an equal
opportunity to learn with their peers Behavior issues will be greatly minimized and classroom
management will be optimized Help me set all students for success I am looking forward to seeing m
y students become active listeners and engaged learners and always happy to go to school nannan

In [19]:

```
# 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', 'e
ach', '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', "d
esn'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 [20]:

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

```
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000
[00:25<00:00, 1938.30it/s]
```

In [21]:

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

Out[21]:

```
'teach title 1 school 73 students receive free reduced lunch school provides free breakfast
students special education certified teacher teach kindergarten general education setting class
consists 52 students special needs disabilities include autism spectrum disorder speech impaired l
anguage impaired health impaired adhd developmentally delayed also 42 students english language le
arners self motivated learners synonym students love learn possess positive outlook attitude schoo
l almost everyday students would ask ms perez going learn today could not ask better greeting stud
ents project greatly impact students learning daily basis wobble chairs provide assistance
students difficulties focusing attending lessons discussions despite fact students participate phy
sical activities p e recess gonoodle dance videos sessions classroom students still energy stand w
iggle seats lessons due special needs beyond students control lot distraction student learning not
really achieved full potential lack appropriate stimulation hinders focus learn class students spe
cial needs able sit wobble chairs whole group small group lessons enable little active bodies move
sitting still without disrupting students result students improve focus increase student attention
learning content areas addition visual timer help students actually see allotted time activities b
enefit especially ell students students special needs whenever independent classwork work centers
students refer self monitor progress completing assignments encourage use time wisely finish tasks
time also help students smoother transition one activity another donating project significantly he
lp students special needs equal opportunity learn peers behavior issues greatly minimized
classroom management optimized help set students success looking forward seeing students become ac
tive listeners engaged learners always happy go school nannan'
```

1.4 Preprocessing of `project_title`

In [22]:

```
# Duisplaying first two datasets
X.head(2)
```

Out[22]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
0	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category
1	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5

In [23]:

```
# printing some random project titles.
print(X['project_title'].values[0])
print("="*50)
print(X['project_title'].values[150])
print("="*50)
print(X['project_title'].values[1000])
print("="*50)
```

```
Flexible Seating for Flexible Learning
==================================================
Elmo for Math Instruction
==================================================
Comfy Carpet for Creative Learning
==================================================
```

In [24]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

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

    # 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 [25]:

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

```
Wiggle, Waggle, Wobble: Hocus Focus!
==================================================
```

In [26]:

```
# \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)
```

```
Wiggle, Waggle, Wobble: Hocus Focus!
```

In [27]:

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

```
wiggle waggle wobble hocus focus
```

In [28]:

```
# 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', 'e
ach', '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', "d
esn'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 [29]:

```
sent = ' '.join(e for e in sent.split() if e not in stopwords)
print(sent)
```

```
wiggle waggle wobble hocus focus
```

In [30]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_project_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(X['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).lower()
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_project_titles.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000
[00:01<00:00, 41269.84it/s]
```

In [31]:

```
# after preprocesing
preprocessed_project_titles[20000]
```

Out[31]:

```
'wiggle waggle wobble hocus focus'
```

2. Splitting Data

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

In [32]:

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

2.1 Preparing data for models

Make Data Model Ready: encoding numerical, categorical features and text data

In [33]:

```
X.columns
```

Out[33]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'price', 'quantity',
       'clean_categories', 'clean_subcategories', 'essay'],
      dtype='object')
```

we are going to consider

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

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

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

2.2 Vectorizing Categorical data

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

One hot encoding the catogorical features: state

In [34]:

```
vectorizer = CountVectorizer()
vectorizer.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 = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.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(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', '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']
====================================================================================================
```

One hot encoding the catogorical features: teacher_prefix

In [35]:

```
X_train['teacher_prefix'].fillna(value='Teacher',inplace=True)
X_cv['teacher_prefix'].fillna(value='Teacher',inplace=True)
X_test['teacher_prefix'].fillna(value='Teacher',inplace=True)
vectorizer = CountVectorizer()
vectorizer.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_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 5) (22445,)
(11055, 5) (11055,)
(16500, 5) (16500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

One hot encoding the catogorical features: project_grade_category

In [36]:

```
vectorizer = CountVectorizer()
vectorizer.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 = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.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(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 3) (22445,)
(11055, 3) (11055,)
(16500, 3) (16500,)
['12', 'grades', 'prek']
====================================================================================================
```

One hot encoding the catogorical features: clean_categories

In [37]:

```
vectorizer = CountVectorizer()
vectorizer.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_category_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_category_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_category_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_category_ohe.shape, y_train.shape)
print(X_cv_category_ohe.shape, y_cv.shape)
print(X_test_category_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================================
```

One hot encoding the catogorical features: clean_subcategories

In [38]:

```
vectorizer = CountVectorizer()
vectorizer.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_subcategory_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_subcategory_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcategory_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subcategory_ohe.shape, y_train.shape)
print(X_cv_subcategory_ohe.shape, y_cv.shape)
print(X_test_subcategory_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================================
```

2.3 Vectorizing Text data

2.3.1 Bag of words(BOW)

Converting the essays to vectors

In [39]:

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

print("="*100)


from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['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)
```

```
(22445, 19) (22445,)
(11055, 19) (11055,)
(16500, 19) (16500,)
====================================================================================================

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

Converting the titles to vectors

In [40]:

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

print("="*100)

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['project_title'].values)
X_cv_title_bow = vectorizer.transform(X_cv['project_title'].values)
X_test_title_bow = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print("="*100)
```

```
(22445, 19) (22445,)
(11055, 19) (11055,)
(16500, 19) (16500,)
====================================================================================================

After vectorizations
(22445, 2651) (22445,)
(11055, 2651) (11055,)
(16500, 2651) (16500,)
====================================================================================================
```

2.3.2 TFIDF vectorizer

Converting the essays to vectors

In [41]:

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

print("="*100)


from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
(22445, 19) (22445,)
(11055, 19) (11055,)
(16500, 19) (16500,)
====================================================================================================

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

Converting the titles to vectors

In [42]:

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

print("="*100)


from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_tfidf = vectorizer.transform(X_train['project_title'].values)
X_cv_title_tfidf = vectorizer.transform(X_cv['project_title'].values)
X_test_title_tfidf = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
print("="*100)
```

```
(22445, 19) (22445,)
(11055, 19) (11055,)
(16500, 19) (16500,)
```

```
==========================================================================================================

After vectorizations
(22445, 2634) (22445,)
(11055, 2634) (11055,)
(16500, 2634) (16500,)
==========================================================================================================
```

2.3.3 Using Pretrained Models: Avg W2V

In [ ]:

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

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

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

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

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

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

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

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


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

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


'''
```

AVG W2V for Essays

In [41]:

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

Using Train Data

In [45]:

```
# average Word2Vec
# compute average word2vec for each review.
from scipy.sparse import csr_matrix
avg_w2v_vectors_for_essays_tr = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_for_essays_tr.append(vector)

print(len(avg_w2v_vectors_for_essays_tr))
print(len(avg_w2v_vectors_for_essays_tr[0]))

avg_w2v_vectors_for_essays_tr=csr_matrix(avg_w2v_vectors_for_essays_tr)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 22445/22445
[00:09<00:00, 2428.41it/s]
```

```
22445
300
```

Using Test Data

In [46]:

```
# average Word2Vec
# compute average word2vec for each review.
from scipy.sparse import csr_matrix
avg_w2v_vectors_for_essays_te = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_for_essays_te.append(vector)

print(len(avg_w2v_vectors_for_essays_te))
print(len(avg_w2v_vectors_for_essays_te[0]))

avg_w2v_vectors_for_essays_te=csr_matrix(avg_w2v_vectors_for_essays_te)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 16500/16500
[00:06<00:00, 2421.12it/s]
```

```
16500
300
```

Using CV Data

In [47]:

```
# average Word2Vec
# compute average word2vec for each review.
from scipy.sparse import csr_matrix
avg_w2v_vectors_for_essays_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_for_essays_cv.append(vector)

print(len(avg_w2v_vectors_for_essays_cv))
print(len(avg_w2v_vectors_for_essays_cv[0]))

avg_w2v_vectors_for_essays_cv=csr_matrix(avg_w2v_vectors_for_essays_cv)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 11055/11055
[00:04<00:00, 2512.60it/s]
```

```
11055
300
```

AVG W2V for Titles

Using Train Data

In [42]:

```
# average Word2Vec
# compute average word2vec for each review.
from scipy.sparse import csr_matrix
avg_w2v_vectors_for_titles_tr = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_for_titles_tr.append(vector)

print(len(avg_w2v_vectors_for_titles_tr))
print(len(avg_w2v_vectors_for_titles_tr[0]))

avg_w2v_vectors_for_titles_tr=csr_matrix(avg_w2v_vectors_for_titles_tr)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 22445/22445
[00:00<00:00, 130584.20it/s]
```

```
22445
300
```

Using Test Data

In [43]:

```
# average Word2Vec
# compute average word2vec for each review.
from scipy.sparse import csr_matrix
avg_w2v_vectors_for_titles_te = []; # the avg-w2v for each sentence/review is stored in this list
```

```
for sentence in tqdm(X_test['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_for_titles_te.append(vector)

print(len(avg_w2v_vectors_for_titles_te))
print(len(avg_w2v_vectors_for_titles_te[0]))

avg_w2v_vectors_for_titles_te=csr_matrix(avg_w2v_vectors_for_titles_te)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 16500/16500
[00:00<00:00, 117328.76it/s]
```

```
16500
300
```

Using CV Data

In [44]:

```
# average Word2Vec
# compute average word2vec for each review.
from scipy.sparse import csr_matrix
avg_w2v_vectors_for_titles_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_for_titles_cv.append(vector)

print(len(avg_w2v_vectors_for_titles_cv))
print(len(avg_w2v_vectors_for_titles_cv[0]))

avg_w2v_vectors_for_titles_cv=csr_matrix(avg_w2v_vectors_for_titles_cv)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 141507.45it/s]
```

```
11055
300
```

2.3.4 Using Pretrained Models: TFIDF weighted W2V

TFIDF weighted W2V for Essays

Using Train Data

In [84]:

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

In [85]:

```
# average Word2Vec
# compute average word2vec for each review.
from scipy.sparse import csr_matrix
tfidf_w2v_vectors_for_essays_tr = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_for_essays_tr.append(vector)

print(len(tfidf_w2v_vectors_for_essays_tr))
print(len(tfidf_w2v_vectors_for_essays_tr[0]))

tfidf_w2v_vectors_for_essays_tr=csr_matrix(tfidf_w2v_vectors_for_essays_tr)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 22445/22445 [01:
32<00:00, 242.41it/s]
```

```
22445
300
```

Using Test Data

In [86]:

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

In [87]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_for_essays_te = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_for_essays_te.append(vector)

print(len(tfidf_w2v_vectors_for_essays_te))
print(len(tfidf_w2v_vectors_for_essays_te[0]))

tfidf_w2v_vectors_for_essays_te=csr_matrix(tfidf_w2v_vectors_for_essays_te)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 16500/16500 [01:
07<00:00, 243.77it/s]
```

```
16500
300
```

Using CV Data

In [88]:

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

In [89]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_for_essays_cv = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_for_essays_cv.append(vector)

print(len(tfidf_w2v_vectors_for_essays_cv))
print(len(tfidf_w2v_vectors_for_essays_cv[0]))

tfidf_w2v_vectors_for_essays_cv=csr_matrix(tfidf_w2v_vectors_for_essays_cv)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 11055/11055 [00:
45<00:00, 240.55it/s]
```

```
11055
300
```

TFIDF weighted W2V for Titles

Using Train Data

In [90]:

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

In [91]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_for_titles_tr = []; # the avg-w2v for each sentence/review is stored in this lis
```

```
t
for sentence in tqdm(X_train['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_for_titles_tr.append(vector)

print(len(tfidf_w2v_vectors_for_titles_tr))
print(len(tfidf_w2v_vectors_for_titles_tr[0]))

tfidf_w2v_vectors_for_titles_tr=csr_matrix(tfidf_w2v_vectors_for_titles_tr)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 22445/22445
[00:00<00:00, 84305.26it/s]
```

```
22445
300
```

Using Test Data

In [92]:

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

In [93]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_for_titles_te = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_test['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_for_titles_te.append(vector)

print(len(tfidf_w2v_vectors_for_titles_te))
print(len(tfidf_w2v_vectors_for_titles_te[0]))

tfidf_w2v_vectors_for_titles_te=csr_matrix(tfidf_w2v_vectors_for_titles_te)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 16500/16500
[00:00<00:00, 79679.03it/s]
```

```
16500
300
```

Using CV Data

In [94]:

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

In [95]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_for_titles_cv = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_cv['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_for_titles_cv.append(vector)

print(len(tfidf_w2v_vectors_for_titles_cv))
print(len(tfidf_w2v_vectors_for_titles_cv[0]))

tfidf_w2v_vectors_for_titles_cv=csr_matrix(tfidf_w2v_vectors_for_titles_cv)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 90431.50it/s]
```

```
11055
300
```

2.4 Vectorizing Numerical features

Normalizing the numerical features: Price

In [48]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(-1,1))

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

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Normalizing the numerical features: Teacher number of previously posted projects

In [49]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_teacher_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'
].values.reshape(-1,1))
X_cv_teacher_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].valu
es.reshape(-1,1))
X_test_teacher_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].
values.reshape(-1,1))

print("After vectorizations")
print(X_train_teacher_norm.shape, y_train.shape)
print(X_cv_teacher_norm.shape, y_cv.shape)
print(X_test_teacher_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Normalizing the numerical features: Quantity

In [50]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['quantity'].values.reshape(-1,1))

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

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_cv_quantity_norm.shape, y_cv.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

```
==========================================================================================================
```

2.5 Merging all the above features

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

Merging features for BOW

In [124]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow,X_train_title_bow, X_train_state_ohe, X_train_teacher_ohe,
X_train_grade_ohe,X_train_category_ohe,X_train_subcategory_ohe,X_train_price_norm,X_train_teacher_n
orm,X_train_quantity_norm)).tocsr()
X_cr = hstack((X_cv_essay_bow,X_cv_title_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,X_cv
_category_ohe,X_cv_subcategory_ohe,X_cv_price_norm,X_cv_teacher_norm,X_cv_quantity_norm)).tocsr()
X_te = hstack((X_test_essay_bow, X_test_title_bow,X_test_state_ohe, X_test_teacher_ohe, X_test_grad
e_ohe,X_test_category_ohe,X_test_subcategory_ohe,X_test_quantity_norm,X_test_teacher_norm,X_test_qu
antity_norm)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 7738) (22445,)
(11055, 7738) (11055,)
(16500, 7738) (16500,)
==========================================================================================================
```

Merging features for TFIDF

In [109]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X2_tr = hstack((X_train_essay_tfidf,X_train_title_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_
train_grade_ohe,X_train_category_ohe,X_train_subcategory_ohe,X_train_price_norm,X_train_teacher_nor
m,X_train_quantity_norm)).tocsr()
X2_cr = hstack((X_cv_essay_tfidf,X_cv_title_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe
,X_cv_category_ohe,X_cv_subcategory_ohe,X_cv_price_norm,X_cv_teacher_norm,X_cv_quantity_norm)).toc
sr()
X2_te = hstack((X_test_essay_tfidf, X_test_title_tfidf,X_test_state_ohe, X_test_teacher_ohe, X_test
_grade_ohe,X_test_category_ohe,X_test_subcategory_ohe,X_test_quantity_norm,X_test_teacher_norm,X_t
est_quantity_norm)).tocsr()

print("Final Data matrix")
print(X2_tr.shape, y_train.shape)
print(X2_cr.shape, y_cv.shape)
print(X2_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 7738) (22445,)
(11055, 7738) (11055,)
(16500, 7738) (16500,)
==========================================================================================================
```

Merging features for AVG W2V

In [51]:

In [51]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X3_tr = hstack((avg_w2v_vectors_for_essays_tr,avg_w2v_vectors_for_titles_tr, X_train_state_ohe,
X_train_teacher_ohe,
X_train_grade_ohe,X_train_category_ohe,X_train_subcategory_ohe,X_train_price_norm,X_train_teacher_n
orm,X_train_quantity_norm)).tocsr()
X3_cr = hstack((avg_w2v_vectors_for_essays_cv,avg_w2v_vectors_for_titles_cv, X_cv_state_ohe, X_cv_t
eacher_ohe,
X_cv_grade_ohe,X_cv_category_ohe,X_cv_subcategory_ohe,X_cv_price_norm,X_cv_teacher_norm,X_cv_quanti
ty_norm)).tocsr()
X3_te = hstack((avg_w2v_vectors_for_essays_te,avg_w2v_vectors_for_titles_te,X_test_state_ohe,
X_test_teacher_ohe,
X_test_grade_ohe,X_test_category_ohe,X_test_subcategory_ohe,X_test_quantity_norm,X_test_teacher_nor
m,X_test_quantity_norm)).tocsr()

print("Final Data matrix")
print(X3_tr.shape, y_train.shape)
print(X3_cr.shape, y_cv.shape)
print(X3_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 701) (22445,)
(11055, 701) (11055,)
(16500, 701) (16500,)
====================================================================================================
```

Merging features for TFIDF W2V

In [96]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X4_tr = hstack((tfidf_w2v_vectors_for_essays_tr,tfidf_w2v_vectors_for_titles_tr, X_train_state_ohe
, X_train_teacher_ohe,
X_train_grade_ohe,X_train_category_ohe,X_train_subcategory_ohe,X_train_price_norm,X_train_teacher_n
orm,X_train_quantity_norm)).tocsr()
X4_cr = hstack((tfidf_w2v_vectors_for_essays_cv,tfidf_w2v_vectors_for_titles_cv, X_cv_state_ohe,
X_cv_teacher_ohe,
X_cv_grade_ohe,X_cv_category_ohe,X_cv_subcategory_ohe,X_cv_price_norm,X_cv_teacher_norm,X_cv_quanti
ty_norm)).tocsr()
X4_te = hstack((tfidf_w2v_vectors_for_essays_te,tfidf_w2v_vectors_for_titles_te,X_test_state_ohe,
X_test_teacher_ohe,
X_test_grade_ohe,X_test_category_ohe,X_test_subcategory_ohe,X_test_quantity_norm,X_test_teacher_nor
m,X_test_quantity_norm)).tocsr()

print("Final Data matrix")
print(X4_tr.shape, y_train.shape)
print(X4_cr.shape, y_cv.shape)
print(X4_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 701) (22445,)
(11055, 701) (11055,)
(16500, 701) (16500,)
====================================================================================================
```

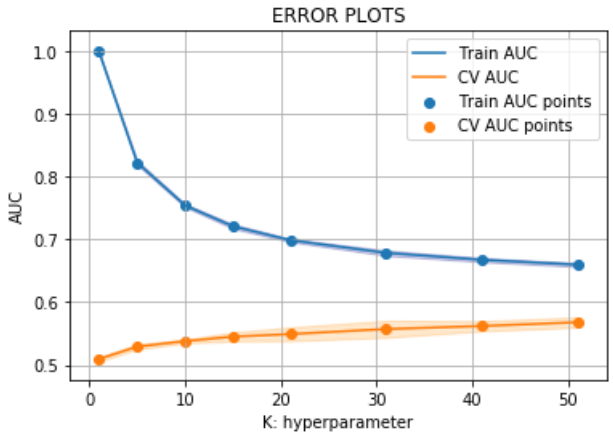
2.6 Apply KNN

2.6.1 Applying KNN brute force on BOW, SET 1

In [54]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
import matplotlib.pyplot as plt
```

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

neigh = KNeighborsClassifier()

parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51]}
clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

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

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

plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')


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



- The error plot between AUC and K hyperparameter gives the best value(Optimum) of K such that the model does not overfit or underfit.

In [125]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between t
he train and cv is less
# Note: based on the method you use you might get different hyperparameter values as best one
# so, you choose according to the method you choose, you use gridsearch if you are having more com
puting power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.

#here we are choosing the best_k based on forloop results
best_k = 12
```

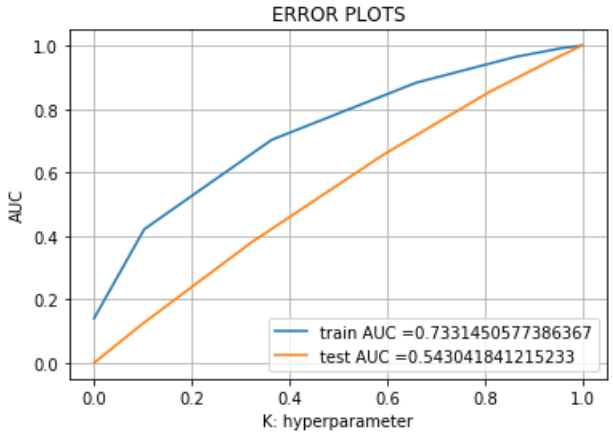
In [126]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
```

```
    # not the predicted outputs

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

    return y_data_pred
```

In [127]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



- Train AUC is 0.7331
- Test AUC is 0.5430 represent the prediction level on the test dataset. In other words if a data point is provided the probabilty of classifying it correctly after the training has been done is 54 %.

In [128]:

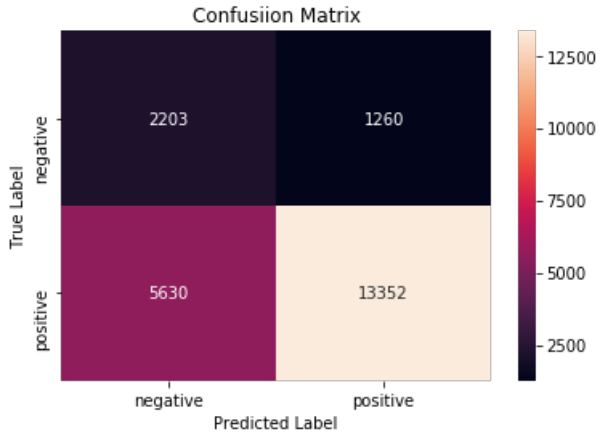
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]
```

```
    # (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
```

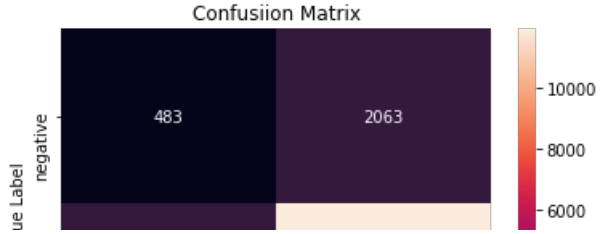
In [129]:

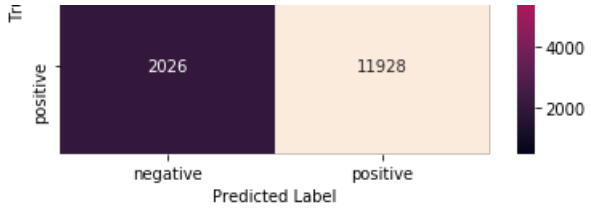
```
print("="*100)
from sklearn.metrics import confusion_matrix
import seaborn as sns
class_label = ["negative", "positive"]

# Reference: https://seaborn.pydata.org/generated/seaborn.heatmap.html
# https://stackoverflow.com/questions/37790429/seaborn-heatmap-using-pandas-dataframe

print("Train confusion matrix")
cm=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
df = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

print("Test confusion matrix")

cm=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
df = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2314621906647469 for threshold 0.833
```




```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24140774401445453 for threshold 0.75
```





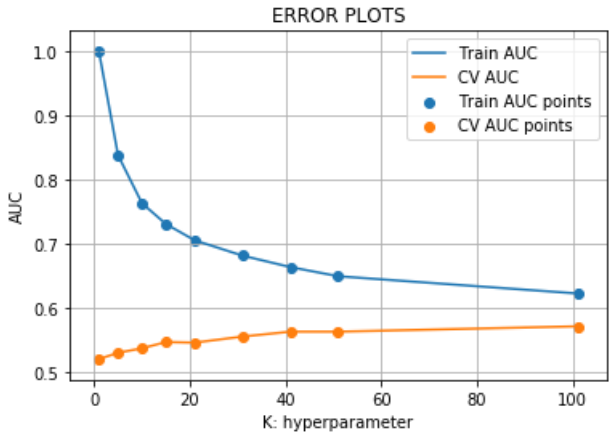
2.6.2 Applying KNN brute force on TFIDF, SET 2

In [110]:

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

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

    return y_data_pred
```

In [47]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X2_tr, y_train)

    y_train_pred = batch_predict(neigh, X2_tr)
    y_cv_pred = batch_predict(neigh, X2_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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



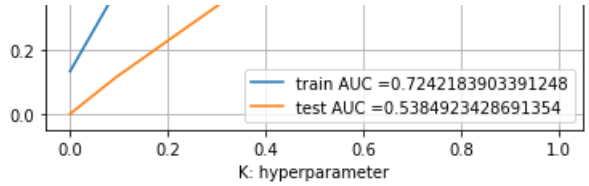
- The error plot between AUC and K hyperparameter gives the best value(Optimum) of K such that the model does not overfit or underfit.

In [111]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between t
he train and cv is less
# Note: based on the method you use you might get different hyperparameter values as best one
# so, you choose according to the method you choose, you use gridsearch if you are having more com
puting power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.

#here we are choosing the best_k based on forloop results
best_k = 15
```

In [112]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X2_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X2_tr)
y_test_pred = batch_predict(neigh, X2_te)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



- Train AUC is 0.7242
- Test AUC is 0.5384 represent the prediction level on the test dataset. In other words if a data point is provided the probabilty of classifying it correctly after the training has been done is 53 %.

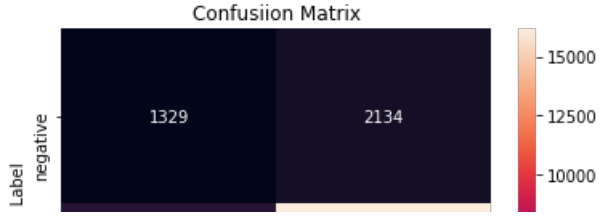
In [113]:

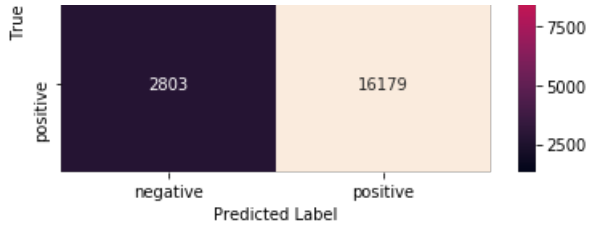
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]

    # (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 [114]:

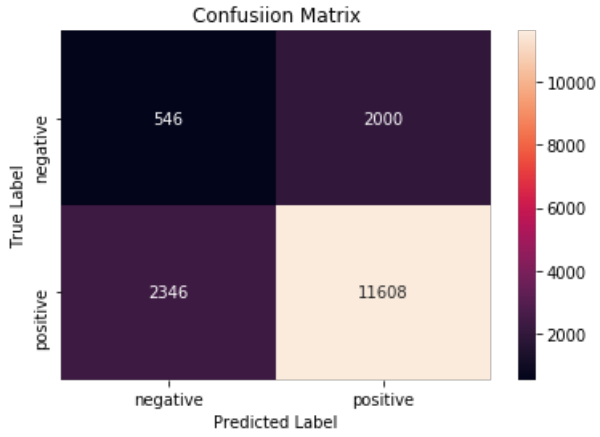
```
print("="*100)
from sklearn.metrics import confusion_matrix
import seaborn as sns
class_label = ["negative", "positive"]

# Reference: https://seaborn.pydata.org/generated/seaborn.heatmap.html
# https://stackoverflow.com/questions/37790429/seaborn-heatmap-using-pandas-dataframe

print("Train confusion matrix")
cm=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
df = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

print("Test confusion matrix")

cm=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
df = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.23649088849751038 for threshold 0.8
```






```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24516685600813068 for threshold 0.8
```



2.6.3 Applying KNN brute force on AVG W2V, SET 3

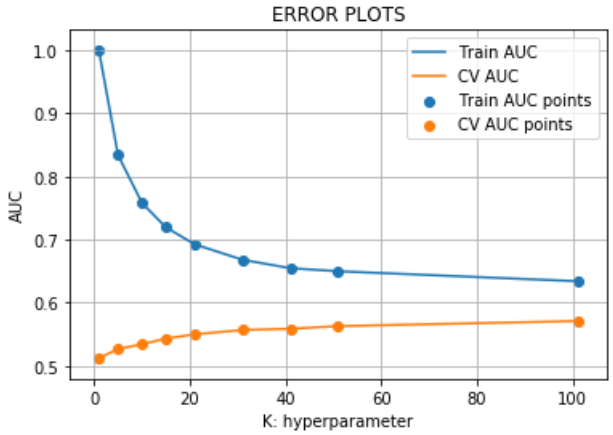
In [136]:

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

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

    return y_data_pred
```

In [50]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X3_tr, y_train)
```

```
    y_train_pred = batch_predict(neigh, X3_tr)
    y_cv_pred = batch_predict(neigh, X3_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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



- The error plot between AUC and K hyperparameter gives the best value(Optimum) of K such that the model does not overfit or underfit.

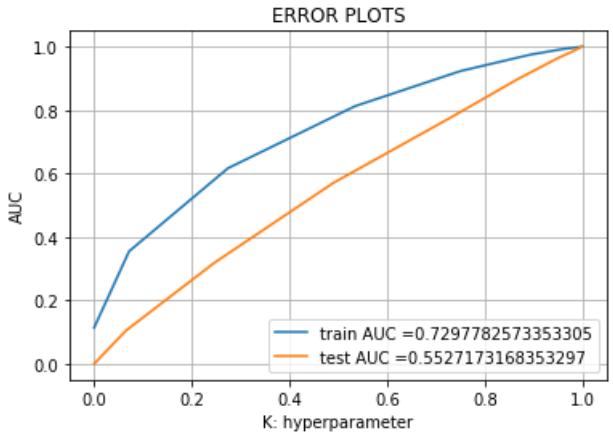
In [137]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between t
he train and cv is less
# Note: based on the method you use you might get different hyperparameter values as best one
# so, you choose according to the method you choose, you use gridsearch if you are having more com
puting power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.

#here we are choosing the best_k based on forloop results
best_k = 16
```

In [138]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X3_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X3_tr)
y_test_pred = batch_predict(neigh, X3_te)
```

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



- Train AUC is 0.7297
- Test AUC is 0.5521 represent the prediction level on the test dataset. In other words if a data point is provided the probabilty of classifying it correctly after the training has been done is 55 %.

In [139]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]

    # (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
```

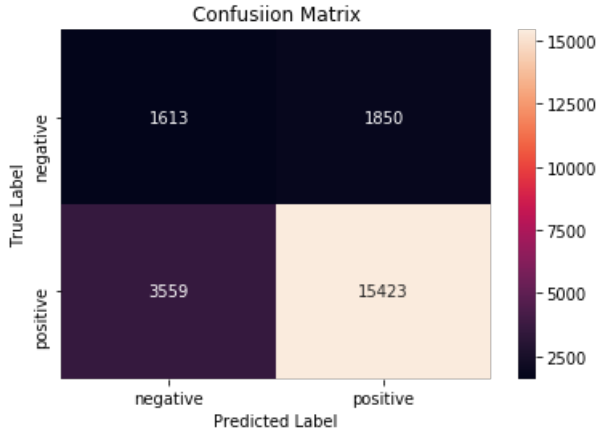
In [140]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
import seaborn as sns
class_label = ["negative", "positive"]

# Reference: https://seaborn.pydata.org/generated/seaborn.heatmap.html
# https://stackoverflow.com/questions/37790429/seaborn-heatmap-using-pandas-dataframe

print("Train confusion matrix")
cm=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
df= pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

print("Test confusion matrix")
```

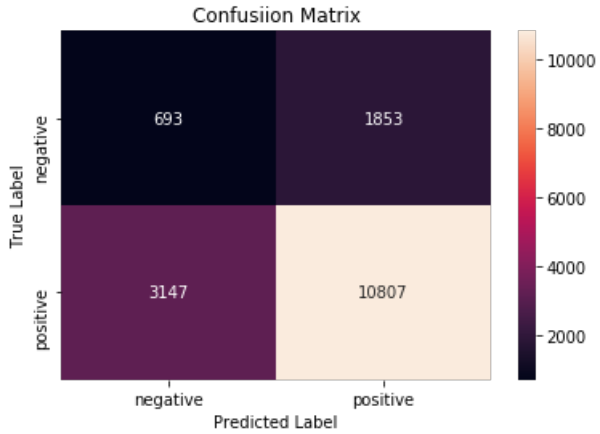
```
cm=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
df = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

==========

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24882906788475237 for threshold 0.812
```




```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24989571306653569 for threshold 0.812
```



2.6.4 Applying KNN brute force on TFIDF W2V, SET 4

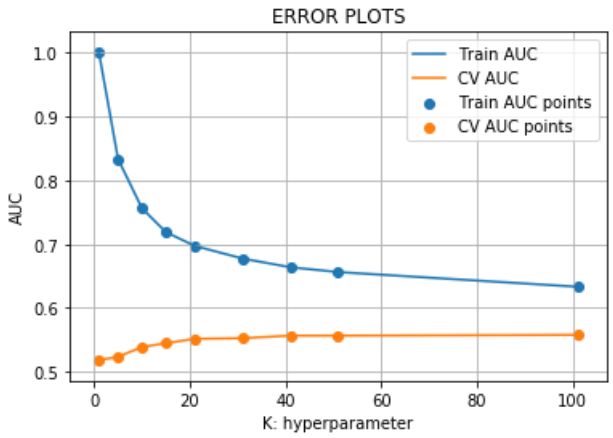
In [56]:

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

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

    return y_data_pred
```

In [57]:

In [96]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X4_tr, y_train)

    y_train_pred = batch_predict(neigh, X4_tr)
    y_cv_pred = batch_predict(neigh, X4_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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



- The error plot between AUC and K hyperparameter gives the best value(Optimum) of K such that the model does not overfit or underfit.

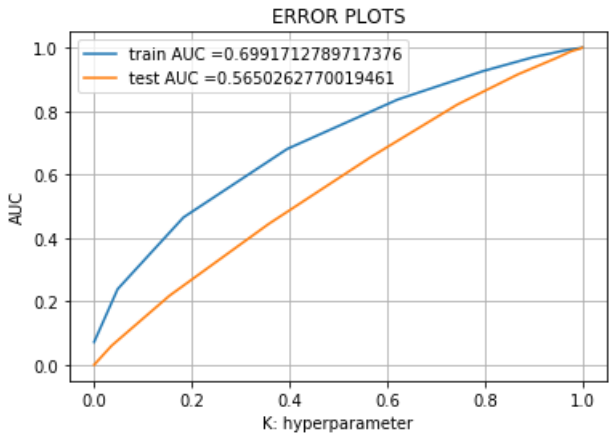
In [97]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between t
he train and cv is less
# Note: based on the method you use you might get different hyperparameter values as best one
# so, you choose according to the method you choose, you use gridsearch if you are having more com
puting power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.
```

```
#here we are choosing the best_k based on forloop results
best_k = 19
```

In [61]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X4_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X4_tr)
y_test_pred = batch_predict(neigh, X4_te)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



- Train AUC is 0.6991
- Test AUC is 0.5650 represent the prediction level on the test dataset. In other words if a data point is provided the probabilty of classifying it correctly after the training has been done is 56 %.

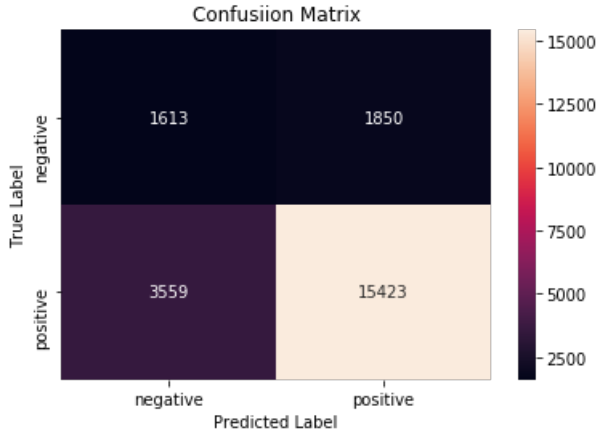
In [141]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]

    # (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
```

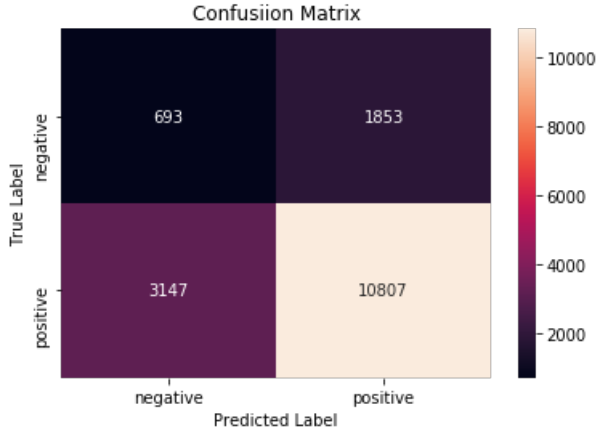
In [142]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
import seaborn as sns
class_label = ["negative", "positive"]

# Reference: https://seaborn.pydata.org/generated/seaborn.heatmap.html
# https://stackoverflow.com/questions/37790429/seaborn-heatmap-using-pandas-dataframe

print("Train confusion matrix")
cm=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
df= pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

print("Test confusion matrix")

cm=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
df= pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24882906788475237 for threshold 0.812
```




```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24989571306653569 for threshold 0.812
```



2.7 Feature selection with `SelectKBest`

In [115]:

```
from sklearn.feature_selection import SelectKBest, chi2

X2_tr_new = SelectKBest(chi2, k=2000).fit_transform(X2_tr, y_train)
print(X2_tr_new.shape)
X2_cr_new = SelectKBest(chi2, k=2000).fit_transform(X2_cr, y_cv)
print(X2_cr_new.shape)
X2_te_new = SelectKBest(chi2, k=2000).fit_transform(X2_te, y_test)
print(X2_te_new.shape)
```

```
(22445, 2000)
(11055, 2000)
(16500, 2000)
```

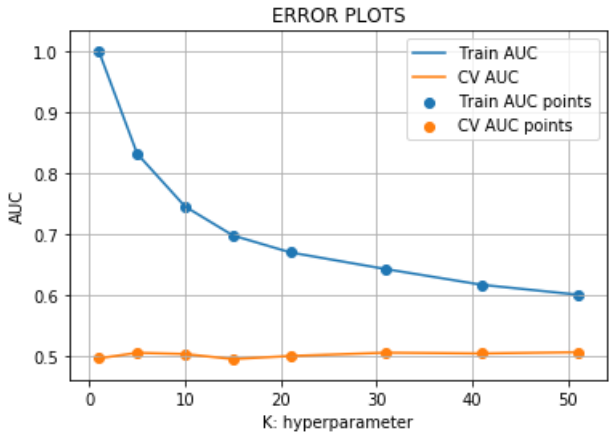
In [116]:

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

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

    return y_data_pred
```

In [91]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X2_tr_new, y_train)

    y_train_pred = batch_predict(neigh, X2_tr_new)
    y_cv_pred = batch_predict(neigh, X2_cr_new)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

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

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



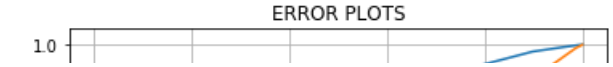
- The error plot between AUC and K hyperparameter gives the best value(Optimum) of K such that the model does not overfit or underfit.

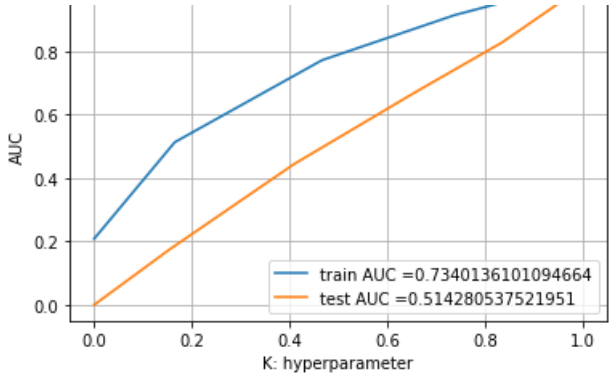
In [118]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between t
he train and cv is less
# Note: based on the method you use you might get different hyperparameter values as best one
# so, you choose according to the method you choose, you use gridsearch if you are having more com
puting power and note it will take more time
# if you increase the cv values in the GridSearchCV you will get more rebust results.

#here we are choosing the best_k based on forloop results
best_k = 12
```

In [119]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X2_tr_new, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X2_tr_new)
y_test_pred = batch_predict(neigh, X2_te_new)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```





- Train AUC is 0.7340 representing the
- Test AUC is 0.5142 represent the prediction level on the test dataset. In other words if a data point is provided the probabilty of classifying it correctly after the training has been done is 51 %.
- The results obtained from SelectKBest features is almost similar to results obtained from TFIDF by considering all features in one go.

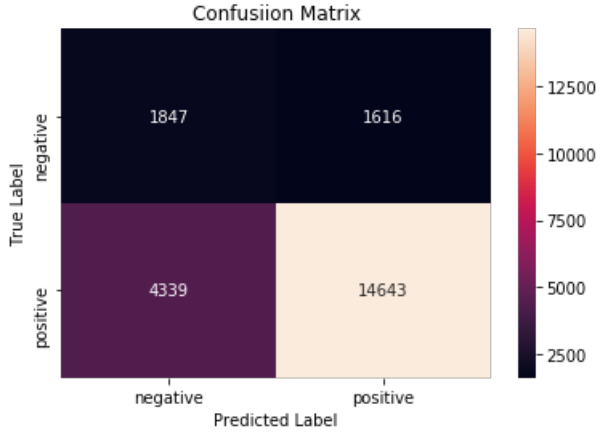
In [120]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(fpr*(1-tpr))]

    # (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
```

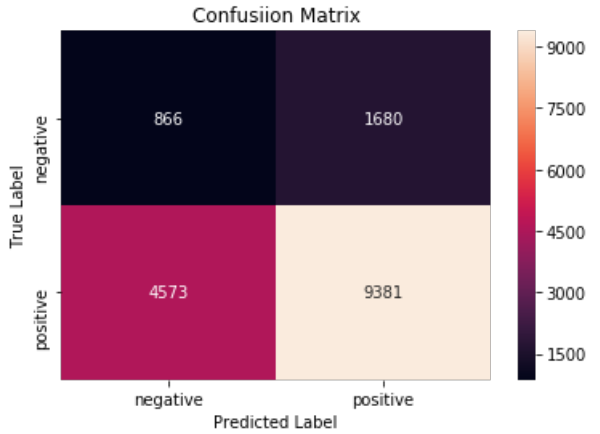
In [121]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
import seaborn as sns
class_label = ["negative", "positive"]

# Reference: https://seaborn.pydata.org/generated/seaborn.heatmap.html
# https://stackoverflow.com/questions/37790429/seaborn-heatmap-using-pandas-dataframe

print("Train confusion matrix")
cm=confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr))
df = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

print("Test confusion matrix")

cm=confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))
df = pd.DataFrame(cm, index = class_label, columns = class_label)
sns.heatmap(df, annot = True, fmt = "d")
plt.title("Confusiion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24888760510954921 for threshold 0.833
```




```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24176796589261898 for threshold 0.833
```



3. Conclusions

In [4]:

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

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]
x.add_row(["BOW", "Brute", 12,0.543 ])
x.add_row(["TFIDF", "Brute", 15,0.538 ])
x.add_row(["AVG W2V", "Brute", 16,0.552 ])
x.add_row(["TFIDF W2V", "Brute", 19, 0.565])
x.add_row(["TFIDF WITH 2000 FEATURES", "Brute", 12, 0.514])
print(x)
```

```
+--------------------------+-------+-----------------+-------+
|        Vectorizer        | Model | Hyper Parameter |  AUC  |
+--------------------------+-------+-----------------+-------+
|           BOW            | Brute |        12       | 0.543 |
|          TFIDF           | Brute |        15       | 0.538 |
|         AVG W2V          | Brute |        16       | 0.552 |
|        TFIDF W2V         | Brute |        19       | 0.565 |
| TFIDF WITH 2000 FEATURES | Brute |        12       | 0.514 |
+--------------------------+-------+-----------------+-------+
```

- In KNN the best value of Hyperparameter 'K' lies between 10 to 20 according to the observations made with BOW,TFIDF,AVG W2V,TFIDF W2V.
- KNN with BOW,TFIDF,AVG W2V,TFIDF W2V have almost the same AUC and give almost same results .
- KNN with TFIDF W2V is a bit higher as compared to other Models having AUC as 0.56.
- The results obtained from SelectKBest features is almost similar to results obtained from TFIDF by considering all features

in one go.

- Thus from the observations made, KNN is not a suitable classifying model for predicting the project is approved or not in this case as it gives prediction rate between 50-60%.
- KNN is computationally very expensive,slow algorithm.