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` • `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*

Feature	Description
`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."
- __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 [0]:

```
%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
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

!pip install chart_studio

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

```
Requirement already satisfied: chart_studio in /usr/local/lib/python3.6/dist-packages (1.0.0)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from chart_studio)
(1.12.0)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from
chart_studio) (2.21.0)
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from
chart_studio) (4.1.1)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packages (from
chart_studio) (1.3.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from
requests->chart_studio) (2019.9.11)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages
(from requests->chart_studio) (3.0.4)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (from
requests->chart_studio) (2.8)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-packages
(from requests->chart_studio) (1.24.3)
```

In [0]:

```
#from google.colab import drive
#drive.mount('/content/drive')
```

1.1 Reading Data

In [0]:

```
project_data = pd.read_csv('/content/drive/My
Drive/Mass3/Assignments_DonorsChoose_2018/train_data.csv')
resource_data = pd.read_csv('/content/drive/My
Drive/Mass3/Assignments_DonorsChoose_2018/resources.csv')
```

In [0]:

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

```
Number of data points in train data (109248, 17)
```

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

In [0]:

```
# 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[0]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	

In [0]:

```
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[0]:

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

1.2 preprocessing of `project_subject_categories`

In [0]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in 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())

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

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

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

In [0]:

```
sorted_cat_dict
```

Out[0]:

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

1.3 preprocessing of `project_subject_subcategories`

In [0]:

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

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & 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())

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

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

```
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

In [0]:

```
print(len(sorted_sub_cat_dict))
print(type(sorted_sub_cat_dict))
```

```
30
<class 'dict'>
```

1.3 Text preprocessing

In [0]:

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

In [0]:

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

Out[0]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_t
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Enginee STEAM the Prin Classrc
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Sens Tools Fo

In [0]:

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

In [0]:

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

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classroom as well as the STEM j
ournals, which my students really enjoyed.  I would love to implement more of the Lakeshore STEM k
its in my classroom for the next school year as they provide excellent and engaging STEM
lessons.My students come from a variety of backgrounds, including language and socioeconomic statu
```

lessons.My students come from a variety of backgrounds, including language and socioeconomic status. Many of them don't have a lot of experience in science and engineering and these kits give me the materials to provide these exciting opportunities for my students.Each month I try to do several science or STEM/STEAM projects. I would use the kits and robot to help guide my science instruction in engaging and meaningful ways. I can adapt the kits to my current language arts pacing guide where we already teach some of the material in the kits like tall tales (Paul Bunyan) or Johnny Appleseed. The following units will be taught in the next school year where I will implement these kits: magnets, motion, sink vs. float, robots. I often get to these units and don't know If I am teaching the right way or using the right materials.   The kits will give me additional ideas, strategies, and lessons to prepare my students in science.It is challenging to develop high quality science activities. These kits give me the materials I need to provide my students with science activities that will go along with the curriculum in my classroom. Although I have some things (like magnets) in my classroom, I don't know how to use them effectively. The kits will provide me with the right amount of materials and show me how to use them in an appropriate way.

==

I teach high school English to students with learning and behavioral disabilities. My students all vary in their ability level. However, the ultimate goal is to increase all students literacy levels. This includes their reading, writing, and communication levels.I teach a really dynamic group of students. However, my students face a lot of challenges. My students all live in poverty and in a dangerous neighborhood. Despite these challenges, I have students who have the the desire to defeat these challenges. My students all have learning disabilities and currently all are performing below grade level. My students are visual learners and will benefit from a classroom that fulfills their preferred learning style.The materials I am requesting will allow my students to be prepared for the classroom with the necessary supplies. Too often I am challenged with students who come to school unprepared for class due to economic challenges. I want my students to be able to focus on learning and not how they will be able to get school supplies. The supplies will last all year. Students will be able to complete written assignments and maintain a classroom journal. The chart paper will be used to make learning more visual in class and to create posters to aid students in their learning. The students have access to a classroom printer. The toner will be used to print student work that is completed on the classroom Chromebooks.I want to try and remove all barriers for the students learning and create opportunities for learning. One of the biggest barriers is the students not having the resources to get pens, paper, and folders. My students will be able to increase their literacy skills because of this project.

==

\"Life moves pretty fast. If you don't stop and look around once in awhile, you could miss it.\" from the movie, Ferris Bueller's Day Off. Think back...what do you remember about your grandparents? How amazing would it be to be able to flip through a book to see a day in their lives?My second graders are voracious readers! They love to read both fiction and nonfiction books. Their favorite characters include Pete the Cat, Fly Guy, Piggie and Elephant, and Mercy Watson. They also love to read about insects, space and plants. My students are hungry bookworms! My students are eager to learn and read about the world around them. My kids love to be at school and are like little sponges absorbing everything around them. Their parents work long hours and usually do not see their children. My students are usually cared for by their grandparents or a family friend. Most of my students do not have someone who speaks English at home. Thus it is difficult for my students to acquire language.Now think forward... wouldn't it mean a lot to your kids, nieces or nephews or grandchildren, to be able to see a day in your life today 30 years from now? Memories are so precious to us and being able to share these memories with future generations will be a rewarding experience. As part of our social studies curriculum, students will be learning about changes over time. Students will be studying photos to learn about how their community has changed over time. In particular, we will look at photos to study how the land, buildings, clothing, and schools have changed over time. As a culminating activity, my students will capture a slice of their history and preserve it through scrap booking. Key important events in their young lives will be documented with the date, location, and names.   Students will be using photos from home and from school to create their second grade memories.   Their scrap books will preserve their unique stories for future generations to enjoy.Your donation to this project will provide my second graders with an opportunity to learn about social studies in a fun and creative manner. Through their scrapbooks, children will share their story with others and have a historical document for the rest of their lives.

==

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

```
m garden in the spring. We will also create our own cookbooks to be printed and shared with famili
es. \r\nStudents will gain math and literature skills as well as a life long enjoyment for healthy
cooking.nannan
==================================================
My classroom consists of twenty-two amazing sixth graders from different cultures and backgrounds.
They are a social bunch who enjoy working in partners and working with groups. They are hard-worki
ng and eager to head to middle school next year. My job is to get them ready to make this
transition and make it as smooth as possible. In order to do this, my students need to come to
school every day and feel safe and ready to learn. Because they are getting ready to head to
middle school, I give them lots of choice- choice on where to sit and work, the order to complete
assignments, choice of projects, etc. Part of the students feeling safe is the ability for them to
come into a welcoming, encouraging environment. My room is colorful and the atmosphere is casual.
I want them to take ownership of the classroom because we ALL share it together. Because my time w
ith them is limited, I want to ensure they get the most of this time and enjoy it to the best of t
heir abilities.Currently, we have twenty-two desks of differing sizes, yet the desks are similar t
o the ones the students will use in middle school. We also have a kidney table with crates for sea
ting. I allow my students to choose their own spots while they are working independently or in
groups. More often than not, most of them move out of their desks and onto the crates. Believe it
or not, this has proven to be more successful than making them stay at their desks! It is because
of this that I am looking toward the “Flexible Seating” option for my classroom.\r\n The students
look forward to their work time so they can move around the room. I would like to get rid of the c
onstricting desks and move toward more “fun” seating options. I am requesting various seating so m
y students have more options to sit. Currently, I have a stool and a papasan chair I inherited fro
m the previous sixth-grade teacher as well as five milk crate seats I made, but I would like to gi
ve them more options and reduce the competition for the “good seats”. I am also requesting two rug
s as not only more seating options but to make the classroom more welcoming and appealing. In orde
r for my students to be able to write and complete work without desks, I am requesting a class set
of clipboards. Finally, due to curriculum that requires groups to work together, I am requesting t
ables that we can fold up when we are not using them to leave more room for our flexible seating o
ptions.\r\nI know that with more seating options, they will be that much more excited about coming
to school! Thank you for your support in making my classroom one students will remember
forever!nannan
==================================================
```

In [0]:

```python
# 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 [0]:

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

```
\"A person is a person, no matter how small.\" (Dr.Seuss) I teach the smallest students with the b
iggest enthusiasm for learning. My students learn in many different ways using all of our senses a
nd multiple intelligences. I use a wide range of techniques to help all my students succeed. \r\nS
tudents in my class come from a variety of different backgrounds which makes for wonderful sharing
of experiences and cultures, including Native Americans.\r\nOur school is a caring community of su
ccessful learners which can be seen through collaborative student project based learning in and ou
t of the classroom. Kindergarteners in my class love to work with hands-on materials and have many
different opportunities to practice a skill before it is mastered. Having the social skills to wor
k cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is the
perfect place to learn about agriculture and nutrition. My students love to role play in our
pretend kitchen in the early childhood classroom. I have had several kids ask me, \"Can we try coo
king with REAL food?\" I will take their idea and create \"Common Core Cooking Lessons\" where we
learn important math and writing concepts while cooking delicious healthy food for snack time. My
students will have a grounded appreciation for the work that went into making the food and knowled
```

```
ge of where the ingredients came from as well as how it is healthy for their bodies. This project
would expand our learning of nutrition and agricultural cooking recipes by having us peel our own
apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classro
om garden in the spring. We will also create our own cookbooks to be printed and shared with famil
ies. \r\nStudents will gain math and literature skills as well as a life long enjoyment for health
y cooking.nannan
==================================================
```

In [0]:

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

```
 A person is a person, no matter how small.  (Dr.Seuss) I teach the smallest students with the big
gest enthusiasm for learning. My students learn in many different ways using all of our senses and
multiple intelligences. I use a wide range of techniques to help all my students succeed.
Students in my class come from a variety of different backgrounds which makes for wonderful
sharing of experiences and cultures, including Native Americans.  Our school is a caring community
of successful learners which can be seen through collaborative student project based learning in a
nd out of the classroom. Kindergarteners in my class love to work with hands-on materials and have
many different opportunities to practice a skill before it is mastered. Having the social skills t
o work cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is
the perfect place to learn about agriculture and nutrition. My students love to role play in our p
retend kitchen in the early childhood classroom. I have had several kids ask me,  Can we try cooki
ng with REAL food?  I will take their idea and create  Common Core Cooking Lessons  where we learn
important math and writing concepts while cooking delicious healthy food for snack time. My
students will have a grounded appreciation for the work that went into making the food and knowled
ge of where the ingredients came from as well as how it is healthy for their bodies. This project
would expand our learning of nutrition and agricultural cooking recipes by having us peel our own
apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classro
om garden in the spring. We will also create our own cookbooks to be printed and shared with famil
ies.   Students will gain math and literature skills as well as a life long enjoyment for healthy
cooking.nannan
```

In [0]:

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

```
 A person is a person no matter how small Dr Seuss I teach the smallest students with the biggest
enthusiasm for learning My students learn in many different ways using all of our senses and multi
ple intelligences I use a wide range of techniques to help all my students succeed Students in my
class come from a variety of different backgrounds which makes for wonderful sharing of
experiences and cultures including Native Americans Our school is a caring community of successful
learners which can be seen through collaborative student project based learning in and out of the
classroom Kindergarteners in my class love to work with hands on materials and have many different
opportunities to practice a skill before it is mastered Having the social skills to work
cooperatively with friends is a crucial aspect of the kindergarten curriculum Montana is the
perfect place to learn about agriculture and nutrition My students love to role play in our
pretend kitchen in the early childhood classroom I have had several kids ask me Can we try cooking
with REAL food I will take their idea and create Common Core Cooking Lessons where we learn
important math and writing concepts while cooking delicious healthy food for snack time My
students will have a grounded appreciation for the work that went into making the food and knowled
ge of where the ingredients came from as well as how it is healthy for their bodies This project w
ould expand our learning of nutrition and agricultural cooking recipes by having us peel our own a
pples to make homemade applesauce make our own bread and mix up healthy plants from our classroom
garden in the spring We will also create our own cookbooks to be printed and shared with families
Students will gain math and literature skills as well as a life long enjoyment for healthy cooking
nannan
```

In [0]:

```python
# 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',
            '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', "do
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 [0]:

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

```
100%|██████████| 109248/109248 [00:52<00:00, 2097.79it/s]
```

In [0]:

```
# after preprocesing
preprocessed_essays[4]
```

Out[0]:

```
'students crave challenge eat obstacles breakfast new texts help ensure materials keep challenged
thinking urban public k 5 elementary school class comprised 12 girls 16 boys incorporate hands
experiences make learning meaningful students eager curious creative learners heart social justice
delight teach new common core standards adopted district students need understand author craft str
ucture analyze framework impacts readers interaction text characters texts also read alouds
classroom rich inner thinking students delve deep examine characters motives change course story r
emarkable gifts provide students complex texts take analytical skills cull ponder would
extravagant remarkable gift would add depth library thank considering classroom donation'
```

In [0]:

```
project_data["essays"] = preprocessed_essays

project_data.drop(['essay'], axis=1, inplace=True)
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [0]:

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

Out[0]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_t
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Enginee STEAM the Prin Classro
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Sens Tools Fo

1.4 Preprocessing of `project_title`

In [0]:

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

```
100%|██████████| 109248/109248 [00:02<00:00, 44717.37it/s]
```

In [0]:

```
# after preprocesing
preprocessed_title[0:4]
```

Out[0]:

```
['engineering steam primary classroom',
 'sensory tools focus',
 'mobile learning mobile listening center',
 'flexible seating flexible learning']
```

In [0]:

```
project_data["project_title"] = preprocessed_title
```

Preprocessing of 'Project_grade_category'

In [0]:

```
project_data['project_grade_category']=project_data['project_grade_category'].replace('Grades
PreK-2','Grades_PreK_2')
project_data['project_grade_category']=project_data['project_grade_category'].replace('Grades 3-5'
,'Grades_3_5')
project_data['project_grade_category']=project_data['project_grade_category'].replace('Grades 6-8'
,'Grades_6_8')
project_data['project_grade_category']=project_data['project_grade_category'].replace('Grades 9-12
','Grades_9_12')
```

1.5 Preparing data for models

In [0]:

```
project_data.columns
```

Out[0]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essays'],
      dtype='object')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

In [0]:

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

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (109248, 9)
```

In [0]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encoding ",sub_categories_one_hot.shape)
```

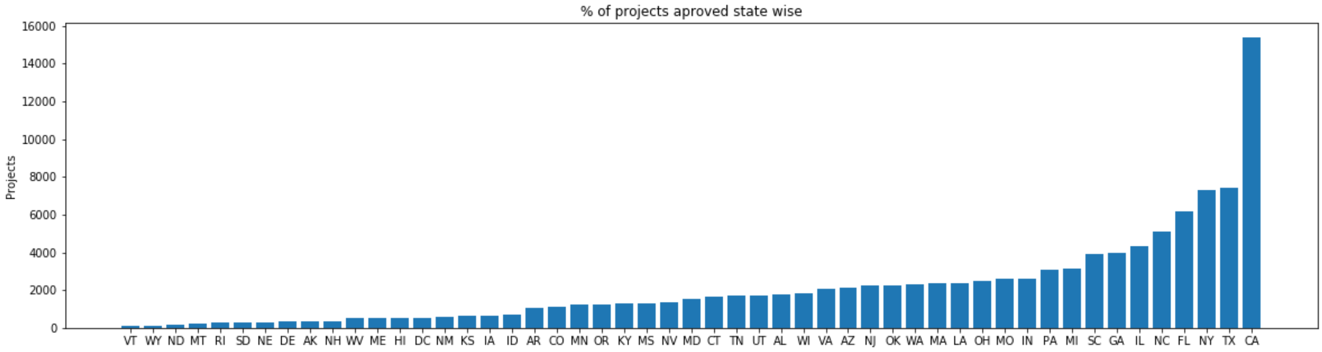
```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encoding  (109248, 30)
```

In [0]:

```
#@title #### State Dictionary
```

In [0]:

```
from collections import Counter
my_counter1 = Counter()
for word in project_data['school_state'].values:
    my_counter1.update(word.split())

state_dict = dict(my_counter1)
sorted_state_dict = dict(sorted(state_dict.items(), key=lambda kv: kv[1]))
ind1 = np.arange(len(sorted_state_dict))
plt.figure(figsize=(20,5))
p2 = plt.bar(ind1, list(sorted_state_dict.values()))
plt.ylabel('Projects')
plt.title('% of projects aproved state wise')
plt.xticks(ind1, list(sorted_state_dict.keys()))
plt.show()


for i, j in sorted_state_dict.items():
    print("{:20} :{:10}".format(i,j))
```



```
VT                   :        80
WY                   :        98
ND                   :       143
MT                   :       245
RI                   :       285
SD                   :       300
NE                   :       309
DE                   :       343
AK                   :       345
NH                   :       348
WV                   :       503
ME                   :       505
HI                   :       507
DC                   :       516
NM                   :       557
KS                   :       634
IA                   :       666
ID                   :       693
AR                   :      1049
CO                   :      1111
MN                   :      1208
OR                   :      1242
KY                   :      1304
MS                   :      1323
NV                   :      1367
MD                   :      1514
CT                   :      1663
TN                   :      1688
UT                   :      1731
AL                   :      1762
WI                   :      1827
VA                   :      2045
AZ                   :      2147
NJ                   :      2237
OK                   :      2276
WA                   :      2334
MA                   :      2389
```

```
LA                      :      2394
OH                      :      2467
MO                      :      2576
IN                      :      2620
PA                      :      3109
MI                      :      3161
SC                      :      3936
GA                      :      3963
IL                      :      4350
NC                      :      5091
FL                      :      6185
NY                      :      7318
TX                      :      7396
CA                      :     15388
```

In [0]:

```
#@title ####State
```

In [0]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=False, binary=Tr
ue)
vectorizer.fit(project_data['school_state'].values)
print(vectorizer.get_feature_names())


state_one_hot = vectorizer.transform(project_data['school_state'].values)
print("Shape of matrix after one hot encodig ",state_one_hot.shape)
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'SD', 'NE', 'DE', 'AK', 'NH', 'WV', 'ME', 'HI', 'DC', 'NM', 'KS', 'I
A', 'ID', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'NV', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ',
'NJ', 'OK', 'WA', 'MA', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX
', 'CA']
Shape of matrix after one hot encodig  (109248, 51)
```

In [0]:

```
#@title #### Teacher_prefix Dictionary
```

In [0]:

```
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna(method='ffill')
```

In [0]:

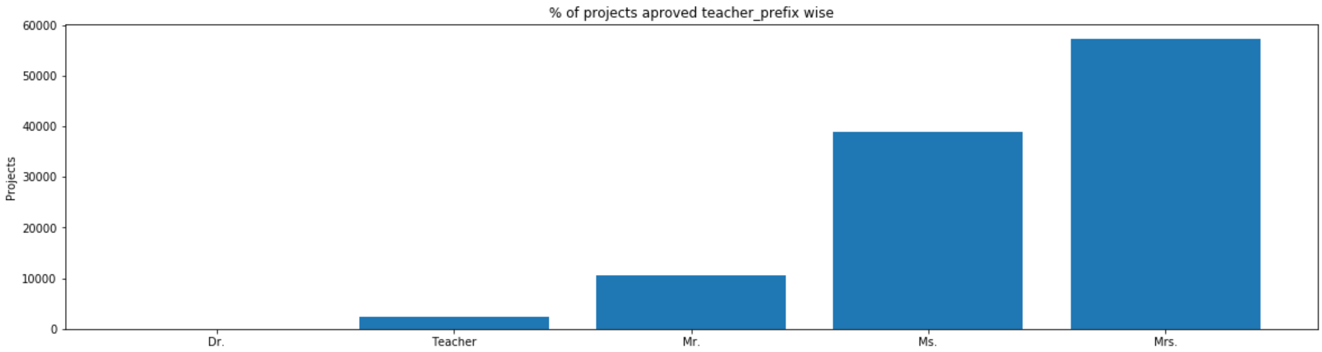
```
project_data['teacher_prefix'][30368:30372]
```

Out[0]:

```
61953    Mrs.
27617     Ms.
84687    Mrs.
39387    Mrs.
Name: teacher_prefix, dtype: object
```

In [0]:

```
from collections import Counter
my_counter2 = Counter()
for word in project_data['teacher_prefix'].values:
    my_counter2.update(word.split())

prefix_dict = dict(my_counter2)
sorted_prefix_dict = dict(sorted(prefix_dict.items(), key=lambda kv: kv[1]))
ind2 = np.arange(len(sorted_prefix_dict))
plt.figure(figsize=(20,5))
p2 = plt.bar(ind2, list(sorted_prefix_dict.values()))
plt.ylabel('Projects')
plt.title('% of projects aproved teacher_prefix wise')
plt.xticks(ind2, list(sorted_prefix_dict.keys()))
plt.show()


for i, j in sorted_prefix_dict.items():
    print("{:20} :{:10}".format(i,j))
```



```
Dr.                  :        13
Teacher              :      2360
Mr.                  :     10648
Ms.                  :     38957
Mrs.                 :     57270
```

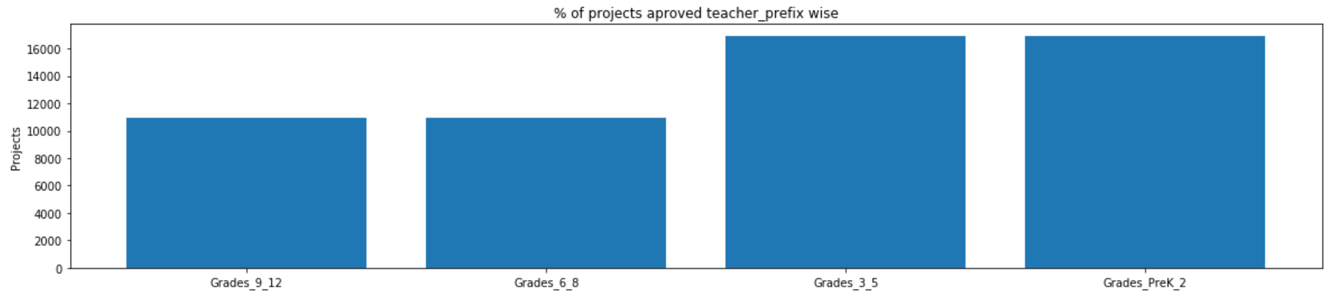
In [0]:

```
#Teacher_prefix

vectorizer = CountVectorizer(vocabulary=list(sorted_prefix_dict.keys()), lowercase=False, binary=T
rue)
vectorizer.fit(project_data['teacher_prefix'].values)
print(vectorizer.get_feature_names())


teacher_prefix_one_hot = vectorizer.transform(project_data['teacher_prefix'].values)
print("Shape of matrix after one hot encodig ",teacher_prefix_one_hot.shape)
```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of matrix after one hot encodig  (109248, 5)
```

In [0]:

```
#project_grade_category Dictionary

#How_to_update_keys: https://www.geeksforgeeks.org/python-ways-to-change-keys-in-dictionary/


from collections import Counter
my_counter3 = Counter()
for word in project_data['project_grade_category'].values:
    my_counter3.update(word.split('_'))


grade_dict = dict(my_counter3)
sorted_grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))
dict((sorted_grade_dict[key], value) for (key, value) in sorted_grade_dict.items())
updt_keys = ['Grades_9_12', 'Grades_6_8', 'Grades_3_5', 'Grades_PreK_2']
sorted_grade_dict = dict(zip(updt_keys, list(sorted_grade_dict.values())))
ind3 = np.arange(len(sorted_grade_dict))
plt.figure(figsize=(20,4))
p3 = plt.bar(ind3, list(sorted_grade_dict.values()))
plt.ylabel('Projects')
plt.title('% of projects aproved teacher_prefix wise')
plt.xticks(ind3, list(sorted_grade_dict.keys()))
plt.show()


for i, j in sorted_grade_dict.items():
    print("{:20} :{:10}".format(i,j))
```



```
Grades_9_12          :     10963
Grades_6_8           :     10963
Grades_3_5           :     16923
Grades_PreK_2        :     16923
```

In [0]:

```
#project_grade_category
vectorizer = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=False, binary=True)
vectorizer.fit(project_data['project_grade_category'].values)
print(vectorizer.get_feature_names())


grade_cat_one_hot = vectorizer.transform(project_data['project_grade_category'].values)
print("Shape of matrix after one hot encoding ",grade_cat_one_hot.shape)
```

```
['Grades_9_12', 'Grades_6_8', 'Grades_3_5', 'Grades_PreK_2']
Shape of matrix after one hot encoding  (109248, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [0]:

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

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

In [0]:

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

```
100%|██████████| 109248/109248 [00:02<00:00, 50637.40it/s]
```

In [0]:

```
# Similarly you can vectorize for title also
vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_title)
text_bow = vectorizer.fit_transform(preprocessed_title)
print("Shape of matrix after one hot encoding ",text_bow.shape)
```

```
Shape of matrix after one hot encoding  (109248, 3222)
```

1.5.2.2 TFIDF vectorizer

In [0]:

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

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

1.5.2.3 Using Pretrained Models: Avg W2V

In [0]:

```
'''
# 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)
```

```
'''
```

Out[0]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
============================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus",
len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\n
print("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [0]:

```python
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('/content/drive/My Drive/Mass3/Assignments_DonorsChoose_2018/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [0]:

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

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

```
100%|██████████| 109248/109248 [00:31<00:00, 3463.11it/s]
```

```
109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [0]:

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

In [0]:

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

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

```
100%|██████████| 109248/109248 [03:08<00:00, 580.13it/s]
```

```
109248
300
```

In [0]:

```
 #### Using Pretrained Models: TFIDF weighted W2V on `project_title`
```

In [0]:

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

In [0]:

```
# Similarly you can vectorize for title also

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

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

```
100%|██████████| 109248/109248 [00:03<00:00, 31748.86it/s]
```

```
109248
300
```

In [0]:

```
# Conversion of a list to sparse matrix

from scipy.sparse import coo_matrix
tfidf_w2v_matrix=np.reshape(np.asarray(tfidf_w2v_vectors),(109248,300))
sparse_tfidf_w2v_matrix=coo_matrix(tfidf_w2v_matrix).tocsr()

print("Shape of matrix after one hot encoding ",sparse_tfidf_w2v_matrix.shape)
```

```
Shape of matrix after one hot encoding  (109248, 300)
```

1.5.3 Vectorizing Numerical features

In [0]:

```
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')
```

In [0]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

# Now standardize the data with above maen and variance.
price_standardized = price_scalar.transform(project_data['price'].values.reshape(-1, 1))
```

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

In [0]:

```
price_standardized
```

Out[0]:

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

1.5.4 Merging all the above features

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

In [0]:

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
```

```
(109248, 9)
(109248, 30)
(109248, 3222)
(109248, 1)
```

In [0]:

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

Out[0]:

```
(109248, 3262)
```

Assignment 3: Apply KNN

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

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

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

   - Find the best hyper parameter which results in the maximum AUC value
   - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
   - Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task

3. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
   - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

4. **[Task-2]**

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

     -
       ```
       from sklearn.datasets import load_digits
       from sklearn.feature_selection import SelectKBest, chi2
       X, y = load_digits(return_X_y=True)
       X.shape
       X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
       X_new.shape
       ========
       output:
       (1797, 64)
       (1797, 20)
       ```
   - Repeat the steps 2 and 3 on the data matrix after feature selection

5. **Conclusion**

   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

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

2. K Nearest Neighbor

In [0]:

```
#Choosing top 50k datapoints due to memory issue.

project_data=project_data.loc[:49999,:]
project_data.shape
```

Out[0]:

```
(50000, 16)
```

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

In [0]:

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

Out[0]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_title
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades_PreK_2	engineering steam primary classroom

In [0]:

```
# 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)
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [0]:

```
#### encoding categorical features: School State
```

In [0]:

```
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']
===================================================================================================
```

In [0]:

```
#### encoding categorical features: teacher_prefix
```

In [0]:

```
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']
===================================================================================================
```

In [0]:

```
#### encoding categorical features: project_grade_category
```

In [0]:

```
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_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, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================================
```

In [0]:

```
#### encoding categorical features: clean_categories
```

In [0]:

```
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_clean_cat_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_clean_cat_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_cat_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_cat_ohe.shape, y_train.shape)
print(X_cv_clean_cat_ohe.shape, y_cv.shape)
print(X_test_clean_cat_ohe.shape, y_test.shape)
print(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']
====================================================================================================
```

In [0]:

```
#### encoding categorical features: clean_subcategories
```

In [0]:

```
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_clean_subcat_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_clean_subcat_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_clean_subcat_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_clean_subcat_ohe.shape, y_train.shape)
print(X_cv_clean_subcat_ohe.shape, y_cv.shape)
print(X_test_clean_subcat_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',
'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']
====================================================================================================
```

In [0]:

```
#### encoding numerical features: Price
```

In [0]:

```
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)
```

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

In [0]:

```
X_train_price_norm=X_train_price_norm.reshape(-1,1)
X_cv_price_norm=X_cv_price_norm.reshape(-1,1)
X_test_price_norm=X_test_price_norm.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)
```

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

In [0]:

```
#### encoding numerical features: teacher_number_of_previously_posted_projects
```

In [0]:

```
normalizer = Normalizer()

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

X_train_prev_posted = normalizer.transform(X_train['teacher_number_of_previously_posted_projects']
.values.reshape(1,-1))
```

```
X_cv_prev_posted =
normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_test_prev_posted = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].v
alues.reshape(1,-1))

print("After vectorizations")
print(X_train_prev_posted.shape, y_train.shape)
print(X_cv_prev_posted.shape, y_cv.shape)
print(X_test_prev_posted.shape, y_test.shape)
print("="*100)
```

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

In [0]:

```
X_train_prev_posted=X_train_prev_posted.reshape(-1,1)
X_cv_prev_posted=X_cv_prev_posted.reshape(-1,1)
X_test_prev_posted=X_test_prev_posted.reshape(-1,1)

print("After vectorizations")
print(X_train_prev_posted.shape, y_train.shape)
print(X_cv_prev_posted.shape, y_cv.shape)
print(X_test_prev_posted.shape, y_test.shape)
print("="*100)
```

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

2.3 Make Data Model Ready: encoding eassay, and project_title

In [0]:

```
### 2.3.1 Bag of Words
```

In [0]:

```
#ESSAY

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

print("="*100)


vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essays'].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['essays'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['essays'].values)
X_test_essay_bow = vectorizer.transform(X_test['essays'].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, 15) (22445,)
(11055, 15) (11055,)
(16500, 15) (16500,)
```

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

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

In [0]:

```
#PROJECT_TITLE

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

print("="*100)


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, 15) (22445,)
(11055, 15) (11055,)
(16500, 15) (16500,)
====================================================================================================

After vectorizations
(22445, 2012) (22445,)
(11055, 2012) (11055,)
(16500, 2012) (16500,)
====================================================================================================
```

In [0]:

```
### 2.3.2 TFIDF
```

In [0]:

```
#ESSAY
```

In [0]:

```
vectorizer = TfidfVectorizer(min_df=10)

vectorizer.fit(X_train['essays'])
essay_tfidf_train=vectorizer.transform(X_train['essays'])

print("Shape of matrix after one hot encoding ",essay_tfidf_train.shape)
print("="*100)



essay_tfidf_cv=vectorizer.transform(X_cv['essays'])

print("Shape of matrix after one hot encoding ",essay_tfidf_cv.shape)
print("="*100)
```

```
essay_tfidf_test=vectorizer.transform(X_test['essays'])

print("Shape of matrix after one hot encoding ",essay_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (22445, 8742)
====================================================================================================

Shape of matrix after one hot encoding  (11055, 8742)
====================================================================================================

Shape of matrix after one hot encoding  (16500, 8742)
```

In [0]:

```
#PROJECT_TITLE
```

In [0]:

```
vectorizer = TfidfVectorizer(min_df=10)

vectorizer.fit(X_train['project_title'])
title_tfidf_train=vectorizer.transform(X_train['project_title'])

print("Shape of matrix after one hot encoding ",title_tfidf_train.shape)
print("="*100)



title_tfidf_cv=vectorizer.transform(X_cv['project_title'])

print("Shape of matrix after one hot encoding ",title_tfidf_cv.shape)
print("="*100)



title_tfidf_test=vectorizer.transform(X_test['project_title'])

print("Shape of matrix after one hot encoding ",title_tfidf_test.shape)
```

```
Shape of matrix after one hot encoding  (22445, 1215)
====================================================================================================

Shape of matrix after one hot encoding  (11055, 1215)
====================================================================================================

Shape of matrix after one hot encoding  (16500, 1215)
```

In [0]:

```
### 2.3.3 AVG W2V
```

In [0]:

```
#ESSAY

# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('/content/drive/My Drive/Mass3/Assignments_DonorsChoose_2018/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_essay_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essays'].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
    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_essay_train.append(vector)

print(len(avg_w2v_essay_train))
print(len(avg_w2v_essay_train[0]))
print(avg_w2v_essay_train[0])
```

```
100%|██████████| 22445/22445 [00:06<00:00, 3323.20it/s]
```

```
22445
300
[ 2.28142010e-02 -6.93313108e-02  3.76597402e-02 -9.94197309e-02
 -3.15723711e-02  9.50108675e-02 -3.07362881e+00  5.86421175e-02
  5.02375289e-02 -2.90509897e-02  3.12117835e-02  8.94666332e-02
  1.74040411e-01 -1.70805258e-01 -1.05483999e-01 -4.15952392e-02
 -1.86696629e-02 -6.42267526e-02 -1.64568660e-02  1.57218278e-02
  4.29314464e-02 -9.20783505e-04 -3.08253299e-02  2.65525392e-02
 -5.61212866e-02 -2.49452113e-02  8.08077526e-02 -7.01005835e-02
 -9.03391031e-02 -6.74663474e-02 -3.44497560e-01 -9.97445278e-02
  1.22592549e-01  8.70661825e-02  1.00452918e-01 -6.36273361e-02
 -7.94133402e-02 -3.60670598e-02  2.31502172e-02 -7.63538258e-02
 -4.66735876e-02  6.80617652e-02  8.94602268e-04 -2.50515247e-01
  1.08990670e-01 -2.42219309e-02  1.14220165e-02  4.20759691e-02
  8.92000406e-02 -8.02857216e-02 -9.11785361e-03  7.76804969e-02
  2.56431515e-02 -2.39741340e-02  6.97898103e-02 -1.19292032e-01
  9.88521649e-04 -1.20509485e-02 -1.09829784e-01  5.50316062e-02
 -1.83548773e-02 -2.61585351e-02  8.36840742e-02 -1.83978557e-03
 -5.10879660e-02  6.16025928e-02  4.42070041e-02 -3.82155845e-02
  2.44459509e-01 -9.02401093e-02 -7.19260629e-02 -5.01348454e-03
  8.53759557e-03 -6.95180412e-02 -7.14029732e-02 -1.24371614e-01
  1.01598680e-02  6.01129278e-02 -3.08082577e-02 -1.94408412e-02
  8.43388969e-02 -2.30355212e-01  2.64929320e-02 -4.42244206e-02
 -1.10542658e-01  3.54360732e-02  7.23176619e-02 -1.11114093e-01
  8.23248423e-02  6.78286948e-02  7.74161546e-02 -1.50123526e-02
 -4.00663897e-02  2.07584639e-02 -2.77204887e-02 -2.66510911e-01
 -2.26775784e+00  3.87249485e-02  8.33559216e-02  2.30843848e-01
 -1.32817376e-01 -7.15392680e-02  1.95038023e-01 -1.14632474e-02
  1.34981004e-01  4.18734742e-02  4.14240010e-02 -9.50328845e-02
  1.34005196e-02  3.23617340e-02  2.08504124e-02 -1.55109856e-01
 -1.92407216e-03  2.15658394e-01  3.68930876e-02  5.67938969e-03
 -2.75248279e-01  1.22747915e-01  9.78846072e-02  9.67085298e-02
 -3.88022680e-02  8.46247907e-02 -7.77330258e-03 -7.92855918e-02
  1.65992268e-01  1.55955608e-02  9.79767381e-02 -1.07090258e-01
  5.50225924e-02  1.83439681e-01  4.93828742e-02 -8.97534330e-03
  6.20891340e-02 -9.84005876e-02 -5.34004330e-03 -1.02520602e-01
  1.60574124e-02 -8.20021856e-02  6.85859990e-02  3.26193375e-01
  1.36896705e-01  4.90571557e-02  7.96243196e-03 -7.24324309e-02
 -4.14411031e-02  1.37634381e-01  8.67780485e-02 -2.14686000e-02
  1.21022113e-01 -4.83590227e-02  2.58123144e-02 -1.60566907e-02
  1.35955019e-01 -9.14728526e-02  9.95332072e-02  1.09019455e-02
  5.88925732e-02 -1.28102029e-01  1.20150639e-02 -2.92650000e-02
  8.05287990e-02  2.71567732e-03  6.32886598e-03 -4.60716845e-02
 -1.97359216e-02  6.14084990e-02 -4.39682608e-02  8.79244928e-02
  2.37623157e-02 -5.35179670e-02 -1.04046247e-01  2.32274485e-02
 -4.65643175e-02 -1.52563759e-01  3.72360465e-02  1.98023485e-02
 -1.81943093e-02  2.50858186e-02 -1.37111835e-01 -8.21680722e-03
 -5.79488577e-03  2.84012753e-01  6.39202402e-02  8.50533959e-02
 -3.29655258e-03 -8.39878784e-02 -9.01932670e-02 -4.34666369e-02
  1.49279732e-01 -5.13852309e-02 -3.74924330e-02 -1.18022093e-01
 -6.64300899e-02  6.72631546e-02  1.79196887e-02 -9.84463711e-02
 -1.43093268e-02  9.92164021e-02  1.23404856e-02  2.97972127e-02
  9.55994742e-02  2.31188124e-02 -1.10119320e-01  1.08488558e-01
 -6.83612505e-02 -1.09813701e-02  1.73082235e-01 -4.21186887e-02
  6.52901227e-02 -8.76282052e-02 -3.85083402e-02 -1.60097320e-02
 -5.77609046e-02 -1.81009347e-01 -1.53321443e-01  5.98979175e-02
 -1.21539106e-01 -8.70334423e-02 -3.74855268e-02 -4.19530495e-02
 -1.82618573e-01 -2.73668598e-02 -1.32649026e-01 -1.92661918e-01
 -2.04423856e+00  1.72139381e-01 -1.36350515e-03  6.89939969e-02
 -6.74384113e-02  1.43957938e-03  3.95864598e-02 -8.92261237e-02
 -3.08405049e-02  1.21681134e-02  2.41131258e-02  9.25498039e-02
```

```
  3.89032022e-02 -1.30236659e-01  8.24371423e-02  1.81094969e-01
 -6.69204742e-02  2.21220856e-02 -1.94992319e-01  2.06658557e-02
  2.60275052e-03 -5.69833165e-02 -3.59842010e-02 -1.30134597e-01
  2.93461763e-03 -6.36396474e-02  4.93630299e-02  9.17515711e-02
  1.08841567e-01 -7.45321546e-02  7.10808175e-02  2.96522722e-02
  1.51705464e-01  5.35948052e-02  5.25585670e-02 -1.13232181e-01
  3.17659794e-02  3.42060000e-02 -6.20586062e-02  2.04603810e-02
  5.16622794e-02 -4.93682808e-02 -1.29974227e-01  7.02744639e-02
  2.50497388e-02  4.80097691e-02 -2.24266113e-02 -1.34113289e-02
 -1.51789806e-01  2.07654227e-03 -4.31512165e-02 -3.23005052e-02
  6.59998320e-02 -6.24173299e-02  3.62309299e-02  8.90339072e-02
 -2.14183907e-02 -1.12805340e-01 -1.20211378e-01  4.90169691e-02
  5.06746794e-02  8.59791476e-02 -2.94678010e-02  6.59600103e-03
  5.02435928e-02 -8.65883918e-02 -6.87465392e-02  2.68645669e-02
 -7.18181320e-02 -8.19646464e-02  4.62694216e-02 -6.51157785e-02
 -2.45226134e-02  1.08739097e-01  1.31270999e-01  4.69178649e-02]
```

In [0]:

```
avg_w2v_essay_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essays'].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_essay_cv.append(vector)
```

```
100%|██████████| 11055/11055 [00:03<00:00, 3049.59it/s]
```

In [0]:

```
avg_w2v_essay_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essays'].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_essay_test.append(vector)
```

```
100%|██████████| 16500/16500 [00:05<00:00, 3273.58it/s]
```

In [0]:

```
#PROJECT_TITLE

# average Word2Vec
# compute average word2vec for each review.
avg_w2v_title_train = []; # 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_title_train.append(vector)

print(len(avg_w2v_title_train))
print(len(avg_w2v_title_train[0]))
print(avg_w2v_title_train[0])
```

```
100%|██████████| 22445/22445 [00:00<00:00, 47188.34it/s]
```

```
22445
300
[-2.873650e-01 -3.163850e-01  1.935595e-01  4.516300e-01  1.120235e-01
  1.499900e-01 -2.672200e+00 -3.968500e-02  3.395955e-01 -1.752350e-01
  1.128340e-01  1.026735e-01  4.435050e-01 -2.162400e-01 -1.500140e-01
  8.315000e-03  2.201000e-02 -7.988500e-02 -2.332680e-01 -8.265000e-03
  2.427810e-01  1.591870e-01  2.818500e-02  5.934700e-02 -1.870000e-02
  8.464500e-02  2.081450e-01 -6.175350e-01 -3.318200e-01 -1.210170e-01
 -2.098450e-01  5.135500e-02 -2.935000e-02 -6.735500e-02 -1.351800e-01
  4.181875e-01  3.715250e-01 -2.634000e-02  1.145300e-01 -2.658730e-01
 -1.804375e-01 -2.289050e-01  3.153600e-01  8.792700e-02  2.586100e-01
 -2.720950e-01  7.089500e-02 -2.830000e-04  1.486110e-01 -4.663000e-02
  3.706000e-01  3.351785e-01 -2.705500e-02 -3.582500e-02  3.416850e-02
 -2.247650e-01  4.265700e-02  1.487500e-01 -3.561305e-01  6.972500e-02
 -8.858800e-02  8.747500e-02 -1.373140e-01 -3.656650e-02  1.963700e-01
  4.121080e-01  3.164050e-01 -2.776850e-01  4.991150e-01 -8.766120e-02
  3.654225e-02  5.681750e-01 -4.730500e-02  1.931950e-01  2.013270e-01
 -4.729900e-01  1.444000e-02  1.484650e-01 -1.373500e-02  1.180275e-01
  4.796600e-01 -5.422110e-01  2.225200e-01  1.237735e-01 -9.230100e-02
  4.047000e-02  4.965900e-01 -1.781120e-01  1.649750e-01 -9.318000e-02
  4.167050e-01 -2.243500e-02 -3.744000e-02 -2.792300e-01  3.918000e-02
  2.064050e-01 -2.602900e+00  4.472600e-01  6.726000e-02  3.297650e-01
 -1.151350e-01 -2.296000e-02 -9.687500e-02 -2.253200e-01 -2.157490e-01
  5.597185e-01 -2.367350e-02 -4.202800e-01  3.089850e-01 -1.662500e-01
 -8.650000e-02  1.451915e-01  1.885350e-01  4.157350e-01 -1.995550e-01
 -1.068250e-01  6.342000e-02  1.266885e-01  1.204450e-01  1.076850e-01
 -2.482000e-02 -1.928900e-01 -3.495250e-01  1.440900e-01  1.623450e-01
  2.820565e-01  4.141950e-02 -3.006250e-01  2.224950e-01  3.173000e-03
  2.781000e-02 -2.080500e-01 -2.464100e-01 -6.111000e-02 -1.786700e-01
 -2.748700e-01  1.620960e-01  3.182250e-01 -6.325500e-02  8.314200e-01
 -1.735505e-01 -1.978711e-01 -3.270550e-01  2.139600e-01 -2.370970e-01
  5.178250e-01 -2.469500e-02  2.949500e-02  1.017600e-01 -1.794500e-02
  3.369380e-01  2.105115e-01 -2.668598e-01 -2.475500e-01  8.924300e-02
 -3.518250e-01 -3.535000e-03 -2.241650e-01  4.899250e-01 -3.847600e-01
  9.599500e-02  7.203050e-02 -2.565500e-01  4.251000e-02  6.431050e-01
  4.140900e-01 -2.686600e-01  2.233550e-01  5.801000e-03 -2.524500e-01
  9.255650e-02  4.024250e-01 -1.705600e-01 -3.822500e-02  7.952000e-02
 -2.511150e-01 -2.489000e-01  3.413100e-01 -4.608700e-01 -1.609380e-01
 -3.616900e-01  9.071900e-01  2.239900e-01  2.481070e-01  3.432175e-01
  4.003200e-01 -9.705700e-02 -2.115700e-01  2.060295e-01  2.004550e-01
  1.891205e-01  1.535625e-01 -7.595950e-01  2.255675e-01 -1.264719e-01
 -4.000500e-01  9.308150e-02  1.756270e-01  1.247815e-01  4.873050e-01
 -2.071200e-01 -5.561600e-02 -2.548095e-01 -3.695500e-02  1.867000e-02
 -2.949600e-01  2.263350e-01 -1.639900e-01  2.944000e-02 -3.117900e-01
 -2.814500e-01 -2.074130e-01 -1.307400e-01 -3.669450e-01  9.935000e-02
  3.645850e-01  5.900500e-02 -6.164000e-02  4.726500e-02 -1.810500e-01
 -2.424495e-01 -1.312800e-01  5.615500e-02 -1.061450e-01 -1.846000e+00
  3.504000e-02  2.895450e-01 -2.214200e-01  2.969350e-01  1.555500e-01
 -4.607580e-01 -3.479400e-01  3.093450e-01 -9.228000e-02  4.810050e-02
 -1.818325e-01  3.032300e-01 -4.950100e-02  3.992350e-01  2.195850e-01
 -2.907285e-01  6.905000e-03  1.624300e-01  2.442100e-01  1.287400e-01
 -4.204900e-01  1.046315e-01 -2.993550e-01  5.130000e-02 -5.392950e-01
 -9.915750e-02  5.130250e-01  2.230000e-01  4.416800e-01 -1.166500e-01
 -7.240050e-02 -1.534300e-01 -9.132055e-02  1.441070e-01 -1.264700e-01
 -1.654900e-01 -8.998250e-02 -3.115250e-01  2.164000e-01 -3.600300e-01
 -2.094150e-01 -4.104600e-02  2.802190e-01  4.288000e-02  5.833000e-02
 -1.209200e-01 -8.880570e-02  5.663450e-02  2.558950e-01  9.772300e-02
  2.110500e-01  2.106250e-01 -2.882000e-01 -1.679850e-01  1.181835e-01
 -3.287965e-01 -1.331720e-01 -2.506950e-01 -3.863650e-01  1.527000e-02
  8.962300e-02  2.180135e-01 -1.537950e-01 -3.844250e-01 -1.148150e-01
  4.733050e-01 -3.895000e-02  6.946000e-03 -6.199300e-02  3.421500e-02
 -3.684880e-01  2.225950e-01  2.420500e-01  2.411900e-01  1.166730e-01]
```

In [0]:

```
avg_w2v_title_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_title_cv.append(vector)
```

```
100%|██████████| 11055/11055 [00:00<00:00, 62003.52it/s]
```

In [0]:

```
avg_w2v_title_test = []; # 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_title_test.append(vector)
```

```
100%|██████████| 16500/16500 [00:00<00:00, 61698.48it/s]
```

In [0]:

```
### 2.3.4 TFIDF W2V
```

In [0]:

```
#ESSAY

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essays'])

# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_essay_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((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_essay_train.append(vector)

print(len(tfidf_w2v_essay_train))
print(len(tfidf_w2v_essay_train[0]))
```

```
100%|██████████| 22445/22445 [00:39<00:00, 569.57it/s]
```

```
22445
300
```

In [0]:

```
tfidf_w2v_matrix_essay_train=np.reshape(np.asarray(tfidf_w2v_essay_train),(22445,300))
```

```
sparse_tfidf_w2v_matrix_essay_train=coo_matrix(tfidf_w2v_matrix_essay_train).tocsr()

print("Shape of matrix after one hot encoding ",sparse_tfidf_w2v_matrix_essay_train.shape)
print("="*100)
```

```
Shape of matrix after one hot encoding  (22445, 300)
====================================================================================================
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_cv['essays'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_essay_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((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_essay_cv.append(vector)

print(len(tfidf_w2v_essay_cv))
print(len(tfidf_w2v_essay_cv[0]))
```

```
100%|██████████| 11055/11055 [00:19<00:00, 560.48it/s]
```

```
11055
300
```

In [0]:

```
tfidf_w2v_matrix_essay_cv=np.reshape(np.asarray(tfidf_w2v_essay_cv),(11055,300))
sparse_tfidf_w2v_matrix_essay_cv=coo_matrix(tfidf_w2v_matrix_essay_cv).tocsr()

print("Shape of matrix after one hot encoding ",sparse_tfidf_w2v_matrix_essay_cv.shape)
print("="*100)
```

```
Shape of matrix after one hot encoding  (11055, 300)
====================================================================================================
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test['essays'])

# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_essay_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((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_essay_test.append(vector)

print(len(tfidf_w2v_essay_test))
print(len(tfidf_w2v_essay_test[0]))
```

```
100%|██████████| 16500/16500 [00:28<00:00, 569.92it/s]
```

```
16500
300
```

In [0]:

```
tfidf_w2v_matrix_essay_test=np.reshape(np.asarray(tfidf_w2v_essay_test),(16500,300))
sparse_tfidf_w2v_matrix_essay_test=coo_matrix(tfidf_w2v_matrix_essay_test).tocsr()

print("Shape of matrix after one hot encoding ",sparse_tfidf_w2v_matrix_essay_test.shape)
```

```
Shape of matrix after one hot encoding  (16500, 300)
```

In [0]:

```
#PROJECT_TITLE
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['project_title'])

# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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_title_train.append(vector)

print(len(tfidf_w2v_title_train))
print(len(tfidf_w2v_title_train[0]))
```

```
100%|██████████| 22445/22445 [00:00<00:00, 23204.85it/s]
```

```
22445
```

```
300
```

In [0]:

```
tfidf_w2v_matrix_title_train=np.reshape(np.asarray(tfidf_w2v_title_train),(22445,300))
sparse_tfidf_w2v_matrix_title_train=coo_matrix(tfidf_w2v_matrix_title_train).tocsr()

print("Shape of matrix after one hot encoding ",sparse_tfidf_w2v_matrix_title_train.shape)
print("="*100)
```

```
Shape of matrix after one hot encoding  (22445, 300)
====================================================================================================
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_cv['project_title'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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_title_cv.append(vector)

print(len(tfidf_w2v_title_cv))
print(len(tfidf_w2v_title_cv[0]))
```

```
100%|██████████| 11055/11055 [00:00<00:00, 24068.48it/s]
```

```
11055
300
```

In [0]:

```
tfidf_w2v_matrix_title_cv=np.reshape(np.asarray(tfidf_w2v_title_cv),(11055,300))
sparse_tfidf_w2v_matrix_title_cv=coo_matrix(tfidf_w2v_matrix_title_cv).tocsr()

print("Shape of matrix after one hot encoding ",sparse_tfidf_w2v_matrix_title_cv.shape)
print("="*100)
```

```
Shape of matrix after one hot encoding  (11055, 300)
====================================================================================================
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test['project_title'])
```

```
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_title_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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_title_test.append(vector)

print(len(tfidf_w2v_title_test))
print(len(tfidf_w2v_title_test[0]))
```

```
100%|██████████| 16500/16500 [00:00<00:00, 31474.46it/s]
```

```
16500
300
```

In [0]:

```
tfidf_w2v_matrix_title_test=np.reshape(np.asarray(tfidf_w2v_title_test),(16500,300))
sparse_tfidf_w2v_matrix_title_test=coo_matrix(tfidf_w2v_matrix_title_test).tocsr()

print("Shape of matrix after one hot encoding ",sparse_tfidf_w2v_matrix_title_test.shape)
```

```
Shape of matrix after one hot encoding  (16500, 300)
```

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

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

In [0]:

```
# Concatinating the features
```

In [0]:

```
### Set 1:
```

In [0]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr1 = hstack((X_train_essay_bow,
X_train_title_bow,X_train_clean_cat_ohe,X_train_clean_subcat_ohe, X_train_state_ohe,
X_train_teacher_ohe, X_train_grade_ohe, X_train_price_norm,X_train_prev_posted)).tocsr()
X_cv1 = hstack((X_cv_essay_bow, X_cv_title_bow,X_cv_clean_cat_ohe,X_cv_clean_subcat_ohe, X_cv_state
_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm,X_cv_prev_posted)).tocsr()
X_te1 = hstack((X_test_essay_bow, X_test_title_bow,X_test_clean_cat_ohe,X_test_clean_subcat_ohe,
X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
X_test_price_norm,X_test_prev_posted)).tocsr()
```

```
print("Final Data matrix")
print(X_tr1.shape, y_train.shape)
print(X_cv1.shape, y_cv.shape)
print(X_te1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 7113) (22445,)
(11055, 7113) (11055,)
(16500, 7113) (16500,)
====================================================================================================
```

In [0]:

```
### Set 2:
```

In [0]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr2 = hstack((essay_tfidf_train,
title_tfidf_train,X_train_clean_cat_ohe,X_train_clean_subcat_ohe, X_train_state_ohe,
X_train_teacher_ohe, X_train_grade_ohe, X_train_price_norm,X_train_prev_posted)).tocsr()
X_cv2 = hstack((essay_tfidf_cv, title_tfidf_cv,X_cv_clean_cat_ohe,X_cv_clean_subcat_ohe, X_cv_state
_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm,X_cv_prev_posted)).tocsr()
X_te2 = hstack((essay_tfidf_test, title_tfidf_test,X_test_clean_cat_ohe,X_test_clean_subcat_ohe,
X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
X_test_price_norm,X_test_prev_posted)).tocsr()

print("Final Data matrix")
print(X_tr2.shape, y_train.shape)
print(X_cv2.shape, y_cv.shape)
print(X_te2.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 10058) (22445,)
(11055, 10058) (11055,)
(16500, 10058) (16500,)
====================================================================================================
```

In [0]:

```
### Set 3:
```

In [0]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_tr3 =
hstack((avg_w2v_essay_train,avg_w2v_title_train,X_train_clean_cat_ohe,X_train_clean_subcat_ohe,
X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_norm,X_train_prev_posted)
).tocsr()
X_cv3 = hstack((avg_w2v_essay_cv,avg_w2v_title_cv,X_cv_clean_cat_ohe,X_cv_clean_subcat_ohe,
X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm,X_cv_prev_posted)).tocsr()
X_te3 = hstack((avg_w2v_essay_test,avg_w2v_title_test,X_test_clean_cat_ohe,X_test_clean_subcat_ohe
, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
X_test_price_norm,X_test_prev_posted)).tocsr()

print("Final Data matrix")
print(X_tr3.shape, y_train.shape)
print(X_cv3.shape, y_cv.shape)
print(X_te3.shape, y_test.shape)
print("="*100)
```

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

In [0]:

```
### Set 4:
```

In [0]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr4 = 
hstack((tfidf_w2v_essay_train,tfidf_w2v_title_train,X_train_clean_cat_ohe,X_train_clean_subcat_ohe
, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, 
X_train_price_norm,X_train_prev_posted)).tocsr()
X_cv4 = hstack((tfidf_w2v_essay_cv,tfidf_w2v_title_cv,X_cv_clean_cat_ohe,X_cv_clean_subcat_ohe, 
X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm,X_cv_prev_posted)).tocsr()
X_te4 = 
hstack((tfidf_w2v_essay_test,tfidf_w2v_title_test,X_test_clean_cat_ohe,X_test_clean_subcat_ohe, X_
test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_norm,X_test_prev_posted)).tocsr
()

print("Final Data matrix")
print(X_tr4.shape, y_train.shape)
print(X_cv4.shape, y_cv.shape)
print(X_te4.shape, y_test.shape)
print("="*100)
```

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

2.4.1 Applying KNN brute force on BOW, SET 1

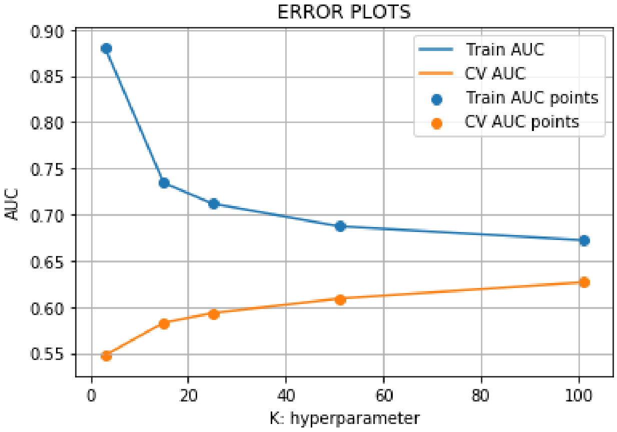
In [0]:

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

    return y_data_pred
```

In [0]:

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

train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    neigh.fit(X_tr1, y_train)

    y_train_pred = batch_predict(neigh, X_tr1)
    y_cv_pred = batch_predict(neigh, X_cv1)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive 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()
```

```
100%|██████████| 5/5 [06:06<00:00, 73.16s/it]
```



In [0]:

```
#### Testing the performance of the model on test data, plotting ROC Curves
```

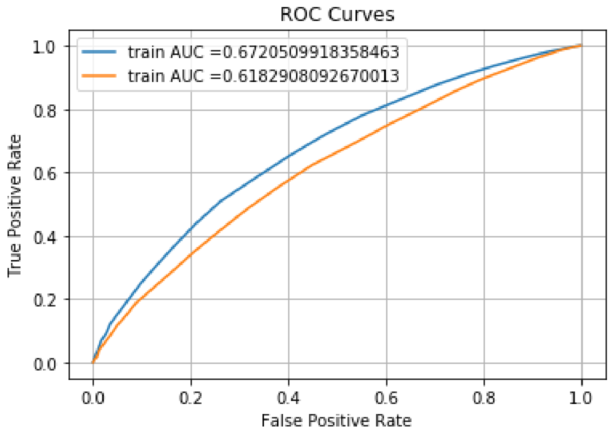
In [0]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between the 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 computing 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 = 101
```

In [0]:

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


neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1,algorithm='brute')
neigh.fit(X_tr1, 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_tr1)
y_test_pred = batch_predict(neigh, X_te1)

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="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves")
plt.grid()
plt.show()
```



In [0]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr

def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

In [0]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

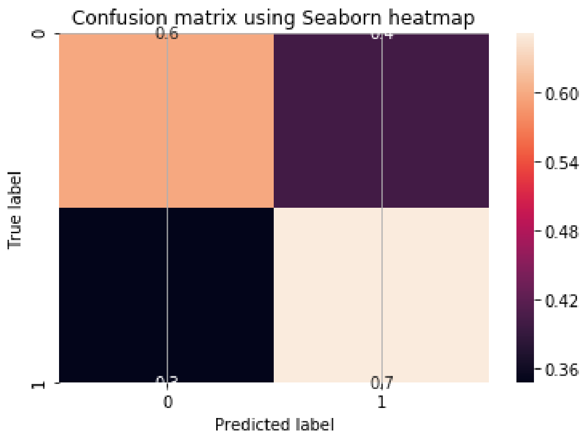
```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.38988328174630993 for threshold 0.772
Train confusion matrix
[[ 2151  1444]
 [ 6567 12283]]
Test confusion matrix
[[1452 1190]
 [5205 8653]]
```

In [0]:

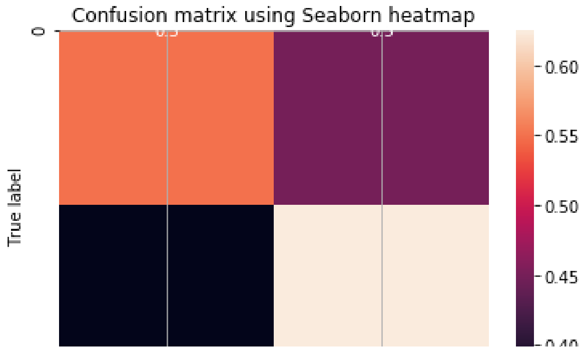
```
#Confusion matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train Confusion matrix")
from sklearn.metrics import confusion_matrix
```

```
conf_mat=confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
print(conf_mat)

#For heatmap

import seaborn as sns
conf_mat=confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
conf_mat_normalized=conf_mat.astype('float')/conf_mat.sum(axis=1)[:,np.newaxis]
sns.heatmap(conf_mat_normalized,annot=True, fmt='.1f')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title("Confusion matrix using Seaborn heatmap")
plt.grid()
plt.show()

print('='*100)

print("Test Confusion matrix")
from sklearn.metrics import confusion_matrix
conf_mat=confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print(conf_mat)

#For heatmap

import seaborn as sns
conf_mat=confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
conf_mat_normalized=conf_mat.astype('float')/conf_mat.sum(axis=1)[:,np.newaxis]
sns.heatmap(conf_mat_normalized,annot=True, fmt= '.1f')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title("Confusion matrix using Seaborn heatmap")
plt.grid()
plt.show()
```

```
Train Confusion matrix
[[ 2151  1444]
 [ 6567 12283]]
```



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

Test Confusion matrix
[[1452 1190]
 [5205 8653]]
```

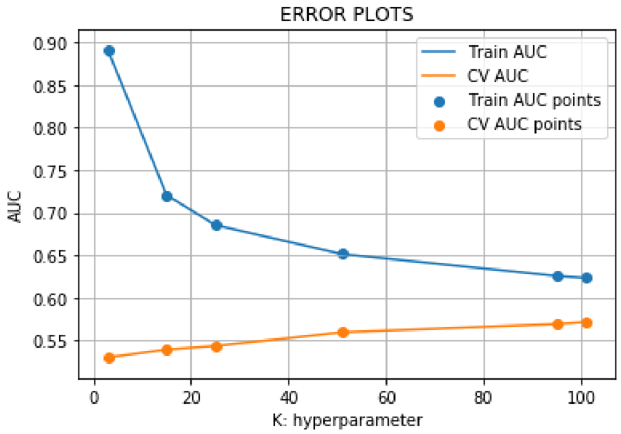


2.4.2 Applying KNN brute force on TFIDF, SET 2

In [0]:

```
train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 95, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    neigh.fit(X_tr2, y_train)

    y_train_pred = batch_predict(neigh, X_tr2)
    y_cv_pred = batch_predict(neigh, X_cv2)

    # 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()
```

```
100%|██████████| 6/6 [06:50<00:00, 68.50s/it]
```



In [0]:

```
#### Testing the performance of the model on test data, plotting ROC Curves
```

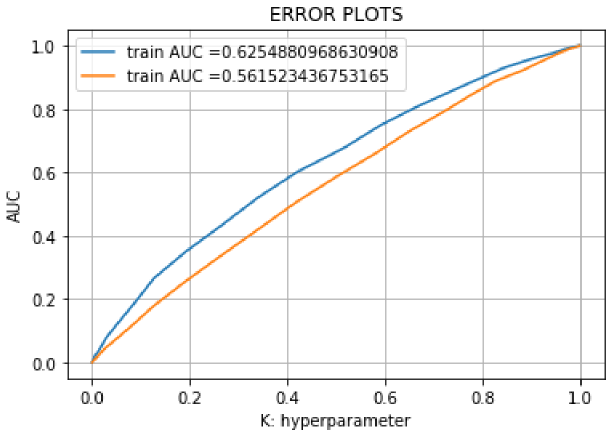
In [0]:

```
# 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_k1 = 95
```

```
best_k1 = 50
```

In [0]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k1, n_jobs=-1)
neigh.fit(X_tr2, 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_pred1 = batch_predict(neigh, X_tr2)
y_test_pred1 = batch_predict(neigh, X_te2)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred1)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred1)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves")
plt.grid()
plt.show()
```



In [0]:

```
print("="*100)

best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred1, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred1, best_t)))
```

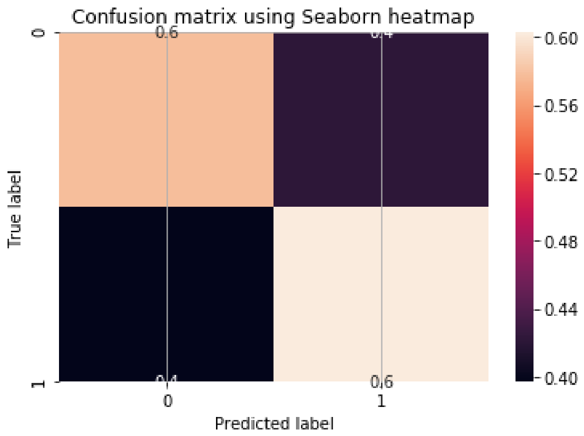
```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.3483175793081313 for threshold 0.842
Train confusion matrix
[[ 2078  1517]
 [ 7491 11359]]
Test confusion matrix
[[1312 1330]
 [5703 8155]]
```

In [0]:

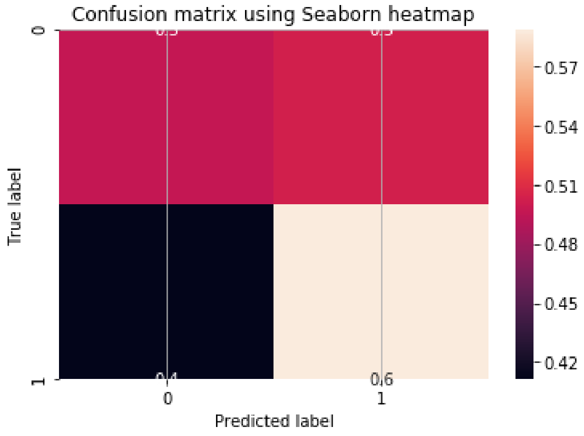
```
#Confusion matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train Confusion matrix")
from sklearn.metrics import confusion_matrix
conf_mat=confusion_matrix(y_train, predict_with_best_t(y_train_pred1, best_t))
print(conf_mat)

#For heatmap
```

```python
import seaborn as sns
conf_mat=confusion_matrix(y_train, predict_with_best_t(y_train_pred1, best_t))
conf_mat_normalized=conf_mat.astype('float')/conf_mat.sum(axis=1)[:,np.newaxis]
sns.heatmap(conf_mat_normalized,annot=True, fmt='.1f')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title("Confusion matrix using Seaborn heatmap")
plt.grid()
plt.show()

print('='*100)

print("Test Confusion matrix")
from sklearn.metrics import confusion_matrix
conf_mat=confusion_matrix(y_test, predict_with_best_t(y_test_pred1, best_t))
print(conf_mat)

#For heatmap

import seaborn as sns
conf_mat=confusion_matrix(y_test, predict_with_best_t(y_test_pred1, best_t))
conf_mat_normalized=conf_mat.astype('float')/conf_mat.sum(axis=1)[:,np.newaxis]
sns.heatmap(conf_mat_normalized,annot=True, fmt= '.1f')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title("Confusion matrix using Seaborn heatmap")
plt.grid()
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.3483175793081313 for threshold 0.842
Train Confusion matrix
[[ 2078  1517]
 [ 7491 11359]]
```



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

Test Confusion matrix
[[1312 1330]
 [5703 8155]]
```

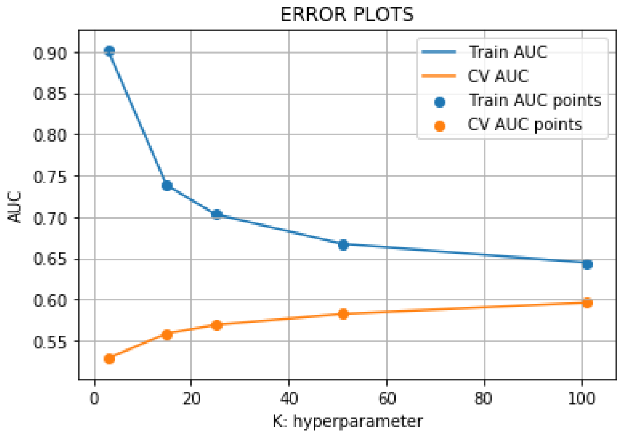


2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [0]:

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

train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    neigh.fit(X_tr3, y_train)

    y_train_pred = batch_predict(neigh, X_tr3)
    y_cv_pred = batch_predict(neigh, X_cv3)

    # 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()
```

```
100%|██████████| 5/5 [1:25:37<00:00, 1027.62s/it]
```



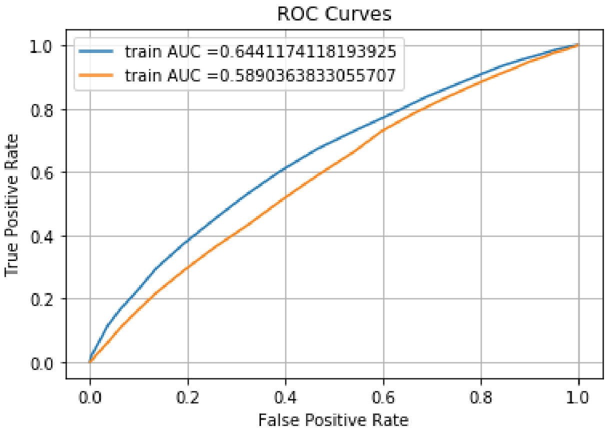
Testing the performance of the model on test data, plotting ROC Curves

In [0]:

```
# 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_k2 = 101
```

In [0]:

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


neigh = KNeighborsClassifier(n_neighbors=best_k2, n_jobs=-1)
neigh.fit(X_tr3, 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_pred2 = batch_predict(neigh, X_tr3)
y_test_pred2 = batch_predict(neigh, X_te3)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred2)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred2)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves")
plt.grid()
plt.show()
```



In [0]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred2, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred2, best_t)))
```

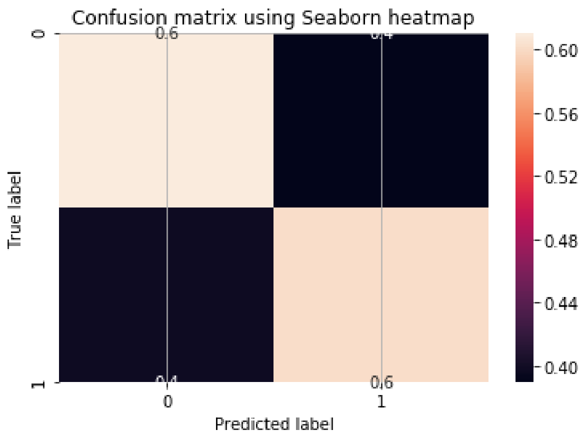
```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.36736752710624465 for threshold 0.851
Train confusion matrix
[[ 2193  1402]
 [ 7498 11352]]
Test confusion matrix
[[1398 1244]
 [5632 8226]]
```

In [0]:

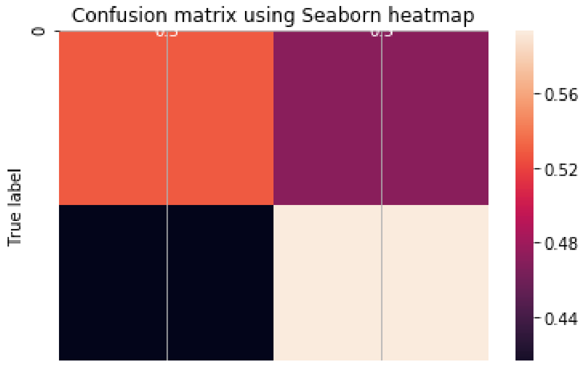
```
#Confusion matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train Confusion matrix")
from sklearn.metrics import confusion_matrix
conf_mat=confusion_matrix(y_train, predict_with_best_t(y_train_pred2, best_t))
```

```
print(conf_mat)

#For heatmap

import seaborn as sns
conf_mat=confusion_matrix(y_train, predict_with_best_t(y_train_pred2, best_t))
conf_mat_normalized=conf_mat.astype('float')/conf_mat.sum(axis=1)[:,np.newaxis]
sns.heatmap(conf_mat_normalized,annot=True, fmt='.1f')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title("Confusion matrix using Seaborn heatmap")
plt.grid()
plt.show()

print('='*100)

print("Test Confusion matrix")
from sklearn.metrics import confusion_matrix
conf_mat=confusion_matrix(y_test, predict_with_best_t(y_test_pred2, best_t))
print(conf_mat)

#For heatmap

import seaborn as sns
conf_mat=confusion_matrix(y_test, predict_with_best_t(y_test_pred2, best_t))
conf_mat_normalized=conf_mat.astype('float')/conf_mat.sum(axis=1)[:,np.newaxis]
sns.heatmap(conf_mat_normalized,annot=True, fmt= '.1f')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title("Confusion matrix using Seaborn heatmap")
plt.grid()
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.36736752710624465 for threshold 0.851
Train Confusion matrix
[[ 2193  1402]
 [ 7498 11352]]
```



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

Test Confusion matrix
[[1398 1244]
 [5632 8226]]
```

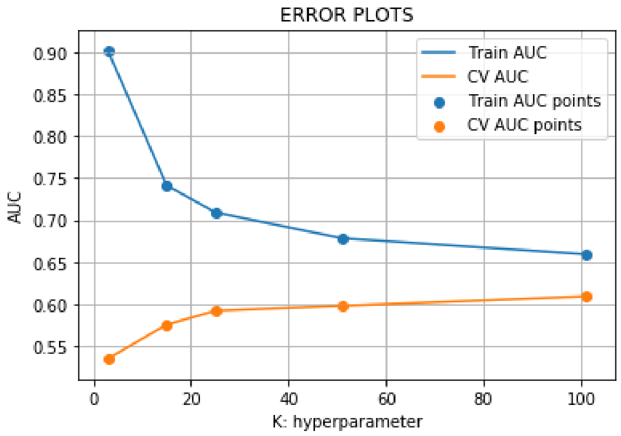


2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [0]:

```
train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1,algorithm='brute')
    neigh.fit(X_tr4, y_train)

    y_train_pred = batch_predict(neigh, X_tr4)
    y_cv_pred = batch_predict(neigh, X_cv4)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive 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()
```

```
100%|██████████| 5/5 [1:25:29<00:00, 1025.93s/it]
```



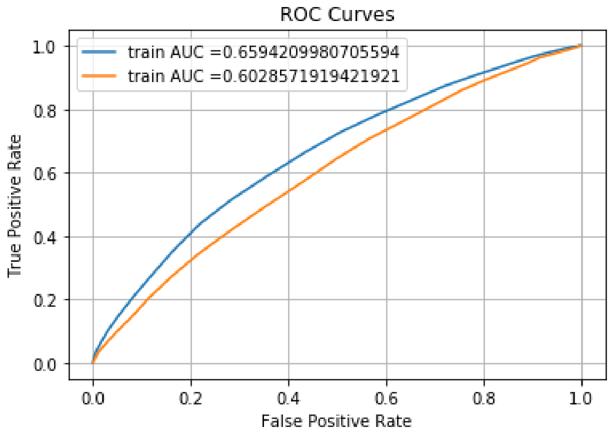
Testing the performance of the model on test data, plotting ROC Curves

In [0]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between the 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 computing 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_k3 = 101
```

In [0]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k3, n_jobs=-1,algorithm='brute')
neigh.fit(X_tr4, 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_pred3 = batch_predict(neigh, X_tr4)
y_test_pred3 = batch_predict(neigh, X_te4)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred3)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred3)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves")
plt.grid()
plt.show()
```



In [0]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred3, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred3, best_t)))
```

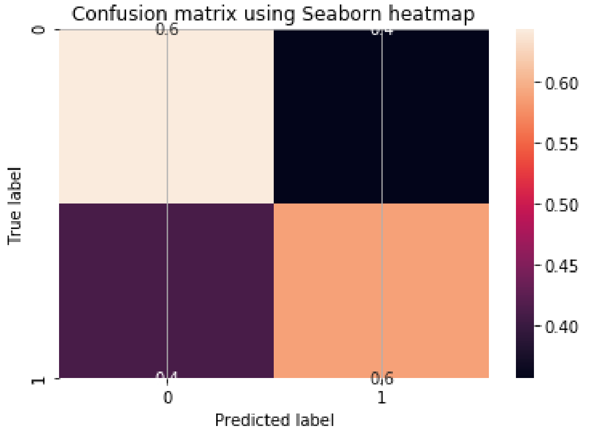
```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.3786704640618602 for threshold 0.851
Train confusion matrix
[[ 2312  1283]
 [ 7751 11099]]
Test confusion matrix
[[1509 1133]
 [5969 7889]]
```

In [0]:

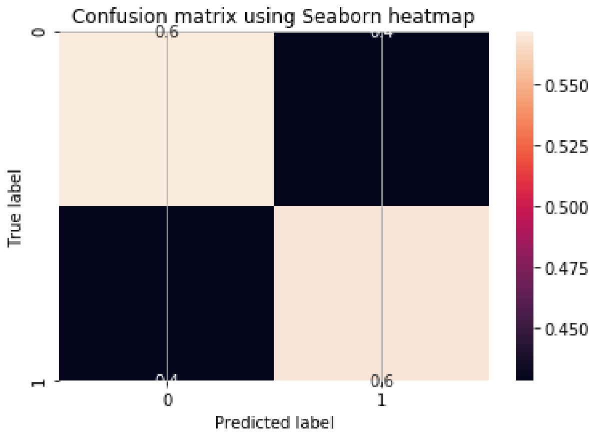
```
#Confusion matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train Confusion matrix")
from sklearn.metrics import confusion_matrix
conf_mat=confusion_matrix(y_train, predict_with_best_t(y_train_pred3, best_t))
print(conf_mat)

#For heatmap

import seaborn as sns
conf_mat=confusion_matrix(y_train, predict_with_best_t(y_train_pred3, best_t))
```

```
conf_mat=confusion_matrix(y_train, predict_with_best_t(y_train_pred3, best_t))
conf_mat_normalized=conf_mat.astype('float')/conf_mat.sum(axis=1)[:,np.newaxis]
sns.heatmap(conf_mat_normalized,annot=True, fmt='.1f')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title("Confusion matrix using Seaborn heatmap")
plt.grid()
plt.show()

print('='*100)

print("Test Confusion matrix")
from sklearn.metrics import confusion_matrix
conf_mat=confusion_matrix(y_test, predict_with_best_t(y_test_pred3, best_t))
print(conf_mat)

#For heatmap

import seaborn as sns
conf_mat=confusion_matrix(y_test, predict_with_best_t(y_test_pred3, best_t))
conf_mat_normalized=conf_mat.astype('float')/conf_mat.sum(axis=1)[:,np.newaxis]
sns.heatmap(conf_mat_normalized,annot=True, fmt= '.1f')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title("Confusion matrix using Seaborn heatmap")
plt.grid()
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.3786704640618602 for threshold 0.851
Train Confusion matrix
[[ 2312  1283]
 [ 7751 11099]]
```



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

Test Confusion matrix
[[1509 1133]
 [5969 7889]]
```



2.5 Feature selection with `SelectKBest`

In [0]:

```
#https://www.programcreek.com/python/example/93974/sklearn.feature_selection.SelectKBest

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


print("Before transform:",X_tr2.shape)
selector = SelectKBest(score_func=chi2, k=2000)
selector.fit(X_tr2, y_train)

print("selected index:", selector.get_support(True))
print("After transform:")

X_tr_top_2000= selector.transform(X_tr2)
X_cv_top_2000= selector.transform(X_cv2)
X_te_top_2000= selector.transform(X_te2)
X_tr_top_2000.shape
```

```
Before transform: (22445, 10058)
selected index: [    2    15    22 ... 10054 10056 10057]
After transform:
```

Out[0]:

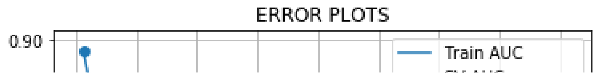
```
(22445, 2000)
```

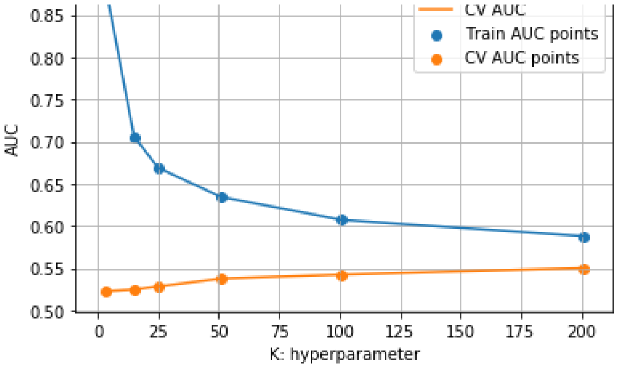
Hyper parameter tuning after selecting top 2000 features

In [0]:

```
#Hyper parameter tuning

train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101, 201]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr_top_2000, y_train)

    y_train_pred = batch_predict(neigh, X_tr_top_2000)
    y_cv_pred = batch_predict(neigh, X_cv_top_2000)

    # 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()
```

```
100%|██████████| 6/6 [05:24<00:00, 54.21s/it]
```





In [0]:

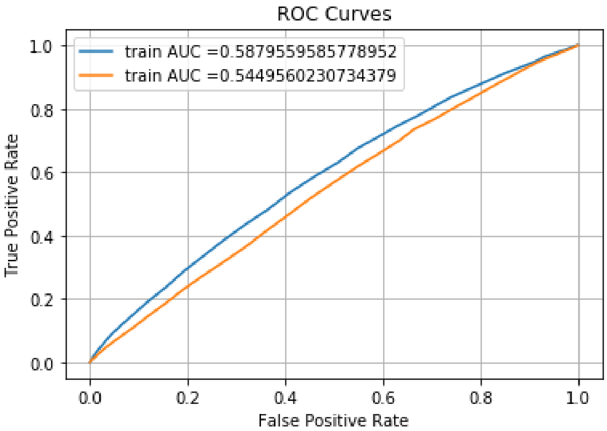
```
#### Testing the performance of the model on test data, plotting ROC Curves
```

In [0]:

```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between the 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 computing 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_k4 = 201
```

In [0]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k4, n_jobs=-1)
neigh.fit(X_tr_top_2000, 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_pred4 = batch_predict(neigh, X_tr_top_2000)
y_test_pred4 = batch_predict(neigh, X_te_top_2000)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred4)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred4)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves")
plt.grid()
plt.show()
```



In [0]:

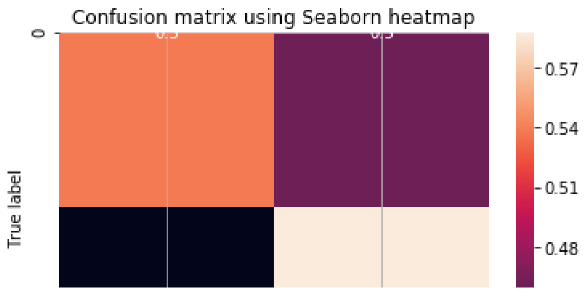
```
print("="*100)

best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred4, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred4, best_t)))
```

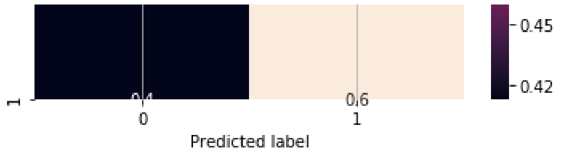
```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.31623030808336067 for threshold 0.841
Train confusion matrix
[[ 1936  1659]
 [ 7781 11069]]
Test confusion matrix
[[1310 1332]
 [5934 7924]]
```

In [0]:

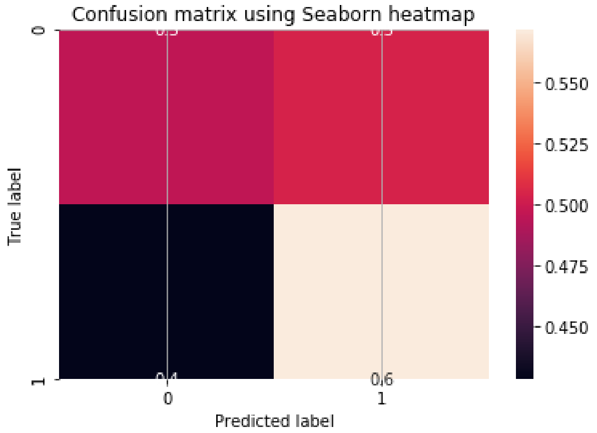
```
#Confusion matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train Confusion matrix")
from sklearn.metrics import confusion_matrix
conf_mat=confusion_matrix(y_train, predict_with_best_t(y_train_pred4, best_t))
print(conf_mat)

#For heatmap

import seaborn as sns
conf_mat=confusion_matrix(y_train, predict_with_best_t(y_train_pred4, best_t))
conf_mat_normalized=conf_mat.astype('float')/conf_mat.sum(axis=1)[:,np.newaxis]
sns.heatmap(conf_mat_normalized,annot=True, fmt='.1f')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title("Confusion matrix using Seaborn heatmap")
plt.grid()
plt.show()

print('='*100)

print("Test Confusion matrix")
from sklearn.metrics import confusion_matrix
conf_mat=confusion_matrix(y_test, predict_with_best_t(y_test_pred4, best_t))
print(conf_mat)

#For heatmap

import seaborn as sns
conf_mat=confusion_matrix(y_test, predict_with_best_t(y_test_pred4, best_t))
conf_mat_normalized=conf_mat.astype('float')/conf_mat.sum(axis=1)[:,np.newaxis]
sns.heatmap(conf_mat_normalized,annot=True, fmt= '.1f')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title("Confusion matrix using Seaborn heatmap")
plt.grid()
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.31623030808336067 for threshold 0.841
Train Confusion matrix
[[ 1936  1659]
 [ 7781 11069]]
```





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

Test Confusion matrix
[[1310 1332]
 [5934 7924]]
```



3. Conclusions

In [1]:

```
#http://zetcode.com/python/prettytable/

from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]

x.add_row(["BOW", "Brute", 101, 0.772])
x.add_row(["TFIDF", "Brute", 95, 0.842])
x.add_row(["W2V", "Brute", 101, 0.85])
x.add_row(["TFIDFW2V", "Brute", 101, 0.85])
x.add_row(["TFIDF-kBest", "Brute", 201, 0.841])

print(x)
```

```
+-------------+-------+-----------------+-------+
|  Vectorizer | Model | Hyper Parameter |  AUC  |
+-------------+-------+-----------------+-------+
|     BOW     | Brute |       101       | 0.772 |
|    TFIDF    | Brute |        95       | 0.842 |
|     W2V     | Brute |       101       |  0.85 |
|   TFIDFW2V  | Brute |       101       |  0.85 |
| TFIDF-kBest | Brute |       201       | 0.841 |
+-------------+-------+-----------------+-------+
```