# 9_DonorsChoose - Ensemble

October 12, 2019

## 1 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

```
<li>How to increase the consistency of project vetting across different volunteers to improve t
<li>How to focus volunteer time on the applications that need the most assistance</li>
</ul>
```

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.

### 1.1 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!</code

`project_essay_1` | First application essay

***`project_essay_2`*** | *Second application essay* `project_essay_3` | Third application essay ***`project_essay_4`*** | *Fourth application essay* `project_submitted_datetime` | Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`

`teacher_id` | A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`

`teacher_prefix` | Teacher's title. One of the following enumerated values:

nan
Dr.
Mr.
Mrs.
Ms.
Teacher.

`teacher_number_of_previously_posted_projects` | Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

### 1.1.1 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.

[251]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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

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

import re
```

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

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

from tqdm import tqdm
import os

from collections import Counter
```

### 1.2 1.1 Reading Data

[252]:
```
project_data = pd.read_csv('train_data.csv',nrows=50000)
resource_data = pd.read_csv('resources.csv')
```

[253]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

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

[254]:
```
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']
```

[254]:

| | 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 | |

```
1    14.95
```

### 1.3 1.2 preprocessing of `project_subject_categories`

[255]:
```
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 & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on␣
 ↪space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to␣
 ↪replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with␣
 ↪''(empty) ex:"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]))
```

### 1.4 1.3 preprocessing of `project_subject_subcategories`

[256]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.
 ↪com/a/47301924/4084039

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science",␣
 ↪"Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on␣
 ↪space "Math & Science"=> "Math","&", "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]))
```

## 1.5 1.3 Text preprocessing

[257]:
```
# 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)
```

[258]:
```
project_data.head(2)
```

[258]:
```
   Unnamed: 0       id                         teacher_id teacher_prefix  \
0      160221  p253737  c90749f5d961ff158d4b4d1e7dc665fc           Mrs.
1      140945  p258326  897464ce9ddc600bced1151f324dd63a            Mr.

  school_state project_submitted_datetime project_grade_category  \
0           IN        2016-12-05 13:43:57          Grades PreK-2
1           FL        2016-10-25 09:22:10             Grades 6-8

                                  project_title  \
0  Educational Support for English Learners at Home
1             Wanted: Projector for Hungry Learners

                                  project_essay_1  \
0  My students are English learners that are work...
1  Our students arrive to our school eager to lea...

                                  project_essay_2 project_essay_3  \
0  \"The limits of your language are the limits o...             NaN
1  The projector we need for our school is very c...             NaN

  project_essay_4                         project_resource_summary  \
0             NaN  My students need opportunities to practice beg...
1             NaN  My students need a projector to help with view...

   teacher_number_of_previously_posted_projects  project_is_approved  \
0                                             0                   0
1                                             7                   1

              clean_categories           clean_subcategories  \
0            Literacy_Language                   ESL Literacy
1  History_Civics Health_Sports  Civics_Government TeamSports

                                              essay
0  My students are English learners that are work...
1  Our students arrive to our school eager to lea...
```

[259]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

[260]:
```
# 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])
```

```
My students are English learners that are working on English as their second or
third languages. We are a melting pot of refugees, immigrants, and native-born
Americans bringing the gift of language to our school. \r\n\r\n We have over 24
languages represented in our English Learner program with students at every
level of mastery.  We also have over 40 countries represented with the families
within our school.  Each student brings a wealth of knowledge and experiences to
us that open our eyes to new cultures, beliefs, and respect.\"The limits of your
language are the limits of your world.\"-Ludwig Wittgenstein  Our English
learner's have a strong support system at home that begs for more resources.
Many times our parents are learning to read and speak English along side of
their children.  Sometimes this creates barriers for parents to be able to help
their child learn phonetics, letter recognition, and other reading
skills.\r\n\r\nBy providing these dvd's and players, students are able to
continue their mastery of the English language even if no one at home is able to
assist.  All families with students within the Level 1 proficiency status, will
be a offered to be a part of this program.  These educational videos will be
specially chosen by the English Learner Teacher and will be sent home regularly
to watch.  The videos are to help the child develop early reading
skills.\r\n\r\nParents that do not have access to a dvd player will have the
opportunity to check out a dvd player to use for the year.  The plan is to use
these videos and educational dvd's for the years to come for other EL
students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this year all
love learning, at least most of the time. At our school, 97.3% of the students
receive free or reduced price lunch. Of the 560 students, 97.3% are minority
students. \r\nThe school has a vibrant community that loves to get together and
celebrate. Around Halloween there is a whole school parade to show off the
beautiful costumes that students wear. On Cinco de Mayo we put on a big festival
with crafts made by the students, dances, and games. At the end of the year the
school hosts a carnival to celebrate the hard work put in during the school
year, with a dunk tank being the most popular activity.My students will use
these five brightly colored Hokki stools in place of regular, stationary,
4-legged chairs. As I will only have a total of ten in the classroom and not
enough for each student to have an individual one, they will be used in a
variety of ways. During independent reading time they will be used as special
chairs students will each use on occasion. I will utilize them in place of
chairs at my small group tables during math and reading times. The rest of the
```

day they will be used by the students who need the highest amount of movement in their life in order to stay focused on school.\r\n\r\nWhenever asked what the classroom is missing, my students always say more Hokki Stools. They can't get their fill of the 5 stools we already have. When the students are sitting in group with me on the Hokki Stools, they are always moving, but at the same time doing their work. Anytime the students get to pick where they can sit, the Hokki Stools are the first to be taken. There are always students who head over to the kidney table to get one of the stools who are disappointed as there are not enough of them. \r\n\r\nWe ask a lot of students to sit for 7 hours a day. The Hokki stools will be a compromise that allow my students to do desk work and move at the same time. These stools will help students to meet their 60 minutes a day of movement by allowing them to activate their core muscles for balance while they sit. For many of my students, these chairs will take away the barrier that exists in schools for a child who can't sit still.nannan
==========
How do you remember your days of school? Was it in a sterile environment with plain walls, rows of desks, and a teacher in front of the room? A typical day in our room is nothing like that. I work hard to create a warm inviting themed room for my students look forward to coming to each day.\r\n\r\nMy class is made up of 28 wonderfully unique boys and girls of mixed races in Arkansas.\r\nThey attend a Title I school, which means there is a high enough percentage of free and reduced-price lunch to qualify. Our school is an \"open classroom\" concept, which is very unique as there are no walls separating the classrooms. These 9 and 10 year-old students are very eager learners; they are like sponges, absorbing all the information and experiences and keep on wanting more.With these resources such as the comfy red throw pillows and the whimsical nautical hanging decor and the blue fish nets, I will be able to help create the mood in our classroom setting to be one of a themed nautical environment. Creating a classroom environment is very important in the success in each and every child's education. The nautical photo props will be used with each child as they step foot into our classroom for the first time on Meet the Teacher evening. I'll take pictures of each child with them, have them developed, and then hung in our classroom ready for their first day of 4th grade.  This kind gesture will set the tone before even the first day of school! The nautical thank you cards will be used throughout the year by the students as they create thank you cards to their team groups.\r\n\r\nYour generous donations will help me to help make our classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of my own pocket on resources to get our classroom ready. Please consider helping with this project to make our new school year a very successful one. Thank you!nannan
==========
My kindergarten students have varied disabilities ranging from speech and language delays, cognitive delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work their hardest working past their limitations. \r\n\r\nThe materials we have are the ones I seek out for my students. I teach in a Title I school where most of the students receive free or reduced price lunch.  Despite their disabilities and limitations, my students love coming to school and come eager to learn and explore.Have you ever felt

like you had ants in your pants and you needed to groove and move as you were in a meeting? This is how my kids feel all the time. The want to be able to move as they learn or so they say.Wobble chairs are the answer and I love then because they develop their core, which enhances gross motor and in Turn fine motor skills. \r\nThey also want to learn through games, my kids don't want to sit and do worksheets. They want to learn to count by jumping and playing. Physical engagement is the key to our success. The number toss and color and shape mats can make that happen. My students will forget they are doing work and just have the fun a 6 year old deserves.nannan

[261]:
```
# 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
```

[262]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

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

```
the fun a 6 year old deserves.nannan
==================================================
```

[263]:
```
# \r \n \t remove from string python: http://texthandler.com/info/
 ↪remove-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

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

[264]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

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

[265]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',␣
 ↪"you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',␣
 ↪'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its',␣
 ↪'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this',␣
 ↪'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',␣
 ↪'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',␣
 ↪'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',␣
 ↪'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',␣
 ↪'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how',␣
 ↪'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so',␣
 ↪'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should',␣
 ↪"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',␣
 ↪"didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't",␣
 ↪'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',␣
 ↪"shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

[266]:
```
# Combining all the above stundents
from tqdm import tqdm
import num2words
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(r"(\d+)", lambda x: num2words.num2words(int(x.group(0))),␣
 ↪sent)
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
```

```
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|| 50000/50000 [00:50<00:00, 984.96it/s]
```

[267]:
```
# after preprocesing
preprocessed_essays[20000]
```

[267]:
```
'my kindergarten students varied disabilities ranging speech language delays
cognitive delays gross fine motor delays autism they eager beavers always strive
work hardest working past limitations the materials ones i seek students i teach
title i school students receive free reduced price lunch despite disabilities
limitations students love coming school come eager learn explore have ever felt
like ants pants needed groove move meeting this kids feel time the want able
move learn say wobble chairs answer i love develop core enhances gross motor
turn fine motor skills they also want learn games kids not want sit worksheets
they want learn count jumping playing physical engagement key success the number
toss color shape mats make happen my students forget work fun six year old
deserves nannan'
```

### 1.4 Preprocessing of project_title

[268]:
```
# similarly you can preprocess the titles also
# Combining all the above statemennts
#Replacing number with text https://stackoverflow.com/questions/40401777/
 ↪search-and-replace-numbers-with-words-in-file
#re.sub(r"(\d+)", lambda x: num2words.num2words(int(x.group(0))), sent)
from tqdm import tqdm
import num2words
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(r"(\d+)", lambda x: num2words.num2words(int(x.group(0))),␣
 ↪sent)
    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%|| 50000/50000 [00:02<00:00, 20655.03it/s]
```

[269]:
```
#process project grade remove replace space with _ and replace - with _
preprocessed_grade = []
for sentance in  tqdm(project_data['project_grade_category'].values):
```

```
    sentance=sentance.replace(" ","_")
    sentance = sentance.replace("-","_")
    preprocessed_grade.append(sentance)
list(set(preprocessed_grade))
```

```
100%|| 50000/50000 [00:00<00:00, 354358.79it/s]
```

[269]:
```
['Grades_PreK_2', 'Grades_6_8', 'Grades_9_12', 'Grades_3_5']
```

### 1.4.1 Combining Resource data and project data‘

[270]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).
 ↪reset_index()
print(project_data.columns)
project_data = pd.merge(project_data, price_data, on='id', how='left')
print(project_data.columns)
```

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

### 1.4.2 Finding wordcount of essay and title and Sentiment Analysis of essay‘

[271]:
```
project_data['essay'] = preprocessed_essays
project_data['project_title']= preprocessed_title
project_data['project_grade_category']=preprocessed_grade
project_data['title_wordcount'] = project_data.project_title.apply(lambda x:␣
 ↪len(str(x).split(' ')))
project_data['essay_wordcount'] = project_data.essay.apply(lambda x: len(str(x).
 ↪split(' ')))
```

[272]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')

sentiment_value=[]
sid = SentimentIntensityAnalyzer()
```

```
for essay in tqdm(project_data['essay']):
    senti_score = sid.polarity_scores(essay)
    # we can use these 4 things as features/attributes (neg, neu, pos,␣
 ↪compound)
    # neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93

    sentiment_value.append(int(senti_score['pos'] *100) )

project_data['essay_sentiment_score'] = sentiment_value
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\5558\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
100%|| 50000/50000 [02:37<00:00, 317.84it/s]
```

### [1.4.3] Splitting data into Train and cross validation(or test): Stratified Sampling

[299]:
```
# train test split for BOW and TFIDF
y_label= project_data['project_is_approved']
X_label = project_data.drop(columns='project_is_approved')

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_label, y_label,␣
 ↪test_size=0.33, stratify=y_label)
```

[300]:
```
# train test split for word2v and TFIDF W2V
X_label1 = X_label.head(20000)
y_label1= y_label.head(20000)

from sklearn.model_selection import train_test_split
X_train_mini, X_test_mini, y_train_mini, y_test_mini =␣
 ↪train_test_split(X_label1, y_label1, test_size=0.33, stratify=y_label1)
```

[ ]:
```

```

[301]:
```
print(X_train.shape)
print(X_test.shape)

print(X_train_mini.shape)
print(X_test_mini.shape)
```

```
(33500, 22)
(16500, 22)
(13400, 22)
(6600, 22)
```

[302]:
```
X_train.columns
```

[302]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'essay', 'price', 'quantity', 'title_wordcount',
       'essay_wordcount', 'essay_sentiment_score'],
      dtype='object')
```

### 1.6 1.5 Preparing data for models

[303]:
```
project_data.columns
```

[303]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_title',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price', 'quantity',
       'title_wordcount', 'essay_wordcount', 'essay_sentiment_score'],
      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
```

#### 1.6.1 1.5.1 Vectorizing Categorical data

#### 1.6.2 1.5.1.1 Vectorizing Categorical data for BOW and TFIDF

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

For classification tasks, we use P(y_i=1 | categorical_value) as the mean-replacement strategy. Let us say, we have 100 data points across the total train_data that have a categorical_value (c) for a feature (f). Now, amongst these 100 data-points, if 67 of them belong to class 1, then P(y_i=1 | f=c) = 67/100=0.67, Now, for feature f, we will replace all rows which contain f=c as f=0.67. This

is equivalent to prob(y=1| category). We can use this probability to represent a given category numerically.

For a category which is not there in train data and present in test data, we will encode them with default values Ex: in our test data if have State: D then we encode it as [0.5, 0.05]

Points to Note:

1) While converting categorical to numerical using probabilty score, we have to handle the categories present only in test data and not in the train data we have to set 0.5 as proabability score

2) We have to find the columns which has 'nan' we have to set as 0.0 as probability score for nan

Checking if any nan is present in categorical columns, so that we can handle it

[304]:
```
#https://stackoverflow.com/questions/36226083/
 ↪how-to-find-which-columns-contain-any-nan-value-in-pandas-dataframe-python
X_train.columns[X_train.isnull().any()].tolist()
```

[304]:
```
['teacher_prefix', 'project_essay_3', 'project_essay_4']
```

[305]:
```
#First we are creating a column approved and filling it will data from y_train␣
 ↪as we need both to find probability
X_train['approved']=y_train
print(X_train.columns)

X_test['approved']=y_test
print(X_test.columns)
```

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

[306]:
```
import math
def truncate(number, digits) -> float:
    stepper = 10.0 ** digits
    return math.trunc(stepper * number) / stepper
truncate(0.324325235, 4)
```

[306]:
```
0.3243
```

[309]:
```
# we use probability scores to convert categorical to numerical values
clean_cat_set = X_train['clean_categories'].values
unique_item_train = list(set(clean_cat_set))

clean_cat_set = X_test['clean_categories'].values
unique_item_test = list(set(clean_cat_set))

#Difference of items present in test but not in train
#https://stackoverflow.com/questions/3462143/get-difference-between-two-lists
print("Difference of item present in test not in␣
 ↪train",list(set(unique_item_test) - set(unique_item_train)))

proba_score_1 = {}
proba_score_0 = {}

for i in unique_item_train:
    total_i = len(X_train[(X_train.clean_categories == i)])
    proba_score_1[i]=len(X_train[(X_train.clean_categories == i) & (X_train.
 ↪approved ==1)])/total_i
    proba_score_1[i] = truncate(proba_score_1[i],4)

    proba_score_0[i]=len(X_train[(X_train.clean_categories == i) & (X_train.
 ↪approved ==0)])/total_i
    proba_score_0[i] = truncate(proba_score_0[i],4)

    #way1 We can directly replace the clean_categories column with probability␣
 ↪score calculated above by using the below code
    # X_train.loc[(X_train.clean_categories == i) & (X_train.approved ==1),␣
 ↪'clean_categories'] = clean_cat_1[i]

    #way2 we can create a new column categories_proba and store the␣
 ↪corresponding probability scores
    X_train.loc[(X_train.clean_categories == i) & (X_train.approved ==1),␣
 ↪'categories_p1'] = proba_score_1[i]
    X_train.loc[(X_train.clean_categories == i) & (X_train.approved ==0),␣
 ↪'categories_p0']  = proba_score_0[i]

    #Creating a new column in X_test and corresponding to the category and␣
 ↪y_testvalue, including the probab
    X_test.loc[(X_test.clean_categories == i) & (X_test.approved ==1),␣
 ↪'categories_p1'] = proba_score_1[i]
    X_test.loc[(X_test.clean_categories == i) & (X_test.approved ==0),␣
 ↪'categories_p0']  = proba_score_0[i]
```

```
#https://stackoverflow.com/questions/41402861/
 ↪trying-to-divide-a-dataframe-column-by-a-float-yields-nan
X_train["categories_p0"].fillna(0.0, inplace = True)
X_test["categories_p0"].fillna(0.0, inplace = True)
X_train["categories_p1"].fillna(0.0, inplace = True)
X_test["categories_p1"].fillna(0.0, inplace = True)


#we are creating a 1D array
train_categories_p0 = X_train['categories_p0'].values.reshape(-1,1)
train_categories_p1 = X_train['categories_p1'].values.reshape(-1,1)
print(train_categories_p0.shape)

test_categories_p0 = X_test['categories_p0'].values.reshape(-1,1)
test_categories_p1 = X_test['categories_p1'].values.reshape(-1,1)
test_categories_p0.shape
```

```
Difference of item present in test not in train []
(33500, 1)
```

[309]:
```
(16500, 1)
```

[311]:
```
# we use probability scores to convert clean_subcategories to numerical values
clean_sub_cat_set = X_train['clean_subcategories'].values
unique_item_train = list(set(clean_sub_cat_set))

clean_sub_cat_set = X_test['clean_subcategories'].values
unique_item_test = list(set(clean_sub_cat_set))

#Difference of items present in test but not in train
print("Difference of item present in test not in train␣
 ↪",list(set(unique_item_test) - set(unique_item_train)))
diff_item= list(set(unique_item_test)- set(unique_item_train))

proba_score_1 = {}
proba_score_0 = {}

for i in unique_item_train:
    total_i = len(X_train[(X_train.clean_subcategories == i)])
    proba_score_1[i]=len(X_train[(X_train.clean_subcategories == i) & (X_train.
 ↪approved ==1)])/total_i
    proba_score_1[i] = truncate(proba_score_1[i],4)

    proba_score_0[i]=len(X_train[(X_train.clean_subcategories == i) & (X_train.
 ↪approved ==0)])/total_i
    proba_score_0[i] = truncate(proba_score_0[i],4)
```

```
    #we can create a new column subcategories_proba and store the corresponding␣
 ↪probability scores
    X_train.loc[(X_train.clean_subcategories == i) & (X_train.approved ==1),␣
 ↪'subcategories_p1'] = proba_score_1[i]
    X_train.loc[(X_train.clean_subcategories == i) & (X_train.approved ==0),␣
 ↪'subcategories_p0']  = proba_score_0[i]

    #Creating a new column in X_test and corresponding to the category and␣
 ↪y_testvalue, including the probab.
    X_test.loc[(X_test.clean_subcategories == i) & (X_test.approved ==1),␣
 ↪'subcategories_p1'] = proba_score_1[i]
    X_test.loc[(X_test.clean_subcategories == i) & (X_test.approved ==0),␣
 ↪'subcategories_p0']  = proba_score_0[i]


#For the difference items we are setting the probability value 0.5,␣
 ↪irrespective of value of X_test.approved
for j in diff_item:
    X_test.loc[(X_test.clean_subcategories == j), 'subcategories_p1'] = 0.5
    X_test.loc[(X_test.clean_subcategories == j), 'subcategories_p0'] = 0.5

X_train["subcategories_p1"].fillna(0.0, inplace = True)
X_test["subcategories_p1"].fillna(0.0, inplace = True)
X_train["subcategories_p0"].fillna(0.0, inplace = True)
X_test["subcategories_p0"].fillna(0.0, inplace = True)

train_subcategories_p1 = X_train['subcategories_p1'].values.reshape(-1,1)
train_subcategories_p0 = X_train['subcategories_p0'].values.reshape(-1,1)
print(train_subcategories_p1.shape)

test_subcategories_p1 = X_test['subcategories_p1'].values.reshape(-1,1)
test_subcategories_p0 = X_test['subcategories_p0'].values.reshape(-1,1)
test_subcategories_p1.shape
```

```
Difference of item present in test not in train  ['Economics Other',
'Extracurricular ParentInvolvement', 'Music Other', 'FinancialLiteracy
Health_LifeScience', 'CommunityService EarlyDevelopment', 'FinancialLiteracy
Health_Wellness', 'Gym_Fitness SocialSciences', 'CommunityService Other', 'ESL
FinancialLiteracy', 'Other PerformingArts', 'ForeignLanguages Other',
'Civics_Government ESL', 'Other Warmth Care_Hunger', 'FinancialLiteracy
ParentInvolvement', 'Literacy Warmth Care_Hunger', 'ForeignLanguages
Gym_Fitness', 'CommunityService Gym_Fitness', 'CommunityService ESL',
'NutritionEducation SocialSciences', 'Gym_Fitness Health_LifeScience',
'CommunityService FinancialLiteracy']
(33500, 1)
```

[311]:
```
(16500, 1)
```

[312]:
```
# we use probability scores to convert school_state categorical to numerical␣
 ↪values
school_state_set = X_train['school_state'].values
unique_item_train = list(set(school_state_set))

school_state_set = X_test['school_state'].values
unique_item_test = list(set(school_state_set))

#Difference of items present in test but not in train
print("Difference of item present in test not in train␣
 ↪",list(set(unique_item_test) - set(unique_item_train)))
diff_item= list(set(unique_item_test)- set(unique_item_train))

proba_score_1 = {}
proba_score_0 = {}

for i in unique_item_train:
    total_i = len(X_train[(X_train.school_state == i)])
    proba_score_1[i]=len(X_train[(X_train.school_state == i) & (X_train.
 ↪approved ==1)])/total_i
    proba_score_1[i] = truncate(proba_score_1[i],4)

    proba_score_0[i]=len(X_train[(X_train.school_state == i) & (X_train.
 ↪approved ==0)])/total_i
    proba_score_0[i] = truncate(proba_score_0[i],4)

    #we can create a new column school_state_proba and store the corresponding␣
 ↪probability scores
    X_train.loc[(X_train.school_state == i) & (X_train.approved ==1),␣
 ↪'state_p1'] = proba_score_1[i]
    X_train.loc[(X_train.school_state == i) & (X_train.approved ==0),␣
 ↪'state_p0']  = proba_score_0[i]

    #Creating a new column in X_test and corresponding to the category and␣
 ↪y_testvalue, including the probab
    X_test.loc[(X_test.school_state == i) & (X_test.approved ==1), 'state_p1']␣
 ↪= proba_score_1[i]
    X_test.loc[(X_test.school_state == i) & (X_test.approved ==0), 'state_p0'] ␣
 ↪= proba_score_0[i]

X_train["state_p1"].fillna(0.0, inplace = True)
X_test["state_p1"].fillna(0.0, inplace = True)
X_train["state_p0"].fillna(0.0, inplace = True)
X_test["state_p0"].fillna(0.0, inplace = True)

train_state_p1= X_train['state_p1'].values.reshape(-1,1)
```

```
train_state_p0= X_train['state_p0'].values.reshape(-1,1)
print(train_state_p1.shape)

test_state_p1 = X_test['state_p1'].values.reshape(-1,1)
test_state_p0 = X_test['state_p0'].values.reshape(-1,1)
test_state_p1.shape
```

```
Difference of item present in test not in train  []
(33500, 1)
```

[312]:
```
(16500, 1)
```

[313]:
```
print(X_train['teacher_prefix'].value_counts(dropna=False))
X_test["teacher_prefix"].value_counts(dropna=False)
```

```
Mrs.       17532
Ms.        12015
Mr.         3239
Teacher      710
Dr.            2
NaN            2
Name: teacher_prefix, dtype: int64
```

[313]:
```
Mrs.       8608
Ms.        5921
Mr.        1620
Teacher     351
Name: teacher_prefix, dtype: int64
```

[314]:
```
# we use probability scores to convert school_state categorical to numerical␣
 ↪values
#We have noticed nan in Teacher prefix column, we are Filling the nan column␣
 ↪with 0.0
#https://www.geeksforgeeks.org/
 ↪python-pandas-dataframe-fillna-to-replace-null-values-in-dataframe/
X_train["teacher_prefix"].fillna(0.0, inplace = True)
X_test["teacher_prefix"].fillna(0.0, inplace = True)

print(X_train['teacher_prefix'].value_counts(dropna=False))


teacher_prefix_set = X_train['teacher_prefix'].values
unique_item_train = list(set(teacher_prefix_set))

teacher_prefix_set = X_test['teacher_prefix'].values
unique_item_test = list(set(teacher_prefix_set))

#Difference of items present in test but not in train
```

```
print("Difference of item present in test not in train␣
 ↪",list(set(unique_item_test) - set(unique_item_train)))
diff_item= list(set(unique_item_test)- set(unique_item_train))

proba_score_1 = {}
proba_score_0 = {}


for i in unique_item_train:
    total_i = len(X_train[(X_train.teacher_prefix == i)])
    proba_score_1[i]=len(X_train[(X_train.teacher_prefix == i) & (X_train.
 ↪approved ==1)])/total_i
    proba_score_1[i] = truncate(proba_score_1[i],4)

    proba_score_0[i]=len(X_train[(X_train.teacher_prefix == i) & (X_train.
 ↪approved ==0)])/total_i
    proba_score_0[i] = truncate(proba_score_0[i],4)

    #we can create a new column teacher_prefix_proba and store the␣
 ↪corresponding probability scores
    X_train.loc[(X_train.teacher_prefix == i) & (X_train.approved ==1),␣
 ↪'prefix_p1'] = proba_score_1[i]
    X_train.loc[(X_train.teacher_prefix == i) & (X_train.approved ==0),␣
 ↪'prefix_p0']  = proba_score_0[i]

    #Creating a new column in X_test and corresponding to the category and␣
 ↪y_testvalue, including the probab
    X_test.loc[(X_test.teacher_prefix == i) & (X_test.approved ==1),␣
 ↪'prefix_p1'] = proba_score_1[i]
    X_test.loc[(X_test.teacher_prefix == i) & (X_test.approved ==0),␣
 ↪'prefix_p0']  = proba_score_0[i]

X_train["prefix_p1"].fillna(0.0, inplace = True)
X_test["prefix_p1"].fillna(0.0, inplace = True)
X_train["prefix_p0"].fillna(0.0, inplace = True)
X_test["prefix_p0"].fillna(0.0, inplace = True)


train_prefix_p1 = X_train['prefix_p1'].values.reshape(-1,1)
train_prefix_p0 = X_train['prefix_p0'].values.reshape(-1,1)

test_prefix_p1 = X_test['prefix_p1'].values.reshape(-1,1)
test_prefix_p0 = X_test['prefix_p0'].values.reshape(-1,1)

print(X_test['prefix_p1'].head(1))
```

```
Mrs.       17532
```

```
Ms.         12015
Mr.          3239
Teacher       710
Dr.             2
0.0             2
Name: teacher_prefix, dtype: int64
Difference of item present in test not in train  []
33629    0.8411
Name: prefix_p1, dtype: float64
```

[315]:
```
# we use probability scores to convert school_state categorical to numerical␣
 ↪values
project_grade_set = X_train['project_grade_category'].values
unique_item_train = list(set(project_grade_set))

project_grade_set = X_test['project_grade_category'].values
unique_item_test = list(set(project_grade_set))

#Difference of items present in test but not in train
print("Difference of item present in test not in train␣
 ↪",list(set(unique_item_test) - set(unique_item_train)))
diff_item= list(set(unique_item_test)- set(unique_item_train))

proba_score_1 = {}
proba_score_0 = {}

for i in unique_item_train:
    total_i = len(X_train[(X_train.project_grade_category == i)])
    proba_score_1[i]=len(X_train[(X_train.project_grade_category == i) &␣
 ↪(X_train.approved ==1)])/total_i
    proba_score_1[i] = truncate(proba_score_1[i],4)

    proba_score_0[i]=len(X_train[(X_train.project_grade_category == i) &␣
 ↪(X_train.approved ==0)])/total_i
    proba_score_0[i] = truncate(proba_score_0[i],4)

    #we can create a new column project_grade_proba and store the corresponding␣
 ↪probability scores
    X_train.loc[(X_train.project_grade_category == i) & (X_train.approved ==1),␣
 ↪'grade_p1'] = proba_score_1[i]
    X_train.loc[(X_train.project_grade_category == i) & (X_train.approved ==0),␣
 ↪'grade_p0']  = proba_score_0[i]

    #Creating a new column in X_test and corresponding to the category and␣
 ↪y_testvalue, including the probab
    X_test.loc[(X_test.project_grade_category == i) & (X_test.approved ==1),␣
 ↪'grade_p1'] = proba_score_1[i]
```

```
    X_test.loc[(X_test.project_grade_category == i) & (X_test.approved ==0),␣
 ↪'grade_p0']  = proba_score_0[i]


X_train["grade_p1"].fillna(0.0, inplace = True)
X_test["grade_p1"].fillna(0.0, inplace = True)
X_train["grade_p0"].fillna(0.0, inplace = True)
X_test["grade_p0"].fillna(0.0, inplace = True)

train_grade_p1 = X_train['grade_p1'].values.reshape(-1,1)
train_grade_p0 = X_train['grade_p0'].values.reshape(-1,1)

test_grade_p1 = X_test['grade_p1'].values.reshape(-1,1)
test_grade_p0 = X_test['grade_p0'].values.reshape(-1,1)
```

```
Difference of item present in test not in train  []
```

### 1.6.3 1.5.1.2 Vectorizing Categorical data for W2V TFIDFW2V

[316]:
```
#https://stackoverflow.com/questions/36226083/
 ↪how-to-find-which-columns-contain-any-nan-value-in-pandas-dataframe-python
X_train_mini.columns[X_train_mini.isnull().any()].tolist()

X_train_mini['approved']=y_train_mini
X_test_mini['approved']=y_test_mini
```

[317]:
```
# we use probability scores to convert categorical to numerical values
clean_cat_set = X_train_mini['clean_categories'].values
unique_item_train = list(set(clean_cat_set))

clean_cat_set = X_test_mini['clean_categories'].values
unique_item_test = list(set(clean_cat_set))

#Difference of items present in test but not in train
#https://stackoverflow.com/questions/3462143/get-difference-between-two-lists
print("Difference of item present in test not in␣
 ↪train",list(set(unique_item_test) - set(unique_item_train)))
diff_item= list(set(unique_item_test)- set(unique_item_train))

proba_score_1 = {}
proba_score_0 = {}

for i in unique_item_train:
    total_i = len(X_train_mini[(X_train_mini.clean_categories == i)])

    print("Total= ",total_i)
```

```
    proba_score_1[i]=len(X_train_mini[(X_train_mini.clean_categories == i) &␣
 ↪(X_train_mini.approved ==1)])/total_i
    proba_score_1[i] = truncate(proba_score_1[i],4)

    proba_score_0[i]=len(X_train_mini[(X_train_mini.clean_categories == i) &␣
 ↪(X_train_mini.approved ==0)])/total_i
    proba_score_0[i] = truncate(proba_score_0[i],4)

    print("1 val= ",proba_score_1[i])
    X_train_mini.loc[(X_train_mini.clean_categories == i) & (X_train_mini.
 ↪approved ==1),'categories_p1']= proba_score_1[i]
    X_test_mini.loc[(X_test_mini.clean_categories == i) & (X_test_mini.approved␣
 ↪==1),'categories_p1']= proba_score_1[i]



    print(proba_score_0[i])
    X_train_mini.loc[(X_train_mini.clean_categories == i) & (X_train_mini.
 ↪approved ==0),'categories_p0']= proba_score_0[i]
    X_test_mini.loc[(X_test_mini.clean_categories == i) & (X_test_mini.approved␣
 ↪==0),'categories_p0'] = proba_score_0[i]

#For the difference items we are setting the probability value 0.5,␣
 ↪irrespective of value of X_test.approved
for j in diff_item:
    X_test_mini.loc[(X_test_mini.clean_categories == j), 'categories_p0'] = 0.5
    X_test_mini.loc[(X_test_mini.clean_categories == j), 'categories_p0'] = 0.5

X_train_mini["categories_p0"].fillna(0.0, inplace = True)
X_test_mini["categories_p0"].fillna(0.0, inplace = True)
X_train_mini["categories_p1"].fillna(0.0, inplace = True)
X_test_mini["categories_p1"].fillna(0.0, inplace = True)

#we are creating a 1D array
train_cat_p0 = X_train_mini['categories_p0'].values.reshape(-1,1)
train_cat_p1 = X_train_mini['categories_p1'].values.reshape(-1,1)
print(train_cat_p0.shape)

test_cat_p0 = X_test_mini['categories_p0'].values.reshape(-1,1)
test_cat_p1 = X_test_mini['categories_p1'].values.reshape(-1,1)
test_cat_p0.shape
```

```
Difference of item present in test not in train []
Total=  2
1 val=  0.0
1.0
Total=  154
1 val=  0.8051
```

```
0.1948
Total=  1804
1 val=  0.8736
0.1263
Total=  237
1 val=  0.8227
0.1772
Total=  467
1 val=  0.8029
0.197
Total=  145
1 val=  0.8413
0.1586
Total=  2
1 val=  1.0
0.0
Total=  33
1 val=  0.7575
0.2424
Total=  189
1 val=  0.8253
0.1746
Total=  127
1 val=  0.7874
0.2125
Total=  167
1 val=  0.9281
0.0718
Total=  69
1 val=  0.7536
0.2463
Total=  28
1 val=  0.8571
0.1428
Total=  7
1 val=  0.7142
0.2857
Total=  3001
1 val=  0.8637
0.1362
Total=  294
1 val=  0.8877
0.1122
Total=  30
1 val=  0.8333
0.1666
Total=  94
1 val=  0.851
```

```
0.1489
Total=  40
1 val=  0.825
0.175
Total=  8
1 val=  0.875
0.125
Total=  269
1 val=  0.8513
0.1486
Total=  75
1 val=  0.7866
0.2133
Total=  172
1 val=  0.9244
0.0755
Total=  3
1 val=  0.6666
0.3333
Total=  506
1 val=  0.8616
0.1383
Total=  3
1 val=  0.6666
0.3333
Total=  7
1 val=  0.7142
0.2857
Total=  4
1 val=  1.0
0.0
Total=  51
1 val=  0.8627
0.1372
Total=  1
1 val=  1.0
0.0
Total=  495
1 val=  0.808
0.1919
Total=  1
1 val=  1.0
0.0
Total=  2003
1 val=  0.8212
0.1787
Total=  2
1 val=  0.5
```

```
0.5
Total=  25
1 val=  0.76
0.24
Total=  2
1 val=  1.0
0.0
Total=  31
1 val=  0.8064
0.1935
Total=  20
1 val=  0.75
0.25
Total=  101
1 val=  0.792
0.2079
Total=  7
1 val=  1.0
0.0
Total=  68
1 val=  0.8676
0.1323
Total=  639
1 val=  0.8669
0.133
Total=  239
1 val=  0.8577
0.1422
Total=  17
1 val=  0.9411
0.0588
Total=  93
1 val=  0.8387
0.1612
Total=  1254
1 val=  0.8444
0.1555
Total=  187
1 val=  0.8074
0.1925
Total=  201
1 val=  0.8457
0.1542
Total=  1
1 val=  0.0
1.0
Total=  25
1 val=  0.8
```

```
0.2
(13400, 1)
```

[317]:
```
(6600, 1)
```

[318]:
```
# we use probability scores to convert clean_subcategories to numerical values
clean_sub_cat_set = X_train_mini['clean_subcategories'].values
unique_item_train = list(set(clean_sub_cat_set))

clean_sub_cat_set = X_test_mini['clean_subcategories'].values
unique_item_test = list(set(clean_sub_cat_set))

#Difference of items present in test but not in train
print("Difference of item present in test not in train␣
 ↪",list(set(unique_item_test) - set(unique_item_train)))
diff_item= list(set(unique_item_test)- set(unique_item_train))

proba_score_1 = {}
proba_score_0 = {}

for i in unique_item_train:
    total_i = len(X_train_mini[(X_train_mini.clean_subcategories == i)])
    proba_score_1[i]=len(X_train_mini[(X_train_mini.clean_subcategories == i) &␣
 ↪(X_train_mini.approved ==1)])/total_i
    proba_score_1[i] = truncate(proba_score_1[i],4)

    proba_score_0[i]=len(X_train_mini[(X_train.clean_subcategories == i) &␣
 ↪(X_train_mini.approved ==0)])/total_i
    proba_score_0[i] = truncate(proba_score_0[i],4)

    #we can create a new column subcategories_proba and store the corresponding␣
 ↪probability scores
    X_train_mini.loc[(X_train_mini.clean_subcategories == i) & (X_train_mini.
 ↪approved ==1), 'subcategories_p1'] = proba_score_1[i]
    X_train_mini.loc[(X_train_mini.clean_subcategories == i) & (X_train_mini.
 ↪approved ==0), 'subcategories_p0']  = proba_score_0[i]

    #Creating a new column in X_test and corresponding to the category and␣
 ↪y_testvalue, including the probab.
    X_test_mini.loc[(X_test_mini.clean_subcategories == i) & (X_test_mini.
 ↪approved ==1), 'subcategories_p1'] = proba_score_1[i]
    X_test_mini.loc[(X_test_mini.clean_subcategories == i) & (X_test_mini.
 ↪approved ==0), 'subcategories_p0']  = proba_score_0[i]


#For the difference items we are setting the probability value 0.5,␣
 ↪irrespective of value of X_test.approved
```

```
for j in diff_item:
    X_test_mini.loc[(X_test_mini.clean_subcategories == j), 'subcategories_p1']␣
 ↪= 0.5
    X_test_mini.loc[(X_test_mini.clean_subcategories == j), 'subcategories_p0']␣
 ↪= 0.5


X_train_mini["subcategories_p1"].fillna(0.0, inplace = True)
X_test_mini["subcategories_p1"].fillna(0.0, inplace = True)
X_train_mini["subcategories_p0"].fillna(0.0, inplace = True)
X_test_mini["subcategories_p0"].fillna(0.0, inplace = True)


train_subcat_p1 = X_train_mini['subcategories_p1'].values.reshape(-1,1)
train_subcat_p0 = X_train_mini['subcategories_p0'].values.reshape(-1,1)
print(train_subcat_p1.shape)

test_subcat_p1 = X_test_mini['subcategories_p1'].values.reshape(-1,1)
test_subcat_p0 = X_test_mini['subcategories_p0'].values.reshape(-1,1)
test_subcat_p1.shape
```

```
Difference of item present in test not in train  ['Economics Other', 'ESL
Other', 'College_CareerPrep ESL', 'Civics_Government Extracurricular',
'EnvironmentalScience Other', 'ForeignLanguages Health_Wellness',
'ForeignLanguages VisualArts', 'Extracurricular SpecialNeeds', 'Gym_Fitness
Literature_Writing', 'Civics_Government College_CareerPrep', 'Gym_Fitness
History_Geography', 'Health_LifeScience Warmth Care_Hunger',
'EnvironmentalScience Music', 'Civics_Government VisualArts',
'CharacterEducation Civics_Government', 'EnvironmentalScience ForeignLanguages',
'ESL SocialSciences', 'ParentInvolvement SocialSciences', 'CharacterEducation
ForeignLanguages', 'Extracurricular Health_Wellness', 'Gym_Fitness
Health_LifeScience', 'CommunityService NutritionEducation',
'EnvironmentalScience ParentInvolvement', 'College_CareerPrep Music']
(13400, 1)
```

[318]:
```
(6600, 1)
```

[319]:
```
# we use probability scores to convert school_state categorical to numerical␣
 ↪values
school_state_set = X_train_mini['school_state'].values
unique_item_train = list(set(school_state_set))

school_state_set = X_test_mini['school_state'].values
unique_item_test = list(set(school_state_set))

#Difference of items present in test but not in train
print("Difference of item present in test not in train␣
 ↪",list(set(unique_item_test) - set(unique_item_train)))
```

```
diff_item= list(set(unique_item_test)- set(unique_item_train))

proba_score_1 = {}
proba_score_0 = {}

for i in unique_item_train:
    total_i = len(X_train_mini[(X_train_mini.school_state == i)])
    proba_score_1[i]=len(X_train_mini[(X_train_mini.school_state == i) &␣
 ↪(X_train_mini.approved ==1)])/total_i
    proba_score_1[i] = truncate(proba_score_1[i],4)

    proba_score_0[i]=len(X_train_mini[(X_train_mini.school_state == i) &␣
 ↪(X_train_mini.approved ==0)])/total_i
    proba_score_0[i] = truncate(proba_score_0[i],4)

    #we can create a new column school_state_proba and store the corresponding␣
 ↪probability scores
    X_train_mini.loc[(X_train_mini.school_state == i) & (X_train_mini.approved␣
 ↪==1), 'state_p1'] = proba_score_1[i]
    X_train_mini.loc[(X_train_mini.school_state == i) & (X_train_mini.approved␣
 ↪==0), 'state_p0']  = proba_score_0[i]

    #Creating a new column in X_test and corresponding to the category and␣
 ↪y_testvalue, including the probab
    X_test_mini.loc[(X_test_mini.school_state == i) & (X_test_mini.approved␣
 ↪==1), 'state_p1'] = proba_score_1[i]
    X_test_mini.loc[(X_test_mini.school_state == i) & (X_test_mini.approved␣
 ↪==0), 'state_p0']  = proba_score_0[i]

X_train_mini["state_p1"].fillna(0.0, inplace = True)
X_test_mini["state_p1"].fillna(0.0, inplace = True)
X_train_mini["state_p0"].fillna(0.0, inplace = True)
X_test_mini["state_p0"].fillna(0.0, inplace = True)

train_state_mini_p1= X_train_mini['state_p1'].values.reshape(-1,1)
train_state_mini_p0= X_train_mini['state_p0'].values.reshape(-1,1)
print(train_state_mini_p0.shape)

test_state_mini_p1 = X_test_mini['state_p1'].values.reshape(-1,1)
test_state_mini_p0 = X_test_mini['state_p0'].values.reshape(-1,1)
test_state_mini_p0.shape
```

```
Difference of item present in test not in train  []
(13400, 1)
```

[319]: (6600, 1)

[320]:
```
# we use probability scores to convert school_state categorical to numerical␣
 ↪values
#We have noticed nan in Teacher prefix column, we are Filling the nan column␣
 ↪with 0.0
#https://www.geeksforgeeks.org/
 ↪python-pandas-dataframe-fillna-to-replace-null-values-in-dataframe/
X_train_mini["teacher_prefix"].fillna(0.0, inplace = True)
X_test_mini["teacher_prefix"].fillna(0.0, inplace = True)

print(X_train_mini['teacher_prefix'].value_counts(dropna=False))


teacher_prefix_set = X_train_mini['teacher_prefix'].values
unique_item_train = list(set(teacher_prefix_set))

teacher_prefix_set = X_test_mini['teacher_prefix'].values
unique_item_test = list(set(teacher_prefix_set))

#Difference of items present in test but not in train
print("Difference of item present in test not in train␣
 ↪",list(set(unique_item_test) - set(unique_item_train)))
diff_item= list(set(unique_item_test)- set(unique_item_train))

proba_score_1 = {}
proba_score_0 = {}


for i in unique_item_train:
    total_i = len(X_train_mini[(X_train_mini.teacher_prefix == i)])
    proba_score_1[i]=len(X_train_mini[(X_train_mini.teacher_prefix == i) &␣
 ↪(X_train_mini.approved ==1)])/total_i
    proba_score_1[i] = truncate(proba_score_1[i],4)

    proba_score_0[i]=len(X_train_mini[(X_train_mini.teacher_prefix == i) &␣
 ↪(X_train_mini.approved ==0)])/total_i
    proba_score_0[i] = truncate(proba_score_0[i],4)

    #we can create a new column teacher_prefix_proba and store the␣
 ↪corresponding probability scores
    X_train_mini.loc[(X_train_mini.teacher_prefix == i) & (X_train_mini.
 ↪approved ==1), 'prefix_p1'] = proba_score_1[i]
    X_train_mini.loc[(X_train_mini.teacher_prefix == i) & (X_train_mini.
 ↪approved ==0), 'prefix_p0']  = proba_score_0[i]

    #Creating a new column in X_test and corresponding to the category and␣
 ↪y_testvalue, including the probab
```

```
    X_test_mini.loc[(X_test_mini.teacher_prefix == i) & (X_test_mini.approved␣
 ↪==1), 'prefix_p1'] = proba_score_1[i]
    X_test_mini.loc[(X_test_mini.teacher_prefix == i) & (X_test_mini.approved␣
 ↪==0), 'prefix_p0']  = proba_score_0[i]


X_train_mini["prefix_p0"].fillna(0.0, inplace = True)
X_test_mini["prefix_p0"].fillna(0.0, inplace = True)
X_train_mini["prefix_p1"].fillna(0.0, inplace = True)
X_test_mini["prefix_p1"].fillna(0.0, inplace = True)


train_prefix_mini_p1 = X_train_mini['prefix_p1'].values.reshape(-1,1)
train_prefix_mini_p0 = X_train_mini['prefix_p0'].values.reshape(-1,1)
print(train_prefix_mini_p0.shape)

test_prefix_mini_p1 = X_test_mini['prefix_p1'].values.reshape(-1,1)
test_prefix_mini_p0 = X_test_mini['prefix_p0'].values.reshape(-1,1)

print(test_prefix_mini_p0.shape)
```

```
Mrs.       6984
Ms.        4838
Mr.        1262
Teacher     316
Name: teacher_prefix, dtype: int64
Difference of item present in test not in train  [0.0]
(13400, 1)
(6600, 1)
```

[321]:
```
# we use probability scores to convert school_state categorical to numerical␣
 ↪values
project_grade_set = X_train_mini['project_grade_category'].values
unique_item_train = list(set(project_grade_set))

project_grade_set = X_test_mini['project_grade_category'].values
unique_item_test = list(set(project_grade_set))

#Difference of items present in test but not in train
print("Difference of item present in test not in train␣
 ↪",list(set(unique_item_test) - set(unique_item_train)))
diff_item= list(set(unique_item_test)- set(unique_item_train))

proba_score_1 = {}
proba_score_0 = {}

for i in unique_item_train:
```

```
    total_i = len(X_train_mini[(X_train_mini.project_grade_category == i)])
    proba_score_1[i]=len(X_train_mini[(X_train_mini.project_grade_category ==␣
↪i) & (X_train_mini.approved ==1)])/total_i
    proba_score_1[i] = truncate(proba_score_1[i],4)

    proba_score_0[i]=len(X_train_mini[(X_train_mini.project_grade_category ==␣
↪i) & (X_train_mini.approved ==0)])/total_i
    proba_score_0[i] = truncate(proba_score_0[i],4)

    #we can create a new column project_grade_proba and store the corresponding␣
↪probability scores
    X_train_mini.loc[(X_train_mini.project_grade_category == i) & (X_train_mini.
↪approved ==1), 'grade_p1'] = proba_score_1[i]
    X_train_mini.loc[(X_train_mini.project_grade_category == i) & (X_train_mini.
↪approved ==0), 'grade_p0']  = proba_score_0[i]

    #Creating a new column in X_test and corresponding to the category and␣
↪y_testvalue, including the probab
    X_test_mini.loc[(X_test_mini.project_grade_category == i) & (X_test_mini.
↪approved ==1), 'grade_p1'] = proba_score_1[i]
    X_test_mini.loc[(X_test_mini.project_grade_category == i) & (X_test_mini.
↪approved ==0), 'grade_p0']  = proba_score_0[i]

X_train_mini["grade_p1"].fillna(0.0, inplace = True)
X_test_mini["grade_p1"].fillna(0.0, inplace = True)
X_train_mini["grade_p0"].fillna(0.0, inplace = True)
X_test_mini["grade_p0"].fillna(0.0, inplace = True)

train_grade_mini_p1 = X_train_mini['grade_p1'].values.reshape(-1,1)
train_grade_mini_p0 = X_train_mini['grade_p0'].values.reshape(-1,1)
print(train_grade_mini_p0.shape)

test_grade_mini_p1 = X_test_mini['grade_p1'].values.reshape(-1,1)
test_grade_mini_p0 = X_test_mini['grade_p0'].values.reshape(-1,1)
print(test_grade_mini_p0.shape)
```

```
Difference of item present in test not in train  []
(13400, 1)
(6600, 1)
```

[322]:
```
X_train_mini.columns[X_train_mini.isnull().any()].tolist()
```

[322]:
```
['project_essay_3', 'project_essay_4']
```

### 1.6.4 1.5.2 Vectorizing Text data

#### 1.5.2.1 Bag of words

[323]:
```
# We are considering only the words which appeared in at least 10␣
 ↪documents(rows or projects).
#BoW
preprocessed_essays =X_train['essay'].astype('U')
count_vect_essay =␣
 ↪CountVectorizer(ngram_range=(1,2),min_df=10,max_features=5000) #in␣
 ↪scikit-learn
count_vect_essay.fit(preprocessed_essays)
print("some feature names ", count_vect_essay.get_feature_names()[:10])
print('='*50)

x_essay_bow1 = count_vect_essay.transform(preprocessed_essays.astype('U'))
#cv_essay_bow1 = count_vect.transform(X_cv['essay'].astype('U'))
essay_bow1 = count_vect_essay.transform(X_test['essay'].astype('U'))
print("the type of count vectorizer ",type(essay_bow1))
print("the shape of out text BOW vectorizer ",essay_bow1.get_shape())
print("the number of unique words ", essay_bow1.get_shape()[1])


#Normalization of BOW
from sklearn.preprocessing import Normalizer
w_normalized = Normalizer()

w_normalized.fit(x_essay_bow1)

x_essay_bow = w_normalized.transform(x_essay_bow1)
#cv_essay_bow = w_normalized.transform(cv_essay_bow1)
essay_bow = w_normalized.transform(essay_bow1)
```

```
some feature names  ['abilities', 'ability', 'ability learn', 'ability levels',
'able', 'able access', 'able choose', 'able complete', 'able control', 'able
create']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (16500, 5000)
the number of unique words  5000
```

[324]:
```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
preprocessed_title =X_train['project_title'].astype('U')
count_vect_title =␣
 ↪CountVectorizer(ngram_range=(1,2),min_df=10,max_features=5000) #in␣
 ↪scikit-learn
count_vect_title.fit(preprocessed_title)
print("some feature names ", count_vect_title.get_feature_names()[:10])
print('='*50)
```

```
x_title_bow1 = count_vect_title.transform(preprocessed_title)
#cv_title_bow1 = count_vect.transform(X_cv['project_title'].astype('U'))
title_bow1 = count_vect_title.transform(X_test['project_title'].astype('U'))
print("the type of count vectorizer ",type(title_bow1))
print("the shape of out title BOW vectorizer ",title_bow1.get_shape())
print("the number of unique words ", title_bow1.get_shape()[1])

#Normalization of BOW title
from sklearn.preprocessing import Normalizer
w_normalized = Normalizer()

w_normalized.fit(x_title_bow1)

x_title_bow = w_normalized.transform(x_title_bow1)
#cv_title_bow = w_normalized.transform(cv_title_bow1)
title_bow = w_normalized.transform(title_bow1)
```

```
some feature names  ['abc', 'about', 'about it', 'about our', 'about reading',
'about science', 'academic', 'academic success', 'academics', 'access']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out title BOW vectorizer  (16500, 2725)
the number of unique words  2725
```

#### 1.5.2.2 TFIDF vectorizer

[325]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
preprocessed_essays =X_train['essay']
tfidf_essay = TfidfVectorizer(ngram_range=(1,2),min_df=10,max_features=5000)
tfidf_essay.fit_transform(preprocessed_essays.astype('U'))
print("Shape of matrix after one hot encodig ",tfidf_essay.get_feature_names()[:
 ↪10])
print('='*50)

x_essay_tfidf1 = tfidf_essay.transform(preprocessed_essays.astype('U'))
#cv_essay_tfidf1= vectorizer.transform(X_cv['essay'].astype('U'))
essay_tfidf1 = tfidf_essay.transform(X_test['essay'].astype('U'))
print("the type of count vectorizer ",type(essay_tfidf1))
print("the shape of out text tfidf vectorizer ",essay_tfidf1.get_shape())
print("the number of unique words ", essay_tfidf1.get_shape()[1])


#Normalization of TfIdf
from sklearn.preprocessing import Normalizer
w_normalized = Normalizer()

w_normalized.fit(x_essay_tfidf1)
```

```
x_essay_tfidf = w_normalized.transform(x_essay_tfidf1)
#cv_essay_tfidf = w_normalized.transform(cv_essay_tfidf1)
essay_tfidf = w_normalized.transform(essay_tfidf1)
```

```
Shape of matrix after one hot encodig  ['abilities', 'ability', 'ability learn',
'ability levels', 'able', 'able access', 'able choose', 'able complete', 'able
control', 'able create']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text tfidf vectorizer  (16500, 5000)
the number of unique words  5000
```

[326]:
```
x_essay_tfidf.shape
```

[326]:
```
(33500, 5000)
```

[327]:
```
# Similarly you can vectorize for title also
from sklearn.feature_extraction.text import TfidfVectorizer
preprocessed_title =X_train['project_title'].astype('U')
tfidf_title = TfidfVectorizer(ngram_range=(1,2),min_df=10,max_features=5000)
tfidf_title.fit_transform(preprocessed_title)
print("some feature names ", tfidf_title.get_feature_names()[:10])
print('='*50)

x_title_tfidf1 = tfidf_title.transform(preprocessed_title)
#cv_title_tfidf1 = vectorizer.transform(X_cv['project_title'].astype('U'))
title_tfidf1 = tfidf_title.transform(X_test['project_title'].astype('U'))
print("the type of count vectorizer ",type(title_tfidf1))
print("the shape of out title tfidf vectorizer ",title_tfidf1.get_shape())
print("the number of unique words ", title_tfidf1.get_shape()[1])


#Normalization of TfIdf title
from sklearn.preprocessing import Normalizer
w_normalized = Normalizer()

w_normalized.fit(x_title_tfidf1)

x_title_tfidf = w_normalized.transform(x_title_tfidf1)
#cv_title_tfidf = w_normalized.transform(cv_title_tfidf1)
title_tfidf = w_normalized.transform(title_tfidf1)
```

```
some feature names  ['abc', 'about', 'about it', 'about our', 'about reading',
'about science', 'academic', 'academic success', 'academics', 'access']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out title tfidf vectorizer  (16500, 2725)
the number of unique words  2725
```

[328]:
```
x_title_tfidf.shape
```

[328]:
```
(33500, 2725)
```

[ ]:
```

```

#### 1.5.2.3 Using Pretrained Models: Avg W2V

[329]:
```
'''
# 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-save-and-load-variables-in-python/

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


'''
```

[329]:
```
'\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        model[word] = embedding\n    print
("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n#
============================\nOutput:\n    \nLoading Glove 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 that are present in both glove vectors and our coupus",       le
n(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_co
urpus = {}\nwords_glove = set(model.keys())\nfor i in words:\n    if i in
words_glove:\n        words_courpus[i] = model[i]\nprint("word 2 vec length",
len(words_courpus))\n\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 pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n
pickle.dump(words_courpus, f)\n\n\n'
```

[330]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/
 ↪how-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

[331]:
```
# average Word2Vec
# compute average word2vec for each review.
def computeAvgW2V(list_of_sentance):
    avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in␣
 ↪this list
    for sentence in tqdm(list_of_sentance): # 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]))
    return avg_w2v_vectors
```

[332]:
```
from scipy.sparse import csr_matrix

preprocessed_essays =X_train_mini['essay'].astype('U')
list_of_sentance=[]
for sentance in preprocessed_essays:
    list_of_sentance.append(sentance)

test_list_of_sentance=[]
for sentance in X_test_mini['essay'].astype('U'):
    test_list_of_sentance.append(sentance)

x_essay_avg1  = computeAvgW2V(list_of_sentance)
essay_avg1 = computeAvgW2V(test_list_of_sentance)

'''
cv_list_of_sentance=[]
for sentance in X_cv['essay'].astype('U'):
    cv_list_of_sentance.append(sentance)
cv_essay_avg1 = computeAvgW2V(cv_list_of_sentance)
'''

#Normalization of AvgW2V
from sklearn.preprocessing import Normalizer
w_normalized = Normalizer()

w_normalized.fit(x_essay_avg1)

x_essay_avg = w_normalized.transform(x_essay_avg1)
```

```
#cv_essay_avg = w_normalized.transform(cv_essay_avg1)
essay_avg = w_normalized.transform(essay_avg1)


#Converting the ndarray to csr_matrix
x_essay_avg = csr_matrix(x_essay_avg)
essay_avg = csr_matrix(essay_avg)
```

```
100%|| 13400/13400 [00:07<00:00, 1838.34it/s]
```

```
13400
300
```

```
100%|| 6600/6600 [00:03<00:00, 1936.95it/s]
```

```
6600
300
```

[333]:
```
preprocessed_title =X_train_mini['project_title']
list_of_title=[]
for sentance in preprocessed_title:
    list_of_title.append(sentance)

test_list_of_title=[]
for sentance in X_test_mini['project_title']:
    test_list_of_title.append(sentance)

x_title_avg1  = computeAvgW2V(list_of_title)
title_avg1 = computeAvgW2V(test_list_of_title)

'''
cv_list_of_title=[]
for sentance in X_cv['project_title']:
    cv_list_of_title.append(sentance)
cv_title_avg1 = computeAvgW2V(cv_list_of_sentance)

'''
#Normalization of AvgW2V title
from sklearn.preprocessing import Normalizer
w_normalized = Normalizer()

w_normalized.fit(x_title_avg1)

x_title_avg = w_normalized.transform(x_title_avg1)
#cv_title_avg = w_normalized.transform(cv_title_avg1)
title_avg = w_normalized.transform(title_avg1)

#Converting the ndarray to csr_matrix
```

```
x_title_avg= csr_matrix(x_title_avg)
title_avg= csr_matrix(title_avg)
```

```
100%|| 13400/13400 [00:00<00:00, 39039.44it/s]

13400
300

100%|| 6600/6600 [00:00<00:00, 38569.12it/s]

6600
300
```

#### 1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

[334]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
preprocessed_essays =X_train_mini['essay'].astype('U')
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())
```

[335]:
```
# average Word2Vec
# compute average word2vec for each review.
def computeTfIdf(list_of_sentance):
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in␣
 ↪this list
    row=0;
    for sentence in tqdm(list_of_sentance): # 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 sent: # 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]*(sent.count(word)/len(sent)) #␣
 ↪getting the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)
        row +=1;
    return tfidf_w2v_vectors
```

[336]:
```
preprocessed_essays =X_train_mini['essay'].astype('U')
list_of_sentance=[]
for sentance in preprocessed_essays:
    list_of_sentance.append(sentance.split())

test_list_of_sentance=[]
for sentance in X_test_mini['essay'].astype('U'):
    test_list_of_sentance.append(sentance.split())

x_essay_tfidf_avg1  = computeTfIdf(list_of_sentance)
essay_tfidf_avg1= computeTfIdf(test_list_of_sentance)

'''
cv_list_of_sentance=[]
for sentance in X_cv['essay'].astype('U'):
    cv_list_of_sentance.append(sentance.split())
cv_essay_tfidf_avg1 = computeTfIdf(cv_list_of_sentance)
'''
#Normalization of TfIdfW2V
from sklearn.preprocessing import Normalizer
w_normalized = Normalizer()

w_normalized.fit(x_essay_tfidf_avg1)

x_essay_tfidf_avg = w_normalized.transform(x_essay_tfidf_avg1)
#cv_essay_tfidf_avg = w_normalized.transform(cv_essay_tfidf_avg1)
essay_tfidf_avg = w_normalized.transform(essay_tfidf_avg1)

#Converting the ndarray to csr_matrix
x_essay_tfidf_avg = csr_matrix(x_essay_tfidf_avg)
essay_tfidf_avg = csr_matrix(essay_tfidf_avg)
```

```
100%|| 13400/13400 [00:00<00:00, 106273.25it/s]
100%|| 6600/6600 [00:00<00:00, 98437.88it/s]
```

TFIDF weighted W2V title

[337]:
```
# Similarly you can vectorize for title also
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
preprocessed_title =X_train_mini['project_title'].astype('U')
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_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())
```

[338]:
```
# average Word2Vec
# compute average word2vec for each review.
def computeTfIdf(list_of_sentance):
```

```
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in␣
↪this list
    row=0;
    for sentence in tqdm(list_of_sentance): # 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 sent: # 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]*(sent.count(word)/len(sent)) #␣
↪getting the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)
        row +=1;
    return tfidf_w2v_vectors
```

[339]:
```
preprocessed_title =X_train_mini['project_title'].astype('U')
list_of_title=[]
for sentance in preprocessed_title:
    list_of_title.append(sentance)

test_list_of_title=[]
for sentance in X_test_mini['project_title'].astype('U'):
    test_list_of_title.append(sentance)

x_title_tfidf_avg1  = computeTfIdf(list_of_title)
title_tfidf_avg1 = computeTfIdf(test_list_of_title)

'''
cv_list_of_title=[]
for sentance in X_cv['project_title'].astype('U'):
    cv_list_of_title.append(sentance)
cv_title_tfidf_avg1 = computeTfIdf(cv_list_of_sentance)
'''

#Normalization of AvgW2V
from sklearn.preprocessing import Normalizer
w_normalized = Normalizer()

w_normalized.fit(x_title_tfidf_avg1)
```

```
x_title_tfidf_avg = w_normalized.transform(x_title_tfidf_avg1)
#cv_title_tfidf_avg = w_normalized.transform(cv_title_tfidf_avg1)
title_tfidf_avg = w_normalized.transform(title_tfidf_avg1)


#Converting the ndarray to csr_matrix
x_title_tfidf_avg = csr_matrix(x_title_tfidf_avg)
title_tfidf_avg = csr_matrix(title_tfidf_avg)
```

```
100%|| 13400/13400 [00:00<00:00, 110667.00it/s]
100%|| 6600/6600 [00:00<00:00, 117773.42it/s]
```

### 1.6.5 1.5.3 Vectorizing Numerical features

### 1.6.6 1.5.3.1 Vectorizing Numerical features for BOW and TFIDF

[340]:
```
#1) For normalizing numerical data, we have to use reshape(1,-1) instead of␣
 ↪(-1,1).
#2) Normalizer by default normalizes on each sample(row).StandardScaler␣
 ↪standardises on each feature(column).
#3) If we use (-1,1) it means any number of rows and one column. So that makes␣
 ↪normalizer on each row containing one column.This makes the value 1.
#4) If we use (-1, 1) then  all your prices are 1. It wont be useful at all.
#5) Note: If the shape mismatch is the problem for not using (1,-1) you can␣
 ↪reshape into (-1,1) again after normalization is done.
#6) see the below example given in the comments
'''
Ex:
After (-1,1) array is [[1],[2],[3]]
Using normalizer results in
[[1/1],[2/2],[3/3]] = [[1],[1],[1]]
If you use (1,-1) array is [1,2,3]
result is
[1/sqrt(14) , 2/sqrt(14) , 3/sqrt(14) ] = [0.26,0.52, 0.78]

'''

from sklearn.preprocessing import Normalizer
price_norm = Normalizer()

price_norm.fit(X_train['price'].values.reshape(1,-1))

train_price_norm = price_norm.transform(X_train['price'].values.reshape(1,-1)).T
test_price_norm = price_norm.transform(X_test['price'].values.reshape(1,-1)).T

print(train_price_norm.shape)
print(train_price_norm)
```

```
(33500, 1)
[[0.00160427]
 [0.00741363]
 [0.00242763]
 ...
 [0.00105653]
 [0.00303943]
 [0.00593885]]
```

[341]:
```
from sklearn.preprocessing import Normalizer
quantity_norm = Normalizer()

quantity_norm.fit(X_train['quantity'].values.reshape(1,-1))

train_quantity_norm = quantity_norm.transform(X_train['quantity'].values.
 ↪reshape(1,-1)).T
test_quantity_norm = quantity_norm.transform(X_test['quantity'].values.
 ↪reshape(1,-1)).T

print(train_quantity_norm.shape)
print(train_quantity_norm)
```

```
(33500, 1)
[[0.00068012]
 [0.00068012]
 [0.00374068]
 ...
 [0.00578105]
 [0.00357065]
 [0.00068012]]
```

[342]:
```
from sklearn.preprocessing import Normalizer
prev_proj_norm = Normalizer()

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

train_prev_proj_norm = prev_proj_norm.
 ↪transform(X_train['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(1,-1)).T
test_prev_proj_norm = prev_proj_norm.
 ↪transform(X_test['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(1,-1)).T


print(train_prev_proj_norm.shape)
print(train_prev_proj_norm)
```

```
(33500, 1)
[[0.00017947]
 [0.00107683]
 [0.00035894]
 ...
 [0.00143578]
 [0.00251261]
 [0.        ]]
```

[343]:
```
from sklearn.preprocessing import Normalizer
senti_norm = Normalizer()

senti_norm.fit(X_train['essay_sentiment_score'].values.reshape(1,-1))

train_senti_norm = senti_norm.transform(X_train['essay_sentiment_score'].values.
 ↪reshape(1,-1)).T
test_senti_norm = senti_norm.transform(X_test['essay_sentiment_score'].values.
 ↪reshape(1,-1)).T


print(train_senti_norm.shape)
print(train_senti_norm)
```

```
(33500, 1)
[[0.00462552]
 [0.0042233 ]
 [0.00884883]
 ...
 [0.0062344 ]
 [0.00321776]
 [0.00341886]]
```

[344]:
```
from sklearn.preprocessing import Normalizer
tile_wordcount_norm = Normalizer()

tile_wordcount_norm.fit(X_train['title_wordcount'].values.reshape(1,-1))

train_title_wordcount_norm = tile_wordcount_norm.
 ↪transform(X_train['title_wordcount'].values.reshape(1,-1)).T
test_title_wordcount_norm = tile_wordcount_norm.
 ↪transform(X_test['title_wordcount'].values.reshape(1,-1)).T


print(train_title_wordcount_norm.shape)
print(train_title_wordcount_norm)
```

```
(33500, 1)
[[0.0034732 ]
```

```
 [0.00463094]
 [0.00810414]
 ...
 [0.00578867]
 [0.0069464 ]
 [0.00231547]]
```

[345]:
```
from sklearn.preprocessing import Normalizer
essay_wordcount_norm = Normalizer()

essay_wordcount_norm.fit(X_train['essay_wordcount'].values.reshape(1,-1))

train_essay_wordcount_norm = essay_wordcount_norm.
 ↪transform(X_train['essay_wordcount'].values.reshape(1,-1)).T
test_essay_wordcount_norm = essay_wordcount_norm.
 ↪transform(X_test['essay_wordcount'].values.reshape(1,-1)).T


print(train_essay_wordcount_norm.shape)
print(train_essay_wordcount_norm)
```

```
(33500, 1)
[[0.00470828]
 [0.00463853]
 [0.00449902]
 ...
 [0.00568481]
 [0.00523142]
 [0.00571969]]
```

### 1.6.7 1.5.3.2 Vectorizing Numerical features for W2V

[346]:
```
#1) For normalizing numerical data, we have to use reshape(1,-1) instead of␣
 ↪(-1,1).
#2) Normalizer by default normalizes on each sample(row).StandardScaler␣
 ↪standardises on each feature(column).
#3) If we use (-1,1) it means any number of rows and one column. So that makes␣
 ↪normalizer on each row containing one column.This makes the value 1.
#4) If we use (-1, 1) then  all your prices are 1. It wont be useful at all.
#5) Note: If the shape mismatch is the problem for not using (1,-1) you can␣
 ↪reshape into (-1,1) again after normalization is done.
#6) see the below example given in the comments
'''
Ex:
After (-1,1) array is [[1],[2],[3]]
Using normalizer results in
[[1/1],[2/2],[3/3]] = [[1],[1],[1]]
```

```
If you use (1,-1) array is [1,2,3]
result is
[1/sqrt(14) , 2/sqrt(14) , 3/sqrt(14) ] = [0.26,0.52, 0.78]

'''

from sklearn.preprocessing import Normalizer
price_norm = Normalizer()

price_norm.fit(X_train_mini['price'].values.reshape(1,-1))

train_price = price_norm.transform(X_train_mini['price'].values.reshape(1,-1)).T
test_price = price_norm.transform(X_test_mini['price'].values.reshape(1,-1)).T

print(train_price.shape)
print(train_price)
```

```
(13400, 1)
[[0.01469983]
 [0.03520792]
 [0.00361954]
 ...
 [0.00529852]
 [0.00448606]
 [0.00489282]]
```

[347]:
```
from sklearn.preprocessing import Normalizer
quantity_norm = Normalizer()

quantity_norm.fit(X_train_mini['quantity'].values.reshape(1,-1))

train_quantity = quantity_norm.transform(X_train_mini['quantity'].values.
 ↪reshape(1,-1)).T
test_quantity = quantity_norm.transform(X_test_mini['quantity'].values.
 ↪reshape(1,-1)).T

print(train_quantity.shape)
print(train_quantity)
```

```
(13400, 1)
[[0.00283657]
 [0.0065241 ]
 [0.00085097]
 ...
 [0.00028366]
 [0.0019856 ]
 [0.00028366]]
```

[348]:
```
from sklearn.preprocessing import Normalizer
prev_proj_norm = Normalizer()

prev_proj_norm.fit(X_train_mini['teacher_number_of_previously_posted_projects'].
 ↪values.reshape(1,-1))

train_prev_proj = prev_proj_norm.
 ↪transform(X_train_mini['teacher_number_of_previously_posted_projects'].
 ↪values.reshape(1,-1)).T
test_prev_proj = prev_proj_norm.
 ↪transform(X_test_mini['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(1,-1)).T


print(train_prev_proj.shape)
print(train_prev_proj)
```

```
(13400, 1)
[[0.00028505]
 [0.00399076]
 [0.00798153]
 ...
 [0.        ]
 [0.00028505]
 [0.00199538]]
```

### 1.6.8 1.5.4 Merging all the above features

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

[351]:
```
#set 1
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a␣
 ↪dense matirx :)
train_bow =␣
 ↪hstack((train_categories_p1,train_categories_p0,train_subcategories_p1,train_subcategories_
                    ␣
 ↪train_state_p1,train_state_p0,train_prefix_p1,train_prefix_p0,
                     train_grade_p1,train_grade_p0,
train_price_norm,train_quantity_norm,train_prev_proj_norm,␣
 ↪x_essay_bow,x_title_bow))
print(train_bow.shape)

test_bow = hstack((test_categories_p1,test_categories_p0,␣
 ↪test_subcategories_p1,test_subcategories_p0,
                   test_state_p1,test_state_p0,test_prefix_p1,test_prefix_p0,
                    test_grade_p1,test_grade_p0,
```

```
                    ␣
 ↪test_price_norm,test_quantity_norm,test_prev_proj_norm,essay_bow,title_bow))
print(test_bow.shape)
```

```
(33500, 7738)
(16500, 7738)
```

[352]:
```
#set 2
train_tfidf =␣
 ↪hstack((train_categories_p1,train_categories_p0,train_subcategories_p1,train_subcategories_
                    ␣
 ↪train_state_p1,train_state_p0,train_prefix_p1,train_prefix_p0,
                     train_grade_p1,train_grade_p0,
                    ␣
 ↪train_price_norm,train_quantity_norm,train_prev_proj_norm,x_essay_tfidf,x_title_tfidf))
print(train_tfidf.shape)

test_tfidf = hstack((test_categories_p1,test_categories_p0,␣
 ↪test_subcategories_p1,test_subcategories_p0,
                    test_state_p1,test_state_p0,test_prefix_p1,test_prefix_p0,
                    test_grade_p1,test_grade_p0,
                   ␣
 ↪test_price_norm,test_quantity_norm,test_prev_proj_norm,essay_tfidf,title_tfidf))
print(test_tfidf.shape)
```

```
(33500, 7738)
(16500, 7738)
```

[353]:
```
#set 3
train_avg = hstack((train_cat_p1,train_cat_p0,train_subcat_p1,train_subcat_p0,
                    ␣
 ↪train_state_mini_p1,train_state_mini_p0,train_prefix_mini_p1,train_prefix_mini_p0,
                     train_grade_mini_p1,train_grade_mini_p0,
                     train_price,train_quantity,
                     train_prev_proj,x_essay_avg,x_title_avg))
print(train_avg.shape)

test_avg = hstack((test_cat_p1,test_cat_p0,test_subcat_p1,test_subcat_p0,
                   ␣
 ↪test_state_mini_p1,test_state_mini_p0,test_prefix_mini_p1,test_prefix_mini_p0,
                     test_grade_mini_p1,test_grade_mini_p0,
                    test_price,test_quantity,test_prev_proj,essay_avg,title_avg))
print(test_avg.shape)
```

```
(13400, 613)
(6600, 613)
```

[354]:
```
#set 4
train_tfidf_avg =␣
 ↪hstack((train_cat_p1,train_cat_p0,train_subcat_p1,train_subcat_p0,
                  ␣
 ↪train_state_mini_p1,train_state_mini_p0,train_prefix_mini_p1,train_prefix_mini_p0,
                   train_grade_mini_p1,train_grade_mini_p0,
                   train_price,train_quantity,
                   train_prev_proj,x_essay_tfidf_avg,x_title_tfidf_avg))
print(train_tfidf_avg.shape)

test_tfidf_avg = hstack((test_cat_p1,test_cat_p0,test_subcat_p1,test_subcat_p0,
                  ␣
 ↪test_state_mini_p1,test_state_mini_p0,test_prefix_mini_p1,test_prefix_mini_p0,
                   test_grade_mini_p1,test_grade_mini_p0,
                  ␣
 ↪test_price,test_quantity,test_prev_proj,essay_tfidf_avg,title_tfidf_avg))
print(test_tfidf_avg.shape)
```

```
(13400, 613)
(6600, 613)
```

## 2 Assignment 9: RF and GBDT

**Response Coding: Example**

> The response tabel is built only on train dataset. For a category which is not there in train data and present in test data, we will encode them with default values Ex: in our test data if have State: D then we encode it as [0.5, 0.05]

Apply both Random Forrest and GBDT on these feature sets

Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(BOW) + preprocessed_eassay (BOW)

Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)

Set 3: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)

Set 4: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

The hyper paramter tuning (Consider any two hyper parameters preferably n_estimators, max_depth)

Consider the following range for hyperparameters n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000], max_depth = [2, 3, 4, 5, 6, 7, 8, 9, 10]

Find the best hyper parameter which will give the maximum AUC value

find the best hyper paramter using k-fold cross validation/simple cross validation data

use gridsearch cv or randomsearch cv or you can write your own for loops to do this task

Representation of results

You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure with X-axis as n_estimators, Y-axis as max_depth,

and Z-axis as AUC Score , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive 3d_scatter_plot.ipynb

```
    <p style="text-align:center;font-size:30px;color:red;"><strong>or</strong></p> <br>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='heat_map.JPG' width=300px> <a href='https://seaborn.pydata.org/generated/seaborn.heat
<li>You can choose either of the plotting techniques: 3d plot or heat map</li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
        </ul>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table for
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

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-leakage, 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. Random Forest and GBDT

[355]:
```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["Algorithm","Vectorizer", "Hyp.param1:n_estimators","Hyp.
↪param2:max_depth", "AUC"]
```

[356]:
```
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
```

Finding the best paramters of RandomForestClassifier using GridSearchCV

[357]:
```
#using gridsearchcv
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.
 ↪GridSearchCV.html
#https://stackoverflow.com/questions/52580023/
 ↪how-to-get-the-best-estimator-parameters-out-from-pipelined-gridsearch-and-cro
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
import math
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
import warnings
warnings.filterwarnings("ignore")

def enable_plotly_in_cell():
  import IPython
  from plotly.offline import init_notebook_mode
  display(IPython.core.display.HTML('''<script src="/static/components/
 ↪requirejs/require.js"></script>'''))
  init_notebook_mode(connected=False)


#max_depth,min_samples_split is the hyper parameter of Decision Tree
def Train_data(X_tr,y_train,vectorizer):
    max_depth= [2,3,4,5,6,7,8,9,10]
    n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]

    tuned_parameters = [{'n_estimators': n_estimators, 'max_depth': max_depth }]

    clf = GridSearchCV(RandomForestClassifier(class_weight='balanced'),␣
 ↪tuned_parameters, cv=3,
                       scoring='roc_auc', return_train_score=True)
    clf.fit(X_tr, y_train)

    best_parameters = clf.best_params_
```

```
   print("The best parameters for using this model is", best_parameters)
   print("Best Estimator ",clf.best_estimator_)

   K =  clf.cv_results_['param_max_depth']

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

   print("="*100)
   print("3D visualisation")
   # https://plot.ly/python/3d-axes/
   trace1 = go.Scatter3d(x=n_estimators,y=max_depth,z=train_auc, name =␣
↪'Train')
   trace2 = go.Scatter3d(x=n_estimators,y=max_depth,z=cv_auc, name = 'Cross␣
↪validation')
   data = [trace1, trace2]
   enable_plotly_in_cell()

   layout = go.Layout(scene = dict(
           xaxis = dict(title='n_estimators'),
           yaxis = dict(title='max_depth'),
           zaxis = dict(title='AUC'),))

   fig = go.Figure(data=data, layout=layout)
   offline.iplot(fig, filename='3d-scatter-colorscale')

   print("="*100)

   print("HeatMap of Train and CV")
   #Heatmap https://www.kaggle.com/arindambanerjee/grid-search-simplified
   max_depth_list = list(clf.cv_results_['param_max_depth'].data)
   n_estimators_list = list(clf.cv_results_['param_n_estimators'].data)

   sns.set_style("whitegrid")
   plt.figure(figsize=(16,6))
   plt.subplot(1,2,1)
   data = pd.DataFrame(data={'n_estimators':n_estimators_list, 'Max Depth':
↪max_depth_list, 'AUC':clf.cv_results_['mean_train_score']})
   data = data.pivot(index='n_estimators', columns='Max Depth', values='AUC')
   sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('AUC for Training␣
↪data')

   plt.subplot(1,2,2)
   data = pd.DataFrame(data={'n_estimators':n_estimators_list, 'Max Depth':
↪max_depth_list, 'AUC':clf.cv_results_['mean_test_score']})
```

```
    data = data.pivot(index='n_estimators', columns='Max Depth', values='AUC')
    sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('AUC for CV data')
    plt.show()


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

Testing dataset with RandomForest best parameters

[358]:
```
#Round off the predicted values to fix value error
#https://stackoverflow.com/questions/38015181/
 ↪accuracy-score-valueerror-cant-handle-mix-of-binary-and-continuous-target
from mlxtend.plotting import plot_confusion_matrix
from sklearn.calibration import CalibratedClassifierCV
from graphviz import Source

def draw_train_confusion_matrix(CM):
    fig, ax = plot_confusion_matrix(conf_mat= CM, colorbar=True,
                                    show_absolute=True,
                                    show_normed=True)
    plt.title("Train Confusion Matrix ")
    plt.ylabel("Actual")
    plt.xlabel("Predicted")
    plt.show()

def draw_test_confusion_matrix(CM):
    fig, ax = plot_confusion_matrix(conf_mat= CM, colorbar=True,
                                    show_absolute=True,
                                    show_normed=True)
    plt.title("Test Confusion Matrix ")
    plt.ylabel("Actual")
    plt.xlabel("Predicted")
    plt.show()


def Test_Data(X_train, y_train,X_test,y_test,max_depth,n_estimators,vectorizer):
    clf = RandomForestClassifier(n_estimators=n_estimators,␣
 ↪max_depth=max_depth,class_weight='balanced')
    clf.fit(X_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability␣
 ↪estimates of the positive class
    # not the predicted outputs

    X_train = X_train.tocsr()
    X_test = X_test.tocsr()
```

```
    clf_calibrated=CalibratedClassifierCV(clf, cv='prefit', method='isotonic')
    clf_calibrated.fit(X_train, y_train)

    y_train_pred = clf_calibrated.predict_proba(X_train)[:,1]
    y_test_pred =  clf_calibrated.predict_proba(X_test)[:,1]


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

    sns.set_style("whitegrid");
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr,␣
↪train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr,␣
↪test_tpr)))
    plt.legend()
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.title("ROC Curve using %s "%vectorizer)
    plt.show()

    print("="*100)

    from sklearn.metrics import confusion_matrix
    print("Train confusion matrix")

    y_train_pred_val = []
    for item in y_train_pred:
        y_train_pred_val.append(int(round(item)))

    y_test_pred_val = []
    for item in y_test_pred:
        round_item =int(round(item))
        y_test_pred_val.append(round_item)

    print("="*100)
    from sklearn.metrics import confusion_matrix
    best_t = find_best_threshold(thresholds, train_fpr, train_tpr)

    print("Train confusion matrix")
    predicted_train=predict_with_best_t(y_train_pred_val, best_t)
    cm_train = confusion_matrix(y_train, predicted_train)
    print(cm_train)
    draw_train_confusion_matrix(cm_train)

    print("Test confusion matrix")
    predicted_test=predict_with_best_t(y_test_pred_val, best_t)
```

```
    cm_test = confusion_matrix(y_test, predicted_test)
    print(cm_test)
    draw_test_confusion_matrix(cm_test)

    #Adding the results to prettytable
    table.add_row(["RF",vectorizer,n_estimators, max_depth, str(auc(test_fpr,␣
↪test_tpr))])
```

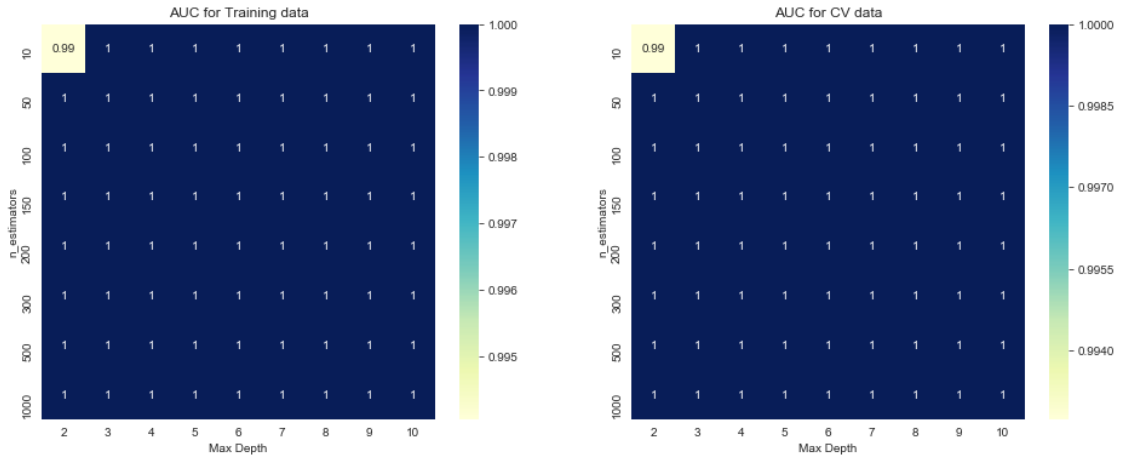
2.4 Applying Random Forest

Apply Random Forest 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 instrucations

#### 2.0.1 2.4.1 Applying Random Forests on BOW, SET 1

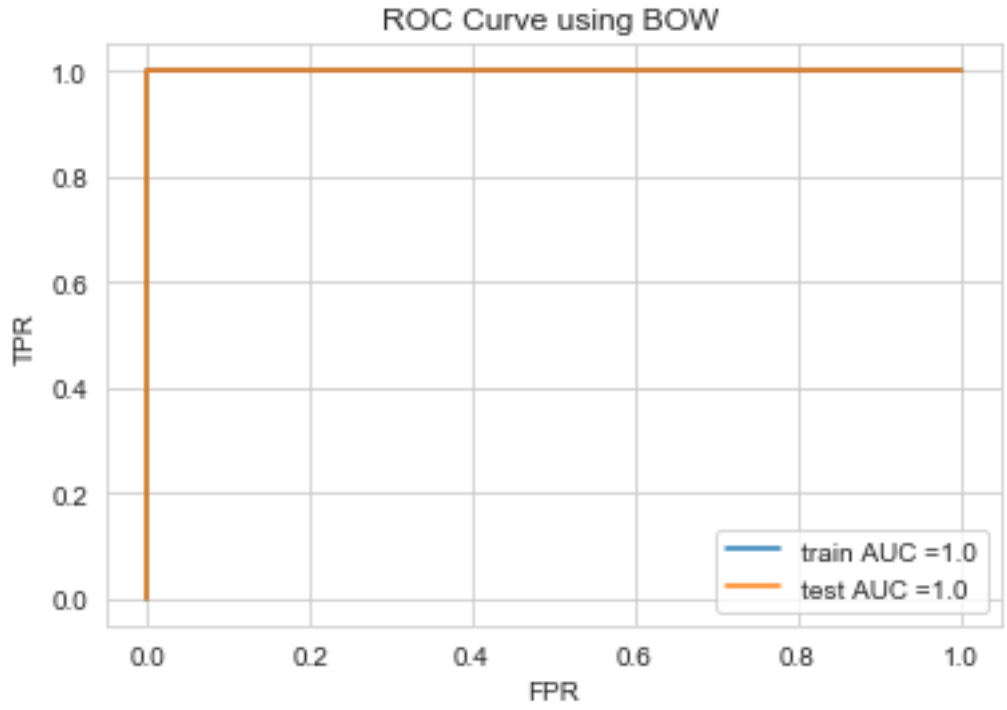
[359]:
```
# Please write all the code with proper documentation
Train_data(train_bow,y_train,vectorizer="BOW")
```

```
The best parameters for using this model is {'max_depth': 2, 'n_estimators': 50}
Best Estimator  RandomForestClassifier(bootstrap=True, class_weight='balanced',
                       criterion='gini', max_depth=2, max_features='auto',
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=50, n_jobs=None, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
========================================================================
====================
3D visualisation

<IPython.core.display.HTML object>


========================================================================
====================
HeatMap of Train and CV
```



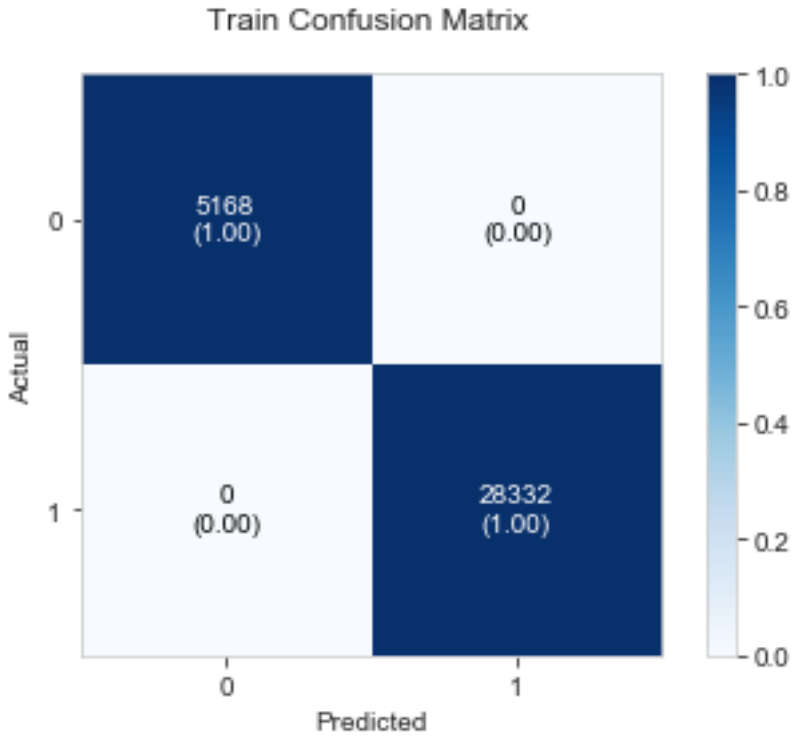

Training BOW with best features from GridSearchcv

[383]:
```
Test_Data(train_bow,y_train,test_bow,y_test, max_depth=2,␣
↪n_estimators=50,vectorizer="BOW")
```

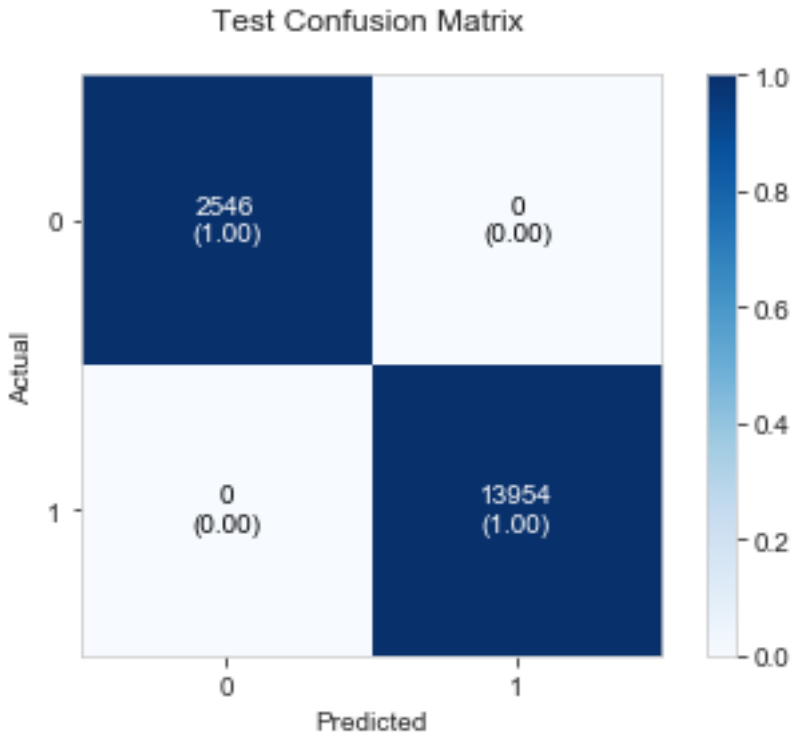



```
====================================================================================================
Train confusion matrix
```

```
================================================================================
====================
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Train confusion matrix
[[ 5168     0]
 [    0 28332]]
```



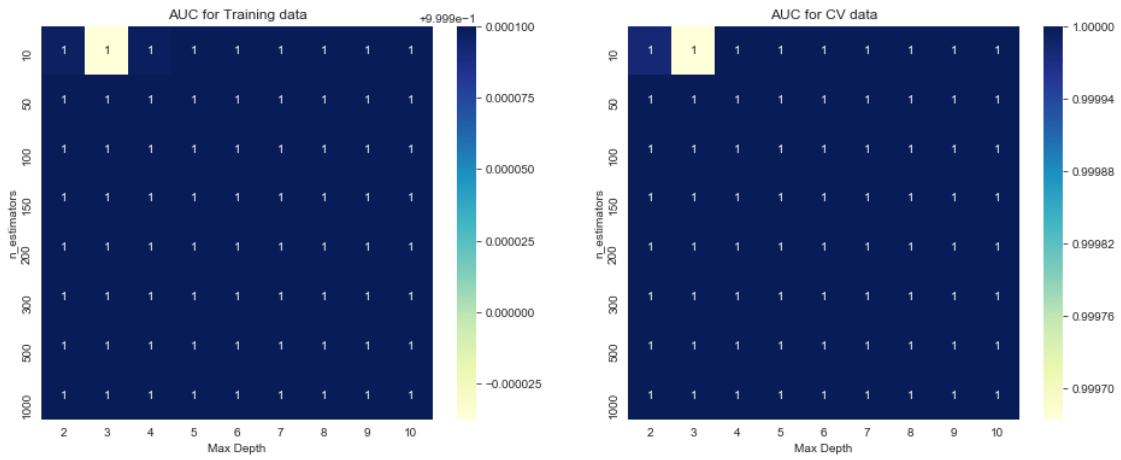

```
Test confusion matrix
[[ 2546     0]
 [    0 13954]]
```



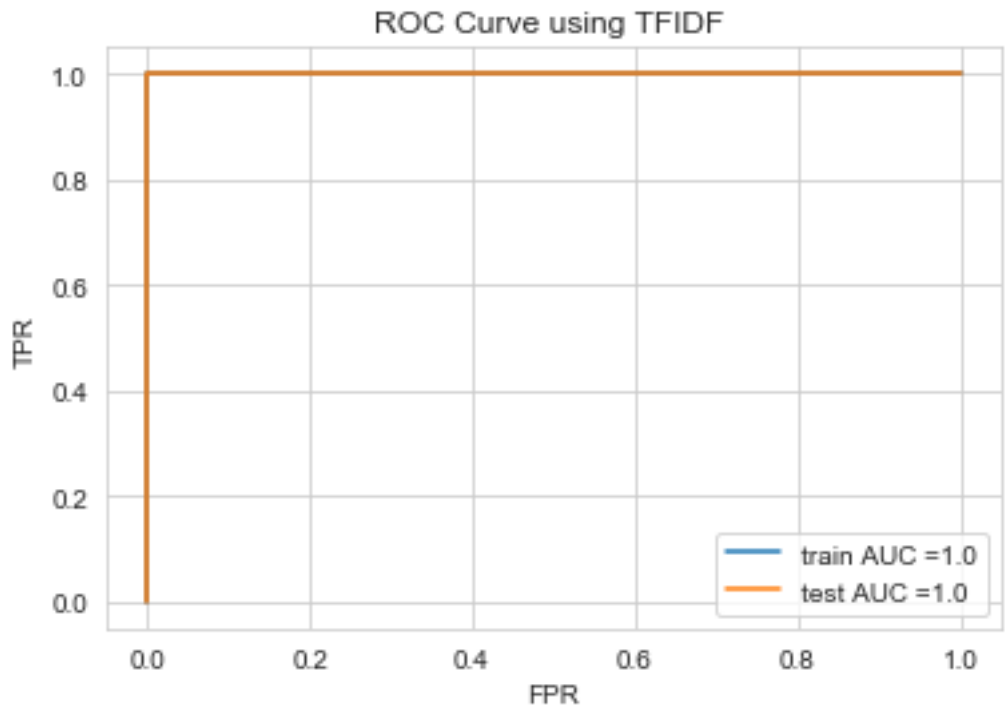

### 2.0.2 2.4.2 Applying Random Forests on TFIDF, SET 2

[361]:
```
Train_data(train_tfidf,y_train,vectorizer="TFIDF")
```

```
The best parameters for using this model is {'max_depth': 2, 'n_estimators': 50}
Best Estimator  RandomForestClassifier(bootstrap=True, class_weight='balanced',
                       criterion='gini', max_depth=2, max_features='auto',
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=50, n_jobs=None, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
========================================================================
====================
3D visualisation

<IPython.core.display.HTML object>


========================================================================
====================
HeatMap of Train and CV
```

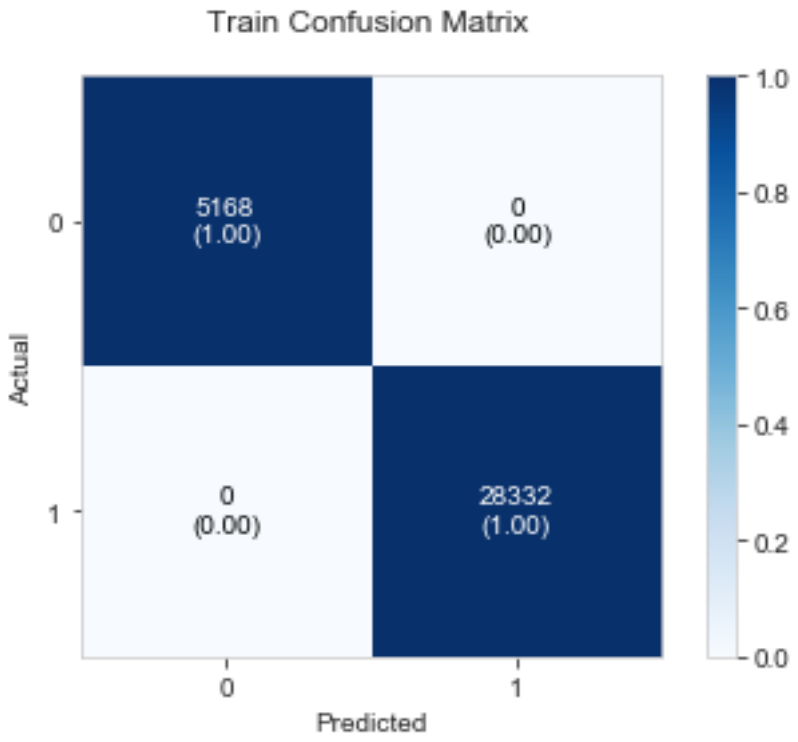



[384]:
```
Test_Data(train_tfidf,y_train,test_tfidf,y_test,max_depth=2,␣
↪n_estimators=50,vectorizer="TFIDF")
```

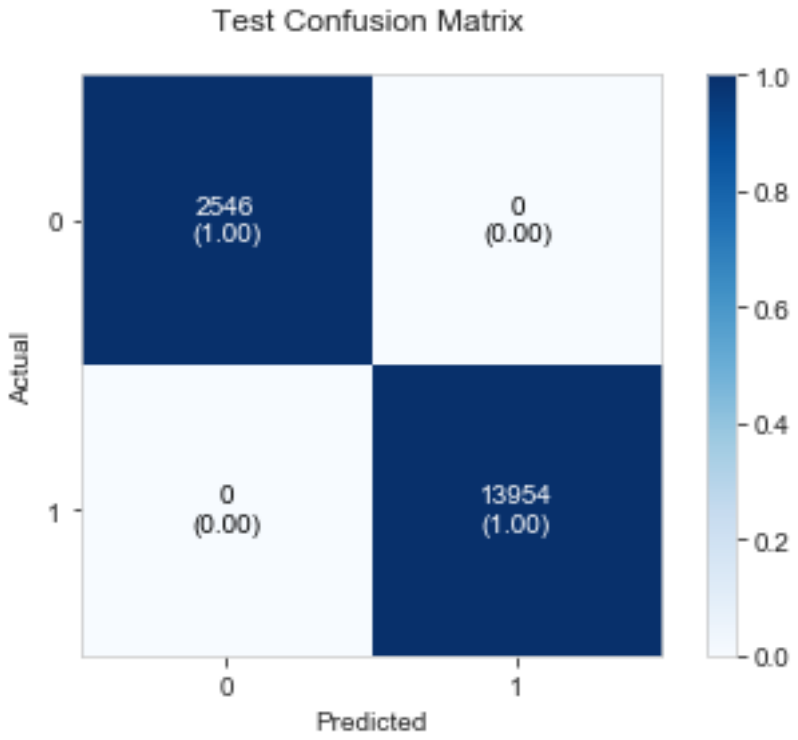



```
====================================================================================================
Train confusion matrix
====================================================================================================
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Train confusion matrix
[[ 5168     0]
 [    0 28332]]
```



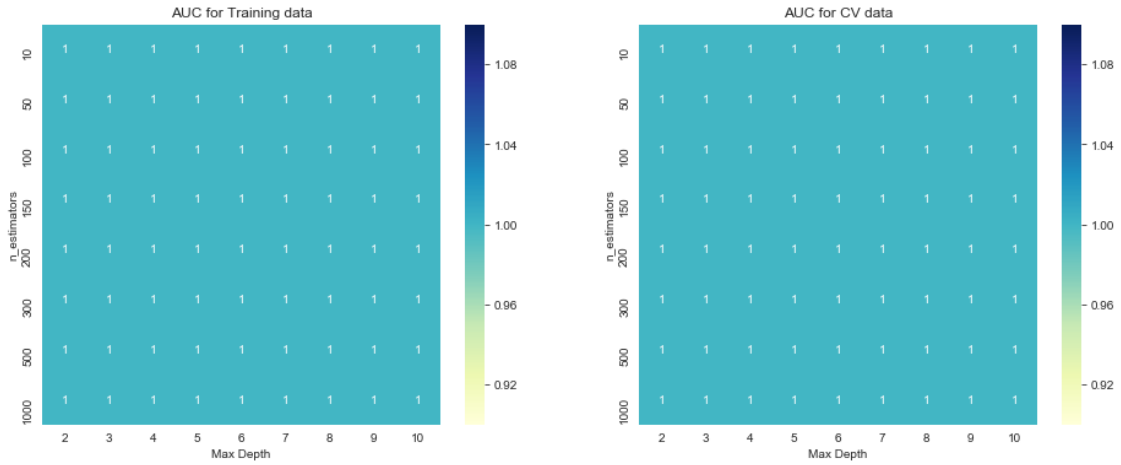

```
Test confusion matrix
[[ 2546     0]
 [    0 13954]]
```



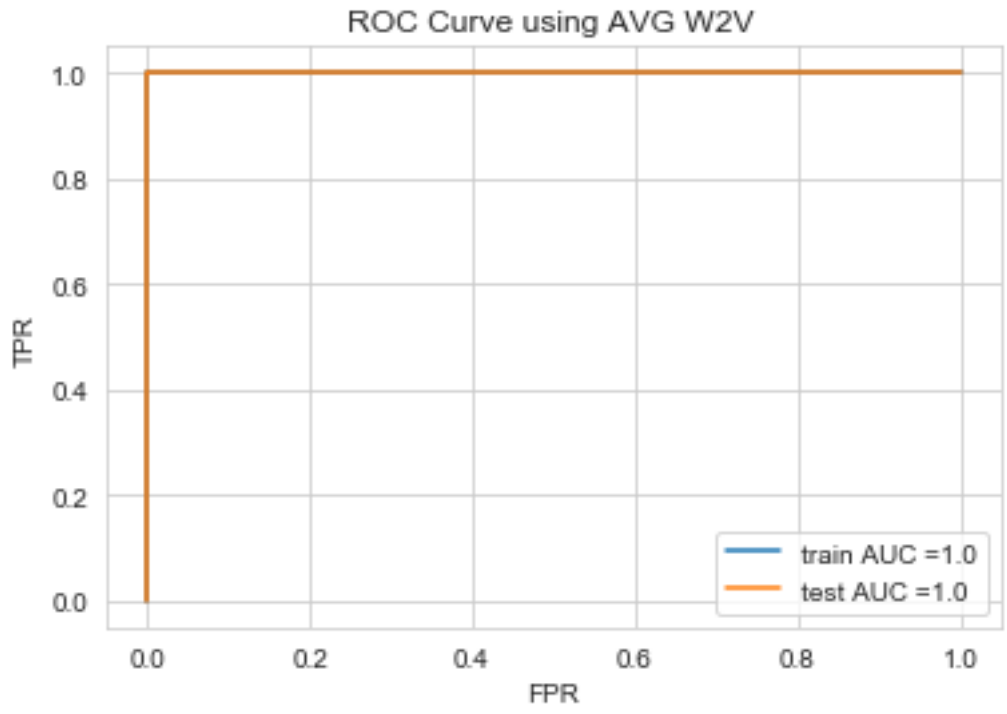

### 2.0.3 2.4.3 Applying Random Forests on AVG W2V, SET 3

[363]:
```
Train_data(train_avg,y_train_mini,vectorizer="AVG W2V")
```

```
The best parameters for using this model is {'max_depth': 2, 'n_estimators': 10}
Best Estimator  RandomForestClassifier(bootstrap=True, class_weight='balanced',
                      criterion='gini', max_depth=2, max_features='auto',
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=10, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
========================================================================
====================
3D visualisation

<IPython.core.display.HTML object>


========================================================================
====================
HeatMap of Train and CV
```

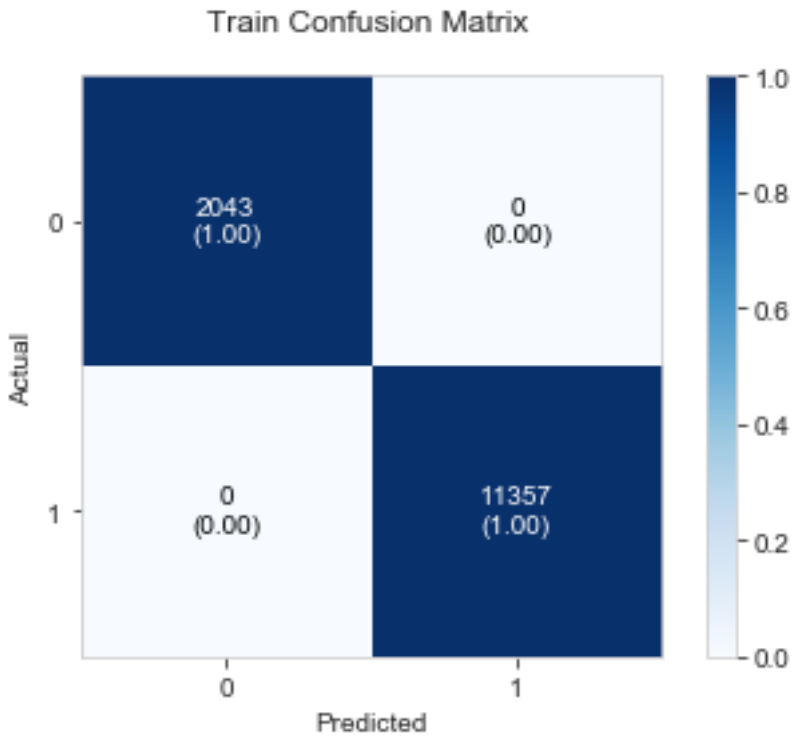



[385]:
```
Test_Data(train_avg,y_train_mini,test_avg,y_test_mini,max_depth=2,␣
↪n_estimators=10,vectorizer="AVG W2V")
```

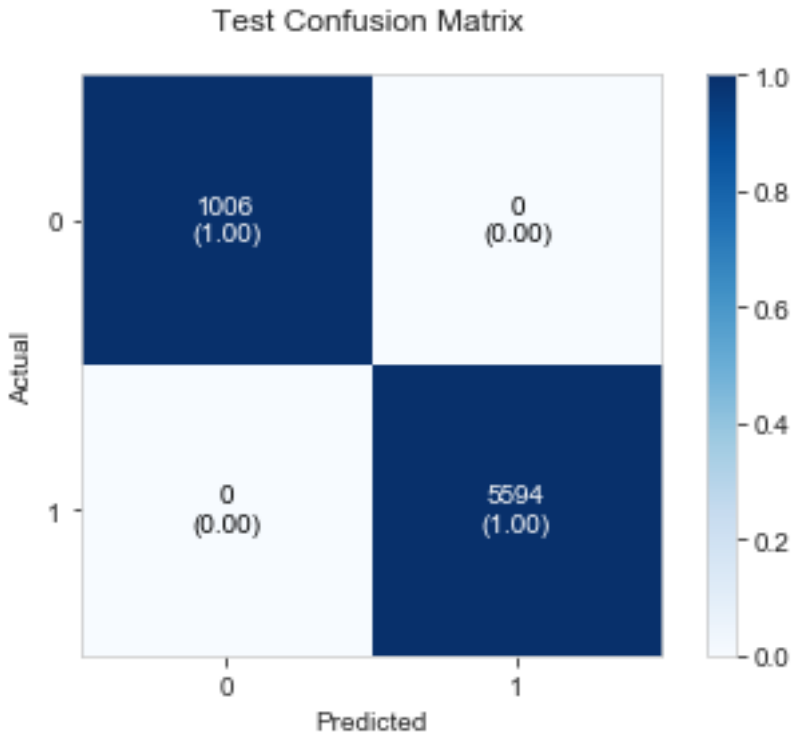



```
==========================================================================
====================
Train confusion matrix
==========================================================================
====================
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Train confusion matrix
[[ 2043     0]
 [    0 11357]]
```



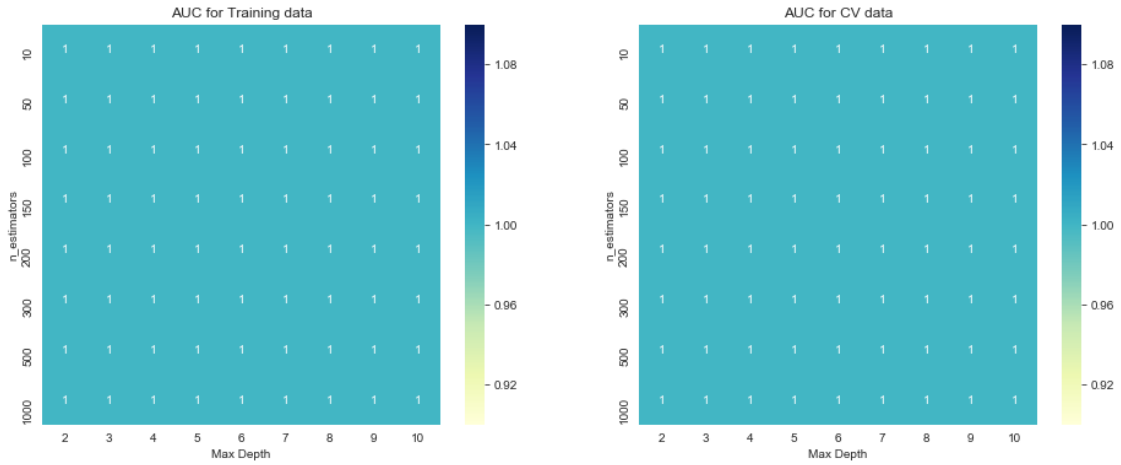

```
Test confusion matrix
[[1006    0]
 [   0 5594]]
```



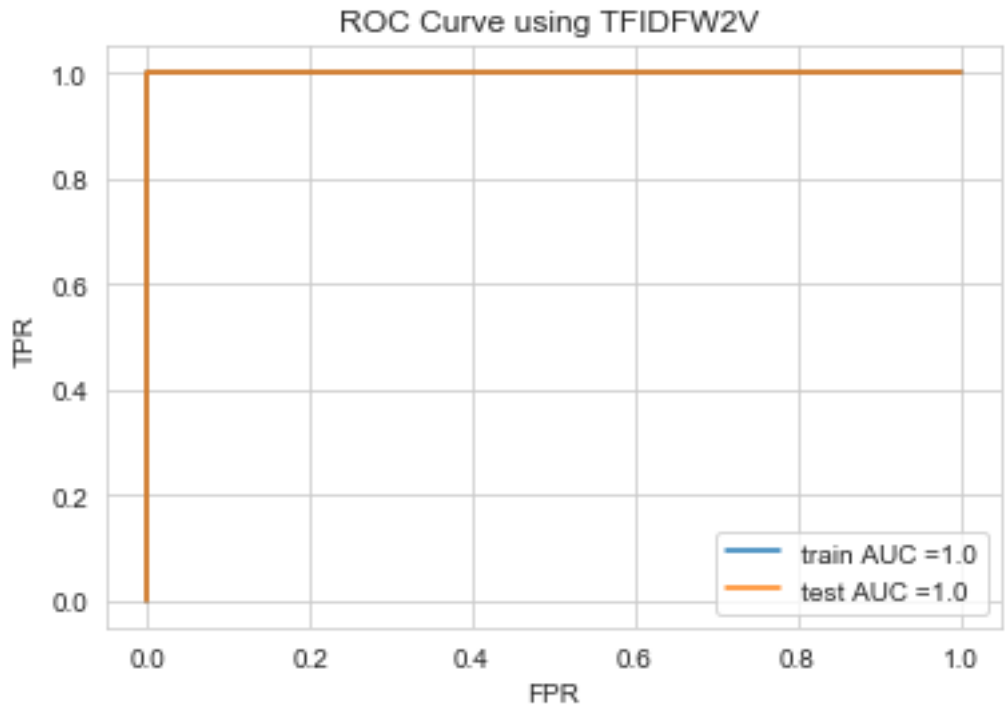

### 2.0.4 2.4.4 Applying Random Forests on TFIDF W2V, SET 4

[365]:
```
Train_data(train_tfidf_avg,y_train_mini,vectorizer="TFIDFW2V")
```

```
The best parameters for using this model is {'max_depth': 2, 'n_estimators': 10}
Best Estimator  RandomForestClassifier(bootstrap=True, class_weight='balanced',
                       criterion='gini', max_depth=2, max_features='auto',
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=10, n_jobs=None, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
================================================================================
====================
3D visualisation

<IPython.core.display.HTML object>


================================================================================
====================
HeatMap of Train and CV
```

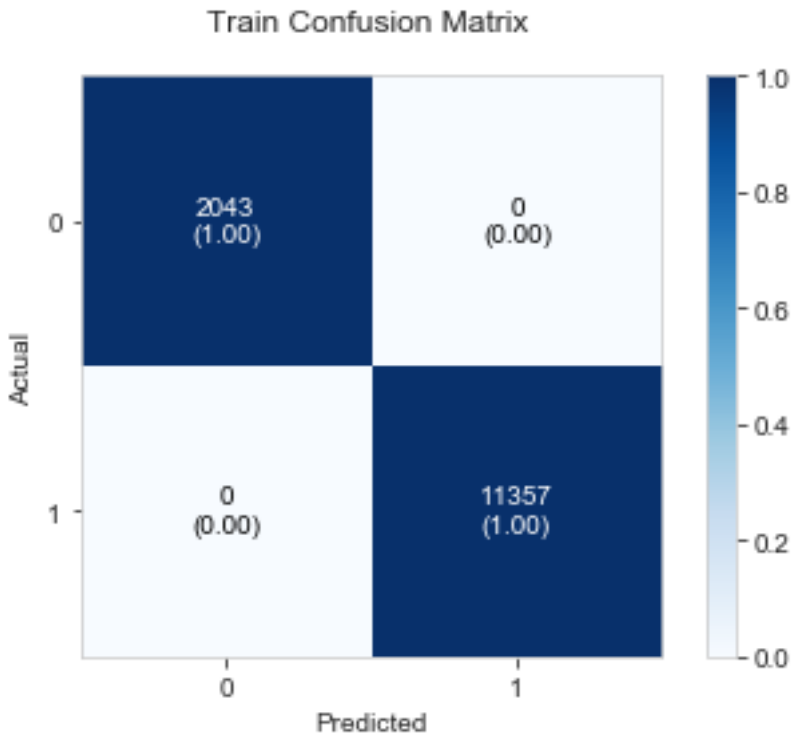



[386]:
```
Test_Data(train_tfidf_avg,y_train_mini,test_tfidf_avg,y_test_mini,max_depth=2,␣
↪n_estimators=10,vectorizer="TFIDFW2V")
```

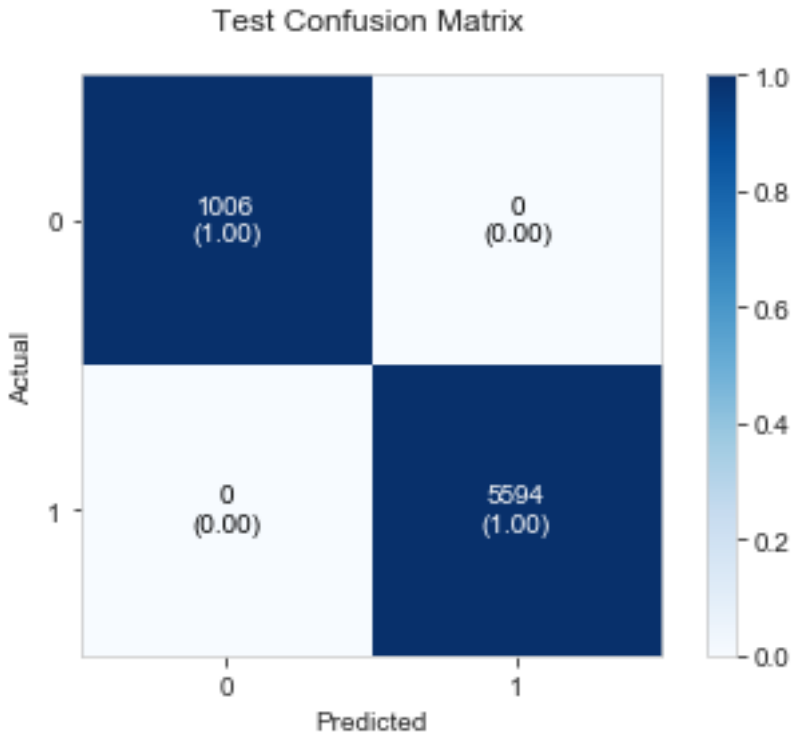



```
==========================================================================
====================
Train confusion matrix
==========================================================================
====================
```

```
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Train confusion matrix
[[ 2043     0]
 [    0 11357]]
```




```
Test confusion matrix
[[1006    0]
 [   0 5594]]
```




2.5 Applying GBDT

Apply GBDT 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 instrucations

[371]:
```
#using gridsearchcv
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.
 ↪GridSearchCV.html
#https://stackoverflow.com/questions/52580023/
 ↪how-to-get-the-best-estimator-parameters-out-from-pipelined-gridsearch-and-cro
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
import math
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
import warnings
warnings.filterwarnings("ignore")
```

```
def enable_plotly_in_cell():
  import IPython
  from plotly.offline import init_notebook_mode
  display(IPython.core.display.HTML('''<script src="/static/components/
 ↪requirejs/require.js"></script>'''))
  init_notebook_mode(connected=False)


#max_depth,min_samples_split is the hyper parameter of Decision Tree
def Train_GB_data(X_tr,y_train,vectorizer):
    max_depth= [2,3,4,5,6,8,10]
    n_estimators = [10, 50, 100, 150, 200, 300, 500, 1000]

    tuned_parameters = [{'n_estimators': n_estimators, 'max_depth': max_depth }]

    clf = GridSearchCV(GradientBoostingClassifier(), tuned_parameters, cv=3,
                       scoring='roc_auc', return_train_score=True)
    clf.fit(X_tr, y_train)

    best_parameters = clf.best_params_
    print("The best parameters for using this model is", best_parameters)
    print("Best Estimator ",clf.best_estimator_)

    K =  clf.cv_results_['param_max_depth']

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

    print("="*100)
    print("3D visualisation")
    # https://plot.ly/python/3d-axes/
    trace1 = go.Scatter3d(x=n_estimators,y=max_depth,z=train_auc, name =␣
 ↪'Train')
    trace2 = go.Scatter3d(x=n_estimators,y=max_depth,z=cv_auc, name = 'Cross␣
 ↪validation')
    data = [trace1, trace2]
    enable_plotly_in_cell()

    layout = go.Layout(scene = dict(
            xaxis = dict(title='n_estimators'),
            yaxis = dict(title='max_depth'),
            zaxis = dict(title='AUC'),))

    fig = go.Figure(data=data, layout=layout)
```

```
    offline.iplot(fig, filename='3d-scatter-colorscale')

    print("="*100)

    print("HeatMap of Train and CV")
    #Heatmap https://www.kaggle.com/arindambanerjee/grid-search-simplified
    max_depth_list = list(clf.cv_results_['param_max_depth'].data)
    n_estimators_list = list(clf.cv_results_['param_n_estimators'].data)

    sns.set_style("whitegrid")
    plt.figure(figsize=(16,6))
    plt.subplot(1,2,1)
    data = pd.DataFrame(data={'n_estimators':n_estimators_list, 'Max Depth':
↪max_depth_list, 'AUC':clf.cv_results_['mean_train_score']})
    data = data.pivot(index='n_estimators', columns='Max Depth', values='AUC')
    sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('AUC for Training␣
↪data')

    plt.subplot(1,2,2)
    data = pd.DataFrame(data={'n_estimators':n_estimators_list, 'Max Depth':
↪max_depth_list, 'AUC':clf.cv_results_['mean_test_score']})
    data = data.pivot(index='n_estimators', columns='Max Depth', values='AUC')
    sns.heatmap(data, annot=True, cmap="YlGnBu").set_title('AUC for CV data')
    plt.show()


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

[368]:
```
#Round off the predicted values to fix value error
#https://stackoverflow.com/questions/38015181/
 ↪accuracy-score-valueerror-cant-handle-mix-of-binary-and-continuous-target
from mlxtend.plotting import plot_confusion_matrix
from sklearn.calibration import CalibratedClassifierCV
from graphviz import Source

def draw_train_confusion_matrix(CM):
    fig, ax = plot_confusion_matrix(conf_mat= CM, colorbar=True,
                                    show_absolute=True,
                                    show_normed=True)
    plt.title("Train Confusion Matrix ")
    plt.ylabel("Actual")
    plt.xlabel("Predicted")
    plt.show()

def draw_test_confusion_matrix(CM):
```

```
    fig, ax = plot_confusion_matrix(conf_mat= CM, colorbar=True,
                                    show_absolute=True,
                                    show_normed=True)
    plt.title("Test Confusion Matrix ")
    plt.ylabel("Actual")
    plt.xlabel("Predicted")
    plt.show()


def Test_GB_Data(X_train,␣
↪y_train,X_test,y_test,max_depth,n_estimators,vectorizer):
    clf = GradientBoostingClassifier(n_estimators=n_estimators,␣
↪max_depth=max_depth)
    clf.fit(X_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability␣
↪estimates of the positive class
    # not the predicted outputs

    X_train = X_train.tocsr()
    X_test = X_test.tocsr()

    clf_calibrated=CalibratedClassifierCV(clf, cv='prefit', method='isotonic')
    clf_calibrated.fit(X_train, y_train)

    y_train_pred = clf_calibrated.predict_proba(X_train)[:,1]
    y_test_pred =  clf_calibrated.predict_proba(X_test)[:,1]


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

    sns.set_style("whitegrid");
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr,␣
↪train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr,␣
↪test_tpr)))
    plt.legend()
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.title("ROC Curve using %s "%vectorizer)
    plt.show()

    print("="*100)

    from sklearn.metrics import confusion_matrix
    print("Train confusion matrix")
```

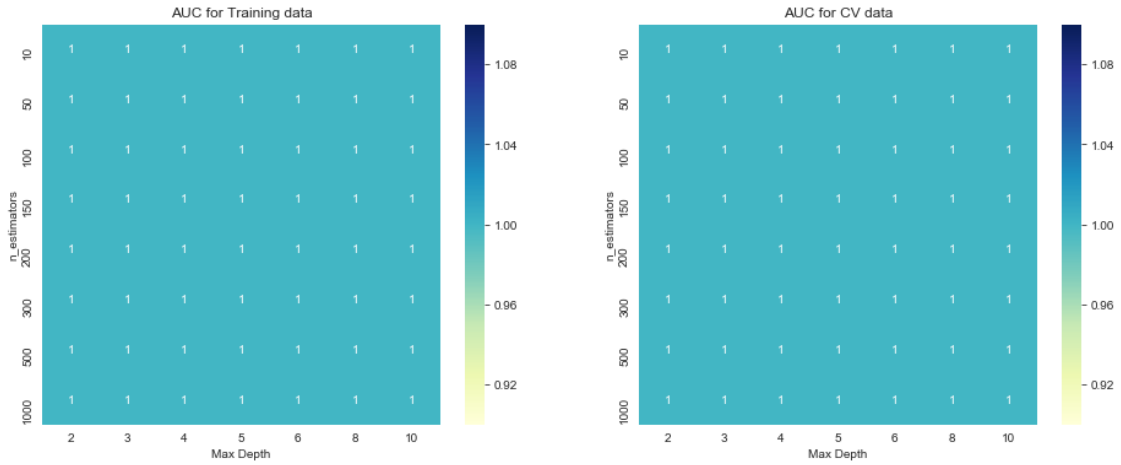
```
    y_train_pred_val = []
    for item in y_train_pred:
        y_train_pred_val.append(int(round(item)))

    y_test_pred_val = []
    for item in y_test_pred:
        round_item =int(round(item))
        y_test_pred_val.append(round_item)

    print("="*100)
    from sklearn.metrics import confusion_matrix
    best_t = find_best_threshold(thresholds, train_fpr, train_tpr)

    print("Train confusion matrix")
    predicted_train=predict_with_best_t(y_train_pred_val, best_t)
    cm_train = confusion_matrix(y_train, predicted_train)
    print(cm_train)
    draw_train_confusion_matrix(cm_train)

    print("Test confusion matrix")
    predicted_test=predict_with_best_t(y_test_pred_val, best_t)
    cm_test = confusion_matrix(y_test, predicted_test)
    print(cm_test)
    draw_test_confusion_matrix(cm_test)

    #Adding the results to prettytable
    table.add_row(["GBDT",vectorizer,n_estimators, max_depth, str(auc(test_fpr,␣
↪test_tpr))])
```

### 2.0.5 2.5.1 Applying GBDT on BOW, SET 1

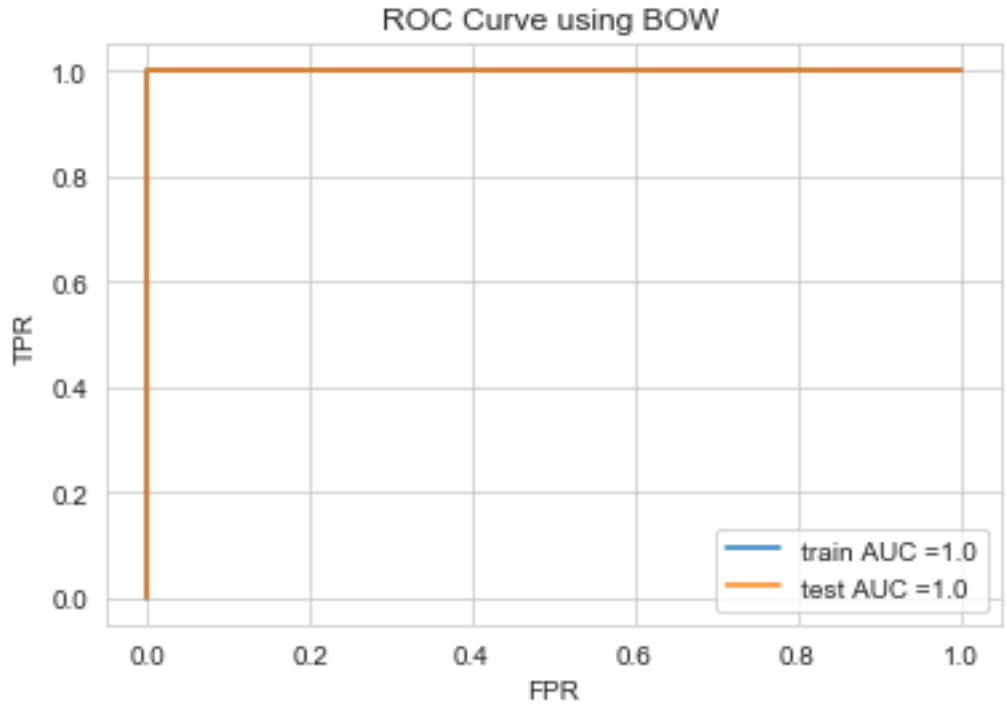
[372]:
```
# Please write all the code with proper documentation
Train_GB_data(train_bow,y_train,vectorizer="BOW")
```

```
The best parameters for using this model is {'max_depth': 2, 'n_estimators': 10}
Best Estimator  GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=2,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=10,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
================================================================================
```

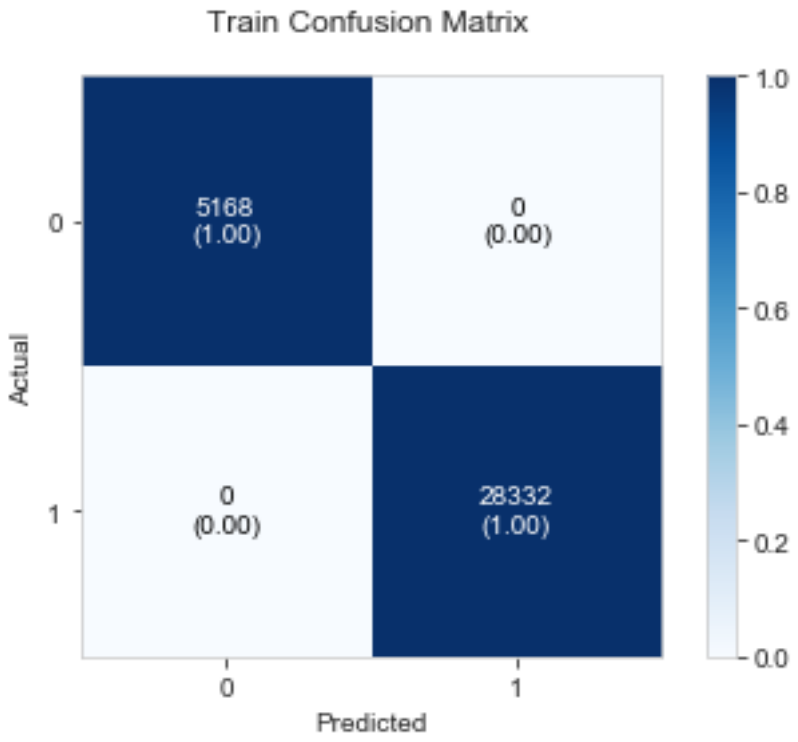
```
====================
3D visualisation

<IPython.core.display.HTML object>

================================================================================
====================
HeatMap of Train and CV
```



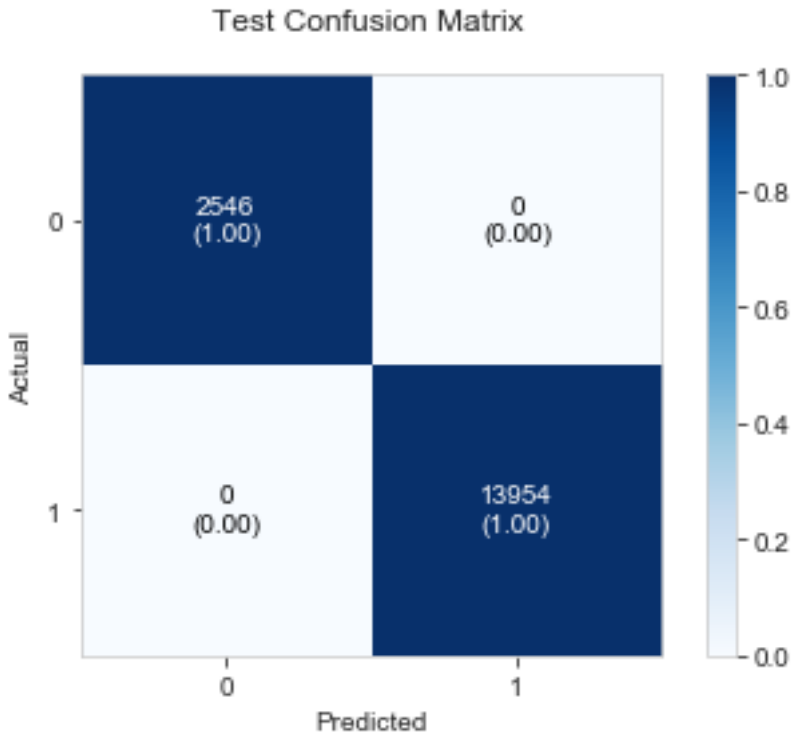

[387]:
```
Test_GB_Data(train_bow,y_train,test_bow,y_test, max_depth=2,␣
↪n_estimators=10,vectorizer="BOW")
```



```
================================================================================
====================
Train confusion matrix
================================================================================
====================
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Train confusion matrix
[[ 5168     0]
 [    0 28332]]
```




```
Test confusion matrix
[[ 2546     0]
 [    0 13954]]
```

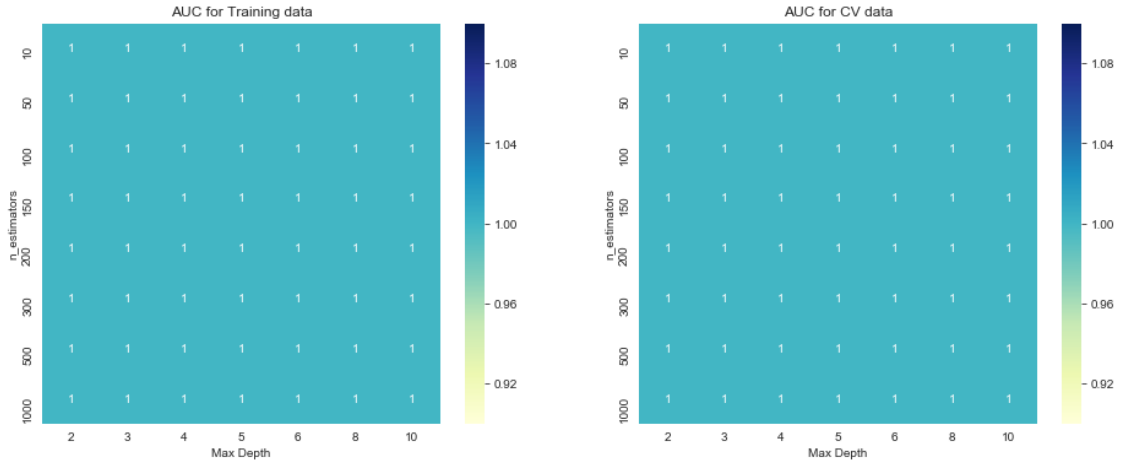



### 2.0.6 2.5.2 Applying GBDT on TFIDF, SET 2

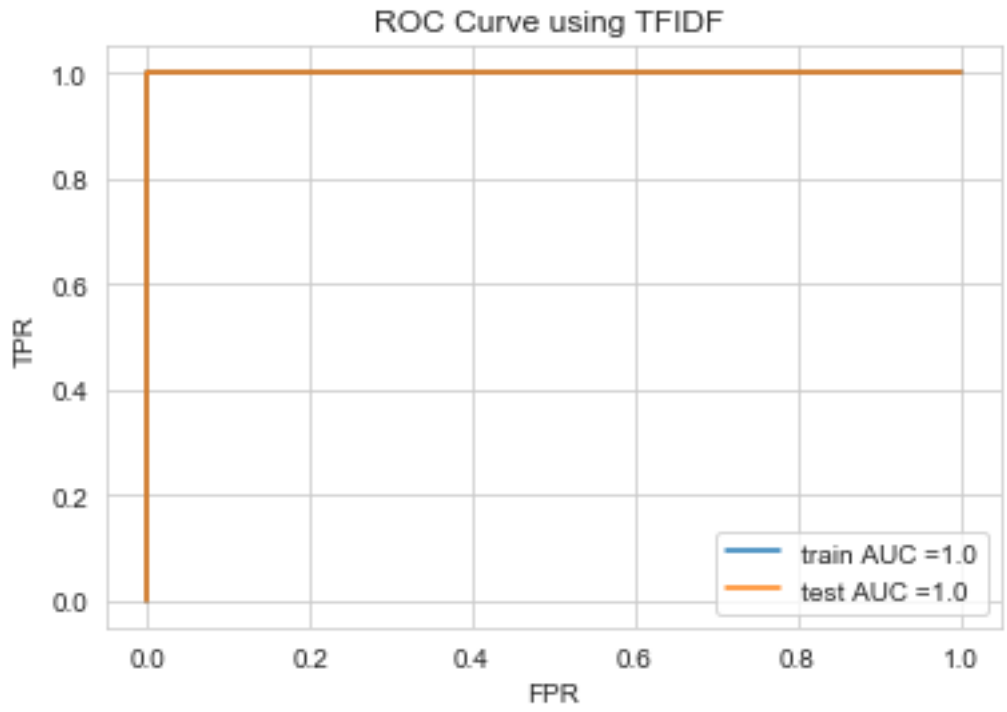
[374]:
```
Train_GB_data(train_tfidf,y_train,vectorizer="TFIDF")
```

```
The best parameters for using this model is {'max_depth': 2, 'n_estimators': 10}
Best Estimator  GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=2,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=10,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
========================================================================
====================
3D visualisation

<IPython.core.display.HTML object>
```

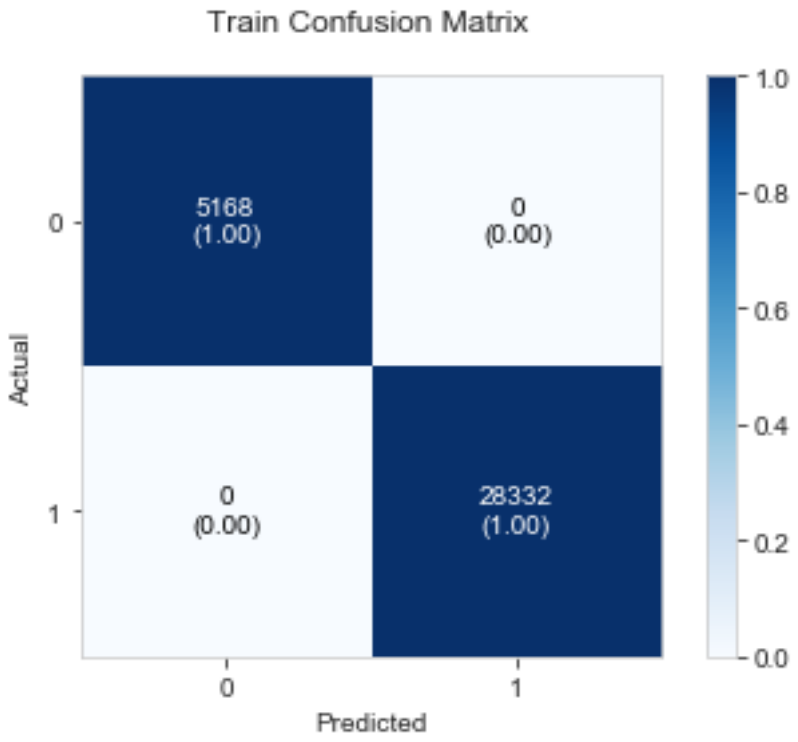
```
==========================================================================================
====================
HeatMap of Train and CV
```



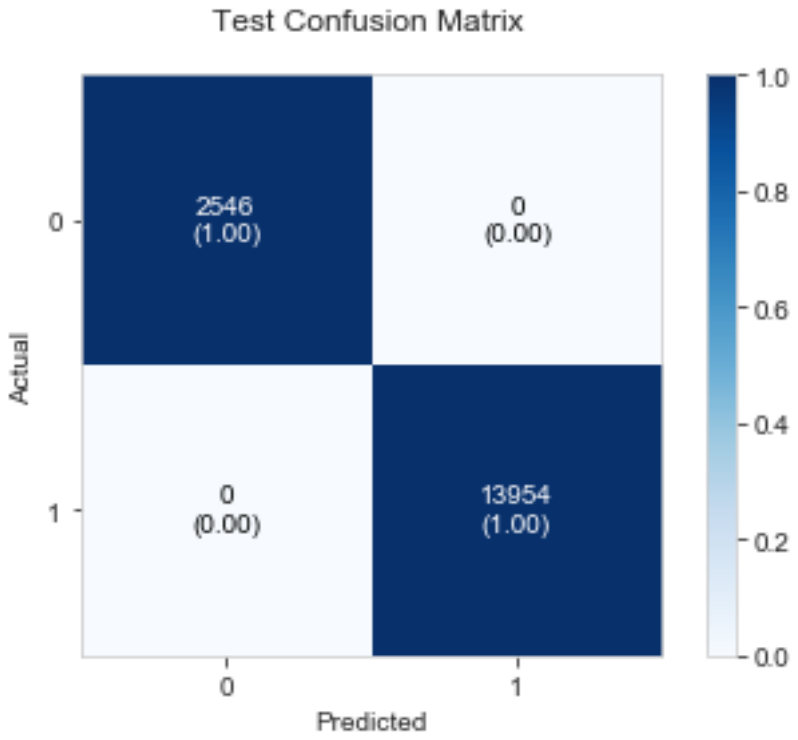

[390]:
```
Test_GB_Data(train_tfidf,y_train,test_tfidf,y_test,max_depth=2,␣
↪n_estimators=10,vectorizer="TFIDF")
```



```
================================================================================
====================
Train confusion matrix
================================================================================
====================
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Train confusion matrix
[[ 5168     0]
 [    0 28332]]
```




```
Test confusion matrix
[[ 2546     0]
 [    0 13954]]
```

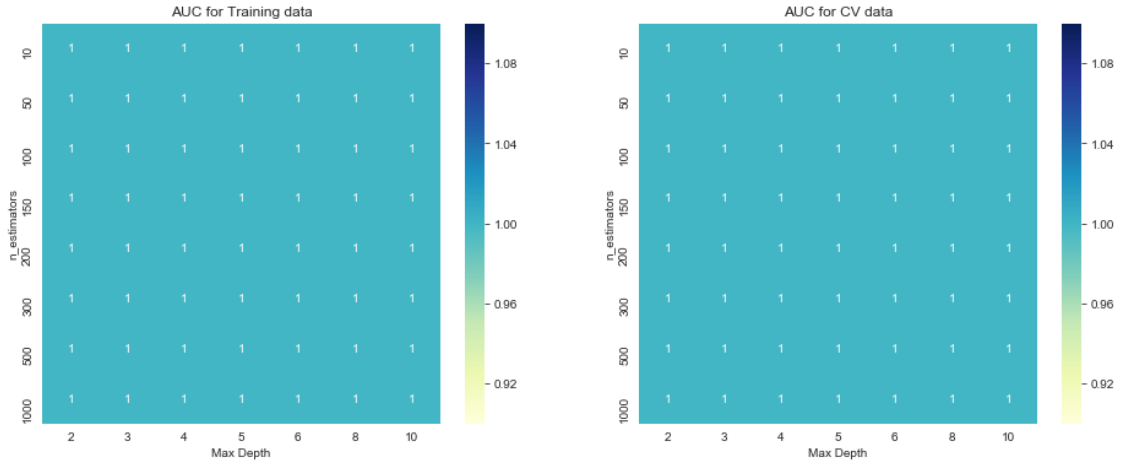



### 2.0.7 2.5.3 Applying GBDT on AVG W2V, SET 3

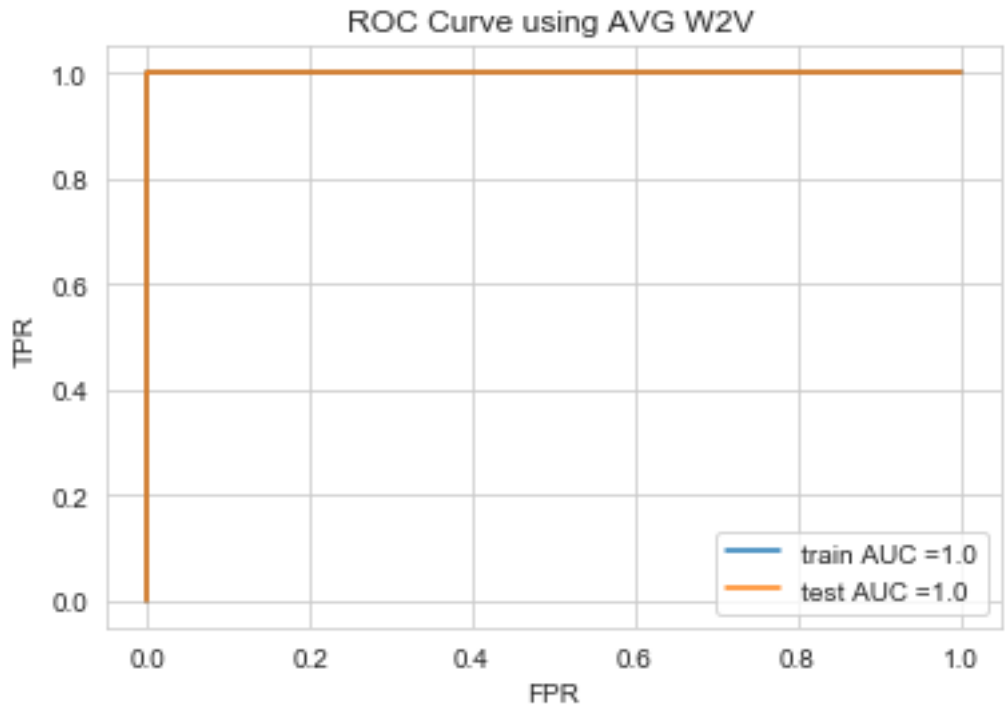
[378]:
```
Train_GB_data(train_avg,y_train_mini,vectorizer="AVG W2V")
```

```
The best parameters for using this model is {'max_depth': 2, 'n_estimators': 10}
Best Estimator  GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=2,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=10,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
========================================================================
====================
3D visualisation

<IPython.core.display.HTML object>
```

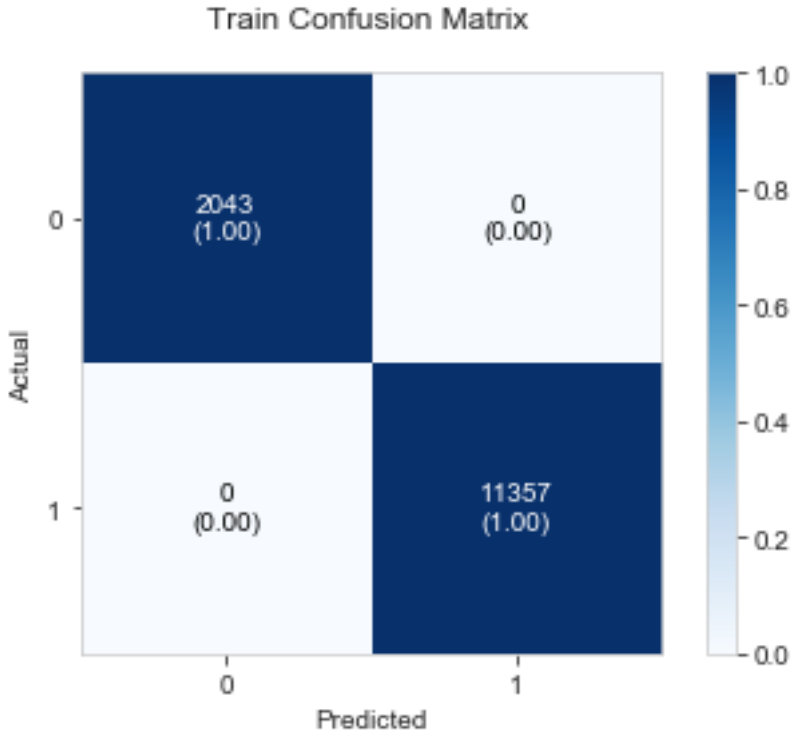
```
==========================================================================================
====================
HeatMap of Train and CV
```



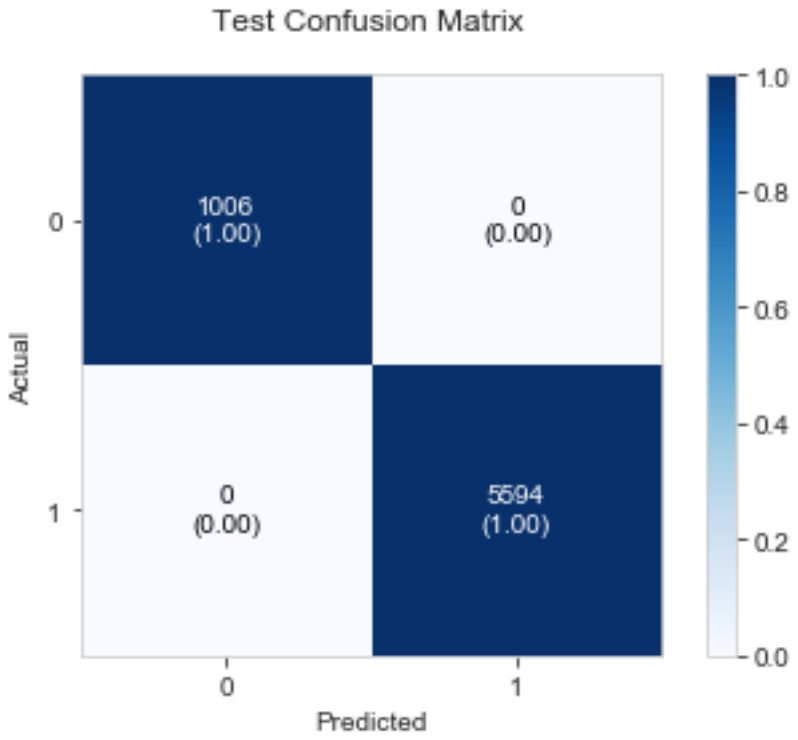

[391]:
```
Test_GB_Data(train_avg,y_train_mini,test_avg,y_test_mini,max_depth=2,␣
 ↪n_estimators=10,vectorizer="AVG W2V")
```



```
================================================================================
====================
Train confusion matrix
================================================================================
====================
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Train confusion matrix
[[ 2043     0]
 [    0 11357]]
```




```
Test confusion matrix
[[1006    0]
 [   0 5594]]
```

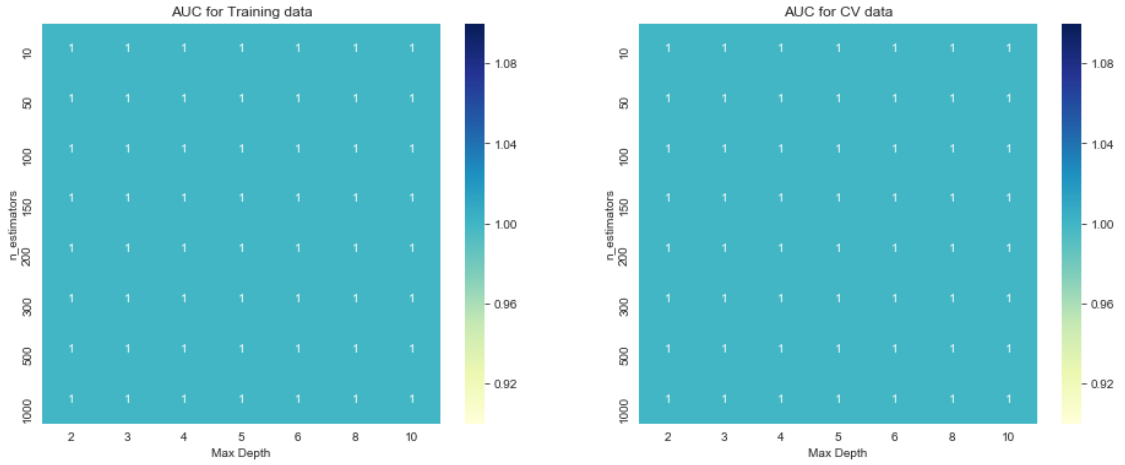



### 2.0.8 2.5.4 Applying GBDT on TFIDF W2V, SET 4

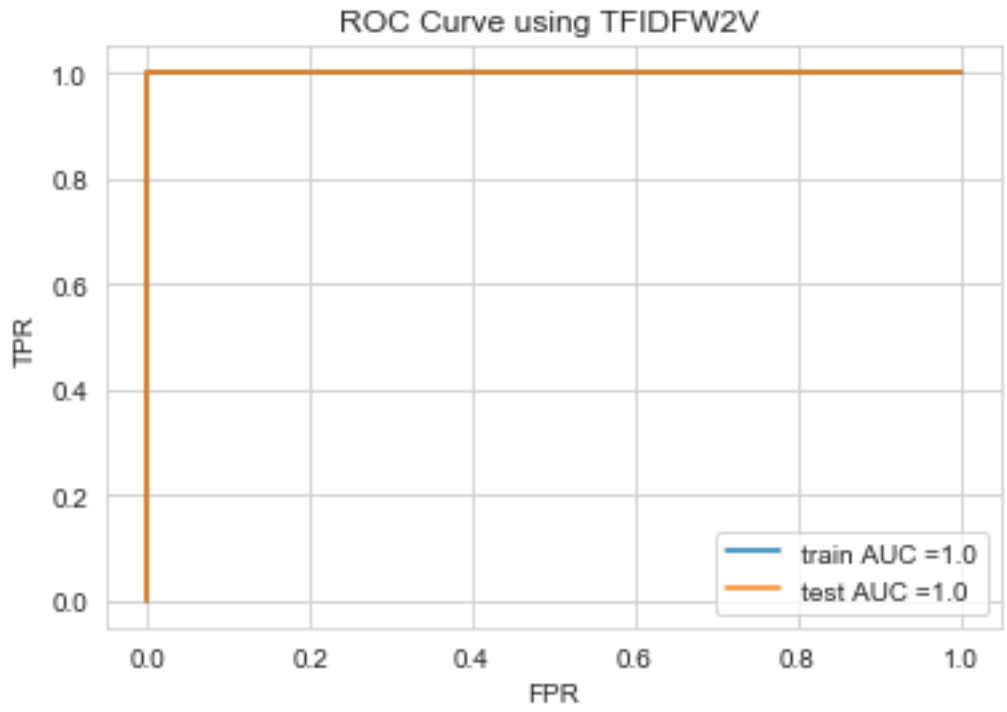
[380]:
```
Train_GB_data(train_tfidf_avg,y_train_mini,vectorizer="TFIDFW2V")
```

```
The best parameters for using this model is {'max_depth': 2, 'n_estimators': 10}
Best Estimator  GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=2,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=10,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
================================================================================
====================
3D visualisation

<IPython.core.display.HTML object>
```

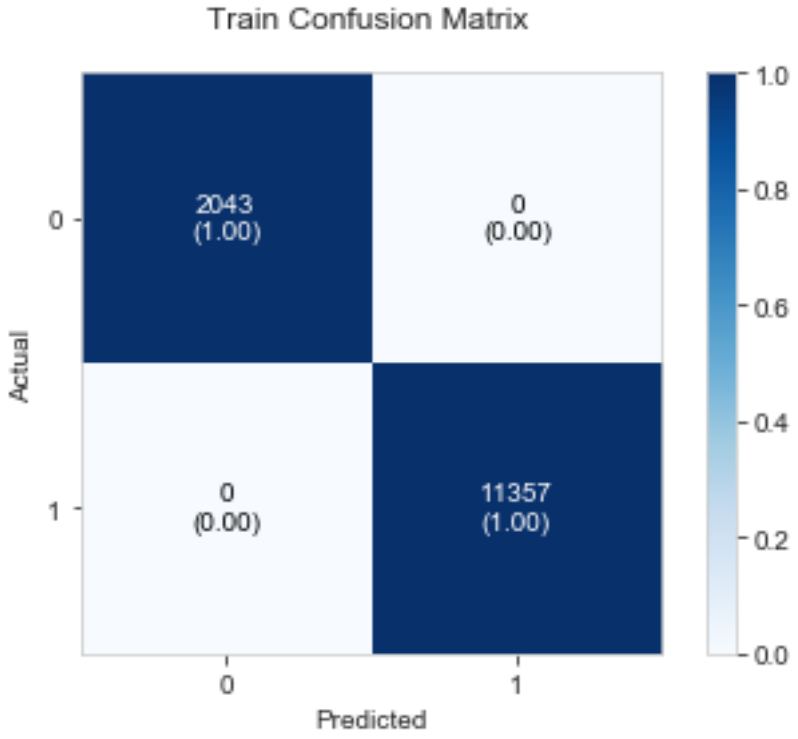
```
==========================================================================================
====================
HeatMap of Train and CV
```



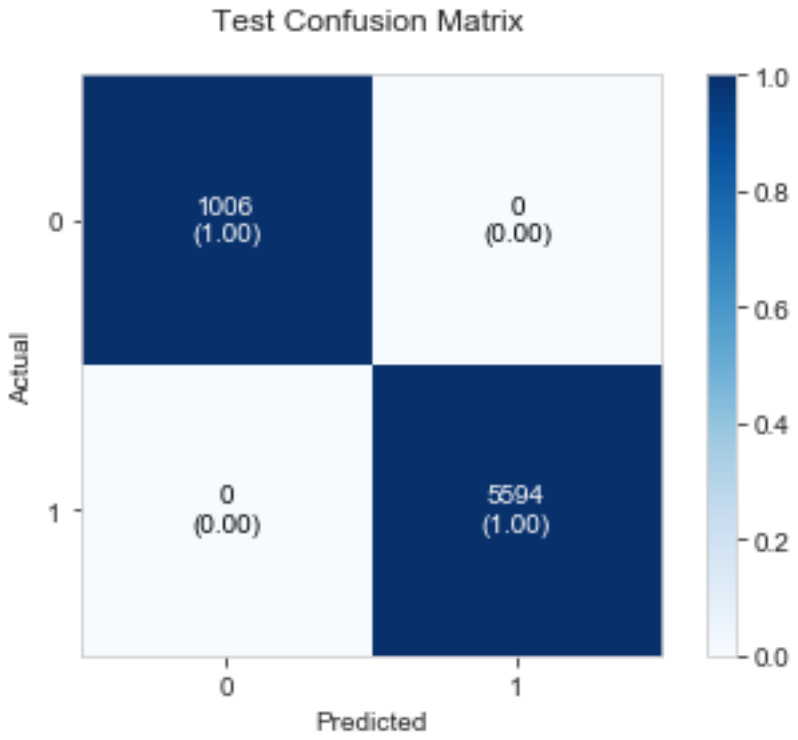

[392]:
```
Test_GB_Data(train_tfidf_avg,y_train_mini,test_tfidf_avg,y_test_mini,max_depth=2,␣
↪n_estimators=10,vectorizer="TFIDFW2V")
```



```
================================================================================
====================
Train confusion matrix
================================================================================
====================
the maximum value of tpr*(1-fpr) 1.0 for threshold 1.0
Train confusion matrix
[[ 2043     0]
 [    0 11357]]
```




```
Test confusion matrix
[[1006    0]
 [   0 5594]]
```




3. Conclusion

[393]:
```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
print(table)
```

```
+-----------+------------+-------------------------+----------------------+-----
+
| Algorithm | Vectorizer | Hyp.param1:n_estimators | Hyp.param2:max_depth | AUC
|
+-----------+------------+-------------------------+----------------------+-----
+
|     RF    |    BOW     |           50            |          2           | 1.0
|
|     RF    |   TFIDF    |           50            |          2           | 1.0
|
|     RF    |  AVG W2V   |           10            |          2           | 1.0
|
|     RF    |  TFIDFW2V  |           10            |          2           | 1.0
|
|    GBDT   |    BOW     |           10            |          2           | 1.0
|
|    GBDT   |   TFIDF    |           10            |          2           | 1.0
```

```
|
|    GBDT    |  AVG W2V   |            10            |          2           | 1.0
|
|    GBDT    |  TFIDFW2V  |            10            |          2           | 1.0
|
+-----------+------------+--------------------------+----------------------+-----
+
```

1. Compared to other classification algorithms like Linear Regression, the performance the ensemble models is the best
2. We can conclude that both GBDT and Random forest is clearly able to distinguish the words in the approved and rejected cases on the unseen data perfectly
3. As expected, the time taken to train is very very high compared to other models.Random forest takes lesser time to train compared to GBDT

4. Currently we are not using multi node cluster,while using multi node cluster like hadoop cluster, the training time will reduce significantly as we parallelize the training the of each boostrap model.
5. The learning point is although the performance is best, especially when we dont work in multi node cluster enviornment, we should try other classfier algorithm first dependening on the data, if data is linearly seperable we should try SVM or linear regression first, if the data is not linearly seperable we should try KNN first, and if the performance of the model is not upto the mark, then we should try with Ensemble models.
6. Overall TFIDFW2V takes very less time to train. It is better to use this for analysis of text data than other methods. TFIDF and BOW takes almost same time to train, but the training time is higher than TFIDFW2V.

7. AVGW2V based method takes very long time to train for all the algorithms we have tried until now, though we dont see any changes in the result.

[ ]: