Testing the code for co_occurence_matrix

In [0]:

```
test_corpus = [["abc def ijk pqr"],

      ["pqr klm opq"],

      ["lmn pqr xyz abc def pqr abc"]
    ]

test_top_words = [ "abc", "pqr", "def"]
test_window_size = 2     #window_size of 2
```

In [11]:

```
import pandas as pd
test_corpus = pd.DataFrame(test_corpus, columns=['testing'])
test_corpus.head()
```

Out[11]:

	testing
0	abc def ijk pqr
1	pqr klm opq
2	lmn pqr xyz abc def pqr abc

In [12]:

```
test_all_words = []
for i in test_corpus['testing']:
    words = i.split()
    test_all_words.extend(words)

print(test_all_words)
```

```
['abc', 'def', 'ijk', 'pqr', 'pqr', 'klm', 'opq', 'lmn', 'pqr', 'xyz', 'abc', 'def', 'pqr', 'abc']
```

In [0]:

```
from collections import defaultdict
def test_get_context_words(x,y,window_size):     #x-all_words, b-top_words, z-window_size
    total_words = len(x)
    context_words = defaultdict(list)

    for i in y:
        for index, j in enumerate(x):
            if i == j and index==0:
                context_words[i].extend(x[index:window_size+1])

            if i==j  and index==1:
                context_words[i].extend(x[index-1:window_size+index+1])

            if i==j and index>=2 and index<=total_words-2:
                context_words[i].extend(x[index-window_size:index+window_size+1])

            if i==j and index==total_words-1:
                context_words[i].extend(x[index-window_size:])

            if i==j and index==total_words:
                context_words[i].extend(x[index-window_size:])

    return context_words
```

In [14]:

```
test_context_words = test_get_context_words(test_all_words,test_top_words,test_window_size)
print(test_context_words)
```

```
defaultdict(<class 'list'>, {'abc': ['abc', 'def', 'ijk', 'pqr', 'xyz', 'abc', 'def', 'pqr', 'def'
, 'pqr', 'abc'], 'pqr': ['def', 'ijk', 'pqr', 'pqr', 'klm', 'ijk', 'pqr', 'pqr', 'klm', 'opq', 'op
q', 'lmn', 'pqr', 'xyz', 'abc', 'abc', 'def', 'pqr', 'abc'], 'def': ['abc', 'def', 'ijk', 'pqr', '
xyz', 'abc', 'def', 'pqr', 'abc']})
```

In [16]:

```
import pandas as pd
test_co_occurence_matrix = pd.DataFrame(columns=list(test_context_words.keys()),
index=list(test_context_words.keys())).fillna(0)
test_co_occurence_matrix.head()
```

Out[16]:

	abc	pqr	def
abc	0	0	0
pqr	0	0	0
def	0	0	0

In [0]:

```
from collections import Counter
for i in b:
    values = Counter(test_context_words[i])
    for j in b:
        if i == j:
            test_co_occurence_matrix[i][j]=0

        if i!=j:
            test_co_occurence_matrix[i][j] = values[j]
```

In [18]:

```
test_co_occurence_matrix
```

Out[18]:

	abc	pqr	def
abc	0	3	3
pqr	3	0	2
def	3	2	0

Note:

- I finally got the test_cooccurence matrix which gives the correct reslut

Truncated SVD

In [19]:

```
#importing libraries
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from tqdm import tqdm

from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objs as go
#configure_plotly_browser_state()
init_notebook_mode(connected=False)
```

In [20]:

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%
b&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly
ttps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [21]:

```
! ls '/content/drive/My Drive/Applied AI/Datasets/New Donors/'
```

```
co_occ_matrix.csv  PreProcessed.csv  resources.csv
glove_vectors    Preprocessed_inc_others.csv train_data.csv
```

In [0]:

```
data = pd.read_csv('/content/drive/My Drive/Applied AI/Datasets/New
Donors/Preprocessed_inc_others.csv')
#data = pd.read_csv('Preprocessed_inc_others.csv')
```

In [23]:

```
data.head()
```

Out[23]:

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	proje
0	0	ca	mrs	grades_prek_2	53	1
1	1	ut	ms	grades_3_5	4	1
2	2	ca	mrs	grades_prek_2	10	1

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	proje
3	3	ga	mrs	grades_prek_2	2	1
4	4	wa	mrs	grades_3_5	2	1

Note:

```
 - Sampling only 50k points
```

In [0]:

```
data = data.sample(50000)
```

In [25]:

```
data.shape
```

Out[25]:

```
(50000, 15)
```

In [26]:

```
data.head(3)
```

Out[26]:

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects
80441	80441	dc	ms	grades_3_5	0
70384	70384	mi	mrs	grades_prek_2	31
25543	25543	wa	mrs	grades_3_5	2

In [27]:

```
data.describe()
```

Out[27]:

	Unnamed: 0	teacher_number_of_previously_posted_projects	project_is_approved	price	quantity
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000
mean	54713.853660	11.303940	0.846640	297.702071	16.915540
std	31545.041742	28.231498	0.360338	365.474881	25.551058
min	3.000000	0.000000	0.000000	0.690000	1.000000
25%	27232.750000	0.000000	1.000000	104.457500	4.000000
50%	54814.000000	2.000000	1.000000	206.690000	9.000000
75%	82113.750000	9.000000	1.000000	378.722500	21.000000
max	109247.000000	451.000000	1.000000	9999.000000	800.000000

In [28]:

```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(2)
```

Out[28]:

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects
80441	80441	dc	ms	grades_3_5	0
70384	70384	mi	mrs	grades_prek_2	31

In [29]:

```
y = y.reshape(-1,1)
print(y.shape)
```

```
(50000, 1)
```

Splitting the Data

In [0]:

```
from sklearn.model_selection import  train_test_split
data_train, data_test, label_train, label_test = train_test_split(X,y, random_state=42, test_size=0
.3, stratify=y)
```

In [31]:

In [31]:

```
print(data_train.shape)
print(data_test.shape)
print(label_train.shape)
print(label_test.shape)
```

```
(35000, 14)
(15000, 14)
(35000, 1)
(15000, 1)
```

In [0]:

```
X_train = data_train
y_train = label_train
X_test = data_test
y_test = label_test
```

1. Vectorizing all features

1.1 School State

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_1 = CountVectorizer(list(X_train['school_state'].values), lowercase=False, binary=True)
```

In [0]:

```
X_train_Sstate = vectorizer_1.fit_transform(X_train['school_state'].values)
X_test_Sstate = vectorizer_1.transform(X_test['school_state'].values)
```

In [38]:

```
print(X_train_Sstate.shape)
print(X_test_Sstate.shape)
```

```
(35000, 51)
(15000, 51)
```

1.2 Clean Categories

In [0]:

```
vectorizer_2 = CountVectorizer(list(X_train['clean_categories'].values), lowercase=False,
binary=True)
```

In [0]:

```
X_train_cat = vectorizer_2.fit_transform(X_train['clean_categories'].values)
X_test_cat = vectorizer_2.transform(X_test['clean_categories'].values)
```

In [41]:

```
print(X_train_cat.shape)
print(X_test_cat.shape)
```

```
(35000, 9)
(15000, 9)
```

1.3 Clean Sub categories

In [0]:

```
vectorizer_3 = CountVectorizer(list(X_train['clean_subcategories'].values), lowercase=False,
binary=True)
```

In [0]:

```
X_train_subcat = vectorizer_3.fit_transform(X_train['clean_subcategories'].values)
X_test_subcat = vectorizer_3.transform(X_test['clean_subcategories'].values)
```

In [44]:

```
print(X_train_subcat.shape)
print(X_test_subcat.shape)
```

```
(35000, 30)
(15000, 30)
```

1.4 Project grade category

In [0]:

```
vectorizer_4 = CountVectorizer(list(X_train['project_grade_category'].values), lowercase=False,
binary=True)
```

In [0]:

```
X_train_grade = vectorizer_4.fit_transform(X_train['project_grade_category'].values)
X_test_grade = vectorizer_4.transform(X_test['project_grade_category'].values)
```

In [47]:

```
print(X_train_grade.shape)
print(X_test_grade.shape)
```

```
(35000, 4)
(15000, 4)
```

1.5 Teacher Prefix

In [0]:

```
vectorizer_5 = CountVectorizer(list(X_train['teacher_prefix'].values), lowercase=False,
binary=True)
```

In [0]:

```
X_train_prefix = vectorizer_5.fit_transform(X_train['teacher_prefix'].values)
X_test_prefix = vectorizer_5.transform(X_test['teacher_prefix'].values)
```

In [50]:

```
print(X_train_prefix.shape)
print(X_test_prefix.shape)
```

```
(35000, 5)
(15000, 5)
```

1.6 Price

1.6.1 Price Unstandardized

1.6.1 Price Unstandardized

In [0]:

```
X_train_price_unstandardized = X_train['price'].values.reshape(-1,1)
X_test_price_unstandardized = X_test['price'].values.reshape(-1,1)
```

In [52]:

```
print(X_train_price_unstandardized.shape)
print(X_test_price_unstandardized.shape)
```

```
(35000, 1)
(15000, 1)
```

1.6.2 Price Standardized

In [0]:

```
from sklearn.preprocessing import StandardScaler
sc_price = StandardScaler()
X_train_price = sc_price.fit_transform(X_train['price'].values.reshape(-1,1))
X_test_price = sc_price.transform(X_test['price'].values.reshape(-1,1))
```

In [54]:

```
print(X_train_price.shape)
print(X_test_price.shape)
```

```
(35000, 1)
(15000, 1)
```

1.7 Previously posted Projects

1.7.1 Unstandardized

In [0]:

```
X_train_previous_unstandardized = X_train['teacher_number_of_previously_posted_projects'].values.r
eshape(-1,1)
X_test_previous_unstandardized =
X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)
```

In [56]:

```
print(X_train_previous_unstandardized.shape)
print(X_test_previous_unstandardized.shape)
```

```
(35000, 1)
(15000, 1)
```

1.7.2 Standardized

In [0]:

```
from sklearn.preprocessing import StandardScaler
sc_previous = StandardScaler()
X_train_previous =
sc_previous.fit_transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-
1,1))
X_test_previous =
sc_previous.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
```

In [58]:

```
print(X_train_previous.shape)
print(X_test_previous.shape)
```

```
(35000, 1)
(15000, 1)
```

2. Building Co-occurence matrix

2.1 Step-1 : Combining "essay" and "title" text

In [0]:

```
concated_df = pd.DataFrame()
concated_df['merged_text'] = data_train['essay'].map(str) +\
                             data_train['title'].map(str)
```

In [60]:

```
concated_df.head()
```

Out[60]:

	merged_text
105914	i visual arts teacher kindergarten fifth grade...
61606	my students many countries many learning engli...
50475	randleman high school school heart small rural...
57271	in dual language classroom start day circle ti...
87981	at wms growing diverse school many exciting ch...

In [61]:

```
concated_df.shape
```

Out[61]:

```
(35000, 1)
```

2.2 Step -2 : Finding the "idf" values for each word in the combined text

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_6 = TfidfVectorizer(list(concated_df['merged_text'].values), min_df=10)
```

In [0]:

```
all_text = vectorizer_6.fit_transform(concated_df['merged_text'].values)
```

In [0]:

```
words = vectorizer_6.get_feature_names()
```

In [65]:

```
#list of words in combined text
words
```

Out[65]:

```
['00',
 '000',
 '04',
 '05',
 '10',
 '100',
 '1000',
 '100th',
 '101',
 '102',
 '103',
 '104',
 '105',
 '108',
 '10th',
 '11',
 '110',
 '1100',
 '112',
 '11th',
 '12',
 '120',
 '1200',
 '123',
 '125',
 '12th',
 '13',
 '130',
 '1300',
 '14',
 '140',
 '1400',
 '15',
 '150',
 '1500',
 '16',
 '160',
 '1600',
 '17',
 '170',
 '175',
 '18',
 '180',
 '1800',
 '19',
 '1950',
 '1st',
 '20',
 '200',
 '2000',
 '2003',
 '2004',
 '2005',
 '2006',
 '2007',
 '2008',
 '2009',
 '2010',
 '2011',
 '2012',
 '2013',
 '2014',
 '2015',
 '2016',
 '2017',
 '2018',
 '20th',
 '21',
 '21st',
 '22',
 '220',
 '23',
 '24',
 '240',
 '25',
 '250',
 '26',
 '27',
 '270',
 '28',
 '280',
 '29',
 '2d',
 '2nd',
 '30',
 '300',
 '3000',
 '31',
 '32',
 '320',
 '33',
 '34',
 '35',
 '350',
 '36',
 '360',
 '37',
 '38',
 '39',
 '3d',
 '3doodler',
 '3doodlers',
 '3rd',
 '40',
 '400',
 '41',
 '42',
 '43',
 '430',
 '44',
 '45',
 '450',
 '46',
 '47',
 '48',
 '480',
 '49',
 '4k',
 '4th',
 '50',
 '500',
 '504',
 '51',
 '52',
 '53',
 '54',
 '55',
 '550',
 '56',
 '560',
 '57',
 '58',
 '59',
 '5k',
 '5th',
 '60',
 '600',
 '61',
 '62',
 '63',
 '64',
 '65',
 '650',
 '66',
 '67',
 '68',
 '69',
 '6th',
 '70',
 '700',
 '71',
 '72',
 '73',
 '74',
 '75',
 '750',
 '76',
 '77',
 '78',
 '79',
 '7th',
 '80',
 '800',
 '81',
 '82',
 '83',
 '84',
 '85',
 '850',
 '86',
 '87',
 '88',
 '89',
 '8th',
 '90',
 '900',
 '91',
 '92',
 '93',
 '94',
 '95',
 '950',
 '96',
 '97',
 '98',
 '99',
 '9th',
 'abandoned',
 'abc',
 'abcmouse',
 'abcs',
 'abcya',
 'abdominal',
 'abilities',
 'ability',
 'able',
 'abled',
 'aboard',
 'abound',
 'abounds',
 'about',
 'above',
 'abraham',
 'abroad',
 'absence',
 'absences',
 'absent',
 'absenteeism',
 'absolute',
 'absolutely',
 'absorb',
 'absorbed',
 'absorbing',
 'abstract',
 'abstractly',
 'abundance',
 'abundant',
 'abuse',
 'abused',
 'academia',
 'academic',
 'academically',
 'academics',
 'academies',
 'academy',
 'accelerate',
 'accelerated',
 'accelerating',
 'acceleration',
 'accent',
 'accept',
 'acceptable',
 'acceptance',
 'accepted',
 'accepting',
 'accepts',
 'access',
 'accessed',
 'accessibility',
 'accessible',
 'accessing',
 'accessories',
 'accessory',
 'accident',
 'accidentally',
 'accidents',
 'acclimate',
 'accolades',
 'accommodate',
 'accommodated',
 'accommodates',
 'accommodating',
 'accommodation',
 'accommodations',
 'accompanied',
 'accompanies',
 'accompaniment',
 'accompany',
 'accompanying',
 'accomplish',
 'accomplished',
 'accomplishing',
 'accomplishment',
 'accomplishments',
 'according',
 'accordingly',
 'account',
 'accountability',
 'accountable',
 'accounts',
 'accreditation',
 'accredited',
 'accumulate',
 'accumulated',
 'accuracy',
 'accurate',
 'accurately',
 'accustomed',
 'ace',
 'acer',
 'aches',
 'achievable',
 'achieve',
 'achieved',
 'achievement',
 'achievements',
 'achiever',
 'achievers',
 'achieves',
 'achieving',
 'acknowledge',
 'acknowledged',
 'acknowledging',
 'acquire',
 'acquired',
 'acquiring',
 'acquisition',
 'acres',
 'acronym',
 'across',
 'acrylic',
 'act',
 'acting',
 'action',
 'actions',
 'activate',
 'activated',
 'activates',
 'activating',
 'active',
 'actively',
 'activism',
 'activist',
 'activists',
 'activites',
 'activities',
 'activity',
 'actor',
 'actors',
 'acts',
 'actual',
 'actually',
 'ad',
 'adams',
 'adapt',
 'adaptability',
 'adaptable',
 'adaptation',
 'adaptations',
 'adapted',
 'adapter',
 'adapting',
 'adaptive',
 'add',
 'added',
 'addicted',
 'addiction',
 'adding',
 'addition',
 'additional',
 'additionally',
 'additions',
 'address',
 'addressed',
 'addresses',
 'addressing',
 'adds',
 'adept',
 'adequate',
 'adequately',
 'adhd',
 'adhere',
 'adhesive',
 'adjacent',
 'adjectives',
 'adjust',
 'adjustable',
 'adjusted',
 'adjusting',
 'adjustment',
 'adjustments',
 'administered',
 'administration',
 'administrative',
 'administrator',
 'administrators',
 'admirable',
 'admire',
 'admission',
 'admit',
 'admitted',
 'adobe',
 'adolescence',
 'adolescent',
 'adolescents',
 'adopt',
 'adopted',
 'adopting',
 'adoption',
 'adorable',
 'adore',
 'adult',
 'adulthood',
 'adults',
 'advance',
 'advanced',
 'advancement',
 'advancements',
 'advances',
 'advancing',
 'advantage',
 'advantaged',
 'advantages',
 'adventure',
 'adventurers',
 'adventures',
 'adventurous',
 'adverse',
 'adversities',
 'adversity',
 'advertisements',
 'advertising',
 'advice',
 'advisory',
 'advocacy',
 'advocate',
 'advocates',
 'advocating',
 'aerobic',
 'aerospace',
 'aesthetic',
 'aesthetically',
 'aesthetics',
 'affect',
 'affected',
 'affecting',
 'affection',
 'affective',
 'affects',
 'affiliated',
 'affirmation',
 'affluent',
 'afford',
 'affordable',
 'afforded',
 'affording',
 'affords',
 'afghanistan',
 'afloat',
 'aforementioned',
 'afraid',
 'africa',
 'african',
 'after',
 'afternoon',
 'afternoons',
 'afterschool',
 'afterward',
 'afterwards',
 'again',
 'against',
 'age',
 'aged',
 'agencies',
 'agency',
 'agenda',
 'agents',
 'ages',
 'aggression',
 'aggressive',
 'agility',
 'aging',
 'ago',
 'agree',
 'agreed',
 'agreement',
 'agricultural',
 'agriculture',
 'ah',
 'aha',
 'ahead',
 'aid',
 'aide',
 'aided',
 'aides',
 'aiding',
 'aids',
 'aig',
 'aim',
 'aimed',
 'aiming',
 'aims',
 'air',
 'airplane',
 'airplanes',
 'airport',
 'aka',
 'al',
 'alabama',
 'alarm',
 'alaska',
 'alaskan',
 'albany',
 'albert',
 'alcohol',
 'alert',
 'alertness',
 'alexa',
 'alexander',
 'algebra',
 'algebraic',
 'algorithms',
 'align',
 'aligned',
 'aligns',
 'alike',
 'alive',
 'all',
 'allergies',
 'allergy',
 'alleviate',
 'allocated',
 'allotted',
 'allow',
 'allowance',
 'allowed',
 'allowing',
 'allows',
 'almost',
 'aloha',
 'alone',
 'along',
 'alongs',
 'alongside',
 'alot',
 'aloud',
 'alouds',
 'alphabet',
 'already',
 'also',
 'alter',
 'altered',
 'altering',
 'alternate',
 'alternative',
 'alternatives',
 'although',
 'alto',
 'altogether',
 'alumni',
 'always',
 'am',
 'amaze',
 'amazed',
 'amazement',
 'amazes',
 'amazing',
 'amazingly',
 'amazon',
 'ambassadors',
 'ambition',
 'ambitions',
 'ambitious',
 'america',
 'american',
 'americans',
 'amharic',
 'amidst',
 'among',
 'amongst',
 'amount',
 'amounts',
 'ample',
 'amplification',
 'amplifier',
 'amplify',
 'amusement',
 'an',
 'analysis',
 'analytical',
 'analyze',
 'analyzed',
 'analyzing',
 'anatomy',
 'ancestors',
 'ancestry',
 'anchor',
 'anchorage',
 'anchors',
 'ancient',
 'and',
 'anderson',
 'android',
 'angeles',
 'angelou',
 'anger',
 'angle',
 'angles',
 'angry',
 'animal',
 'animals',
 'animate',
 'animated',
 'animation',
 'animations',
 'anne',
 'annie',
 'anniversary',
 'annotate',
 'annotating',
 'annotation',
 'annotations',
 'announce',
 'announced',
 'announcement',
 'announcements',
 'annual',
 'annually',
 'anonymous',
 'another',
 'answer',
 'answered',
 'answering',
 'answers',
 'ant',
 'anti',
 'anticipate',
 'anticipated',
 'anticipating',
 'anticipation',
 'antiquated',
 'antonio',
 'antonyms',
 'ants',
 'antsy',
 'anxieties',
 'anxieties',
 'anxiety',
 'anxious',
 'anxiously',
 'any',
 'anybody',
 'anymore',
 'anyone',
 'anything',
 'anytime',
 'anyway',
 'anywhere',
 'ap',
 'apart',
 'apartment',
 'apartments',
 'apathetic',
 'app',
 'appalachia',
 'appalachian',
 'apparatus',
 'apparent',
 'appeal',
 'appealing',
 'appeals',
 'appear',
 'appearance',
 'appears',
 'appetite',
 'appetites',
 'apple',
 'apples',
 'applesauce',
 'appliances',
 'applicable',
 'application',
 'applications',
 'applied',
 'applies',
 'apply',
 'applying',
 'appreciate',
 'appreciated',
 'appreciates',
 'appreciating',
 'appreciation',
 'appreciative',
 'apprehensive',
 'approach',
 'approached',
 'approaches',
 'approaching',
 'appropriate',
 'appropriately',
 'approval',
 'approved',
 'approx',
 'approximately',
 'apps',
 'april',
 'apron',
 'aprons',
 'apt',
 'aptitude',
 'aquaponics',
 'aquarium',
 'aquatic',
 'ar',
 'arabic',
 'archery',
 'architect',
 'architects',
 'architectural',
 'architecture',
 'arctic',
 'arduino',
 'are',
 'area',
 'area',
 'areal',
 'areas',
 'arena',
 'argue',
 'arguing',
 'argument',
 'argumentative',
 'arguments',
 'arise',
 'arises',
 'aristotle',
 'arithmetic',
 'arizona',
 'arkansas',
 'arm',
 'armed',
 'arms',
 'army',
 'around',
 'arrange',
 'arranged',
 'arrangement',
 'arrangements',
 'arranging',
 'array',
 'arrays',
 'arrival',
 'arrivals',
 'arrive',
 'arrived',
 'arrives',
 'arriving',
 'arrows',
 'arsenal',
 'art',
 'arthur',
 'article',
 'articles',
 'articulate',
 'articulation',
 'artifacts',
 'artificial',
 'artist',
 'artistic',
 'artistically',
 'artists',
 'arts',
 'artsy',
 'artwork',
 'artworks',
 'as',
 'asd',
 'asia',
 'asian',
 'asians',
 'aside',
 'ask',
 'asked',
 'asking',
 'asks',
 'asl',
 'asleep',
 'aspect',
 'aspects',
 'asperger',
 'aspirations',
 'aspire',
 'aspiring',
 'assemble',
 'assemblies',
 'assembly',
 'assess',
 'assessed',
 'assesses',
 'assessing',
 'assessment',
 'assessments',
 'assessments',
 'asset',
 'assets',
 'assign',
 'assigned',
 'assigning',
 'assignment',
 'assignments',
 'assimilate',
 'assist',
 'assistance',
 'assistant',
 'assistants',
 'assisted',
 'assisting',
 'assistive',
 'assists',
 'associate',
 'associated',
 'association',
 'assorted',
 'assortment',
 'assume',
 'assure',
 'assured',
 'asthma',
 'astound',
 'astounding',
 'astronaut',
 'astronauts',
 'astronomy',
 'asus',
 'at',
 'ate',
 'athlete',
 'athletes',
 'athletic',
 'athletics',
 'atlanta',
 'atlantic',
 'atlases',
 'atmosphere',
 'atoms',
 'atpe',
 'attach',
 'attached',
 'attachment',
 'attachments',
 'attack',
 'attain',
 'attainable',
 'attained',
 'attaining',
 'attempt',
 'attempted',
 'attempting',
 'attempts',
 'attend',
 'attendance',
 'attended',
 'attending',
 'attends',
 'attention',
 'attentive',
 'attentiveness',
 'attire',
 'attitude',
 'attitudes',
 'attract',
 'attracted',
 'attractive',
 'attracts',
 'attribute',
 'attributed',
 'attributes',
 'atypical',
 'audible',
 'audience',
 'audience',
 'audiences',
 'audio',
 'audiobook',
 'audiobooks',
 'audition',
 'auditorium',
 'auditory',
 'augment',
 'augmentative',
 'augmented',
 'august',
 'aunt',
 'aunts',
 'aural',
 'austin',
 'authentic',
 'authentically',
 'authenticity',
 'author',
 'authority',
 'authors',
 'autism',
 'autistic',
 'automatic',
 'automatically',
 'automaticity',
 'autonomous',
 'autonomy',
 'availability',
 'available',
 'avenue',
 'avenues',
 'average',
 'averages',
 'avid',
 'avoid',
 'avoiding',
 'await',
 'awaiting',
 'awaits',
 'awake',
 'awaken',
 'award',
 'awarded',
 'awards',
 'aware',
 'awareness',
 'away',
 'awe',
 'awesome',
 'awesomeness',
 'awful',
 'awhile',
 'awkward',
 'az',
 'babies',
 'baby',
 'babysitting',
 'baccalaureate',
 'back',
 'backbone',
 'backdrop',
 'backdrops',
 'backed',
 'background',
 'backgrounds',
 'backpack',
 'backpacks',
 'backpatter',
 'backs',
 'backwards',
 'backyard',
 'bacteria',
 'bad',
 'badges',
 'badly',
 'badminton'
```

```
 'badminton',
 'bag',
 'baggage',
 'baggies',
 'bags',
 'bake',
 'baker',
 'baking',
 'balance',
 'balanced',
 'balances',
 'balancing',
 'ball',
 'ballet',
 'balloon',
 'balloons',
 'balls',
 'balm',
 'baltimore',
 'banana',
 'bananas',
 'band',
 'bands',
 'bang',
 'bangladesh',
 'bank',
 'banking',
 'banks',
 'bar',
 'bare',
 'barely',
 'baritone',
 'barn',
 'barred',
 'barrier',
 'barriers',
 'bars',
 'basal',
 'base',
 'baseball',
 'baseballs',
 'based',
 'baseline',
 'bases',
 'basic',
 'basically',
 'basics',
 'basis',
 'basket',
 'basketball',
 'basketballs',
 'baskets',
 'bass',
 'bat',
 'batch',
 'bath',
 'bathroom',
 'batman',
 'baton',
 'bats',
 'batteries',
 'battery',
 'batting',
 'battle',
 'battles',
 'battling',
 'bay',
 'be',
 'beach',
 'beaches',
 'beacon',
 'bead',
 'beads',
 'beakers',
 'beam',
 'beaming',
 'beams',
...]
```

In [66]:

```
len(words)
```

Out[66]:

```
11385
```

In [67]:

```
len(vectorizer_6.idf_)
```

Out[67]:

```
11385
```

In [68]:

```
#getting the idf_values for each word in the combined text
idf_values = vectorizer_6.idf_
len(idf_values)
```

Out[68]:

```
11385
```

In [0]:

```
#Storing it in new dataframe so that we can sort it
new_df = pd.DataFrame()
new_df['words'] = words
new_df['idf'] = idf_values
```

In [70]:

```
new_df
```

Out[70]:

	words	idf
0	00	7.385594
1	000	5.861013
2	04	8.629919
3	05	8.467400
4	10	4.467366
...	...	...
11380	zones	7.571312
11381	zoo	7.492840
11382	zoom	8.372089
11383	zoos	8.898183
11384	zumba	8.824075

11385 rows × 2 columns

2.3 Step -3: Sort the idf values in descending order

In [0]:

```
new_df.sort_values(by='idf', ascending=False, inplace=True)
new_df.sort_values(by='idf', ascending=False, inplace=True)
```

In [72]:

```
new_df
```

Out[72]:

	words	idf
2937	disrupts	9.065237
8967	sanitize	9.065237
5328	jacqueline	9.065237
5319	iv	9.065237
7332	overabundance	9.065237
...	...	...
1840	classroom	1.374078
5615	learning	1.338583
6381	my	1.247904
9028	school	1.156716
9889	students	1.006334

11385 rows × 2 columns

2.4 Step -4: Select top 2000 words which has high "idf" values

In [0]:

```
new_df = new_df[0:2000]
```

In [74]:

```
new_df.head()
```

Out[74]:

	words	idf
2937	disrupts	9.065237
8967	sanitize	9.065237
5328	jacqueline	9.065237
5319	iv	9.065237
7332	overabundance	9.065237

In [75]:

```
new_df.tail()
```

Out[75]:

	words	idf
3234	edward	8.690543
3236	effected	8.690543
6404	nannan3doodler	8.690543
6405	nannan3rd	8.690543

	words	idf
6281	montana	8.690546

2.5 Step -5: Building a Co-occurence matrix

In [0]:

```
top_words = []
for word in new_df['words']:
    top_words.append(word)
```

In [77]:

```
len(top_words)
```

Out[77]:

```
2000
```

In [78]:

```
concated_df['merged_text'].head()
```

Out[78]:

```
105914    i visual arts teacher kindergarten fifth grade...
61606     my students many countries many learning engli...
50475     randleman high school school heart small rural...
57271     in dual language classroom start day circle ti...
87981     at wms growing diverse school many exciting ch...
Name: merged_text, dtype: object
```

In [79]:

```
all_corpus_words = []

for row in concated_df['merged_text']:
    split_sent = row.split()
    all_corpus_words.extend(split_sent)

print(len(all_corpus_words))
```

```
5411787
```

In [0]:

```
# Co-occurence matrix
#https://stackoverflow.com/questions/35562789/how-do-i-calculate-a-word-word-co-occurrence-matrix-
with-sklearn
'''def co_occur_mat(input_text,top_words,window_size):
    co_occur = pd.DataFrame(index=top_words, columns=top_words)

    for row,nrow in zip(top_words,range(len(top_words))):
        for colm,ncolm in zip(top_words,range(len(top_words))):
            count = 0
            if row == colm:
                co_occur.iloc[nrow,ncolm] = count
            else:
                for single_essay in input_text:
                    essay_split = single_essay.split()
                    max_len = len(essay_split)
                    top_word_index = [index for index, split in enumerate(essay_split) if row in sp
lit]

                    for index in top_word_index:
                        if index == 0:
# if the top word is in 0th index of corpus in a sentence
                            count = count + essay_split[:window_size + 1].count(colm)
                        elif index == (max_len -1):
# if the top word is in last index of corpus in a sentence
```

```
                            count = count + essay_split[-(window_size + 1):].count(colm)
                        else:
                            count = count + essay_split[index + 1 : (index + window_size +
1)].count(colm)  # if the top word is in the middle of the corpus on right side
                            if index < window_size:
# if the top word in middle but less than the window size
                                count = count + essay_split[: index].count(colm)
                            else:
                                count = count + essay_split[(index - window_size): index].count(co
)       # if the top word in middle of the corpus on left side
                co_occur.iloc[nrow,ncolm] = count

    return co_occur

'''
```

In [0]:

```
from collections import defaultdict
def get_context_words(x,y,window_size):     #x-all_words, b-top_words, z-window_size
    total_words = len(x)
    context_words = defaultdict(list)

    for i in y:
        for index, j in enumerate(x):
            if i == j and index==0:
                context_words[i].extend(x[index:window_size+1])

            if i==j  and index==1:
                context_words[i].extend(x[index-1:window_size+index+1])

            if i==j and index>=2 and index<=total_words-2:
                context_words[i].extend(x[index-window_size:index+window_size+1])

            if i==j and index==total_words-1:
                context_words[i].extend(x[index-window_size:])

            if i==j and index==total_words:
                context_words[i].extend(x[index-window_size:])

    return context_words
```

In [0]:

```
window_size = 5
context_words = get_context_words(all_corpus_words,top_words,window_size)
```

In [85]:

```
import pandas as pd
co_occurence_matrix = pd.DataFrame(columns=list(context_words.keys()), index=list(context_words.ke
ys())).fillna(0)
co_occurence_matrix.head()
```

Out[85]:

	disrupts	sanitize	jacqueline	iv	overabundance	editors	satiate	satellite	sam	jane	eip	ipevo	waldo	ov
disrupts	0	0	0	0	0	0	0	0	0	0	0	0	0	0
sanitize	0	0	0	0	0	0	0	0	0	0	0	0	0	0
jacqueline	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iv	0	0	0	0	0	0	0	0	0	0	0	0	0	0
overabundance	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 rows × 2000 columns

In [0]:

```
from collections import Counter
```

```
for i in top_words:
    values = Counter(context_words[i])
    for j in top_words:
        if i == j:
            co_occurence_matrix[i][j]=0

        if i!=j:
            co_occurence_matrix[i][j] = values[j]
```

In [89]:

```
co_occurence_matrix.head()
```

Out[89]:

	disrupts	sanitize	jacqueline	iv	overabundance	editors	satiate	satellite	sam	jane	eip	ipevo	waldo	ov
disrupts	0	0	0	0	0	0	0	0	0	0	0	0	0	0
sanitize	0	0	0	0	0	0	0	0	0	0	0	0	0	0
jacqueline	0	0	0	0	0	0	0	0	0	0	0	0	0	0
iv	0	0	0	0	0	0	0	0	0	0	0	0	0	0
overabundance	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 rows × 2000 columns

In [0]:

```
#https://stackoverflow.com/questions/53898836/export-dataframe-as-csv-file-from-google-colab-to-go
ogle-drive
#from google.colab import drive
#drive.mount('/content/drive')
co_occurence_matrix.to_csv('co_occurence_matrix.csv')
!cp co_occurence_matrix.csv "/content/drive/My Drive/Applied AI/Datasets/New Donors/"
```

In [0]:

```
#co_occ_matrix.to_csv('co_occ_matrix.csv')
```

2.6 Truncated SVD

In [0]:

```
from sklearn.decomposition import TruncatedSVD
```

In [0]:

```
exp_var = []
n_comp = [i for i in range(1, 2000, 100)]
for i in range(1, 2000, 100):
    svd = TruncatedSVD(n_components=i)
    svd.fit(co_occurence_matrix)
    exp_var.append(svd.explained_variance_ratio_.sum())
```

In [93]:

```
exp_var
```

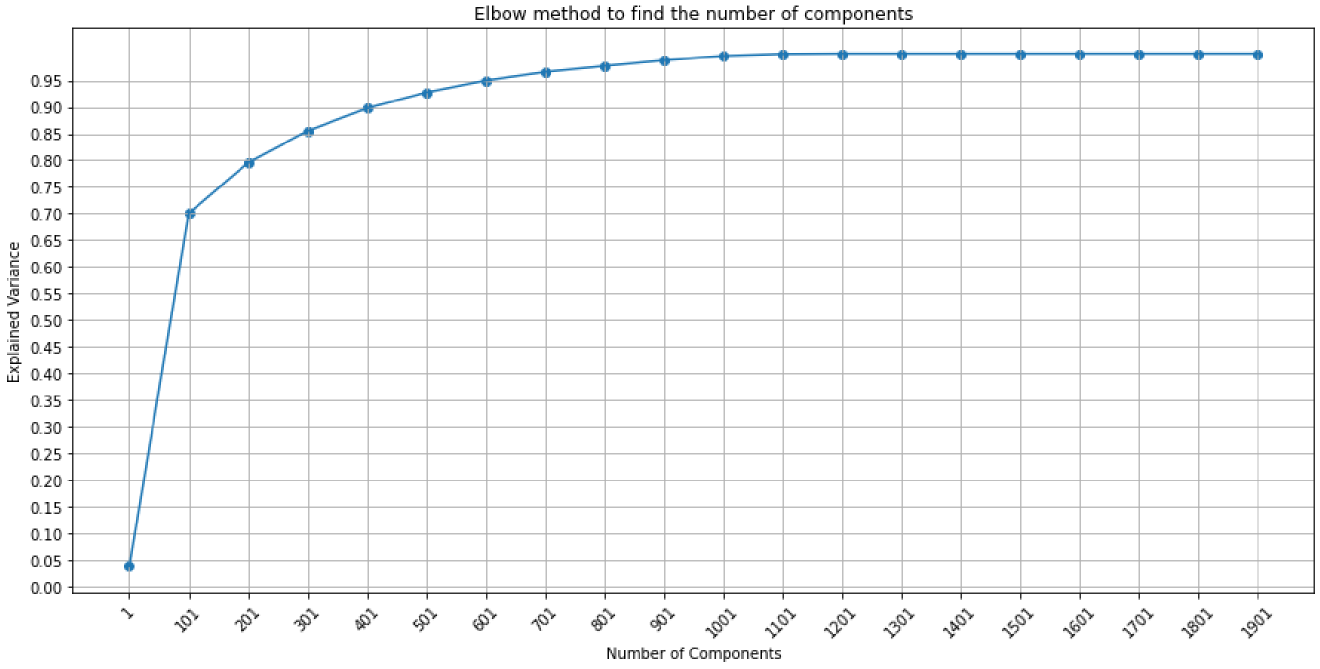
Out[93]:

```
[0.03804902072076934,
 0.7002263977741602,
 0.7953489820081505,
 0.8549064131705116,
 0.8976212227786746,
 0.9271639692655119,
 0.9495318922894621,
 0.9657459146574672,
 0.976960444202126,
 0.9876534130879453,
 0.995720954409133,
 0.9993343746122196,
 0.9999999999999982,
 0.9999999999999976,
 0.9999999999999973,
 1.0000000000000044,
 0.9999999999999971,
 1.0000000000000029,
 1.000000000000002,
 0.999999999999998]
```

In [95]:

```
plt.figure(figsize=(15,7))
plt.plot(n_comp, exp_var)
plt.scatter(n_comp, exp_var)

plt.xlabel('Number of Components')
plt.ylabel('Explained Variance')
plt.title('Elbow method to find the number of components')
plt.xticks(ticks=n_comp, rotation=45)
plt.yticks(list(np.arange(0.0, 1.0, 0.05)))      #https://stackoverflow.com/questions/477486/how-to-
use-a-decimal-range-step-value
plt.grid()
plt.show()
```



Note:

- It shows that the 601 components out fo 2000 explains 95% of variance and after that there is no improvement in explaining the variance. So i am going to take number of components = 600

In [96]:

```
# Truncating into the n_components which complains 95% variance
from sklearn.decomposition import TruncatedSVD
svd_1 = TruncatedSVD(n_components = 600)
truncated_co_occ_matrix = svd_1.fit_transform(co_occurence_matrix)
print(truncated_co_occ_matrix.shape)
```

```
(2000, 600)
```

In [0]:

```
#this co_occ_words contains all the top 2000 words
co_occ_words = list(co_occurence_matrix.columns)
```

2.7 Vectorizing text feature

2.7.1 Essay - AVGW2V

In [0]:

```
from tqdm import tqdm
```

In [100]:

```
truncated_co_occ_matrix = pd.DataFrame(truncated_co_occ_matrix, index=co_occurence_matrix.columns)
truncated_co_occ_matrix.head(3)
```

Out[100]:

	0	1	2	3	4	5	6	7	8	
disrupts	8.179514e-14	7.163242e-14	5.901680e-14	-1.332773e-14	-6.418767e-13	-6.808670e-13	-4.554471e-13	-1.706521e-12	3.651698e-13	-4.47 13
sanitize	-1.035703e-14	6.186251e-15	7.262355e-15	6.295261e-14	-4.905010e-14	-3.636866e-14	1.238988e-13	-3.232490e-14	-3.547028e-14	-1.89 13
jacqueline	2.980305e-14	3.964202e-14	2.332759e-14	7.444913e-12	-7.587767e-14	-1.354659e-13	-1.635824e-14	3.590669e-06	1.289133e-06	9.46 12

3 rows × 600 columns

In [102]:

```
truncated_co_occ_matrix.loc['sanitize'].shape
```

Out[102]:

```
(600,)
```

In [0]:

```
zzz = np.zeros(600)
xxx = truncated_co_occ_matrix.loc['sanitize']
```

In [0]:

```
zzz += xxx
```

In [105]:

```
print(zzz)
print(zzz.shape)
```

```
0     -1.035703e-14
1      6.186251e-15
2      7.262355e-15
3      6.295261e-14
4     -4.905010e-14
           ...
595   -1.076590e-04
596   -4.242684e-04
597   -7.115555e-04
```

```
598    3.676548e-04
599    2.404527e-04
Name: sanitize, Length: 600, dtype: float64
(600,)
```

In [106]:

```
X_train_essay_avg_w2v = []

for i in tqdm(X_train['essay']):
    vector = np.zeros(600)
    cnt_words = 0

    for word in i.split():
        if word in co_occ_words:
            vector += truncated_co_occ_matrix.loc[word]
            cnt_words += 1

        if cnt_words !=0:
            vector = vector/cnt_words

    X_train_essay_avg_w2v.append(vector)
```

```
100%|██████████| 35000/35000 [08:03<00:00, 72.39it/s]
```

In [107]:

```
X_test_essay_avg_w2v = []

for i in tqdm(X_test['essay']):
    vector = np.zeros(600)
    cnt_words = 0

    for word in i.split():
        if word in co_occ_words:
            vector += truncated_co_occ_matrix.loc[word]
            cnt_words += 1

        if cnt_words != 0:
            vector = vector/cnt_words

    X_test_essay_avg_w2v.append(vector)
```

```
100%|██████████| 15000/15000 [03:24<00:00, 73.21it/s]
```

In [108]:

```
print(len(X_train_essay_avg_w2v))
print(len(X_train_essay_avg_w2v[0]))
print(len(X_test_essay_avg_w2v))
print(len(X_test_essay_avg_w2v[0]))
```

```
35000
600
15000
600
```

2.7.2 TITLE AVGW2V

In [109]:

```
X_train_title_avg_w2v = []
X_test_title_avg_w2v = []

for i in tqdm(X_train['title']):
    vector = np.zeros(600)
    cnt_words = 0

    for word in i.split():
        if word in co_occ_words:
            vector += truncated_co_occ_matrix.loc[word]
            cnt_words += 1

        if cnt_words != 0:
            vector = vector/cnt_words

    X_train_title_avg_w2v.append(vector)


for i in tqdm(X_test['title']):
    vector = np.zeros(600)
    cnt_words = 0

    for word in i.split():
        if word in co_occ_words:
            vector +=  truncated_co_occ_matrix.loc[word]
            cnt_words += 1

        if cnt_words != 0:
            vector = vector/cnt_words

    X_test_title_avg_w2v.append(vector)
```

```
100%|██████████| 35000/35000 [00:07<00:00, 4976.87it/s]
100%|██████████| 15000/15000 [00:03<00:00, 4851.46it/s]
```

In [110]:

```
print(len(X_train_title_avg_w2v))
print(len(X_train_title_avg_w2v[0]))
print(len(X_test_title_avg_w2v))
print(len(X_test_title_avg_w2v[0]))
```

```
35000
600
15000
600
```

3. Model - LightGBM

3.1 Merging all features

In [0]:

```
from scipy.sparse import hstack
X_train_1 = hstack((X_train_Sstate, X_train_cat, X_train_subcat, X_train_grade, X_train_prefix, X_t
rain_essay_avg_w2v, X_train_title_avg_w2v,
                    X_train_previous, X_train_price)).tocsr()

X_test_1 = hstack((X_test_Sstate, X_test_cat, X_test_subcat, X_test_grade, X_test_prefix, X_test_es
say_avg_w2v, X_test_title_avg_w2v,
                   X_test_previous, X_test_price)).tocsr()
```

In [112]:

```
print(X_train_1.shape)
print(X_test_1.shape)
```

```
(35000, 1301)
(15000, 1301)
```

3.2 Grid Search

In [0]:

```
from lightgbm import LGBMClassifier
classifier_1 = LGBMClassifier()
```

In [0]:

```
from sklearn.model_selection import GridSearchCV
parameters = [
                {
                    'n_estimators' : [10,50,100,150,200,300,500,1000],
                    'max_depth' : [2,3,4,5,6,8,10]
                }
            ]
gridsearch_1 = GridSearchCV(estimator=classifier_1, param_grid=parameters, scoring='roc_auc', cv=2
, n_jobs=-1, return_train_score=True)
```

3.3 HyperParameter vs AUC

In [0]:

```
gridsearch_1 = gridsearch_1.fit(X_train_1, y_train)
```

In [116]:

```
results_1 = pd.DataFrame.from_dict(gridsearch_1.cv_results_)
results_1 = results_1.sort_values(['param_n_estimators'])
results_1.head()
```

Out[116]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_estimators	params
0	2.527342	0.013120	0.041839	0.000061	2	10	{'max_depth': 2, 'n_estimators': 10}
32	5.259082	0.143404	0.045065	0.001161	6	10	{'max_depth': 6, 'n_estimators': 10}
16	3.581144	0.124354	0.048907	0.000652	4	10	{'max_depth': 4, 'n_estimators': 10}
40	5.776280	0.140621	0.046433	0.000448	8	10	{'max_depth': 8, 'n_estimators': 10}
8	2.960429	0.097924	0.041568	0.001110	3	10	{'max_depth': 3, 'n_estimators': 10}

In [0]:

```
train_auc_1 = results_1['mean_train_score']
test_auc_1 = results_1['mean_test_score']
n_estimators_1 = results_1['param_n_estimators']
max_depth_1 = results_1['param_max_depth']
```

In [148]:

```
#x_1 = np.array(n_estimators_1)
#y_1 = np.array(max_depth_1)
#z_1 = np.array(train_auc_1)
#z_2= np.array(test_auc_1)
```

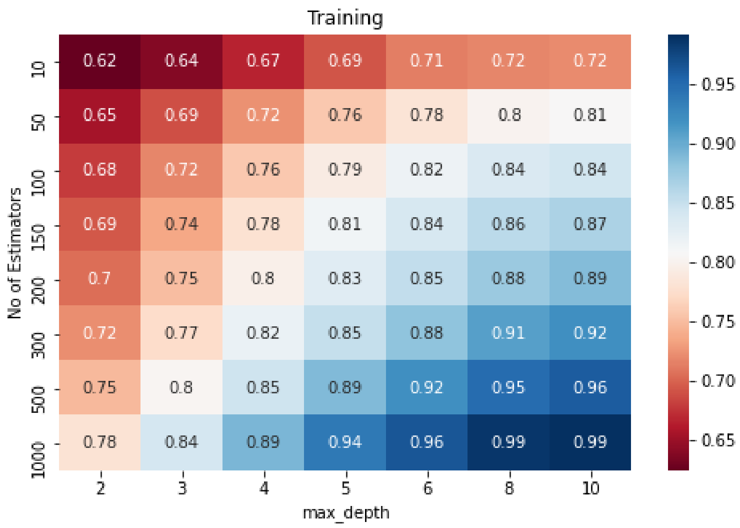
```
df_1 = pd.DataFrame.from_dict(np.array([n_estimators_1, max_depth_1, train_auc_1, test_auc_1]).T)
df_1.columns = ['No of Estimators', 'max_depth', 'train_auc', 'test_auc']
df_1['train_auc'] = pd.to_numeric(df_1['train_auc'])
df_1['test_auc'] = pd.to_numeric(df_1['test_auc'])

pivotted_1 = df_1.pivot_table(index='No of Estimators', columns='max_depth', values='train_auc', ag
gfunc='mean')

plt.figure(figsize=(8,5))
plt.title('Training')
sns.heatmap(pivotted_3, annot=True, cmap='RdBu')
```

Out[148]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f450b8bb908>
```



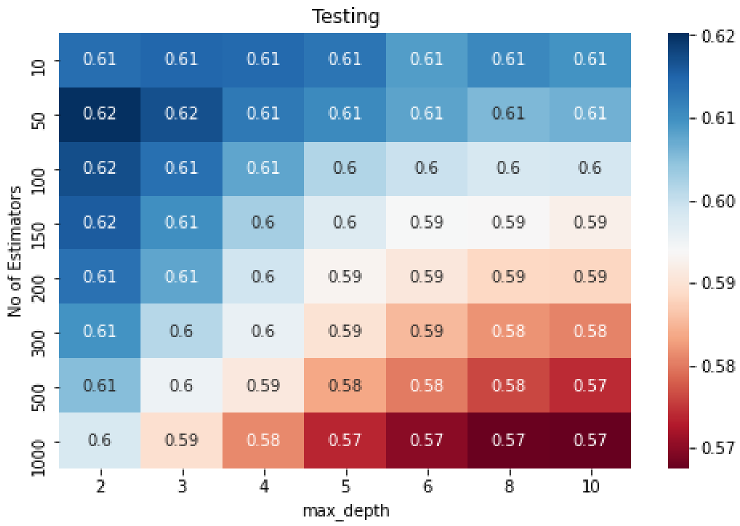
In [120]:

```
pivotted_2 = df_1.pivot_table(index='No of Estimators', columns='max_depth', values='test_auc', agg
func='mean')

plt.figure(figsize=(8,5))
plt.title('Testing')
sns.heatmap(pivotted_2, annot=True, cmap='RdBu')
```

Out[120]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f450ef822b0>
```



3.4 Modelling with Parameters

In [121]:

```
best_n_estimator = gridsearch_1.best_params_['n_estimators']
best_max_depth = gridsearch_1.best_params_['max_depth']
print('Best number of estimators:', best_n_estimator)
print('Best max depth:', best_max_depth)
```

```
Best number of estimators: 50
Best max depth: 2
```

In [122]:

```
classifier_withParam_1 = LGBMClassifier(n_estimators=best_n_estimator, max_depth=best_max_depth)
classifier_withParam_1.fit(X_train_1, y_train)
```

Out[122]:

```
LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
               importance_type='split', learning_rate=0.1, max_depth=2,
               min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
               n_estimators=50, n_jobs=-1, num_leaves=31, objective=None,
               random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
               subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

3.5 Cross Validation

In [0]:

```
from sklearn.model_selection import cross_val_score
cv_1 = cross_val_score(estimator=classifier_withParam_1, X=X_train_1, y=y_train, cv=2, scoring='roc
_auc')
```

In [124]:

```
best_auc_1 = cv_1.mean()
print('Best AUC: %4f' %best_auc_1)
```

```
Best AUC: 0.620257
```

3.6 ROC curve on train and test data

In [0]:

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

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

    return y_data_pred
```

In [0]:

```
y_train_pred_1 = batch_predict(classifier_withParam_1, X_train_1)
y_test_pred_1 = batch_predict(classifier_withParam_1, X_test_1)
```

In [0]:

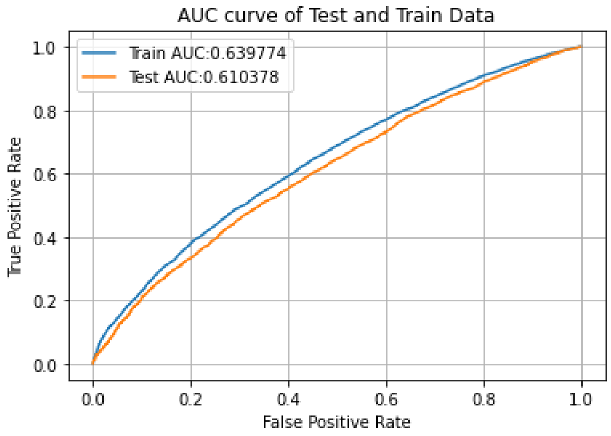
```
from sklearn.metrics import roc_curve, auc
train_fpr_1, train_tpr_1, train_thresh_1 = roc_curve(y_train, y_train_pred_1)
```

```
test_fpr_1, test_tpr_1, test_thresh_1 = roc_curve(y_test, y_test_pred_1)
```

In [128]:

```
plt.plot(train_fpr_1, train_tpr_1, label='Train AUC:%4f'%auc(train_fpr_1, train_tpr_1))
plt.plot(test_fpr_1, test_tpr_1, label='Test AUC:%4f'%auc(test_fpr_1, test_tpr_1))

plt.title('AUC curve of Test and Train Data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid()
plt.show()
```



3.7 Confusion Matrix

In [0]:

```
#finding best threshold by ourselves with the concept threshold will be high when fpr is low.
#So tpr*(1-fpr) gives max threshold
def find_best_threshold(fpr, tpr, threshold):
    t = threshold[np.argmax(tpr*(1-fpr))]
    print('the maximum tpr*(1-fpr) is :', max(tpr*(1-fpr)), 'for threshold', np.round(t,3))
    return t
```

In [130]:

```
best_t = find_best_threshold(train_fpr_1, train_tpr_1, train_thresh_1)
```

```
the maximum tpr*(1-fpr) is : 0.35623655137395266 for threshold 0.839
```

In [0]:

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

    return predictions
```

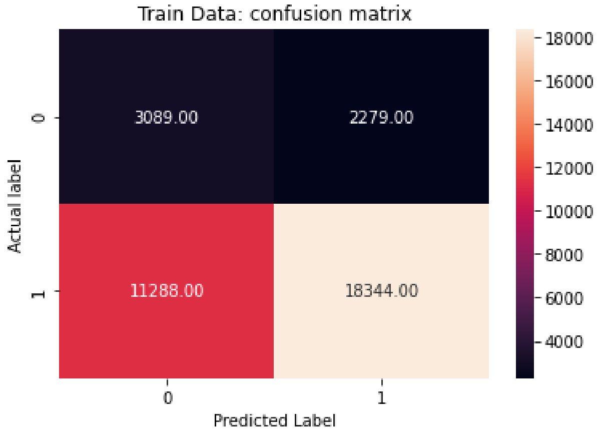
In [0]:

```
from sklearn.metrics import confusion_matrix
cm_train_1 = confusion_matrix(y_train, predict_with_threshold(y_train_pred_1, best_t))
cm_test_1 = confusion_matrix(y_test, predict_with_threshold(y_test_pred_1, best_t))
```

In [133]:

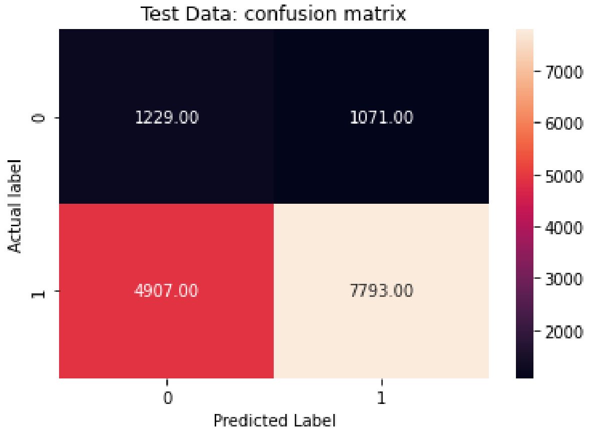
```
sns.heatmap(cm_train_1, annot=True, fmt='.2f')
```

```
plt.title('Train Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



In [135]:

```
sns.heatmap(cm_test_1, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



4. Summary:

In [140]:

```
from prettytable import PrettyTable
y = PrettyTable()

y.field_names = ['Set Number', 'Vectorizer', 'Model', 'Hyperparameter: n_estimators', 'Hyperparamet
er: max_depth', 'Best AUC']
y.add_row(['Set-1', 'Co-Occurence Matirx', 'LightGBM', str(best_n_estimator), str(best_max_depth),
str(best_auc_2)])

print(y)
```

```
+------------+---------------------+----------+------------------------------+---------------------
---+--------------------+
| Set Number |      Vectorizer     |  Model   | Hyperparameter: n_estimators | Hyperparameter: max_
depth |      Best AUC      |
+------------+---------------------+----------+------------------------------+---------------------
---+--------------------+
|   Set-1    | Co-Occurence Matirx | LightGBM |              50              |                2    
| 0.6202571715276191 |
+------------+---------------------+----------+------------------------------+---------------------
---+--------------------+
```

That's the end of the code