# RANDOM FOREST AND GRADIENT BOOSTING

## Note:

```
- Since we already did all these in previous assignments and we can use the same from it.
```

In [1]:

```
#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 [2]:

```
data = pd.read_csv('Preprocessed_inc_others.csv')
data.head()
```

Out[2]:

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

In [3]:

```
data.describe()
```

Out[3]:

| | Unnamed: 0 | teacher_number_of_previously_posted_projects | project_is_approved | price | quantity | sentiment_sco |
|---|---|---|---|---|---|---|
| count | 109248.000000 | 109248.000000 | 109248.000000 | 109248.000000 | 109248.000000 | 109248.0000 |
| mean | 54623.500000 | 11.153165 | 0.848583 | 298.119343 | 16.965610 | 0.2100 |
| std | 31537.325441 | 27.777154 | 0.358456 | 367.498030 | 26.182942 | 0.0835 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.660000 | 1.000000 | -0.1897 |
| 25% | 27311.750000 | 0.000000 | 1.000000 | 104.310000 | 4.000000 | 0.1543 |
| 50% | 54623.500000 | 2.000000 | 1.000000 | 206.220000 | 9.000000 | 0.2082 |
| 75% | 81935.250000 | 9.000000 | 1.000000 | 379.000000 | 21.000000 | 0.2643 |
| max | 109247.000000 | 451.000000 | 1.000000 | 9999.000000 | 930.000000 | 0.6633 |

In [4]:

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

Out[4]:

| | Unnamed: 0 | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projects | project_is_approved |
|---|---|---|---|---|---|---|
| 0 | 0 | ca | mrs | grades_prek_2 | 53 | 1 |
| 1 | 1 | ut | ms | grades_3_5 | 4 | 1 |

In [5]:

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

```
(109248, 1)
```

## Splitting the Data

In [6]:

```
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 [7]:

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

```
(76473, 15)
```

```
(32775, 15)
(76473, 1)
(32775, 1)
```

In [8]:

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

# 1. Vectorizing all features

## 1.1 Response Coding

In [9]:

```
def get_feat_dict(alpha, feature, df):
    value_counts = X_train[feature].value_counts()

    feat_dict = dict()

    for i,demominator in value_counts.items():
        vec = []

        for k in range(2):
            cls_cnt = X_train[(X_train['project_is_approved']==k) & (X_train[feature]==i)]
            vec.append((cls_cnt.shape[0]+alpha*10)/ (demominator+ alpha*20))

        feat_dict[i] = vec

    return feat_dict

def get_respCoded_feature(alpha, feature, df):
    feat_dict = get_feat_dict(alpha, feature, df)
    value_counts = X_train[feature].value_counts()

    encoded_feat = []

    for index, row in df.iterrows():
        if row[feature] in dict(value_counts).keys():
            encoded_feat.append(feat_dict[row[feature]])
        else:
                encoded_feat.append([1/2, 1/2])

    return encoded_feat
```

## 1.2 School State - Response Coding

In [10]:

```
data['school_state'].unique()
```

Out[10]:

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

In [11]:

```
len(data['school_state'].unique())
```

Out[11]:

```
51
```

In [12]:

```
X_train['school_state'].value_counts()
```

Out[12]:

```
ca    10840
tx     5168
ny     5126
fl     4379
nc     3579
il     3070
ga     2792
sc     2771
mi     2174
pa     2119
in     1835
mo     1798
oh     1730
ma     1669
la     1661
wa     1642
nj     1587
ok     1574
az     1475
va     1437
wi     1277
ut     1237
al     1214
tn     1167
ct     1165
md     1038
nv      960
ms      929
ky      878
or      876
mn      867
co      771
ar      715
id      488
ia      471
ks      443
nm      384
wv      364
hi      361
dc      359
me      350
ak      239
de      239
nh      231
ne      222
sd      210
ri      195
mt      176
nd      108
wy       63
vt       50
Name: school_state, dtype: int64
```

In [13]:

```
alpha = 1
X_train_Sstate_responseCoding = np.array(get_respCoded_feature(alpha, 'school_state', X_train))
X_test_Sstate_responseCoding = np.array(get_respCoded_feature(alpha, 'school_state', X_test))
```

In [14]:

```
print(X_train_Sstate_responseCoding.shape)
print(X_test_Sstate_responseCoding.shape)
```

```
(76473, 2)
(32775, 2)
```

## 1.3 Clean Categories

In [15]:

```
X_train['clean_categories'].unique()
```

Out[15]:

```
array(['specialneeds', 'health_sports', 'literacy_language specialneeds',
       'appliedlearning literacy_language', 'math_science',
       'literacy_language', 'literacy_language math_science',
       'history_civics music_arts', 'appliedlearning specialneeds',
       'math_science literacy_language', 'math_science music_arts',
       'health_sports literacy_language',
       'literacy_language appliedlearning', 'math_science history_civics',
       'math_science specialneeds', 'literacy_language music_arts',
       'health_sports specialneeds', 'music_arts', 'history_civics',
       'math_science appliedlearning', 'appliedlearning',
       'warmth care_hunger', 'health_sports math_science',
       'appliedlearning math_science', 'literacy_language history_civics',
       'music_arts specialneeds', 'history_civics specialneeds',
       'appliedlearning music_arts', 'math_science health_sports',
       'health_sports appliedlearning', 'appliedlearning health_sports',
       'health_sports music_arts', 'history_civics literacy_language',
       'history_civics math_science', 'specialneeds music_arts',
       'appliedlearning history_civics', 'health_sports history_civics',
       'history_civics appliedlearning',
       'literacy_language health_sports',
       'specialneeds warmth care_hunger', 'specialneeds health_sports',
       'music_arts history_civics', 'health_sports warmth care_hunger',
       'music_arts health_sports', 'history_civics health_sports',
       'appliedlearning warmth care_hunger', 'music_arts appliedlearning',
       'math_science warmth care_hunger',
       'literacy_language warmth care_hunger',
       'music_arts warmth care_hunger',
       'history_civics warmth care_hunger'], dtype=object)
```

In [16]:

```
len(X_train['clean_categories'].unique())
```

Out[16]:

```
51
```

In [17]:

```
X_train_cat_responseCoding = np.array(get_respCoded_feature(alpha, 'clean_categories', X_train))
X_test_cat_responseCoding = np.array(get_respCoded_feature(alpha, 'clean_categories', X_test))
```

In [18]:

```
print(X_train_cat_responseCoding.shape)
print(X_test_cat_responseCoding.shape)
```

```
(76473, 2)
(32775, 2)
```

## 1.4 Clean Sub categories

In [19]:

```
X_train['clean_subcategories'].unique()
```

Out[19]:

```
array(['specialneeds', 'health_wellness', 'literacy specialneeds',
       'earlydevelopment literacy', 'mathematics',
       'earlydevelopment literacy', 'mathematics',
       'literacy literature_writing', 'literature_writing mathematics',
       'esl', 'gym_fitness teamsports', 'literature_writing',
       'civics_government visualarts', 'literacy mathematics',
       'foreignlanguages', 'other specialneeds', 'appliedsciences',
       'health_lifescience literature_writing',
       'appliedsciences visualarts', 'gym_fitness', 'literacy',
       'health_wellness literature_writing', 'esl parentinvolvement',
       'health_lifescience socialsciences',
       'appliedsciences specialneeds', 'health_lifescience literacy',
       'literature_writing music', 'gym_fitness health_wellness',
       'health_wellness specialneeds', 'performingarts',
       'environmentalscience mathematics', 'history_geography',
       'literature_writing specialneeds',
       'appliedsciences environmentalscience',
       'appliedsciences mathematics',
       'college_careerprep literature_writing',
       'history_geography socialsciences', 'health_lifescience',
       'appliedsciences extracurricular',
       'appliedsciences college_careerprep', 'mathematics specialneeds',
       'college_careerprep', 'music performingarts',
       'literacy visualarts', 'visualarts', 'warmth care_hunger',
       'gym_fitness health_lifescience', 'esl specialneeds',
       'environmentalscience health_lifescience',
       'college_careerprep health_lifescience',
       'literature_writing visualarts', 'financialliteracy',
       'literature_writing other', 'nutritioneducation specialneeds',
       'other', 'earlydevelopment other',
       'health_wellness nutritioneducation', 'esl literacy',
       'literacy socialsciences', 'college_careerprep literacy',
       'charactereducation college_careerprep',
       'appliedsciences socialsciences', 'music specialneeds',
       'socialsciences', 'charactereducation specialneeds',
       'financialliteracy specialneeds',
       'earlydevelopment literature_writing',
       'health_wellness teamsports', 'appliedsciences literacy',
       'environmentalscience history_geography',
       'charactereducation literature_writing', 'appliedsciences other',
       'earlydevelopment mathematics', 'college_careerprep visualarts',
       'charactereducation other', 'health_lifescience mathematics',
       'earlydevelopment environmentalscience', 'environmentalscience',
       'environmentalscience literacy', 'literacy parentinvolvement',
       'environmentalscience nutritioneducation', 'economics',
       'teamsports', 'music', 'literature_writing socialsciences',
       'communityservice', 'esl foreignlanguages', 'charactereducation',
       'gym_fitness specialneeds',
       'charactereducation health_lifescience',
       'earlydevelopment specialneeds', 'college_careerprep specialneeds',
       'environmentalscience literature_writing', 'extracurricular',
       'mathematics visualarts', 'health_wellness other',
       'mathematics other', 'health_wellness mathematics',
       'charactereducation health_wellness', 'earlydevelopment',
       'foreignlanguages literacy', 'civics_government socialsciences',
       'college_careerprep other', 'appliedsciences health_lifescience',
       'health_wellness literacy', 'esl literature_writing',
       'gym_fitness performingarts', 'history_geography literacy',
       'nutritioneducation', 'civics_government',
       'civics_government environmentalscience',
       'college_careerprep parentinvolvement',
       'financialliteracy mathematics', 'college_careerprep mathematics',
       'specialneeds visualarts', 'appliedsciences literature_writing',
       'appliedsciences music', 'music visualarts',
       'earlydevelopment visualarts',
       'foreignlanguages literature_writing',
       'college_careerprep foreignlanguages',
       'socialsciences specialneeds',
       'history_geography literature_writing',
       'history_geography visualarts',
       'college_careerprep history_geography',
       'charactereducation literacy', 'extracurricular teamsports',
       'civics_government history_geography',
       'earlydevelopment gym_fitness', 'foreignlanguages visualarts',
       'economics history_geography', 'economics financialliteracy',
       'earlydevelopment health_wellness', 'extracurricular music',
       'nutritioneducation socialsciences', 'civics_government literacy',
       'appliedsciences earlydevelopment',
       'gym_fitness literature_writing', 'civics_government mathematics',
       'literacy other', 'foreignlanguages mathematics',
       'literacy other', 'foreignlanguages mathematics',
       'history_geography performingarts',
       'appliedsciences parentinvolvement',
       'environmentalscience specialneeds', 'gym_fitness music',
       'history_geography specialneeds',
       'earlydevelopment health_lifescience',
       'health_lifescience health_wellness',
       'environmentalscience health_wellness',
       'gym_fitness nutritioneducation', 'history_geography mathematics',
       'literature_writing performingarts',
       'health_lifescience visualarts',
       'college_careerprep communityservice',
       'mathematics performingarts',
       'college_careerprep nutritioneducation',
       'mathematics socialsciences', 'charactereducation socialsciences',
       'appliedsciences history_geography',
       'nutritioneducation teamsports', 'environmentalscience visualarts',
       'civics_government literature_writing',
       'health_wellness socialsciences',
       'communityservice literature_writing', 'other visualarts',
       'appliedsciences communityservice',
       'charactereducation extracurricular',
       'civics_government college_careerprep',
       'civics_government parentinvolvement', 'extracurricular literacy',
       'mathematics parentinvolvement', 'appliedsciences health_wellness',
       'earlydevelopment history_geography', 'esl health_wellness',
       'performingarts visualarts', 'teamsports visualarts',
       'communityservice visualarts', 'earlydevelopment extracurricular',
       'health_lifescience nutritioneducation',
       'health_lifescience music', 'parentinvolvement specialneeds',
       'extracurricular gym_fitness',
       'charactereducation environmentalscience',
       'charactereducation communityservice',
       'communityservice socialsciences',
       'literature_writing parentinvolvement',
       'extracurricular visualarts',
       'charactereducation earlydevelopment',
       'civics_government communityservice',
       'health_lifescience history_geography', 'mathematics teamsports',
       'health_wellness history_geography', 'gym_fitness literacy',
       'charactereducation financialliteracy', 'communityservice other',
       'esl earlydevelopment', 'environmentalscience performingarts',
       'communityservice specialneeds', 'charactereducation teamsports',
       'esl socialsciences', 'other parentinvolvement', 'esl mathematics',
       'other teamsports', 'college_careerprep financialliteracy',
       'college_careerprep performingarts',
       'charactereducation mathematics',
       'specialneeds warmth care_hunger', 'literacy teamsports',
       'extracurricular mathematics', 'esl health_lifescience',
       'health_wellness performingarts', 'economics mathematics',
       'literacy music', 'mathematics music',
       'health_lifescience specialneeds',
       'earlydevelopment performingarts', 'appliedsciences economics',
       'environmentalscience teamsports',
       'environmentalscience socialsciences',
       'civics_government economics',
       'college_careerprep health_wellness', 'socialsciences visualarts',
       'communityservice environmentalscience',
       'health_lifescience parentinvolvement',
       'health_wellness visualarts', 'charactereducation visualarts',
       'communityservice health_wellness',
       'charactereducation history_geography', 'literacy performingarts',
       'extracurricular nutritioneducation',
       'college_careerprep extracurricular', 'charactereducation music',
       'financialliteracy literacy', 'gym_fitness mathematics',
       'extracurricular specialneeds', 'esl financialliteracy',
       'civics_government health_lifescience',
       'charactereducation parentinvolvement',
       'foreignlanguages health_wellness', 'specialneeds teamsports',
       'parentinvolvement', 'appliedsciences esl',
       'environmentalscience other', 'appliedsciences charactereducation',
       'appliedsciences financialliteracy',
       'extracurricular health_wellness',
       'college_careerprep earlydevelopment',
       'performingarts socialsciences', 'communityservice literacy',
       'economics socialsciences', 'communityservice nutritioneducation',
       'health_wellness warmth care_hunger', 'appliedsciences teamsports',
       'extracurricular financialliteracy',
       'charactereducation civics_government',
       'communityservice history_geography',
       'financialliteracy literature_writing',
       'earlydevelopment parentinvolvement', 'esl environmentalscience',
       'college_careerprep gym_fitness', 'performingarts specialneeds',
       'earlydevelopment music',
       'college_careerprep environmentalscience', 'other performingarts',
       'environmentalscience foreignlanguages', 'college_careerprep esl',
       'gym_fitness warmth care_hunger',
       'foreignlanguages history_geography', 'communityservice economics',
       'extracurricular literature_writing', 'nutritioneducation other',
       'college_careerprep socialsciences', 'health_wellness music',
       'appliedsciences civics_government',
       'mathematics nutritioneducation',
       'earlydevelopment socialsciences',
       'civics_government specialneeds', 'gym_fitness socialsciences',
       'foreignlanguages performingarts', 'economics visualarts',
       'music teamsports', 'esl visualarts',
       'environmentalscience parentinvolvement',
       'civics_government financialliteracy', 'history_geography music',
       'foreignlanguages socialsciences', 'esl other',
       'other socialsciences', 'esl music', 'performingarts teamsports',
       'communityservice financialliteracy',
       'parentinvolvement visualarts', 'extracurricular performingarts',
       'economics literature_writing', 'appliedsciences performingarts',
       'parentinvolvement teamsports',
       'environmentalscience extracurricular',
       'earlydevelopment financialliteracy', 'esl history_geography',
       'charactereducation esl', 'financialliteracy health_wellness',
       'environmentalscience financialliteracy',
       'civics_government performingarts',
       'communityservice extracurricular', 'economics other',
       'history_geography other', 'earlydevelopment teamsports',
       'economics environmentalscience',
       'charactereducation performingarts',
       'parentinvolvement warmth care_hunger',
       'literature_writing teamsports',
       'financialliteracy history_geography', 'foreignlanguages music',
       'foreignlanguages other', 'communityservice parentinvolvement',
       'parentinvolvement performingarts',
       'health_wellness parentinvolvement', 'extracurricular other',
       'appliedsciences gym_fitness', 'financialliteracy performingarts',
       'economics nutritioneducation', 'foreignlanguages specialneeds',
       'music other', 'college_careerprep music',
       'extracurricular socialsciences',
       'extracurricular parentinvolvement', 'economics foreignlanguages',
       'nutritioneducation visualarts',
       'communityservice earlydevelopment', 'gym_fitness visualarts',
       'health_lifescience warmth care_hunger',
       'health_lifescience teamsports', 'music parentinvolvement',
       'financialliteracy visualarts',
       'literature_writing warmth care_hunger',
       'financialliteracy foreignlanguages', 'environmentalscience music',
       'health_lifescience performingarts',
       'history_geography parentinvolvement',
       'appliedsciences foreignlanguages', 'esl performingarts',
       'literacy warmth care_hunger', 'environmentalscience gym_fitness',
       'charactereducation foreignlanguages',
       'parentinvolvement socialsciences',
       'financialliteracy socialsciences', 'music socialsciences',
       'extracurricular history_geography',
       'college_careerprep economics', 'civics_government esl',
       'foreignlanguages health_lifescience',
       'earlydevelopment nutritioneducation',
       'nutritioneducation warmth care_hunger', 'economics literacy',
       'gym_fitness other', 'appliedsciences nutritioneducation',
       'communityservice mathematics', 'earlydevelopment economics',
       'literacy nutritioneducation',
       'financialliteracy health_lifescience',
       'charactereducation gym_fitness', 'economics specialneeds',
       'esl nutritioneducation', 'literature_writing nutritioneducation',
       'communityservice esl', 'economics music',
       'civics_government teamsports', 'extracurricular foreignlanguages',
       'other warmth care_hunger', 'gym_fitness history_geography',
       'charactereducation warmth care_hunger',
       'communityservice health_lifescience',
       'gym_fitness parentinvolvement', 'health_lifescience other',
       'foreignlanguages gym_fitness', 'college_careerprep teamsports',
       'civics_government foreignlanguages',
       'environmentalscience warmth care_hunger',
       'charactereducation nutritioneducation', 'esl gym_fitness',
       'charactereducation economics', 'history_geography teamsports',
       'civics_government nutritioneducation', 'esl teamsports',
       'earlydevelopment foreignlanguages',
       'extracurricular health_lifescience',
       'earlydevelopment warmth care_hunger',
       'economics health_lifescience',
       'appliedsciences warmth care_hunger',
       'communityservice gym_fitness', 'esl extracurricular',
       'visualarts warmth care_hunger', 'socialsciences teamsports',
       'history_geography warmth care_hunger'], dtype=object)
```

In [20]:

```
len(X_train['clean_subcategories'].unique())
```

Out[20]:

```
392
```

In [21]:

```
X_train_subcat_responseCoding = np.array(get_respCoded_feature(alpha, 'clean_subcategories', X_trai
n))
X_test_subcat_responseCoding = np.array(get_respCoded_feature(alpha, 'clean_subcategories', X_test)
)
```

In [22]:

```
print(X_train_subcat_responseCoding.shape)
print(X_test_subcat_responseCoding.shape)
```

```
(76473, 2)
(32775, 2)
```

### 1.5 Project Grade Category

In [23]:

```
X_train['project_grade_category'].unique()
```

Out[23]:

```
array(['grades_prek_2', 'grades_3_5', 'grades_9_12', 'grades_6_8'],
      dtype=object)
```

In [24]:

```
len(X_train['project_grade_category'].unique())
```

Out[24]:

```
4
```

In [25]:

```
alpha = 1
X_train_grade_responseCoding = np.array(get_respCoded_feature(alpha, 'project_grade_category',
X_train))
X_test_grade_responseCoding = np.array(get_respCoded_feature(alpha, 'project_grade_category',
X_test))
```

In [26]:

```
print(X_train_grade_responseCoding.shape)
print(X_test_grade_responseCoding.shape)
```

```
(76473, 2)
(32775, 2)
```

## 1.5 Teacher Prefix

In [27]:

```
X_train['teacher_prefix'].unique()
```

Out[27]:

```
array(['mrs', 'ms', 'mr', 'teacher', 'dr'], dtype=object)
```

In [28]:

```
len(X_train['teacher_prefix'].unique())
```

Out[28]:

```
5
```

In [29]:

```
X_train_prefix_responseCoding = np.array(get_respCoded_feature(alpha, 'teacher_prefix', X_train))
X_test_prefix_responseCoding = np.array(get_respCoded_feature(alpha, 'teacher_prefix', X_test))
```

In [30]:

```
print(X_train_prefix_responseCoding.shape)
print(X_test_prefix_responseCoding.shape)
```

```
(76473, 2)
(32775, 2)
```

## 1.6 Essay

### 1.6.1 BOW

In [31]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_6 = CountVectorizer(list(X_train['essay'].values), min_df=10)
```

In [32]:

```
#We are considering the words which occur in atleast 10 documents and max_features=5000 because we
need only those important 5000 words
X_train_essay_bow = vectorizer_6.fit_transform(X_train['essay'].values)
X_test_essay_bow = vectorizer_6.transform(X_test['essay'].values)
```

In [33]:

```
print(X_train_essay_bow.shape)
print(X_test_essay_bow.shape)
```

```
(76473, 14484)
(32775, 14484)
```

### 1.6.2 TFIDF

In [34]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_7 = TfidfVectorizer(list(X_train['essay'].values), min_df=10)
```

In [35]:

```
X_train_essay_tfidf = vectorizer_7.fit_transform(X_train['essay'].values)
X_test_essay_tfidf = vectorizer_7.transform(X_test['essay'].values)
```

In [36]:

```
print(X_train_essay_tfidf.shape)
print(X_test_essay_tfidf.shape)
```

```
(76473, 14484)
(32775, 14484)
```

### 1.6.3 AvgW2V

In [37]:

```
X_train.head(2)
```

Out[37]:

| | Unnamed: 0 | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projects | project_is_appro |
|---|---|---|---|---|---|---|
| 64526 | 64526 | tx | mrs | grades_prek_2 | 2 | |
| 82028 | 82028 | wa | ms | grades_prek_2 | 7 | |

In [38]:

```
#Unpickling
import pickle
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words = set(model.keys())
```

In [39]:

```
X_train_essay_avg_w2v = []
X_test_essay_avg_w2v = []

for i in tqdm(X_train['essay']):
    vector = np.zeros(300)
    cnt_words = 0

    for word in i.split():
        if word in glove_words:
            vector += model[word]
            cnt_words +=1

    if cnt_words !=0:
            vector /= cnt_words
    X_train_essay_avg_w2v.append(vector)

for i in tqdm(X_test['essay']):
    vector = np.zeros(300)
    cnt_words = 0
    for word in i.split():
        if word in glove_words :
            vector += model[word]
            cnt_words += 1

    if cnt_words != 0:
        vector /= cnt_words
    X_test_essay_avg_w2v.append(vector)
```

```
100%|███████████████████████████████████████████████████████████████████████████| 76473/76473
[00:22<00:00, 3454.61it/s]
100%|███████████████████████████████████████████████████████████████████████████| 32775/32775
[00:10<00:00, 3244.41it/s]
```

In [40]:

```
print(len(X_train_essay_avg_w2v))
print(len(X_train_essay_avg_w2v[0]))
print('='*50)
print(len(X_test_essay_avg_w2v))
print(len(X_test_essay_avg_w2v[0]))
```

```
76473
300
==================================================
32775
300
```

### 1.6.4 TFIDF W2V

In [41]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
Tf_idf_model = TfidfVectorizer()
Tf_idf_model.fit(X_train['essay'])

dictionary = dict(zip(Tf_idf_model.get_feature_names(), Tf_idf_model.idf_))
tf_idf_words = set(Tf_idf_model.get_feature_names())
```

In [42]:

```
# TFIDF Word2Vec
# compute TFIDF word2vec for each review.
X_train_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tf_idf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_essay_tfidf_w2v.append(vector)


X_test_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tf_idf_words):
```

```
        if (word in glove_words) and (word in tf_idf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_essay_tfidf_w2v.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 76473/76473 [02:
24<00:00, 527.95it/s]
100%|████████████████████████████████████████████████████████████████████████████| 32775/32775 [01:
02<00:00, 527.12it/s]
```

In [43]:

```
print(len(X_train_essay_tfidf_w2v))
print(len(X_test_essay_tfidf_w2v))
```

```
76473
32775
```

## 1.7 Project Title

### 1.7.1 BOW

In [44]:

```
vectorizer_8 = CountVectorizer(list(X_train['title'].values), min_df=10)
```

In [46]:

```
X_train_title_bow = vectorizer_8.fit_transform(X_train['title'].values)
X_test_title_bow = vectorizer_8.transform(X_test['title'].values)
```

In [47]:

```
print(X_train_title_bow.shape)
print(X_test_title_bow.shape)
```

```
(76473, 2689)
(32775, 2689)
```

### 1.7.2 TFIDF

In [48]:

```
vectorizer_9 = TfidfVectorizer(list(X_train['title'].values), min_df=10)
```

In [49]:

```
X_train_title_tfidf = vectorizer_9.fit_transform(X_train['title'].values)
X_test_title_tfidf = vectorizer_9.transform(X_test['title'].values)
```

In [50]:

```
print(X_train_title_tfidf.shape)
print(X_test_title_tfidf.shape)
```

```
(76473, 2689)
(32775, 2689)
```

### 1.7.3 AvgW2V

In [51]:

```
#Avgw2V vector for preprcessed essay

X_train_title_avg_w2v = []
X_test_title_avg_w2v = []

for i in tqdm(X_train['title']):
    vector = np.zeros(300)
    cnt_words = 0

    for word in i.split():
        if word in glove_words:
            vector += model[word]
            cnt_words +=1

    if cnt_words !=0:
            vector /= cnt_words

    X_train_title_avg_w2v.append(vector)

for i in tqdm(X_test['title']):
    vector = np.zeros(300)
    cnt_words = 0

    for word in i.split():
        if word in glove_words:
            vector += model[word]
            cnt_words +=1

    if cnt_words !=0:
            vector /= cnt_words

    X_test_title_avg_w2v.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 76473/76473
[00:01<00:00, 67857.13it/s]
100%|██████████████████████████████████████████████████████████████████████████| 32775/32775
[00:00<00:00, 70370.91it/s]
```

In [52]:

```
print(len(X_train_title_avg_w2v))
print(len(X_train_title_avg_w2v[0]))
print('='*50)
print(len(X_test_title_avg_w2v))
print(len(X_test_title_avg_w2v[0]))
```

```
76473
300
==================================================
32775
300
```

### 1.7.4 TFIDFW2V

In [53]:

```
#Training
Tf_idf_model = TfidfVectorizer()
Tf_idf_model.fit(X_train['title'])

dictionary = dict(zip(Tf_idf_model.get_feature_names(), Tf_idf_model.idf_))
tf_idf_words = set(Tf_idf_model.get_feature_names())
```

In [54]:

```
# TFIDF Word2Vec
# compute TFIDF word2vec for each review.
X_train_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tf_idf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_title_tfidf_w2v.append(vector)


X_test_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tf_idf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_title_tfidf_w2v.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 76473/76473
[00:02<00:00, 33557.01it/s]
100%|██████████████████████████████████████████████████████████████████████████| 32775/32775
[00:00<00:00, 33984.61it/s]
```

In [55]:

```
print(len(X_train_title_tfidf_w2v))
print(len(X_test_title_tfidf_w2v))
```

```
76473
32775
```

## 1.8 Price

### 1.8.1 Price Unstandardised

In [56]:

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

In [57]:

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

```
(76473, 1)
(32775, 1)
```

### 1.8.2 Price Standardized

In [58]:

```
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 [59]:

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

```
(76473, 1)
(32775, 1)
```

## 1.9 Previously posted Projects

### 1.9.1 Unstandardized

In [60]:

```
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 [61]:

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

```
(76473, 1)
(32775, 1)
```

### 1.9.2 Standardized

In [62]:

```
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 [63]:

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

```
(76473, 1)
(32775, 1)
```

## 1.10 Quantity

### 1.10.1 Unstandardized

In [64]:

```
X_train_quantity_unstandardized = X_train['quantity'].values.reshape(-1,1)
X_train_quantity_unstandardized = X_train['quantity'].values.reshape(-1,1)
X_test_quantity_unstandardized = X_test['quantity'].values.reshape(-1,1)
```

In [65]:

```
print(X_train_quantity_unstandardized.shape)
print(X_test_quantity_unstandardized.shape)
```

```
(76473, 1)
(32775, 1)
```

### 1.10.2 Standardized

In [66]:

```
sc_quantity = StandardScaler()
X_train_quantity = sc_quantity.fit_transform(X_train['quantity'].values.reshape(-1,1))
X_test_quantity = sc_quantity.transform(X_test['quantity'].values.reshape(-1,1))
```

In [67]:

```
print(X_train_quantity.shape)
print(X_test_quantity.shape)
```

```
(76473, 1)
(32775, 1)
```

## 1.11 Sentiment Score

In [68]:

```
X_train_sentiment = X_train['sentiment_score'].values.reshape(-1,1)
X_test_sentiment = X_test['sentiment_score'].values.reshape(-1,1)
```

In [69]:

```
print(X_train_sentiment.shape)
print(X_test_sentiment.shape)
```

```
(76473, 1)
(32775, 1)
```

## 1.12 Number of Words in the title

### 1.12.1 Unstandardized

In [70]:

```
X_train_words_title_unstandardized = X_train['No of words in project title'].values.reshape(-1,1)
X_test_words_title_unstandardized = X_test['No of words in project title'].values.reshape(-1,1)
```

In [71]:

```
print(X_train_words_title_unstandardized.shape)
print(X_test_words_title_unstandardized.shape)
```

```
(76473, 1)
(32775, 1)
```

### 1.12.2 Standardized

In [72]:

```
sc_title_words = StandardScaler()
X_train_words_title = sc_title_words.fit_transform(X_train['No of words in project title'].values.reshape(-1,1))
X_test_words_title = sc_title_words.transform(X_test['No of words in project title'].values.reshape(-1,1))
```

In [73]:

```
print(X_train_words_title.shape)
print(X_test_words_title.shape)
```

```
(76473, 1)
(32775, 1)
```

## 1.13 Number of Words in the essay

### 1.13.1 Unstandardized

In [74]:

```
X_train_words_essay_unstandardized = X_train['No of words in essay'].values.reshape(-1,1)
X_test_words_essay_unstandardized = X_test['No of words in essay'].values.reshape(-1,1)
```

In [75]:

```
print(X_train_words_essay_unstandardized.shape)
print(X_test_words_essay_unstandardized.shape)
```

```
(76473, 1)
(32775, 1)
```

### 1.13.2 Standardized

In [76]:

```
sc_essay_words = StandardScaler()
X_train_words_essay = sc_essay_words.fit_transform( X_train['No of words in essay'].values.reshape(-1,1))
X_test_words_essay = sc_essay_words.transform(X_test['No of words in essay'].values.reshape(-1,1))
```

In [77]:

```
print(X_train_words_essay.shape)
print(X_test_words_essay.shape)
```

```
(76473, 1)
(32775, 1)
```

In [78]:

```
print(X_train_Sstate_responseCoding.shape)
print(X_train_cat_responseCoding.shape)
print(X_train_subcat_responseCoding.shape)
print(X_train_grade_responseCoding.shape)
print(X_train_prefix_responseCoding.shape)
print(X_train_price.shape)
print(X_train_previous.shape)

print(X_train_essay_bow.shape)
print(X_train_essay_tfidf.shape)
print(len(X_train_essay_avg_w2v))
print(len(X_train_essay_tfidf_w2v))
```

```python
print(X_train_title_bow.shape)
print(X_train_title_tfidf.shape)
print(len(X_train_title_avg_w2v))
print(len(X_train_title_tfidf_w2v))

print(X_train_quantity.shape)
print(X_train_sentiment.shape)
print(X_train_words_essay.shape)
print(X_train_words_title.shape)

print('='*50)
print(X_test_Sstate_responseCoding.shape)
print(X_test_cat_responseCoding.shape)
print(X_test_subcat_responseCoding.shape)
print(X_test_grade_responseCoding.shape)
print(X_test_prefix_responseCoding.shape)
print(X_test_price.shape)
print(X_test_previous.shape)

print(X_test_essay_bow.shape)
print(X_test_essay_tfidf.shape)
print(len(X_test_essay_avg_w2v))
print(len(X_test_essay_tfidf_w2v))

print(X_test_title_bow.shape)
print(X_test_title_tfidf.shape)
print(len(X_test_title_avg_w2v))
print(len(X_test_title_tfidf_w2v))

print(X_test_quantity.shape)
print(X_test_sentiment.shape)
print(X_test_words_essay.shape)
print(X_test_words_title.shape)
```

```
(76473, 2)
(76473, 2)
(76473, 2)
(76473, 2)
(76473, 2)
(76473, 1)
(76473, 1)
(76473, 14484)
(76473, 14484)
76473
76473
(76473, 2689)
(76473, 2689)
76473
76473
(76473, 1)
(76473, 1)
(76473, 1)
(76473, 1)
==================================================
(32775, 2)
(32775, 2)
(32775, 2)
(32775, 2)
(32775, 2)
(32775, 1)
(32775, 1)
(32775, 14484)
(32775, 14484)
32775
32775
(32775, 2689)
(32775, 2689)
32775
32775
(32775, 1)
(32775, 1)
(32775, 1)
(32775, 1)
```

# Model - RANDOM FOREST

## 2. Set - 1

### 2.1 Merging the above features required for Set-1

In [0]:

```
from scipy.sparse import hstack
X_train_1 = hstack((X_train_Sstate_responseCoding, X_train_cat_responseCoding,
X_train_subcat_responseCoding,
                    X_train_grade_responseCoding, X_train_prefix_responseCoding, X_train_essay_bow,

                    X_train_title_bow, X_train_previous, X_train_price)).tocsr()

X_test_1 = hstack((X_test_Sstate_responseCoding, X_test_cat_responseCoding,
X_test_subcat_responseCoding,
                   X_test_grade_responseCoding, X_test_prefix_responseCoding, X_test_essay_bow, X_t
est_title_bow,
                   X_test_previous, X_test_price)).tocsr()
```

In [91]:

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

```
(76473, 17185)
(32775, 17185)
```

### 2.2 Grid Search CV -

In [0]:

```
from sklearn.ensemble import RandomForestClassifier
classifier_1 = RandomForestClassifier()
```

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)
```

In [0]:

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

In [98]:

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

Out[98]:

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_max_depth | param_n_estimators | params | split0_test_sc |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.339530 | 0.013261 | 0.212754 | 0.000222 | 2 | 10 | {'max_depth': 2, 'n_estimators': 10} | 0.6005 |

|  | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_max_depth | param_n_estimators | params | split0_test_sc |
|---|---|---|---|---|---|---|---|---|
| **32** | 0.343403 | 0.003464 | 0.259001 | 0.006278 | 6 | 10 | {'max_depth': 6, 'n_estimators': 10} | 0.620 |
| **16** | 0.299457 | 0.002333 | 0.251793 | 0.002223 | 4 | 10 | {'max_depth': 4, 'n_estimators': 10} | 0.609 |
| **40** | 0.400789 | 0.008480 | 0.222691 | 0.005816 | 8 | 10 | {'max_depth': 8, 'n_estimators': 10} | 0.642 |
| **8** | 0.296549 | 0.011832 | 0.255416 | 0.004061 | 3 | 10 | {'max_depth': 3, 'n_estimators': 10} | 0.573 |

In [99]:

```
best_n_estimators_1 = gridsearch_1.best_params_['n_estimators']
best_max_depth_1 = gridsearch_1.best_params_['max_depth']
print('Best Number of Estimators:', best_n_estimators_1)
print('Best Max depth in a tree:', best_max_depth_1)
```

```
Best Number of Estimators: 300
Best Max depth in a tree: 10
```

**Note:**

```
- We can see that the best number of estimators is 1000 and Regulariser is 10
```

## 2.3 AUC vs Hyperparameter

In [0]:

```
train_auc_1 = results['mean_train_score']
test_auc_1 = results['mean_test_score']
n_estimator_1 = results['param_n_estimators']
max_depth_1 =results['param_max_depth']
```

In [0]:

```
#To plot this into plotly it should be in list
n_estimator_1 = n_estimator_1.to_list()
max_depth_1 = max_depth_1.to_list()
train_auc_1 = train_auc_1.to_list()
test_auc_1 = test_auc_1.to_list()
```

In [102]:

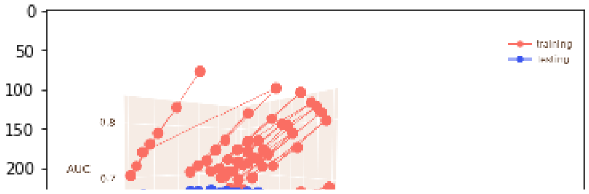
```
# To enable plotly plot in Google Colab
# https://stackoverflow.com/questions/47230817/plotly-notebook-mode-with-google-
colaboratory/47230966
def configure_plotly_browser_state():
    import IPython
    display(IPython.core.display.HTML('''
        <script src="/static/components/requirejs/require.js"></script>
        <script>
          requirejs.config({
            paths: {
              base: '/static/base',
              plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
            },
          });
        </script>
        '''))

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"></s
cript>'''))
    init_notebook_mode(connected=False)


#To enable plotly plot in this cell
enable_plotly_in_cell()

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=n_estimator_1, y=max_depth_1, z=train_auc_1, name='training')
trace2 = go.Scatter3d(x=n_estimator_1, y=max_depth_1, z=test_auc_1, name='Testing')
data = [trace1, trace2]

layout = go.Layout(
                    scene=dict(
                                xaxis = dict(title='No of Estimators'),
                                yaxis = dict(title='Max Depth'),
                                zaxis = dict(title='AUC'),
                                )
                    )
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='3d-scatter-colorscale')
```

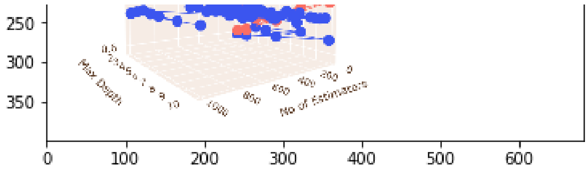
In [113]:

```
import matplotlib.pyplot as plt
import cv2
img_1 = cv2.imread(r'C:\Users\prem.k\Desktop\My projects\Assign\RFGBDT pics\1.PNG')
plt.imshow(img_1)
```

Out[113]:

```
<matplotlib.image.AxesImage at 0x1a50372ea90>
```





## 2.4 Modelling with best C and penalty to find best AUC

In [0]:

```
classifier_withParam_1 = RandomForestClassifier(n_estimators=best_n_estimators_1,
max_depth=best_max_depth_1)
classifier_withParam_1.fit(X_train_1, y_train)
```

Out[0]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=10, max_features='auto',
                       max_leaf_nodes=None, max_samples=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=1000,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

## 2.5 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 [0]:

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

```
Best AUC: 0.678692
```

## 2.6 ROC curve on train and test data

In [0]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [0]:

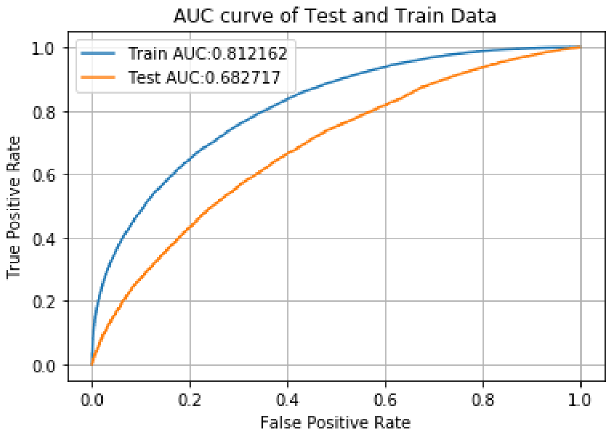
```
y_train_predproba_1 = batch_predict(classifier_withParam_1, X_train_1)
y_test_predproba_1 = batch_predict(classifier_withParam_1, X_test_1)
```

In [0]:

```
from sklearn.metrics import roc_curve, auc
train_fpr_1, train_tpr_1, train_thresh_1 = roc_curve(y_train, y_train_predproba_1)
test_fpr_1, test_tpr_1, test_thresh_1 = roc_curve(y_test, y_test_predproba_1)
```

In [0]:

```
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()
```



## 2.7 Finding Confusion Matrix

In [0]:

```
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 [0]:

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

```
the maximum tpr*(1-fpr) is : 0.5303004217846466 for threshold 0.844
```

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]:

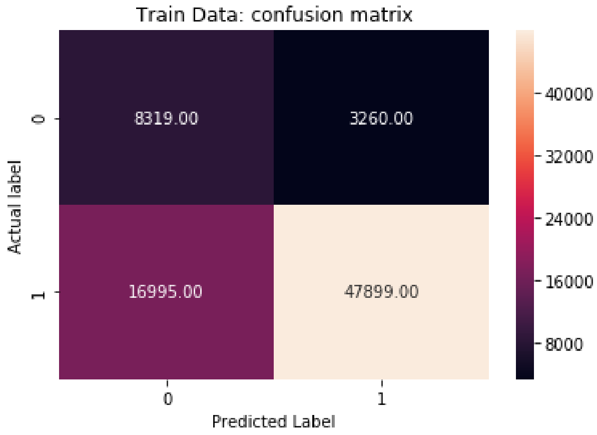
```
y_train_pred_1 = predict_with_threshold(y_train_predproba_1, best_t)
y_test_pred_1 = predict_with_threshold(y_test_predproba_1, best_t)
```

In [0]:

```
from sklearn.metrics import confusion_matrix

cm_train_1 = confusion_matrix(y_train, y_train_pred_1)
cm_test_1 = confusion_matrix(y_test, y_test_pred_1)
```

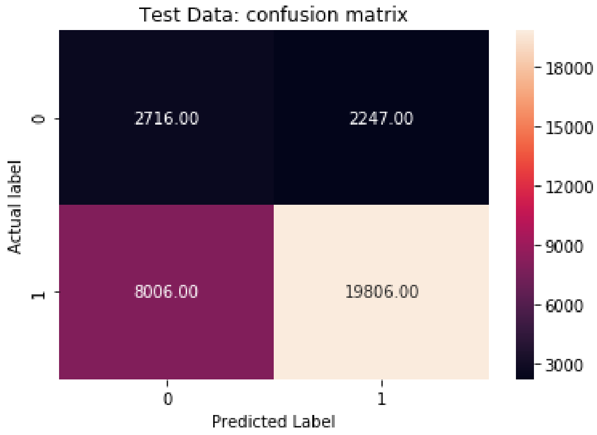
In [0]:

```
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 [0]:

```
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()
```



# Set - 2

## 3.1 Merging all columns

In [0]:

```
X_train_2 = hstack((X_train_Sstate_responseCoding, X_train_cat_responseCoding,
X_train_subcat_responseCoding,
                    X_train_grade_responseCoding, X_train_prefix_responseCoding,
X_train_essay_tfidf,
                    X_train_title_tfidf, X_train_previous, X_train_price)).tocsr()

X_test_2 = hstack((X_test_Sstate_responseCoding, X_test_cat_responseCoding,
X_test_subcat_responseCoding,
                   X_test_grade_responseCoding, X_test_prefix_responseCoding, X_test_essay_tfidf,
                   X_test_title_tfidf, X_test_previous, X_test_price)).tocsr()
```

In [104]:

```
print(X_train_2.shape)
print(X_test_2.shape)
```

```
(76473, 17185)
(32775, 17185)
```

### 3.2 GridSearch

In [0]:

```
from sklearn.ensemble import RandomForestClassifier
classifier_2 = RandomForestClassifier()
```

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_2 = GridSearchCV(estimator=classifier_2, param_grid=parameters, scoring='roc_auc', cv=2
, n_jobs=-1, return_train_score=True)
```

In [0]:

```
gridsearch_2 = gridsearch_2.fit(X_train_2, y_train)
```

In [108]:

```
results_2 = pd.DataFrame.from_dict(gridsearch_2.cv_results_)
results_2 = results_2.sort_values(['param_n_estimators'])
results_2.head()
```

Out[108]:

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_max_depth | param_n_estimators | params | split0_test_sc |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.375176 | 0.008768 | 0.195809 | 0.000245 | 2 | 10 | {'max_depth': 2, 'n_estimators': 10} | 0.5653 |
| **32** | 0.491254 | 0.023783 | 0.208451 | 0.002876 | 6 | 10 | {'max_depth': 6, 'n_estimators': 10} | 0.6103 |
| **16** | 0.407842 | 0.000513 | 0.201795 | 0.001208 | 4 | 10 | {'max_depth': 4, 'n_estimators': 10} | 0.6062 |
| **40** | 0.597254 | 0.002774 | 0.205626 | 0.000111 | 8 | 10 | {'max_depth': 8, 'n_estimators': 10} | 0.6317 |
| **8** | 0.355334 | 0.005091 | 0.259690 | 0.002112 | 3 | 10 | {'max_depth': 3, 'n_estimators': 10} | 0.6258 |

### 3.3 AUC vs HyperParameter

In [0]:

```
train_auc_2 = results_2['mean_train_score']
test_auc_2 = results_2['mean_test_score']
n_estimator_2 = results_2['param_n_estimators']
```

```
max_depth_2 =results_2['param_max_depth']
```

In [0]:

```
#To plot this into plotly it should be in list
n_estimator_2 = n_estimator_2.to_list()
max_depth_2 = max_depth_2.to_list()
train_auc_2 = train_auc_2.to_list()
test_auc_2 = test_auc_2.to_list()
```

In [111]:

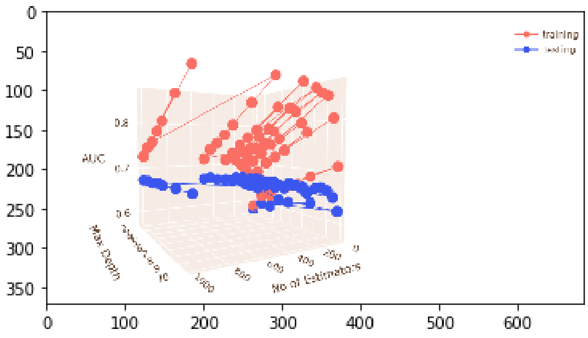
```
# To enable plotly plot in Google Colab
# https://stackoverflow.com/questions/47230817/plotly-notebook-mode-with-google-colaboratory/47230966
def configure_plotly_browser_state():
    import IPython
    display(IPython.core.display.HTML('''
        <script src="/static/components/requirejs/require.js"></script>
        <script>
          requirejs.config({
            paths: {
              base: '/static/base',
              plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
            },
          });
        </script>
        '''))

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)




#To enable plotly plot in this cell
enable_plotly_in_cell()

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=n_estimator_2, y=max_depth_2, z=train_auc_2, name='training')
trace2 = go.Scatter3d(x=n_estimator_2, y=max_depth_2, z=test_auc_2, name='Testing')
data = [trace1, trace2]

layout = go.Layout(
                    scene=dict(
                                xaxis = dict(title='No of Estimators'),
                                yaxis = dict(title='Max Depth'),
                                zaxis = dict(title='AUC'),
                                )
                    )
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='3d-scatter-colorscale')
```

In [120]:

```
import matplotlib.pyplot as plt
import cv2
img_2 = cv2.imread(r'C:\Users\prem.k\Desktop\My projects\Assign\RFGBDT pics\2.PNG')
plt.imshow(img_2)
```

Out[120]:

```
<matplotlib.image.AxesImage at 0x1a503d1d3c8>
```



## 3.4 Modelling with best Parameters

In [0]:

```
best_n_estimators_2 = gridsearch_2.best_params_['n_estimators']
best_max_depth_2 = gridsearch_2.best_params_['max_depth']
print('Best Number of Estimators:', best_n_estimators_2)
print('Best Max depth in a tree:', best_max_depth_2)
```

```
Best Number of Estimators: 500
Best Max depth in a tree: 10
```

In [0]:

```
classifier_withParam_2 = RandomForestClassifier(n_estimators=best_n_estimators_2,
max_depth=best_max_depth_2)
classifier_withParam_2.fit(X_train_2, y_train)
```

Out[0]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=10, max_features='auto',
                       max_leaf_nodes=None, max_samples=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=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

## 3.5 Cross Validation

### 3.5 Cross Validation

In [0]:

```
from sklearn.model_selection import cross_val_score
cv_2 = cross_val_score(estimator=classifier_withParam_2, X=X_train_2, y=y_train, cv=2, scoring='roc
_auc')
```

In [0]:

```
best_auc_2 = cv_2.mean()
print('Best AUC: %4f' %best_auc_2)
```

```
Best AUC: 0.683514
```

### 3.6 ROC curve on train and test data

In [0]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [0]:

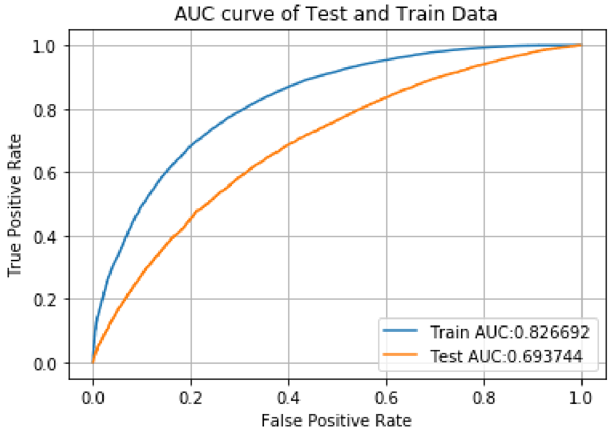
```
y_train_predproba_2 = batch_predict(classifier_withParam_2, X_train_2)
y_test_predproba_2 = batch_predict(classifier_withParam_2, X_test_2)
```

In [0]:

```
from sklearn.metrics import roc_curve, auc
train_fpr_2, train_tpr_2, train_thresh_2 = roc_curve(y_train, y_train_predproba_2)
test_fpr_2, test_tpr_2, test_thresh_2 = roc_curve(y_test, y_test_predproba_2)
```

In [0]:

```
plt.plot(train_fpr_2, train_tpr_2, label='Train AUC:%4f'%auc(train_fpr_2, train_tpr_2))
plt.plot(test_fpr_2, test_tpr_2, label='Test AUC:%4f'%auc(test_fpr_2, test_tpr_2))

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]:

```
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 [0]:

```
best_t = find_best_threshold(train_fpr_2, train_tpr_2, train_thresh_2)
```

```
the maximum tpr*(1-fpr) is : 0.5580240411347649 for threshold 0.844
```

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]:

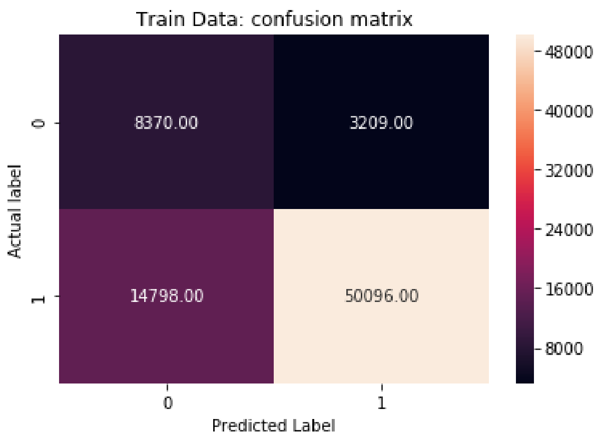
```
y_train_pred_2 = predict_with_threshold(y_train_predproba_2, best_t)
y_test_pred_2 = predict_with_threshold(y_test_predproba_2, best_t)
```

In [0]:

```
from sklearn.metrics import confusion_matrix

cm_train_2 = confusion_matrix(y_train, y_train_pred_2)
cm_test_2 = confusion_matrix(y_test, y_test_pred_2)
```

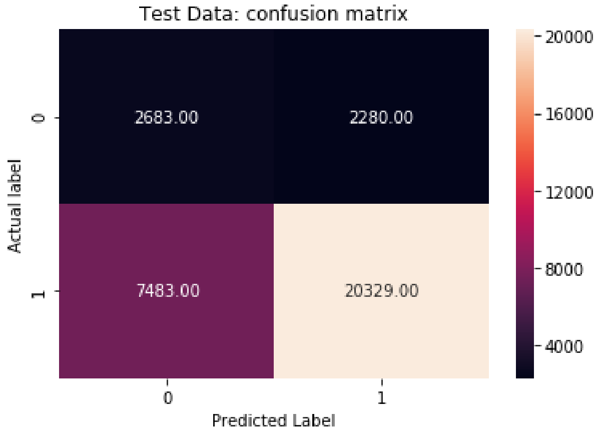
In [0]:

```
sns.heatmap(cm_train_2, annot=True, fmt='.2f')
plt.title('Train Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



In [0]:

```
sns.heatmap(cm_test_2, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



# 4. Set-3

## 4.1 Merging all the features

In [0]:

```
print(X_train_Sstate_responseCoding.shape)
print(X_train_cat_responseCoding.shape)
print(X_train_subcat_responseCoding.shape)
print(X_train_grade_responseCoding.shape)
print(X_train_prefix_responseCoding.shape)
print(X_train_previous.shape)
print(X_train_price.shape)
print(len(X_train_essay_avg_w2v))
print(len(X_train_essay_avg_w2v[1]))
print(len(X_train_title_avg_w2v))
print(len(X_train_title_avg_w2v[1]))
```

```
(76473, 2)
(76473, 2)
(76473, 2)
(76473, 2)
(76473, 2)
(76473, 1)
(76473, 1)
76473
300
76473
300
```

In [0]:

```
X_train_3 = np.hstack((X_train_Sstate_responseCoding, X_train_cat_responseCoding,
X_train_subcat_responseCoding,
                    X_train_grade_responseCoding, X_train_prefix_responseCoding, X_train_previous,
                    X_train_price, X_train_essay_avg_w2v, X_train_title_avg_w2v))

X_test_3 = np.hstack((X_test_Sstate_responseCoding, X_test_cat_responseCoding,
X_test_subcat_responseCoding,
                   X_test_grade_responseCoding, X_test_prefix_responseCoding, X_test_previous, X_te
st_price,
                   X_test_essay_avg_w2v, X_test_title_avg_w2v))
```

In [113]:

```
print(X_train_3.shape)
print(X_test_3.shape)
```

```
(76473, 612)
(32775, 612)
```

## 4.2 GridSearch

In [0]:

```
from sklearn.ensemble import RandomForestClassifier
classifier_3 = RandomForestClassifier()
```

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_3 = GridSearchCV(estimator=classifier_3, param_grid=parameters, scoring='roc_auc', cv=2
, n_jobs=-1, return_train_score=True)
```

In [0]:

```
gridsearch_3 = gridsearch_3.fit(X_train_3, y_train)
```

In [117]:

```
results_3 = pd.DataFrame.from_dict(gridsearch_3.cv_results_)
results_3 = results_3.sort_values(['param_n_estimators'])
results_3.head()
```

Out[117]:

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_max_depth | param_n_estimators | params | split0_test_sc |
|---|---|---|---|---|---|---|---|---|
| **0** | 2.561029 | 0.015046 | 0.120870 | 0.000943 | 2 | 10 | {'max_depth': 2, 'n_estimators': 10} | 0.6279 |
| **32** | 6.375637 | 0.006324 | 0.152876 | 0.001424 | 6 | 10 | {'max_depth': 6, 'n_estimators': 10} | 0.6443 |
| **16** | 4.655614 | 0.032885 | 0.136830 | 0.001086 | 4 | 10 | {'max_depth': 4, 'n_estimators': 10} | 0.6572 |
| **40** | 8.304157 | 0.017712 | 0.173687 | 0.001947 | 8 | 10 | {'max_depth': 8, 'n_estimators': 10} | 0.6452 |
| **8** | 3.498260 | 0.002466 | 0.126324 | 0.001222 | 3 | 10 | {'max_depth': 3, 'n_estimators': 10} | 0.6303 |

## 4.3 AUC vs HyperParameter

In [0]:

```
train_auc_3 = results_3['mean_train_score']
test_auc_3 = results_3['mean_test_score']
n_estimator_3 = results_3['param_n_estimators']
max_depth_3 =results_3['param_max_depth']
```

In [0]:

```
#To plot this into plotly it should be in list
n_estimator_3 = n_estimator_3.to_list()
max_depth_3 = max_depth_3.to_list()
train_auc_3 = train_auc_3.to_list()
test_auc_3 = test_auc_3.to_list()
```

In [120]:

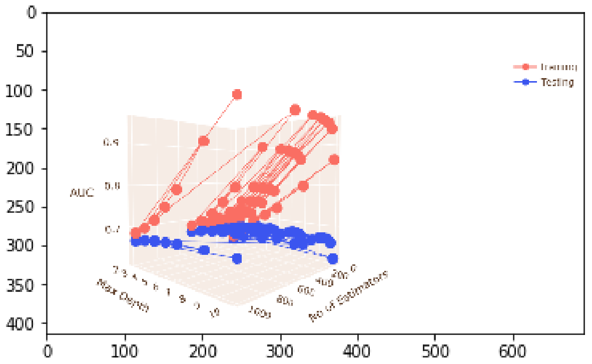
```
# To enable plotly plot in Google Colab
# https://stackoverflow.com/questions/47230817/plotly-notebook-mode-with-google-
colaboratory/47230966
def configure_plotly_browser_state():
    import IPython
    display(IPython.core.display.HTML('''
        <script src="/static/components/requirejs/require.js"></script>
        <script>
          requirejs.config({
            paths: {
              base: '/static/base',
              plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
            },
          });
        </script>
        '''))

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"></s
cript>'''))
    init_notebook_mode(connected=False)



#To enable plotly plot in this cell
enable_plotly_in_cell()

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=n_estimator_3, y=max_depth_3, z=train_auc_3, name='training')
trace2 = go.Scatter3d(x=n_estimator_3, y=max_depth_3, z=test_auc_3, name='Testing')
data = [trace1, trace2]

layout = go.Layout(
                    scene=dict(
                                xaxis = dict(title='No of Estimators'),
                                yaxis = dict(title='Max Depth'),
                                zaxis = dict(title='AUC'),
                                )
                    )
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='3d-scatter-colorscale')
```

In [119]:

```
import matplotlib.pyplot as plt
import cv2
img_3 = cv2.imread(r'C:\Users\prem.k\Desktop\My projects\Assign\RFGBDT pics\3.PNG')
plt.imshow(img_3)
```

Out[119]:

```
<matplotlib.image.AxesImage at 0x1a5010a9588>
```



## 4.4 Modelling with Parameters

In [0]:

```
best_n_estimators_3 = gridsearch_3.best_params_['n_estimators']
best_max_depth_3 = gridsearch_3.best_params_['max_depth']
print('Best Number of Estimators:', best_n_estimators_3)
print('Best Max depth in a tree:', best_max_depth_3)
```

```
Best Number of Estimators: 1000
Best Max depth in a tree: 8
```

In [0]:

```
classifier_withParam_3 = RandomForestClassifier(n_estimators=best_n_estimators_3,
max_depth=best_max_depth_3)
classifier_withParam_3.fit(X_train_3, y_train)
```

Out[0]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=8, max_features='auto',
                       max_leaf_nodes=None, max_samples=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=1000,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

## 4.5 Cross Validation

In [0]:

```
from sklearn.model_selection import cross_val_score
cv_3 = cross_val_score(estimator=classifier_withParam_3, X=X_train_3, y=y_train, cv=2, scoring='roc
_auc')
```

In [0]:

```
best_auc_3 = cv_3.mean()
print('Best AUC: %4f' %best_auc_3)
```

```
Best AUC: 0.686137
```

## 4.6 ROC curve on Train and Test data

In [0]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [0]:

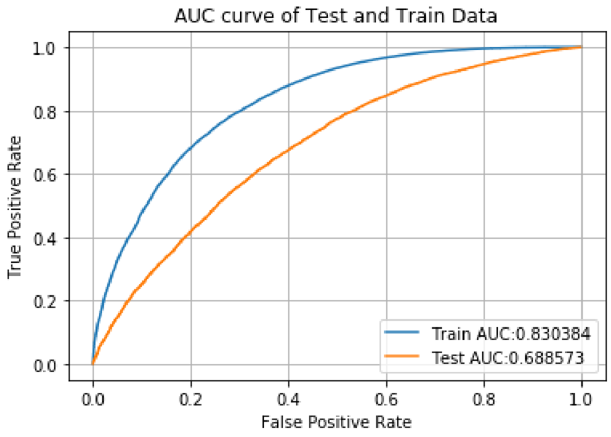
```
y_train_predproba_3 = batch_predict(classifier_withParam_3, X_train_3)
y_test_predproba_3 = batch_predict(classifier_withParam_3, X_test_3)
```

In [0]:

```
from sklearn.metrics import roc_curve, auc
train_fpr_3, train_tpr_3, train_thresh_3 = roc_curve(y_train, y_train_predproba_3)
test_fpr_3, test_tpr_3, test_thresh_3 = roc_curve(y_test, y_test_predproba_3)
```

In [0]:

```
plt.plot(train_fpr_3, train_tpr_3, label='Train AUC:%4f'%auc(train_fpr_3, train_tpr_3))
plt.plot(test_fpr_3, test_tpr_3, label='Test AUC:%4f'%auc(test_fpr_3, test_tpr_3))

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()
```



## 4.7 Confusion Matrix

In [0]:

```
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 [0]:

```
best_t = find_best_threshold(train_fpr_3, train_tpr_3, train_thresh_3)
```

```
the maximum tpr*(1-fpr) is : 0.56202614983841 for threshold 0.836
```

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]:

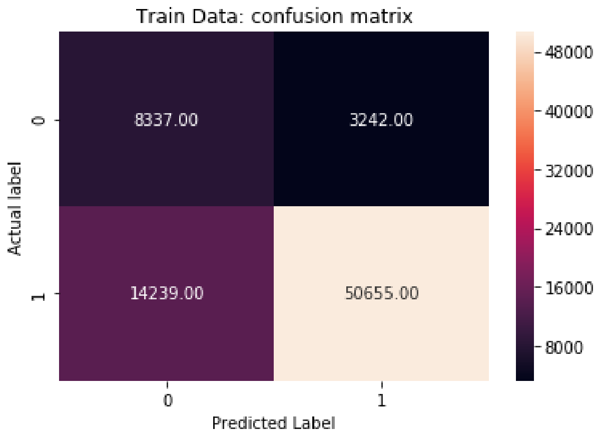
```
y_train_pred_3 = predict_with_threshold(y_train_predproba_3, best_t)
y_test_pred_3 = predict_with_threshold(y_test_predproba_3, best_t)
```

In [0]:

```
from sklearn.metrics import confusion_matrix

cm_train_3 = confusion_matrix(y_train, y_train_pred_3)
cm_test_3 = confusion_matrix(y_test, y_test_pred_3)
```

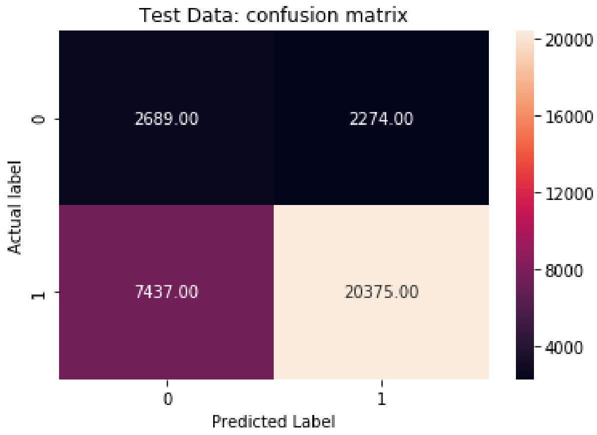
In [0]:

```
sns.heatmap(cm_train_3, annot=True, fmt='.2f')
plt.title('Train Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



In [0]:

```
sns.heatmap(cm_test_3, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



# 5. Set - 4

## 5.1 Merging all the features

In [0]:

```
from scipy.sparse import hstack
X_train_4 = np.hstack((X_train_Sstate_responseCoding, X_train_cat_responseCoding, 
X_train_subcat_responseCoding,
                    X_train_grade_responseCoding, X_train_prefix_responseCoding, X_train_previous,
                    X_train_price, X_train_essay_tfidf_w2v, X_train_title_tfidf_w2v))

X_test_4 = np.hstack((X_test_Sstate_responseCoding, X_test_cat_responseCoding, 
X_test_subcat_responseCoding,
                   X_test_grade_responseCoding, X_test_prefix_responseCoding, X_test_previous, X_te
st_price,
                   X_test_essay_tfidf_w2v, X_test_title_tfidf_w2v))
```

In [122]:

```
print(X_train_4.shape)
print(X_test_4.shape)
```

```
(76473, 612)
(32775, 612)
```

## 5.2 GridSearch

In [0]:

```
from sklearn.ensemble import RandomForestClassifier
classifier_4 = RandomForestClassifier()
```

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_4 = GridSearchCV(estimator=classifier_4, param_grid=parameters, scoring='roc_auc', cv=2
, n_jobs=-1, return_train_score=True)
```

In [0]:

```
gridsearch_4 = gridsearch_4.fit(X_train_4, y_train)
```

In [127]:

```
results_4 = pd.DataFrame.from_dict(gridsearch_4.cv_results_)
results_4 = results_4.sort_values(['param_n_estimators'])
results_4.head()
```

Out[127]:

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_max_depth | param_n_estimators | params | split0_test_sc |
|---|---|---|---|---|---|---|---|---|
| **0** | 2.622657 | 0.003415 | 0.124250 | 0.001381 | 2 | 10 | {'max_depth': 2, 'n_estimators': 10} | 0.6320 |
| **32** | 6.410497 | 0.017726 | 0.153608 | 0.001991 | 6 | 10 | {'max_depth': 6, 'n_estimators': 10} | 0.6526 |
| **16** | 4.588486 | 0.013861 | 0.166723 | 0.030836 | 4 | 10 | {'max_depth': 4, 'n_estimators': 10} | 0.6516 |
| **40** | 8.138636 | 0.076318 | 0.177458 | 0.010768 | 8 | 10 | {'max_depth': 8, 'n_estimators': 10} | 0.6499 |
| **8** | 3.564604 | 0.014185 | 0.130388 | 0.002953 | 3 | 10 | {'max_depth': 3, 'n_estimators': 10} | 0.6541 |

### 5.3 AUC vs HyperParameter

In [0]:

```
train_auc_4 = results_4['mean_train_score']
test_auc_4 = results_4['mean_test_score']
n_estimator_4 = results_4['param_n_estimators']
max_depth_4 =results_4['param_max_depth']
```

In [0]:

```
#To plot this into plotly it should be in list
n_estimator_4 = n_estimator_4.to_list()
max_depth_4 = max_depth_4.to_list()
train_auc_4 = train_auc_4.to_list()
test_auc_4 = test_auc_4.to_list()
```

In [131]:

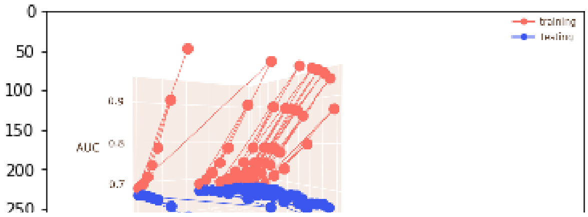
```
# To enable plotly plot in Google Colab
# https://stackoverflow.com/questions/47230817/plotly-notebook-mode-with-google-
colaboratory/47230966
def configure_plotly_browser_state():
    import IPython
    display(IPython.core.display.HTML('''
        <script src="/static/components/requirejs/require.js"></script>
        <script>
          requirejs.config({
            paths: {
              base: '/static/base',
              plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
            },
          });
        </script>
        '''))

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"></s
    display(IPython.core.display.HTML('''<script src="/static/components/requirejs/require.js"></s
cript>'''))
    init_notebook_mode(connected=False)



#To enable plotly plot in this cell
enable_plotly_in_cell()

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=n_estimator_4, y=max_depth_4, z=train_auc_4, name='training')
trace2 = go.Scatter3d(x=n_estimator_4, y=max_depth_4, z=test_auc_4, name='Testing')
data = [trace1, trace2]

layout = go.Layout(
                    scene=dict(
                                xaxis = dict(title='No of Estimators'),
                                yaxis = dict(title='Max Depth'),
                                zaxis = dict(title='AUC'),
                                )
                    )
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='3d-scatter-colorscale')
```

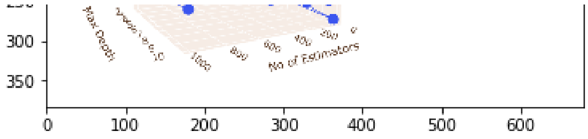
In [117]:

```
import cv2
import matplotlib.pyplot as plt
img_4 = cv2.imread(r'C:\Users\prem.k\Desktop\My projects\Assign\RFGBDT pics/4.PNG')
plt.imshow(img_4)
```

Out[117]:

```
<matplotlib.image.AxesImage at 0x1a5010fe128>
```





## 5.4 Modelling with Parameters

In [132]:

```
best_n_estimators_4 = gridsearch_4.best_params_['n_estimators']
best_max_depth_4 = gridsearch_4.best_params_['max_depth']
print('Best Number of Estimators:', best_n_estimators_4)
print('Best Max depth in a tree:', best_max_depth_4)
```

```
Best Number of Estimators: 1000
Best Max depth in a tree: 8
```

In [133]:

```
classifier_withParam_4 = RandomForestClassifier(n_estimators=best_n_estimators_4,
max_depth=best_max_depth_4)
classifier_withParam_4.fit(X_train_4, y_train)
```

Out[133]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=8, max_features='auto',
                       max_leaf_nodes=None, max_samples=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=1000,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

## 5.5 Cross Validation

In [0]:

```
from sklearn.model_selection import cross_val_score
cv_4 = cross_val_score(estimator=classifier_withParam_4, X=X_train_4, y=y_train, cv=2, scoring='roc
_auc')
```

In [135]:

```
best_auc_4 = cv_4.mean()
print('Best AUC: %4f' %best_auc_4)
```

```
Best AUC: 0.686826
```

## 5.6 ROC curve on Train and Test data

In [0]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [0]:

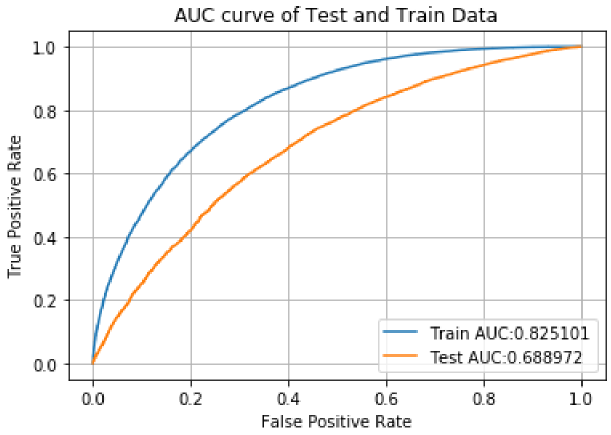
```
y_train_predproba_4 = batch_predict(classifier_withParam_4, X_train_4)
y_test_predproba_4 = batch_predict(classifier_withParam_4, X_test_4)
```

In [0]:

```
from sklearn.metrics import roc_curve, auc
train_fpr_4, train_tpr_4, train_thresh_4 = roc_curve(y_train, y_train_predproba_4)
test_fpr_4, test_tpr_4, test_thresh_4 = roc_curve(y_test, y_test_predproba_4)
```

In [139]:

```
plt.plot(train_fpr_4, train_tpr_4, label='Train AUC:%4f'%auc(train_fpr_4, train_tpr_4))
plt.plot(test_fpr_4, test_tpr_4, label='Test AUC:%4f'%auc(test_fpr_4, test_tpr_4))

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()
```



## 5.7 Confusion Matrix

In [0]:

```
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 [141]:

```
best_t = find_best_threshold(train_fpr_4, train_tpr_4, train_thresh_4)
```

```
the maximum tpr*(1-fpr) is : 0.5558069236310891 for threshold 0.832
```

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]:

```
y_train_pred_4 = predict_with_threshold(y_train_predproba_4, best_t)
```

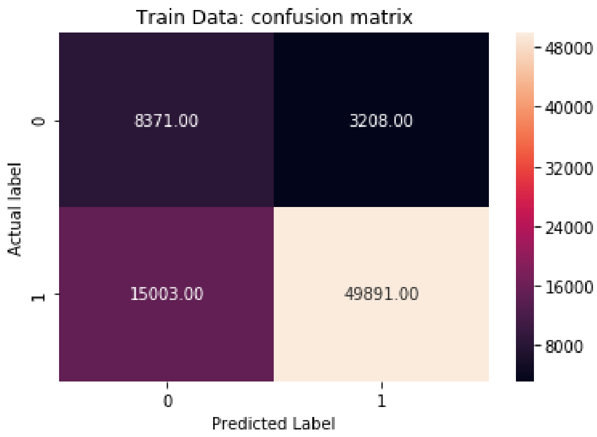
```
y_train_pred_4 = predict_with_threshold(y_train_predproba_4, best_t)
y_test_pred_4 = predict_with_threshold(y_test_predproba_4, best_t)
```

In [0]:

```
from sklearn.metrics import confusion_matrix

cm_train_4 = confusion_matrix(y_train, y_train_pred_4)
cm_test_4 = confusion_matrix(y_test, y_test_pred_4)
```

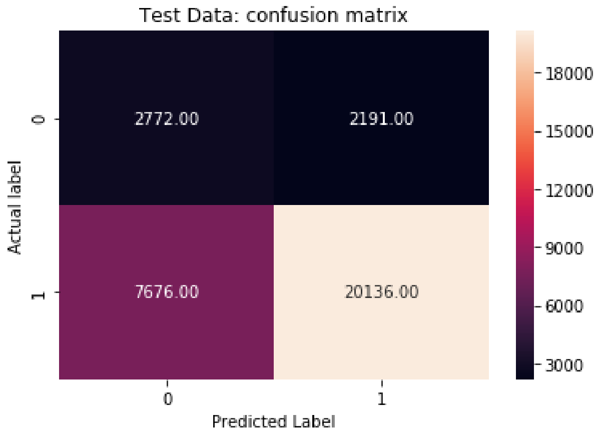
In [145]:

```
sns.heatmap(cm_train_4, annot=True, fmt='.2f')
plt.title('Train Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



In [146]:

```
sns.heatmap(cm_test_4, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



# Model -2 : GRADIENT BOOSTING

## 6. Set -5

### 6.1 Merging all together

In [0]:

In [0]:

```
from scipy.sparse import hstack
X_train_5 = hstack((X_train_Sstate_responseCoding, X_train_cat_responseCoding, 
X_train_subcat_responseCoding,
                    X_train_grade_responseCoding, X_train_prefix_responseCoding, X_train_essay_bow, 

                    X_train_title_bow, X_train_previous, X_train_price)).tocsr()

X_test_5 = hstack((X_test_Sstate_responseCoding, X_test_cat_responseCoding, 
X_test_subcat_responseCoding,
                   X_test_grade_responseCoding, X_test_prefix_responseCoding, X_test_essay_bow, X_t
est_title_bow,
                   X_test_previous, X_test_price)).tocsr()
```

In [83]:

```
print(X_train_5.shape)
print(X_test_5.shape)
```

```
(76473, 17185)
(32775, 17185)
```

**Note:**

- Since GBDT is computationally expensive i will take 30k points for training.

In [0]:

```
X_train_5 = X_train_5[0:30000, :]
y_train = y_train[0:30000, :]
```

In [92]:

```
print(X_train_5.shape)
print(y_train.shape)
```

```
(30000, 17185)
(30000, 1)
```

## 6.2 GridSearch

In [0]:

```
from sklearn.ensemble import GradientBoostingClassifier
classifier_5 = GradientBoostingClassifier()
```

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_5 = GridSearchCV(estimator=classifier_5, param_grid=parameters, scoring='roc_auc', cv=2
, n_jobs=-1, return_train_score=True)
```

In [0]:

```
gridsearch_5 = gridsearch_5.fit(X_train_5, y_train)
```

In [94]:

```
results_5 = pd.DataFrame.from_dict(gridsearch_5.cv_results_)
```

```
results_5 = results_5.sort_values(['param_n_estimators'])
results_5.head()
```

Out[94]:

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_max_depth | param_n_estimators | params | split0_test_sc |
|---|---|---|---|---|---|---|---|---|
| **0** | 2.352499 | 0.017494 | 0.030329 | 0.000612 | 2 | 10 | {'max_depth': 2, 'n_estimators': 10} | 0.6465 |
| **32** | 8.920089 | 0.248130 | 0.036363 | 0.003004 | 6 | 10 | {'max_depth': 6, 'n_estimators': 10} | 0.6607 |
| **16** | 4.916665 | 0.137015 | 0.032439 | 0.000215 | 4 | 10 | {'max_depth': 4, 'n_estimators': 10} | 0.6602 |
| **40** | 14.392339 | 0.453486 | 0.037717 | 0.001456 | 8 | 10 | {'max_depth': 8, 'n_estimators': 10} | 0.6567 |
| **8** | 3.427374 | 0.028215 | 0.030277 | 0.000902 | 3 | 10 | {'max_depth': 3, 'n_estimators': 10} | 0.6546 |

## 6.3 AUC vs HyperParameter

In [0]:

```
train_auc_5 = results_5['mean_train_score']
test_auc_5 = results_5['mean_test_score']
n_estimator_5 = results_5['param_n_estimators']
max_depth_5 =results_5['param_max_depth']
```

In [0]:

```
#To plot this into plotly it should be in list
n_estimator_5 = n_estimator_5.to_list()
max_depth_5 = max_depth_5.to_list()
train_auc_5 = train_auc_5.to_list()
test_auc_5 = test_auc_5.to_list()
```

In [97]:

```
# To enable plotly plot in Google Colab
# https://stackoverflow.com/questions/47230817/plotly-notebook-mode-with-google-
colaboratory/47230966
def configure_plotly_browser_state():
    import IPython
    display(IPython.core.display.HTML('''
        <script src="/static/components/requirejs/require.js"></script>
        <script>
          requirejs.config({
            paths: {
              base: '/static/base',
              plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
            },
          });
        </script>
        '''))

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"></s
cript>'''))
    init_notebook_mode(connected=False)
```

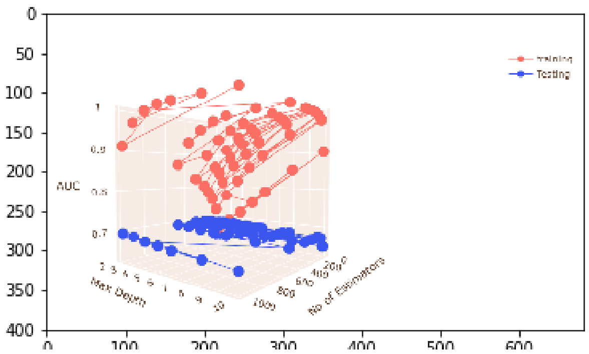
```
#To enable plotly plot in this cell
enable_plotly_in_cell()

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=n_estimator_5, y=max_depth_5, z=train_auc_5, name='training')
trace2 = go.Scatter3d(x=n_estimator_5, y=max_depth_5, z=test_auc_5, name='Testing')
data = [trace1, trace2]

layout = go.Layout(
                    scene=dict(
                            xaxis = dict(title='No of Estimators'),
                            yaxis = dict(title='Max Depth'),
                            zaxis = dict(title='AUC'),
                            )
                    )
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='3d-scatter-colorscale')
```

In [116]:

```
import cv2
import matplotlib.pyplot as plt
img_5 = cv2.imread(r'C:\Users\prem.k\Desktop\My projects\Assign\RFGBDT pics/5.PNG')
plt.imshow(img_5)
```

Out[116]:

```
<matplotlib.image.AxesImage at 0x1a5011299e8>
```



### 6.4 Modelling with Parameters

In [98]:

```
best_n_estimators_5 = gridsearch_5.best_params_['n_estimators']
best_max_depth_5 = gridsearch_5.best_params_['max_depth']
print('Best Number of Estimators:', best_n_estimators_5)
print('Best Max depth in a tree:', best_max_depth_5)
```

```
Best Number of Estimators: 1000
Best Max depth in a tree: 2
```

In [100]:

```
classifier_withParam_5 = GradientBoostingClassifier(n_estimators=best_n_estimators_5, max_depth=be
st_max_depth_5)
classifier_withParam_5.fit(X_train_5, y_train)
```

Out[100]:

```
GradientBoostingClassifier(ccp_alpha=0.0, 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=1000,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

### 6.5 Cross Validation

In [0]:

```
from sklearn.model_selection import cross_val_score
cv_5 = cross_val_score(estimator=classifier_withParam_5, X=X_train_5, y=y_train, cv=2, scoring='roc
_auc')
```

In [102]:

```
best_auc_5 = cv_5.mean()
print('Best AUC: %4f' %best_auc_5)
```

```
Best AUC: 0.693973
```

### 6.6 ROC curve on Train and Test data

In [0]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [0]:

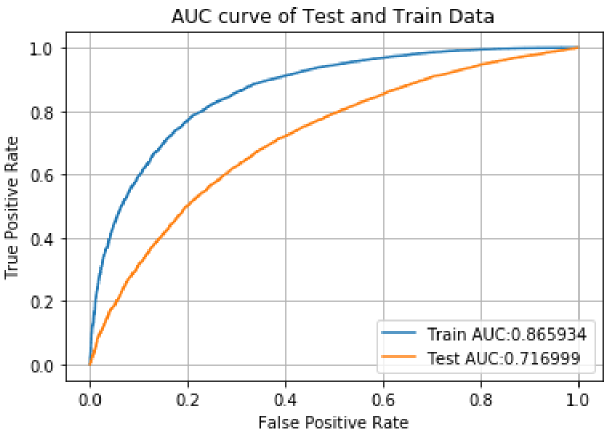
```
y_train_predproba_5 = batch_predict(classifier_withParam_5, X_train_5)
y_test_predproba_5 = batch_predict(classifier_withParam_5, X_test_5)
```

In [0]:

```
from sklearn.metrics import roc_curve, auc
train_fpr_5, train_tpr_5, train_thresh_5 = roc_curve(y_train, y_train_predproba_5)
test_fpr_5, test_tpr_5, test_thresh_5 = roc_curve(y_test, y_test_predproba_5)
```

In [106]:

```
plt.plot(train_fpr_5, train_tpr_5, label='Train AUC:%4f'%auc(train_fpr_5, train_tpr_5))
plt.plot(test_fpr_5, test_tpr_5, label='Test AUC:%4f'%auc(test_fpr_5, test_tpr_5))

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()
```



## 6.7 Confusion Matrix

In [0]:

```
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 [108]:

```
best_t = find_best_threshold(train_fpr_5, train_tpr_5, train_thresh_5)
```

```
the maximum tpr*(1-fpr) is : 0.6203424463828772 for threshold 0.82
```

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]:

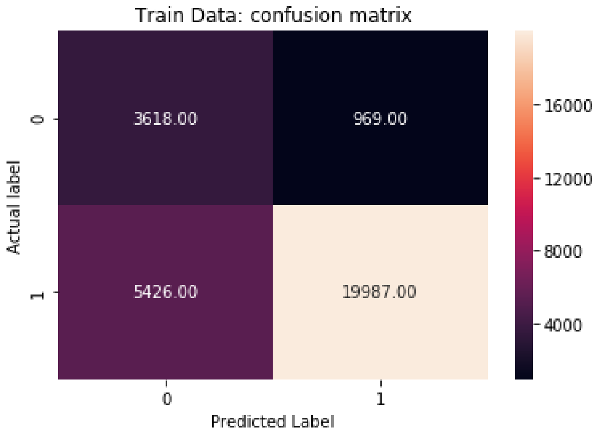
```
y_train_pred_5 = predict_with_threshold(y_train_predproba_5, best_t)
y_test_pred_5 = predict_with_threshold(y_test_predproba_5, best_t)
```

In [0]:

```
from sklearn.metrics import confusion_matrix

cm_train_5 = confusion_matrix(y_train, y_train_pred_5)
cm_test_5 = confusion_matrix(y_test, y_test_pred_5)
```

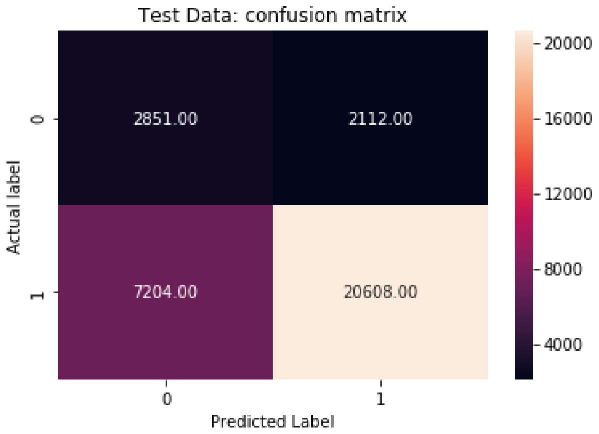
In [112]:

```
sns.heatmap(cm_train_5, annot=True, fmt='.2f')
plt.title('Train Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



In [113]:

```
sns.heatmap(cm_test_5, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



# 7. Set -6

## 7.1 Merging all the features

In [78]:

```
from scipy.sparse import hstack
X_train_6 = hstack((X_train_Sstate_responseCoding, X_train_cat_responseCoding,
X_train_subcat_responseCoding,
                    X_train_grade_responseCoding, X_train_prefix_responseCoding,
X_train_essay_tfidf,
                    X_train_title_tfidf, X_train_previous, X_train_price)).tocsr()

X_test_6 = hstack((X_test_Sstate_responseCoding, X_test_cat_responseCoding,
X_test_subcat_responseCoding,
                   X_test_grade_responseCoding, X_test_prefix_responseCoding, X_test_essay_tfidf,
                   X_test_title_tfidf, X_test_previous, X_test_price)).tocsr()
```

In [79]:

```
print(X_train_6.shape)
print(X_test_6.shape)
```

```
(76473, 17185)
(32775, 17185)
```

In [80]:

```
X_train_6 = X_train_6[0:30000, :]
y_train = y_train[0:30000, :]
print(X_train_6.shape)
print(y_train.shape)
```

```
(30000, 17185)
(30000, 1)
```

## 7.2 Grid Search

In [81]:

```
from sklearn.ensemble import GradientBoostingClassifier
classifier_6 = GradientBoostingClassifier()
```

In [93]:

```
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_6 = GridSearchCV(estimator=classifier_6, param_grid=parameters, scoring='roc_auc', cv=2
, n_jobs=-1, return_train_score=True)
```

In [94]:

```
gridsearch_6 = gridsearch_6.fit(X_train_6, y_train)
```

In [95]:

```
results_6 = pd.DataFrame.from_dict(gridsearch_6.cv_results_)
results_6 = results_6.sort_values(['param_n_estimators'])
results_6.head()
```

Out[95]:

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_max_depth | param_n_estimators | params | split0_test_sc |
|---|---|---|---|---|---|---|---|---|
| 0 | 5.045022 | 0.248844 | 0.036900 | 3.989458e-03 | 2 | 10 | {'max_depth': 2, 'n_estimators': 10} | 0.6482 |
| 32 | 14.668584 | 0.453811 | 0.034909 | 8.344650e-07 | 6 | 10 | {'max_depth': 6, 'n_estimators': 10} | 0.6646 |
| [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | 4 | 10 | {'max_depth': 4, | [illegible] |

|  | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_max_depth | param_n_estimators | params | split0_test_sc |
|---|---|---|---|---|---|---|---|---|
| 16 | 9.111189 | 0.070314 | 0.032912 | 3.576279e-07 | 4 | 10 | 'n_estimators': 10} | 0.6676 |
| 40 | 21.096567 | 0.665221 | 0.039893 | 1.994610e-03 | 8 | 10 | {'max_depth': 8, 'n_estimators': 10} | 0.6628 |
| 8 | 6.748965 | 0.004986 | 0.031914 | 8.344650e-07 | 3 | 10 | {'max_depth': 3, 'n_estimators': 10} | 0.6619 |

## 7.3 AUC vs HyperParameters

In [96]:

```
train_auc_6 = results_6['mean_train_score']
test_auc_6 = results_6['mean_test_score']
n_estimator_6 = results_6['param_n_estimators']
max_depth_6 =results_6['param_max_depth']
```

In [97]:

```
#To plot this into plotly it should be in list
n_estimator_6 = n_estimator_6.to_list()
max_depth_6 = max_depth_6.to_list()
train_auc_6 = train_auc_6.to_list()
test_auc_6 = test_auc_6.to_list()
```

In [98]:

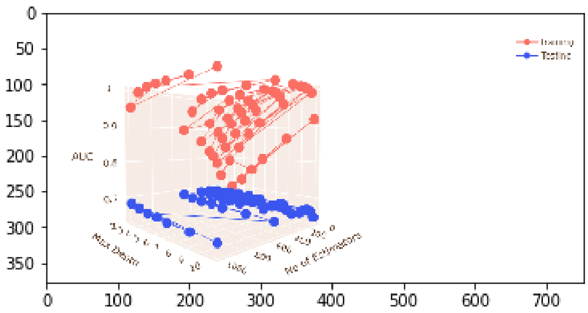
```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=n_estimator_6, y=max_depth_6, z=train_auc_6, name='training')
trace2 = go.Scatter3d(x=n_estimator_6, y=max_depth_6, z=test_auc_6, name='Testing')
data = [trace1, trace2]

layout = go.Layout(
                    scene=dict(
                            xaxis = dict(title='No of Estimators'),
                            yaxis = dict(title='Max Depth'),
                            zaxis = dict(title='AUC'),
                            )
                    )
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='3d-scatter-colorscale')
```

In [115]:

```
import cv2
import matplotlib.pyplot as plt
img_6 = cv2.imread(r'C:\Users\prem.k\Desktop\My projects\Assign\RFGBDT pics/6.PNG')
plt.imshow(img_6)
```

Out[115]:

```
<matplotlib.image.AxesImage at 0x1a501159278>
```



## 7.4 Modelling with Parameters

In [99]:

```
best_n_estimators_6 = gridsearch_6.best_params_['n_estimators']
best_max_depth_6 = gridsearch_6.best_params_['max_depth']
print('Best Number of Estimators:', best_n_estimators_6)
print('Best Max depth in a tree:', best_max_depth_6)
```

```
Best Number of Estimators: 300
Best Max depth in a tree: 2
```

In [102]:

```
from sklearn.ensemble import GradientBoostingClassifier
classifier_withParam_6 = GradientBoostingClassifier(n_estimators=best_n_estimators_6, max_depth=be
st_max_depth_6)
classifier_withParam_6.fit(X_train_6, y_train)
```

Out[102]:

```
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=300,
                           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)
```

## 7.5 Cross Validation

In [103]:

```
from sklearn.model_selection import cross_val_score
cv_6 = cross_val_score(estimator=classifier_withParam_6, X=X_train_6, y=y_train, cv=2, scoring='roc
_auc')
```

In [104]:

```
best_auc_6 = cv_6.mean()
print('Best AUC: %4f' %best_auc_6)
```

```
Best AUC: 0.687575
```

### 7.6 ROC curve on Train and Test data

In [105]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [106]:

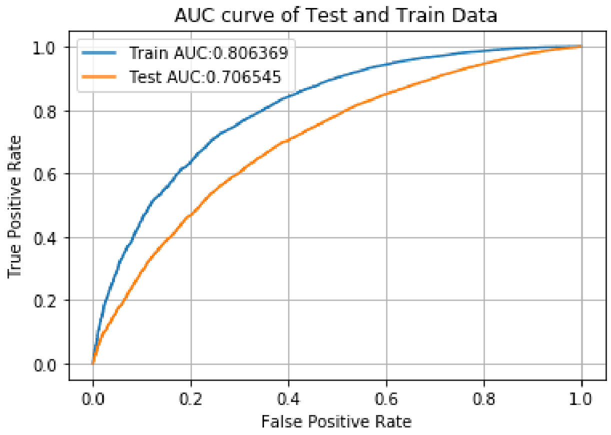
```
y_train_predproba_6 = batch_predict(classifier_withParam_6, X_train_6)
y_test_predproba_6 = batch_predict(classifier_withParam_6, X_test_6)
```

In [107]:

```
from sklearn.metrics import roc_curve, auc
train_fpr_6, train_tpr_6, train_thresh_6 = roc_curve(y_train, y_train_predproba_6)
test_fpr_6, test_tpr_6, test_thresh_6 = roc_curve(y_test, y_test_predproba_6)
```

In [108]:

```
plt.plot(train_fpr_6, train_tpr_6, label='Train AUC:%4f'%auc(train_fpr_6, train_tpr_6))
plt.plot(test_fpr_6, test_tpr_6, label='Test AUC:%4f'%auc(test_fpr_6, test_tpr_6))

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()
```



### 7.7 Confusion Matrix

In [109]:

```
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))
    print('the maximum tpr*(1-fpr) is :', max(tpr*(1-fpr)), 'for threshold', np.round(t,3))
    return t
```

In [110]:

```
best_t = find_best_threshold(train_fpr_6, train_tpr_6, train_thresh_6)
```

```
the maximum tpr*(1-fpr) is : 0.5367026454817301 for threshold 0.835
```

In [111]:

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

    return predictions
```

In [112]:

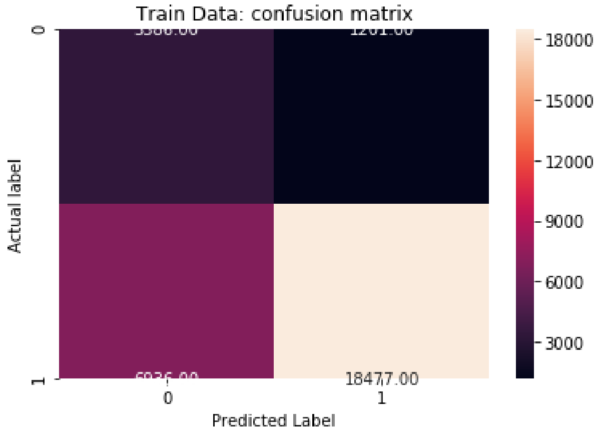
```
y_train_pred_6 = predict_with_threshold(y_train_predproba_6, best_t)
y_test_pred_6 = predict_with_threshold(y_test_predproba_6, best_t)
```

In [114]:

```
from sklearn.metrics import confusion_matrix

cm_train_6 = confusion_matrix(y_train, y_train_pred_6)
cm_test_6 = confusion_matrix(y_test, y_test_pred_6)
```

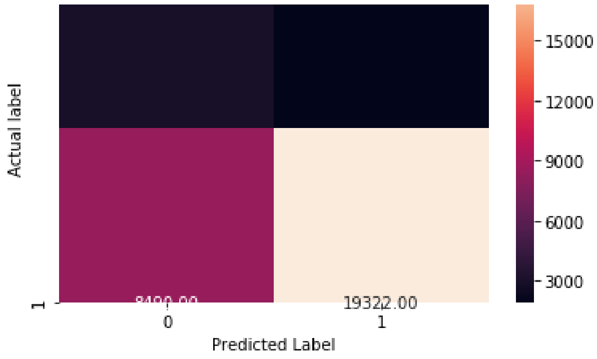
In [115]:

```
sns.heatmap(cm_train_6, annot=True, fmt='.2f')
plt.title('Train Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



In [116]:

```
sns.heatmap(cm_test_6, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



# 8. Set -7

## 8.1 Merging all features

In [78]:

```
X_train_7 = np.hstack((X_train_Sstate_responseCoding, X_train_cat_responseCoding, 
X_train_subcat_responseCoding,
                    X_train_grade_responseCoding, X_train_prefix_responseCoding, 
X_train_essay_avg_w2v,
                    X_train_title_avg_w2v, X_train_previous, X_train_price))

X_test_7 = np.hstack((X_test_Sstate_responseCoding, X_test_cat_responseCoding, 
X_test_subcat_responseCoding,
                   X_test_grade_responseCoding, X_test_prefix_responseCoding, X_test_essay_avg_w2v,

                   X_test_title_avg_w2v, X_test_previous, X_test_price))
```

In [79]:

```
print(X_train_7.shape)
print(X_test_7.shape)
```

```
(76473, 612)
(32775, 612)
```

In [80]:

```
from scipy import sparse
X_train_7 = sparse.csr_matrix(X_train_7)
X_test_7 = sparse.csr_matrix(X_test_7)
```

In [81]:

```
X_train_7 = X_train_7[0:20000, :]
y_train = y_train[0:20000, :]
print(X_train_7.shape)
print(y_train.shape)
```

```
(20000, 612)
(20000, 1)
```

## 8.2 Grid Search

In [82]:

```
from sklearn.ensemble import GradientBoostingClassifier
classifier_7 = GradientBoostingClassifier()
```

In [83]:

```
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_7 = GridSearchCV(estimator=classifier_7, param_grid=parameters, scoring='roc_auc', cv=2
, n_jobs=-1, return_train_score=True)
```

In [84]:

```
gridsearch_7 = gridsearch_7.fit(X_train_7, y_train)
```

In [85]:

```
results_7 = pd.DataFrame.from_dict(gridsearch_7.cv_results_)
results_7 = results_7.sort_values(['param_n_estimators'])
results_7.head()
```

Out[85]:

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_max_depth | param_n_estimators | params | split0_test_sc |
|---|---|---|---|---|---|---|---|---|
| 0 | 15.790856 | 0.227502 | 0.070262 | 0.000062 | 2 | 10 | {'max_depth': 2, 'n_estimators': 10} | 0.6562 |
| 32 | 49.916450 | 0.328053 | 0.072494 | 0.002306 | 6 | 10 | {'max_depth': 6, 'n_estimators': 10} | 0.6597 |
| 16 | 31.528388 | 0.064943 | 0.075218 | 0.005008 | 4 | 10 | {'max_depth': 4, 'n_estimators': 10} | 0.6673 |
| 40 | 73.440738 | 1.357955 | 0.076250 | 0.003981 | 8 | 10 | {'max_depth': 8, 'n_estimators': 10} | 0.6488 |
| 8 | 23.181195 | 0.050044 | 0.070249 | 0.000043 | 3 | 10 | {'max_depth': 3, 'n_estimators': 10} | 0.6670 |

## 8.3 AUC vs HyperParameters

In [86]:

```
train_auc_7 = results_7['mean_train_score']
test_auc_7 = results_7['mean_test_score']
n_estimator_7 = results_7['param_n_estimators']
max_depth_7 =results_7['param_max_depth']
```

In [87]:

```
#To plot this into plotly it should be in list
n_estimator_7 = n_estimator_7.to_list()
max_depth_7 = max_depth_7.to_list()
train_auc_7 = train_auc_7.to_list()
test_auc_7 = test_auc_7.to_list()
```

In [89]:

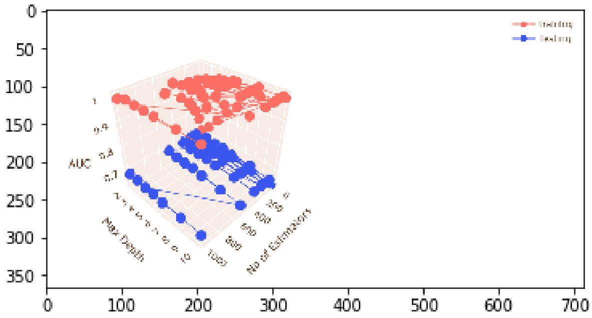
```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=n_estimator_7, y=max_depth_7, z=train_auc_7, name='training')
trace2 = go.Scatter3d(x=n_estimator_7, y=max_depth_7, z=test_auc_7, name='Testing')
data = [trace1, trace2]

layout = go.Layout(
                    scene=dict(
                    scene=dict(
                                xaxis = dict(title='No of Estimators'),
                                yaxis = dict(title='Max Depth'),
                                zaxis = dict(title='AUC'),
                                )
                    )
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='3d-scatter-colorscale')
```

In [114]:

```
import cv2
import matplotlib.pyplot as plt
img_7 = cv2.imread(r'C:\Users\prem.k\Desktop\My projects\Assign\RFGBDT pics/7.PNG')
plt.imshow(img_7)
```

Out[114]:

```
<matplotlib.image.AxesImage at 0x1a501185908>
```



## 8.4 Modelling with Parameters

In [90]:

```
best_n_estimators_7 = gridsearch_7.best_params_['n_estimators']
best_max_depth_7 = gridsearch_7.best_params_['max_depth']
print('Best Number of Estimators:', best_n_estimators_7)
print('Best Max depth in a tree:', best_max_depth_7)
```

```
Best Number of Estimators: 150
Best Max depth in a tree: 2
```

In [91]:

```
classifier_withParam_7 = GradientBoostingClassifier(n_estimators=best_n_estimators_7, max_depth=be
st_max_depth_7)
classifier_withParam_7.fit(X_train_7, y_train)
```

Out[91]:

```
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=150,
                           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)
```

## 8.5 Cross Validation

In [92]:

```
from sklearn.model_selection import cross_val_score
cv_7 = cross_val_score(estimator=classifier_withParam_7, X=X_train_7, y=y_train, cv=2, scoring='roc
_auc')
```

In [93]:

```
best_auc_7 = cv_7.mean()
print('Best AUC: %4f' %best_auc_7)
```

```
Best AUC: 0.693946
```

## 8.6 ROC curve on Train and Test data

In [94]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [95]:

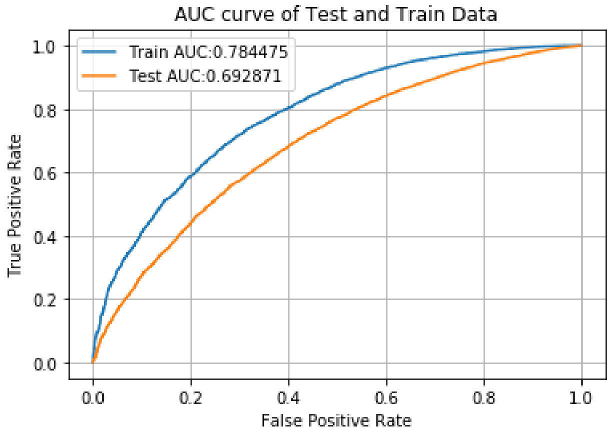
```
y_train_predproba_7 = batch_predict(classifier_withParam_7, X_train_7)
y_test_predproba_7 = batch_predict(classifier_withParam_7, X_test_7)
```

In [96]:

```
from sklearn.metrics import roc_curve, auc
train_fpr_7, train_tpr_7, train_thresh_7 = roc_curve(y_train, y_train_predproba_7)
test_fpr_7, test_tpr_7, test_thresh_7 = roc_curve(y_test, y_test_predproba_7)
```

In [97]:

```
plt.plot(train_fpr_7, train_tpr_7, label='Train AUC:%4f'%auc(train_fpr_7, train_tpr_7))
plt.plot(test_fpr_7, test_tpr_7, label='Test AUC:%4f'%auc(test_fpr_7, test_tpr_7))
plt.plot(test_fpr_7, test_tpr_7, label='Test AUC:%4f'%auc(test_fpr_7, test_tpr_7))

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()
```



## 7.7 Confusion Matrix

In [98]:

```
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 [99]:

```
best_t = find_best_threshold(train_fpr_7, train_tpr_7, train_thresh_7)
```

```
the maximum tpr*(1-fpr) is : 0.5042110647849568 for threshold 0.836
```

In [100]:

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

    return predictions
```

In [101]:

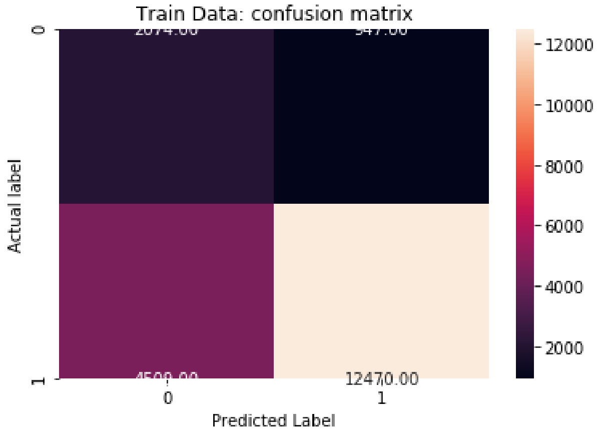
```
y_train_pred_7 = predict_with_threshold(y_train_predproba_7, best_t)
y_test_pred_7 = predict_with_threshold(y_test_predproba_7, best_t)
```

In [102]:

```
from sklearn.metrics import confusion_matrix

cm_train_7 = confusion_matrix(y_train, y_train_pred_7)
cm_test_7 = confusion_matrix(y_test, y_test_pred_7)
```

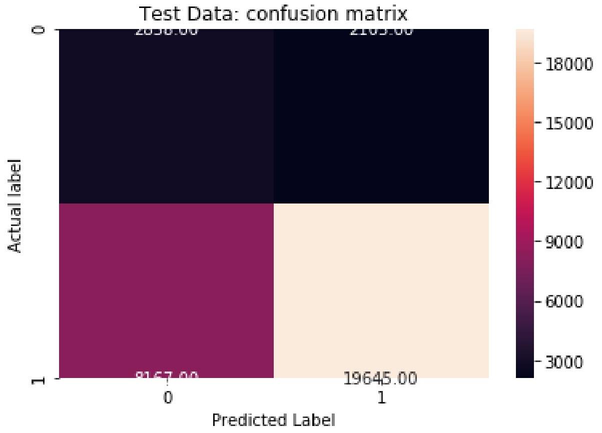
In [104]:

```
sns.heatmap(cm_train_7, annot=True, fmt='.2f')
plt.title('Train Data: confusion matrix')
plt.title('Train Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



In [105]:

```
sns.heatmap(cm_test_7, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



# 9. Set -8

## 9.1 Merging all features

In [79]:

```
X_train_8 = np.hstack((X_train_Sstate_responseCoding, X_train_cat_responseCoding,
X_train_subcat_responseCoding,
                    X_train_grade_responseCoding, X_train_prefix_responseCoding,
X_train_essay_tfidf_w2v,
                    X_train_title_tfidf_w2v, X_train_previous, X_train_price))

X_test_8 = np.hstack((X_test_Sstate_responseCoding, X_test_cat_responseCoding,
X_test_subcat_responseCoding,
                   X_test_grade_responseCoding, X_test_prefix_responseCoding,
X_test_essay_tfidf_w2v,
                   X_test_title_tfidf_w2v, X_test_previous, X_test_price))
```

In [80]:

```
from scipy import sparse
X_train_8 = sparse.csr_matrix(X_train_8)
X_test_8 = sparse.csr_matrix(X_test_8)
```

```
X_test_8 = sparse.csr_matrix(X_test_8)
```

In [85]:

```
X_train_8 = X_train_8[0:20000, :]
y_train = y_train[0:20000, :]
print(X_train_8.shape)
print(y_train.shape)
```

```
(20000, 612)
(20000, 1)
```

## 9.2 GridSearch

In [86]:

```
from sklearn.ensemble import GradientBoostingClassifier
classifier_8 = GradientBoostingClassifier()
```

In [87]:

```
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_8 = GridSearchCV(estimator=classifier_8, param_grid=parameters, scoring='roc_auc', cv=2
, n_jobs=-1, return_train_score=True)
```

In [88]:

```
gridsearch_8 = gridsearch_8.fit(X_train_8, y_train)
```

In [89]:

```
results_8 = pd.DataFrame.from_dict(gridsearch_8.cv_results_)
results_8 = results_8.sort_values(['param_n_estimators'])
results_8.head()
```

Out[89]:

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_max_depth | param_n_estimators | params | split0_test_sc |
|---|---|---|---|---|---|---|---|---|
| 0 | 16.180439 | 0.032413 | 0.072306 | 0.000498 | 2 | 10 | {'max_depth': 2, 'n_estimators': 10} | 0.6576 |
| 32 | 59.706996 | 4.866167 | 0.095255 | 0.017035 | 6 | 10 | {'max_depth': 6, 'n_estimators': 10} | 0.6623 |
| 16 | 39.629723 | 0.446238 | 0.099286 | 0.000988 | 4 | 10 | {'max_depth': 4, 'n_estimators': 10} | 0.6647 |
| 40 | 94.690932 | 0.627219 | 0.091266 | 0.001014 | 8 | 10 | {'max_depth': 8, 'n_estimators': 10} | 0.643 |
| 8 | 29.001862 | 0.055212 | 0.101298 | 0.009064 | 3 | 10 | {'max_depth': 3, 'n_estimators': 10} | 0.6618 |

## 9.3 AUC vs HyperParameters

In [90]:

```
train_auc_8 = results_8['mean_train_score']
test_auc_8 = results_8['mean_test_score']
n_estimator_8 = results_8['param_n_estimators']
max_depth_8 =results_8['param_max_depth']
```

In [91]:

```
#To plot this into plotly it should be in list
n_estimator_8 = n_estimator_8.to_list()
max_depth_8 = max_depth_8.to_list()
train_auc_8 = train_auc_8.to_list()
test_auc_8 = test_auc_8.to_list()
```

In [92]:

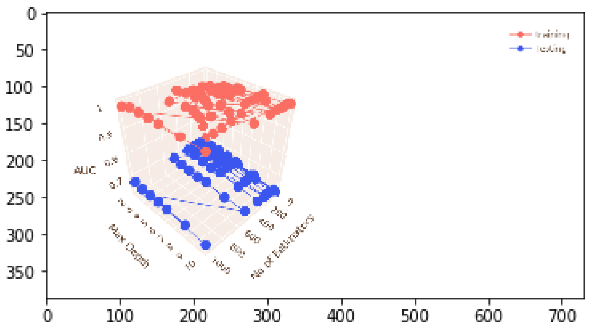
```

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=n_estimator_8, y=max_depth_8, z=train_auc_8, name='training')
trace2 = go.Scatter3d(x=n_estimator_8, y=max_depth_8, z=test_auc_8, name='Testing')
data = [trace1, trace2]

layout = go.Layout(
                    scene=dict(
                                xaxis = dict(title='No of Estimators'),
                                yaxis = dict(title='Max Depth'),
                                zaxis = dict(title='AUC'),
                                )
                    )
fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='3d-scatter-colorscale')
```

In [134]:

```
import cv2
import matplotlib.pyplot as plt
img_8 = cv2.imread(r'C:\Users\prem.k\Desktop\My projects\Assign\RFGBDT pics/8.PNG')
plt.imshow(img_8)
```

Out[134]:

```
<matplotlib.image.AxesImage at 0x1a586b07128>
```



## 9.4 Modelling with Parameters

In [93]:

```
best_n_estimators_8 = gridsearch_8.best_params_['n_estimators']
best_max_depth_8 = gridsearch_8.best_params_['max_depth']
print('Best Number of Estimators:', best_n_estimators_8)
print('Best Max depth in a tree:', best_max_depth_8)
```

```
Best Number of Estimators: 150
Best Max depth in a tree: 2
```

In [94]:

```
classifier_withParam_8 = GradientBoostingClassifier(n_estimators=best_n_estimators_8, max_depth=be
st_max_depth_8)
classifier_withParam_8.fit(X_train_8, y_train)
```

Out[94]:

```
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=150,
                           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)
```

## 9.5 Cross Validation

In [95]:

```
from sklearn.model_selection import cross_val_score
cv_8 = cross_val_score(estimator=classifier_withParam_8, X=X_train_8, y=y_train, cv=2, scoring='roc
_auc')
```

In [96]:

```
best_auc_8 = cv_8.mean()
print('Best AUC: %4f' %best_auc_8)
```

```
Best AUC: 0.692483
```

## 9.6 ROC curve on Train and Test data

In [97]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [98]:

```
y_train_predproba_8 = batch_predict(classifier_withParam_8, X_train_8)
y_test_predproba_8 = batch_predict(classifier_withParam_8, X_test_8)
```

In [99]:

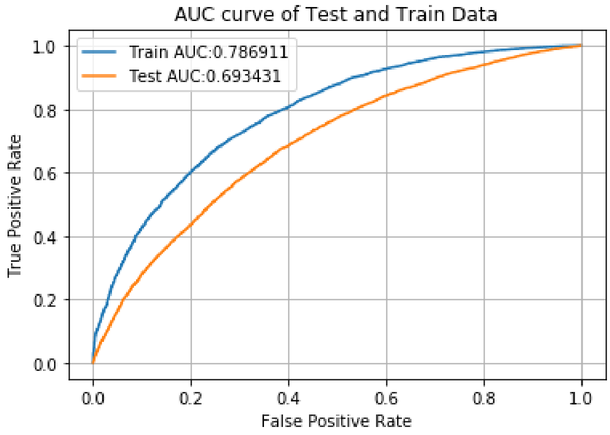
```
from sklearn.metrics import roc_curve, auc
train_fpr_8, train_tpr_8, train_thresh_8 = roc_curve(y_train, y_train_predproba_8)
test_fpr_8, test_tpr_8, test_thresh_8 = roc_curve(y_test, y_test_predproba_8)
```

In [101]:

```
from sklearn.metrics import roc_curve, auc
train_fpr_8, train_tpr_8, train_thresh_8 = roc_curve(y_train, y_train_predproba_8)
test_fpr_8, test_tpr_8, test_thresh_8 = roc_curve(y_test, y_test_predproba_8)
```

In [102]:

```
plt.plot(train_fpr_8, train_tpr_8, label='Train AUC:%4f'%auc(train_fpr_8, train_tpr_8))
plt.plot(test_fpr_8, test_tpr_8, label='Test AUC:%4f'%auc(test_fpr_8, test_tpr_8))

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()
```



## 9.7 Confusion Matrix

In [103]:

```
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 [104]:

```
best_t = find_best_threshold(train_fpr_8, train_tpr_8, train_thresh_8)
```

```
the maximum tpr*(1-fpr) is : 0.5079449254047668 for threshold 0.84
```

In [105]:

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

    return predictions
```

In [106]:

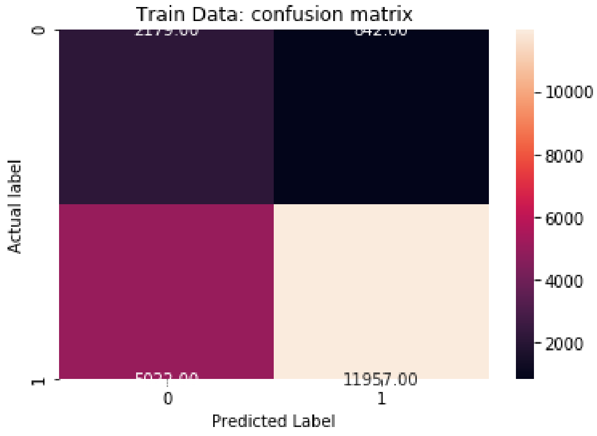
```
y_train_pred_8 = predict_with_threshold(y_train_predproba_8, best_t)
y_test_pred_8 = predict_with_threshold(y_test_predproba_8, best_t)
```

In [107]:

```
from sklearn.metrics import confusion_matrix

cm_train_8 = confusion_matrix(y_train, y_train_pred_8)
cm_test_8 = confusion_matrix(y_test, y_test_pred_8)
```

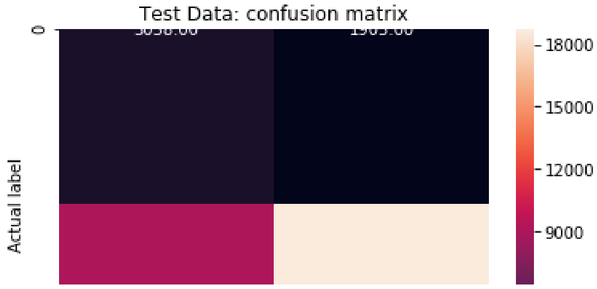
In [108]:

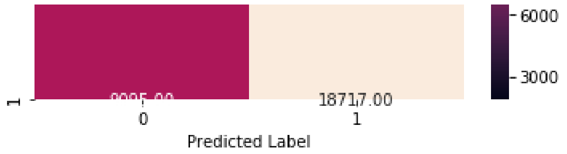
```
sns.heatmap(cm_train_8, annot=True, fmt='.2f')
plt.title('Train Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```



In [110]:

```
sns.heatmap(cm_test_8, annot=True, fmt='.2f')
plt.title('Test Data: confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual label')
plt.show()
```





In [133]:

```
from prettytable import PrettyTable
#from prettytable import MSWORD_FRIENDLY, PLAIN_COLUMN
x = PrettyTable()

x.field_names = ['Set Number', 'Vectorizer', 'Model', 'Hyperparameter: n_estimators', 'Hyperparamet
er: min_samples_split', 'Best AUC', 'No of Data Points for training']
x.add_row(['Set-1', 'BOW + Response Coding', 'Random Forest', str(best_depth_1),
str(best_n_estimators_1), str(best_auc_1), str(no_training_points_1)])
x.add_row(['Set-2', 'TFIDF + Response Coding', 'Random Forest', str(best_depth_2),
str(best_n_estimators_2), str(best_auc_2), str(no_training_points_2)])
x.add_row(['Set-3', 'AvgW2v + Response Coding', 'Random Forest', str(best_depth_3),
str(best_n_estimators_3), str(best_auc_3), str(no_training_points_3)])
x.add_row(['Set-4', 'TFIDFW2v + Response Coding', 'Gradient Boosting', str(best_depth_4), str(best
_n_estimators_4), str(best_auc_4), str(no_training_points_4)])
x.add_row(['Set-1', 'BOW + Response Coding', 'Gradient Boosting', str(best_depth_5),
str(best_n_estimators_5), str(best_auc_5), str(no_training_points_5)])
x.add_row(['Set-2', 'TFIDF + Response Coding', 'Gradient Boosting', str(best_depth_6),
str(best_n_estimators_6), str(best_auc_6), str(no_training_points_6)])
x.add_row(['Set-3', 'AvgW2v + Response Coding', 'Gradient Boosting', str(best_depth_7),
str(best_n_estimators_7), str(best_auc_7), str(no_training_points_7)])
x.add_row(['Set-4', 'TFIDFW2v + Response Coding', 'Gradient Boosting', str(best_depth_8), str(best
_n_estimators_8), str(best_auc_8), str(no_training_points_8)])

print(x)
```

```
+------------+----------------------------+-------------------+------------------------------+-----
------------------------------+----------+--------------------------------+
| Set Number |         Vectorizer         |       Model       | Hyperparameter: n_estimators | Hype
parameter: min_samples_split | Best AUC | No of Data Points for training |
+------------+----------------------------+-------------------+------------------------------+-----
------------------------------+----------+--------------------------------+
|    Set-1   |    BOW + Response Coding   |   Random Forest   |              10              |
300                | 0.678692 |          76473              |
|    Set-2   |   TFIDF + Response Coding  |   Random Forest   |              10              |
500                | 0.683514 |          76473              |
|    Set-3   |  AvgW2v + Response Coding  |   Random Forest   |              8               |
1000               | 0.686137 |          76473              |
|    Set-4   | TFIDFW2v + Response Coding | Gradient Boosting |              8               |
1000               | 0.686826 |          76473              |
|    Set-1   |    BOW + Response Coding   | Gradient Boosting |              2               |
1000               | 0.693973 |          30000              |
|    Set-2   |   TFIDF + Response Coding  | Gradient Boosting |              2               |
300                | 0.687575 |          30000              |
|    Set-3   |  AvgW2v + Response Coding  | Gradient Boosting |              2               |
150                | 0.693946 |          20000              |
|    Set-4   | TFIDFW2v + Response Coding | Gradient Boosting |              2               |
150                | 0.698483 |          20000              |
+------------+----------------------------+-------------------+------------------------------+-----
------------------------------+----------+--------------------------------+
```

## That's the end of the code