# Assignment 9: GBDT

Response Coding: Example

Train Data

| State | class |
|---|---|
| A | 0 |
| B | 1 |
| C | 1 |
| A | 0 |
| A | 1 |
| B | 1 |
| A | 0 |
| A | 1 |
| C | 1 |
| C | 0 |

Resone table(only from train)

| State | Class=0 | Class=1 |
|---|---|---|
| A | 3 | 2 |
| B | 0 | 2 |
| C | 1 | 2 |

Encoded Train Data

| State_0 | State_1 | class |
|---|---|---|
| 3/5 | 2/5 | 0 |
| 0/2 | 2/2 | 1 |
| 1/3 | 2/3 | 1 |
| 3/5 | 2/5 | 0 |
| 3/5 | 2/5 | 1 |
| 0/2 | 2/2 | 1 |
| 3/5 | 2/5 | 0 |
| 3/5 | 2/5 | 1 |
| 1/3 | 2/3 | 1 |
| 1/3 | 2/3 | 0 |

Test Data

| State |
|---|
| A |
| C |
| D |
| C |
| B |
| E |

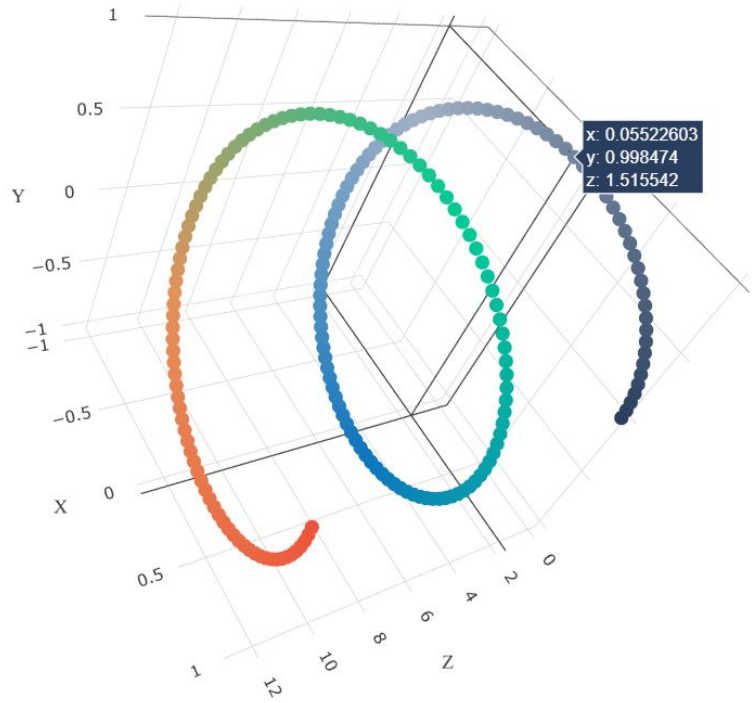
Encoded Test Data

| State_0 | State_1 |
|---|---|
| 3/5 | 2/5 |
| 1/3 | 2/3 |
| 1/2 | 1/2 |
| 1/3 | 2/3 |
| 0/2 | 2/2 |
| 1/2 | 1/2 |

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

1. **Apply GBDT on these feature sets**
   - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check the bellow example, include all 4 values as 4 features)
   - Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (Consider any two hyper parameters)**
   - Find the best hyper parameter which will give the maximum AUC value
   - find the best hyper paramter using k-fold cross validation/simple cross validation data
   - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task
3. **Representation of results**

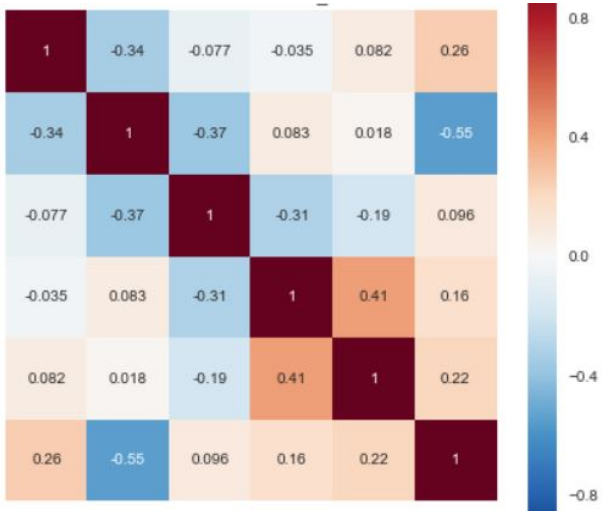
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure




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

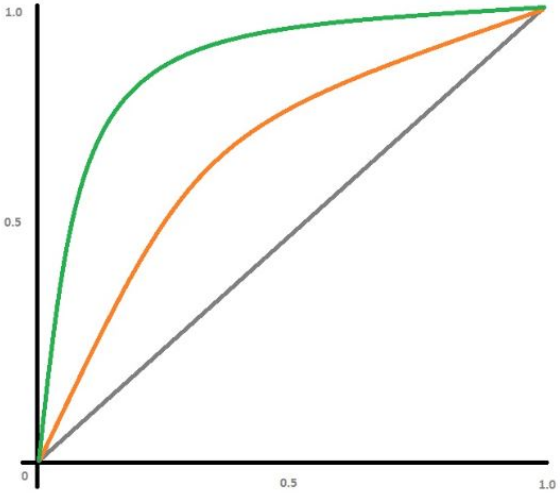
**or**

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure




seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**

- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.



- Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

| | Predicted: NO | Predicted: YES |
|---|---|---|
| Actual: NO | TN = ?? | FP = ?? |
| Actual: YES | FN = ?? | TP = ?? |

4. You need to summarize the results at the end of the notebook, summarize it in the table format

| Vectorizer | Model | Hyper parameter | AUC |
|---|---|---|---|
| BOW | Brute | 7 | 0.78 |
| TFIDF | Brute | 12 | 0.79 |
| W2V | Brute | 10 | 0.78 |
| TFIDFW2V | Brute | 6 | 0.78 |

In [25]:
```
import nltk
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\salma\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[25]: True

In [26]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# import nltk
# nltk.download('vader_lexicon')
```

```
sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students with the biggest ent
for learning my students learn in many different ways using all of our senses and multiple intelligences i use a wide
of techniques to help all my students succeed students in my class come from a variety of different backgrounds which
for wonderful sharing of experiences and cultures including native americans our school is a caring community of succ
learners which can be seen through collaborative student project based learning in and out of the classroom kindergar
in my class love to work with hands on materials and have many different opportunities to practice a skill before it
mastered having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curricul
montana is the perfect place to learn about agriculture and nutrition my students love to role play in our pretend ki
in the early childhood classroom i have had several kids ask me can we try cooking with real food i will take their i
and create common core cooking lessons where we learn important math and writing concepts while cooking delicious hea
food for snack time my students will have a grounded appreciation for the work that went into making the food and knc
of where the ingredients came from as well as how it is healthy for their bodies this project would expand our learni
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce make our own
and mix up healthy plants from our classroom garden in the spring we will also create our own cookbooks to be printed
shared with families students will gain math and literature skills as well as a life long enjoyment for healthy cooki
nannan'
ss = sid.polarity_scores(for_sentiment)

for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

# 1. GBDT (xgboost/lightgbm)

## 1.1 Loading Data

In [27]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv')
data2=pandas.read_csv('train_data.csv')
```

In [28]:
```
print(data.shape, data2.shape)
```

```
(109248, 9) (109248, 17)
```

In [29]:
```
data2.columns
```

Out[29]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'project_subject_categories', 'project_subject_subcategories',
       'project_title', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved'],
      dtype='object')
```

In [30]:
```
data2['project_title'][9]
```

Out[30]:
```
'Just For the Love of Reading--\\r\\nPure Pleasure'
```

In [ ]:
```

```

In [31]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

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

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase

# https://gist.github.com/sebleier/554280
```

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

In [32]:

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

In [33]:

```
final_titles = preprocess_text(data2['project_title'].values)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 109248/109248 [00:05<00:00, 21200.83it/s]
```

In [34]:

```
final_titles[9]
```

Out[34]:
```
'love reading pure pleasure'
```

In [35]:
```
data['project_title']=final_titles
```

In [36]:
```
data.shape
```

Out[36]:
```
(109248, 10)
```

In [37]:
```
data=data[:55000]
```

In [38]:
```
data.shape
```

Out[38]:
```
(55000, 10)
```

In [39]:
```
#https://medium.com/swlh/simple-sentiment-analysis-for-nlp-beginners-and-everyone-else-using-vader-and-textblob-728da
from tqdm import tqdm
analyzer = SentimentIntensityAnalyzer()
data_compound=[]
data_neg=[]
data_neu=[]
data_pos=[]
for ele in tqdm(data['essay']):
    data_compound.append(analyzer.polarity_scores(ele)['compound'])
    data_neg.append(analyzer.polarity_scores(ele)['neg'])
    data_neu.append(analyzer.polarity_scores(ele)['neu'])
    data_pos.append(analyzer.polarity_scores(ele)['pos'])
```

```
100%|██████████| 55000/55000 [12:06<00:00, 75.72it/s]
```

In [40]:
```
#data.to_csv('final_data5.csv')
```

In [41]:
```
data.shape
```

Out[41]: (55000, 10)

In [42]:
```
data['compound'] = data_compound
data['neg']=data_neg
data['neu']=data_neu
data['pos']=data_pos
print(data.shape)
data.head(3)
```

(55000, 14)

Out[42]:

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projects | project_is_approved | clean_categories | clean_ |
|---|---|---|---|---|---|---|---|
| **0** | ca | mrs | grades_prek_2 | 53 | 1 | math_science | a hea |
| **1** | ut | ms | grades_3_5 | 4 | 1 | specialneeds | |
| **2** | ca | mrs | grades_prek_2 | 10 | 1 | literacy_language | |

In [ ]:

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

In [43]:
```
# please write all the code with proper documentation, and proper titles for each subsection
```

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

In [44]:
```
y=data['project_is_approved']
X=data.drop(columns=['project_is_approved'])
```

In [45]:
```
print(X.shape, y.shape)
```

```
(55000, 13) (55000,)
```

In [46]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X,y,test_size=0.24, random_state=0, stratify=y)
```

In [47]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(41800, 13) (41800,)
(13200, 13) (13200,)
```

## 1.3 Make Data Model Ready: encoding eassay, and project_title

In [48]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
```

```
    # c. X-axis label
    # d. Y-axis label
```

In [49]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_essay = TfidfVectorizer(min_df=10)
essay_tfidf_train = vectorizer_tfidf_essay.fit_transform(X_train['essay'].values).toarray()
essay_tfidf_test = vectorizer_tfidf_essay.transform(X_test['essay'].values).toarray()
```

In [50]:
```
print(essay_tfidf_train.shape, essay_tfidf_test.shape)
```

```
(41800, 11312) (13200, 11312)
```

In [51]:
```
essay_tfidf_train
```

Out[51]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [52]:
```
type(essay_tfidf_train)
```

Out[52]:
```
numpy.ndarray
```

In [53]:
```
vectorizer_tfidf_prjtitle = TfidfVectorizer(min_df=10)
project_title_tfidf_train=vectorizer_tfidf_prjtitle.fit_transform(X_train['project_title'].values).toarray()
project_title_tfidf_test=vectorizer_tfidf_prjtitle.transform(X_test['project_title'].values).toarray()
```

In [54]:
```
print(project_title_tfidf_train.shape, project_title_tfidf_test.shape)
```

```
(41800, 1802) (13200, 1802)
```

In [55]:
```
import numpy as np
```

Response Coding:

In [56]:

```
class ResponseCoder():
    import numpy as np

    def __init__(self):
        self.final_dict=0

    def fit(self,feature_vec, response_var):
        from tqdm import tqdm
        if len(feature_vec)==len(response_var):
            unq_lst=list(set(feature_vec))
            dct=dict()
            for ele in tqdm(unq_lst):
                cnt_0=0;
                cnt_1=0;
                total=0;
                for i in range(len(feature_vec)):
                    if feature_vec.iloc[i]==ele:
                        total+=1
                    if feature_vec.iloc[i]== ele and response_var.iloc[i]==1:
                        cnt_1+=1
                    if feature_vec.iloc[i]==ele and response_var.iloc[i]==0:
                        cnt_0+=1
                dct[ele]=[cnt_1/total, cnt_0/total]
            self.final_dict=dct
    def transform(self,feat_vec):
        output_array=list()
        for ele in feat_vec:
            if ele in self.final_dict:
                output_array.append(self.final_dict[ele])
            else:
                output_array.append([0.5,0.5])
        return np.array(output_array)
    def fit_transform(self,feature_vec, response_var):
        self.fit(feature_vec, response_var)
        return self.transform(feature_vec)
```

## 1.4 Make Data Model Ready: encoding numerical, categorical features

In [57]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [58]:

```
from sklearn.preprocessing import StandardScaler
scalar=StandardScaler()
teacher_number_of_previously_posted_projects_train = scalar.fit_transform(X_train['teacher_number_of_previously_poste
teacher_number_of_previously_posted_projects_test = scalar.transform(X_test['teacher_number_of_previously_posted_pro
sc=StandardScaler()
price_train = sc.fit_transform(X_train['price'].values.reshape(-1,1))
price_test = sc.transform(X_test['price'].values.reshape(-1,1))
```

In [59]:

```
encoder=ResponseCoder()
school_state_train=encoder.fit_transform(X_train['school_state'],y_train)
school_state_test=encoder.transform(X_test['school_state'])
# school_state_train

encoder=ResponseCoder()
teacher_prefix_train=encoder.fit_transform(X_train['teacher_prefix'],y_train)
teacher_prefix_test=encoder.transform(X_test['teacher_prefix'])

encoder=ResponseCoder()
project_grade_category_train=encoder.fit_transform(X_train['project_grade_category'],y_train)
project_grade_category_test=encoder.transform(X_test['project_grade_category'])

encoder=ResponseCoder()
clean_categories_train=encoder.fit_transform(X_train['clean_categories'], y_train)
clean_categories_test=encoder.transform(X_test['clean_categories'])
```

```
encoder=ResponseCoder()
clean_subcategories_train=encoder.fit_transform(X_train['clean_subcategories'], y_train)
clean_subcategories_test=encoder.transform(X_test['clean_subcategories'])
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 51/51 [01:03<00:00,  1.24s/i
t]
100%|██████████████████████████████████████████████████████████████████████████████| 5/5 [00:07<00:00,  1.40s/i
t]
100%|██████████████████████████████████████████████████████████████████████████████| 4/4 [00:05<00:00,  1.45s/i
t]
100%|██████████████████████████████████████████████████████████████████████████████| 45/45 [00:56<00:00,  1.25s/i
t]
100%|███████████████████████████████████████████████████████████████████████████| 354/354 [07:14<00:00,  1.23s/i
t]
```

In [60]:
```
print(project_grade_category_train)
```

```
[[0.83887558 0.16112442]
 [0.84782154 0.15217846]
 [0.83887558 0.16112442]
 ...
 [0.83887558 0.16112442]
 [0.83005817 0.16994183]
 [0.83404469 0.16595531]]
```

In [61]:
```
train_X=np.hstack((essay_tfidf_train,project_title_tfidf_train,clean_categories_train,clean_subcategories_train,teach
```

In [62]:
```
test_X=np.hstack((essay_tfidf_test,project_title_tfidf_test,clean_categories_test,clean_subcategories_test,teacher_nu
```

In [63]:
```
print(len(school_state_test),len(school_state_train))
```

```
13200 41800
```

In [64]:
```
print(train_X.shape, test_X.shape)
```

```
(41800, 13130) (13200, 13130)
```

In [65]:
```
print(essay_tfidf_train.shape, project_title_tfidf_train.shape, clean_categories_train.shape, clean_subcategories_tra
```

```
(41800, 11312) (41800, 1802) (41800, 2) (41800, 2) (41800, 1) (41800, 1) (41800, 2) (41800, 2) (41800, 2) (41800, 1)
```

In [66]:
```
clean_categories_train
```

Out[66]:
```
array([[0.85685574, 0.14314426],
       [0.79511143, 0.20488857],
       [0.85685574, 0.14314426],
       ...,
       [0.86596737, 0.13403263],
       [0.85245902, 0.14754098],
       [0.80474934, 0.19525066]])
```

In [67]:
```
essay_tfidf_train
```

Out[67]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [68]:
```
print(project_grade_category_train[5][0]+project_grade_category_train[5][1])
print(project_grade_category_train[51][0]+project_grade_category_train[51][1])
print(project_grade_category_train[99][0]+project_grade_category_train[99][1])
```

```
1.0
1.0
1.0
```

In [69]:
```
train_X.shape
```

Out[69]:
```
(41800, 13130)
```

In [70]:
```
# X_final_train=pandas.DataFrame(train_X)
# X_final_test=pandas.DataFrame(test_X)
```

In [71]:
```
# X_final_train.shape
```

In [72]:
```
# y_final_train=pandas.DataFrame(y_train)
# y_final_test=pandas.DataFrame(y_test)
```

In [73]:
```
# X_final_train.to_csv('final_train1.csv')
# X_final_test.to_csv('final_test1.csv')
# y_train.to_csv('y_final_train.csv')
# y_test.to_csv('y_final_test.csv')
```

In [74]:
```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [75]:
```
tfidf_model_proj_title = TfidfVectorizer()
tfidf_model_proj_title.fit(X_train['project_title'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_proj_title = dict(zip(tfidf_model_proj_title.get_feature_names(), list(tfidf_model_proj_title.idf_)))
tfidf_words_proj_title = set(tfidf_model_proj_title.get_feature_names())
```

In [76]:
```
try:
    import dill as pickle
except ImportError:
    import pickle
```

In [77]:
```
#please use below code to load glove vectors
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [78]:
```
#tfidf_w2v for essay column
from tqdm import tqdm
import numpy as np
def compute_tfidf_W2V(X):
    # average Word2Vec
    # compute average word2vec for each review.
    tfidf_w2v_vectors= []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(X.values): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence/review
        for word in sentence.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words):
                vec = model[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sent
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += 3

                tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)
    return tfidf_w2v_vectors

tfidf_w2v_essay_train=compute_tfidf_W2V(X_train['essay'])
tfidf_w2v_essay_test=compute_tfidf_W2V(X_test['essay'])
print(len(tfidf_w2v_essay_train))
print(len(tfidf_w2v_essay_train[0]))
print(len(tfidf_w2v_essay_test))
print(len(tfidf_w2v_essay_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████| 41800/41800 [03:09<00:00, 220.66it/
s]
100%|████████████████████████████████████████████████████████████████████████| 13200/13200 [00:59<00:00, 223.54it/
s]
41800
300
13200
300
```

In [79]:
```
type(tfidf_w2v_essay_train)
```

Out[79]: list

In [80]:
```
#tfidf_w2v_essay_train
```

In [81]:
```
#tfidf_w2v for project tiltle column
from tqdm import tqdm
import numpy as np
def compute_tfidf_W2V(X):
    # average Word2Vec
    # compute average word2vec for each review.
    tfidf_w2v_vectors= []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(X.values): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence/review
        for word in sentence.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words_proj_title):
                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(sent
                tf_idf = dictionary_proj_title[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += 3

                tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)
    return tfidf_w2v_vectors

tfidf_w2v_project_title_train=compute_tfidf_W2V(X_train['project_title'])
tfidf_w2v_project_title_test=compute_tfidf_W2V(X_test['project_title'])
print(len(tfidf_w2v_project_title_train))
print(len(tfidf_w2v_project_title_train[0]))
print(len(tfidf_w2v_project_title_test))
print(len(tfidf_w2v_project_title_test[0]))
```

```
100%|██████████| 41800/41800 [00:02<00:00, 17819.12it/s]
```

```
100%|██████████| 13200/13200 [00:00<00:00, 17679.04it/s]
41800
300
13200
300
```

In [82]:
```
train_X2=np.hstack((tfidf_w2v_essay_train,tfidf_w2v_project_title_train,clean_categories_train,clean_subcategories_tr
```

In [83]:
```
test_X2=np.hstack((tfidf_w2v_essay_test,tfidf_w2v_project_title_test,clean_categories_test,clean_subcategories_test,t
```

In [84]:
```
print(train_X2.shape, test_X2.shape)
```

```
(41800, 616) (13200, 616)
```

In [85]:
```
# X_final_train2=pandas.DataFrame(train_X2)
# X_final_test2=pandas.DataFrame(test_X2)
```

In [86]:
```
# X_final_train2.to_csv('final_train2.csv')
# X_final_test2.to_csv('final_test2.csv')
```

In [ ]:

In [ ]:

In [ ]:

## 1.5 Appling Models on different kind of featurization as mentioned in the instructions

Apply GBDT on different kind of featurization as mentioned in the instructions

For Every model that you work on make sure you do the step 2 and step 3 of instrucations

In [87]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [88]:
```
#!pip install xgboost
```

# TFIDF

In [89]:
```
import warnings
warnings.filterwarnings("ignore")
# from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import GradientBoostingClassifier

clf=GradientBoostingClassifier()
param={
    'n_estimators':[50,75],
    'max_depth':[1,3]
}

search=GridSearchCV(clf, param_grid=param, cv=3, scoring='roc_auc', verbose=10, return_train_score=True, error_score=
search.fit(train_X, y_train)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
[CV 1/3; 1/4] START max_depth=1, n_estimators=50.............................
[CV 1/3; 1/4] END .............max_depth=1, n_estimators=50; total time=23.4min
```

```
[CV 2/3; 1/4] START max_depth=1, n_estimators=50.............................
[CV 2/3; 1/4] END .............max_depth=1, n_estimators=50; total time=23.0min
[CV 3/3; 1/4] START max_depth=1, n_estimators=50.............................
[CV 3/3; 1/4] END .............max_depth=1, n_estimators=50; total time=22.7min
[CV 1/3; 2/4] START max_depth=1, n_estimators=75.............................
[CV 1/3; 2/4] END .............max_depth=1, n_estimators=75; total time=31.9min
[CV 2/3; 2/4] START max_depth=1, n_estimators=75.............................
[CV 2/3; 2/4] END .............max_depth=1, n_estimators=75; total time=15.7min
[CV 3/3; 2/4] START max_depth=1, n_estimators=75.............................
[CV 3/3; 2/4] END .............max_depth=1, n_estimators=75; total time=15.7min
[CV 1/3; 3/4] START max_depth=3, n_estimators=50.............................
[CV 1/3; 3/4] END .............max_depth=3, n_estimators=50; total time=26.0min
[CV 2/3; 3/4] START max_depth=3, n_estimators=50.............................
[CV 2/3; 3/4] END .............max_depth=3, n_estimators=50; total time=26.2min
[CV 3/3; 3/4] START max_depth=3, n_estimators=50.............................
[CV 3/3; 3/4] END .............max_depth=3, n_estimators=50; total time=30.7min
[CV 1/3; 4/4] START max_depth=3, n_estimators=75.............................
[CV 1/3; 4/4] END .............max_depth=3, n_estimators=75; total time=40.7min
[CV 2/3; 4/4] START max_depth=3, n_estimators=75.............................
[CV 2/3; 4/4] END .............max_depth=3, n_estimators=75; total time=56.4min
[CV 3/3; 4/4] START max_depth=3, n_estimators=75.............................
[CV 3/3; 4/4] END .............max_depth=3, n_estimators=75; total time=78.3min
```

Out[89]:
```
GridSearchCV(cv=3, error_score='raise', estimator=GradientBoostingClassifier(),
             param_grid={'max_depth': [1, 3], 'n_estimators': [50, 75]},
             return_train_score=True, scoring='roc_auc', verbose=10)
```

In [90]:
```
search.cv_results_
```

Out[90]:
```
{'mean_fit_time': array([1381.48611093, 1265.09251364, 1656.48110723, 3507.05092923]),
 'std_fit_time': array([ 17.39951128, 460.40433724, 131.44381912, 924.1224749 ]),
 'mean_score_time': array([1.22048696, 0.63140114, 0.75456691, 0.88602114]),
 'std_score_time': array([0.07820842, 0.04591742, 0.16434798, 0.34842526]),
 'param_max_depth': masked_array(data=[1, 1, 3, 3],
              mask=[False, False, False, False],
       fill_value='?',
            dtype=object),
 'param_n_estimators': masked_array(data=[50, 75, 50, 75],
              mask=[False, False, False, False],
       fill_value='?',
            dtype=object),
 'params': [{'max_depth': 1, 'n_estimators': 50},
  {'max_depth': 1, 'n_estimators': 75},
  {'max_depth': 3, 'n_estimators': 50},
```

```
 {'max_depth': 3, 'n_estimators': 75}],
'split0_test_score': array([0.66869504, 0.67834235, 0.69865808, 0.70448246]),
'split1_test_score': array([0.66305228, 0.67359112, 0.69807774, 0.70507426]),
'split2_test_score': array([0.65514825, 0.66734628, 0.68881908, 0.69687563]),
'mean_test_score': array([0.66229852, 0.67309325, 0.69518497, 0.70214412]),
'std_test_score': array([0.00555608, 0.00450291, 0.00450759, 0.00373321]),
'rank_test_score': array([4, 3, 2, 1]),
'split0_train_score': array([0.67135099, 0.68590943, 0.75592624, 0.77897189]),
'split1_train_score': array([0.67430428, 0.68636168, 0.75374293, 0.77882538]),
'split2_train_score': array([0.67367072, 0.68644204, 0.75120903, 0.77593241]),
'mean_train_score': array([0.67310867, 0.68623772, 0.75362607, 0.7779099 ]),
'std_train_score': array([0.00126949, 0.00023444, 0.00192757, 0.00139957])}
```

In [94]:
```
search.best_estimator_
```

Out[94]:
```
GradientBoostingClassifier(n_estimators=75)
```

In [95]:
```
search.best_params_
```

Out[95]:
```
{'max_depth': 3, 'n_estimators': 75}
```

In [98]:
```
#!pip install plotly
```

```
Collecting plotly
  Downloading plotly-4.14.3-py2.py3-none-any.whl (13.2 MB)
Collecting retrying>=1.3.3
  Downloading retrying-1.3.3.tar.gz (10 kB)
Requirement already satisfied: six in c:\users\salma\anaconda3\lib\site-packages (from plotly) (1.15.0)
Building wheels for collected packages: retrying
  Building wheel for retrying (setup.py): started
  Building wheel for retrying (setup.py): finished with status 'done'
  Created wheel for retrying: filename=retrying-1.3.3-py3-none-any.whl size=11429 sha256=12905a97d5e960436efd8635a04d
28e40388a9a2ae9c576cd078c6b16d68e430
  Stored in directory: c:\users\salma\appdata\local\pip\cache\wheels\c4\a7\48\0a434133f6d56e878ca511c0e6c38326907c079
2f67b476e56
Successfully built retrying
Installing collected packages: retrying, plotly
Successfully installed plotly-4.14.3 retrying-1.3.3
```

In [99]:
```
import plotly.offline as offline
```

```
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [103…

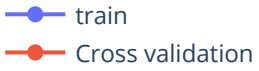
```
search.cv_results_['param_max_depth'], search.cv_results_['param_n_estimators']
```

Out[103…

```
(masked_array(data=[1, 1, 3, 3],
              mask=[False, False, False, False],
        fill_value='?',
             dtype=object),
 masked_array(data=[50, 75, 50, 75],
              mask=[False, False, False, False],
        fill_value='?',
             dtype=object))
```

In [104…

```
gbdt_max_depth=search.cv_results_['param_max_depth']
gbdt_n_estimators=search.cv_results_['param_n_estimators']
auc1_train=search.cv_results_['mean_train_score']
auc1_test=search.cv_results_['mean_test_score']
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=gbdt_max_depth,y=gbdt_n_estimators,z=auc1_train, name = 'train')
trace2 = go.Scatter3d(x=gbdt_max_depth,y=gbdt_n_estimators,z=auc1_test, name = 'Cross validation')
data = [trace1, trace2]
layout = go.Layout(scene = dict(
 xaxis = dict(title='max_depth'),
 yaxis = dict(title='n_estimators'),
 zaxis = dict(title='AUC'),))
fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



In [105…

```
model1=GradientBoostingClassifier(n_estimators=search.best_params_['n_estimators'] , max_depth=search.best_params_['m
model1.fit(train_X, y_train)
```

Out[105… GradientBoostingClassifier(n_estimators=75)

In [106…

```
from sklearn.metrics import roc_curve, roc_auc_score,auc
import matplotlib.pyplot as plt
train_pred =model1.predict_proba(train_X)[:,1]
test_pred =model1.predict_proba(test_X)[:,1]
```

In [108…

```
model1.predict_proba(train_X)
```

Out[108…

```
array([[0.13589846, 0.86410154],
```

```
       [0.25030058, 0.74969942],
       [0.1065707 , 0.8934293 ],
       ...,
       [0.12508744, 0.87491256],
       [0.11634383, 0.88365617],
       [0.15479946, 0.84520054]])
```

In [109…

```
train_pred
```

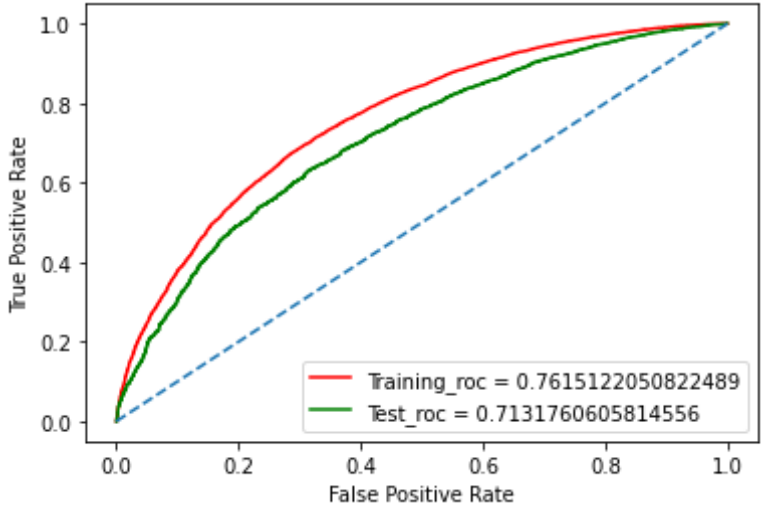
Out[109…

```
array([0.86410154, 0.74969942, 0.8934293 , ..., 0.87491256, 0.88365617,
       0.84520054])
```

In [110…

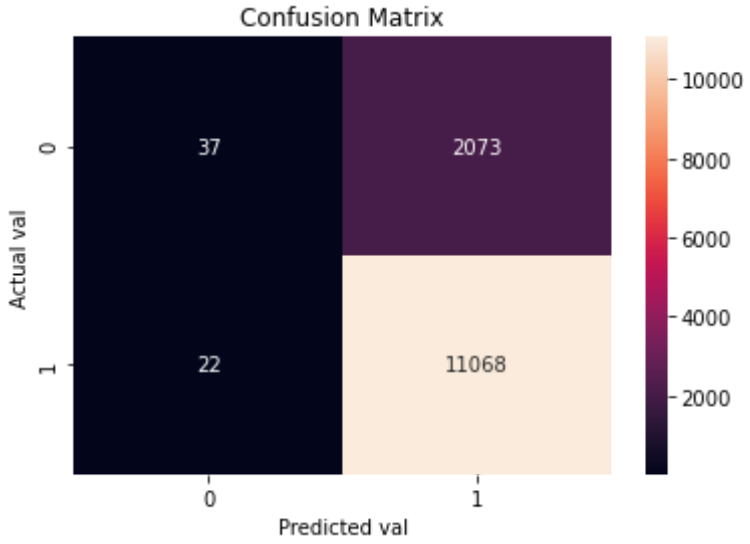
```
fpr1, tpr1, _=roc_curve(y_train, train_pred)
fpr2, tpr2, _=roc_curve(y_test, test_pred)

plt.plot(fpr1, tpr1, color='red', label='Training_roc = '+str(auc(fpr1, tpr1)))
plt.plot(fpr2, tpr2, color='green', label='Test_roc = '+str(auc(fpr2, tpr2)))
plt.plot([0,1],[0,1],'--')
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```



In [111…

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
ax=plt.subplot()
cnf_mtx=confusion_matrix(y_test, model1.predict(test_X))
sns.heatmap(cnf_mtx, annot=True, ax=ax, fmt='d')
ax.set_xlabel('Predicted val');ax.set_ylabel('Actual val');
ax.set_ylim(2.0, 0)
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['0','1']);
ax.yaxis.set_ticklabels(['0','1']);
```




# TFIDF-W2V

In [112…

```
train_X2.shape
```

Out[112… (41800, 616)

In [113…

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import GradientBoostingClassifier
```

```
clf=GradientBoostingClassifier()
param={
    'n_estimators':[50, 75, 100],
    'max_depth':[1, 3, 5]
}

search=GridSearchCV(clf, param_grid=param, cv=3, scoring='roc_auc', verbose=10, return_train_score=True, error_score=
search.fit(train_X2, y_train)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
[CV 1/3; 1/9] START max_depth=1, n_estimators=50.............................
[CV 1/3; 1/9] END .............max_depth=1, n_estimators=50; total time= 2.9min
[CV 2/3; 1/9] START max_depth=1, n_estimators=50.............................
[CV 2/3; 1/9] END .............max_depth=1, n_estimators=50; total time= 3.0min
[CV 3/3; 1/9] START max_depth=1, n_estimators=50.............................
[CV 3/3; 1/9] END .............max_depth=1, n_estimators=50; total time= 2.9min
[CV 1/3; 2/9] START max_depth=1, n_estimators=75.............................
[CV 1/3; 2/9] END .............max_depth=1, n_estimators=75; total time= 4.3min
[CV 2/3; 2/9] START max_depth=1, n_estimators=75.............................
[CV 2/3; 2/9] END .............max_depth=1, n_estimators=75; total time= 4.4min
[CV 3/3; 2/9] START max_depth=1, n_estimators=75.............................
[CV 3/3; 2/9] END .............max_depth=1, n_estimators=75; total time= 4.4min
[CV 1/3; 3/9] START max_depth=1, n_estimators=100............................
[CV 1/3; 3/9] END .............max_depth=1, n_estimators=100; total time= 5.8min
[CV 2/3; 3/9] START max_depth=1, n_estimators=100............................
[CV 2/3; 3/9] END .............max_depth=1, n_estimators=100; total time= 5.8min
[CV 3/3; 3/9] START max_depth=1, n_estimators=100............................
[CV 3/3; 3/9] END .............max_depth=1, n_estimators=100; total time= 5.9min
[CV 1/3; 4/9] START max_depth=3, n_estimators=50.............................
[CV 1/3; 4/9] END .............max_depth=3, n_estimators=50; total time= 8.4min
[CV 2/3; 4/9] START max_depth=3, n_estimators=50.............................
[CV 2/3; 4/9] END .............max_depth=3, n_estimators=50; total time= 8.4min
[CV 3/3; 4/9] START max_depth=3, n_estimators=50.............................
[CV 3/3; 4/9] END .............max_depth=3, n_estimators=50; total time= 8.4min
[CV 1/3; 5/9] START max_depth=3, n_estimators=75.............................
[CV 1/3; 5/9] END .............max_depth=3, n_estimators=75; total time= 8.7min
[CV 2/3; 5/9] START max_depth=3, n_estimators=75.............................
[CV 2/3; 5/9] END .............max_depth=3, n_estimators=75; total time= 7.1min
[CV 3/3; 5/9] START max_depth=3, n_estimators=75.............................
[CV 3/3; 5/9] END .............max_depth=3, n_estimators=75; total time= 7.1min
[CV 1/3; 6/9] START max_depth=3, n_estimators=100............................
[CV 1/3; 6/9] END .............max_depth=3, n_estimators=100; total time= 9.6min
[CV 2/3; 6/9] START max_depth=3, n_estimators=100............................
```

```
[CV 2/3; 6/9] END .............max_depth=3, n_estimators=100; total time= 9.5min
[CV 3/3; 6/9] START max_depth=3, n_estimators=100.............................
[CV 3/3; 6/9] END .............max_depth=3, n_estimators=100; total time= 9.5min
[CV 1/3; 7/9] START max_depth=5, n_estimators=50..............................
[CV 1/3; 7/9] END ..............max_depth=5, n_estimators=50; total time= 7.6min
[CV 2/3; 7/9] START max_depth=5, n_estimators=50..............................
[CV 2/3; 7/9] END ..............max_depth=5, n_estimators=50; total time= 7.6min
[CV 3/3; 7/9] START max_depth=5, n_estimators=50..............................
[CV 3/3; 7/9] END ..............max_depth=5, n_estimators=50; total time= 7.7min
[CV 1/3; 8/9] START max_depth=5, n_estimators=75..............................
[CV 1/3; 8/9] END ..............max_depth=5, n_estimators=75; total time=11.5min
[CV 2/3; 8/9] START max_depth=5, n_estimators=75..............................
[CV 2/3; 8/9] END ..............max_depth=5, n_estimators=75; total time=11.4min
[CV 3/3; 8/9] START max_depth=5, n_estimators=75..............................
[CV 3/3; 8/9] END ..............max_depth=5, n_estimators=75; total time=11.5min
[CV 1/3; 9/9] START max_depth=5, n_estimators=100.............................
[CV 1/3; 9/9] END ..............max_depth=5, n_estimators=100; total time=15.3min
[CV 2/3; 9/9] START max_depth=5, n_estimators=100.............................
[CV 2/3; 9/9] END ..............max_depth=5, n_estimators=100; total time=15.4min
[CV 3/3; 9/9] START max_depth=5, n_estimators=100.............................
[CV 3/3; 9/9] END ..............max_depth=5, n_estimators=100; total time=15.3min
```

Out[113…

```
GridSearchCV(cv=3, error_score='raise', estimator=GradientBoostingClassifier(),
             param_grid={'max_depth': [1, 3, 5], 'n_estimators': [50, 75, 100]},
             return_train_score=True, scoring='roc_auc', verbose=10)
```

In [114…

```
search.cv_results_
```

Out[114…

```
{'mean_fit_time': array([177.02914039, 262.60771179, 348.95479711, 504.61252594,
        458.35764941, 572.96198455, 459.15565634, 688.23477101,
        919.39744099]),
 'std_fit_time': array([ 1.42204333,  1.8745234 ,  1.7667388 ,  2.26365854, 45.79535745,
         4.28674548,  1.34687553,  2.99889958,  1.654626  ]),
 'mean_score_time': array([0.09602308, 0.09332252, 0.11440428, 0.16805387, 0.08909957,
        0.10803421, 0.09952402, 0.13299704, 0.16365568]),
 'std_score_time': array([0.00655905, 0.00377518, 0.01606933, 0.01307867, 0.00082731,
        0.0026596 , 0.00073184, 0.00123642, 0.00170842]),
 'param_max_depth': masked_array(data=[1, 1, 1, 3, 3, 3, 5, 5, 5],
              mask=[False, False, False, False, False, False, False, False,
                    False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[50, 75, 100, 50, 75, 100, 50, 75, 100],
              mask=[False, False, False, False, False, False, False, False,
```

```
                    False],
       fill_value='?',
            dtype=object),
 'params': [{'max_depth': 1, 'n_estimators': 50},
  {'max_depth': 1, 'n_estimators': 75},
  {'max_depth': 1, 'n_estimators': 100},
  {'max_depth': 3, 'n_estimators': 50},
  {'max_depth': 3, 'n_estimators': 75},
  {'max_depth': 3, 'n_estimators': 100},
  {'max_depth': 5, 'n_estimators': 50},
  {'max_depth': 5, 'n_estimators': 75},
  {'max_depth': 5, 'n_estimators': 100}],
 'split0_test_score': array([0.67644422, 0.68364907, 0.68826149, 0.69615522, 0.70162722,
        0.70418111, 0.70007658, 0.70297088, 0.69987168]),
 'split1_test_score': array([0.67970386, 0.68857638, 0.69490707, 0.70919173, 0.71295302,
        0.71528427, 0.70867311, 0.71344417, 0.70834825]),
 'split2_test_score': array([0.6686578 , 0.67962286, 0.68580582, 0.69902665, 0.70405181,
        0.7049165 , 0.7025892 , 0.70670397, 0.70761888]),
 'mean_test_score': array([0.67493529, 0.68394944, 0.68965813, 0.70145787, 0.70621068,
        0.70812729, 0.70377963, 0.70770634, 0.7052796 ]),
 'std_test_score': array([0.00463404, 0.00366142, 0.00384458, 0.0055929 , 0.00486922,
        0.00506964, 0.00360905, 0.00433405, 0.00383555]),
 'rank_test_score': array([9, 8, 7, 6, 3, 1, 5, 2, 4]),
 'split0_train_score': array([0.69266991, 0.70353155, 0.71011127, 0.7586037 , 0.77724181,
        0.79248998, 0.85609558, 0.89028148, 0.91601448]),
 'split1_train_score': array([0.68869584, 0.6997098 , 0.70825935, 0.75467736, 0.77224382,
        0.78756139, 0.84555887, 0.88793501, 0.91335639]),
 'split2_train_score': array([0.69196158, 0.70193944, 0.70943225, 0.75649384, 0.77361404,
        0.78912619, 0.85130439, 0.88562428, 0.91134616]),
 'mean_train_score': array([0.69110911, 0.70172693, 0.70926762, 0.75659163, 0.77436656,
        0.78972585, 0.85098628, 0.88794693, 0.91357235]),
 'std_train_score': array([0.00173077, 0.00156744, 0.00076495, 0.00160441, 0.00210867,
        0.00205628, 0.00430747, 0.00190131, 0.00191194])}
```

In [115…

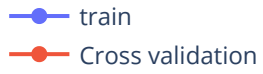
```
search.best_params_
```

Out[115… `{'max_depth': 3, 'n_estimators': 100}`

In [116…

```
gbdt_max_depth=search.cv_results_['param_max_depth']
gbdt_n_estimators=search.cv_results_['param_n_estimators']
auc1_train=search.cv_results_['mean_train_score']
auc1_test=search.cv_results_['mean_test_score']
```

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=gbdt_max_depth,y=gbdt_n_estimators,z=auc1_train, name = 'train')
trace2 = go.Scatter3d(x=gbdt_max_depth,y=gbdt_n_estimators,z=auc1_test, name = 'Cross validation')
data = [trace1, trace2]
layout = go.Layout(scene = dict(
 xaxis = dict(title='max_depth'),
 yaxis = dict(title='n_estimators'),
 zaxis = dict(title='AUC'),))
fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



In [117…

```
model2=GradientBoostingClassifier(n_estimators=search.best_params_['n_estimators'] , max_depth=search.best_params_['m
model2.fit(train_X2, y_train)
```

Out[117… GradientBoostingClassifier()

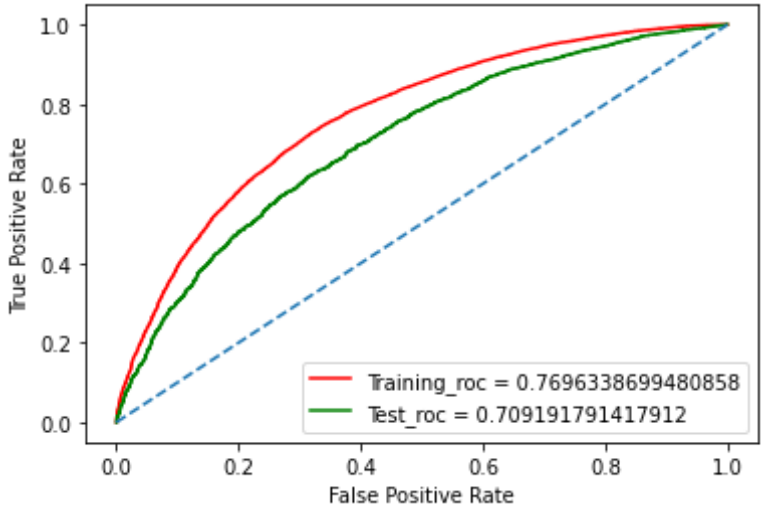
In [118…

```
train_pred =model2.predict_proba(train_X2)[:,1]
test_pred =model2.predict_proba(test_X2)[:,1]
```

In [119…

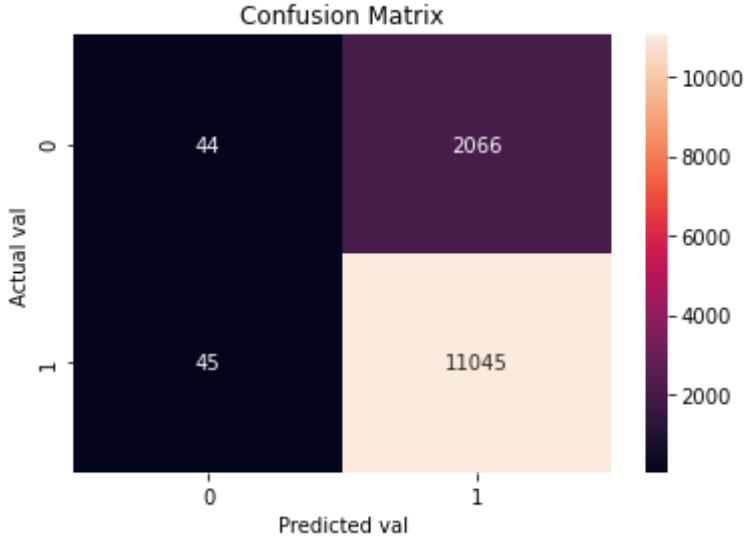
```
fpr1, tpr1, _=roc_curve(y_train, train_pred)
fpr2, tpr2, _=roc_curve(y_test, test_pred)

plt.plot(fpr1, tpr1, color='red', label='Training_roc = '+str(auc(fpr1, tpr1)))
plt.plot(fpr2, tpr2, color='green', label='Test_roc = '+str(auc(fpr2, tpr2)))
plt.plot([0,1],[0,1],'--')
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```



In [120…

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
ax=plt.subplot()
cnf_mtx=confusion_matrix(y_test, model2.predict(test_X2))
sns.heatmap(cnf_mtx, annot=True, ax=ax, fmt='d')
ax.set_xlabel('Predicted val');ax.set_ylabel('Actual val');
ax.set_ylim(2.0, 0)
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['0','1']);
ax.yaxis.set_ticklabels(['0','1']);
```




In [ ]:

# 3. Summary

as mentioned in the step 4 of instructions

In [122…
```
#!pip install prettytable
```

```
Collecting prettytable
  Downloading prettytable-2.1.0-py3-none-any.whl (22 kB)
Requirement already satisfied: wcwidth in c:\users\salma\anaconda3\lib\site-packages (from prettytable) (0.2.5)
Installing collected packages: prettytable
Successfully installed prettytable-2.1.0
```

In [125…
```
from prettytable import PrettyTable
table= PrettyTable()

table.field_names = ["Vectorizer", "Model", "Max_depth","n_estimators", "AUC"]
table.add_rows(
    [
        ["TFIDF", "GBDT", 3, 75, 0.713],
        ["TFIDF_W2V", "GBDT", 3, 100,0.709]

    ]
)
print(table)
```

```
+------------+-------+-----------+--------------+-------+
| Vectorizer | Model | Max_depth | n_estimators |  AUC  |
+------------+-------+-----------+--------------+-------+
|   TFIDF    |  GBDT |     3     |      75      | 0.713 |
| TFIDF_W2V  |  GBDT |     3     |     100      | 0.709 |
+------------+-------+-----------+--------------+-------+
```

In [ ]:
```

```