DecisionTree Analysis

In [ ]:

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.
Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
  F
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

Notes on the Essay Data ¶

Prior to May 17, 2016, the prompts for the essays were as follows:

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_3:__ "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

Step 1: Import the necessary Libraries

we will need to import libraries that allow for data analysis and data visualization to get acclimated to the dataset. We will be using pandas, numpy, matplotlib and seaborn to conduct this. Data Exploration libraries

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore', category=UserWarning, module='gensim')
warnings.filterwarnings("ignore",'detected Windows; aliasing chunkize to chunkize_serial')
warnings.filterwarnings("ignore", message="numpy.dtype size changed")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns

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

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

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from tqdm import tqdm
import os
from collections import Counter
```

Step 2: Read in the dataset.

We will use the pandas .read_csv() method to read in the dataset. Then we will use the. head() method to observe the first few rows of the data, to understand the information better. In our case, the feature(column) headers tell us pretty little. This is fine because we are merely trying to gain insight via classifying new data points by referencing it’s neighboring elements.

1.1 Reading Data

In [2]:

```python
project_data = pd.read_csv("D:\\VipinML\\Assignment 2\\Assignments_DonorsChoose_2018\\train_data.csv")
resource_data = pd.read_csv("D:\\VipinML\Assignment 2\\Assignments_DonorsChoose_2018\\resources.csv")
#Limit the data for testing purpose since processing takes few hours for full set..

project_data = project_data.head(50000)
resource_data = resource_data.head (50000)

resource_data.head(1)
```

Out[2]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.0

In [3]:

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

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

In [4]:

```python
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]
#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)

# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]
project_data.head(1)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:0

1.2 preprocessing of `project_subject_categories`

In [5]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-fr
om-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-strin
g-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Scienc
e", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on
space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to
replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(emp
ty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the tra
iling spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [6]:

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

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

In [7]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in teacher_cat:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
    temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
    temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.3 Text preprocessing

In [8]:

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

In [9]:

```
#clean project_grade_category
project_data["project_grade_category"] = \
project_data.apply(lambda x: (x['project_grade_category'].replace(' ', '_')), axis=1)
project_data["project_grade_category"] = \
project_data.apply(lambda x: (x['project_grade_category'].replace('-', '_')), axis=1)
```

In [10]:
```
project_data.head(1)
```

Out[10]:

	Unnamed: 0	id	teacher_id	school_state	Date	project_grade
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	GA	2016-04-27 00:53:00	Grade

In [11]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [12]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [13]:
```
sent = decontracted(project_data['essay'].values[500])
print(sent[1:200])
print("="*100)
```

```
ore subjects like math and science must be relevant for students, and at the
same time, foster creativity, curiosity and a passion for problem solving.My
students are eager to go beyond basic learnin
==============================================================================
======================
```

In [14]:

```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent[1:200])
print(sent[1:200])
```

```
ore subjects like math and science must be relevant for students, and at the
same time, foster creativity, curiosity and a passion for problem solving.My
students are eager to go beyond basic learnin
ore subjects like math and science must be relevant for students, and at the
same time, foster creativity, curiosity and a passion for problem solving.My
students are eager to go beyond basic learnin
```

In [15]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent[1:200])
```

```
ore subjects like math and science must be relevant for students and at the s
ame time foster creativity curiosity and a passion for problem solving My stu
dents are eager to go beyond basic learning a
```

In [16]:

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

1.4.3 Merging price with project_data

In [17]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'
}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
print (price_data[1:3])
project_data.head(1)
```

```
        id   price  quantity
1  p000052  114.98         2
2  p000147   13.13        25
```

Out[17]:

	Unnamed: 0	id	teacher_id	school_state	Date	project_grade_c
0	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	GA	2016-04-27 00:53:00	Grades_

1.4.3.1 Merge Project Title Count with project_data

In [18]:
```
# Add count (total number of words) in Project Title in each row.

project_title_count = project_data['project_title'].str.split().str.len()
project_data['project_title_count'] = project_title_count
project_data.head(1)
```

Out[18]:

	Unnamed: 0	id	teacher_id	school_state	Date	project_grade_c
0	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	GA	2016-04-27 00:53:00	Grades_

1 rows × 21 columns

1.4.3.2 Essay count of words for each row and merge with project_data

In [19]:

```
# Add count (total number of words) in essay in each row.

essay_count = project_data['essay'].str.split().str.len()
project_data['essay_count'] = essay_count
project_data.head(1)
```

Out[19]:

	Unnamed: 0	id	teacher_id	school_state	Date	project_grade_c
0	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	GA	2016-04-27 00:53:00	Grades_

1 rows × 22 columns

In [20]:

```
#Convert NaN value to mean of the column
project_data.fillna(project_data.mean(), inplace=True)
project_data.head(1)
```

Out[20]:

	Unnamed: 0	id	teacher_id	school_state	Date	project_grade_c
0	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	GA	2016-04-27 00:53:00	Grades_

1 rows × 22 columns

In [21]:

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

# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stra
tify=y)
```

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

In [22]:

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

# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stra
tify=y)
```

In [23]:

```
catogories_essay = list(project_data['essay'].values)
# remove special characters from list of strings python: https://stackoverflo
w.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-fr
om-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-strin
g-in-python
cat_essay_list = []
for i in catogories_essay:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Scienc
e", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on
space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to
replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(emp
ty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the tra
iling spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_essay_list.append(temp.strip())

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

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

In [24]:

```
catogories_title = list(project_data['project_title'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
project_title_list = []
for i in catogories_title:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    project_title_list.append(temp.strip())

#project_data.drop('project_title', axis=1, inplace=True)
#project_data['project_title'] = project_title_list

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

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

In [25]:

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

```
100%|██████████| 33500/33500 [00:15<00:00, 2121.29it/s]
```

In [26]:

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

```
100%|██████████| 16500/16500 [00:07<00:00, 2094.40it/s]
```

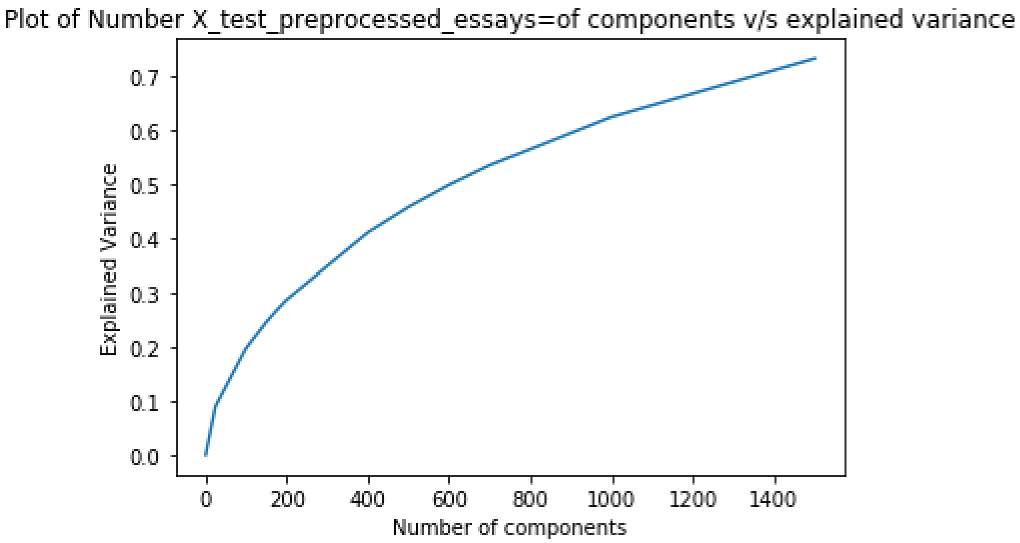
TruncatedSVD

Split TrancatedSVd Data:

Transcatred SVD Data Using Elbow Method for Test data. Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`) using elbow method : numerical data

In [27]:
```
#https://medium.com/swlh/truncated-singular-value-decomposition-svd-using-amazon-food-reviews-891d97af5d8d
from sklearn.decomposition import TruncatedSVD
# Program to find the optimal number of components for Truncated SVD
n_comp = [1,25,100,150,175,200,400,500,600,700,1000,1500] # different values of components
explained = [] # explained variance ratio for each component of Truncated SVD
tfidf_vec = TfidfVectorizer(analyzer="word", max_features=5000, ngram_range=(1,2))
for x in n_comp:
    svd = TruncatedSVD(n_components=x)
    Xtest_tfidf_df = tfidf_vec.fit_transform(X_test_preprocessed_essays)
    Xtest_svd_tfidf_essay = svd.fit_transform(Xtest_tfidf_df)
    explained.append(svd.explained_variance_ratio_.sum())
    print("Number of components = %r and explained variance = %r"%(x,svd.explained_variance_ratio_.sum()))
plt.plot(n_comp, explained)
plt.xlabel('Number of components')
plt.ylabel("Explained Variance")
plt.title("Plot of Number X_test_preprocessed_essays=of components v/s explained variance")
plt.show()
```

```
Number of components = 1 and explained variance = 0.001991129628392215
Number of components = 25 and explained variance = 0.09163475095436638
Number of components = 100 and explained variance = 0.19869342386626807
Number of components = 150 and explained variance = 0.2471785942579518
Number of components = 175 and explained variance = 0.268412405256456
Number of components = 200 and explained variance = 0.28800954276606605
Number of components = 400 and explained variance = 0.41108771279063605
Number of components = 500 and explained variance = 0.4582001377675169
Number of components = 600 and explained variance = 0.49921927325529514
Number of components = 700 and explained variance = 0.5355974291925665
Number of components = 1000 and explained variance = 0.6248619840453806
Number of components = 1500 and explained variance = 0.7327014589387854
```



Apply N_comp found fron elbow method as logic givren abve

In [28]:

```
# Since we found  n_comp  using elbow method using above logic, we will apply
 here to get
#best value of Xtest_svd_tfidf_essay
from sklearn.decomposition import TruncatedSVD
tfidf_vec = TfidfVectorizer(analyzer="word", max_features=5000, ngram_range=(1
,2))
n_comp = 600
svd = TruncatedSVD(n_components=n_comp)
Xtest_tfidf_df = tfidf_vec.fit_transform(X_test_preprocessed_essays)
Xtest_svd_tfidf_essay = svd.fit_transform(Xtest_tfidf_df)
print (Xtest_svd_tfidf_essay)
```

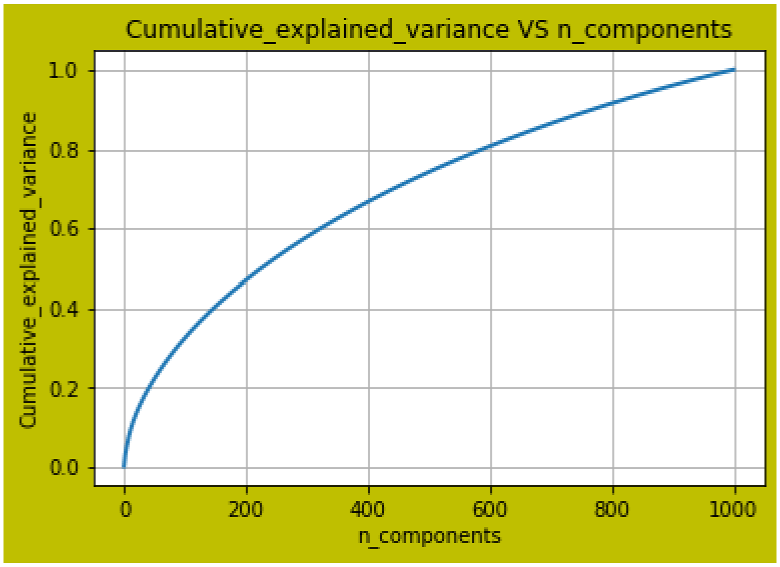
```
[[ 0.29275164 -0.05451869 -0.1985749  ...  0.01039873  0.01193932
  -0.01989576]
 [ 0.37778191 -0.08680799  0.14571252 ... -0.02035208  0.00110428
  -0.03320378]
 [ 0.22783942  0.00451241 -0.05978316 ...  0.00269138  0.02673616
  -0.05528908]
 ...
 [ 0.21897129 -0.01654926  0.00730349 ... -0.00147665 -0.01300295
   0.01368082]
 [ 0.29588604 -0.05155958 -0.02718337 ... -0.03478232 -0.01619516
   0.00681362]
 [ 0.34862655 -0.15308058  0.14082668 ...  0.03058579 -0.00704548
   0.01387263]]
```

Transcatred SVD Data Using Elbow Method for Train Data. Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`) using elbow method : numerical data

In [29]:

```
#https://medium.com/swlh/truncated-singular-value-decomposition-svd-using-amazon-food-reviews-891d97af5d8d
from sklearn.decomposition import TruncatedSVD
#Apply TruncatedSVD on TfidfVectorizer of essay text, choose the number of components (`n_components`) using elbow method : numerical data

# Program to find the optimal number of components for Truncated SVD
n_comp = [50,60,70,100,150,175,200,300,400,600,700,1000] # different values of components
explained = [] # explained variance ratio for each component of Truncated SVD
tfidf_vec = TfidfVectorizer(analyzer="word", max_features=5000, ngram_range=(1,2))
MaxExp = -1 # Max Explained varience
Max_svd = 0 # initially 0

for x in n_comp:
    svd = TruncatedSVD(n_components=x)
    Xtrain_tfidf_df = tfidf_vec.fit_transform(X_train_preprocessed_essays)
    Xtrain_svd_tfidf_essay = svd.fit_transform(Xtrain_tfidf_df)
    explained.append(svd.explained_variance_ratio_.sum())
    exp_sum = svd.explained_variance_ratio_.sum()
    if exp_sum >  MaxExp :
            Max_svd = svd
            MaxExp  = exp_sum

    print("Number of components = %r and explained variance = %r"%(x,svd.explained_variance_ratio_.sum()))
print("MaxExp==" ,MaxExp )
percentage_var_explained =  Max_svd .explained_variance_ / np.sum( Max_svd .explained_variance_)
cum_var_explained = np.cumsum(percentage_var_explained)

# Plotting for  MaxExp value in  list_component

fig4 = plt.figure( facecolor='y', edgecolor='k')
plt.clf()
plt.plot( cum_var_explained , linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.title("Cumulative_explained_variance VS n_components")
plt.show()
```

```
Number of components = 50 and explained variance = 0.13319953997475997
Number of components = 60 and explained variance = 0.14730900674093403
Number of components = 70 and explained variance = 0.16031100159812894
Number of components = 100 and explained variance = 0.19473431095052726
Number of components = 150 and explained variance = 0.24192090510737124
Number of components = 175 and explained variance = 0.262484714094932
Number of components = 200 and explained variance = 0.28134859392835837
Number of components = 300 and explained variance = 0.3466773793400977
Number of components = 400 and explained variance = 0.3999757759879922
Number of components = 600 and explained variance = 0.48479169375192105
Number of components = 700 and explained variance = 0.5196980930721529
Number of components = 1000 and explained variance = 0.6053597075838744
MaxExp== 0.6053597075838744
```



Apply N_comp found fron elbow method as logic givren abve

In [30]:

```
# Since we found  n_comp  using elbow method using above logic, we will apply
 here to get
#best value of Xrain_svd_tfidf_essay
from sklearn.decomposition import TruncatedSVD
tfidf_vec = TfidfVectorizer(analyzer="word", max_features=5000, ngram_range=(1
,2))
n_comp = 600
svd = TruncatedSVD(n_components=n_comp)
#Xtest_tfidf_df = tfidf_vec.fit_transform(X_test_preprocessed_essays)
#Xtest_svd_tfidf_essay = svd.fit_transform(Xtest_tfidf_df)

Xtrain_tfidf_df = tfidf_vec.fit_transform(X_train_preprocessed_essays)
Xtrain_svd_tfidf_essay = svd.fit_transform(Xtrain_tfidf_df)
#print (Xtrain_svd_tfidf_essay)
```

Step 3: Standardize (normalize) the data scale to prep for DecisionTree.

Because the distance between pairs of points plays a critical part on the classification, it is necessary to normalize the data This will generate an array of values.

1.4.1 Vectorizing Categorical data

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

Vectorization of clean_categories for X_train,X_test

In [31]:

```python
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000,lowercase=False, binary=True)
X_train_categories_one_hot = vectorizer.fit_transform(X_train['clean_categories'].values)
X_test_categories_one_hot = vectorizer.transform(X_test['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix X_train_categories_one_hot  after one hot encodig ",X_train_categories_one_hot.shape)
print("Shape of matrix X_test_categories_one_hot after one hot encodig ",X_test_categories_one_hot.shape)
```

```
['AppliedLearning', 'AppliedLearning Health_Sports', 'AppliedLearning History_Civics', 'AppliedLearning Literacy_Language', 'AppliedLearning Math_Science', 'AppliedLearning Music_Arts', 'AppliedLearning SpecialNeeds', 'Care_Hunger', 'Health_Sports', 'Health_Sports AppliedLearning', 'Health_Sports History_Civics', 'Health_Sports Literacy_Language', 'Health_Sports Math_Science', 'Health_Sports Music_Arts', 'Health_Sports SpecialNeeds', 'Health_Sports Warmth', 'Health_Sports Warmth Care_Hunger', 'History_Civics', 'History_Civics AppliedLearning', 'History_Civics Literacy_Language', 'History_Civics Math_Science', 'History_Civics Music_Arts', 'History_Civics SpecialNeeds', 'Literacy_Language', 'Literacy_Language AppliedLearning', 'Literacy_Language Health_Sports', 'Literacy_Language History_Civics', 'Literacy_Language Math_Science', 'Literacy_Language Music_Arts', 'Literacy_Language SpecialNeeds', 'Math_Science', 'Math_Science AppliedLearning', 'Math_Science Health_Sports', 'Math_Science History_Civics', 'Math_Science Literacy_Language', 'Math_Science Music_Arts', 'Math_Science SpecialNeeds', 'Music_Arts', 'Music_Arts SpecialNeeds', 'SpecialNeeds', 'SpecialNeeds Music_Arts', 'Warmth', 'Warmth Care_Hunger']
Shape of matrix X_train_categories_one_hot  after one hot encodig  (33500, 43)
Shape of matrix X_test_categories_one_hot after one hot encodig  (16500, 43)
```

In [32]:

```python
### Vectorization of project_grade_category for X_train,X_test
```

In [33]:

```python
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000, lowercase=False, binary=True)
X_train_project_grade_category_one_hot = vectorizer.fit_transform(X_train['project_grade_category'].values)
X_test_project_grade_category_one_hot = vectorizer.transform(X_test['project_grade_category'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix X_train_project_grade_category_one_hot  after one hot encodig ",X_train_project_grade_category_one_hot.shape)
print("Shape of matrix X_test_project_grade_category_one_hot after one hot encodig ",X_test_project_grade_category_one_hot.shape)
```

```
['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
Shape of matrix X_train_project_grade_category_one_hot  after one hot encodig 
(33500, 4)
Shape of matrix X_test_project_grade_category_one_hot after one hot encodig 
(16500, 4)
```

In [34]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000, l
owercase=False, binary=True)
X_train_school_state_one_hot = vectorizer.fit_transform(X_train['school_state'
].values)
X_test_school_state_one_hot = vectorizer.transform(X_test['school_state'].valu
es)
print(vectorizer.get_feature_names())
print("Shape of matrix X_train_school_state_one_hot  after one hot encodig ",X
_train_school_state_one_hot.shape)
print("Shape of matrix X_test_school_state_one_hot after one hot encodig ",X_t
est_school_state_one_hot.shape)
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'I
A', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO',
'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY', 'OH', 'OK', 'OR',
'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV', 'WY']
Shape of matrix X_train_school_state_one_hot  after one hot encodig  (33500,
51)
Shape of matrix X_test_school_state_one_hot after one hot encodig  (16500, 5
1)
```

Vectorization of clean_subcategories for X_train,X_test

In [35]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000,vo
cabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=True)
X_train_sub_categories_one_hot = vectorizer.fit_transform(X_train['clean_subca
tegories'].values)
X_test_sub_categories_one_hot = vectorizer.transform(X_test['clean_subcategori
es'].values)

print(vectorizer.get_feature_names())
print("Shape of matrix X_train_sub_categories_one_hot  after one hot encodig "
,X_train_sub_categories_one_hot.shape)
print("Shape of matrix X_test_sub_categories_one_hot after oneX_test_sub_categ
ories_one_hot  hot encodig ",X_test_sub_categories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement',
'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducati
on', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'CharacterE
ducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_Geo
graphy', 'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness', 'Env
ironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'Spec
ialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix X_train_sub_categories_one_hot  after one hot encodig  (3350
0, 30)
Shape of matrix X_test_sub_categories_one_hot after oneX_test_sub_categories_
one_hot  hot encodig  (16500, 30)
```

In [36]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_ca
tegory also
```

TFIDF of preprocessed_essays for X_train,X_test

In [37]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train_preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
X_train_dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_mode
l.idf_)))
X_train_tfidf_words = set(tfidf_model.get_feature_names())
print (len(X_train_tfidf_words))
X_train_tfidf = tfidf_model.transform(X_train_preprocessed_essays)
X_test_tfidf = tfidf_model.transform(X_test_preprocessed_essays)
X_test_dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model
.idf_)))
X_test_tfidf_words = set(tfidf_model.get_feature_names())
```

```
35503
```

1.4.2 Vectorizing Text data

In [38]:

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

Vectorization of preprocessed_essays for X_train,X_test

In [39]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored
in this list
for sentence in tqdm(X_train_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_avg_w2v_vectors.append(vector)

print(len(X_train_avg_w2v_vectors))
print(len(X_train_avg_w2v_vectors[0]))
```

```
100%|██████████| 33500/33500 [00:08<00:00, 4138.87it/s]

33500
300
```

In [40]:

```
# average Word2Vec
# compute average word2vec for each review.
X_test_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored
 in this list
for sentence in tqdm(X_test_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_avg_w2v_vectors.append(vector)

print(len(X_test_avg_w2v_vectors))
print(len(X_test_avg_w2v_vectors[0]))
```

```
100%|██████████| 16500/16500 [00:04<00:00, 4006.13it/s]

16500
300
```

In [41]:

```
## TFIDF-W2W Vecorization
```

In [42]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_tfidf_w2v_vectors_pessays = []; # the avg-w2v for each sentence/review
is stored in this list
for sentence in tqdm(X_train_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revie
w
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in X_train_tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf v
alue((sentence.count(word)/len(sentence.split())))
            tf_idf = X_train_dictionary[word]*(sentence.count(word)/len(senten
ce.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_tfidf_w2v_vectors_pessays.append(vector)

print(len(X_train_tfidf_w2v_vectors_pessays))
print(len(X_train_tfidf_w2v_vectors_pessays[0]))
```

```
100%|██████████| 33500/33500 [00:55<00:00, 601.07it/s]

33500
300
```

In [43]:

```
# average Word2Vec
# compute average word2vec for each review.
X_test_tfidf_w2v_vectors_pessays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in X_test_tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = X_test_dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_tfidf_w2v_vectors_pessays.append(vector)

print(len(X_test_tfidf_w2v_vectors_pessays))
print(len(X_test_tfidf_w2v_vectors_pessays[0]))
```

```
100%|██████████| 16500/16500 [00:27<00:00, 597.63it/s]

16500
300
```

In [44]:

```
# average Word2Vec
# compute average word2vec for each review.
X_test_tfidf_w2v_vectors_ptitle = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in X_test_tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = X_test_dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_tfidf_w2v_vectors_ptitle.append(vector)

print(len(X_test_tfidf_w2v_vectors_ptitle))
print(len(X_test_tfidf_w2v_vectors_ptitle[0]))
```

```
100%|██████████| 16500/16500 [00:00<00:00, 88414.51it/s]

16500
300
```

In [45]:

```
# average Word2Vec
# compute average word2vec for each review.

X_train_tfidf_w2v_vectors_ptitle = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in X_train_tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = X_train_dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_tfidf_w2v_vectors_ptitle.append(vector)

print(len(X_train_tfidf_w2v_vectors_ptitle))
print(len(X_train_tfidf_w2v_vectors_ptitle[0]))
```

```
100%|██████████| 33500/33500 [00:00<00:00, 96491.10it/s]

33500
300
```

Vectorization of teacher_prefix for X_train,X_test, X_cv

In [46]:

```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), vocabulary=list(sorted_teacher_dict.keys()),max_features=5000, lowercase=False, binary=True)
X_train_teacher_prefix_data = X_train['teacher_prefix']

X_train_teacher_prefix_data.fillna("Mrs.", inplace = True)

teacher_prefix_notnull = X_train_teacher_prefix_data[pd.notnull(X_train_teacher_prefix_data)]

vectorizer.fit(teacher_prefix_notnull.values)

#print(vectorizer.get_feature_names())

#print(teacher_prefix_notnull.values)

X_train_teacher_prefix_one_hot = vectorizer.fit_transform(teacher_prefix_notnull.values)
print("Shape of matrix after one hot encodig ",X_train_teacher_prefix_one_hot.shape)
```

```
Shape of matrix after one hot encodig  (33500, 30)
```

In [47]:

```
# we use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer
vectorizer1 = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000,vocabulary=list(sorted_teacher_dict.keys()), lowercase=False, binary=True)
X_test_teacher_prefix_data = X_test['teacher_prefix']
X_test_teacher_prefix_data.fillna("Mrs.", inplace = True)
teacher_prefix_notnull = X_test_teacher_prefix_data[pd.notnull(X_test_teacher_prefix_data)]
vectorizer.fit(teacher_prefix_notnull.values)
X_test_teacher_prefix_one_hot = vectorizer1.transform(teacher_prefix_notnull.values)
print("Shape of matrix after one hot encodig ",X_test_teacher_prefix_one_hot.shape)
```

```
Shape of matrix after one hot encodig  (16500, 30)
```

Vectorization of price for X_train,X_test

In [48]:
```
X_train.head(1)
X_test.head(1)
```

Out[48]:

	Unnamed: 0	id	teacher_id	school_state	Date	project_gra
28474	60772	p233048	54f205cf700a70f77b1f60cd8b138a4e	NV	2016-10-26 11:39:26	Gra

1 rows × 21 columns

In [49]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

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

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1
))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))
X_train_price_norm= X_train_price_norm.reshape(-1,1)
X_test_price_norm=X_test_price_norm.reshape(-1,1)

print("After vectorizations")
print (type(X_train_price_norm))
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)

print("="*100)
```

```
After vectorizations
<class 'numpy.ndarray'>
(33500, 1) (33500,)
(16500, 1) (16500,)
=============================================================================
=======================
```

Normalization of Project Title Count.

In [50]:

```python
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

#print (X_train['project_title_count'])

X_train_project_title_count_norm = normalizer.fit_transform(X_train['project_title_count'].values.reshape(1,-1))
X_test_project_title_count_norm = normalizer.transform(X_test['project_title_count'].values.reshape(1,-1))

X_train_project_title_count_norm= X_train_project_title_count_norm.reshape(-1,1)
X_test_project_title_count_norm=X_test_project_title_count_norm.reshape(-1,1)

print("After vectorizations")
print(X_train_project_title_count_norm.shape, y_train.shape)
print(X_test_project_title_count_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
=============================================================================
=======================
```

Normalization of essay count words.

In [51]:

```python
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

#print (X_train['project_title_count'])

X_train_essay_count_norm = normalizer.fit_transform(X_train['essay_count'].values.reshape(1,-1))
X_test_essay_count_norm = normalizer.transform(X_test['essay_count'].values.reshape(1,-1))

X_train_essay_count_norm= X_train_essay_count_norm.reshape(-1,1)
X_test_essay_count_norm=X_test_essay_count_norm.reshape(-1,1)

print("After vectorizations")
print(X_train_essay_count_norm.shape, y_train.shape)
print(X_test_essay_count_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
=============================================================================
=======================
```

In [52]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

#print (X_train['project_title_count'])

X_train_quantity_norm = normalizer.fit_transform(X_train['quantity'].values.reshape(1,-1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(1,-1))

X_train_quantity_norm= X_train_quantity_norm.reshape(-1,1)
X_test_quantity_norm=X_test_quantity_norm.reshape(-1,1)

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

In [53]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

X_train_teacher_number_of_previously_posted_projects_norm = normalizer.fit_transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_test_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

X_train_teacher_number_of_previously_posted_projects_norm= X_train_teacher_number_of_previously_posted_projects_norm.reshape(-1,1)
X_test_teacher_number_of_previously_posted_projects_norm=X_test_teacher_number_of_previously_posted_projects_norm.reshape(-1,1)

print("After vectorizations")
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

Bag of words of preprocessed_essays for X_train,X_test

In [54]:

```
# We are considering only the words which appeared in at least 10 documents(ro
ws or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4),  max_features=5000)
X_train_text_bow = vectorizer.fit_transform(X_train_preprocessed_essays)
X_test_text_bow = vectorizer.transform(X_test_preprocessed_essays)

print("Shape of matrix X_train_text_bow after one hot encodig ",X_train_text_b
ow.shape)
print("Shape of matrix X_test_text_bow after one hot encodig ",X_test_text_bow
.shape)
```

```
Shape of matrix X_train_text_bow after one hot encodig  (33500, 5000)
Shape of matrix X_test_text_bow after one hot encodig  (16500, 5000)
```

Bag of words of project_title for X_train,X_test

In [55]:

```
# PROJECT_TITLE BOW
# We are considering only the words which appeared in at least 10 documents(ro
ws or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
X_train_project_title_bow = vectorizer.fit_transform(X_train['project_title'])
X_test_project_title_bow = vectorizer.transform(X_test['project_title'])

print("Shape of matrix X_train_project_title_bow after one hot encodig ",X_tra
in_project_title_bow .shape)
print("Shape of matrix X_test_project_title_bow after one hot encodig ",X_test
_project_title_bow .shape)
```

```
Shape of matrix X_train_project_title_bow after one hot encodig  (33500, 395
0)
Shape of matrix X_test_project_title_bow after one hot encodig  (16500, 3950)
```

TFIDF of preprocessed_essays for X_train,X_test

In [56]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X_train_text_tfidf = vectorizer.fit_transform(X_train_preprocessed_essays)
X_test_text_tfidf = vectorizer.transform(X_test_preprocessed_essays)

print("Shape of matrix X_train_text_tfidf after one hot encodig ",X_train_text
_tfidf.shape)
print("Shape of matrix X_test_text_tfidf after one hot encodig ",X_test_text_t
fidf.shape)
```

```
Shape of matrix X_train_text_tfidf after one hot encodig  (33500, 35503)
Shape of matrix X_test_text_tfidf after one hot encodig  (16500, 35503)
```

TFIDF of Project Title for X_train,X_test

In [57]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
X_train_project_title_tfidf = vectorizer.fit_transform((X_train['project_title']))
X_test_project_title_tfidf = vectorizer.transform((X_test['project_title']))

print("Shape of matrix  X_train_project_title_tfidf after one hot encodig ",X_train_project_title_tfidf.shape)
print("Shape of matrix  X_test_project_title_tfidf after one hot encodig ",X_test_project_title_tfidf.shape)
```

```
Shape of matrix  X_train_project_title_tfidf after one hot encodig  (33500, 1653)
Shape of matrix  X_test_project_title_tfidf after one hot encodig  (16500, 1653)
```

TFIDF AVG W2V for Project Title for X_train,X_test

In [58]:

```
# average Word2Vec
# compute average word2vec for each review.
X_train_project_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_train_project_title_avg_w2v_vectors.append(vector)

print(len(X_train_project_title_avg_w2v_vectors))
print(len(X_train_project_title_avg_w2v_vectors[0]))
```

```
100%|██████████| 33500/33500 [00:00<00:00, 130189.70it/s]

33500
300
```

In [59]:

```
# average Word2Vec
# compute average word2vec for each review.
X_test_project_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    X_test_project_title_avg_w2v_vectors.append(vector)

print(len(X_test_project_title_avg_w2v_vectors))
print(len(X_test_project_title_avg_w2v_vectors[0]))
```

```
100%|██████████| 16500/16500 [00:00<00:00, 130579.36it/s]

16500
300
```

In [60]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((X_train_price_norm,X_train_sub_categories_one_hot,X_train_teacher_prefix_one_hot)).tocsr()
X_te = hstack((X_test_price_norm,X_test_sub_categories_one_hot,X_test_teacher_prefix_one_hot)).tocsr()

#print (X_train_price_norm)
X_tr_bow = hstack((X_train_price_norm,X_train_sub_categories_one_hot,X_train_teacher_prefix_one_hot,\
                  X_train_project_title_bow )).tocsr()
X_te_bow = hstack((X_test_price_norm,X_test_sub_categories_one_hot,X_test_teacher_prefix_one_hot,\
                   X_test_project_title_bow)).tocsr()

X_tr_tfidf = hstack((X_train_sub_categories_one_hot,X_train_teacher_prefix_one_hot,X_train_price_norm,\
                     X_train_project_title_tfidf)).tocsr()
X_te_tfidf = hstack((X_test_sub_categories_one_hot,X_test_teacher_prefix_one_hot,X_test_price_norm,\
                     X_test_project_title_tfidf)).tocsr()

X_tr_tfidf_w2v = hstack((X_train_sub_categories_one_hot,X_train_teacher_prefix_one_hot,X_train_price_norm,\
                         X_train_project_title_avg_w2v_vectors,X_train_tfidf_w2v_vectors_pessays)).tocsr()
X_te_tfidf_w2v = hstack((X_test_sub_categories_one_hot,X_test_teacher_prefix_one_hot,X_test_price_norm,\
                         X_test_project_title_avg_w2v_vectors,X_test_tfidf_w2v_vectors_pessays)).tocsr()


X_tr_avg_w2v = hstack((X_train_sub_categories_one_hot,X_train_teacher_prefix_one_hot,X_train_price_norm,\
                       X_train_project_title_avg_w2v_vectors)).tocsr()
X_te_avg_w2v = hstack((X_test_sub_categories_one_hot,X_test_teacher_prefix_one_hot,X_test_price_norm,\
                       X_test_project_title_avg_w2v_vectors)).tocsr()

#set 5

X_te_set5 =hstack((X_test_school_state_one_hot,X_test_categories_one_hot,X_test_sub_categories_one_hot,\
                  X_test_teacher_prefix_one_hot,X_test_quantity_norm,X_test_price_norm,X_test_project_grade_category_one_hot,\
                  X_test_teacher_number_of_previously_posted_projects_norm,X_test_price_norm,\
                  X_test_project_title_count_norm,X_test_essay_count_norm))

X_tr_set5 =hstack((X_train_school_state_one_hot,X_train_categories_one_hot, X_train_sub_categories_one_hot,\
                  X_train_teacher_prefix_one_hot,X_train_quantity_norm,X_train_price_norm,X_train_project_grade_category_one_hot,\
                  X_train_teacher_number_of_previously_posted_projects_norm,X_train_price_norm,X_train_project_title_count_norm,\
                  X_train_essay_count_norm))

X_te_tfidf_avg_w2v = hstack((X_test_school_state_one_hot,X_test_categories_one
_hot,X_test_sub_categories_one_hot,\
                  X_test_tfidf_w2v_vectors_pessays,X_test_tfidf_w2v_vectors_pt
itle))

X_tr_tfidf_avg_w2v = hstack((X_train_school_state_one_hot,X_train_categories_o
ne_hot, X_train_sub_categories_one_hot,\
                  X_train_tfidf_w2v_vectors_pessays,X_train_tfidf_w2v_vectors_
ptitle))


## Set 6 id added to cover TransactSVD logic.


X_te_set6 =hstack((X_test_school_state_one_hot,X_test_categories_one_hot,X_tes
t_sub_categories_one_hot,\
                  X_test_teacher_prefix_one_hot,X_test_quantity_norm,X_test_pr
oject_grade_category_one_hot,\
                  X_test_teacher_number_of_previously_posted_projects_norm,X_t
est_price_norm,\
                  X_test_project_title_count_norm,X_test_tfidf_w2v_vectors_pes
says,X_test_essay_count_norm,\
                  X_test_tfidf_w2v_vectors_ptitle,Xtest_svd_tfidf_essay))

X_tr_set6 =hstack((X_train_school_state_one_hot,X_train_categories_one_hot,X_t
rain_sub_categories_one_hot,\
                  X_train_teacher_prefix_one_hot,X_train_quantity_norm,X_train
_project_grade_category_one_hot,\
                  X_train_teacher_number_of_previously_posted_projects_norm,X_
train_price_norm,\
                  X_train_project_title_count_norm,X_train_tfidf_w2v_vectors_p
essays,X_train_essay_count_norm,\
                  X_train_tfidf_w2v_vectors_ptitle,Xtrain_svd_tfidf_essay))

#print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)

print("="*100)

print(X_tr_tfidf.shape, y_train.shape)
print(X_te_tfidf.shape, y_test.shape)
print("="*100)

print(X_tr_set5.shape,y_train.shape )
print(X_te_set5.shape,y_test.shape )
print("="*100)

print(X_tr_set6.shape,y_train.shape )
print(X_te_set6.shape,y_test.shape )
print("="*100)
```

```
(33500, 61) (33500,)
(16500, 61) (16500,)
=============================================================================
=======================
(33500, 1714) (33500,)
(16500, 1714) (16500,)
=============================================================================
=======================
(33500, 164) (33500,)
(16500, 164) (16500,)
=============================================================================
=======================
(33500, 1363) (33500,)
(16500, 1363) (16500,)
=============================================================================
=======================
```

In [ ]:

Assignment 5: DecisionTree

1. **[Task-1] DecisionTree (either SGDClassifier with log loss, or DecisionTree) on these feature sets**

   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (`BOW with bi-grams` with `min_df=10` and `max_features=5000`)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (`TFIDF with bi-grams` with `min_df=10` and `max_features=5000`)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

2. **Hyper paramter tuning (find best hyper parameter which is "max depth and mininmum sample" corresponding the algorithm that you choose)**

   - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - Find the best hyper paramter in a loop
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning of "max depth and mininmum sample"

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.
   - 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

     (https://seaborn.pydata.org/generated/seaborn.heatmap.html)
     (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

   (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

4. **[Task-2] Apply DecisionTree on the below feature set Set 5 by finding the best hyper parameter as suggested in step 2 and step 3.**
5. Consider these set of features Set 5 :

   - **school_state** : categorical data
   - **clean_categories** : categorical data
   - **clean_subcategories** : categorical data
   - **project_grade_category** :categorical data
   - **teacher_prefix** : categorical data
   - **quantity** : numerical data

- **teacher_number_of_previously_posted_projects** : numerical data
- **price** : numerical data
- **sentiment score's of each of the essay** : numerical data
- **number of words in the title** : numerical data
- **number of words in the combine essays** : numerical data

And apply the DecisionTree on these features by finding the best hyper paramter as suggested in step 2 and step 3

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

6. **Conclusion** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

   (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library (https://seaborn.pydata.org/generated/seaborn.heatmap.html) link (http://zetcode.com/python/prettytable/)

2. DecisionTree

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

Apply DecisionTree 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 [61]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability e
stimates of the positive class
    # not the predicted outputs
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 4904
1%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [62]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

Split the normalized data into training and test sets

Logic below is simialr as covred in kanalysis_cross_validation(X,y), but here logic is for to calculate confusion matrix, acuarcy ration for best K as we already foound best K after trying best accuracy for multiple K values. We can apply in lopp "max depth and mininmum sample" for hyperparameter tuning, performance reporting, or both. The advantage of this approach is that the performance is less sensitive to unfortunate splits of data. In addition, it utilize data better since each example can be used for both training and validation/testing.

How to speculate the performance of the model using ROC Curve?

An excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means model has no class separation capacity whatsoever

Split the normalized data into training and test sets

This step is required to prepare us for the fitting (i.e. training) the #model later. The “X” variable is a collection of all the features. The “y” variable is the target label which specifies the #classification of 1 or 0 based. Our goal will be to identify which category the new data point should fall into. Evaluate the predictions. Evaluate the Model by reviewing the classification report or confusion matrix. By reviewing these tables, we are able to evaluate how accurate our model is with new values.

In [63]:

```
# This function is used for passsing various Hyper parameter which is "max dep
th and mininmum sample" in a loop,  and get the best parameter
# that would give best accuracy. For  each Hyperparemer, predicted value and a
ccuracy  is calculated.
# best Hyperparameter  is reHyper  for best accuracy. this is like gridCVSearc
h but shown plots for various hymer parametr.
# after best Hyperparam is returnbed, that is used  and again best AUC plot is
drawn.
# Items 4 in your query, first three items are covered here.

def DecisionTree_HyperParam_Analysis(X_train,y_train,X_test,y_test):
    from sklearn import model_selection
    from mlxtend.plotting import plot_decision_regions
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import roc_auc_score
    import math
    from mpl_toolkits.mplot3d import Axes3D
    import matplotlib.pyplot as plt

    # Import classification report and confusion matrix to evaluate prediction
s
    from sklearn.metrics import classification_report, confusion_matrix

    train_auc = []
    test_auc = []
    max_depth = [1, 5, 10, 20, 30, 50,100,200,300]
    min_samples_split =  [5, 10,30,50, 100,200, 300, 500,600,800,1000]
    best_accuracy=0.0001
    max_depthList =[]
    min_samples_splitList = []
    Bestmax_depth =-99
    Bestmin_samples_split =-99
    for depth in max_depth:
        for min_sample in min_samples_split:
            #HyperParameter = math.log(i)

            max_depthList.append(depth)
            min_samples_splitList.append(min_sample)
            model  = DecisionTreeClassifier(max_depth=depth,min_samples_split
= min_sample,class_weight = 'balanced')

            # fitting the model on crossvalidation train
            model.fit(X_train, y_train)

            # predict the response on the crossvalidation train
            y_train_pred = model.predict(X_train)  # predicting the value usin
g cross validation data.

            # predict the response on the crossvalidation test
            y_test_pred = model.predict(X_test)  # predicting the value using
 cross validation data.


            # evaluate CV accuracy
            acc = accuracy_score(y_test, y_test_pred, normalize=True) * float(
100)  # I get the accuracy score.

            #print ('Best Accuracy score = %s, For Best Depth = %s, and Best s
ample  = %s' % (acc, depth,min_sample))

            if acc > best_accuracy:
                best_accuracy =acc
                Bestmax_depth = depth
                Bestmin_samples_split=min_sample
                print ('Best Accuracy score = %s, For Best Depth = %s, and Bes
t sample  = %s' % (acc, depth,min_sample))

            train_auc.append(roc_auc_score(y_train,y_train_pred))
            test_auc.append(roc_auc_score(y_test, y_test_pred))
        else:
            continue
    ax = plt.axes(projection="3d")
    ax.plot3D (min_samples_splitList,max_depthList , train_auc)

    ax.set_xlabel('Min Samples Split')
    ax.set_ylabel('Max Depths')
    ax.set_zlabel('Train AUC')
    print ("Done")
    return Bestmax_depth,Bestmin_samples_split
```

In [64]:

```
#refer https://www.geeksforgeeks.org/generating-word-cloud-python/
import pandas as pd
import seaborn as sns

def WordCloudForFalsePositivePoints(x_test,y_test,y_test_pred):
    import numpy as np
    from wordcloud import WordCloud ,STOPWORDS
    y_test1 = y_test.tolist()
    y_test_pred1 = y_test_pred.tolist()
    x_test =  X_test['clean_subcategories'].values.tolist()

    data = {'corpus':x_test, "y_true":y_test1,"y_pred":y_test_pred1}
    data_frame=pd.DataFrame(data)

    false_positives = data_frame[(data_frame['y_true']== 0) & (data_frame['y_p
red'] == 1)]
    comment_words = ' '
    stopwords = set(STOPWORDS)
    for val in false_positives['corpus']:
       # print (val)

        # typecaste each val to string
        val = str(val)

        # split the value
        tokens = val.split()

        # Converts each token into lowercase
        for i in range(len(tokens)):
            tokens[i] = tokens[i].lower()

        for words in tokens:
            comment_words = comment_words + words + ' '

    wordcloud = WordCloud(width = 1800, height = 400,
                        background_color ='white',
                        stopwords = stopwords,
                        min_font_size = 10).generate(comment_words)

    # plot the WordCloud image
    plt.figure(figsize = (4, 4), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad = 0)
    plt.show()

    #Draw boxplot
    #print (data_frame.head(1))
    #sns.boxplot(y = 'y_true', x = 'corpus', data = data_frame)
```

In [65]:

```
def positive_false_points(X_train,y_train,X_test,y_test, Bestmax_depth,Bestmin
_samples_split):

    from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import roc_auc_score

    model = DecisionTreeClassifier(max_depth=Bestmax_depth,min_samples_split =
Bestmin_samples_split,\
                                   class_weight = 'balanced')
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
    WordCloudForFalsePositivePoints(X_test,y_test,y_test_pred)
```

In [66]:

```
# Plot Accuracy curce for best  Bestmax_depth,Bestmin_samples_split caluclated
already
# from fn DecisionTree_HyperParam_Analysis

def DecisionTree_for_Best_Hyper_Parameter(X_train,y_train,X_test,y_test, Bestm
ax_depth,Bestmin_samples_split):
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import roc_auc_score
    import matplotlib.pyplot as plt
    import numpy as np
    import pandas as pd

    model = DecisionTreeClassifier(max_depth=Bestmax_depth,min_samples_split =
Bestmin_samples_split,\
                                   class_weight = 'balanced')
    model.fit(X_train, y_train)

    y_train_pred = model.predict_proba(X_train)
    y_test_pred = model.predict_proba(X_test)

    #The ROC curve is plotted with TPR against the FPR where TPR is on y-axis
 and FPR is on the x-axis.
    train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred[:, 1
])
    test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred[:, 1])

    import seaborn as sns

    print("="*100)
    from sklearn.metrics import confusion_matrix
    best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
    print("Train confusion matrix")
    train_matrix = confusion_matrix(y_train, predict_with_best_t(y_train_pred
[:, 1], best_t))
    print(train_matrix)

    print("Test confusion matrix")
    test_matrix = confusion_matrix(y_test, predict_with_best_t(y_test_pred[:,
1], best_t))
    print(test_matrix)


    trainAUC= float("{0:.2f}".format(auc(train_fpr, train_tpr)))
    testAUC = float("{0:.2f}".format(auc(test_fpr, test_tpr)))

    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(trainAUC))
    plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(testAUC))

    # plot AUC curve. AUC curve should show best accuracy rate, since best apl
ha is used in the logic.
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(trainAUC))
    plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(testAUC))
    plt.legend()
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.title("AUC PLOT")
    plt.grid()

     # Confusiomatrix heatmap.

    fig, (ax1, ax2) = plt.subplots(1,2,sharex=True, sharey=True)
    # plt.figure(figsize=(30, 60))
    g1=sns.heatmap(test_matrix, annot=True,fmt='',cbar=True, linewidths =0.3,
ax=ax1)
    g1.set_xlabel("Test confusion matrix")
    g1.axes.get_xaxis().set_visible(True)
    g1.axes.get_yaxis().set_visible(False)

    g2=sns.heatmap(train_matrix,annot=True,fmt='',cbar=True,linewidths =0.3,ax
=ax2)
    g2.set_xlabel("Train confusion matrix")
    g2.axes.get_xaxis().set_visible(True)
    g2.axes.get_yaxis().set_visible(False)
    plt.show()
    return [trainAUC,testAUC]
```

In [67]:

```
# Remove unwanted features from model and regenerate model using better featur
es to see the impact on accuracy.

def DecisionTree_for_better_important_features(X_train,y_train,X_test,y_test,B
estmax_depth,Bestmin_samples_split):
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import roc_auc_score
    from sklearn.metrics import accuracy_score

    model = DecisionTreeClassifier(class_weight = 'balanced')
    model.fit(X_train, y_train)

##################################################################3
    # Calculate feature importances
    importances = model.feature_importances_
    newX_train =X_train.tocsr()[:,model.feature_importances_.argsort()[::-1][:
50]]
    newX_test =X_test.tocsr()[:,model.feature_importances_.argsort()[::-1][:50
]]

    # pass lesser X_train and X_test features to regenarte model and get the a
ccuracy using regenerated model.

    X_train= newX_train
    X_test = newX_test

    model = DecisionTreeClassifier(max_depth=Bestmax_depth,min_samples_split =
Bestmin_samples_split,\
                                   class_weight = 'balanced')
    model.fit(X_train, y_train)

    train_auc = []
    test_auc = []
    max_depth = [1, 5, 10, 20, 30, 50,100,200,300]
    min_samples_split =  [5, 10,30,50, 100,200, 300, 500,600,800,1000]
    best_accuracy=0.0001
    max_depthList =[]
    min_samples_splitList = []
    Bestmax_depth =-99
    Bestmin_samples_split =-99
    for depth in max_depth:
        for min_sample in min_samples_split:
            #HyperParameter = math.log(i)

            max_depthList.append(depth)
            min_samples_splitList.append(min_sample)
            model  = DecisionTreeClassifier(max_depth=depth,min_samples_split
= min_sample,class_weight = 'balanced')

            # fitting the model on crossvalidation train
            model.fit(X_train, y_train)

            # predict the response on the crossvalidation train
            y_train_pred = model.predict(X_train)  # predicting the value usin
g cross validation data.
```

```
            # predict the response on the crossvalidation test
            y_test_pred = model.predict(X_test)  # predicting the value using
 cross validation data.


            # evaluate CV accuracy
            acc = accuracy_score(y_test, y_test_pred, normalize=True) * float(
100)  # I get the accuracy score.

            #print ('Best Accuracy score = %s, For Best Depth = %s, and Best s
ample  = %s' % (acc, depth,min_sample))

            if acc > best_accuracy:
                best_accuracy =acc
                Bestmax_depth = depth
                Bestmin_samples_split=min_sample
                print ('Best Accuracy score = %s, For Best Depth = %s, and Bes
t sample  = %s' % (acc, depth,min_sample))

            train_auc.append(roc_auc_score(y_train,y_train_pred))
            test_auc.append(roc_auc_score(y_test, y_test_pred))
        else:
            continue
    ax = plt.axes(projection="3d")
    ax.plot3D (min_samples_splitList,max_depthList , train_auc)

    ax.set_xlabel('Min Samples Split')
    ax.set_ylabel('Max Depths')
    ax.set_zlabel('Train AUC')
    print ("Done")
    return Bestmax_depth,Bestmin_samples_split
```

In [68]:

```
# This function is used for passsing various Hyper parameter in a loop,  and get the best parameter
# that would give best accuracy. For  each Hyperparemer, predicted value and accuracy  is calculated.
# best Hyperparameter  is reHyper  for best accuracy. this is like gridCVSearch but shown plots for various hymer parametr.
# after best Hyperparam is returnbed, that is used  and again best AUC plot is drawn.
# Items 4 in your query, first three items are covered here.

def DecisionTree_GridSearch(X_train,y_train,X_test,y_test):
    from sklearn import model_selection
    from mlxtend.plotting import plot_decision_regions
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import roc_auc_score
    import math
    import pandas as pd
    from sklearn.model_selection import learning_curve, GridSearchCV

    # Import classification report and confusion matrix to evaluate predictions
    from sklearn.metrics import classification_report, confusion_matrix

    dt = DecisionTreeClassifier(class_weight = 'balanced')
    parameters = {'max_depth': [1, 5, 10, 20,30,50, 100, 500, 1000], 'min_samples_split': [5, 10, 20, 45, 75, 100, 135, 270, 500]}
    model = GridSearchCV(dt, parameters, cv=3, scoring='roc_auc',return_train_score=True)
    model.fit(X_train, y_train)

    #https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier.decision_function
    y_train_pred = model.predict_proba(X_train) [:,1]
    y_test_pred = model.predict_proba(X_test) [:,1]
    train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
    test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ERROR PLOTS")
    plt.grid(True)
    plt.show()
```

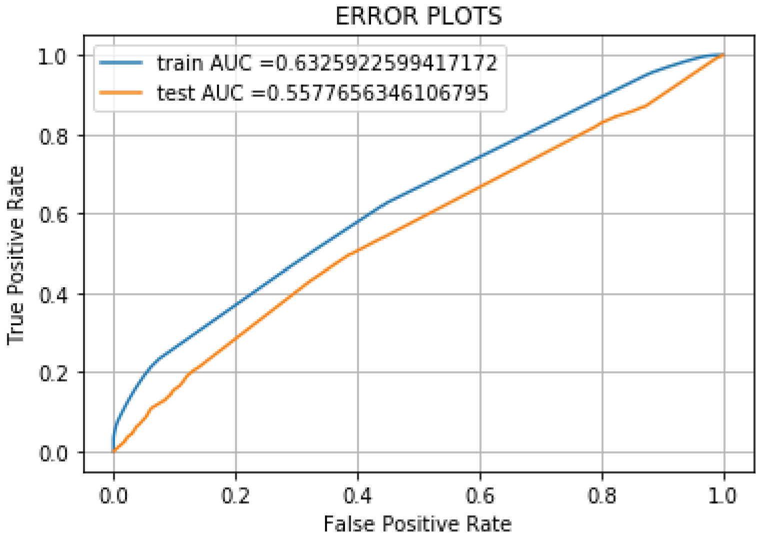
In [69]:

```
#feature_importance()
```

2.5 Feature selection for Best Hyper Parameter /font>

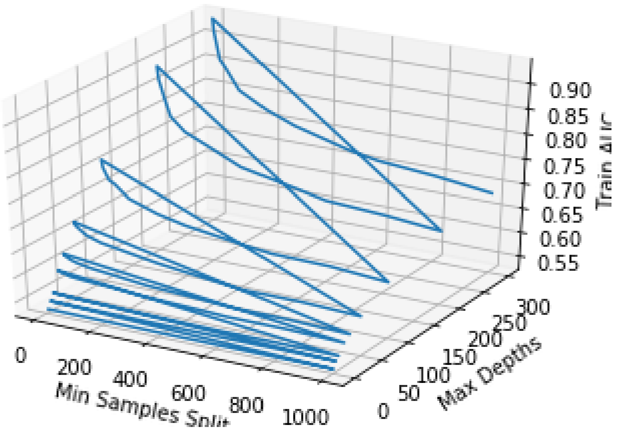
In [70]:
```
## 2.5.1 <font color='red'> Hyper Param-Analysis  <font color='blue'> categori
cal, numerical features + project_title(BOW) + preprocessed_essay (BOW),<font
 color='red'> SET 1</font>
```

In [71]:
```
# Check if gridsearch gives optimal solution.
DecisionTree_GridSearch(X_tr_bow,y_train,X_te_bow,y_test)
```



```
In [72]: bow_Bestmax_depth,bow_Bestmin_samples_split= DecisionTree_HyperParam_Analysis(
         X_tr_bow,y_train,X_te_bow,y_test)
         #print ("Hyper Param to apply is %s" % bow_hyperparam)
```

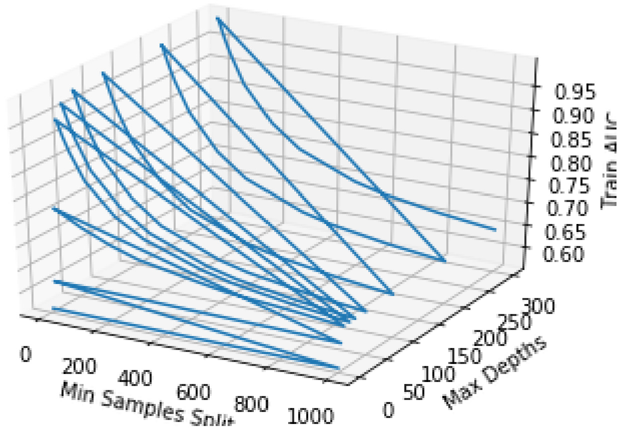
```
Best Accuracy score = 38.75757575757576, For Best Depth = 1, and Best sample
= 5
Best Accuracy score = 41.45454545454545, For Best Depth = 5, and Best sample
= 5
Best Accuracy score = 41.484848484848484, For Best Depth = 5, and Best sample
= 300
Best Accuracy score = 41.5030303030303, For Best Depth = 5, and Best sample
= 500
Best Accuracy score = 42.07272727272727, For Best Depth = 10, and Best sample
= 5
Best Accuracy score = 42.10909090909091, For Best Depth = 10, and Best sample
= 500
Best Accuracy score = 51.85454545454545, For Best Depth = 20, and Best sample
= 5
Best Accuracy score = 52.0, For Best Depth = 20, and Best sample  = 10
Best Accuracy score = 52.24848484848484, For Best Depth = 30, and Best sample
= 5
Best Accuracy score = 53.87878787878788, For Best Depth = 30, and Best sample
= 10
Best Accuracy score = 54.6969696969697, For Best Depth = 100, and Best sample
= 10
Best Accuracy score = 59.1939393939394, For Best Depth = 300, and Best sample
= 5
Best Accuracy score = 59.71515151515151, For Best Depth = 300, and Best sampl
e  = 100
Done
```



2.5.1 DecisionTree Hyper Param -Analysis on Set 6 , SET 6

```
In [73]: set6_Bestmax_depth,set6_Bestmin_samples_split = DecisionTree_HyperParam_Analys
         is(X_tr_set6,y_train,X_te_set6,y_test)
         print (set6_Bestmax_depth)

         #print ("Hyper Param to apply is %s" % set6_hyperparam)
```

```
Best Accuracy score = 49.527272727272724, For Best Depth = 1, and Best sample
= 5
Best Accuracy score = 57.418181818181814, For Best Depth = 5, and Best sample
= 5
Best Accuracy score = 57.896969696969705, For Best Depth = 5, and Best sample
= 500
Best Accuracy score = 58.193939393939395, For Best Depth = 5, and Best sample
= 800
Best Accuracy score = 59.61818181818182, For Best Depth = 5, and Best sample
= 1000
Best Accuracy score = 60.92727272727273, For Best Depth = 10, and Best sample
= 1000
Best Accuracy score = 70.07272727272728, For Best Depth = 20, and Best sample
= 5
Best Accuracy score = 73.15151515151516, For Best Depth = 30, and Best sample
= 5
Best Accuracy score = 75.10909090909091, For Best Depth = 50, and Best sample
= 5
Best Accuracy score = 75.26666666666667, For Best Depth = 100, and Best sampl
e  = 5
Done
100
```

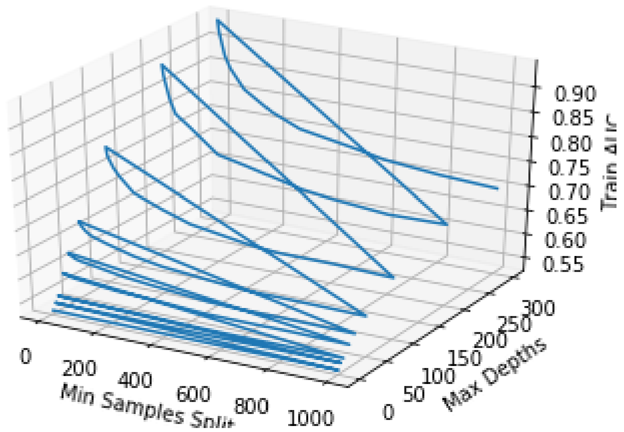


2.5.2 DecisionTree Hyper Param -Analysis on TFIDF Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF), SET 2

In [74]:

```
tfidf_Bestmax_depth,tfidf_Bestmin_samples_split= DecisionTree_HyperParam_Analy
sis(X_tr_tfidf,y_train,X_te_tfidf,y_test)
#print ("Hyper Param to apply is %s" % tfidf_hyperparam)
```

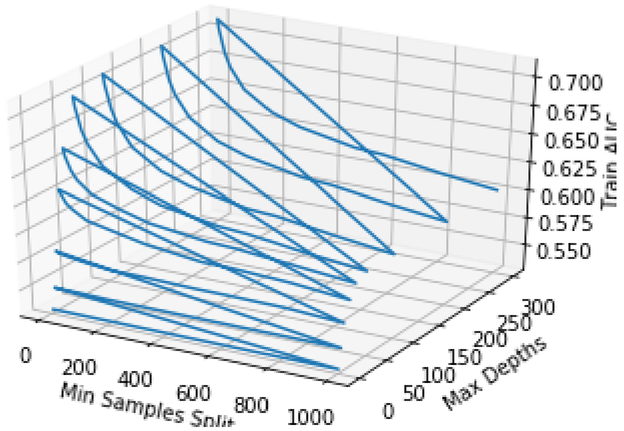
```
Best Accuracy score = 38.75757575757576, For Best Depth = 1, and Best sample
= 5
Best Accuracy score = 41.46060606060606, For Best Depth = 5, and Best sample
= 5
Best Accuracy score = 41.50909090909091, For Best Depth = 5, and Best sample
= 30
Best Accuracy score = 41.53333333333333, For Best Depth = 5, and Best sample
= 50
Best Accuracy score = 41.53939393939394, For Best Depth = 5, and Best sample
= 100
Best Accuracy score = 41.56363636363636, For Best Depth = 5, and Best sample
= 500
Best Accuracy score = 41.848484848484844, For Best Depth = 10, and Best sampl
e  = 5
Best Accuracy score = 41.86666666666667, For Best Depth = 10, and Best sample
= 200
Best Accuracy score = 42.0, For Best Depth = 10, and Best sample  = 300
Best Accuracy score = 42.018181818181816, For Best Depth = 10, and Best sampl
e  = 500
Best Accuracy score = 51.74545454545455, For Best Depth = 20, and Best sample
= 5
Best Accuracy score = 51.78181818181818, For Best Depth = 20, and Best sample
= 10
Best Accuracy score = 51.981818181818184, For Best Depth = 30, and Best sampl
e  = 600
Best Accuracy score = 52.01212121212121, For Best Depth = 30, and Best sample
= 800
Best Accuracy score = 52.06060606060606, For Best Depth = 30, and Best sample
= 1000
Best Accuracy score = 55.90909090909091, For Best Depth = 50, and Best sample
= 5
Best Accuracy score = 63.01212121212121, For Best Depth = 200, and Best sampl
e  = 5
Best Accuracy score = 64.74545454545455, For Best Depth = 300, and Best sampl
e  = 5
Done
```



2.5.3 Hyper Parameter-Analysis on AVG W2V - categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V), SET 3

In [75]:
```
avg_w2v_Bestmax_depth,avg_w2v_Bestmin_samples_split= DecisionTree_HyperParam_A
nalysis(X_tr_avg_w2v,y_train,X_te_avg_w2v,y_test)
#print ("Hyper Param to apply is %s" % avgw2v_hyperparam)
```

```
Best Accuracy score = 38.75757575757576, For Best Depth = 1, and Best sample
= 5
Best Accuracy score = 54.43636363636364, For Best Depth = 5, and Best sample
= 5
Best Accuracy score = 54.442424242424245, For Best Depth = 5, and Best sample
= 100
Best Accuracy score = 56.2060606060606, For Best Depth = 10, and Best sample
= 5
Best Accuracy score = 57.07878787878787, For Best Depth = 20, and Best sample
= 5
Best Accuracy score = 61.80606060606061, For Best Depth = 50, and Best sample
= 5
Best Accuracy score = 63.67272727272727, For Best Depth = 100, and Best sampl
e  = 5
Best Accuracy score = 63.733333333333334, For Best Depth = 200, and Best samp
le  = 10
Done
```

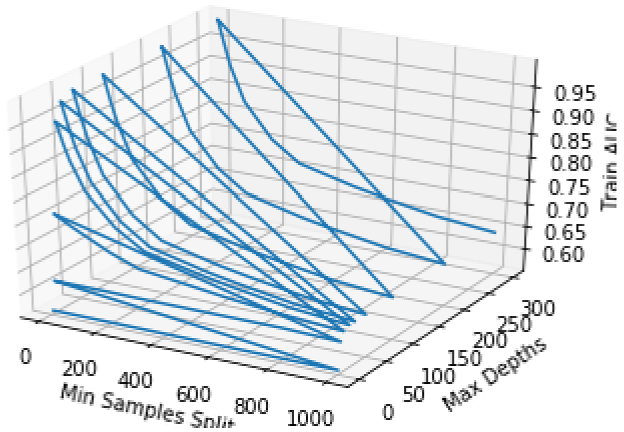


2.5.3 Hyper Parameter-Analysis on TFIDF W2V - categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V), SET 3

In [76]:

```
tfidf_w2v_Bestmax_depth,tfidf_w2v_Bestmin_samples_split= DecisionTree_HyperPar
am_Analysis(X_tr_tfidf_w2v,y_train,X_te_tfidf_w2v,y_test)
#print ("Hyper Param to apply is %s" % tfidfw2v_hyperparam)
```

```
Best Accuracy score = 60.915151515151514, For Best Depth = 1, and Best sample
= 5
Best Accuracy score = 61.509090909090915, For Best Depth = 10, and Best sampl
e  = 5
Best Accuracy score = 69.87272727272727, For Best Depth = 20, and Best sample
= 5
Best Accuracy score = 73.0969696969697, For Best Depth = 30, and Best sample
= 5
Best Accuracy score = 74.61212121212121, For Best Depth = 50, and Best sample
= 5
Best Accuracy score = 74.66666666666667, For Best Depth = 300, and Best sampl
e  = 5
Done
```



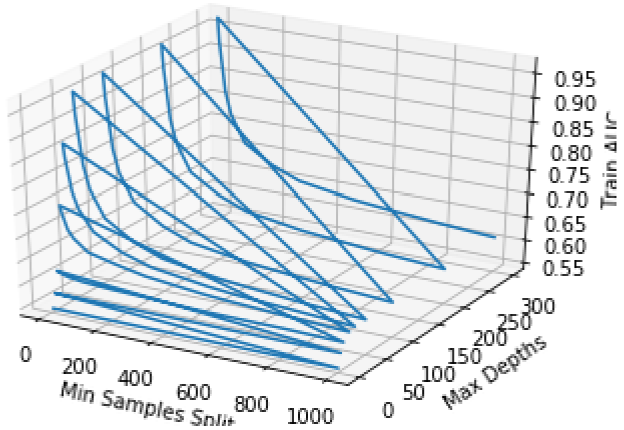
Hyper Param Analysis on TFIDF Set 5: categorical, numerical features + SET 5

DecisionTree Analysis on Best Hyper Parameter

In [77]:

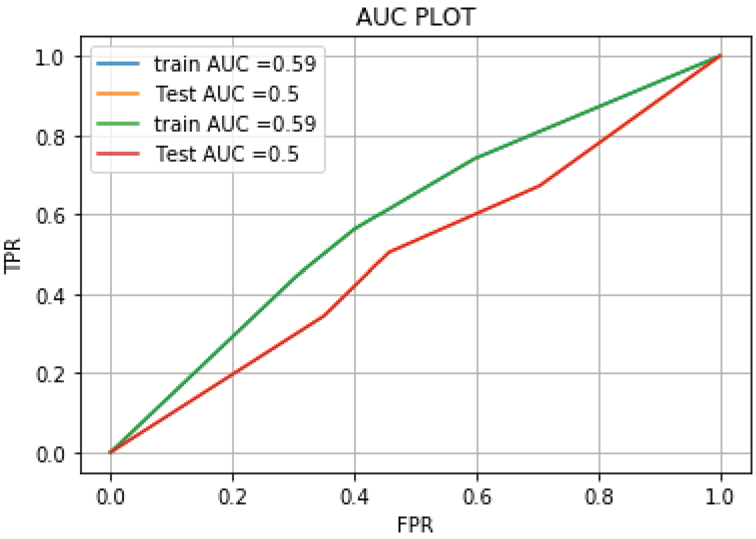
```
set5_Bestmax_depth, set5_Bestmin_samples_split = DecisionTree_HyperParam_Analysis(X_tr_set5,y_train,X_te_set5,y_test)
#print ("Hyper Param to apply is %s" % set5_hyperparam)
```

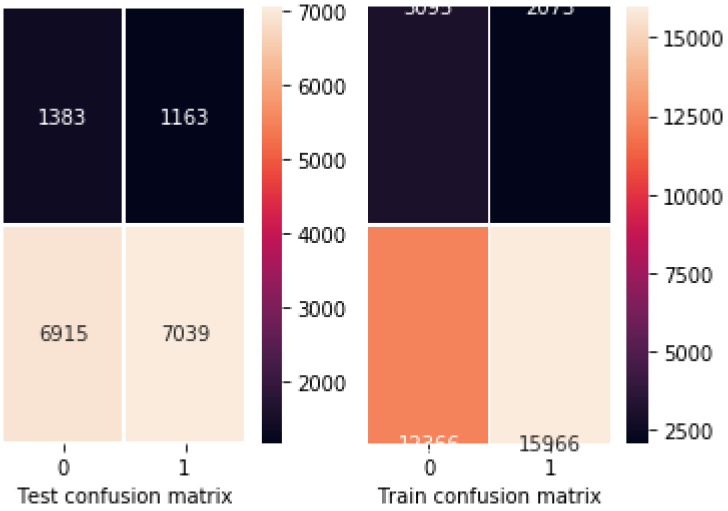
```
Best Accuracy score = 40.81818181818182, For Best Depth = 1, and Best sample = 5
Best Accuracy score = 71.98181818181818, For Best Depth = 5, and Best sample = 5
Best Accuracy score = 73.10909090909091, For Best Depth = 10, and Best sample = 5
Done
```



In [78]:
```
print ("Best set6_Bestmax_depth  = %s" %set6_Bestmax_depth)
print ("set6_Bestmin_samples_split = %s" % set6_Bestmin_samples_split)

set6_Bestmax_depth=2
set6_Bestmin_samples_split=50
Set6_trainAUC_analysis,Set6_testAUC_analysis = \
DecisionTree_for_Best_Hyper_Parameter(X_tr_set6,y_train,X_te_set6,y_test,set6_
Bestmax_depth,set6_Bestmin_samples_split)
```

```
Best set6_Bestmax_depth  = 100
set6_Bestmin_samples_split = 5
=============================================================================
=======================
the maximum value of tpr*(1-fpr) 0.33748699356021417 for threshold 0.554
Train confusion matrix
[[ 3095  2073]
 [12366 15966]]
Test confusion matrix
[[1383 1163]
 [6915 7039]]
```

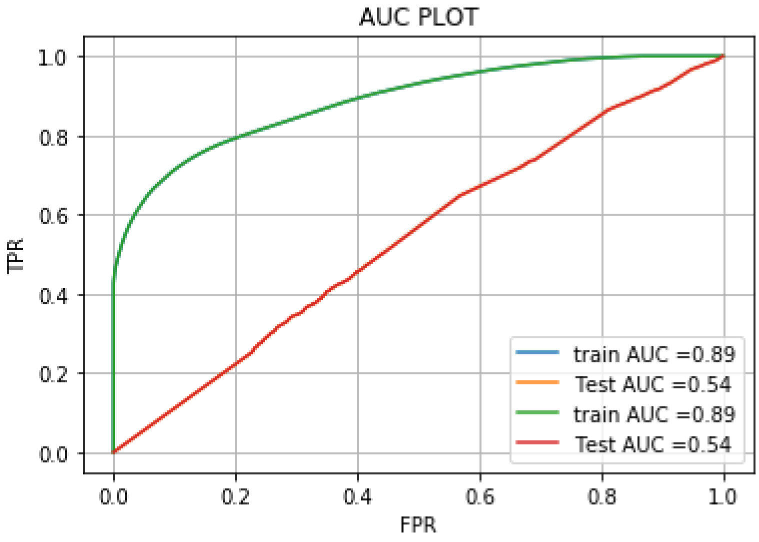


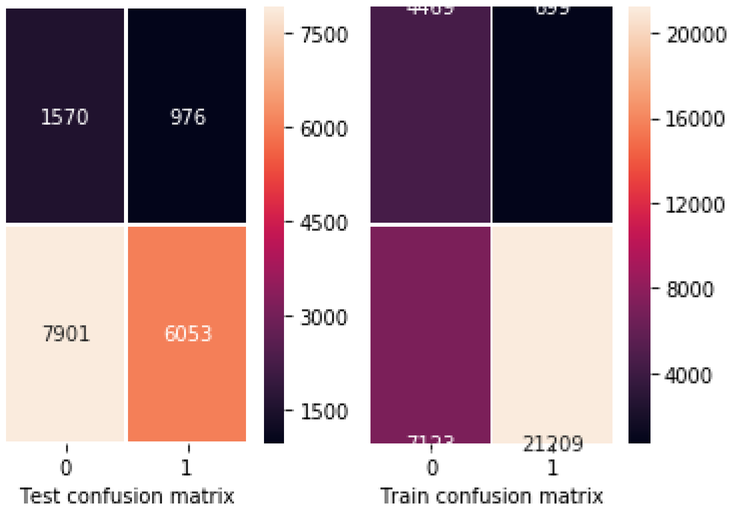


In [79]:

```
print ("Best bow_Bestmax_depth  = %s" %bow_Bestmax_depth)
print ("bow_Bestmin_samples_split = %s" % bow_Bestmin_samples_split)

trainAUC_bow_analysis, testAUC_bow_analysis = \
DecisionTree_for_Best_Hyper_Parameter(X_tr_bow,y_train,X_te_bow,y_test, bow_Be
stmax_depth,bow_Bestmin_samples_split)
```

```
Best bow_Bestmax_depth  = 300
bow_Bestmin_samples_split = 100
=============================================================================
=======================
the maximum value of tpr*(1-fpr) 0.6473375631991132 for threshold 0.464
Train confusion matrix
[[ 4469   699]
 [ 7123 21209]]
Test confusion matrix
[[1570  976]
 [7901 6053]]
```

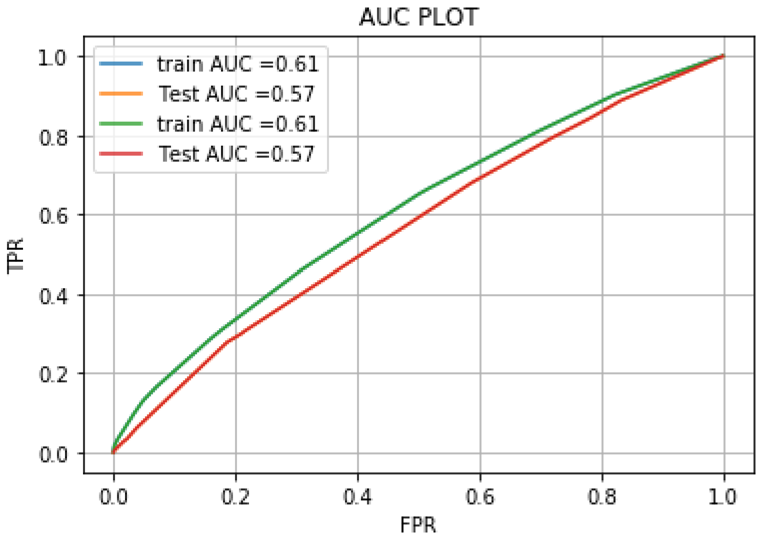


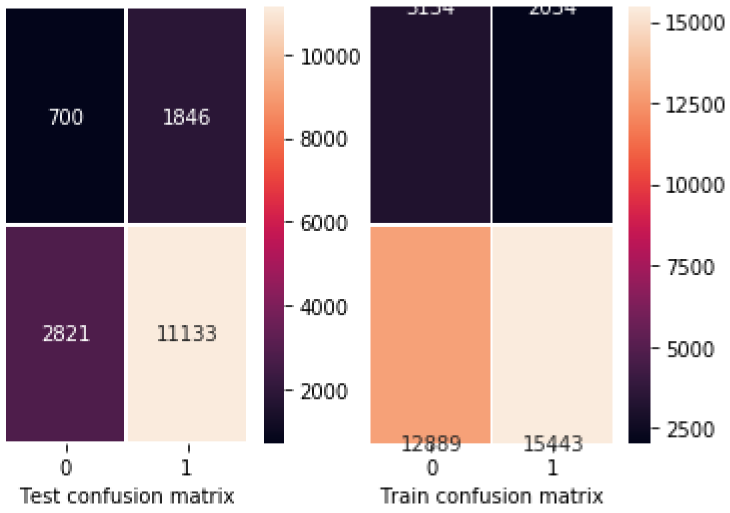


In [80]:

```
print ("Best set5_Bestmax_depth  = %s" %set5_Bestmax_depth)
print ("set5_Bestmin_samples_split = %s" % set5_Bestmin_samples_split)

set5_Bestmax_depth =5
set5_Bestmin_samples_split=50

trainAUC_avg_w2v_analysis, testAUC_avg_w2v_analysis = \
DecisionTree_for_Best_Hyper_Parameter(X_tr_set5,y_train,X_te_set5,y_test,  set
5_Bestmax_depth,set5_Bestmin_samples_split)
```

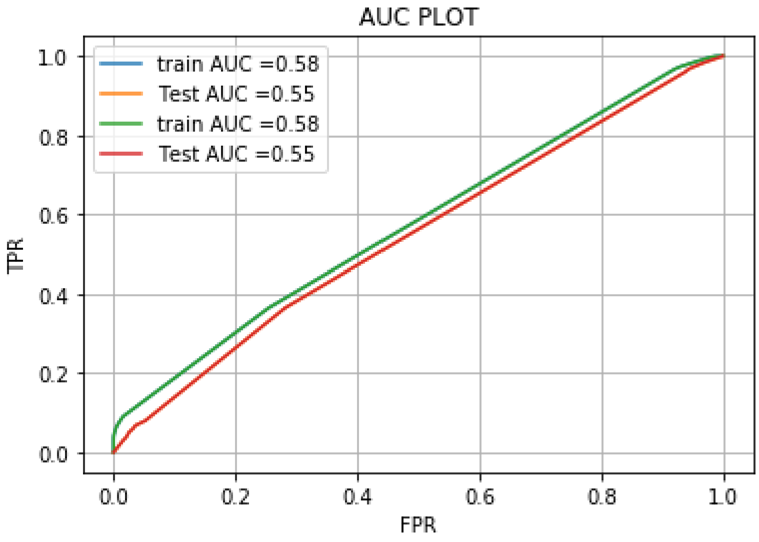
```
Best set5_Bestmax_depth  = 10
set5_Bestmin_samples_split = 5
=============================================================================
=======================
the maximum value of tpr*(1-fpr) 0.3305452536684662 for threshold 0.5
Train confusion matrix
[[ 3134  2034]
 [12889 15443]]
Test confusion matrix
[[  700  1846]
 [ 2821 11133]]
```

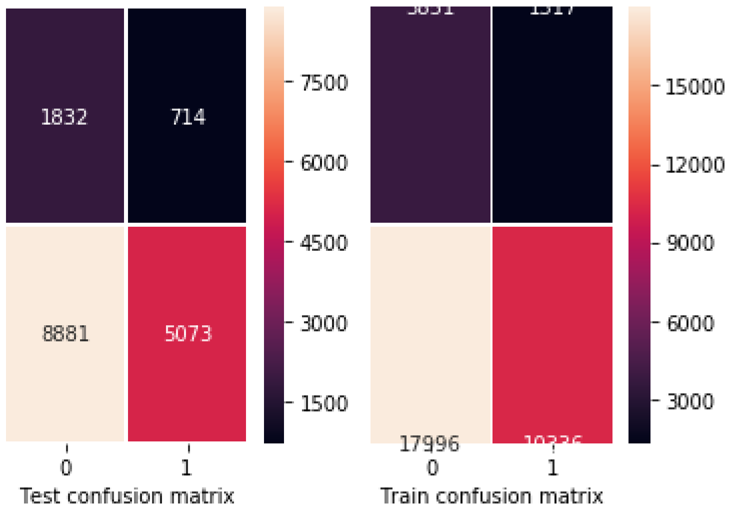




In [81]:
```
print ("Best tfidf_Bestmax_depth  = %s" %tfidf_Bestmax_depth)
print ("tfidf_Bestmin_samples_split = %s" % tfidf_Bestmin_samples_split)
tfidf_Bestmax_depth=10
tfidf_Bestmin_samples_split=3
trainAUC_tfidf_analysis, testAUC_tfidf_analysis = \
DecisionTree_for_Best_Hyper_Parameter(X_tr_tfidf,y_train,X_te_tfidf,y_test,tfi
df_Bestmax_depth,tfidf_Bestmin_samples_split)
```

```
Best tfidf_Bestmax_depth  = 300
tfidf_Bestmin_samples_split = 5
=============================================================================
=======================
the maximum value of tpr*(1-fpr) 0.27184808696879853 for threshold 0.477
Train confusion matrix
[[ 3851  1317]
 [17996 10336]]
Test confusion matrix
[[1832  714]
 [8881 5073]]
```

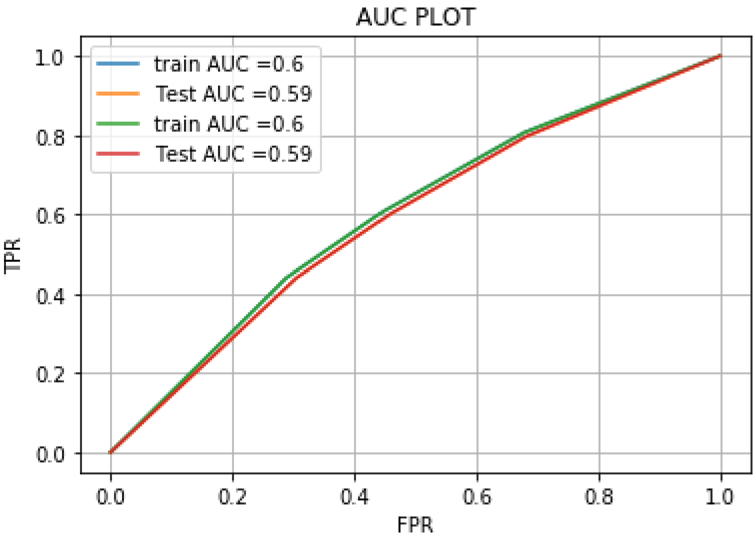


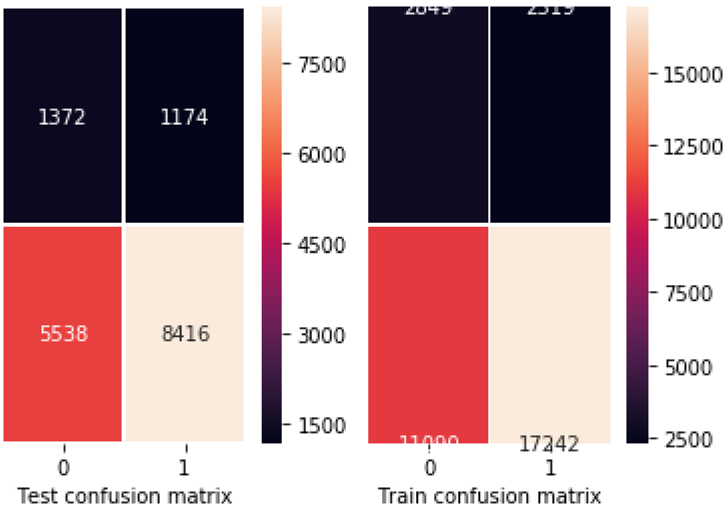


In [82]:

```
print ("Best tfidf_w2v_Bestmax_depth  = %s" %tfidf_w2v_Bestmax_depth)
print ("tfidf_w2v_Bestmin_samples_split = %s" % tfidf_w2v_Bestmin_samples_spli
t)
tfidf_w2v_Bestmax_depth = 2
tfidf_w2v_Bestmin_samples_split = 70
trainAUC_tfidf_w2v_analysis, testAUC_tfidf_w2v_analysis = \
DecisionTree_for_Best_Hyper_Parameter(X_tr_tfidf_w2v,y_train,X_te_tfidf_w2v,y_
test,tfidf_w2v_Bestmax_depth,tfidf_w2v_Bestmin_samples_split)
```

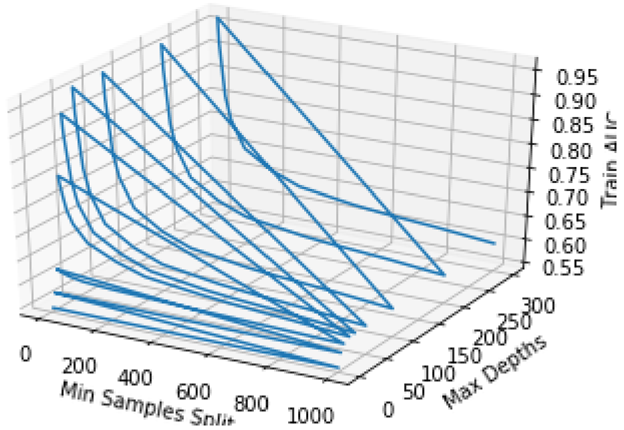
```
Best tfidf_w2v_Bestmax_depth  = 300
tfidf_w2v_Bestmin_samples_split = 5
==============================================================================
=======================
the maximum value of tpr*(1-fpr) 0.3354905965707801 for threshold 0.513
Train confusion matrix
[[ 2849  2319]
 [11090 17242]]
Test confusion matrix
[[1372 1174]
 [5538 8416]]
```





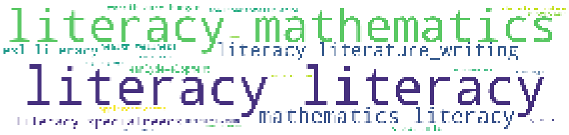
In [83]:
```
# Remove unwanted features from model and regenerate model using better featur
es to see the impact on accuracy.

Bestmax_depth,Bestmin_samples_split=DecisionTree_for_better_important_features
(X_tr_set5,y_train,X_te_set5,y_test,set5_Bestmax_depth,set5_Bestmin_samples_sp
lit)
```

```
Best Accuracy score = 40.81818181818182, For Best Depth = 1, and Best sample
= 5
Best Accuracy score = 70.34545454545454, For Best Depth = 5, and Best sample
= 5
Best Accuracy score = 70.47272727272727, For Best Depth = 5, and Best sample
= 200
Best Accuracy score = 70.60606060606061, For Best Depth = 5, and Best sample
= 300
Best Accuracy score = 72.79393939393938, For Best Depth = 30, and Best sample
= 5
Best Accuracy score = 73.28484848484848, For Best Depth = 50, and Best sample
= 5
Best Accuracy score = 73.4909090909091, For Best Depth = 100, and Best sample
= 5
Done
```



Positive False Points and WordCloud

In [84]:
```
text_Bestmax_depth=2
text_Bestmin_samples_split=50
X_tr_text =  X_train_sub_categories_one_hot
X_te_text =  X_test_sub_categories_one_hot
positive_false_points(X_tr_text,y_train,X_te_text,y_test,text_Bestmax_depth,te
xt_Bestmin_samples_split)
```



3.0 Pretty Table SET 3

In [85]:

```
#Draw Pretty Table using Best Analysis. Pretty table is drawn based on
#best AUC is calcukated by passing varipus Hyperparameter in loop.

from prettytable import PrettyTable

LRTable = PrettyTable()

LRTable.field_names = ["Model Type", "Train AUC", "Test AUC"]
#LRTable.add_row(["Regular_Analyis", Set6_trainAUC_analysis, Set6_trainAUC_ana
lysis])
LRTable.add_row(["BoW_Analysis", trainAUC_bow_analysis, testAUC_bow_analysis])
#LRTable.add_row(["TFIDF_Analysis", trainAUC_tfidf, testAUC_tfidf])
LRTable.add_row(["TFIDF_Avg_W2V_Analysis", trainAUC_avg_w2v_analysis, testAUC_
avg_w2v_analysis])
LRTable.add_row(["TFIDF_W2V_Analysis", trainAUC_tfidf_w2v_analysis, testAUC_tf
idf_w2v_analysis])
LRTable.add_row(["Set 6 Analysis", Set6_trainAUC_analysis, Set6_testAUC_analys
is])

print (LRTable)
```

```
+------------------------+-----------+----------+
|       Model Type       | Train AUC | Test AUC |
+------------------------+-----------+----------+
|      BoW_Analysis      |    0.89   |   0.54   |
| TFIDF_Avg_W2V_Analysis |    0.61   |   0.57   |
|   TFIDF_W2V_Analysis   |    0.6    |   0.59   |
|     Set 6 Analysis     |    0.59   |   0.5    |
+------------------------+-----------+----------+
```

In [86]:

```
#data = feature_importance()
```

3. Conclusions

DecisionTree is a stastical method for analyzing a dataset in which there are one or more independent variables that determine the outcome.

Summary of above program as below:

Lot of plots are drawn for different data set between train and test data. Test data is very near to train data. Please see pretty table for all comparasions.

Step 1: Import the necessary Libraries

we will need to import libraries that allow for data analysis and data visualization to get acclimated to the dataset. We will be using pandas, numpy, matplotlib and seaborn to conduct this. Data Exploration libraries

Step 2: Read in the dataset.

We will use the pandas .read_csv() method to read in the dataset. Then we will use the. head() method to observe the first few rows of the data, to understand the information better. In our case, the feature(column) headers tell us pretty little. This is fine because we are merely trying to gain insight via classifying new data points by referencing it’s neighboring elements.

Step 3: Standardize (normalize) the data scale to prep for DecisionTree.

Because the distance between pairs of points plays a critical part on the classification, it is necessary to normalize the data This will generate an array of values.

Step 4: Split the normalized data into training and test sets.

This step is required to prepare us for the fitting (i.e. training) the model later. The “X” variable is a collection of all the features. The “y” variable is the target label which specifies the classification of 1 or 0 based. Our goal will be to identify which category the new data point should fall into.

Step 5: Create and Train the Model.

Here we create a DecisionTree Object and use the .fit() method to train the model. Upon completion of the model we should receive confirmation that the training has been complete

Please see functions as covered below, used in above program: def DecisionTree_validation(X,y): def

Step 6: Make Predictions.

Here we review where our model was accurate and where it misclassified elements.

Please see functions as covered below, used in above program: def DecisionTree_validation(X,y):

Step 7: Evaluate the predictions.

Evaluate the Model by reviewing the classification report or confusion matrix. By reviewing these tables, we are able to evaluate how accurate our model is with new values.

def DecisionTree_validation(X,y):

Setp 8:Classification Report :

This tells us our model was around 84% accurate… Print out classification report and confusion matrix

I have covered various set to show confusion matrix.

Please see section 2. covered various data sets and created confusion matrix.

Step 9: Evaluate alternative Hyper Parameter for better predictions.

To simplify the process of evaluating multiple cases for max depth and min_samples_split values, we create a function to derive the error using the average where our predictions were not equal to the test values.

Please see section 2. covered various data sets and created error accuracy reports.

Step 10: Adjust Hyper Parameter value per error rate evaluations

This is just fine tuning our model to increase accuracy. We will need to retrain our model with new max depth and min_samples_split values. Please see section 3 in above program. we have created confusion matrix for optimal max depth and min_samples_split values value for various data sets. As we can see for optimal max depth and min_samples_split values, Accuracy is much higher - so prediction is much better.

Othe summary:

Pretty Table:

Prety table shows the overall peformance of few sets test vrs predcited. we can see prediected values are comparable to test values.

WordClud:

Word clouds (also known as text clouds or tag clouds) work in a simple way: the more a specific word appears in a source of textual data (such as a speech, blog post, or database), the bigger and bolder it appears in the word cloud.

A word cloud is a collection, or cluster, of words depicted in different sizes. The bigger and bolder the word appears, the more often it’s mentioned within a given text and the more important it is.

As you can see wordcoud is generated above ib the graph for most used words.

In [ ]: