SVD on Donors Choose dataset

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 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.

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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

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

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
```

```python
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
import chart_studio.plotly as py
from scipy.sparse import hstack
import chart_studio.plotly as py

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 collections import Counter
```

1. LOAD AND PROCESS DATA

1.1 Reading Data

In [2]:

```python
data=pd.read_csv("train_data.csv",nrows=50000)
resource_data=pd.read_csv("resources.csv")
data.columns
```

Out[2]:

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

In [3]:

```python
price_data=resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
```

In [4]:
```
project_data=pd.merge(data, price_data, on='id', how='left')
```

In [5]:
```
project_data.columns
```

Out[5]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'project_subject_categories', 'project_subject_subcategories',
       'project_title', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'price', 'quantity'],
      dtype='object')
```

1.2 process Project Essay

In [6]:
```
project_data.head(3)
```

Out[6]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	Histc
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Grades 6-8	

In [7]:
```
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 [8]:

```
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 [9]:

```
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'furthe
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'mor
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "were
            'won', "won't", 'wouldn', "wouldn't"]
```

In [10]:

```
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
```

```
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
project_data['cleaned_essay']=preprocessed_essays
```

```
100%|██████████| 50000/50000 [00:27<00:00, 1847.94it/s]
```

1.2 process Project Title

In [11]:

```
# https://stackoverflow.com/a/47091490/4084039
from tqdm import tqdm
preprocessed_title = []
# tqdm is for printing the status bar
for sentance in tqdm(data['project_title'].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)
    preprocessed_title.append(sent.lower().strip())
project_data['cleaned_project_title']=preprocessed_title
```

```
100%|██████████| 50000/50000 [00:01<00:00, 38276.23it/s]
```

1.3 teacher_prefix

In [12]:

```
temp1=data.teacher_prefix.apply(lambda x: str(x).replace('.', ''))
project_data['teacher_prefix']=temp1
project_data['teacher_prefix'].value_counts()
```

Out[12]:

```
Mrs        26140
Ms         17936
Mr          4859
Teacher     1061
nan            2
```

```
Dr              2
Name: teacher_prefix, dtype: int64
```

1.4 project grade

In [13]:
```
project_data.project_grade_category.value_counts()
```

Out[13]:
```
Grades PreK-2    20316
Grades 3-5       16968
Grades 6-8        7750
Grades 9-12       4966
Name: project_grade_category, dtype: int64
```

In [14]:
```
grade_list=[]
for i in project_data['project_grade_category'].values:
    i=i.replace(' ','_')
    i=i.replace('-','_')
    grade_list.append(i.strip())


project_data['project_grade_category']=grade_list
```

In [15]:
```
project_data['project_grade_category'].value_counts()
```

Out[15]:
```
Grades_PreK_2    20316
Grades_3_5       16968
Grades_6_8        7750
Grades_9_12       4966
Name: project_grade_category, dtype: int64
```

1.5 project_subject_categories

In [16]:
```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

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

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

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

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

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

1.6 project_subject_subcategories

In [17]:

```
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","&", '
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The',
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science'
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
```

```
        sub_cat_list.append(temp.strip())

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

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

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

1.7 counting words in title

In [18]:

```
#https://stackoverflow.com/questions/49984905/count-number-of-words-per-row
project_data['totalwords_title'] = project_data['cleaned_project_title'].str.split().str.len()
```

1.8 number of words in the essay

In [19]:

```
project_data['totalwords_essay'] = project_data['cleaned_essay'].str.split().str.len()
```

1.9 sentiment score's of each of the essay

In [20]:

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()
neg=[]
compound=[]
pos=[]
neu=[]
for sent in (project_data['cleaned_essay'].values):
    score = analyser.polarity_scores(sent)
    neg.append(score.get('neg'))
    neu.append(score.get('neu'))
    pos.append(score.get('pos'))
    compound.append(score.get('compound'))
project_data['neg']=neg
project_data['neu']=neu
```

```
project_data['pos']=pos
project_data['compound']=compound
```

1.10 droping unnecesarry columns

In [21]:
```
project_data.drop(['project_title'], axis=1, inplace=True)
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

1.11 Concate Essay and title

In [22]:
```
project_data['essay_title']=project_data['cleaned_essay']+" "+project_data['cleaned_project_title']
essay_title=project_data['essay_title']
```

1.12 TFIDF vectorizer of concatinated Essat_titile

In [23]:
```
#https://stackoverflow.com/questions/48431173/is-there-a-way-to-get-only-the-idf-values-of-words-using-scikit-or-any-

tf = TfidfVectorizer(use_idf=True,min_df=10)
tf.fit_transform(project_data['essay_title'].values)
feature = tf.get_feature_names()


idf = tf.idf_ #print numerical idf

indexes = np.argsort(idf)[::-1] #sorting as per value w.r.t indexes
indexes = indexes[0:2000]
idf_feature=[]
fet_2000=[]
for i in indexes:
    fet_2000.append(feature[i])
    idf_feature.append([feature[i],idf[i]])
```

In [24]:
```
len(fet_2000)
```

Out[24]:
```
2000
```

1.13 Co-occurance matrix

In [25]:
```
def get_co_occur_matrix(data, vocab, context_window=5):
    a = pd.DataFrame(np.zeros((len(vocab), len(vocab))), index=vocab, columns=vocab)
    for review in data:
        words = review.split()
        for idx in range(len(words)):
            if a.get(words[idx]) is None:
                continue
            for i in range(1, context_window+1):
                if idx-i >= 0:
                    if a.get(words[idx-i]) is not None:
                        a[words[idx-i]].loc[words[idx]] = a.get(words[idx-i]).loc[words[idx]] + 1
                        a[words[idx]].loc[words[idx-i]] = a.get(words[idx]).loc[words[idx-i]] + 1
                if idx+i < len(words):
                    if a.get(words[idx+i]) is not None:
                        a[words[idx+i]].loc[words[idx]] = a.get(words[idx+i]).loc[words[idx]] + 1
                        a[words[idx]].loc[words[idx+i]] = a.get(words[idx]).loc[words[idx+i]] + 1
    np.fill_diagonal(a.values, 0)
    return a

co_matrix = get_co_occur_matrix(project_data['essay_title'], fet_2000)
```

In [26]:
```
co_matrix
```

Out[26]:

	kelly	detention	devises	devotion	symphonic	dilemmas	susceptible	disappearing	ditching	sunday	...	keepers	medias	shovels
kelly	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
detention	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
devises	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
devotion	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
symphonic	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
heal	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
royal	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
rub	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0

	kelly	detention	devises	devotion	symphonic	dilemmas	susceptible	disappearing	ditching	sunday	...	keepers	medias	shovels
ordinarily	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0
moderately	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0

2000 rows × 2000 columns

1.14 SVD on cooccurance matrix

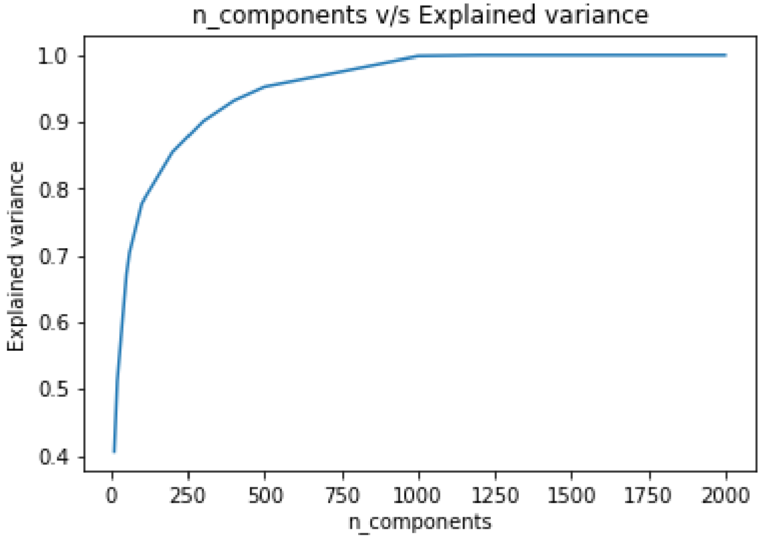
In [27]:

```
from sklearn.decomposition import TruncatedSVD
n_components=[10,20,50,60,100,200,300,400,500,1000,1200,1500,1600,1700,1800,1900,1999]
explained_variance=[]
for i in n_components:
 svd=TruncatedSVD(n_components=i,random_state=42)
 svd.fit(co_matrix)
 exp_var=svd.explained_variance_ratio_.sum()
 explained_variance.append(exp_var)

 print('n_components=',i,'variance=',exp_var)
```

```
n_components= 10 variance= 0.40607391483910527
n_components= 20 variance= 0.5129279118824488
n_components= 50 variance= 0.6722847588030308
n_components= 60 variance= 0.7047263478101818
n_components= 100 variance= 0.7781856441556061
n_components= 200 variance= 0.8549650520220569
n_components= 300 variance= 0.9014224986700434
n_components= 400 variance= 0.9324144628011418
n_components= 500 variance= 0.9531639495527733
n_components= 1000 variance= 0.9988219060277594
n_components= 1200 variance= 1.0000000000000084
n_components= 1500 variance= 1.0000000000000115
n_components= 1600 variance= 1.0000000000000095
n_components= 1700 variance= 1.0000000000000093
n_components= 1800 variance= 1.0000000000000142
n_components= 1900 variance= 1.0000000000000147
n_components= 1999 variance= 1.0000000000000093
```

1.15 components vs variance

In [28]:
```
plt.plot(n_components, explained_variance)
plt.xlabel('n_components')
plt.ylabel("Explained variance")
plt.title("n_components v/s Explained variance")
plt.show()
```



In [29]:
```
from sklearn.decomposition import TruncatedSVD
tsvd=TruncatedSVD(n_components=1500,random_state=42)
final_co_matrix=tsvd.fit_transform(co_matrix)
```

In [30]:
```
print(final_co_matrix.shape)
```

```
(2000, 1500)
```

In [31]:
```
co_matrix.columns
```

Out[31]:
```
Index(['kelly', 'detention', 'devises', 'devotion', 'symphonic', 'dilemmas',
       'susceptible', 'disappearing', 'ditching', 'sunday',
       ...
       'keepers', 'medias', 'shovels', 'footrest', 'zoned', 'heal', 'royal',
       'rub', 'ordinarily', 'moderately'],
      dtype='object', length=2000)
```

In [32]:
```
word_names = list(co_matrix.columns)
```

1.16 Making dependant(label) and independant variables

In [33]:
```
project_data=project_data[:2000]

y = project_data['project_is_approved'].values



x=project_data
x.head(3)
```

Out[33]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	project_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs	IN	2016-12-05 13:43:57	Grades_PreK_2	op
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr	FL	2016-10-25 09:22:10	Grades_6_8	My stud
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms	AZ	2016-08-31 12:03:56	Grades_6_8	My gua

3 rows × 24 columns

1.17 Traing and Test split

In [34]:
```
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, stratify=y,random_state=42)
```

```
#X_train, X_cv, Y_train, Y_cv = train_test_split(X_train,Y_train, test_size=0.33, stratify=Y_train,random_state=42)
```

2.Text Vectorization using co_matrix and encoding catagories,normalization numerical features¶

2.1 essay and title with matix words

In [35]:

```
Text_avg_w2v_train_essay_co_matrix= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['cleaned_essay'].values): # for each review/sentence
    vector = np.zeros(1500) # 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 word_names:
            vector += final_co_matrix[word_names.index(word)]
    Text_avg_w2v_train_essay_co_matrix.append(vector)

print(len(Text_avg_w2v_train_essay_co_matrix))
print(len(Text_avg_w2v_train_essay_co_matrix[0]))
```

```
100%|██████████| 1340/1340 [00:05<00:00, 244.72it/s]
1340
1500
```

In [36]:

```
Text_avg_w2v_test_essay_co_matrix= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['cleaned_essay'].values): # for each review/sentence
    vector = np.zeros(1500) # 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 word_names:
            vector += final_co_matrix[word_names.index(word)]
    Text_avg_w2v_test_essay_co_matrix.append(vector)

print(len(Text_avg_w2v_test_essay_co_matrix))
print(len(Text_avg_w2v_test_essay_co_matrix[0]))
```

```
100%|██████████| 660/660 [00:02<00:00, 243.85it/s]
660
1500
```

In [37]:
```python
Text_avg_w2v_train_title_co_matrix= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['cleaned_project_title'].values): # for each review/sentence
    vector = np.zeros(1500) # 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 word_names:
            vector += final_co_matrix[word_names.index(word)]

    Text_avg_w2v_train_title_co_matrix.append(vector)

print(len(Text_avg_w2v_train_title_co_matrix))
print(len(Text_avg_w2v_train_title_co_matrix[0]))
```

```
100%|██████████| 1340/1340 [00:00<00:00, 8070.28it/s]
1340
1500
```

In [38]:
```python
Text_avg_w2v_test_title_co_matrix= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['cleaned_project_title'].values): # for each review/sentence
    vector = np.zeros(1500) # 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 word_names:
            vector += final_co_matrix[word_names.index(word)]

    Text_avg_w2v_test_title_co_matrix.append(vector)

print(len(Text_avg_w2v_test_title_co_matrix))
print(len(Text_avg_w2v_test_title_co_matrix[0]))
```

```
100%|██████████| 660/660 [00:00<00:00, 8047.09it/s]
660
1500
```

2.2 Categories with response coding

In [39]:
```python
def Responsetable(table, col) :
    cat = table[col].unique()
    alpha=1
    freq_Pos = []
    for i in cat :
```

```
        freq_Pos.append(len(table.loc[(table[col] == i) & (table['project_is_approved'] == 1)]))


    freq_Neg = []
    for i in cat :
        freq_Neg.append(len(table.loc[(table[col] == i) & (table['project_is_approved'] == 0)]))


    encoded_Pos = []
    for i in range(len(cat)) :
        encoded_Pos.append(((freq_Pos[i]+alpha)/(freq_Pos[i] + freq_Neg[i]+alpha)))

    encoded_Neg = []
    encoded_Neg[:] = [1 - x for x in encoded_Pos]

    encoded_Pos_val = dict(zip(cat, encoded_Pos))
    encoded_Neg_val = dict(zip(cat, encoded_Neg))

    return encoded_Pos_val, encoded_Neg_val
```

In [40]:

```
def Responsecode(table) :
    pos_cleancat, neg_cleancat = Responsetable(table,'clean_categories')
    pos_cleansubcat, neg_cleansubcat = Responsetable(table,'clean_subcategories')
    pos_schoolstate, neg_schoolstate = Responsetable(table, 'school_state')
    pos_teacherprefix, neg_teacherprefix = Responsetable(table, 'teacher_prefix')
    pos_projgradecat, neg_projgradecat = Responsetable(table, 'project_grade_category')

    df = pd.DataFrame()
    df['clean_cat_pos'] = table['clean_categories'].map(pos_cleancat)
    df['clean_cat_neg'] = table['clean_categories'].map(neg_cleancat)
    df['clean_subcat_pos'] = table['clean_subcategories'].map(pos_cleansubcat)
    df['clean_subcat_neg'] = table['clean_subcategories'].map(neg_cleansubcat)
    df['school_state_pos'] = table['school_state'].map(pos_schoolstate)
    df['school_state_neg'] = table['school_state'].map(neg_schoolstate)
    df['teacher_prefix_pos'] = table['teacher_prefix'].map(pos_teacherprefix)
    df['teacher_prefix_neg'] = table['teacher_prefix'].map(neg_teacherprefix)
    df['proj_grade_cat_pos'] = table['project_grade_category'].map(pos_projgradecat)
    df['proj_grade_cat_neg'] = table['project_grade_category'].map(neg_projgradecat)

    return df
```

```
In [41]: newTrain = Responsecode(X_train)
newTest = Responsecode(X_test)
#newCv=Responsecode(X_cv)
```

```
In [42]: def mergeEncoding(table, p, n) :
    lstPos = table[p].values.tolist()
    lstNeg = table[n].values.tolist()
    frame = pd.DataFrame(list(zip(lstNeg, lstPos)))

    return frame
```

2.3 response code of clean_categories

```
In [43]: X_train_clean_cat_resposecode = mergeEncoding(newTrain, 'clean_cat_pos', 'clean_cat_neg')
X_test_clean_cat_resposecode = mergeEncoding(newTest, 'clean_cat_pos', 'clean_cat_neg')
#X_cv_clean_cat_resposecode=mergeEncoding(newCv, 'clean_cat_pos', 'clean_cat_neg')
print(X_train_clean_cat_resposecode.shape)
```

```
(1340, 2)
```

2.4 response code of clean_sub_categories

```
In [44]: X_train_clean_subcat_resposecode = mergeEncoding(newTrain, 'clean_subcat_pos', 'clean_subcat_neg')
X_test_clean_subcat_resposecode = mergeEncoding(newTest, 'clean_subcat_pos', 'clean_subcat_neg')
#X_cv_clean_subcat_resposecode = mergeEncoding(newCv, 'clean_subcat_pos', 'clean_subcat_neg')
print(X_train_clean_subcat_resposecode.shape)
print(X_test_clean_subcat_resposecode.shape)
#print(X_cv_clean_subcat_resposecode.shape)
```

```
(1340, 2)
(660, 2)
```

2.5 response code of project grade

```
In [45]: X_train_grade_resposecode = mergeEncoding(newTrain, 'proj_grade_cat_pos', 'proj_grade_cat_neg')
X_test_grade_resposecode = mergeEncoding(newTest, 'proj_grade_cat_pos', 'proj_grade_cat_neg')
#X_cv_grade_resposecode = mergeEncoding(newCv, 'proj_grade_cat_pos', 'proj_grade_cat_neg')
print(X_train_grade_resposecode.shape)
print(X_test_grade_resposecode.shape)
#print(X_cv_grade_resposecode.shape)
```

```
(1340, 2)
(660, 2)
```

2.6 response code of school state

In [46]:
```
X_train_state_resposecode = mergeEncoding(newTrain, 'school_state_pos', 'school_state_neg')
X_test_state_resposecode = mergeEncoding(newTest, 'school_state_pos', 'school_state_neg')
#X_cv_state_resposecode = mergeEncoding(newCv, 'school_state_pos', 'school_state_neg')
print(X_train_state_resposecode.shape)
print(X_test_state_resposecode.shape)
#print(X_cv_state_resposecode.shape)
```

```
(1340, 2)
(660, 2)
```

2.7 response code of teacher prefix

In [47]:
```
X_train_teacher_resposecode = mergeEncoding(newTrain, 'teacher_prefix_pos', 'teacher_prefix_neg')
X_test_teacher_resposecode = mergeEncoding(newTest, 'teacher_prefix_pos', 'teacher_prefix_neg')
#X_cv_teacher_resposecode = mergeEncoding(newCv, 'teacher_prefix_pos', 'teacher_prefix_neg')
print(X_train_teacher_resposecode.shape)
print(X_test_teacher_resposecode.shape)
#print(X_cv_teacher_resposecode.shape)
```

```
(1340, 2)
(660, 2)
```

2.8 Normalizing the numerical features: Price

In [48]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))

print("After vectorizations")
```

```
print(X_train_price_norm.shape, Y_train.shape)
#print(X_cv_price_norm.shape, Y_cv.shape)
print(X_test_price_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(1340, 1) (1340,)
(660, 1) (660,)
====================================================================================================
```

2.9 Normalizing the numerical features:teacher_number_of_previously_posted_projects

In [49]:

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

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

X_train_TPPP_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)
#X_cv_TPPP_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_test_TPPP_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_TPPP_norm.shape, Y_train.shape)
#print(X_cv_TPPP_norm.shape, Y_cv.shape)
print(X_test_TPPP_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(1340, 1) (1340,)
(660, 1) (660,)
====================================================================================================
```

2.10 Normalizing the numerical features: quantity

In [50]:

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

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

X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.reshape(-1,1))
#X_cv_quantity_norm = normalizer.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_quantity_norm.shape, Y_train.shape)
#print(X_cv_quantity_norm.shape, Y_cv.shape)
print(X_test_quantity_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(1340, 1) (1340,)
(660, 1) (660,)
====================================================================================================
```

2.11 Normalizing the numerical features: totalwords_title

In [51]:

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

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

X_train_totalwords_title_norm = normalizer.transform(X_train['totalwords_title'].values.reshape(-1,1))

X_test_totalwords_title_norm = normalizer.transform(X_test['totalwords_title'].values.reshape(-1,1))
#X_cv_totalwords_title_norm = normalizer.transform(X_cv['totalwords_title'].values.reshape(-1,1))
print("After vectorizations")
print(X_train_totalwords_title_norm.shape, Y_train.shape)
#print(X_cv_totalwords_title_norm.shape, Y_cv.shape)
print(X_test_totalwords_title_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(1340, 1) (1340,)
(660, 1) (660,)
====================================================================================================
```

2.12 Normalizing the numerical features: totalwords_essay

In [52]:

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

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

X_train_totalwords_essay_norm = normalizer.transform(X_train['totalwords_essay'].values.reshape(-1,1))
```

```
#X_cv_totalwords_essay_norm = normalizer.transform(X_cv['totalwords_essay'].values.reshape(-1,1))
X_test_totalwords_essay_norm = normalizer.transform(X_test['totalwords_essay'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_totalwords_essay_norm.shape, Y_train.shape)
#print(X_cv_totalwords_essay_norm.shape, Y_cv.shape)
print(X_test_totalwords_essay_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(1340, 1) (1340,)
(660, 1) (660,)
====================================================================================================
```

2.13 Finally merge all data

In [53]:

```
X_tr_svd = np.hstack((Text_avg_w2v_train_essay_co_matrix,Text_avg_w2v_train_title_co_matrix,X_train_clean_cat_respose

X_te_svd = np.hstack((Text_avg_w2v_test_essay_co_matrix,Text_avg_w2v_test_title_co_matrix,X_test_clean_cat_resposecod

print("Final Data matrix")
print(X_tr_svd.shape, Y_train.shape)

print(X_te_svd.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(1340, 3015) (1340,)
(660, 3015) (660,)
====================================================================================================
```

3.XGB classifier

3.1 model on data

In [63]:

```
import matplotlib.pyplot as plt
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV

xgb = XGBClassifier(n_jobs=-1,class_weight='balanced')
```

```
parameters = {'n_estimators':[10, 50, 100, 150, 200, 300,500], 'max_depth': [2,4,5,6,7,8]}
clf=GridSearchCV(xgb, parameters, cv=3, scoring='roc_auc', n_jobs=-1, return_train_score=True)
clf.fit(X_tr_svd, Y_train)
```

```
[10:33:58] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:573:
Parameters: { "class_weight" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[10:33:58] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGB
oost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[63]:

```
GridSearchCV(cv=3,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     class_weight='balanced',
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estimators=100, n_jobs=-1,
                                     num_parallel_tree=None, random_state=None,
                                     reg_alpha=None, reg_lambda=None,
                                     scale_pos_weight=None, subsample=None,
                                     tree_method=None, validate_parameters=None,
                                     verbosity=None),
             n_jobs=-1,
             param_grid={'max_depth': [2, 4, 5, 6, 7, 8],
                         'n_estimators': [10, 50, 100, 150, 200, 300, 500]},
             return_train_score=True, scoring='roc_auc')
```

In [65]:

```
clf.cv_results_
```

Out[65]:

```
{'mean_fit_time': array([ 0.84325306,  3.02359247,  5.665277  ,  8.22885354, 10.42034737,
        14.80200267, 23.11987631,  0.95354851,  3.841199  ,  7.91511599,
        11.80232588, 14.9517018 , 21.4161582 , 35.75172305,  1.06757315,
         4.40999468,  8.93234603, 13.67374754, 17.55428799, 25.50741561,
        42.34387604,  1.23961194,  5.04847097, 10.15662249, 15.46615171,
```

```
        20.23355977, 29.56699586, 49.59851154,  1.44832579,  5.83698098,
        11.57294035, 17.64164209, 23.06553229, 34.0073297 , 56.35469818,
         1.53401224,  6.64082932, 13.64007386, 19.74411511, 25.31403859,
        37.08502388, 44.61071992]),
 'std_fit_time': array([0.007365  , 0.05100758, 0.12456194, 0.06226118, 0.21110419,
        0.10336825, 0.27814414, 0.01857483, 0.13771795, 0.27564266,
        0.13030408, 0.15060293, 0.18438852, 0.4508373 , 0.01866459,
        0.12633289, 0.13415784, 0.10449839, 0.09160366, 0.16458835,
        0.40405974, 0.06410441, 0.18200014, 0.23409357, 0.11722173,
        0.09242222, 0.0997034 , 0.38287007, 0.09463592, 0.18670828,
        0.33253682, 0.16650962, 0.08835375, 0.04004514, 0.52428378,
        0.02910754, 0.13885997, 0.15008145, 0.14719948, 0.12147606,
        0.2303999 , 6.04663362]),
 'mean_score_time': array([0.04034265, 0.03067374, 0.03034051, 0.02400621, 0.02767436,
        0.03067454, 0.02467219, 0.02100579, 0.0226721 , 0.02233903,
        0.02400549, 0.02233887, 0.0230062 , 0.02467211, 0.01800474,
        0.01767023, 0.02033981, 0.02100507, 0.02200484, 0.02200691,
        0.02467322, 0.02433928, 0.02167273, 0.02233783, 0.0183386 ,
        0.02300557, 0.02667387, 0.02467171, 0.01867247, 0.0203383 ,
        0.02133902, 0.02200691, 0.02300445, 0.02367202, 0.02367242,
        0.0236721 , 0.02333919, 0.02867277, 0.02167225, 0.02133743,
        0.02533952, 0.02133814]),
 'std_score_time': array([0.00826123, 0.00665312, 0.00170225, 0.00216142, 0.0033993 ,
        0.00449924, 0.00377136, 0.00245194, 0.00124711, 0.00684964,
        0.00294506, 0.00249502, 0.00355951, 0.00124664, 0.00215969,
        0.00124664, 0.00368229, 0.00216042, 0.00215998, 0.00294268,
        0.00124719, 0.00047148, 0.00205477, 0.00205598, 0.0024943 ,
        0.00216046, 0.00449668, 0.00309182, 0.00464294, 0.00330077,
        0.00169991, 0.00454526, 0.00326585, 0.00492251, 0.00249315,
        0.00286848, 0.00419059, 0.00262522, 0.00169922, 0.00249506,
        0.00402826, 0.00634275]),
 'param_max_depth': masked_array(data=[2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5,
                    5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 8,
                    8, 8, 8, 8, 8, 8],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[10, 50, 100, 150, 200, 300, 500, 10, 50, 100, 150, 200,
                    300, 500, 10, 50, 100, 150, 200, 300, 500, 10, 50, 100,
                    150, 200, 300, 500, 10, 50, 100, 150, 200, 300, 500,
```

```
                     10, 50, 100, 150, 200, 300, 500],
               mask=[False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False, False, False, False, False, False, False,
                     False, False],
         fill_value='?',
              dtype=object),
 'params': [{'max_depth': 2, 'n_estimators': 10},
  {'max_depth': 2, 'n_estimators': 50},
  {'max_depth': 2, 'n_estimators': 100},
  {'max_depth': 2, 'n_estimators': 150},
  {'max_depth': 2, 'n_estimators': 200},
  {'max_depth': 2, 'n_estimators': 300},
  {'max_depth': 2, 'n_estimators': 500},
  {'max_depth': 4, 'n_estimators': 10},
  {'max_depth': 4, 'n_estimators': 50},
  {'max_depth': 4, 'n_estimators': 100},
  {'max_depth': 4, 'n_estimators': 150},
  {'max_depth': 4, 'n_estimators': 200},
  {'max_depth': 4, 'n_estimators': 300},
  {'max_depth': 4, 'n_estimators': 500},
  {'max_depth': 5, 'n_estimators': 10},
  {'max_depth': 5, 'n_estimators': 50},
  {'max_depth': 5, 'n_estimators': 100},
  {'max_depth': 5, 'n_estimators': 150},
  {'max_depth': 5, 'n_estimators': 200},
  {'max_depth': 5, 'n_estimators': 300},
  {'max_depth': 5, 'n_estimators': 500},
  {'max_depth': 6, 'n_estimators': 10},
  {'max_depth': 6, 'n_estimators': 50},
  {'max_depth': 6, 'n_estimators': 100},
  {'max_depth': 6, 'n_estimators': 150},
  {'max_depth': 6, 'n_estimators': 200},
  {'max_depth': 6, 'n_estimators': 300},
  {'max_depth': 6, 'n_estimators': 500},
  {'max_depth': 7, 'n_estimators': 10},
  {'max_depth': 7, 'n_estimators': 50},
  {'max_depth': 7, 'n_estimators': 100},
  {'max_depth': 7, 'n_estimators': 150},
  {'max_depth': 7, 'n_estimators': 200},
  {'max_depth': 7, 'n_estimators': 300},
  {'max_depth': 7, 'n_estimators': 500},
  {'max_depth': 8, 'n_estimators': 10},
```

```
 {'max_depth': 8, 'n_estimators': 50},
 {'max_depth': 8, 'n_estimators': 100},
 {'max_depth': 8, 'n_estimators': 150},
 {'max_depth': 8, 'n_estimators': 200},
 {'max_depth': 8, 'n_estimators': 300},
 {'max_depth': 8, 'n_estimators': 500}],
'split0_test_score': array([0.72979183, 0.73532993, 0.72064022, 0.71141005, 0.70141398,
       0.68857031, 0.67486253, 0.7141791 , 0.70720738, 0.69497251,
       0.67582482, 0.66490573, 0.65343676, 0.64605263, 0.70653967,
       0.68886489, 0.68690102, 0.67409662, 0.6634132 , 0.64758445,
       0.62782797, 0.71576984, 0.69954831, 0.66667321, 0.66258837,
       0.65842498, 0.63976826, 0.62912412, 0.72164179, 0.70445797,
       0.67016889, 0.66003535, 0.64817361, 0.63552632, 0.62623723,
       0.72252553, 0.68258052, 0.66926551, 0.65869992, 0.65068735,
       0.63709741, 0.62802435]),
'split1_test_score': array([0.73440168, 0.728368  , 0.72165528, 0.69895623, 0.68450256,
       0.66793419, 0.64799007, 0.73740882, 0.70031429, 0.6595142 ,
       0.65606084, 0.64263542, 0.64961974, 0.65113301, 0.72646671,
       0.69942185, 0.66572249, 0.65959181, 0.65349992, 0.64942573,
       0.65171504, 0.73151094, 0.67336644, 0.65842775, 0.64989135,
       0.64899891, 0.64158777, 0.63883284, 0.73996973, 0.66188111,
       0.66762378, 0.6658777 , 0.65990222, 0.66190051, 0.66071706,
       0.73030809, 0.69544467, 0.69629831, 0.68717989, 0.68617104,
       0.68027316, 0.67332764]),
'split2_test_score': array([0.66715236, 0.65866577, 0.65437325, 0.64458709, 0.63411176,
       0.62452251, 0.62507384, 0.66549837, 0.66754617, 0.65051392,
       0.63773481, 0.63462372, 0.63501752, 0.62861812, 0.68384988,
       0.64486276, 0.63462372, 0.63233962, 0.62387272, 0.61713858,
       0.61223566, 0.69054464, 0.67172055, 0.64214547, 0.63907376,
       0.63678967, 0.62769267, 0.61851691, 0.66878667, 0.65439294,
       0.63938881, 0.62237625, 0.61997401, 0.61406687, 0.61316111,
       0.68004962, 0.65590911, 0.62332139, 0.61477573, 0.60971528,
       0.6086323 , 0.60835663]),
'mean_test_score': array([0.71044862, 0.70745457, 0.69888959, 0.68498446, 0.67334277,
       0.66034233, 0.64930881, 0.70569543, 0.69168928, 0.66833354,
       0.65654016, 0.64738829, 0.64602468, 0.64193459, 0.70561875,
       0.6777165 , 0.66241574, 0.65534268, 0.64692861, 0.63804959,
       0.63059289, 0.71260847, 0.6815451 , 0.65574881, 0.65051783,
       0.64807119, 0.63634957, 0.62882462, 0.71013273, 0.67357734,
       0.65906049, 0.64942976, 0.64268328, 0.63716457, 0.6333718 ,
       0.71096108, 0.6779781 , 0.66296174, 0.65355184, 0.64885789,
       0.64200096, 0.63656954]),
'std_test_score': array([0.03067287, 0.03461577, 0.03148053, 0.02901419, 0.02858674,
       0.02669279, 0.02034753, 0.02996395, 0.01730214, 0.01919161,
       0.01555388, 0.01281126, 0.00793769, 0.00964188, 0.01741043,
```

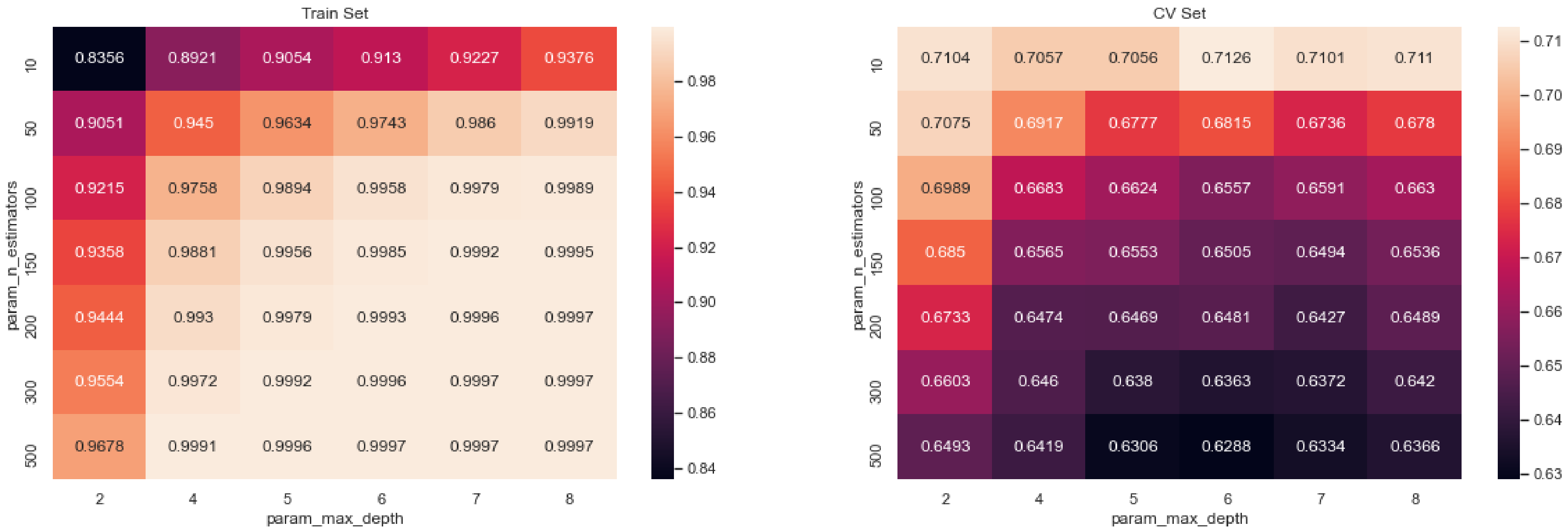
```
       0.02362751, 0.02146983, 0.01730998, 0.0167978 , 0.01480541,
       0.01623554, 0.01687316, 0.01274791, 0.01019101, 0.00961002,
       0.00885691, 0.00616625, 0.00829665, 0.03017837, 0.02204886,
       0.01394873, 0.01927784, 0.01675656, 0.01956233, 0.02005939,
       0.02208741, 0.01646515, 0.03012431, 0.02978218, 0.03123973,
       0.02945207, 0.02720384]),
'rank_test_score': array([ 3,  5,  8, 10, 15, 19, 27,  6,  9, 16, 21, 30, 32, 35,  7, 13, 18,
       23, 31, 36, 41,  1, 11, 22, 25, 29, 39, 42,  4, 14, 20, 26, 33, 37,
       40,  2, 12, 17, 24, 28, 34, 38]),
'split0_train_score': array([0.82838855, 0.90963061, 0.92670771, 0.9394508 , 0.9513779 ,
       0.96264536, 0.97330206, 0.89739568, 0.95195935, 0.97936578,
       0.99041825, 0.99458126, 0.99776703, 0.99942832, 0.90592202,
       0.97230529, 0.99138571, 0.99632073, 0.99862699, 0.9995065 ,
       0.99986807, 0.91762435, 0.9770302 , 0.9968582 , 0.99890062,
       0.99946741, 0.99981921, 0.99987785, 0.93197498, 0.98384149,
       0.9985537 , 0.99946252, 0.99976546, 0.99987296, 0.99987296,
       0.94231408, 0.99211864, 0.99921333, 0.99969217, 0.99986807,
       0.99987785, 0.99987785]),
'split1_train_score': array([0.83571274, 0.89188937, 0.90575777, 0.9199064 , 0.92760506,
       0.9402346 , 0.95690028, 0.87481565, 0.92944369, 0.96821721,
       0.98581696, 0.99105756, 0.99590486, 0.99847108, 0.89708572,
       0.94885258, 0.98500088, 0.99323049, 0.99647513, 0.9985989 ,
       0.99938548, 0.89649578, 0.96749946, 0.99353529, 0.99765501,
       0.99887421, 0.99937565, 0.99947397, 0.91106228, 0.98245433,
       0.99636698, 0.99874639, 0.99925766, 0.99943464, 0.99944448,
       0.93263918, 0.98894362, 0.99806796, 0.99921834, 0.99940515,
       0.99943464, 0.99944448]),
'split2_train_score': array([0.84274142, 0.91371688, 0.93215244, 0.94818719, 0.95425267,
       0.96329966, 0.97306885, 0.90406968, 0.95367687, 0.97970527,
       0.9880203 , 0.99324159, 0.99796516, 0.99926316, 0.91323867,
       0.96909676, 0.99169473, 0.99737959, 0.99864832, 0.99949739,
       0.99968282, 0.92475967, 0.97844142, 0.99690138, 0.99901918,
       0.99952667, 0.99969258, 0.99972186, 0.92507686, 0.99168009,
       0.99881423, 0.99948763, 0.99963402, 0.99964378, 0.99965354,
       0.9377202 , 0.99471039, 0.999351  , 0.99968282, 0.99970234,
       0.99973162, 0.99975113]),
'mean_train_score': array([0.83561424, 0.90507895, 0.92153931, 0.93584813, 0.94441187,
       0.95539321, 0.96775707, 0.89209367, 0.94502663, 0.97576275,
       0.98808517, 0.99296014, 0.99721235, 0.99905419, 0.90541547,
       0.96341821, 0.98936044, 0.9956436 , 0.99791681, 0.99920093,
       0.99964546, 0.91295994, 0.97432369, 0.99576496, 0.99852493,
       0.99928943, 0.99962915, 0.99969123, 0.92270471, 0.98599197,
       0.99791164, 0.99923218, 0.99955238, 0.99965046, 0.99965699,
       0.93755782, 0.99192422, 0.99887743, 0.99953111, 0.99965852,
       0.99968137, 0.99969115]),
```

```
'std_train_score': array([0.00585995, 0.00947447, 0.01137846, 0.01182329, 0.01194202,
       0.01072208, 0.0076775 , 0.01251754, 0.01104109, 0.0053373 ,
       0.00187903, 0.00145225, 0.00092806, 0.00041779, 0.00660413,
       0.01038242, 0.00308525, 0.00176024, 0.00101946, 0.00042571,
       0.00019878, 0.01200082, 0.00485973, 0.00157671, 0.00061703,
       0.0002946 , 0.00018656, 0.0001663 , 0.00870079, 0.00406178,
       0.0010974 , 0.00034366, 0.0002152 , 0.000179  , 0.00017494,
       0.00395143, 0.00235828, 0.00057513, 0.0002212 , 0.00019151,
       0.00018439, 0.00018194])}
```

3.2 heatmap

In [67]:

```python
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max().unstack()[['mear
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [68]:
```python
# Print params
print(clf.best_estimator_)
print(clf.score(X_tr_svd, Y_train))
print(clf.score(X_te_svd, Y_test))
```

```
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=10, n_jobs=-1,
              num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
0.9006855870121283
0.841984911503628
```

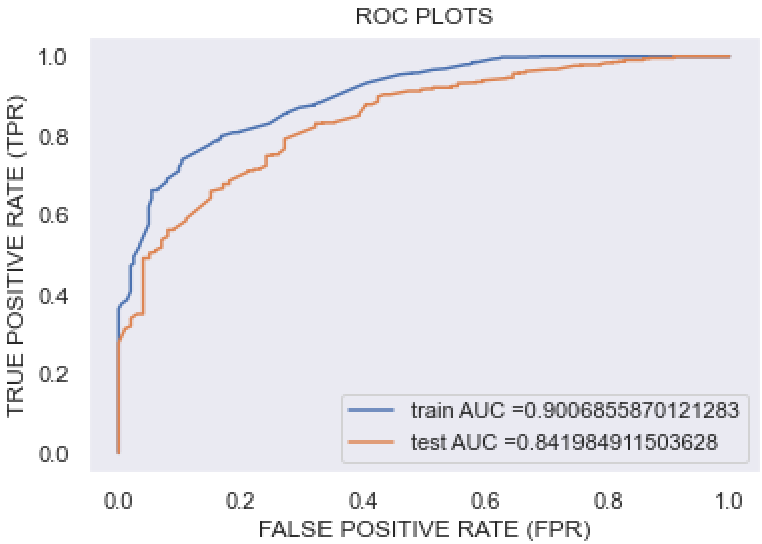
In [75]:
```python
def batch_predict(clf, data):


    data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000

    for i in range(0, tr_loop, 1000):
        data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])

    data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return data_pred
```

3.3 ROC curve

In [89]:
```python
from sklearn.metrics import roc_curve, auc
import xgboost as xgb

gbdt = xgb.XGBClassifier(max_depth = 6, n_estimators = 10,n_jobs=-1,class_weight='balanced')
gbdt.fit(X_tr_svd, Y_train)

y_train_pred = clf.predict_proba(X_tr_svd)[:,1]
y_test_pred = clf.predict_proba(X_te_svd)[:,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 (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```

```
[11:19:24] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:573:
Parameters: { "class_weight" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[11:19:24] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGB
oost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logl
oss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```



observation

1.Model perform good on train data than test data

In [85]:
```
def predict(proba, 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))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [60]:
```
def myplot_matrix1(data):
    plt.clf()
    plt.imshow(data, interpolation='nearest', cmap=plt.cm.Wistia)
    classNames = ['Negative','Positive']
    plt.title('Approved not approved matrix')
    tick_marks = np.arange(len(classNames))

    plt.xticks(tick_marks, classNames, rotation=45)
    plt.yticks(tick_marks, classNames)
    s = [['TN','FN'], ['FP', 'TP']]
    for i in range(2):
        for j in range(2):
            plt.text(j,i, str(s[i][j])+" = "+str(data[i][j]))
    plt.show()
```

3.4 confusion matrix

In [90]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

y_train_predicted_withthroshold=predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
y_test_predicted_withthroshold=predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)

cm_train=confusion_matrix(Y_train,y_train_predicted_withthroshold,labels=[0, 1])
```

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(cm_train)
print("="*100)
print("Accuracy score  for Train")
print(accuracy_score(Y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*100)

cm_test=confusion_matrix(Y_test,y_test_predicted_withthroshold,labels=[0, 1])

print("Test confusion matrix")
print(cm_test)
print("="*100)
print("Accuracy score  for Test")
accuracy_score_bow=accuracy_score(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr))
print(accuracy_score_bow)
print("="*100)
```

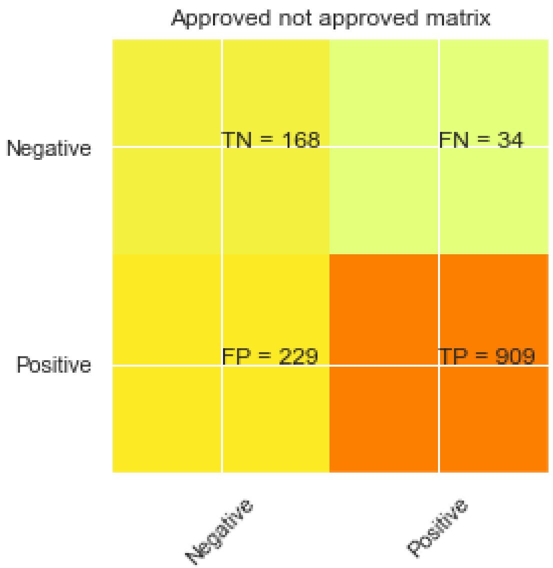
```
====================================================================================================
Train confusion matrix
[[168  34]
 [229 909]]
====================================================================================================
Accuracy score  for Train
0.8037313432835821
====================================================================================================
Test confusion matrix
[[ 75  24]
 [151 410]]
====================================================================================================
Accuracy score  for Test
0.7348484848484849
====================================================================================================
```

In [91]:

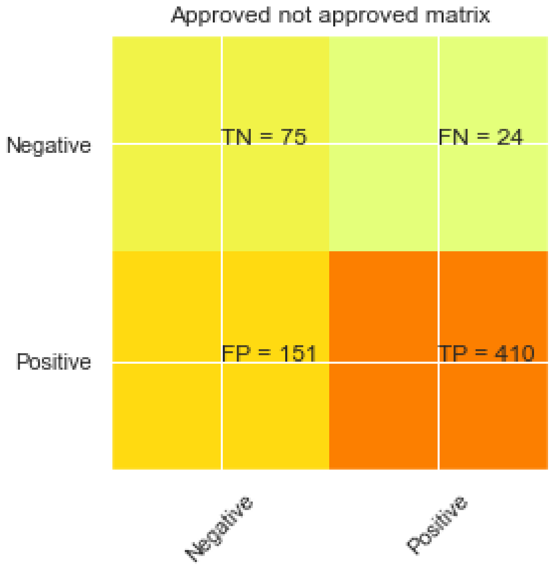
```
print("confusion matrix for train data")
print("="*100)
myplot_matrix1(cm_train)
print("confusion matrix for Test data")
```

```
print("="*100)
myplot_matrix1(cm_test)
```

```
confusion matrix for train data
====================================================================================================
```




```
confusion matrix for Test data
====================================================================================================
```



4.Model Performance Table

In [92]:
```
from prettytable import PrettyTable
x = PrettyTable()

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]
x.add_row(["AVG W2V", "XGBoost", "Max Depth:6 , n_estimators:10", 0.84])

print(x)
```

```
+------------+---------+-------------------------------+------+
| Vectorizer |  Model  |        Hyper Parameter        | AUC  |
+------------+---------+-------------------------------+------+
|  AVG W2V   | XGBoost | Max Depth:6 , n_estimators:10 | 0.84 |
+------------+---------+-------------------------------+------+
```

5.Observation

1.Using svd we get 84% accuracy in test data

2.True postive number in confusion matrix is good for test data