KNN on DonorsChoose

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
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.svm import SVC
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
from sklearn.model_selection import GridSearchCV

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

import warnings
warnings.filterwarnings(action='ignore', category=UserWarning, module='gensim')
import gensim

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

```
C:\Users\Others\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows; aliasi
ng chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

1.Load and process Data

In [2]:

```
data  = pd.read_csv('train_data.csv', nrows=30000)
resource_data=pd.read_csv('resources.csv')
data.head(5)
data.shape
```

Out[2]:

```
(30000, 17)
```

1.1 Merging resourse and project data

In [3]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
price_data.head(2)
```

Out[3]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [4]:

```
data = pd.merge(data, price_data, on='id', how='left')
```

1.2 process Project Essay

In [5]:

```
data["essay"] = data["project_essay_1"].map(str) +\
                data["project_essay_2"].map(str) + \
                data["project_essay_3"].map(str) + \
                data["project_essay_4"].map(str)
```

In [6]:

```
# 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 [7]:

```
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', 't
heir',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
```

```
'those', \
             'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'd
o', 'does', \
             'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'whil
e', 'of', \
             'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'bef
ore', 'after',\
             'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'a
gain', '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"]
```

In [8]:

```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(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())
data['cleaned_essay']=preprocessed_essays
```

```
100%|██████████| 30000/30000 [00:15<00:00, 1990.66it/s]
```

1.3 Project Title

In [9]:

```
# 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())
data['cleaned_project_title']=preprocessed_title
```

```
100%|██████████| 30000/30000 [00:00<00:00, 38612.53it/s]
```

In [10]:

```
data.head(2)
```

Out[10]:

Unnamed: id teacher_id teacher_prefix school_state project_submitted_datetime project_grade_c

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_c
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra

2 rows × 22 columns

1.4 teacher_prefix

In [11]:

```
#https://stackoverflow.com/questions/9452108/how-to-use-string-replace-in-python-3-x
temp1=data.teacher_prefix.apply(lambda x: str(x).replace('.', ''))
data['teacher_prefix']=temp1
data['teacher_prefix'].value_counts()
```

Out[11]:

```
Mrs        15682
Ms         10779
Mr          2895
Teacher      643
nan            1
Name: teacher_prefix, dtype: int64
```

1.5 project grade

In [12]:

```
data['project_grade_category'].value_counts()
```

Out[12]:

```
Grades PreK-2    12204
Grades 3-5       10160
Grades 6-8        4663
Grades 9-12       2973
Name: project_grade_category, dtype: int64
```

In [13]:

```
#https://stackoverflow.com/questions/9452108/how-to-use-string-replace-in-python-3-x
grade_list = []
for i in data['project_grade_category'].values:
     i=i.replace(' ','_')
     i=i.replace('-','_')
     grade_list.append(i.strip())


data['project_grade_category']=grade_list
```

In [14]:

```
data['project_grade_category'].value_counts()
```

Out[14]:

```
Grades PreK 2    12204
```

```
Grades_3_5       10160
Grades_6_8        4663
Grades_9_12       2973
Name: project_grade_category, dtype: int64
```

1.6 Making dependant(label) and independant variables

In [15]:

```
y = data['project_is_approved'].values
data.drop(['project_is_approved'], axis=1, inplace=True)
data.head(1)
x=data
```

1.7 Traing and Test split

In [16]:

```
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,rand
om_state=42)
```

2.Text Vectorization and encoding catagory

2.1 converting the essay to vectors using BOW

In [17]:

```
print(X_train.shape, Y_train.shape)
print(X_cv.shape, Y_cv.shape)
print(X_test.shape, Y_test.shape)

print("="*100)



from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['cleaned_essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['cleaned_essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['cleaned_essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['cleaned_essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, Y_train.shape)
print(X_cv_essay_bow.shape, Y_cv.shape)
print(X_test_essay_bow.shape, Y_test.shape)
print("="*100)
```

```
(13467, 21) (13467,)
(6633, 21) (6633,)
(9900, 21) (9900,)
====================================================================================================
After vectorizations
(13467, 5000) (13467,)
(6633, 5000) (6633,)
(9900, 5000) (9900,)
====================================================================================================
```

2.2 converting the title to vectors using BOW

In [18]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['cleaned_project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['cleaned_project_title'].values)
X_cv_title_bow = vectorizer.transform(X_cv['cleaned_project_title'].values)
X_test_title_bow = vectorizer.transform(X_test['cleaned_project_title'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, Y_train.shape)
print(X_cv_essay_bow.shape, Y_cv.shape)
print(X_test_essay_bow.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(13467, 5000) (13467,)
(6633, 5000) (6633,)
(9900, 5000) (9900,)
====================================================================================================
```

2.3 converting the essay to vectors using TFIDF

In [19]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['cleaned_essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['cleaned_essay'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['cleaned_essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['cleaned_essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, Y_train.shape)
print(X_cv_essay_tfidf.shape, Y_cv.shape)
print(X_test_essay_tfidf.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(13467, 7142) (13467,)
(6633, 7142) (6633,)
(9900, 7142) (9900,)
====================================================================================================
```

2.4 converting the title to vectors using TFIDF

In [20]:

```
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['cleaned_project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_tfidf = vectorizer.transform(X_train['cleaned_project_title'].values)
X_cv_title_tfidf = vectorizer.transform(X_cv['cleaned_project_title'].values)
X_test_title_tfidf = vectorizer.transform(X_test['cleaned_project_title'].values)

print("After vectorizations")
print(X_train_title_tfidf.shape, Y_train.shape)
print(X_cv_title_tfidf.shape, Y_cv.shape)
print(X_test_title_tfidf.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(13467, 839) (13467,)
(6633, 839) (6633,)
(9900, 839) (9900,)
====================================================================================================
```

2.5 load glove model for AvgW2V

In [21]:

```
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

# ============================
'''Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!
'''
# ============================
```

```
Loading Glove Model
```

```
1917495it [03:37, 8826.25it/s]
```

```
Done. 1917495  words loaded!
```

Out[21]:

```
'Output:\n    \nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n'
```

In [22]:

```
words = []
for i in X_train['cleaned_essay'].values:
    words.extend(i.split(' '))

for i in X_train['cleaned_project_title'].values:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and our coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-an
d-load-variables-in-python/

import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_courpus, f)
```

```
all the words in the coupus 2103255
```

```
the unique words in the coupus 25970
The number of words that are present in both glove vectors and our coupus 24794 ( 95.472 %)
word 2 vec length 24794
```

In [23]:

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

2.6 Avg w2v on essay using glove model

In [24]:

```python
Text_avg_w2v_train_essay= []; # 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(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
    Text_avg_w2v_train_essay.append(vector)

print(len(Text_avg_w2v_train_essay))
print(len(Text_avg_w2v_train_essay[0]))
```

```
100%|██████████| 13467/13467 [00:04<00:00, 3243.71it/s]
```

```
13467
300
```

In [25]:

```python
Text_avg_w2v_cv_essay= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['cleaned_essay'].values): # 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
    Text_avg_w2v_cv_essay.append(vector)

print(len(Text_avg_w2v_cv_essay))
print(len(Text_avg_w2v_cv_essay[0]))
```

```
100%|██████████| 6633/6633 [00:01<00:00, 3345.67it/s]
```

```
6633
300
```

In [26]:

```python
Text_avg_w2v_test_essay= []; # 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(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
    Text_avg_w2v_test_essay.append(vector)

print(len(Text_avg_w2v_test_essay))
print(len(Text_avg_w2v_test_essay[0]))
```

```
100%|██████████| 9900/9900 [00:02<00:00, 3325.41it/s]
```

```
9900
300
```

2.7 Avg w2v on title using glove model

In [27]:

```
Text_avg_w2v_train_title= []; # 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(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
    Text_avg_w2v_train_title.append(vector)

print(len(Text_avg_w2v_train_title))
print(len(Text_avg_w2v_train_title[0]))
```

```
100%|██████████| 13467/13467 [00:00<00:00, 71406.51it/s]
```

```
13467
300
```

In [28]:

```
Text_avg_w2v_cv_title= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['cleaned_project_title'].values): # 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
    Text_avg_w2v_cv_title.append(vector)

print(len(Text_avg_w2v_cv_title))
print(len(Text_avg_w2v_cv_title[0]))
```

```
100%|██████████| 6633/6633 [00:00<00:00, 66527.06it/s]
```

```
6633
300
```

In [29]:

```
Text_avg_w2v_test_title= []; # 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(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]
```

```
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    Text_avg_w2v_test_title.append(vector)

print(len(Text_avg_w2v_test_title))
print(len(Text_avg_w2v_test_title[0]))
```

```
100%|██████████| 9900/9900 [00:00<00:00, 67770.97it/s]
```

```
9900
300
```

2.8 Using Pretrained Models: TFIDF weighted W2V on essay

In [30]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['cleaned_essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [31]:

```
Text_tfidf_w2v_train_essay= []; # 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(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word
)/len(sentence.split())))
            tf_idf = 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
    Text_tfidf_w2v_train_essay.append(vector)

print(len(Text_tfidf_w2v_train_essay))
print(len(Text_tfidf_w2v_train_essay[0]))
```

```
100%|██████████| 13467/13467 [00:32<00:00, 415.71it/s]
```

```
13467
300
```

In [32]:

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

In [33]:

```
Text_tfidf_w2v_cv_essay= [];
for sentence in tqdm(X_cv['cleaned_essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
```

```
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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
    Text_tfidf_w2v_cv_essay.append(vector)

print(len(Text_tfidf_w2v_cv_essay))
print(len(Text_tfidf_w2v_cv_essay[0]))
```

```
100%|██████████| 6633/6633 [00:15<00:00, 422.24it/s]
```

```
6633
300
```

In [34]:

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

In [35]:

```
Text_tfidf_w2v_test_essay= [];
for sentence in tqdm(X_test['cleaned_essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = 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
    Text_tfidf_w2v_test_essay.append(vector)

print(len(Text_tfidf_w2v_test_essay))
print(len(Text_tfidf_w2v_test_essay[0]))
```

```
100%|██████████| 9900/9900 [00:23<00:00, 424.78it/s]
```

```
9900
300
```

2.9 TFIDF weighted W2V on title

In [36]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['cleaned_project_title'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [37]:

```
Text_tfidf_w2v_train_title= [];
for sentence in tqdm(X_train['cleaned_project_title'].values): # for each review/sentence
```

```python
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = 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
    Text_tfidf_w2v_train_title.append(vector)

print(len(Text_tfidf_w2v_train_title))
print(len(Text_tfidf_w2v_train_title[0]))
```

```
100%|██████████| 13467/13467 [00:00<00:00, 28602.41it/s]
```

```
13467
300
```

In [38]:

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

In [39]:

```python
Text_tfidf_w2v_cv_title= [];
for sentence in tqdm(X_cv['cleaned_project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = 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
    Text_tfidf_w2v_cv_title.append(vector)

print(len(Text_tfidf_w2v_cv_title))
print(len(Text_tfidf_w2v_cv_title[0]))
```

```
100%|██████████| 6633/6633 [00:00<00:00, 28118.51it/s]
```

```
6633
300
```

In [40]:

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

In [41]:

```python
Text_tfidf_w2v_test_title= [];
```

```
for sentence in tqdm(X_test['cleaned_essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word
)/len(sentence.split())))
            tf_idf = 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
    Text_tfidf_w2v_test_title.append(vector)

print(len(Text_tfidf_w2v_test_title))
print(len(Text_tfidf_w2v_test_title[0]))
```

```
100%|██████████| 9900/9900 [00:23<00:00, 423.55it/s]
```

```
9900
300
```

2.10 one hot encoding the catogorical features: teacher_prefix

In [42]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, Y_train.shape)
print(X_cv_teacher_ohe.shape, Y_cv.shape)
print(X_test_teacher_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(13467, 5) (13467,)
(6633, 5) (6633,)
(9900, 5) (9900,)
['mr', 'mrs', 'ms', 'nan', 'teacher']
====================================================================================================
```

2.11 one hot encoding the catogorical features: project Grade

In [43]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, Y_train.shape)
print(X_cv_grade_ohe.shape, Y_cv.shape)
print(X_test_grade_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(13467, 4) (13467,)
(6633, 4) (6633,)
(9900, 4) (9900,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================================
```

2.12 one hot encoding the catogorical features: state

In [44]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, Y_train.shape)
print(X_cv_state_ohe.shape, Y_cv.shape)
print(X_test_state_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(13467, 51) (13467,)
(6633, 51) (6633,)
(9900, 51) (9900,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', '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']
====================================================================================================
```

2.13 Normalizing the numerical features: Price

In [45]:

```
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_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['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
(13467, 1) (13467,)
(6633, 1) (6633,)
(9900, 1) (9900,)
====================================================================================================
```

3. Applying KNN on BOW

3.1 BOW:Concatinating all the features

In [46]:

```
from scipy.sparse import hstack
```

```
X_tr_bow = hstack((X_train_essay_bow,X_train_title_bow, X_train_state_ohe, X_train_teacher_ohe, X_train
_grade_ohe, X_train_price_norm)).tocsr()
X_cr_bow = hstack((X_cv_essay_bow,X_cv_title_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_c
v_price_norm)).tocsr()
X_te_bow = hstack((X_test_essay_bow,X_test_title_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grad
e_ohe, X_test_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr_bow.shape, Y_train.shape)
print(X_cr_bow.shape, Y_cv.shape)
print(X_te_bow.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(13467, 6280) (13467,)
(6633, 6280) (6633,)
(9900, 6280) (9900,)
====================================================================================================
```

3.2 Hyper parameter Tuning:simple for loop for Train and cross validation

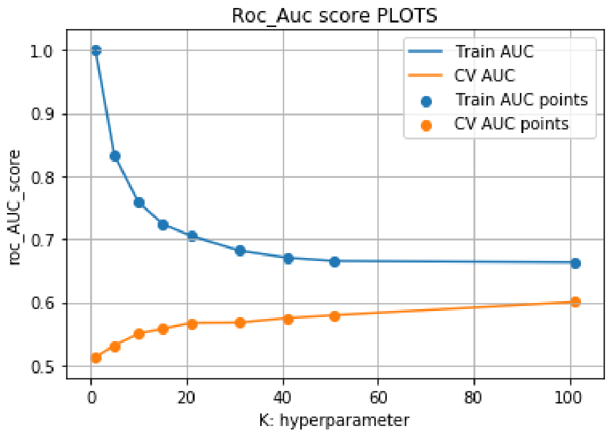
In [47]:

```
#https://stackabuse.com/understanding-roc-curves-with-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
```

In [48]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []


K = [1, 5, 10, 15, 21, 31, 41, 51, 101]

for i in K:
    model = KNeighborsClassifier(n_neighbors=i,algorithm="brute")
    model.fit(X_tr_bow,Y_train)
    y_tr_prob = batch_predict(model,X_tr_bow)
    y_cr_prob =batch_predict(model, X_cr_bow)

    train_auc.append(roc_auc_score(Y_train,y_tr_prob))
    cv_auc.append(roc_auc_score(Y_cv,y_cr_prob))


plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("roc_AUC_score")
plt.title("Roc_Auc score PLOTS")
plt.grid()
```

```
plt.show()
```



observations

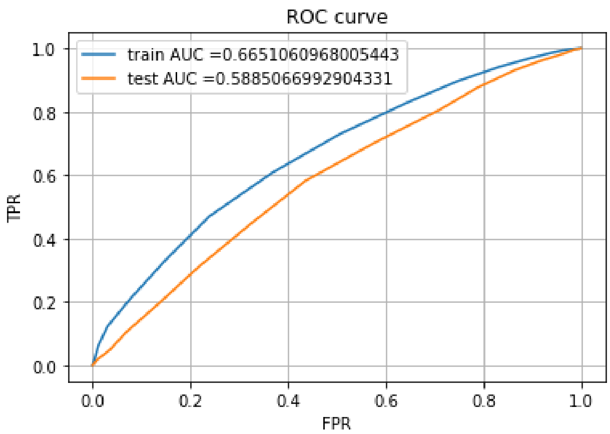
1.By observing plot of auc score of each k for train and cross validation we understand k=51 is best hyperparameter as train auc and cross validation auc is very close when k=51.

2.Choosen K value has high cross validation AUC.If K is small it shows overfit and if K is large it shows underfit.K=51 is best K where value of cross validation is high.it is neither small nor high.

3.3 ROC curve with best K

In [92]:

```
from sklearn.metrics import roc_curve, auc
k_bow=51

k_val = KNeighborsClassifier(n_neighbors=k_bow)
k_val.fit(X_tr_bow, Y_train)


y_train_pred = batch_predict(k_val, X_tr_bow)
y_test_pred = batch_predict(k_val, X_te_bow)

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)
auc_bow=auc(test_fpr, test_tpr)
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("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```



observations

1.By looking ROC curve of Training FPR and TPR it looks sensible as it is greater than diagonal line or 0.5

2.By looking ROC curve of Test FPR and TPR is not that much sensible as it is very close to diagonal.If this curve is less than 0.5 then we can revert values i.e assign 1 for 0 and 0 for 1

3.4 confusion matrix

In [50]:

```
# https://tatwan.github.io/How-To-Plot-A-Confusion-Matrix-In-Python/
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()
```

In [51]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
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 [81]:

```
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)
```

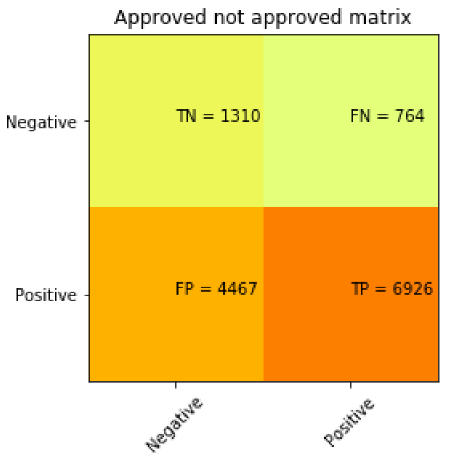
```
cm_test=confusion_matrix(Y_test,y_test_predicted_withthroshold,labels=[0, 1])

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
[[1215  859]
 [4756 6637]]
====================================================================================================
Accuracy score  for Train
0.5830548748793347
====================================================================================================
Test confusion matrix
[[ 481 1044]
 [2427 5948]]
====================================================================================================
Accuracy score  for Test
0.6493939393939394
====================================================================================================
```

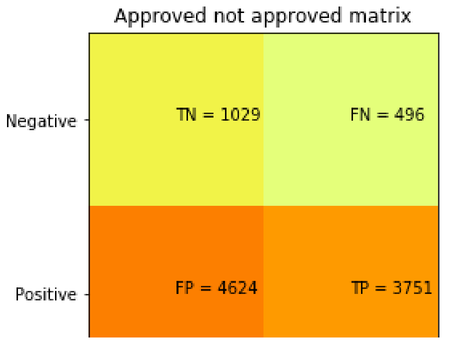
In [53]:

```
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
====================================================================================================
```



observations

1.TN and TP of train data is higher than test data.Model perform good on train data than test data.

2.Accuracy score on train data is 65% and test data is 54%.

4. Apply KNN on TFIDF

4.1 TFIDF:Concatinating all the features

In [54]:

```
from scipy.sparse import hstack
X_tr_tfidf = hstack((X_train_essay_tfidf,X_train_title_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_norm)).tocsr()
X_cr_tfidf = hstack((X_cv_essay_tfidf,X_cv_title_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm)).tocsr()
X_te_tfidf = hstack((X_test_essay_tfidf,X_test_title_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr_tfidf.shape, Y_train.shape)
print(X_cr_tfidf.shape, Y_cv.shape)
print(X_te_tfidf.shape, Y_test.shape)
print("="*100)
```

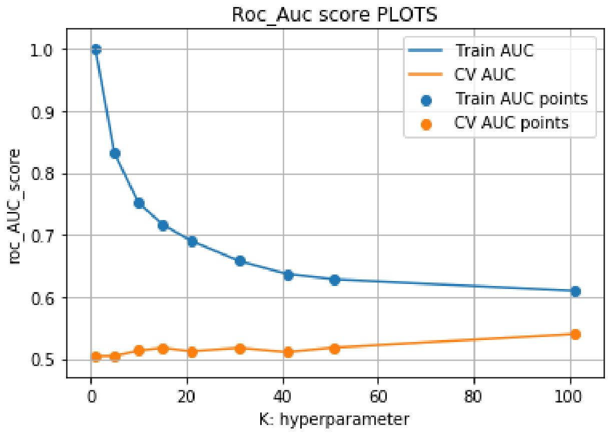
```
Final Data matrix
(13467, 8042) (13467,)
(6633, 8042) (6633,)
(9900, 8042) (9900,)
====================================================================================================
```

4.2 Hyper parameter Tuning:simple for loop for Train and cross validation

In [55]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []


K = [1, 5, 10, 15, 21, 31, 41, 51]

for i in K:
    model = KNeighborsClassifier(n_neighbors=i,algorithm="brute")
    model.fit(X_tr_tfidf,Y_train)
    y_tr_prob = batch_predict(model,X_tr_tfidf)
    y_cr_prob =batch_predict(model, X_cr_tfidf)

    train_auc.append(roc_auc_score(Y_train,y_tr_prob))
    cv_auc.append(roc_auc_score(Y_cv,y_cr_prob))


plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')
```

```
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("roc_AUC_score")
plt.title("Roc_Auc score PLOTS")
plt.grid()
plt.show()
```



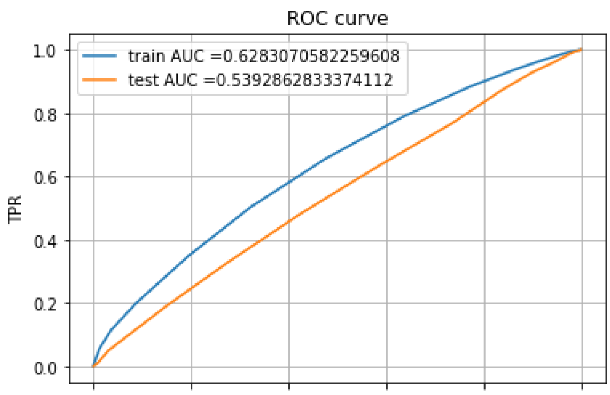
observations

1.By observing plot of auc score of each k for train and cross validation we understand k=51 is best hyperparameter as cross validation auc is high for k=51

4.3 ROC curve with best K

In [94]:

```
k_tfidf=51 # by grid search

k_val = KNeighborsClassifier(n_neighbors=k_tfidf)
k_val.fit(X_tr_tfidf, Y_train)


y_train_pred = batch_predict(k_val, X_tr_tfidf)
y_test_pred = batch_predict(k_val, X_te_tfidf)


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)
auc_tfidf=auc(test_fpr, test_tpr)
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("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```



observations

1.model perform good on train data than test data

2.model is not sensible for test data.

4.4 confusion matrix

In [82]:

```
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_tfidf=accuracy_score(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr))
print(accuracy_score_tfidf)
print("="*100)
```

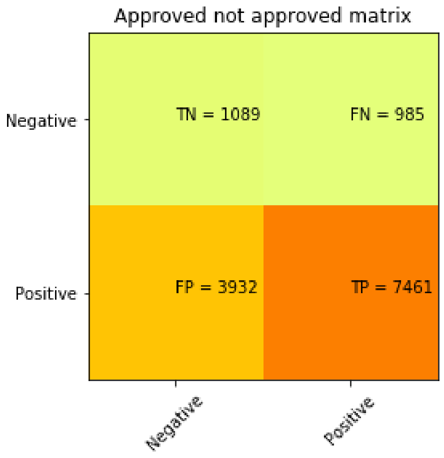
```
====================================================================================================
Train confusion matrix
[[1215  859]
 [4756 6637]]
====================================================================================================
Accuracy score  for Train
0.5830548748793347
====================================================================================================
Test confusion matrix
[[ 481 1044]
 [2427 5948]]
====================================================================================================
Accuracy score  for Test
0.6493939393939394
====================================================================================================
```

In [58]:

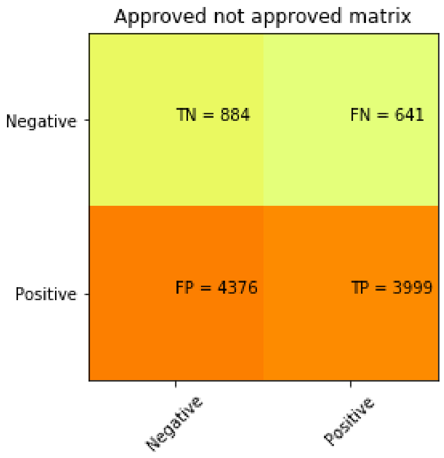
```
print("confusion matrix for train data")
print("="*100)
myplot_matrix1(cm_train)
print("confusion matrix for Test data")

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

```
confusion matrix for train data
```

==




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



observations

1.TN and TP of train data is higher than test data.Model perform good on train data than test data.

2.Accuracy score on train data is 63% and test data is 49%.

5.Knn on AVGW2V

5.1 Avgw2v:Concatinating all the features

In [59]:

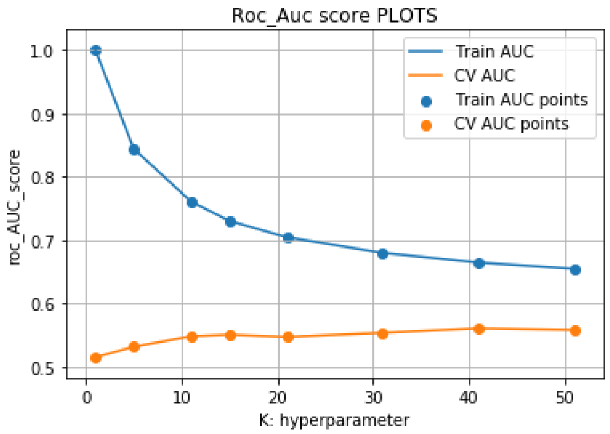
```
from scipy.sparse import hstack
X_tr_avgw2v = hstack((Text_avg_w2v_train_essay,Text_avg_w2v_train_title, X_train_state_ohe, X_train_tea
cher_ohe, X_train_grade_ohe, X_train_price_norm)).tocsr()
X_cr_avgw2v = hstack((Text_avg_w2v_cv_essay,Text_avg_w2v_cv_title, X_cv_state_ohe, X_cv_teacher_ohe, X_
cv_grade_ohe, X_cv_price_norm)).tocsr()
X_te_avgw2v = hstack((Text_avg_w2v_test_essay,Text_avg_w2v_test_title, X_test_state_ohe, X_test_teacher
_ohe, X_test_grade_ohe, X_test_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr_avgw2v.shape, Y_train.shape)
print(X_cr_avgw2v.shape, Y_cv.shape)
print(X_te_avgw2v.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(13467, 661) (13467,)
(6633, 661) (6633,)
(9900, 661) (9900,)
====================================================================================================
```

5.2 Hyper parameter Tuning:simple for loop for Train and cross validation

In [79]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []


K = [1, 5, 11, 15, 21, 31, 41, 51]

for i in K:
    model = KNeighborsClassifier(n_neighbors=i,algorithm="brute")
    model.fit(X_tr_avgw2v,Y_train)
    y_tr_prob = batch_predict(model,X_tr_avgw2v)
    y_cr_prob =batch_predict(model, X_cr_avgw2v)

    train_auc.append(roc_auc_score(Y_train,y_tr_prob))
    cv_auc.append(roc_auc_score(Y_cv,y_cr_prob))


plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("roc_AUC_score")
plt.title("Roc_Auc score PLOTS")
plt.grid()
plt.show()
```

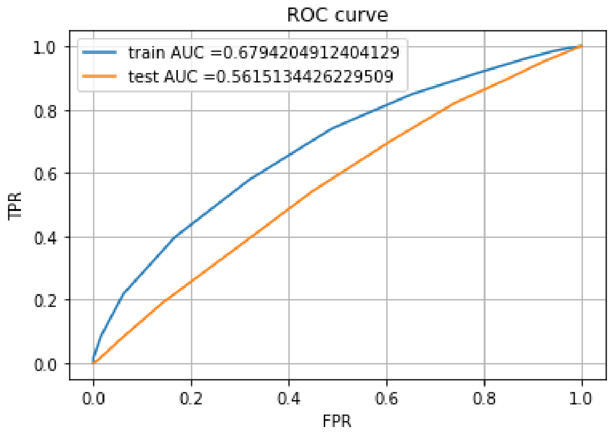


5.3 ROC curve with best k

In [95]:

```
k_avgw2v=31 # by grid search

k_val = KNeighborsClassifier(n_neighbors=k_avgw2v)
k_val.fit(X_tr_avgw2v, Y_train)


y_train_pred = batch_predict(k_val, X_tr_avgw2v)
```

```
y_test_pred = batch_predict(k_val, X_te_avgw2v)


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)
auc_avgw2v=auc(test_fpr, test_tpr)
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("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```



observations

1.Model perform good on train data than test data.

2.Test AUC is very close to random model.

5.4 confusion matrix

In [85]:

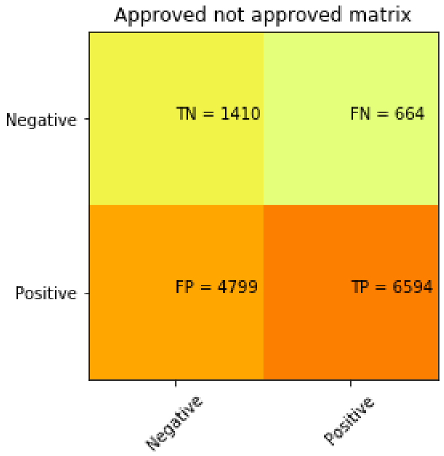
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

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_avgw2v=accuracy_score(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr))
print(accuracy_score_avgw2v)
print("="*100)
```

```
====================================================================================================
Train confusion matrix
[[1215  859]
 [4756 6637]]
====================================================================================================
Accuracy score  for Train
0.5830548748793347
====================================================================================================
Test confusion matrix
[[ 481 1044]
 [2427 5948]]
====================================================================================================
Accuracy score  for Test
0.6493939393939394
====================================================================================================
```

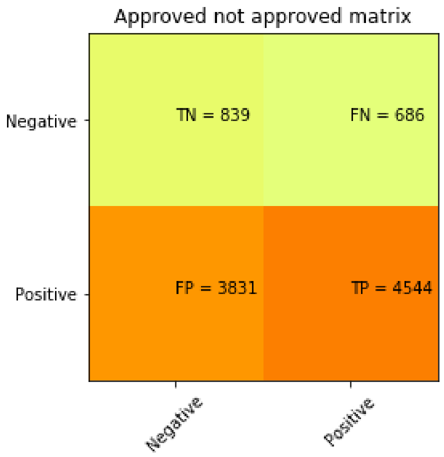
In [67]:

```
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
====================================================================================================
```



observations

In [ ]:

```
1.TN and TP of train data is higher than test data.Model perform good on train data than test data
```

In [ ]:

```
2..Accuracy score on train data is 59% and test data is 54%.
```

6. Knn on TFIDF W2V

6.1 TFIDF:Concatinating all the features

In [64]:

```
from scipy.sparse import hstack
X_tr_tfidfw2v = hstack((Text_tfidf_w2v_train_essay,Text_tfidf_w2v_train_title, X_train_state_ohe, X_tra
in_teacher_ohe, X_train_grade_ohe, X_train_price_norm)).tocsr()
X_cr_tfidfw2v = hstack((Text_tfidf_w2v_cv_essay,Text_tfidf_w2v_cv_title, X_cv_state_ohe, X_cv_teacher_o
he, X_cv_grade_ohe, X_cv_price_norm)).tocsr()
X_te_tfidfw2v = hstack((Text_tfidf_w2v_test_essay,Text_tfidf_w2v_test_title, X_test_state_ohe, X_test_t
eacher_ohe, X_test_grade_ohe, X_test_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr_tfidfw2v.shape, Y_train.shape)
print(X_cr_tfidfw2v.shape, Y_cv.shape)
print(X_te_tfidfw2v.shape, Y_test.shape)
print("="*100)
```

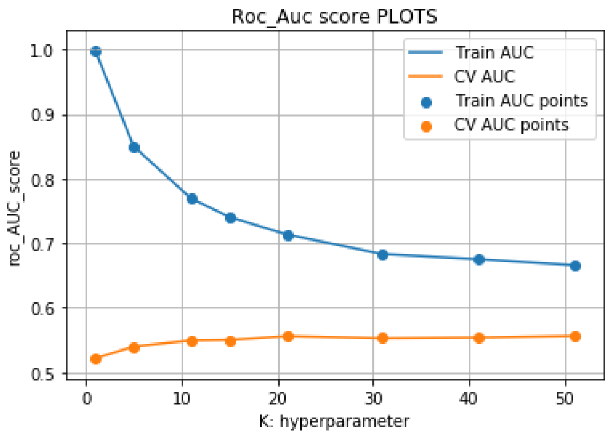
```
Final Data matrix
(13467, 661) (13467,)
(6633, 661) (6633,)
(9900, 661) (9900,)
====================================================================================================
```

6.2 Hyper parameter Tuning:simple for loop for Train and cross validation

In [127]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []


K = [1, 5, 11, 15, 21, 31, 41, 51]

for i in K:
    model = KNeighborsClassifier(n_neighbors=i,algorithm="brute")
    model.fit(X_tr_avgw2v,Y_train)
    y_tr_prob = batch_predict(model,X_tr_tfidfw2v)
    y_cr_prob =batch_predict(model, X_cr_tfidfw2v)

    train_auc.append(roc_auc_score(Y_train,y_tr_prob))
    cv_auc.append(roc_auc_score(Y_cv,y_cr_prob))


plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("roc_AUC_score")
plt.title("Roc Auc score PLOTS")
```

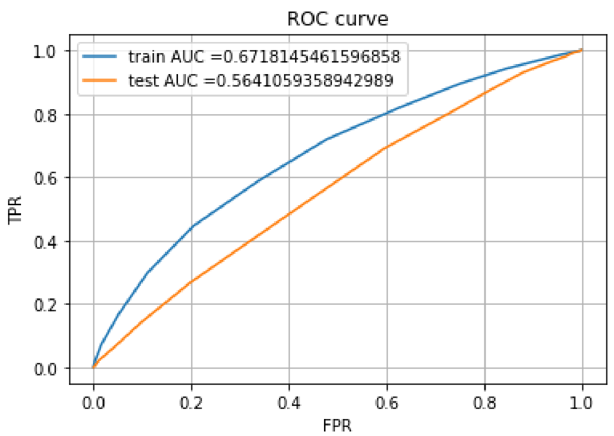
```
plt.title("Roc_Auc score PLOTS")
plt.grid()
plt.show()
```



6.3 ROC curve with best k

In [96]:

```
k_tfidfw2v=41 # by grid search

k_val = KNeighborsClassifier(n_neighbors=k_tfidfw2v)
k_val.fit(X_tr_tfidfw2v, Y_train)


y_train_pred = batch_predict(k_val, X_tr_avgw2v)
y_test_pred = batch_predict(k_val, X_te_avgw2v)


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)
auc_tfidfw2v=auc(test_fpr, test_tpr)
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("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```



observations

1..Model perform good on train data than test data

2.TEST AUC is very close to random model

6.4 confusion matrix

In [83]:

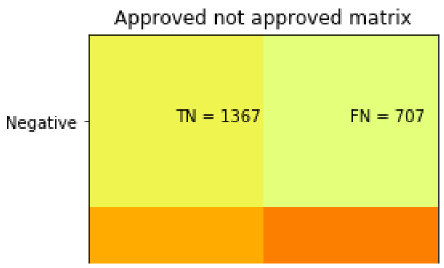
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

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_tfidfw2v=accuracy_score(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr))
print(accuracy_score_tfidfw2v)
print("="*100)
```

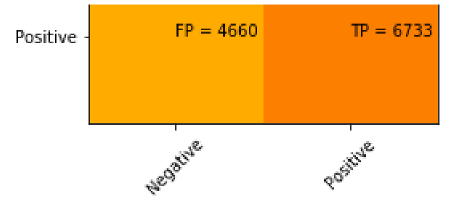
```
====================================================================================================
Train confusion matrix
[[1215  859]
 [4756 6637]]
====================================================================================================
Accuracy score  for Train
0.5830548748793347
====================================================================================================
Test confusion matrix
[[ 481 1044]
 [2427 5948]]
====================================================================================================
Accuracy score  for Test
0.6493939393939394
====================================================================================================
```

In [70]:

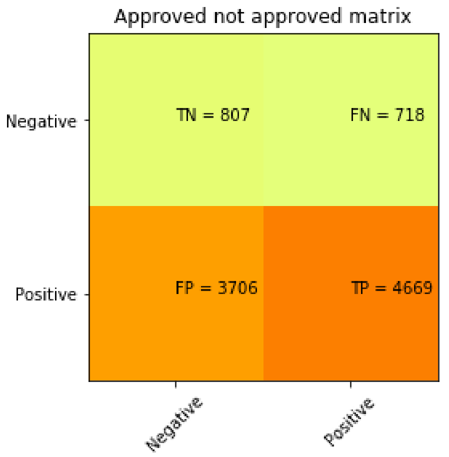
```
print("confusion matrix for train data")
print("="*100)
myplot_matrix1(cm_train)
print("confusion matrix for Test data")

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

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






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



observations

1.TN and TP of train data is higher than test data.Model perform good on train data than test data

2.Accuracy score on train data is 60% and test data is 55%.

7.Considering 2000 points of TFIDF

7.1 TFIDF:Concatinating all the features

In [ ]:

```
from scipy.sparse import hstack
X_tr_tfidf = hstack((X_train_essay_tfidf,X_train_title_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_norm)).tocsr()
X_cr_tfidf = hstack((X_cv_essay_tfidf,X_cv_title_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm)).tocsr()
X_te_tfidf = hstack((X_test_essay_tfidf,X_test_title_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr_tfidf.shape, Y_train.shape)
print(X_cr_tfidf.shape, Y_cv.shape)
print(X_te_tfidf.shape, Y_test.shape)
print("="*100)
```

7.2 Select best 2000 features on train by chi2

In [71]:

```
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, chi2
select_feature = SelectKBest(chi2, k=2000).fit(X_tr_tfidf,Y_train)
```

7.3 Transform train test cross validation

In [72]:

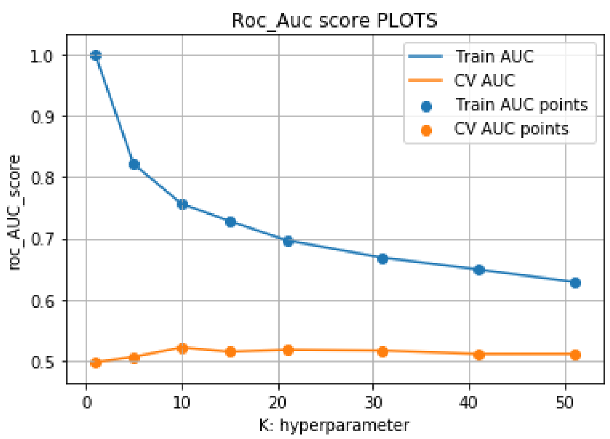
```
X_tr_feature_select=select_feature.transform(X_tr_tfidf)
X_cr_feature_select=select_feature.transform(X_cr_tfidf)
X_te_feature_select=select_feature.transform(X_te_tfidf)
print(X_tr_feature_select.shape,Y_train.shape)
print(X_cr_feature_select.shape,Y_cv.shape)
print(X_te_feature_select.shape,Y_test.shape)
```

```
(13467, 2000) (13467,)
(6633, 2000) (6633,)
(9900, 2000) (9900,)
```

7.4 Hyper parameter Tuning:simple for loop for Train and cross validation

In [123]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []


K = [1, 5, 10, 15, 21, 31, 41, 51]

for i in K:
    model = KNeighborsClassifier(n_neighbors=i,algorithm="brute")
    model.fit(X_tr_feature_select,Y_train)
    y_tr_prob = batch_predict(model,X_tr_feature_select)
    y_cr_prob =batch_predict(model, X_cr_feature_select)

    train_auc.append(roc_auc_score(Y_train,y_tr_prob))
    cv_auc.append(roc_auc_score(Y_cv,y_cr_prob))


plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("roc_AUC_score")
plt.title("Roc_Auc score PLOTS")
plt.grid()
plt.show()
```

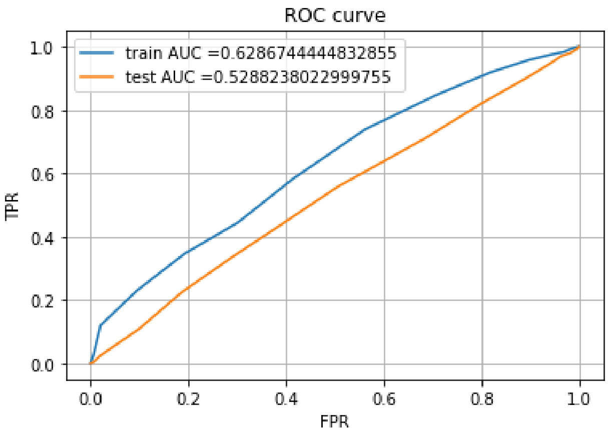


7.6 ROC curve with best k

7.6 ROC curve with best k

In [126]:

```
k_tfidfs=31 # by grid search

k_val = KNeighborsClassifier(n_neighbors=k_tfidf)
k_val.fit(X_tr_feature_select, Y_train)


y_train_pred = batch_predict(k_val, X_tr_feature_select)
y_test_pred = batch_predict(k_val, X_te_feature_select)


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("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```



observations

1.Model perform good on train data than test data

2.TEST AUC is very close to random model

7.7 confusion matrix

In [75]:

```
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)
```

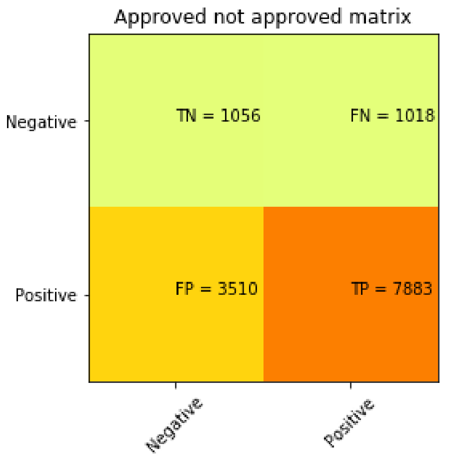
```python
cm_test=confusion_matrix(Y_test,y_test_predicted_withthroshold,labels=[0, 1])

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

```
====================================================================================================
Train confusion matrix
[[1056 1018]
 [3510 7883]]
====================================================================================================
Accuracy score  for Train
0.6637706987450805
====================================================================================================
Test confusion matrix
[[ 825  700]
 [4129 4246]]
====================================================================================================
Accuracy score  for Test
0.5122222222222222
====================================================================================================
```

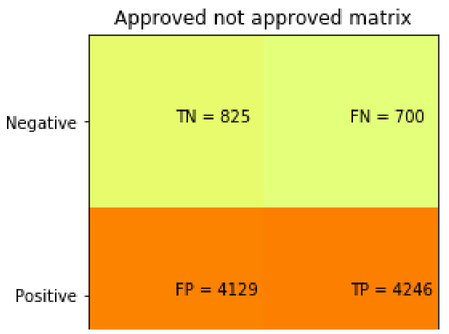
In [76]:

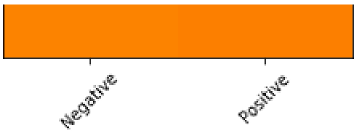
```python
print("confusion matrix for train data")
print("="*100)
myplot_matrix1(cm_train)
print("confusion matrix for Test data")

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

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




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





observations

1.TN and TP of train data is higher than test data.Model perform good on train data than test data

2.Accuracy score on train data is 66% and test data is 51%.

8.MODEL PERFORMANCE TABLE

In [125]:

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

x.field_names = ["Vectorizer", "Model", "Hyper Parameter(k)", "AUC"]
x.add_row(["KNN with Bow", "Brute", k_bow,auc_bow])
x.add_row(["KNN with TFIDF","Brute",k_tfidf,auc_tfidf])
x.add_row(["KNN with AVGW2V","Brute", k_avgw2v,auc_avgw2v])
x.add_row(["KNN with TFIDF W2V","Brute",k_tfidfw2v,auc_tfidfw2v])
print(x)
```

```
+--------------------+-------+--------------------+--------------------+
|     Vectorizer     | Model | Hyper Parameter(k) |        AUC         |
+--------------------+-------+--------------------+--------------------+
|    KNN with Bow    | Brute |         51         | 0.5885066992904331 |
|   KNN with TFIDF   | Brute |         51         | 0.5392862833374112 |
|  KNN with AVGW2V   | Brute |         31         | 0.5615134426229509 |
| KNN with TFIDF W2V | Brute |         41         | 0.5641059358942989 |
+--------------------+-------+--------------------+--------------------+
```

summary

1.By looking AUC score plots with diiferent k values we get best value for hyperparameter k=51

2.All Models performs good on training data but poor performence on unseen data(test data).

3.False Positive: (Type 1 Error):in both train and test of confusion matrix is high

4.False Negative: (Type 2 Error):in both train and test of confusion matrix is low

5.KNN with Bow with hyperparamer k=51 give high AUC than other models.