Assignment-3 Apply k-NN on Donors Choose dataset.

In [1]:

```
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
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from tqdm import tqdm
import os
from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

1.1 Loading Data

In [2]:

```
data = pd.read_csv('preprocessed_data.csv', nrows=50000)
data.head(2)
```

Out[2]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_s
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL

2 rows × 23 columns

In [3]:

```
# To ensure we are dealing with an imbalanced data set.
data['project_is_approved'].value_counts()
```

Out[3]:

```
1    42286
0     7714
Name: project_is_approved, dtype: int64
```

In [4]:

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

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_sta
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN

1 rows × 22 columns

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

In [5]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, strat
ify=y_train)
```

1.3 Make Data Model Ready: encoding essay, and project_title

1.3.1 Vectorizing preprocessed essays & project_title using BOW

In [6]:

```
# preprocessed essays
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)


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

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

```
(22445, 22) (22445,)
(11055, 22) (11055,)
(16500, 22) (16500,)
==========================================================================
==========================
```

In [7]:

```
print("After vectorization")
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 vectorization
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
==========================================================================
==========================
```

In [8]:

```
#project_title
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['preprocessed_titles'].values)

X_train_title_bow = vectorizer.transform(X_train['preprocessed_titles'].values)
X_cv_title_bow = vectorizer.transform(X_cv['preprocessed_titles'].values)
X_test_title_bow = vectorizer.transform(X_test['preprocessed_titles'].values)
```

In [9]:

```
print("After vectorization")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(22445, 1984) (22445,)
(11055, 1984) (11055,)
(16500, 1984) (16500,)
==========================================================================
==========================
```

1.3.2 Vectorizing preprocessed essays & project_title using TFIDF

In [10]:

```
#TFIDF for preprocessed_essays
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['preprocessed_essays'].values)

X_train_essay_tfidf = vectorizer.transform(X_train['preprocessed_essays'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['preprocessed_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['preprocessed_essays'].values)
```

In [11]:

```
print("After vectorization")
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 vectorization
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
==========================================================================
==========================
```

In [12]:

```
#TFIDF for preprocessed_titles
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['preprocessed_titles'].values)

X_train_titles_tfidf = vectorizer.transform(X_train['preprocessed_titles'].values)
X_cv_titles_tfidf = vectorizer.transform(X_cv['preprocessed_titles'].values)
X_test_titles_tfidf = vectorizer.transform(X_test['preprocessed_titles'].values)
```

In [13]:

```
print("After vectorization")
print(X_train_titles_tfidf.shape, y_train.shape)
print(X_cv_titles_tfidf.shape, y_cv.shape)
print(X_test_titles_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(22445, 1984) (22445,)
(11055, 1984) (11055,)
(16500, 1984) (16500,)
===========================================================================
==========================
```

1.3.3 Vectorizing preprocessed essays & project_title using Avg W2V

1.3.3.1 For preprocessed_titles

In [14]:

```
#Avg W2V for preprocessed_titles
#Train your own Word2Vec model using your own text corpus
import warnings
warnings.filterwarnings("ignore")
#train data
w2v_data= X_train['preprocessed_titles']
split_title_train=[]
for row in w2v_data:
    split_title_train.append([word for word in row.split()])     #splitting words

#train your W2v
train_w2v = Word2Vec(split_title_train,min_count=1,size=50, workers=4)
word_vectors_train = train_w2v.wv
w2v_words_train =list(word_vectors_train.vocab)
print(len(w2v_words_train ))
```

```
8048
```

In [15]:

```
# compute average word2vec for each title.
sent_vectors_train = [] # the avg-w2v for each title is stored in this list
for sent in tqdm(split_title_train):   # for each title
    sent_vec = np.zeros(50)  # as word vectors are of zero length 50
    cnt_words =0   # num of words with a valid vector in the title
    for word in sent:   # for each word in a title
        if word in w2v_words_train:
            vec = word_vectors_train[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
        sent_vectors_train.append(sent_vec)
print(len(sent_vectors_train))
print(len(sent_vectors_train[3]))
#from scipy.sparse import coo_matrix
#Avg_w2v_train=coo_matrix(sent_vectors_train) #https://docs.scipy.org/doc/scipy/referen
ce/generated/scipy.sparse.coo_matrix.html
#Avg_w2v_train.shape
```

```
100%|
| 22445/22445 [00:03<0
0:00, 6611.20it/s]

22445
50
```

In [16]:

```
# For CV data

# compute average word2vec for each title.
sent_vectors_cv = [] # the avg-w2v for each title is stored in this list
for sent in tqdm(X_cv['preprocessed_titles']):   # for each title
    sent_vec = np.zeros(50)  # as word vectors are of zero length 50
    cnt_words =0   # num of words with a valid vector in the title
    for word in sent:   # for each word in a title
        if word in w2v_words_train:
            vec = word_vectors_train[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
        sent_vectors_cv.append(sent_vec)
print(len(sent_vectors_cv))
print(len(sent_vectors_cv[3]))
#from scipy.sparse import coo_matrix
#Avg_w2v_cv=coo_matrix(sent_vectors_cv) #https://docs.scipy.org/doc/scipy/reference/gen
erated/scipy.sparse.coo_matrix.html
#Avg_w2v_cv.shape
```

```
100%|
| 11055/11055 [00:13<0
0:00, 796.24it/s]

11055
50
```

In [17]:

```
# For test data

# compute average word2vec for each title.
sent_vectors_test = [] # the avg-w2v for each title is stored in this list
for sent in tqdm(X_test['preprocessed_titles']):   # for each title
    sent_vec = np.zeros(50)  # as word vectors are of zero length 50
    cnt_words =0   # num of words with a valid vector in the title
    for word in sent:   # for each word in a title
        if word in w2v_words_train:
            vec = word_vectors_train[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
        sent_vectors_test.append(sent_vec)
print(len(sent_vectors_test))
print(len(sent_vectors_test[3]))
#from scipy.sparse import coo_matrix
#Avg_w2v_test=coo_matrix(sent_vectors_test) #https://docs.scipy.org/doc/scipy/referenc
e/generated/scipy.sparse.coo_matrix.html
#Avg_w2v_test.shape
```

```
100%|██████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████| 16500/16500 [00:20<0
0:00, 802.74it/s]

16500
50
```

1.3.3.1 For preprocessed_essays

Using Pretrained Models: Avg W2V

In [18]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-p
ickle-to-save-and-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())
print ("Done.",len(model)," words loaded!")
```

```
Done. 51510  words loaded!
```

In [19]:

```
# Avg W2V for train data
# compute average word2vec for each review.
avg_w2v_essay_train = []   # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_train['preprocessed_essays']):   # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0 # num of words with a valid vector in the sentence/review
    for word in sentence.split():  # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_train.append(vector)
print(len(avg_w2v_essay_train))
print(len(avg_w2v_essay_train[0]))
```

```
100%|[redacted]| 22445/22445 [00:09<0
0:00, 2337.87it/s]

22445
300
```

In [20]:

```
# Avg W2V for cv data

avg_w2v_essay_cv = []   # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['preprocessed_essays']):   # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0 # num of words with a valid vector in the sentence/review
    for word in sentence.split():  # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_cv.append(vector)
print(len(avg_w2v_essay_cv))
print(len(avg_w2v_essay_cv[0]))
```

```
100%|[redacted]| 11055/11055 [00:04<0
0:00, 2369.27it/s]

11055
300
```

In [21]:

```
# Avg W2V for test data

avg_w2v_essay_test = []   # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_essays']):   # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0 # num of words with a valid vector in the sentence/review
    for word in sentence.split():  # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_test.append(vector)
print(len(avg_w2v_essay_test))
print(len(avg_w2v_essay_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
██████████████████████████████████████████████████| 16500/16500 [00:07<0
0:00, 2328.84it/s]

16500
300
```

1.3.4 Vectorizing preprocessed essays & project_title using TFIDF weighted W2V

1.3.4.1 For preprocessed essays

In [22]:

```
# For train data

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

In [23]:

```
# average Word2Vec using pretrained models
# compute average word2vec for each review.
tfidf_w2v_train_essay = [] # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_essays):
            vec = model[word]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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
        tfidf_w2v_train_essay.append(vector)
print(len(tfidf_w2v_train_essay))
print(len(tfidf_w2v_train_essay[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████| 22445/22445 [01:07<0
0:00, 332.15it/s]

22445
300
```

In [24]:

```
#For cv data

tfidf_w2v_cv_essay = [] # the avg-w2v for each sentence/review is stored in this list
for sentence1 in tqdm(X_cv['preprocessed_essays']): # for each review/sentence
    vector1 = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight1 =0; # num of words with a valid vector in the sentence/review
    for word1 in sentence1.split(): # for each word in a review/sentence
        if (word1 in glove_words) and (word1 in tfidf_words_essays):
            vec1 = model[word1]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf1 = dictionary[word1]*(sentence1.count(word1)/len(sentence1.split()))
# getting the tfidf value for each word
            vector1 += (vec1 * tf_idf1) # calculating tfidf weighted w2v
            tf_idf_weight1 += tf_idf1
    if tf_idf_weight1 != 0:
        vector1 /= tf_idf_weight1
        tfidf_w2v_cv_essay.append(vector1)
print(len(tfidf_w2v_cv_essay))
print(len(tfidf_w2v_cv_essay[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████| 11055/11055 [00:33<0
0:00, 328.39it/s]

11055
300
```

In [25]:

```
# For test data

tfidf_w2v_test_essay = [] # the avg-w2v for each sentence/review is stored in this list
for sentence2 in tqdm(X_test['preprocessed_essays']): # for each review/sentence
    vector2 = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight2 =0; # num of words with a valid vector in the sentence/review
    for word2 in sentence2.split(): # for each word in a review/sentence
        if (word2 in glove_words) and (word2 in tfidf_words_essays):
            vec2 = model[word2]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf2 = dictionary[word2]*(sentence2.count(word2)/len(sentence2.split()))
# getting the tfidf value for each word
            vector2 += (vec2 * tf_idf2) # calculating tfidf weighted w2v
            tf_idf_weight2 += tf_idf2
    if tf_idf_weight2 != 0:
        vector2 /= tf_idf_weight2
        tfidf_w2v_test_essay.append(vector2)
print(len(tfidf_w2v_test_essay))
print(len(tfidf_w2v_test_essay[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████| 16500/16500 [00:51<0
0:00, 322.57it/s]

16500
300
```

1.3.4.2 For preprocessed titles

Using pretrained models

In [26]:

```
# For train data

tfidf_model1 = TfidfVectorizer()
tfidf_model1.fit(X_train['preprocessed_titles'])
#we are converting a dictionary with word as a key, and the idf as a value
dictionary_title = dict(zip(tfidf_model1.get_feature_names(), list(tfidf_model1.idf_)))
tfidf_words_titles = set(tfidf_model1.get_feature_names())
```

In [29]:

```
# average Word2Vec using pretrained models
# compute average word2vec for each review.
tfidf_w2v_train_title = [] # the avg-w2v for each sentence/review is stored in this lis
t
for sentence_title in tqdm(X_train['preprocessed_titles']): # for each review/sentence
    vector3 = np.zeros(300) # as word vectors are of zero length
    #tf_idf_weight3=0; # num of words with a valid vector in the sentence/review
    for word3 in sentence_title.split(): # for each word in a review/sentence
        if (word3 in glove_words) and (word3 in tfidf_words_titles):
            vec4 = model[word3]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf3 = dictionary_title[word3]*(sentence_title.count(word3)/len(sentence
_title.split())) # getting the tfidf value for each word
            vector3 += (vec4 * tf_idf3) # calculating tfidf weighted w2v
            tf_idf_weight3 += tf_idf3
    if tf_idf_weight3 != 0:
        vector3 /= tf_idf_weight3
        tfidf_w2v_train_title.append(vector3)
print(len(tfidf_w2v_train_title))
print(len(tfidf_w2v_train_title[0]))
```

```
100%|
| 22445/22445 [00:01<00:
00, 20790.21it/s]

22445
300
```

In [31]:

```
# For cv data
tfidf_w2v_cv_title = [] # the avg-w2v for each sentence/review is stored in this list
for sentence_cv in tqdm(X_cv['preprocessed_titles']): # for each review/sentence
    vector4 = np.zeros(300) # as word vectors are of zero length
    #tf_idf_weight4 =0; # num of words with a valid vector in the sentence/review
    for word4 in sentence_cv.split(): # for each word in a review/sentence
        if (word4 in glove_words) and (word4 in tfidf_words_titles):
            vec5 = model[word4]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf4 = dictionary_title[word4]*(sentence_cv.count(word4)/len(sentence_cv
.split())) # getting the tfidf value for each word
            vector4 += (vec5 * tf_idf4) # calculating tfidf weighted w2v
            tf_idf_weight4 += tf_idf4
    if tf_idf_weight4 != 0:
        vector4 /= tf_idf_weight4
        tfidf_w2v_cv_title.append(vector4)
print(len(tfidf_w2v_cv_title))
print(len(tfidf_w2v_cv_title[0]))
```

```
100%|
| 11055/11055 [00:00<00:
00, 20264.45it/s]

11055
300
```

In [33]:

```
# For test data
tfidf_w2v_test_title = [] # the avg-w2v for each sentence/review is stored in this list
for sentence_test in tqdm(X_test['preprocessed_titles']): # for each review/sentence
    vector5 = np.zeros(300) # as word vectors are of zero length
    #f_idf_weight5 =0; # num of words with a valid vector in the sentence/review
    for word5 in sentence_test.split(): # for each word in a review/sentence
        if (word5 in glove_words) and (word5 in tfidf_words_titles):
            vec6 = model[word5]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf5 = dictionary_title[word5]*(sentence_test.count(word5)/len(sentence_
test.split())) # getting the tfidf value for each word
            vector5 += (vec6 * tf_idf5) # calculating tfidf weighted w2v
            tf_idf_weight5 += tf_idf5
    if tf_idf_weight5 != 0:
        vector5 /= tf_idf_weight5
        tfidf_w2v_test_title.append(vector5)
print(len(tfidf_w2v_test_title))
print(len(tfidf_w2v_test_title[0]))
```

```
100%|████████████████████████████████████████| 16500/16500 [00:00<00:
00, 22705.71it/s]

16500
300
```

1.4 Make Data Model Ready: encoding numerical, categorical features

1.4.1 Encoding categorical features: School State

In [34]:

```
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 = vectorizer.transform(X_train['school_state'].values)
X_cv_state = vectorizer.transform(X_cv['school_state'].values)
X_test_state = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state.shape, y_train.shape)
print(X_cv_state.shape, y_cv.shape)
print(X_test_state.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi',
'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'm
o', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'o
k', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'w
i', 'wv', 'wy']
==========================================================================
==========================
```

1.4.2 Encoding categorical features: teacher_prefix

In [35]:

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

Out[35]:

```
Mrs.       11837
Ms.         7990
Mr.         2144
Teacher      472
none           1
Dr.            1
Name: teacher_prefix, dtype: int64
```

In [36]:

```
X_cv['teacher_prefix'].value_counts()
```

Out[36]:

```
Mrs.       5691
Ms.        4059
Mr.        1059
Teacher     245
Dr.           1
Name: teacher_prefix, dtype: int64
```

In [37]:

```
X_test['teacher_prefix'].value_counts()
```

Out[37]:

```
Mrs.       8612
Ms.        5887
Mr.        1656
Teacher     344
none          1
Name: teacher_prefix, dtype: int64
```

In [38]:

```
# Data imputation by replacing none values with highest mode category
#https://www.geeksforgeeks.org/python-pandas-dataframe-fillna-to-replace-null-values-in
-dataframe/

X_train['teacher_prefix'].fillna("Mrs.", inplace = True)
X_train['teacher_prefix'].value_counts()
```

Out[38]:

```
Mrs.       11837
Ms.         7990
Mr.         2144
Teacher      472
none           1
Dr.            1
Name: teacher_prefix, dtype: int64
```

In [39]:

```
X_test['teacher_prefix'].fillna("Mrs.", inplace = True)
X_test['teacher_prefix'].value_counts()
```

Out[39]:

```
Mrs.       8612
Ms.        5887
Mr.        1656
Teacher     344
none          1
Name: teacher_prefix, dtype: int64
```

In [40]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values)

X_train_teacher = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher.shape, y_train.shape)
print(X_cv_teacher.shape, y_cv.shape)
print(X_test_teacher.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 6) (22445,)
(11055, 6) (11055,)
(16500, 6) (16500,)
['dr', 'mr', 'mrs', 'ms', 'none', 'teacher']
==========================================================================
==========================
```

1.4.3 Encoding categorical features: project_grade_category

In [131]:

```
#This step is to intialize a vectorizer with vocab from train data
#Ref: https://www.kaggle.com/shashank49/donors-choose-knn#Concatinating-all-features-(T
FIDF)
from collections import Counter
my_counter = Counter()
for word in X_train['project_grade_category'].values:
    my_counter.update([word[i:i+14] for i in range(0, len(word),14)]) #https://www.geek
sforgeeks.org/python-string-split/

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
project_grade_category_dict = dict(my_counter)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), k
ey=lambda kv: kv[1]))
```

In [132]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys
()), lowercase=False, binary=True,max_features=4)
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on tr
ain data

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

print("After vectorizations")
print(X_train_grade.shape, y_train.shape)
print(X_cv_grade.shape, y_cv.shape)
print(X_test_grade.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['Grades 9-12', 'Grades 6-8', 'Grades 3-5', 'Grades PreK-2']
```

1.4.4 Encoding categorical features: clean_categories

In [43]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_cat = vectorizer.transform(X_train['clean_categories'].values)
X_cv_cat = vectorizer.transform(X_cv['clean_categories'].values)
X_test_cat = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_cat.shape, y_train.shape)
print(X_cv_cat.shape, y_cv.shape)
print(X_test_cat.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'lit
eracy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
===========================================================================
=========================
```

1.4.5 Encoding categorical features: clean_subcategories

In [44]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcat = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_subcat = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcat = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subcat.shape, y_train.shape)
print(X_cv_subcat.shape, y_cv.shape)
print(X_test_subcat.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_governmen
t', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economi
cs', 'environmentalscience', 'esl', 'extracurricular', 'financialliterac
y', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_welln
ess', 'history_geography', 'literacy', 'literature_writing', 'mathematic
s', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performi
ngarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'wa
rmth']
==========================================================================
==========================
```

1.4.6 Encoding numerical features: Price

In [45]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
#this will rise an error Expected 2D array, got 1D array instead:
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
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
==========================================================================
==========================
```

1.4.7 Encoding numerical features: Quantity

In [46]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
#this will rise an error Expected 2D array, got 1D array instead:
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
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
===========================================================================
=========================
```

1.4.8 Encoding numerical features: teacher_number_of_previously_posted_projects

In [47]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
#this will rise an error Expected 2D array, got 1D array instead:
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-
1,1))

X_train_projects_norm = normalizer.transform(X_train['teacher_number_of_previously_post
ed_projects'].values.reshape(-1,1))
X_cv_projects_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_pro
jects'].values.reshape(-1,1))
X_test_projects_norm = normalizer.transform(X_test['teacher_number_of_previously_posted
_projects'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_projects_norm.shape, y_train.shape)
print(X_cv_projects_norm.shape, y_cv.shape)
print(X_test_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
==========================================================================
==========================
```

1.4.5 Concatinating all the features

1.4.5.1 Set 1: Using categorical features + numerical features + preprocessed_titles(BOW) + preprocessed_essays(BOW)

In [133]:

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

X_tr_bow = hstack((X_train_essay_bow, X_train_title_bow, X_train_state, X_train_teacher
, X_train_grade, X_train_cat, X_train_subcat, X_train_price_norm, X_train_quantity_norm
, X_train_projects_norm )).tocsr()

X_cv_bow = hstack((X_cv_essay_bow, X_cv_title_bow, X_cv_state, X_cv_teacher, X_cv_grade
, X_cv_cat, X_cv_subcat, X_cv_price_norm, X_cv_quantity_norm, X_cv_projects_norm )).toc
sr()

X_test_bow = hstack((X_test_essay_bow, X_test_title_bow, X_test_state, X_test_teacher,
X_test_grade, X_test_cat, X_test_subcat, X_test_price_norm, X_test_quantity_norm, X_tes
t_projects_norm )).tocsr()

print("Final Data Matrix")
print(X_tr_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_train.shape)
print(X_test_bow.shape, y_train.shape)
```

```
Final Data Matrix
(22445, 7087) (22445,)
(11055, 7087) (22445,)
(16500, 7087) (22445,)
```

1.4.5.2 Set 2: Using categorical features + numerical features + preprocessed_titles(TFIDF) + preprocessed_essays(TFIDF)

In [135]:

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

X_tr_tfidf = hstack((X_train_essay_tfidf, X_train_titles_tfidf, X_train_state, X_train_
teacher, X_train_grade, X_train_cat, X_train_subcat, X_train_price_norm, X_train_quanti
ty_norm, X_train_projects_norm )).tocsr()

X_cv_tfidf = hstack((X_cv_essay_tfidf, X_cv_titles_tfidf, X_cv_state, X_cv_teacher, X_c
v_grade, X_cv_cat, X_cv_subcat, X_cv_price_norm, X_cv_quantity_norm, X_cv_projects_norm
)).tocsr()

X_test_tfidf = hstack((X_test_essay_tfidf, X_test_titles_tfidf, X_test_state, X_test_te
acher, X_test_grade, X_test_cat, X_test_subcat, X_test_price_norm, X_test_quantity_norm
, X_test_projects_norm )).tocsr()

print("Final Data Matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_cv_tfidf.shape, y_train.shape)
print(X_test_tfidf.shape, y_train.shape)
```

```
Final Data Matrix
(22445, 7087) (22445,)
(11055, 7087) (22445,)
(16500, 7087) (22445,)
```

1.4.5.3 Set 3: Using categorical features + numerical features + preprocessed_titles(Avg W2V) + preprocessed_essays(Avg W2V)

In [136]:

```
from scipy.sparse import hstack

X_tr_avgw2v = hstack((sent_vectors_train, avg_w2v_essay_train, X_train_state, X_train_t
eacher, X_train_grade, X_train_cat, X_train_subcat, X_train_price_norm, X_train_quantit
y_norm, X_train_projects_norm )).tocsr()

X_cv_avgw2v = hstack((sent_vectors_cv, avg_w2v_essay_cv, X_cv_state, X_cv_teacher, X_cv
_grade, X_cv_cat, X_cv_subcat, X_cv_price_norm, X_cv_quantity_norm, X_cv_projects_norm
)).tocsr()

X_test_avgw2v = hstack((sent_vectors_test, avg_w2v_essay_test, X_test_state, X_test_tea
cher, X_test_grade, X_test_cat, X_test_subcat, X_test_price_norm, X_test_quantity_norm,
X_test_projects_norm )).tocsr()

print("Final Data Matrix")
print(X_tr_avgw2v.shape, y_train.shape)
print(X_cv_avgw2v.shape, y_train.shape)
print(X_test_avgw2v.shape, y_train.shape)
```

```
Final Data Matrix
(22445, 453) (22445,)
(11055, 453) (22445,)
(16500, 453) (22445,)
```

1.4.5.4 Set 4: Using categorical features + numerical features + preprocessed_titles(TFIDF W2V) + preprocessed_essays(TFIDF W2V)

In [137]:

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

X_tr_tfidf_w2v = hstack((tfidf_w2v_train_essay, tfidf_w2v_train_title, X_train_state, X
_train_teacher, X_train_grade, X_train_cat, X_train_subcat, X_train_price_norm, X_train
_quantity_norm, X_train_projects_norm )).tocsr()

X_cv_tfidf_w2v = hstack((tfidf_w2v_cv_essay, tfidf_w2v_cv_title, X_cv_state, X_cv_teach
er, X_cv_grade, X_cv_cat, X_cv_subcat, X_cv_price_norm, X_cv_quantity_norm, X_cv_projec
ts_norm )).tocsr()

X_test_tfidf_w2v = hstack((tfidf_w2v_test_essay, tfidf_w2v_test_title, X_test_state, X_
test_teacher, X_test_grade, X_test_cat, X_test_subcat, X_test_price_norm, X_test_quanti
ty_norm, X_test_projects_norm )).tocsr()

print("Final Data Matrix")
print(X_tr_tfidf_w2v.shape, y_train.shape)
print(X_cv_tfidf_w2v.shape, y_train.shape)
print(X_test_tfidf_w2v.shape, y_train.shape)
```

```
Final Data Matrix
(22445, 703) (22445,)
(11055, 703) (22445,)
(16500, 703) (22445,)
```

1.5 Appling KNN

1.5.1 Appling KNN: BOW featurization

1.5.1.1 Hyper parameter Tuning

In [52]:

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

    return y_data_pred
```

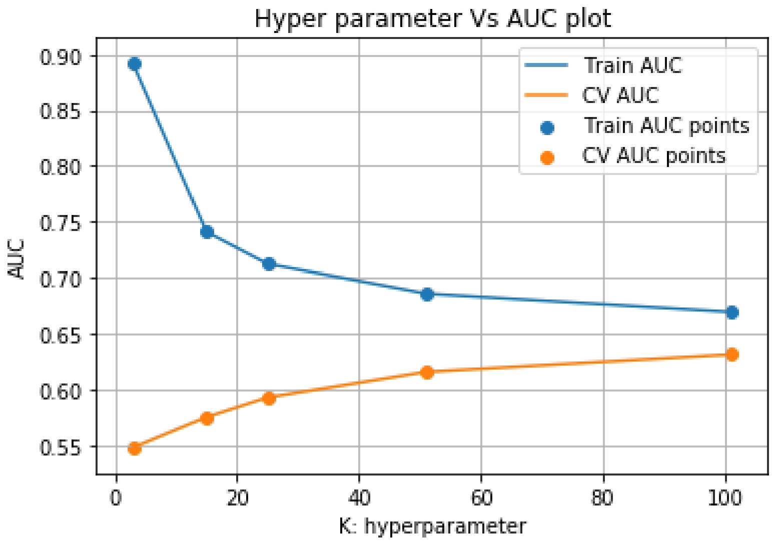
In [53]:

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

#y_true : array, shape = [n_samples] or [n_samples, n_classes]
#True binary labels or binary label indicators.

#y_score : array, shape = [n_samples] or [n_samples, n_classes]

#Target scores, can either be probability estimates of the positive class, confidence v
alues, or non-thresholded measure of decisions

#For binary y_true, y_score is supposed to be the score of the class with greater label
```

In [54]:

```
# Simple CV using for loops.
train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr_bow, y_train)
    y_train_pred = batch_predict(neigh, X_tr_bow)
    y_cv_pred = batch_predict(neigh, X_cv_bow)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
100%|██████████████████████████████| 5/5 [12:18<00:00, 147.66s/it]
```

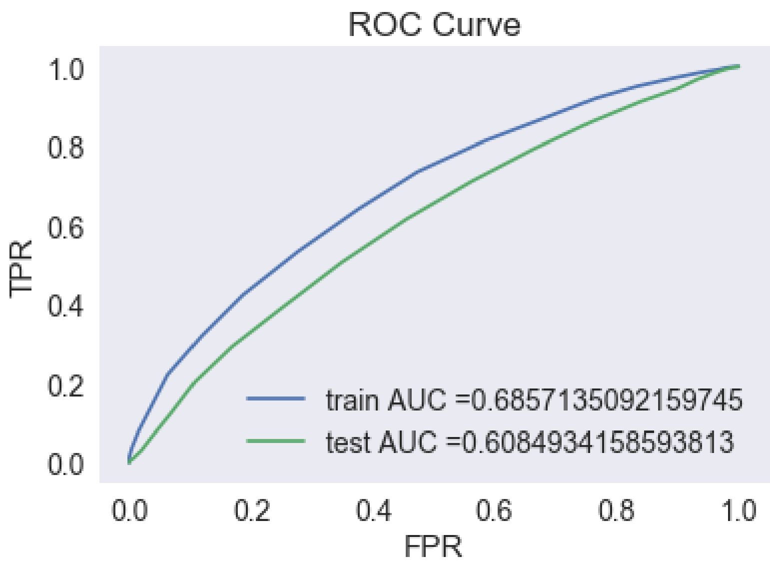


From the Hyper parameter Vs AUC plot it can be observed that the inflexion point of both the curves is between 50-60. Hence the optimum K value that I will be choosing is "51".

1.5.1.2 Testing the performance of the model on test data & plotting ROC Curves for train & test data

In [147]:

```
best_k = 51
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_tr_bow, y_train)
y_train_pred_bow = batch_predict(neigh, X_tr_bow)
y_test_pred_bow = batch_predict(neigh, X_test_bow)

train_tpr, train_fpr, tr_thresholds = roc_curve(y_train, y_train_pred_bow)
test_tpr, test_fpr, te_thresholds = roc_curve(y_test, y_test_pred_bow)

plt.plot(train_tpr, train_fpr,label="train AUC ="+str(auc(train_tpr, train_fpr)))
plt.plot(test_tpr, test_fpr, label="test AUC ="+str(auc(test_tpr, test_fpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



In [63]:

```
## we will pick a threshold that will give the least fpr

def find_best_threshold(threshold, fpr, tpr):
    t = threshold[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.roun
d(t,3))
    return t

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

print("="*100)
```

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

In [68]:

```
#function to get heatmap of confusion matrix
# Reference: https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-ma
trix

def cm_heatmap(cm):
    #y_pred = clf.predict(X_te)
    df_cm = pd.DataFrame(cm, range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

1.5.1.3 Confusion matrices

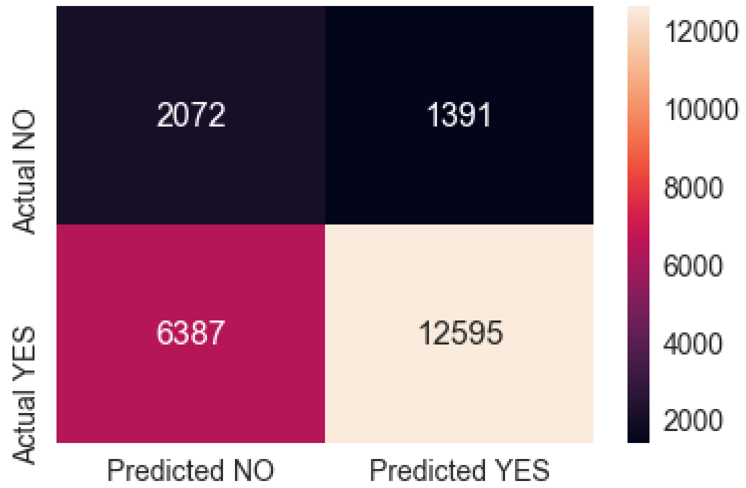
In [66]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
cm_train=confusion_matrix(y_train, predict_with_best_t(y_train_pred_bow, best_t))
print(cm_train)
print("Test confusion matrix")
cm_test=confusion_matrix(y_test, predict_with_best_t(y_test_pred_bow, best_t))
print(cm_test)
```

```
The maximum value of tpr*(1-fpr) 0.1351542122386383 for threshold 0.8
Train confusion matrix
[[ 2072  1391]
 [ 6387 12595]]
Test confusion matrix
[[1324 1222]
 [5059 8895]]
```

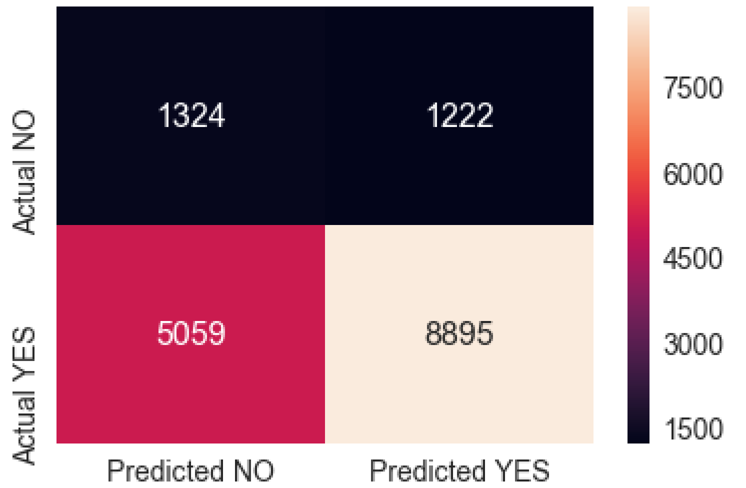
In [69]:

```
# confusion matrix heatmap for train data
cm_heatmap(cm_train)
```



In [70]:

```
# confusion matrix heatmap for test data
cm_heatmap(cm_test)
```



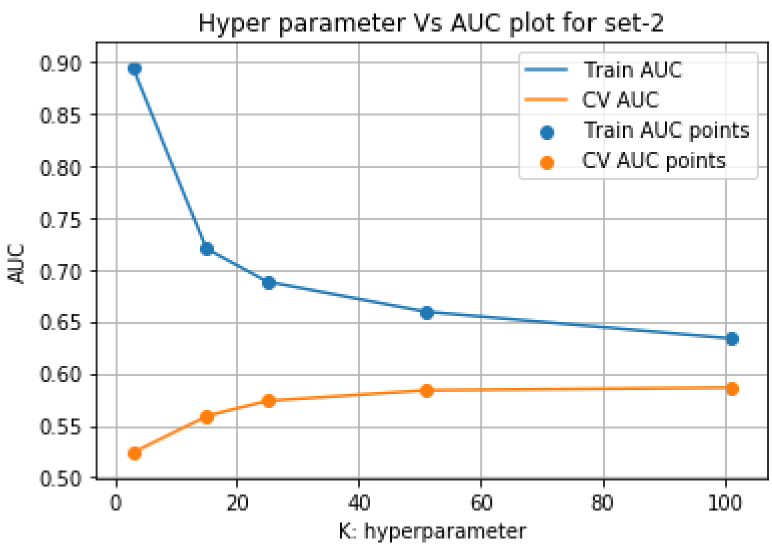
1.5.2 Appling KNN: TFIDF featurization

1.5.2.1 Hyper parameter Tuning

In [58]:

```
# Hyper parameter tunning for TFIDF

train_auc_tfidf = []
cv_auc_tfidf = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr_tfidf, y_train)
    y_train_pred_2 = batch_predict(neigh, X_tr_tfidf)
    y_cv_pred_2 = batch_predict(neigh, X_cv_tfidf)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    train_auc_tfidf.append(roc_auc_score(y_train,y_train_pred_2))
    cv_auc_tfidf.append(roc_auc_score(y_cv, y_cv_pred_2))

plt.plot(K, train_auc_tfidf, label='Train AUC')
plt.plot(K, cv_auc_tfidf, label='CV AUC')

plt.scatter(K, train_auc_tfidf, label='Train AUC points')
plt.scatter(K, cv_auc_tfidf, label='CV AUC points')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot for set-2")
plt.grid()
plt.show()
```

```
  0%|
| 0/5 [00:00<?, ?it/s]
 20%|
| 1/5 [02:47<11:08, 167.21s/it]
 40%|
| 2/5 [05:22<08:10, 163.49s/it]
 60%|
                                                               | 3/5 [07:57<0
5:22, 161.09s/it]
 80%|
                                                               | 4/5 [10:32<0
2:39, 159.27s/it]
100%|
                                                              | 5/5 [13:08<0
0:00, 158.33s/it]
```

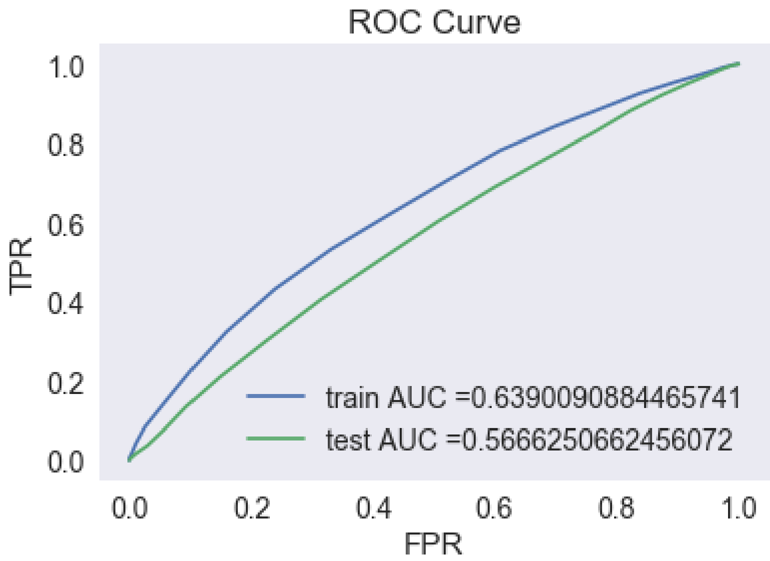


From the Hyper parameter Vs AUC plot it can be observed that the inflexion point of both the curves is between 80-100. Hence the optimum K value that I will be choosing is "81".

1.5.2.2 Testing the performance of the model on test data & plotting ROC Curves for train & test data

In [71]:

```
best_k = 81
neigh1 = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh1.fit(X_tr_tfidf, y_train)
y_train_pred_tfidf = batch_predict(neigh1, X_tr_tfidf)
y_test_pred_tfidf = batch_predict(neigh1, X_test_tfidf)

train_tpr_1, train_fpr_1, tr_thresholds_1 = roc_curve(y_train, y_train_pred_tfidf)
test_tpr_1, test_fpr_1, te_thresholds_1 = roc_curve(y_test, y_test_pred_tfidf)

plt.plot(train_tpr_1, train_fpr_1 , label="train AUC ="+str(auc(train_tpr_1, train_fpr_
1)))
plt.plot(test_tpr_1, test_fpr_1 , label="test AUC ="+str(auc(test_tpr_1, test_fpr_1)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



1.5.2.3 Confusion matrices

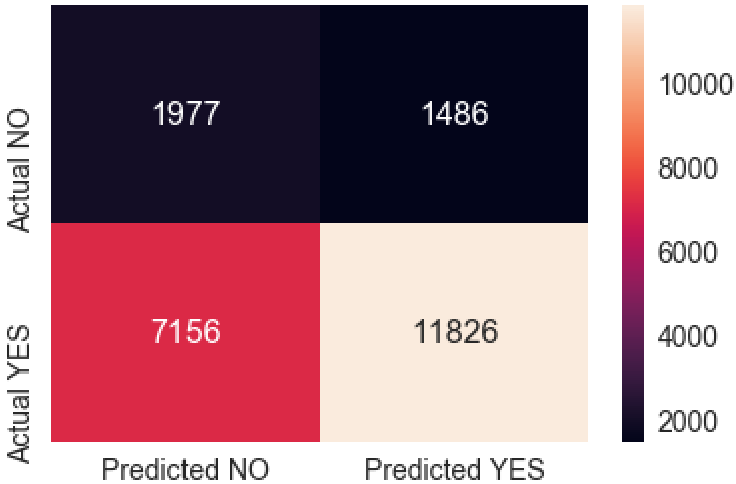
In [72]:

```
#confusion matrices

from sklearn.metrics import confusion_matrix
best_t_tfidf = find_best_threshold(tr_thresholds_1, train_fpr_1, train_tpr_1)
print("Train confusion matrix")
cm_train_tfidf=confusion_matrix(y_train, predict_with_best_t(y_train_pred_tfidf, best_t
_tfidf))
print(cm_train_tfidf)
print("Test confusion matrix")
cm_test_tfidf=confusion_matrix(y_test, predict_with_best_t(y_test_pred_tfidf, best_t_tf
idf))
print(cm_test_tfidf)
```

```
The maximum value of tpr*(1-fpr) 0.1617687690084255 for threshold 0.85
Train confusion matrix
[[ 1977  1486]
 [ 7156 11826]]
Test confusion matrix
[[1255 1291]
 [5532 8422]]
```

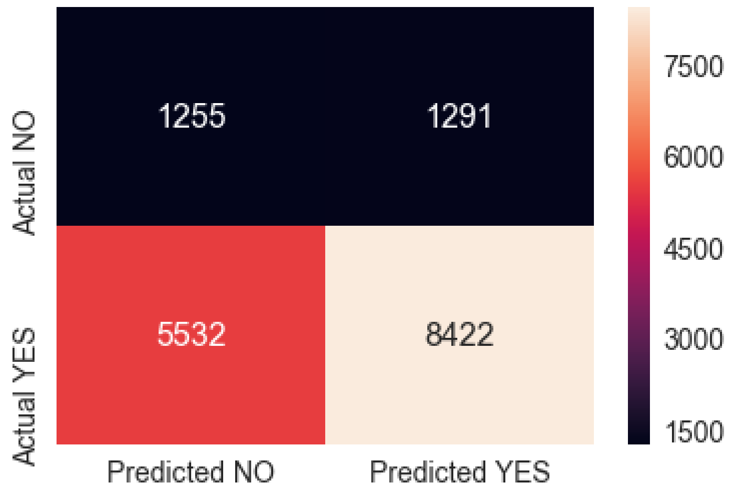
In [73]:

```
# confusion matrix heatmap for train data
cm_heatmap(cm_train_tfidf)
```



In [74]:

```
# confusion matrix heatmap for test data
cm_heatmap(cm_test_tfidf)
```



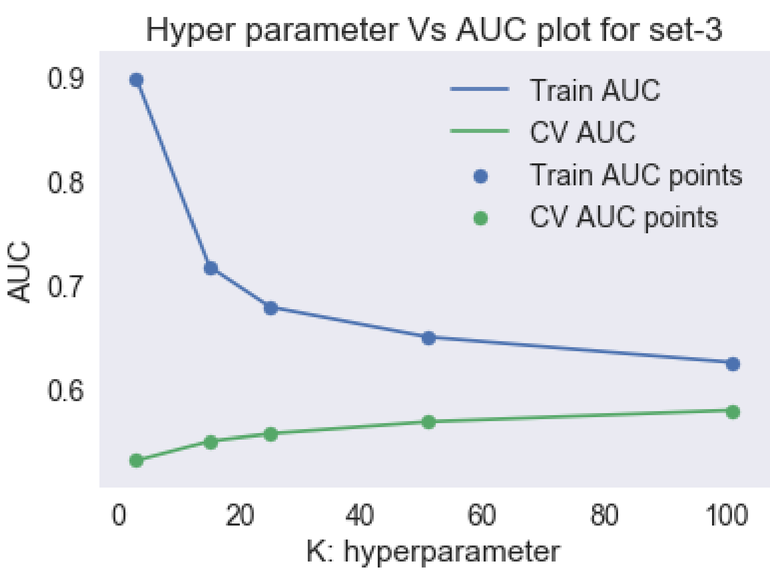
1.5.3 Appling KNN: Avg w2v featurization

1.5.3.1 Hyper parameter Tuning

In [78]:

```
# Hyper parameter tunning for Avg W2V

train_auc_avg = []
cv_auc_avg = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh2 = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh2.fit(X_tr_avgw2v, y_train)
    y_train_pred_3 = batch_predict(neigh2, X_tr_avgw2v)
    y_cv_pred_3 = batch_predict(neigh2, X_cv_avgw2v)
    train_auc_avg.append(roc_auc_score(y_train,y_train_pred_3))
    cv_auc_avg.append(roc_auc_score(y_cv, y_cv_pred_3))

plt.plot(K, train_auc_avg, label='Train AUC')
plt.plot(K, cv_auc_avg, label='CV AUC')

plt.scatter(K, train_auc_avg, label='Train AUC points')
plt.scatter(K, cv_auc_avg, label='CV AUC points')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot for set-3")
plt.grid()
plt.show()
```

```
  0%|
| 0/5 [00:00<?, ?it/s]
 20%|████████████████
| 1/5 [07:50<31:22, 470.66s/it]
 40%|████████████████████████████████
| 2/5 [18:23<25:58, 519.38s/it]
 60%|████████████████████████████████████████████
█████                                      | 3/5 [28:58<1
8:27, 553.96s/it]
 80%|████████████████████████████████████████████
██████████████████████                     | 4/5 [39:12<0
9:32, 572.01s/it]
100%|████████████████████████████████████████████
██████████████████████████████████████| 5/5 [47:04<0
0:00, 542.03s/it]
```

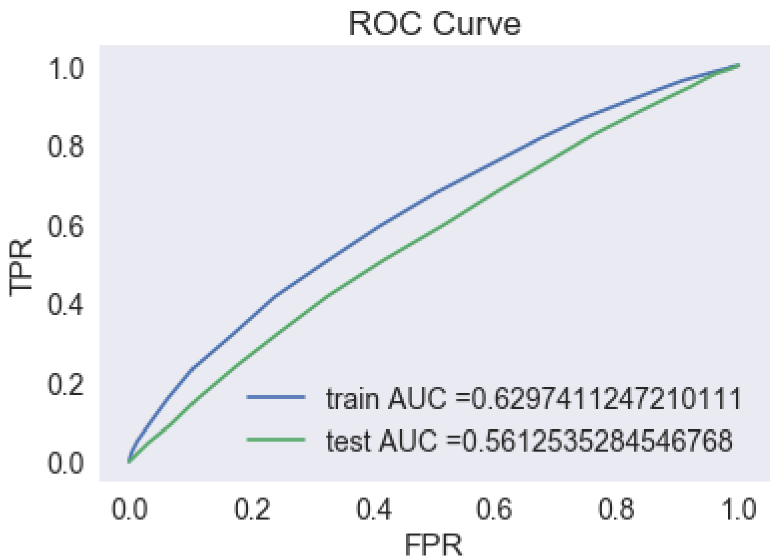


From the Hyper parameter Vs AUC plot it can be observed that the inflexion point of both the curves is between 80-100. Hence the optimum K value that I will be choosing is "91".

1.5.3.2 Testing the performance of the model on test data & plotting ROC Curves for train & test data

In [79]:

```
best_k = 91
neigh3 = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh3.fit(X_tr_avgw2v, y_train)
y_train_pred_avg = batch_predict(neigh3, X_tr_avgw2v)
y_test_pred_avg = batch_predict(neigh3, X_test_avgw2v)

train_tpr_2, train_fpr_2, tr_thresholds_2 = roc_curve(y_train, y_train_pred_avg)
test_tpr_2, test_fpr_2, te_thresholds_2 = roc_curve(y_test, y_test_pred_avg)

plt.plot(train_tpr_2, train_fpr_2 , label="train AUC ="+str(auc(train_tpr_2, train_fpr_
2)))
plt.plot(test_tpr_2, test_fpr_2 , label="test AUC ="+str(auc(test_tpr_2, test_fpr_2)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



1.5.3.3 Confusion matrices

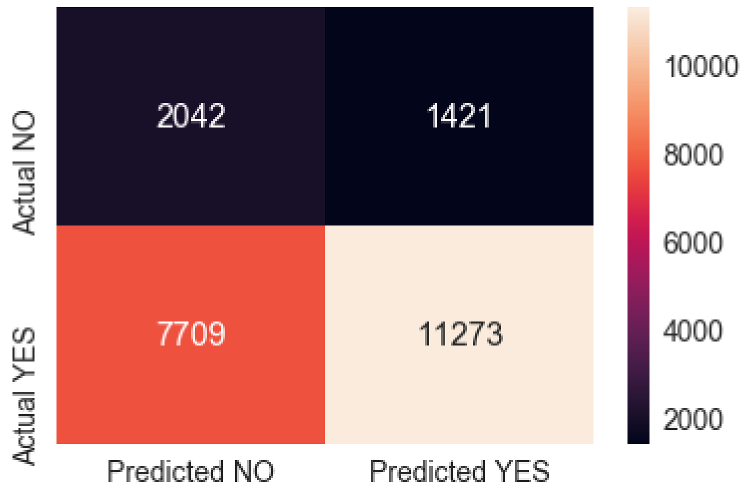
In [80]:

```
#confusion matrices

from sklearn.metrics import confusion_matrix
best_t_avg = find_best_threshold(tr_thresholds_2, train_fpr_2, train_tpr_2)
print("Train confusion matrix")
cm_train_avg=confusion_matrix(y_train, predict_with_best_t(y_train_pred_avg, best_t_avg
))
print(cm_train_avg)
print("Test confusion matrix")
cm_test_avg=confusion_matrix(y_test, predict_with_best_t(y_test_pred_avg, best_t_avg))
print(cm_test_avg)
```

```
The maximum value of tpr*(1-fpr) 0.16664706260164158 for threshold 0.856
Train confusion matrix
[[ 2042  1421]
 [ 7709 11273]]
Test confusion matrix
[[1228 1318]
 [5601 8353]]
```

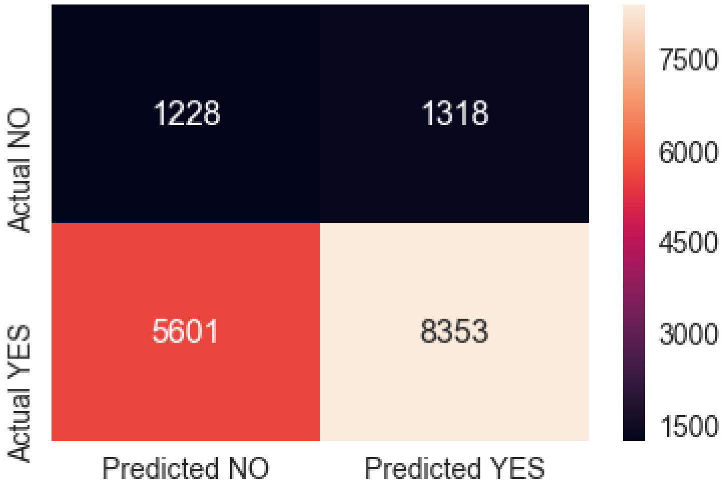
In [81]:

```
# confusion matrix heatmap for train data
cm_heatmap(cm_train_avg)
```



In [82]:

```
# confusion matrix heatmap for test data
cm_heatmap(cm_test_avg)
```



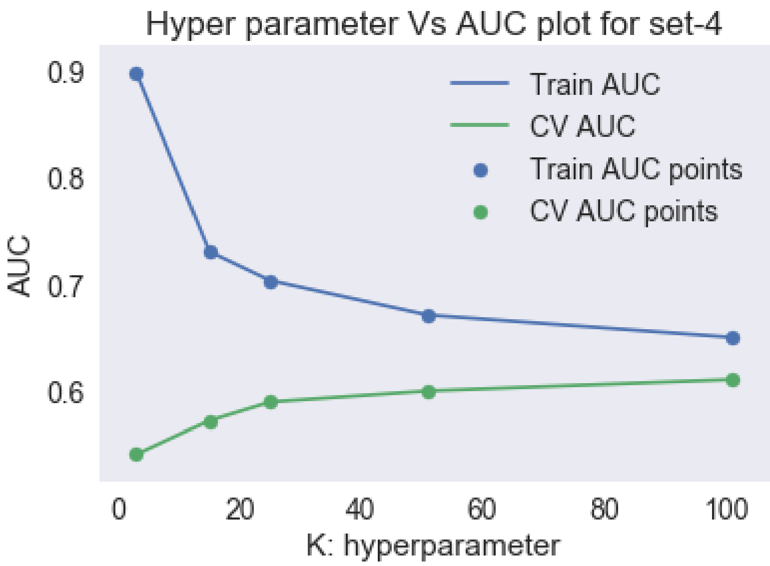
1.5.4 Appling KNN: TFIDF weighted w2v featurization

1.5.4.1 Hyper parameter Tuning

In [87]:

```
# Hyper parameter tunning for tfidf weighted W2V

train_auc_wt = []
cv_auc_wt = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh4 = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh4.fit(X_tr_tfidf_w2v, y_train)
    y_train_pred_4 = batch_predict(neigh4, X_tr_tfidf_w2v)
    y_cv_pred_4 = batch_predict(neigh4, X_cv_tfidf_w2v)
    train_auc_wt.append(roc_auc_score(y_train,y_train_pred_4))
    cv_auc_wt.append(roc_auc_score(y_cv, y_cv_pred_4))

plt.plot(K, train_auc_wt, label='Train AUC')
plt.plot(K, cv_auc_wt, label='CV AUC')

plt.scatter(K, train_auc_wt, label='Train AUC points')
plt.scatter(K, cv_auc_wt, label='CV AUC points')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot for set-4")
plt.grid()
plt.show()
```

```
  0%|
| 0/5 [00:00<?, ?it/s]

 20%|█████████████████▌
| 1/5 [12:24<49:39, 744.94s/it]

 40%|███████████████████████████████████
| 2/5 [24:53<37:17, 745.90s/it]

 60%|███████████████████████████████████████████████
██████▌                                          | 3/5 [37:18<2
4:51, 745.78s/it]

 80%|███████████████████████████████████████████████
███████████████████████▏                         | 4/5 [49:43<1
2:25, 745.53s/it]

100%|███████████████████████████████████████████████
████████████████████████████████████████| 5/5 [1:02:03<0
0:00, 743.74s/it]
```

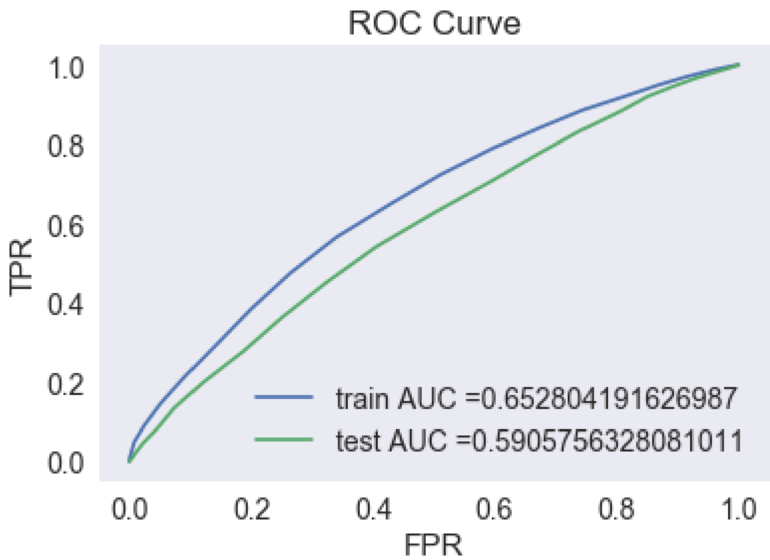


From the Hyper parameter Vs AUC plot it can be observed that the inflexion point of both the curves is between 80-100. Hence the optimum K value that I will be choosing is "91".

1.5.4.2 Testing the performance of the model on test data & plotting ROC Curves for train & test data

In [88]:

```
best_k = 91
neigh5 = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh5.fit(X_tr_tfidf_w2v, y_train)
y_train_pred_wt = batch_predict(neigh5, X_tr_tfidf_w2v)
y_test_pred_wt = batch_predict(neigh5, X_test_tfidf_w2v)

train_tpr_3, train_fpr_3, tr_thresholds_3 = roc_curve(y_train, y_train_pred_wt)
test_tpr_3, test_fpr_3, te_thresholds_3 = roc_curve(y_test, y_test_pred_wt)

plt.plot(train_tpr_3, train_fpr_3 , label="train AUC ="+str(auc(train_tpr_3, train_fpr_
3)))
plt.plot(test_tpr_3, test_fpr_3 , label="test AUC ="+str(auc(test_tpr_3, test_fpr_3)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



1.5.4.3 Confusion matrices

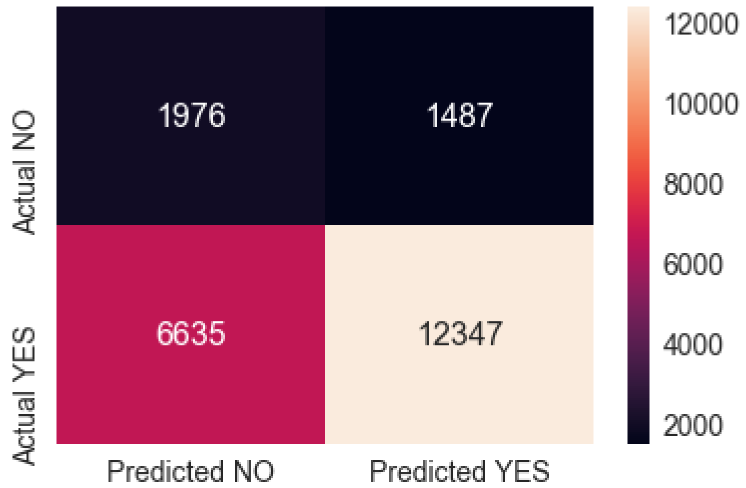
In [89]:

```
#confusion matrices

from sklearn.metrics import confusion_matrix
best_t_wt = find_best_threshold(tr_thresholds_3, train_fpr_3, train_tpr_3)
print("Train confusion matrix")
cm_train_wt=confusion_matrix(y_train, predict_with_best_t(y_train_pred_wt, best_t_wt))
print(cm_train_wt)
print("Test confusion matrix")
cm_test_wt=confusion_matrix(y_test, predict_with_best_t(y_test_pred_wt, best_t_wt))
print(cm_test_wt)
```

```
The maximum value of tpr*(1-fpr) 0.15009196213151826 for threshold 0.844
Train confusion matrix
[[ 1976  1487]
 [ 6635 12347]]
Test confusion matrix
[[1263 1283]
 [5156 8798]]
```

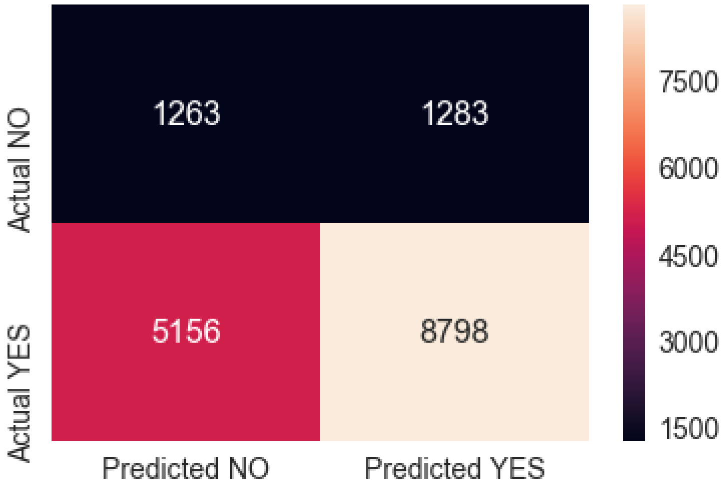
In [90]:

```
# confusion matrix heatmap for train data
cm_heatmap(cm_train_wt)
```



In [91]:

```
# confusion matrix heatmap for test data
cm_heatmap(cm_test_wt)
```



1.6 Feature selection & applying KNN

1.6.1 Selecting top 2K features from Set -2 using SelectKBest

In [151]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBes
t.html

from sklearn.feature_selection import SelectKBest, chi2
print("Original Shapes of set-2 before feature selection")
print("Train data: ",X_tr_tfidf.shape,y_train.shape)
print("CV data: ",X_cv_tfidf.shape,y_cv.shape)
print("Test data: ",X_test_tfidf.shape,y_test.shape)
print("="*50)
fit_trian= SelectKBest(chi2, k=2000).fit(X_tr_tfidf, y_train)
X_train_2k=fit_trian.transform(X_tr_tfidf)
X_cv_2k = fit_trian.transform(X_cv_tfidf)
X_test_2k = fit_trian.transform(X_test_tfidf)
print("Shapes after feature selection")
print("Train data: ",X_train_2k.shape,y_train.shape)
print("CV data: ",X_cv_2k.shape,y_cv.shape)
print("Test data: ",X_test_2k.shape,y_test.shape)
```

```
Original Shapes of set-2 before feature selection
Train data:  (22445, 7087) (22445,)
CV data:  (11055, 7087) (11055,)
Test data:  (16500, 7087) (16500,)
==================================================
Shapes after feature selection
Train data:  (22445, 2000) (22445,)
CV data:  (11055, 2000) (11055,)
Test data:  (16500, 2000) (16500,)
```

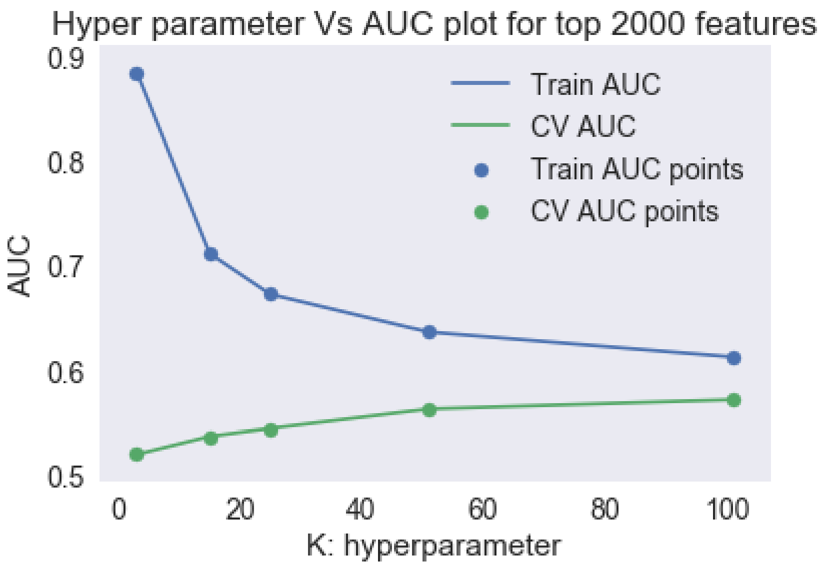
1.6.2 Apply KNN

1.6.2.1 Hyperparameter tuning

In [152]:

```
train_auc_2k = []
cv_auc_2k = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh5 = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh5.fit(X_train_2k, y_train)
    y_train_pred_5 = batch_predict(neigh5, X_train_2k)
    y_cv_pred_5 = batch_predict(neigh5, X_cv_2k)
    train_auc_2k.append(roc_auc_score(y_train,y_train_pred_5))
    cv_auc_2k.append(roc_auc_score(y_cv, y_cv_pred_5))

plt.plot(K, train_auc_2k, label='Train AUC')
plt.plot(K, cv_auc_2k, label='CV AUC')

plt.scatter(K, train_auc_2k, label='Train AUC points')
plt.scatter(K, cv_auc_2k, label='CV AUC points')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot for top 2000 features")
plt.grid()
plt.show()
```

```
  0%|
| 0/5 [00:00<?, ?it/s]

 20%|
| 1/5 [02:04<08:16, 124.10s/it]

 40%|
| 2/5 [04:09<06:13, 124.40s/it]

 60%|
                                                              | 3/5 [06:14<0
4:09, 124.65s/it]

 80%|
                                                              | 4/5 [08:21<0
2:05, 125.31s/it]

100%|
                                                             | 5/5 [10:40<0
0:00, 129.57s/it]
```

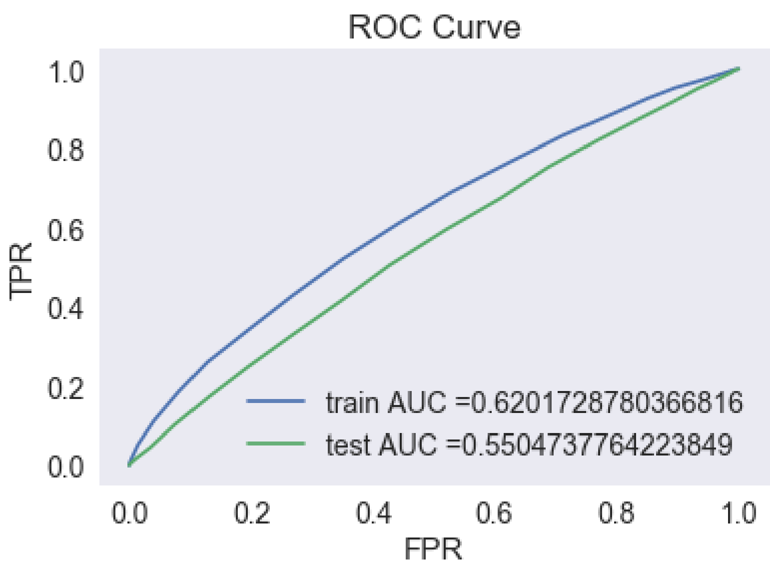


From the Hyper parameter Vs AUC plot it can be observed(not so convincingly though) that the inflexion point of both the curves is between 80-100. Hence the optimum K value that I will be choosing is "81".

1.6.2.2 Testing the performance of the model on test data & plotting ROC Curves for train & test data

In [153]:

```
best_k = 81
neigh6 = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh6.fit(X_train_2k, y_train)
y_train_pred_2k = batch_predict(neigh6, X_train_2k)
y_test_pred_2k = batch_predict(neigh6, X_test_2k)

train_tpr_4, train_fpr_4, tr_thresholds_4 = roc_curve(y_train, y_train_pred_2k)
test_tpr_4, test_fpr_4, te_thresholds_4 = roc_curve(y_test, y_test_pred_2k)

plt.plot(train_tpr_4, train_fpr_4 , label="train AUC ="+str(auc(train_tpr_4, train_fpr_
4)))
plt.plot(test_tpr_4, test_fpr_4 , label="test AUC ="+str(auc(test_tpr_4, test_fpr_4)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



1.6.2.3 Confusion matrices

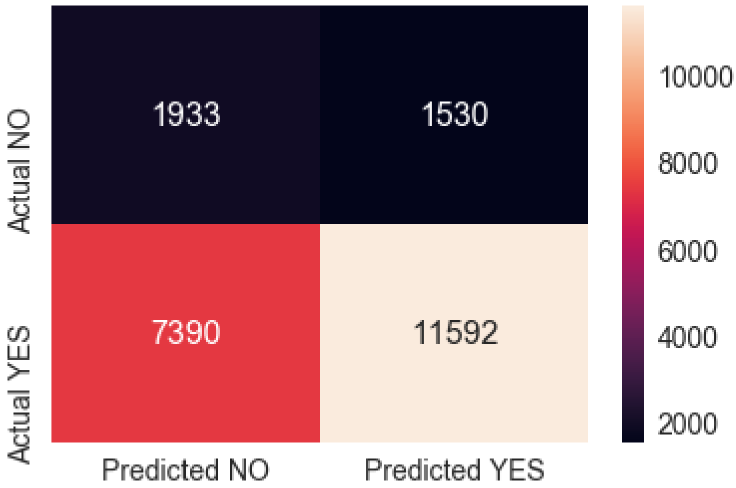
In [154]:

```
#confusion matrices

from sklearn.metrics import confusion_matrix
best_t_2k = find_best_threshold(tr_thresholds_4, train_fpr_4, train_tpr_4)
print("Train confusion matrix")
cm_train_2k=confusion_matrix(y_train, predict_with_best_t(y_train_pred_2k, best_t_2k))
print(cm_train_2k)
print("Test confusion matrix")
cm_test_2k=confusion_matrix(y_test, predict_with_best_t(y_test_pred_2k, best_t_2k))
print(cm_test_2k)
```

```
The maximum value of tpr*(1-fpr) 0.17200513348618823 for threshold 0.84
Train confusion matrix
[[ 1933  1530]
 [ 7390 11592]]
Test confusion matrix
[[1222 1324]
 [5650 8304]]
```

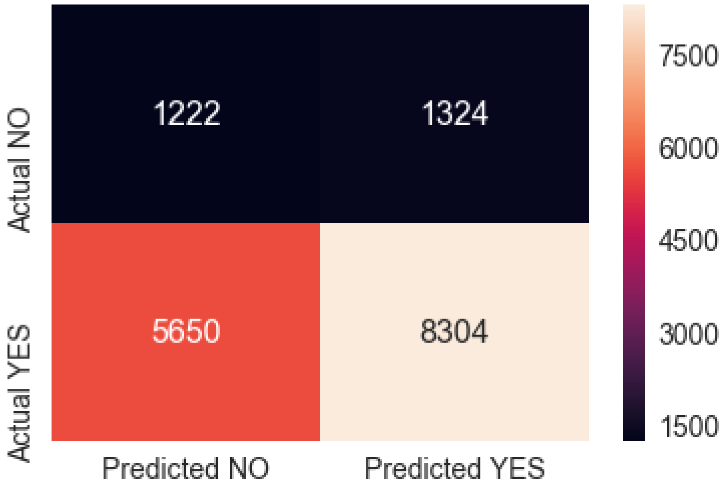
In [155]:

```
# confusion matrix heatmap for train data
cm_heatmap(cm_train_2k)
```



In [156]:

```
# confusion matrix heatmap for test data
cm_heatmap(cm_test_2k)
```



2.0 Summary

In [158]:

```
#Ref: http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameter", "Test AUC", "Train AUC"]
x.add_row(["BOW", "Brute", 51, 0.61, 0.68])
x.add_row(["TFIDF", "Brute", 81, 0.57, 0.64])
x.add_row(["Avg W2V", "Brute", 91, 0.56, 0.63])
x.add_row(["TFIDF W2V", "Brute", 91, 0.60, 0.65])
x.add_row(["TFIDF using top 2K selected features", "Brute", 81, 0.55, 0.62])

print(x)
```

```
+--------------------------------------+-------+----------------+---------
-+-----------+
|              Vectorizer              | Model | Hyperparameter | Test AUC
| Train AUC |
+--------------------------------------+-------+----------------+---------
-+-----------+
|                 BOW                  | Brute |       51       |   0.61
|    0.68   |
|                TFIDF                 | Brute |       81       |   0.57
|    0.64   |
|               Avg W2V                | Brute |       91       |   0.56
|    0.63   |
|              TFIDF W2V               | Brute |       91       |   0.6
|    0.65   |
| TFIDF using top 2K selected features | Brute |       81       |   0.55
|    0.62   |
+--------------------------------------+-------+----------------+---------
-+-----------+
```

- With reference to the above table, it can be observed that with the use of feature selection(i.e.using top 2K features) the model performance(test AUC score:55%) was almost in par with the performance of the model when all the 7K features were used(test AUC score:57%).
- Selecting the best features turned out to be fruitful as the computational time was reduced as a result of this.