2) K Nearest Neighbor

In [91]:

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

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

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

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
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 chart_studio.plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

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

we are going to consider

```
    - school_state : categorical data
    - clean_categories : categorical data
    - clean_subcategories : categorical data
    - project_grade_category : categorical data
    - teacher_prefix : categorical data

    - project_title : text data
    - text : text data
    - project_resource_summary: text data (optinal)

    - quantity : numerical (optinal)
    - teacher_number_of_previously_posted_projects : numerical
    - price : numerical
```

In [92]:

```
project_data = pd.read_csv("train_data.csv", nrows = 75000)
resource_data = pd.read_csv('resources.csv',nrows = 75000)
```

In [93]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [4]:

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

```
Number of data points in train data (75000, 19)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved'
 'price' 'quantity']
```

In [5]:

```
# Let's check for "missing" or "NaN" values in our dataset
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 75000 entries, 0 to 74999
Data columns (total 19 columns):
Unnamed: 0                                      75000 non-null int64
id                                              75000 non-null object
teacher_id                                      75000 non-null object
teacher_prefix                                  74997 non-null object
school_state                                    75000 non-null object
project_submitted_datetime                      75000 non-null object
project_grade_category                          75000 non-null object
project_subject_categories                      75000 non-null object
project_subject_subcategories                   75000 non-null object
project_title                                   75000 non-null object
project_essay_1                                 75000 non-null object
project_essay_2                                 75000 non-null object
project_essay_3                                 2558 non-null object
project_essay_4                                 2558 non-null object
project_resource_summary                        75000 non-null object
teacher_number_of_previously_posted_projects    75000 non-null int64
project_is_approved                             75000 non-null int64
price                                           3614 non-null float64
quantity                                        3614 non-null float64
dtypes: float64(2), int64(3), object(14)
memory usage: 11.4+ MB
```

It seems to be a missing values present for "teacher_prefix" feature. We'll replace the missing values with the mode of "project_prefix" columns itself.

In [6]:

```
project_data['teacher_prefix'].isna().sum()
```

Out[6]:

```
3
```

In [7]:

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('Mrs.')
```

Since, price feature column has too many "missing" values, I am selecting 75k data points and therefore price does not fit good feature because of so many missing values. Hence I am droping price column. Also 'project_resource_summary' is an optional, so dropping that as well.

In [8]:

```
# Let's select only the selected features or columns
#
project_data.drop(['id','teacher_id','project_submitted_datetime','price','project_resource_summary
','quantity'],axis=1, inplace=True)
project_data.columns
```

Out[8]:

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

Let's first merge all the project_essays into single columns

In [9]:

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

In [10]:

```
project_data.head(5)
```

Out[10]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
0	160221	Mrs.	IN	Grades PreK-2	Literacy & Language	ESL, Literacy
1	140945	Mr.	FL	Grades 6-8	History & Civics, Health & Sports	Civics & Government, Team Sports
2	21895	Ms.	AZ	Grades 6-8	Health & Sports	Health & Wellness, Team Sp
3	45	Mrs.	KY	Grades PreK-2	Literacy & Language, Math & Science	Literacy, Mathematics

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
4	172407	Mrs.	TX	Grades PreK-2	Math & Science	Mathematics

In [11]:

```
# Let's drop the project essay columns from the dadaset now, as we have captured the essay text da
ta into single "essay" column
project_data.drop(['project_essay_1','project_essay_2','project_essay_3','project_essay_4'],axis=1
, inplace=True)
```

In [12]:

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

Out[12]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
0	160221	Mrs.	IN	Grades PreK-2	Literacy & Language	ESL, Literacy

In [13]:

```
# 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, stratify=y_train)
```

2.2) Make Data Model Ready: encoding numerical, categorical features

In [14]:

```
def cleaning_text_data(list_text_feature,df,old_col_name,new_col_name):

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

    df[new_col_name] = feature_list
    df.drop([old_col_name], axis=1, inplace=True)
    from collections import Counter
    my_counter = Counter()
    for word in df[new_col_name].values:
        my_counter.update(word.split())

    feature_dict = dict(my_counter)
    sorted_feature_dict = dict(sorted(feature_dict.items(), key=lambda kv: kv[1]))
    return sorted_feature_dict
```

In [15]:

```
def clean_project_grade(list_text_feature,df,old_col_name,new_col_name):

    # remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
    # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
    # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
    # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
    feature_list = []
    for i in list_text_feature:
        temp = i.split(' ')
        last_dig = temp[-1].split('-')
        fin = [temp[0]]
        fin.extend(last_dig)
        feature = '_'.join(fin)
        feature_list.append(feature.strip())

    df[new_col_name] = feature_list
    df.drop([old_col_name], axis=1, inplace=True)

    from collections import Counter
    my_counter = Counter()
    for word in df[new_col_name].values:
        my_counter.update(word.split())

    feature_dict = dict(my_counter)
    sorted_feature_dict = dict(sorted(feature_dict.items(), key=lambda kv: kv[1]))
    return sorted_feature_dict
```

2.2.1) Text Preprocessing: project_subject_categories

In [16]:

```
x_train_sorted_category_dict = cleaning_text_data(X_train['project_subject_categories'],X_train,'p
roject_subject_categories','clean_categories')
x_test_sorted_category_dict =
cleaning_text_data(X_test['project_subject_categories'],X_test,'project_subject_categories','clean_
categories')
x_cv_sorted_category_dict =
cleaning_text_data(X_cv['project_subject_categories'],X_cv,'project_subject_categories','clean_cate
gories')
```

2.2.2) Text Preprocessing : project_subject_subcategories

In [17]:

```
x_train_sorted_subcategories = cleaning_text_data(X_train['project_subject_subcategories'],X_train
,'project_subject_subcategories','clean_subcategories')
x_test_sorted_subcategories = cleaning_text_data(X_test['project_subject_subcategories'],X_test,'p
roject_subject_subcategories','clean_subcategories')
x_cv_sorted_subcategories =
cleaning_text_data(X_cv['project_subject_subcategories'],X_cv,'project_subject_subcategories','cle
an_subcategories')
```

2.2.4) Text Preprocessing: project_grade_category

In [18]:

```
x_train_sorted_grade =
clean_project_grade(X_train['project_grade_category'],X_train,'project_grade_category','clean_grade
')
x_test_sorted_grade =
clean_project_grade(X_test['project_grade_category'],X_test,'project_grade_category','clean_grade'
)
x_cv_sorted_grade =
clean_project_grade(X_cv['project_grade_category'],X_cv,'project_grade_category','clean_grade')
```

2.2.4) Text Preprocessing: project_essay, project_title

In [19]:

```
# 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 [20]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', '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', "do
esn'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 [21]:

```
# Combining all the above stundents
from tqdm import tqdm
def process_text(df,col_name):
    preprocessed_feature = []
    # tqdm is for printing the status bar
    for sentance in tqdm(df[col_name].values):
```

```
    for sentance in tqdm(df[col_name].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.lower() not in stopwords)
        preprocessed_feature.append(sent.lower().strip())
    return preprocessed_feature
```

In [22]:

```
x_train_essay_preprocessed = process_text(X_train,'essay')
x_test_essay_preprocessed = process_text(X_test,'essay')
x_cv_essay_preprocessed = process_text(X_cv,'essay')
```

```
100%|██████████████████████████████████████████████████████████████████████████| 33667/33667
[00:20<00:00, 1678.26it/s]
100%|██████████████████████████████████████████████████████████████████████████| 24750/24750
[00:14<00:00, 1667.00it/s]
100%|██████████████████████████████████████████████████████████████████████████| 16583/16583
[00:10<00:00, 1650.64it/s]
```

In [23]:

```
x_train_title_preprocessed = process_text(X_train,'project_title')
x_test_title_preprocessed = process_text(X_test,'project_title')
x_cv_title_preprocessed = process_text(X_cv,'project_title')
```

```
100%|█████████████████████████████████████████████████████████████████████████| 33667/33667
[00:01<00:00, 29598.61it/s]
100%|█████████████████████████████████████████████████████████████████████████| 24750/24750
[00:00<00:00, 30175.34it/s]
100%|█████████████████████████████████████████████████████████████████████████| 16583/16583
[00:00<00:00, 31392.36it/s]
```

In [45]:

```
x_train_title_preprocessed[:2]
```

Out[45]:

```
['technology 21st century', 'lights camera action language arts']
```

In [50]:

```
X_train['clean_categories']
```

Out[50]:

```
1375       Literacy_Language Math_Science
35345        Literacy_Language Music_Arts
36565                     AppliedLearning
10541                     AppliedLearning
43510                       Health_Sports
                       ...
16767      Literacy_Language Math_Science
25303    Literacy_Language History_Civics
44268                     AppliedLearning
14906                      History_Civics
28954                   Literacy_Language
Name: clean_categories, Length: 33667, dtype: object
```

2.2.5) Vectorizing Categorical Data

project_subject_categories (clean_categories)

In [24]:

In [24]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
def cat_vectorizer(X_train,df,col_name):
    vectorizer = CountVectorizer()
    vectorizer.fit(X_train[col_name].values)
    feature_one_hot = vectorizer.transform(df[col_name].values)
    print(vectorizer.get_feature_names())
    return feature_one_hot
```

In [25]:

```
x_train_cat_one_hot = cat_vectorizer(X_train,X_train,'clean_categories')
x_test_cat_one_hot = cat_vectorizer(X_train,X_test,'clean_categories')
x_cv_cat_one_hot = cat_vectorizer(X_train,X_cv,'clean_categories')
```

```
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
```

In [26]:

```
# shape after categorical one hot encoding
print(x_train_cat_one_hot.shape)
print(x_test_cat_one_hot.shape)
print(x_cv_cat_one_hot.shape)
```

```
(33667, 9)
(24750, 9)
(16583, 9)
```

project_subject_subcategory (clean_subcategory)

In [27]:

```
x_train_subcat_one_hot = cat_vectorizer(X_train,X_train,'clean_subcategories')
x_test_subcat_one_hot = cat_vectorizer(X_train,X_test,'clean_subcategories')
x_cv_subcat_one_hot = cat_vectorizer(X_train,X_cv,'clean_subcategories')
```

```
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
```

In [28]:

```
# shape after categorical one hot encoding
print(x_train_subcat_one_hot.shape)
print(x_test_subcat_one_hot.shape)
print(x_cv_subcat_one_hot.shape)
```

```
(33667, 30)
(24750, 30)
```

```
(24750, 30)
(16583, 30)
```

school_state

In [29]:

```
# we use count vectorizer to convert the values into one hot encoding
# CountVectorizer for "school_state"
x_train_state_one_hot = cat_vectorizer(X_train,X_train,'school_state')
x_test_state_one_hot = cat_vectorizer(X_train,X_test,'school_state')
x_cv_state_one_hot = cat_vectorizer(X_train,X_cv,'school_state')
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', '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']
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', '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']
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', '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']
```

In [30]:

```
# shape after categorical one hot encoding
print(x_train_state_one_hot.shape)
print(x_test_state_one_hot.shape)
print(x_cv_state_one_hot.shape)
```

```
(33667, 51)
(24750, 51)
(16583, 51)
```

teacher_prefix

In [31]:

```
# we use count vectorizer to convert the values into one hot encoding
# CountVectorizer for teacher_prefix
x_train_teacher_prefix_one_hot = cat_vectorizer(X_train,X_train,'teacher_prefix')
x_test_teacher_prefix_one_hot = cat_vectorizer(X_train,X_test,'teacher_prefix')
x_cv_teacher_prefix_one_hot = cat_vectorizer(X_train,X_cv,'teacher_prefix')
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
['dr', 'mr', 'mrs', 'ms', 'teacher']
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

In [32]:

```
# shape after categorical one hot encoding
print(x_train_teacher_prefix_one_hot.shape)
print(x_test_teacher_prefix_one_hot.shape)
print(x_cv_teacher_prefix_one_hot.shape)
```

```
(33667, 5)
(24750, 5)
(16583, 5)
```

project_grade_category

In [33]:

```
# using count vectorizer for one-hot encoding of project_grade_category
# using count vectorizer for one hot encoding of project_grade_category
x_train_grade_one_hot = cat_vectorizer(X_train,X_train,'clean_grade')
x_test_grade_one_hot = cat_vectorizer(X_train,X_test,'clean_grade')
x_cv_grade_one_hot = cat_vectorizer(X_train,X_cv,'clean_grade')
```

```
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

In [34]:

```
# shape after categorical one hot encoding
print(x_train_grade_one_hot.shape)
print(x_test_grade_one_hot.shape)
print(x_cv_grade_one_hot.shape)
```

```
(33667, 4)
(24750, 4)
(16583, 4)
```

2.2.6) Vectorizing Text Data

2.2.6.1) Bag of words

In [35]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
def bow_vectorizer(X_train,col_name,df):
    vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
    vectorizer.fit(X_train[col_name].values)
    df_bow = vectorizer.transform(df[col_name].values)
    return df_bow
```

In [36]:

```
x_train_essay_bow = bow_vectorizer(X_train,'essay',X_train)
x_test_essay_bow = bow_vectorizer(X_train,'essay',X_test)
x_cv_essay_bow = bow_vectorizer(X_train,'essay',X_cv)
```

In [37]:

```
print(x_train_essay_bow.shape)
print(x_test_essay_bow.shape)
print(x_cv_essay_bow.shape)
```

```
(33667, 5000)
(24750, 5000)
(16583, 5000)
```

In [40]:

```
x_train_title_bow = bow_vectorizer(X_train,'project_title',X_train)
x_test_title_bow = bow_vectorizer(X_train,'project_title',X_test)
x_cv_title_bow = bow_vectorizer(X_train,'project_title',X_cv)
```

In [41]:

```
print(x_train_title_bow.shape)
print(x_test_title_bow.shape)
print(x_cv_title_bow.shape)
```

```
(33667, 3942)
(24750, 3942)
(16583, 3942)
```

2.2.6.2) TFIDF Vectorizer

In [47]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
def tfidf_vectorizer(X_train,col_name,df):
    vectorizer = TfidfVectorizer(min_df=10)
    vectorizer.fit(X_train[col_name].values)
    df_tfidf = vectorizer.transform(df[col_name].values)
    return df_tfidf
```

In [48]:

```
# Lets vectorize essay
x_train_essay_tfidf = tfidf_vectorizer(X_train,'essay',X_train)
x_test_essay_tfidf = tfidf_vectorizer(X_train,'essay',X_test)
x_cv_essay_tfidf = tfidf_vectorizer(X_train,'essay',X_cv)
```

In [49]:

```
print(x_train_essay_tfidf.shape)
print(x_test_essay_tfidf.shape)
print(x_cv_essay_tfidf.shape)
```

```
(33667, 10758)
(24750, 10758)
(16583, 10758)
```

In [52]:

```
# Lets vectorize project_title as clean_title
x_train_title_tfidf = tfidf_vectorizer(X_train,'project_title',X_train)
x_test_title_tfidf = tfidf_vectorizer(X_train,'project_title',X_test)
x_cv_title_tfidf = tfidf_vectorizer(X_train,'project_title',X_cv)
```

In [53]:

```
print(x_train_title_tfidf.shape)
print(x_test_title_tfidf.shape)
print(x_cv_title_tfidf.shape)
```

```
(33667, 1651)
(24750, 1651)
(16583, 1651)
```

2.2.6.3) Using Pretrained Models: Avg W2V

In [54]:

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

# ============================

words = []
for i in preproced_texts:
    words.extend(i.split(' '))

for i in preproced_titles:
    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-sa
ve-and-load-variables-in-python/

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


'''
```

Out[54]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n        
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
============================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus",
len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\n
print("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [55]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-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())
```

In [56]:

```
# average Word2Vec
# compute average word2vec for each review.
def compute_avg_W2V(preprocessed_feature):
    avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(preprocessed_feature): # 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_vectors.append(vector)
    return avg_w2v_vectors
```

In [57]:

```
x_train_avg_w2v_essay = compute_avg_W2V(x_train_essay_preprocessed)
x_test_avg_w2v_essay = compute_avg_W2V(x_test_essay_preprocessed)
x_cv_avg_w2v_essay = compute_avg_W2V(x_cv_essay_preprocessed)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 33667/33667
[00:21<00:00, 1592.35it/s]
100%|████████████████████████████████████████████████████████████████████████████| 24750/24750
[00:15<00:00, 1598.48it/s]
100%|████████████████████████████████████████████████████████████████████████████| 16583/16583
[00:10<00:00, 1580.17it/s]
```

In [58]:

```
x_train_avg_w2v_title = compute_avg_W2V(x_train_title_preprocessed)
x_test_avg_w2v_title = compute_avg_W2V(x_test_title_preprocessed)
x_cv_avg_w2v_title = compute_avg_W2V(x_cv_title_preprocessed)
```

```
100%|███████████████████████████████████████████████████████████████████████████| 33667/33667
[00:01<00:00, 30474.75it/s]
100%|███████████████████████████████████████████████████████████████████████████| 24750/24750
[00:00<00:00, 33238.29it/s]
100%|███████████████████████████████████████████████████████████████████████████| 16583/16583
[00:00<00:00, 33411.78it/s]
```

2.2.6.4) Using Pretrained Models: TFIDF Weighted W2V

In [59]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
def get_tfidf_dict(preprocessed_feature):
    tfidf_model = TfidfVectorizer()
    tfidf_model.fit(preprocessed_feature)
    # 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())
    return dictionary, tfidf_words
```

In [60]:

```
# average Word2Vec
# compute average word2vec for each review.
def compute_tfidf_w2v_vectors(preprocessed_feature):
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    dictionary, tfidf_words = get_tfidf_dict(preprocessed_feature)
    for sentence in tqdm(preprocessed_feature): # 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
        tfidf_w2v_vectors.append(vector)
    return tfidf_w2v_vectors
```

In [61]:

```
x_train_weighted_w2v_essay = compute_tfidf_w2v_vectors(x_train_essay_preprocessed)
x_test_weighted_w2v_essay= compute_tfidf_w2v_vectors(x_test_essay_preprocessed)
x_cv_weighted_w2v_essay = compute_tfidf_w2v_vectors(x_cv_essay_preprocessed)
```

```
100%|███████████████████████████████████████████████████████████████████████████| 33667/33667 [02:
17<00:00, 245.48it/s]
100%|███████████████████████████████████████████████████████████████████████████| 24750/24750 [01:
42<00:00, 241.69it/s]
100%|███████████████████████████████████████████████████████████████████████████| 16583/16583 [01:
08<00:00, 241.18it/s]
```

In [62]:

```
x_train_weighted_w2v_title = compute_tfidf_w2v_vectors(x_train_title_preprocessed)
x_test_weighted_w2v_title= compute_tfidf_w2v_vectors(x_test_title_preprocessed)
x_cv_weighted_w2v_title = compute_tfidf_w2v_vectors(x_cv_title_preprocessed)
```

```
100%|█████████████████████████████████████████████████████████████████████████| 33667/33667
[00:02<00:00, 13881.73it/s]
100%|█████████████████████████████████████████████████████████████████████████| 24750/24750
[00:01<00:00, 14179.40it/s]
100%|█████████████████████████████████████████████████████████████████████████| 16583/16583
[00:01<00:00, 14960.49it/s]
```

2.2.7) Vectorizing Numerical Features

-teacher_number_of_previously_posted_projects

In [63]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler
def scaler_function(df,col_name):

    scalar = StandardScaler()
    scalar.fit(df[col_name].values.reshape(-1,1)) # finding the mean and standard deviation of this
data
    print(f"Mean : {scalar.mean_[0]}, Standard deviation : {np.sqrt(scalar.var_[0])}")

    # Now standardize the data with above maen and variance.
    standardized = scalar.transform(df[col_name].values.reshape(-1, 1))
    return standardized
```

In [64]:

```
x_train_teacher_number = scaler_function(X_train,'teacher_number_of_previously_posted_projects')
x_test_teacher_number = scaler_function(X_test,'teacher_number_of_previously_posted_projects')
x_cv_teacher_number = scaler_function(X_cv,'teacher_number_of_previously_posted_projects')
```

```
Mean : 11.349749012386017, Standard deviation : 28.18534056560529
Mean : 11.095959595959595, Standard deviation : 27.697356542325675
Mean : 11.215823433636857, Standard deviation : 27.936626112869888
```

In [65]:

```
x_train_teacher_number
```

Out[65]:

```
array([[-0.29624439],
       [-0.26076495],
       [-0.15432664],
       ...,
       [-0.4026827 ],
       [-0.4026827 ],
       [-0.15432664]])
```

In [66]:

```
x_test_teacher_number
```

Out[66]:

```
array([[ 4.50960149],
       [-0.2923008 ],
       [ 0.28537165],
       ...,
       [-0.32840533],
       [-0.36450986],
       [-0.40061439]])
```

In [67]:

```
x_cv_teacher_number
```

Out[67]:

```
array([[-0.4014738 ],
       [-0.4014738 ],
       [-0.4014738 ],
       ...,
       [-0.32988319],
       [-0.4014738 ],
       [-0.4014738 ]])
```

2.2.8) Merging all the above features

Let's analyse the features for train,test and cv datasets

In [68]:

```
# train dataset
print("After Vectorization and One hot encoding train dataset shape becomes:")
print(x_train_cat_one_hot.shape)
print(x_train_subcat_one_hot.shape)
print(x_train_state_one_hot.shape)
print(x_train_teacher_prefix_one_hot.shape)
print(x_train_grade_one_hot.shape)
print(x_train_essay_bow.shape)
print(x_train_title_bow.shape)
print(x_train_essay_tfidf.shape)
print(x_train_title_tfidf.shape)
print(np.asarray(x_train_avg_w2v_essay).shape)
print(np.asarray(x_train_avg_w2v_title).shape)
print(np.asarray(x_train_weighted_w2v_essay).shape)
print(np.asarray(x_train_weighted_w2v_title).shape)
print("="*50)
# test dataset
print("After Vectorization and One hot encoding test dataset shape becomes:")
print(x_test_cat_one_hot.shape)
print(x_test_subcat_one_hot.shape)
print(x_test_state_one_hot.shape)
print(x_test_teacher_prefix_one_hot.shape)
print(x_test_grade_one_hot.shape)
print(x_test_essay_bow.shape)
print(x_test_title_bow.shape)
print(x_test_essay_tfidf.shape)
print(x_test_title_tfidf.shape)
print(np.asarray(x_test_avg_w2v_essay).shape)
print(np.asarray(x_test_avg_w2v_title).shape)
print(np.asarray(x_test_weighted_w2v_essay).shape)
print(np.asarray(x_test_weighted_w2v_title).shape)
print("="*50)
# cv dataset
```

```python
print("After Vectorization and One hot encoding cv dataset shape becomes:")
print(x_cv_cat_one_hot.shape)
print(x_cv_subcat_one_hot.shape)
print(x_cv_state_one_hot.shape)
print(x_cv_teacher_prefix_one_hot.shape)
print(x_cv_grade_one_hot.shape)
print(x_cv_essay_bow.shape)
print(x_cv_title_bow.shape)
print(x_cv_essay_tfidf.shape)
print(x_cv_title_tfidf.shape)
print(np.asarray(x_cv_avg_w2v_essay).shape)
print(np.asarray(x_cv_avg_w2v_title).shape)
print(np.asarray(x_cv_weighted_w2v_essay).shape)
print(np.asarray(x_cv_weighted_w2v_title).shape)
print("="*50)

##########################################
# print(categories_one_hot.shape)
# print(sub_categories_one_hot.shape)
# print(school_state_one_hot.shape)
# print(teacher_prefix_one_hot.shape)
# print(project_category_one_hot.shape)
# print(price_standardized.shape)
# print(teacher_number_standardized.shape)
# print(text_bow.shape)
# print(title_text_bow.shape)
# print(text_tfidf.shape)
# print(title_text_tfidf.shape)
# print(np.asarray(avg_w2v_vectors).shape)
# print(np.asarray(titles_avg_w2v_vectors).shape)
# print(np.asarray(tfidf_w2v_vectors).shape)
# print(np.asarray(titles_tfidf_w2v_vectors).shape)
```

```
After Vectorization and One hot encoding train dataset shape becomes:
(33667, 9)
(33667, 30)
(33667, 51)
(33667, 5)
(33667, 4)
(33667, 5000)
(33667, 3942)
(33667, 10758)
(33667, 1651)
(33667, 300)
(33667, 300)
(33667, 300)
(33667, 300)
==================================================
After Vectorization and One hot encoding test dataset shape becomes:
(24750, 9)
(24750, 30)
(24750, 51)
(24750, 5)
(24750, 4)
(24750, 5000)
(24750, 3942)
(24750, 10758)
(24750, 1651)
(24750, 300)
(24750, 300)
(24750, 300)
(24750, 300)
==================================================
After Vectorization and One hot encoding cv dataset shape becomes:
(16583, 9)
(16583, 30)
(16583, 51)
(16583, 5)
(16583, 4)
(16583, 5000)
(16583, 3942)
(16583, 10758)
(16583, 1651)
(16583, 300)
(16583, 300)
(16583, 300)
```

```
(16583, 300)
==================================================
```

Let's merge train,test and cv dataset features into single data Matrix, X_train_matrix, X_test_matrix and X_cv_matrix respectively

In [69]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_matrix =
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\

x_train_grade_one_hot,x_train_essay_bow,x_train_title_bow,x_train_essay_tfidf,x_train_title_tfidf,
\

x_train_avg_w2v_essay,x_train_avg_w2v_title,x_train_weighted_w2v_essay,x_train_weighted_w2v_title)
).tocsr()
X_train_matrix.shape
```

Out[69]:

```
(33667, 22650)
```

In [70]:

```
X_test_matrix =
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                         x_test_grade_one_hot,x_test_essay_bow,x_test_title_bow,x_test_essay_tfidf,
x_test_title_tfidf,\

x_test_avg_w2v_essay,x_test_avg_w2v_title,x_test_weighted_w2v_essay,x_test_weighted_w2v_title)).to
csr()
X_test_matrix.shape
```

Out[70]:

```
(24750, 22650)
```

In [96]:

```
X_cv_matrix =
hstack((x_cv_cat_one_hot,x_cv_subcat_one_hot,x_cv_state_one_hot,x_cv_teacher_prefix_one_hot,\

x_cv_grade_one_hot,x_cv_essay_bow,x_cv_title_bow,x_cv_essay_tfidf,x_cv_title_tfidf,\

x_cv_avg_w2v_essay,x_cv_avg_w2v_title,x_cv_weighted_w2v_essay,x_cv_weighted_w2v_title)).tocsr()
X_cv_matrix.shape
```

Out[96]:

```
(16583, 22650)
```

In [103]:

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

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

2) Apply KNN - Brute Force Version

2.1) Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)

Let's Merge categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)

In [72]:

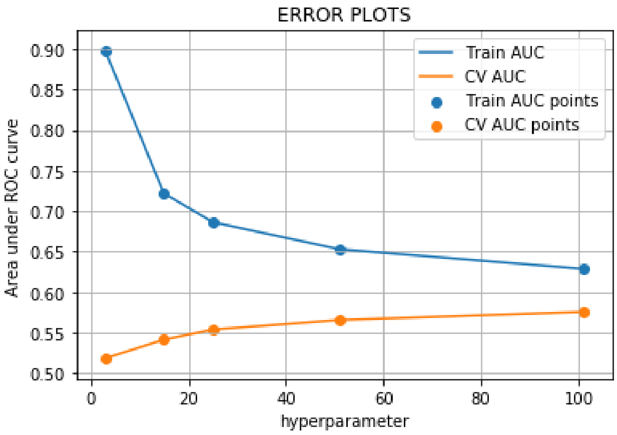
```
X_train_set_1 =
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\
                         x_train_grade_one_hot,x_train_essay_bow,x_train_title_bow)).tocsr()
X_test_set_1 =
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                         x_test_grade_one_hot,x_test_essay_bow,x_test_title_bow)).tocsr()
X_cv_set_1 =
hstack((x_cv_cat_one_hot,x_cv_subcat_one_hot,x_cv_state_one_hot,x_cv_teacher_prefix_one_hot,\
                         x_cv_grade_one_hot,x_cv_essay_bow,x_cv_title_bow)).tocsr()
```

In [73]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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 unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [101]:

```
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 values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

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_train_set_1, y_train)

    y_train_pred = batch_predict(neigh, X_train_set_1)
    y_cv_pred = batch_predict(neigh, X_cv_set_1)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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("hyperparameter")
plt.ylabel("Area under ROC curve")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

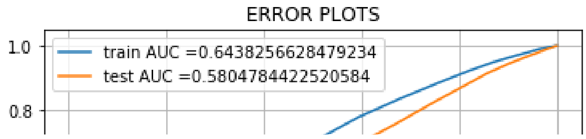


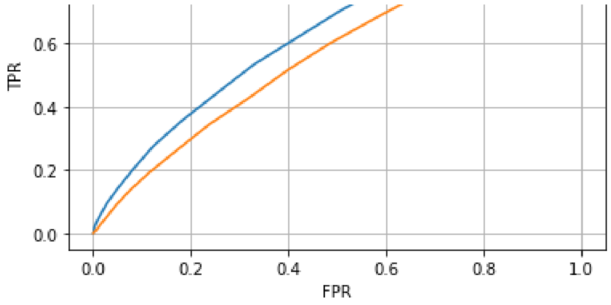
In [154]:

```
# from the plot, we got the optimum value for K as 101
best_k = 101
```

In [156]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_set_1, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_set_1)
y_test_pred = batch_predict(neigh, X_test_set_1)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

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





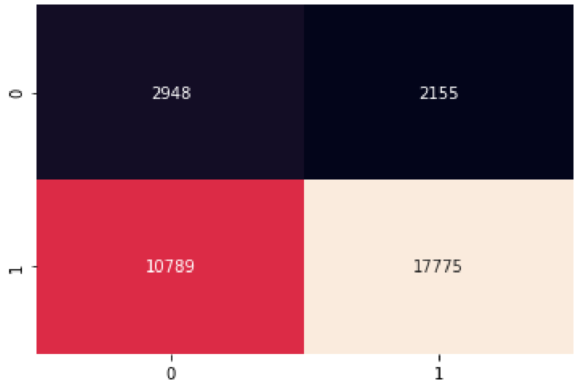
In [105]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)),annot = True, fmt
= "d", cbar=False)
```

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

the maximum value of tpr*(1-fpr) 0.3594947031907308 for threshold 0.832
Train confusion matrix
```

Out[105]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f300a5d8d0>
```



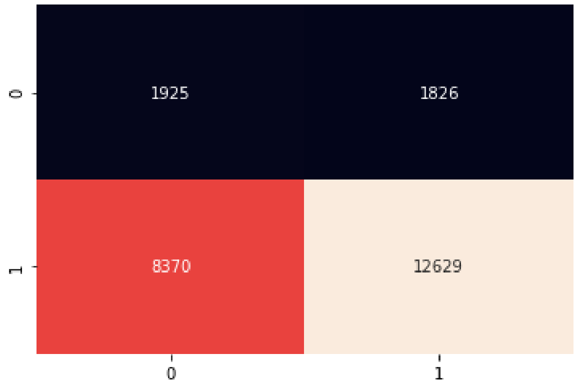
In [106]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), annot = True, fmt =
"d", cbar=False)
```

```
Test confusion matrix
```

Out[106]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f300aa3908>
```



2.2) Set 2: categorical, numerical features + project_title(TFIDF) + preprocessed_essay (TFIDF)

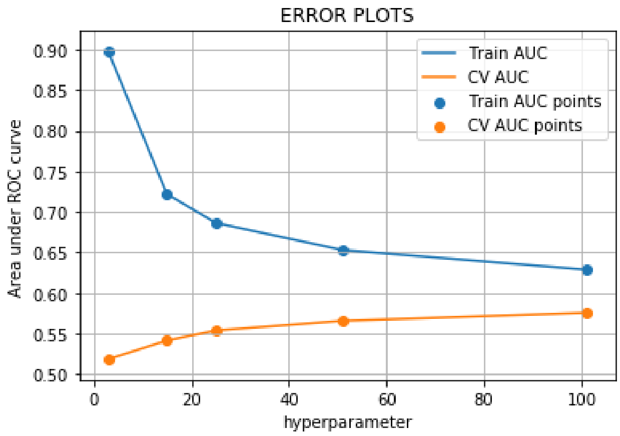
In [77]:

```
X_train_set_2 =
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\
                         x_train_grade_one_hot,x_train_essay_tfidf,x_train_title_tfidf)).tocsr()
X_test_set_2 =
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                         x_test_grade_one_hot,x_test_essay_tfidf,x_test_title_tfidf)).tocsr()
X_cv_set_2 =
hstack((x_cv_cat_one_hot,x_cv_subcat_one_hot,x_cv_state_one_hot,x_cv_teacher_prefix_one_hot,\
                         x_cv_grade_one_hot,x_cv_essay_tfidf,x_cv_title_tfidf)).tocsr()
```

In [107]:

```
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_train_set_2, y_train)

    y_train_pred = batch_predict(neigh, X_train_set_2)
    y_cv_pred = batch_predict(neigh, X_cv_set_2)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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("hyperparameter")
plt.ylabel("Area under ROC curve")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████████████████████████████████████████████████████████████| 5/5 [21:
07<00:00, 253.42s/it]
```

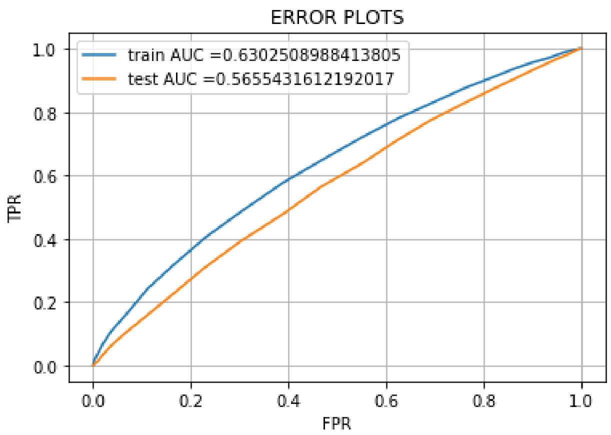


In [157]:

```
best_k = 95
```

In [158]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_set_2, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_set_2)
y_test_pred = batch_predict(neigh, X_test_set_2)

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("ERROR PLOTS")
plt.grid()
plt.show()
```



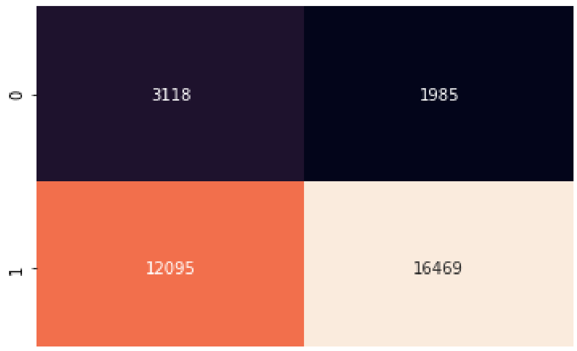
In [110]:

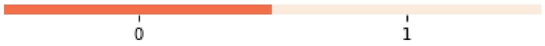
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)),annot = True, fmt
= "d", cbar=False)
```

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

the maximum value of tpr*(1-fpr) 0.35228872812829826 for threshold 0.853
Train confusion matrix
```

Out[110]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f3014edba8>
```





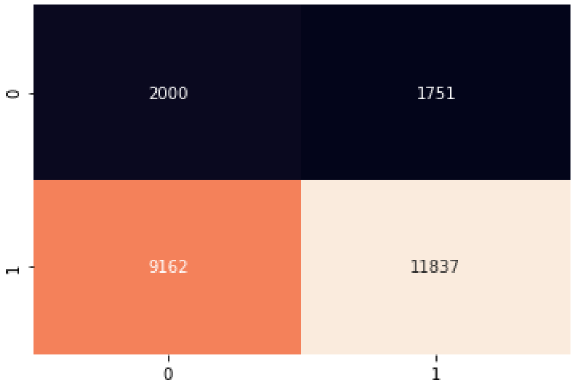
In [111]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), annot = True, fmt =
"d", cbar=False)
```

```
Test confusion matrix
```

Out[111]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f3014ff550>
```



2.3) Set 3: categorical, numerical features + project_title(AVG_W2V) + preprocessed_essay (AVG_W2V)

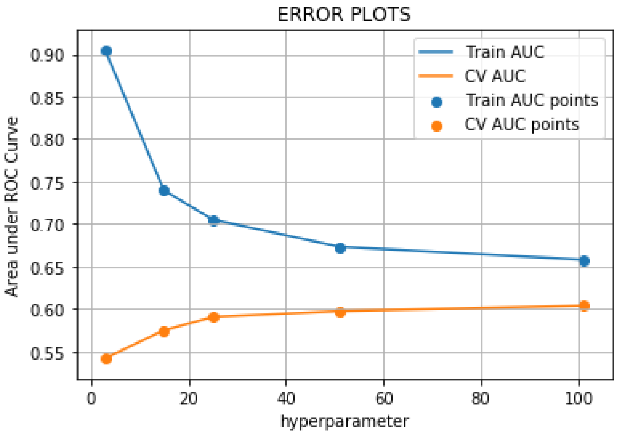
In [83]:

```
X_train_set_3 =
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\
                         x_train_grade_one_hot,x_train_avg_w2v_essay,x_train_avg_w2v_title)).tocsr(
)
X_test_set_3 =
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                         x_test_grade_one_hot,x_test_avg_w2v_essay,x_test_avg_w2v_title)).tocsr()
X_cv_set_3 =
hstack((x_cv_cat_one_hot,x_cv_subcat_one_hot,x_cv_state_one_hot,x_cv_teacher_prefix_one_hot,\
                         x_cv_grade_one_hot,x_cv_avg_w2v_essay,x_cv_avg_w2v_title)).tocsr()
```

In [112]:

```
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_train_set_3, y_train)

    y_train_pred = batch_predict(neigh, X_train_set_3)
    y_cv_pred = batch_predict(neigh, X_cv_set_3)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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.scatter(K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Area under ROC Curve")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████████████████████████████████████████████| 5/5
[1:56:28<00:00, 1397.66s/it]
```

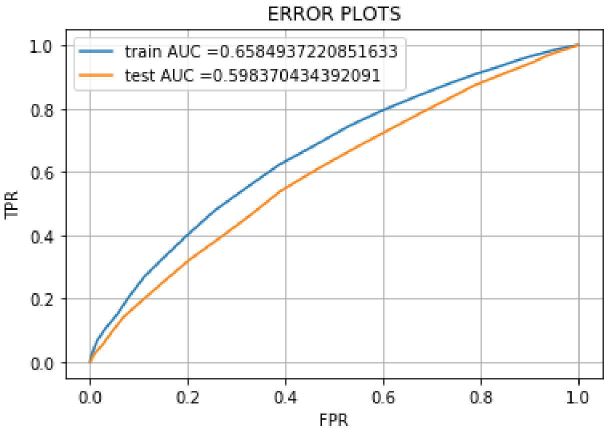


In [159]:

```
best_k = 99
```

In [160]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_set_3, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_set_3)
y_test_pred = batch_predict(neigh, X_test_set_3)

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("ERROR PLOTS")
plt.grid()
plt.show()
```



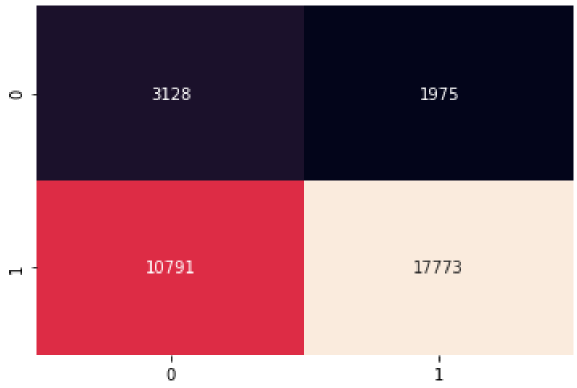
In [115]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)),annot = True, fmt
= "d", cbar=False)
```

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

the maximum value of tpr*(1-fpr) 0.381401935422277 for threshold 0.848
Train confusion matrix
```

Out[115]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f3014da198>
```



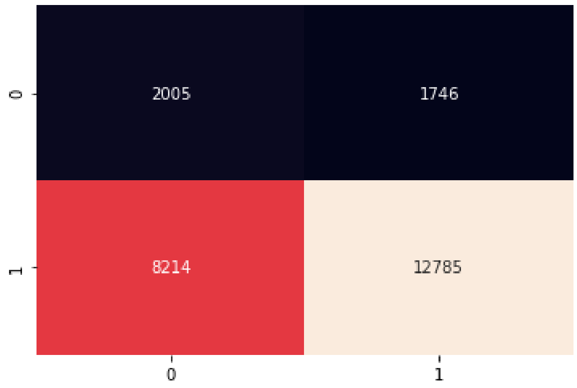
In [116]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), annot = True, fmt =
"d", cbar=False)
```

```
Test confusion matrix
```

Out[116]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f301599898>
```



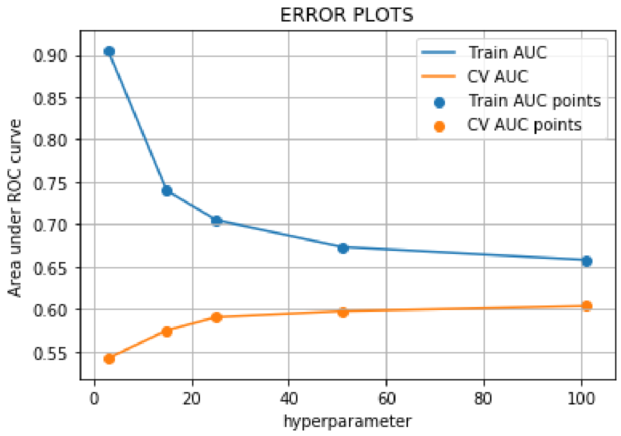
2.4) Set 4: categorical, numerical features + project_title(TFIDF_W2V) + preprocessed_essay (TFIDF_W2V)

In [85]:

```
X_train_set_4 =
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\
x_train_grade_one_hot,x_train_weighted_w2v_essay,x_train_weighted_w2v_title)).tocsr()
X_test_set_4 =
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                                 x_test_grade_one_hot,x_test_weighted_w2v_essay,x_test_weighted_w2v_title))
.tocsr()
X_cv_set_4 =
hstack((x_cv_cat_one_hot,x_cv_subcat_one_hot,x_cv_state_one_hot,x_cv_teacher_prefix_one_hot,\
                                 x_cv_grade_one_hot,x_cv_weighted_w2v_essay,x_cv_weighted_w2v_title)).tocsr
()
```

In [117]:

```
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_train_set_4, y_train)

    y_train_pred = batch_predict(neigh, X_train_set_4)
    y_cv_pred = batch_predict(neigh, X_cv_set_4)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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("hyperparameter")
plt.ylabel("Area under ROC curve")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



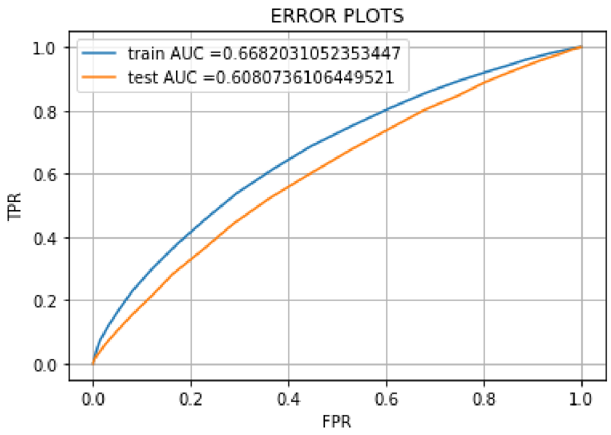
In [161]:

```
best_k = 85
```

In [162]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_set_4, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_set_4)
y_test_pred = batch_predict(neigh, X_test_set_4)
```

```
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("ERROR PLOTS")
plt.grid()
plt.show()
```



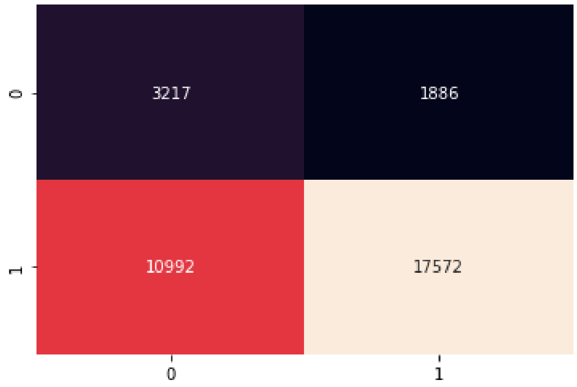
In [121]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)),annot = True, fmt
= "d", cbar=False)
```

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

the maximum value of tpr*(1-fpr) 0.38781773247326884 for threshold 0.847
Train confusion matrix
```

Out[121]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f3015d4390>
```



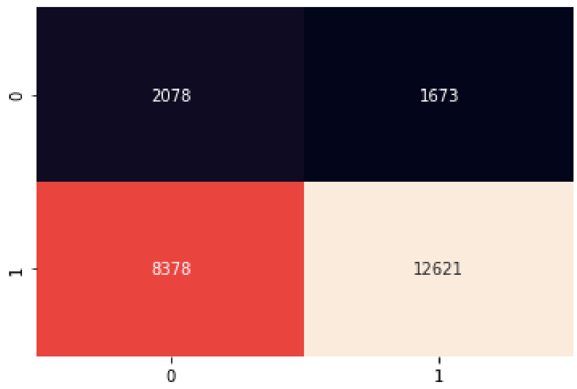
In [122]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), annot = True, fmt =
"d", cbar=False)
```

```
Test confusion matrix
```

Out[122]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f3016877f0>
```



Task 2) Applying SelectKBest on Set 2

In [124]:

```
from sklearn.feature_selection import SelectKBest, chi2
```

In [163]:

```
algorithm = SelectKBest(chi2, k = 2000)
X_train_new = algorithm.fit_transform(X_train_set_2, y_train)
```

In [164]:

```
X_cv_new = algorithm.transform(X_cv_set_2)
X_test_new = algorithm.transform(X_test_set_2)
```

In [165]:

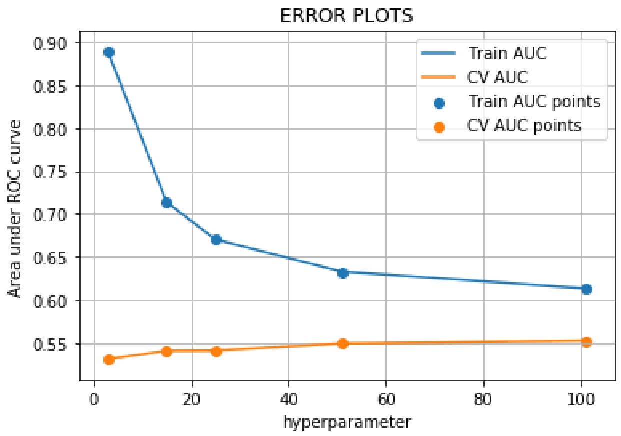
```
print(X_train_new.shape, y_train.shape)
print(X_test_new.shape, y_test.shape)
print(X_cv_new.shape, y_cv.shape)
```

```
(33667, 2000) (33667,)
(24750, 2000) (24750,)
(16583, 2000) (16583,)
```

In [152]:

```
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_train_new, y_train)

    y_train_pred = batch_predict(neigh, X_train_new)
    y_cv_pred = batch_predict(neigh, X_cv_new)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive 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.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Area under ROC curve")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████████████████████████████████████████████████████████████████████| 5/5 [12:
14<00:00, 146.97s/it]
```

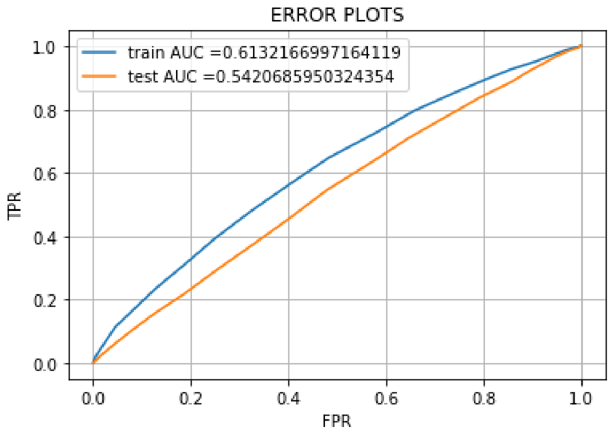


In [166]:

```
best_k = 99
```

In [167]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_train_new, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train_new)
y_test_pred = batch_predict(neigh, X_test_new)

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("ERROR PLOTS")
plt.grid()
plt.show()
```



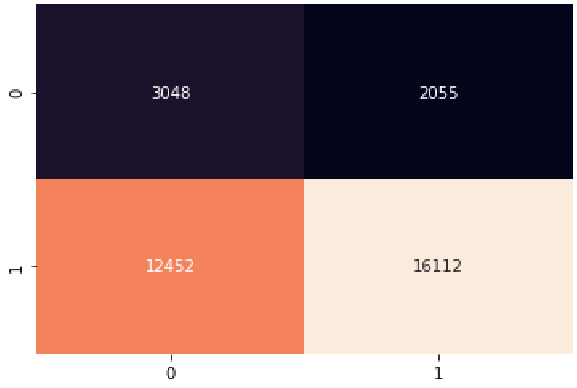
In [144]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)),annot = True, fmt
= "d", cbar=False)
```

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

the maximum value of tpr*(1-fpr) 0.33691459367913024 for threshold 0.848
Train confusion matrix
```

Out[144]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f3015da710>
```



	0	1
0	3048	2055
1	12452	16112

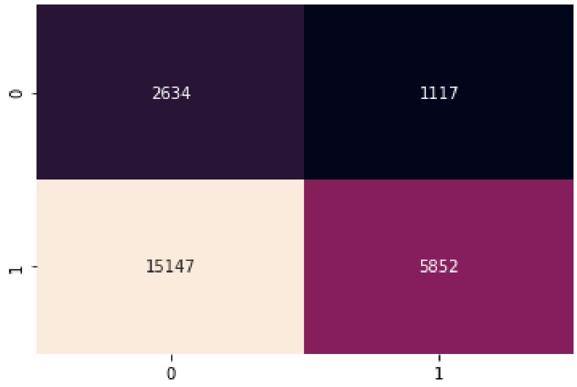
In [145]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), annot = True, fmt =
"d", cbar=False)
```

```
Test confusion matrix
```

Out[145]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f3014e04a8>
```



	0	1
0	2634	1117
1	15147	5852

In [146]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model", "Dataset", "Best_K", "Train AUC","Test AUC"]

x.add_row(["KNeighborsClaassifier", "Set 1: (BOW)", 101, 0.6438, 0.5804])
x.add_row(["KNeighborsClaassifier", "Set 2: (TFIDF)",95, 0.6302, 0.5655])
x.add_row(["KNeighborsClaassifier", "Set 3: (AVG_W2V)", 99, 0.6584, 0.5983])
x.add_row(["KNeighborsClaassifier", "Set 4: (TFIDF_W2V)", 85, 0.6682, 0.6080])
x.add_row(["KNeighborsClaassifier", "Set 2: SelectKBest(k=2000)(TFIDF)", 85, 0.6132, 0.4924])
```

```
x.add_row(["KNeighborsClaassifier", "Set 2: SelectKBest(k=2000)(TFIDF)", 85, 0.6132, 0.4924])

print(x)
```

```
+-----------------------+-------------------------------------+---------+-----------+----------+
|         Model         |               Dataset               | Best_K | Train AUC | Test AUC |
+-----------------------+-------------------------------------+---------+-----------+----------+
| KNeighborsClaassifier |            Set 1: (BOW)             |   101   |   0.6438  |  0.5804  |
| KNeighborsClaassifier |           Set 2: (TFIDF)            |    95   |   0.6302  |  0.5655  |
| KNeighborsClaassifier |          Set 3: (AVG_W2V)           |    99   |   0.6584  |  0.5983  |
| KNeighborsClaassifier |         Set 4: (TFIDF_W2V)          |    85   |   0.6682  |  0.608   |
| KNeighborsClaassifier | Set 2 SelectKBest(k=2000): (TFIDF) |    85   |   0.6132  |  0.4924  |
+-----------------------+-------------------------------------+---------+-----------+----------+
```