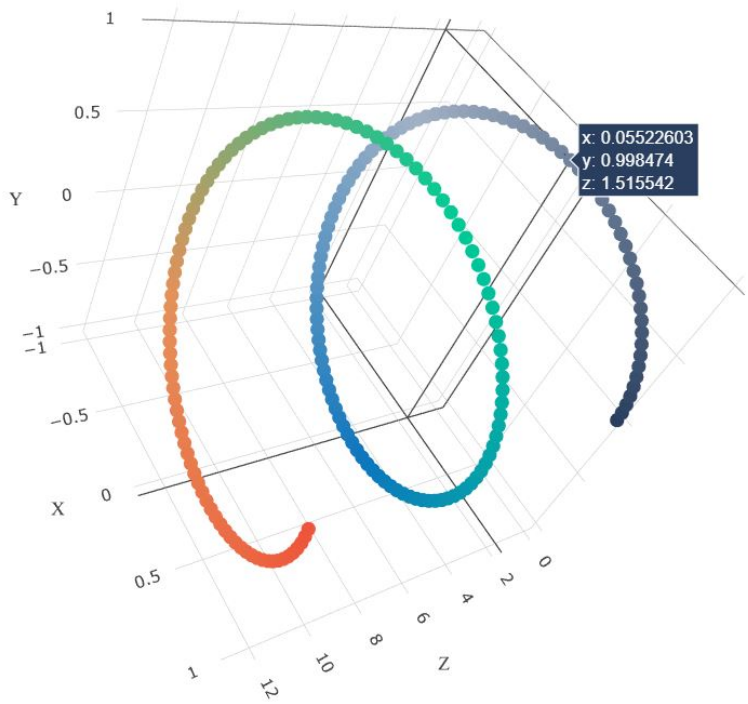
DonorsChoose

1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**

    - Set 1: categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)
    - Set 2: categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)
      </ul> </li>

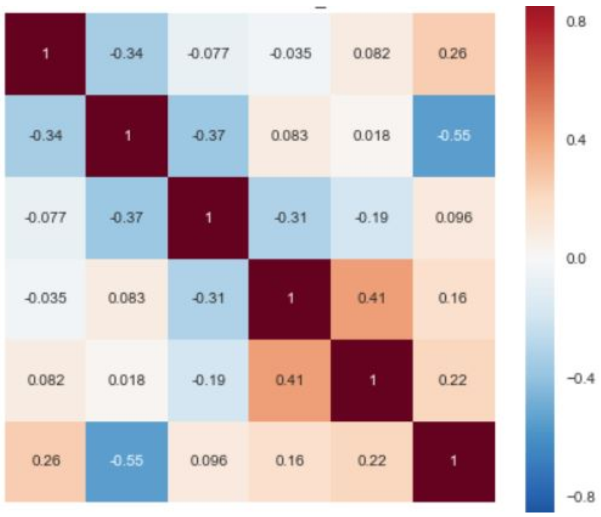
- **The hyper paramter tuning (best `depth` in range [1, 5, 10, 50], and the best `min_samples_split` in range [5, 10, 100, 500])**
    - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
    - find the best hyper paramter using k-fold cross validation(use gridsearch cv or randomsearch cv)/simple cross validation data(you can write your own for loops refer sample solution)
      </ul> </li>
    - **Representation of results**
        - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



with X-axis as **min_sample_split**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

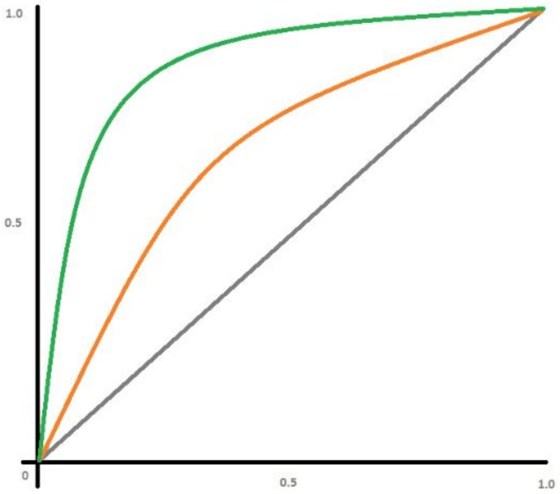
or

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure

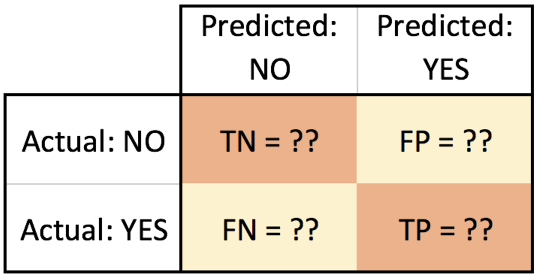


seaborn heat maps (https://seaborn.pydata.org/generated/seaborn.heatmap.html) with rows as **min_sample_split**, columns as **max_depth**, and values inside the cell representing **AUC Score**

- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.



- Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points



	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

- Once after you plot the confusion matrix with the test data, get all the `false positive data points`
  - Plot the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) with the words of essay text of these `false positive data points`
  - Plot the box plot with the `price` of these `false positive data points`

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

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

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

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	
`project_id`	A unique identifier for the proposed project. **Examp
`project_title`	Title of the proje • `Art Will Make` • `First`
`project_grade_category`	Grade level of students for which the project is targeted. One c enum • `Gra` • • • `G`
`project_subject_categories`	One or more (comma-separated) subject categories for the p following enumeratec • `Applie` • `Car` • `Healt` • `Histor` • `Literacy` • `Math` • `Music` • `Spe` • • `Music` • `Literacy & Language, Math`
`school_state`	State where school is located (Two-letter U. (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#F
`project_subject_subcategories`	One or more (comma-separated) subject subcategories • • `Literature & Writing, Socia`
`project_resource_summary`	An explanation of the resources needed for the proj • `My students need hands on literacy materials sens`
`project_essay_1`	First app
`project_essay_2`	Second app
`project_essay_3`	Third app
`project_essay_4`	Fourth app
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `12`
`teacher_id`	A unique identifier for the teacher of the proposed pro `bdf8baa8fedef6bfeec7ae`

Feature	
`teacher_prefix`	Teacher's title. One of the following enum • • • • • •
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the

* See the section **Notes on the Essay Data** for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p036502`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved.

Notes on the Essay Data

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

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

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

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

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

In [62]:

```
!pip install chart_studio
```

```
Requirement already satisfied: chart_studio in /usr/local/lib/python
3.7/dist-packages (1.1.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/
dist-packages (from chart_studio) (2.23.0)
Requirement already satisfied: plotly in /usr/local/lib/python3.7/di
st-packages (from chart_studio) (4.4.1)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/pyt
hon3.7/dist-packages (from chart_studio) (1.3.3)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-
packages (from chart_studio) (1.15.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.2
1.1 in /usr/local/lib/python3.7/dist-packages (from requests->chart_
studio) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/
python3.7/dist-packages (from requests->chart_studio) (2021.5.30)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/p
ython3.7/dist-packages (from requests->chart_studio) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python
3.7/dist-packages (from requests->chart_studio) (2.10)
```

In [15]:

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

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

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

import pickle
from tqdm import tqdm
import os

import chart_studio.plotly as plotly
import plotly.graph_objs as go

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

In [16]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remo
unt, call drive.mount("/content/drive", force_remount=True).
```

1. Loading Data

In [127]:

```
data.shape
```

Out[127]:

```
(50000, 9)
```

In [126]:

```
data  = pd.read_csv('/content/drive/MyDrive/DT/preprocessed_data.csv', nrows=500
00)
data.head(3)
```

Out[126]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_p
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	
2	ca	mrs	grades_prek_2	

In [128]:

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

Out[128]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_p
0	ca	mrs	grades_prek_2	

In [130]:

```
# 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, strati
fy=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33
, stratify=y_train)
```

In [131]:

```python
print("Before vectorizations")
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
print("="*100)
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)

# encoding eassay
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data
# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("After TFIDF vectorization of essay")
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)
```

```
Before vectorizations
(22445, 8) (22445,)
(11055, 8) (11055,)
(16500, 8) (16500,)
====================================================================
================================
After TFIDF vectorization of essay
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================
================================
```

In [133]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train
data
# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values )
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)
print("After vectorization of School-state")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorization of School-state
(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', 'm
i', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'n
v', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'u
t', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
====================================================================
================================
```

In [134]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on tra
in data
# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)
print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 5) (22445,)
(11055, 5) (11055,)
(16500, 5) (16500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================
================================
```

In [135]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to h appen on
ly on train data
# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].value
s)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category' ].values
)
print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================
================================
```

In [136]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on t
rain data
# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_categories_ohe = vectorizer.transform(X_train['clean_categories'].
values)
X_cv_clean_categories_ohe = vectorizer.transform(X_cv['clean_categories'].values
)
X_test_clean_categories_ohe = vectorizer.transform(X_test['clean_categories'].va
lues)
print("After vectorizations")
print(X_train_clean_categories_ohe.shape, y_train.shape)
print(X_cv_clean_categories_ohe.shape, y_cv.shape)
print(X_test_clean_categories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 7) (22445,)
(11055, 7) (11055,)
(16500, 7) (16500,)
['appliedlearning', 'health_sports', 'history_civics', 'literacy_lan
guage', 'math_science', 'music_arts', 'specialneeds']
====================================================================
================================
```

In [137]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happ en only
 on train data
# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategories_ohe = vectorizer.transform(X_train['clean_subcategor
ies'].values)
X_cv_clean_subcategories_ohe = vectorizer.transform(X_cv['clean_subcategories'].
values)
X_test_clean_subcategories_ohe = vectorizer.transform(X_test['clean_subcategorie
s'].values)
print("After vectorizations")
print(X_train_clean_subcategories_ohe.shape, y_train.shape)
print(X_cv_clean_subcategories_ohe.shape, y_cv.shape)
print(X_test_clean_subcategories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 28) (22445,)
(11055, 28) (11055,)
(16500, 28) (16500,)
['appliedsciences', 'charactereducation', 'civics_government', 'coll
ege_careerprep', 'communityservice', 'earlydevelopment', 'economic
s', 'environmentalscience', 'esl', 'extracurricular', 'financiallite
racy', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'hea
lth_wellness', 'history_geography', 'literacy', 'literature_writin
g', 'mathematics', 'music', 'nutritioneducation', 'other', 'parentin
volvement', 'performingarts', 'socialsciences', 'specialneeds', 'tea
msports', 'visualarts']
====================================================================
================================
```

In [138]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['price'].values.reshape(1,-1))
X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,- 1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape (1,-1))
X_train_price_norm =X_train_price_norm.reshape(-1,1)
X_cv_price_norm = X_cv_price_norm.reshape(-1,1)
X_test_price_norm = X_test_price_norm.reshape(-1,1)
print("After vectorizations")
#print(X_train_price_norm_1.shape, y_train.shape)
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,)
====================================================================
================================
```

In [139]:

```
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead: # array=[105.
22 215.96 96.01 ... 368.98 80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1) if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.re
shape(1,-1))
X_train_teacher_number_of_previously_posted_projects_norm = normalizer.transform
(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_cv_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X_
cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_test_teacher_number_of_previously_posted_projects_norm = normalizer.transform(
X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_train_teacher_number_of_previously_posted_projects_norm = X_train_teacher_numb
er_of_previously_posted_projects_norm.reshape(-1,1)
X_cv_teacher_number_of_previously_posted_projects_norm = X_cv_teacher_number_of_
previously_posted_projects_norm.reshape(-1,1)
X_test_teacher_number_of_previously_posted_projects_norm = X_test_teacher_number
_of_previously_posted_projects_norm.reshape(-1,1)
print("After vectorizations")
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.s
hape)
print(X_cv_teacher_number_of_previously_posted_projects_norm.shape, y_cv.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.sha
pe)
print("="*100)
```

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

In [ ]:

```
import nltk
nltk.download('all')
```

In [140]:

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA
sentiment_neg, sentiment_neu, sentiment_pos, sentiment_comp = [], [], [], []

for sentence in tqdm(data['essay'].values):
  sid = SIA()
  sentiment_dict=sid.polarity_scores(sentence)
  sentiment_neg.append(sentiment_dict['neg'])
  sentiment_neu.append(sentiment_dict['neu'])
  sentiment_pos.append(sentiment_dict['pos'])
  sentiment_comp.append(sentiment_dict['compound'])
```

```
100%|██████████| 50000/50000 [09:33<00:00, 87.15it/s]
```

In [141]:

```
import numpy as np
neg = np.array((sentiment_neg))
pos = np.array((sentiment_pos))
neu = np.array((sentiment_neu))
comp = np.array((sentiment_comp))
```

In [142]:

```
X_train_neg   = neg[0:X_train.shape[0]].reshape(-1,1)
X_train_pos   = pos[0:X_train.shape[0]].reshape(-1,1)
X_train_comp  = comp[0:X_train.shape[0]].reshape(-1,1)
X_train_neu   = neu[0:X_train.shape[0]].reshape(-1,1)
```

In [143]:

```
X_cv_neg   = neg[X_train.shape[0]:X_train.shape[0]+X_cv.shape[0]].reshape(-1,1)
X_cv_pos   = pos[X_train.shape[0]:X_train.shape[0]+X_cv.shape[0]].reshape(-1,1)
X_cv_comp  = comp[X_train.shape[0]:X_train.shape[0]+X_cv.shape[0]].reshape(-1,1)
X_cv_neu   = neu[X_train.shape[0]:X_train.shape[0]+X_cv.shape[0]].reshape(-1,1)
```

In [144]:

```
X_test_neg   = neg[X_train.shape[0]+X_cv.shape[0]:].reshape(-1,1)
X_test_pos   = pos[X_train.shape[0]+X_cv.shape[0]:].reshape(-1,1)
X_test_comp  = comp[X_train.shape[0]+X_cv.shape[0]:].reshape(-1,1)
X_test_neu   = neu[X_train.shape[0]+X_cv.shape[0]:].reshape(-1,1)
```

In [146]:

```
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_tr
ain_grade_ohe,X_train_clean_categories_ohe,X_train_clean_subcategories_ohe, X_tr
ain_price_norm,X_train_teacher_number_of_previously_posted_projects_norm, X_trai
n_neg, X_train_pos, X_train_comp, X_train_neu)).tocsr()
X_cr = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_oh
e, X_cv_clean_categories_ohe,X_cv_clean_subcategories_ohe,X_cv_price_norm,X_cv_t
eacher_number_of_previously_posted_projects_norm, X_cv_neg, X_cv_pos, X_cv_comp,
X_cv_neu)).tocsr()
X_te = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_
grade_ohe,X_test_clean_categories_ohe,X_test_clean_subcategories_ohe, X_test_pri
ce_norm,X_test_teacher_number_of_previously_posted_projects_norm, X_test_neg, X_
test_pos, X_test_comp, X_test_neu)).tocsr()
print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 5101) (22445,)
(11055, 5101) (11055,)
(16500, 5101) (16500,)
====================================================================
================================
```

Hyper parameter Tuning

1. Simple for loop (if you are having memory limitations use this)

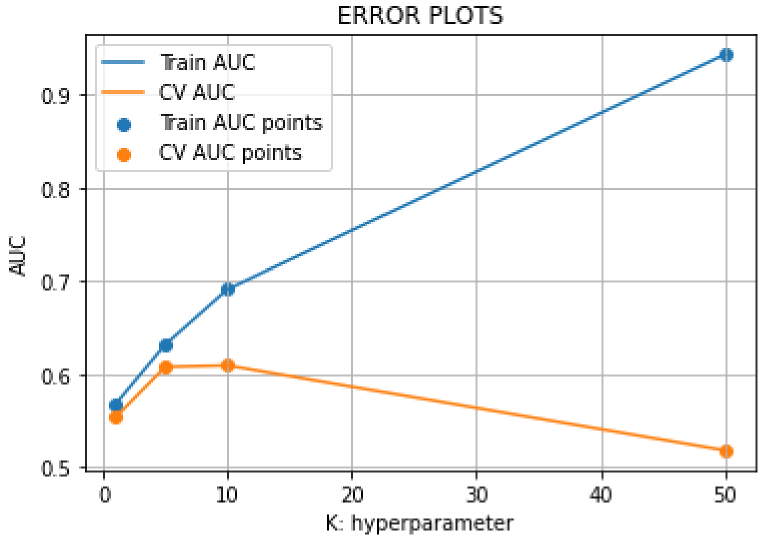
In [147]:

```
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 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 [148]:

```
import matplotlib.pyplot as plt
#from sklearn.neighbors import KNeighborsClassifier
#from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
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, confid
ence values, or non-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 wit h greate
r label.
"""
#hyper paramter tuning (best `depth` in range [1, 5, 10, 50], and the best `min_
samples_split` in range [5, 10, 100, 500]
train_auc = []
cv_auc = []
K=[1, 5, 10, 50]

min_samples_split= [5, 10, 100, 500]

for i in tqdm(K):
  clf = DecisionTreeClassifier(max_depth=i)
  clf.fit(X_tr, y_train)
  y_train_pred = batch_predict(clf, X_tr)
  y_cv_pred = batch_predict(clf, X_cr)
  # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estim
ates of the 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.xscale('log')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 4/4 [00:38<00:00,  9.63s/it]
```



Find AUC and Alpha

In [149]:

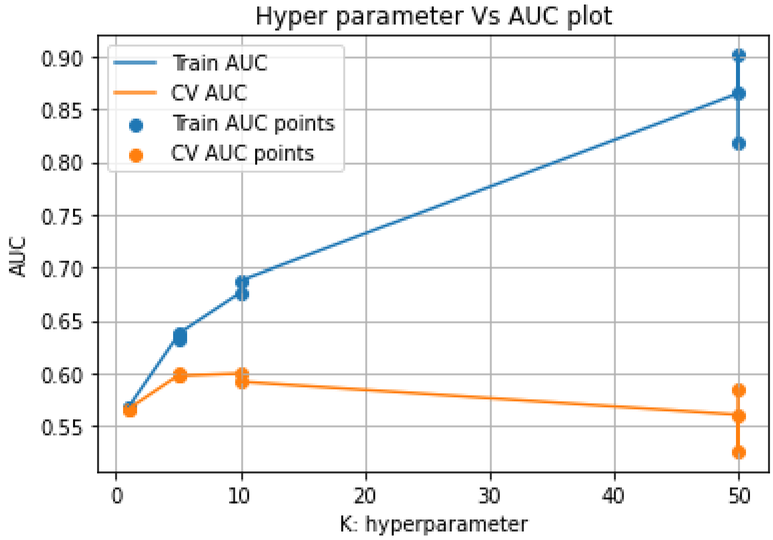
```
score_t_cv = [x for x in cv_auc]
opt_t_cv = K[score_t_cv.index(max(score_t_cv))]
print("Maximum AUC score of cv is:" + ' ' + str(max(score_t_cv)))
print("Corresponding alpha value of cv is:",opt_t_cv, '\n')
best_alp=opt_t_cv
print(best_alp)
```

```
Maximum AUC score of cv is: 0.6090948282418943
Corresponding alpha value of cv is: 10

10
```

1. RandomSearchCV method for hyper-parameter tunning

In [150]:

```
from sklearn.model_selection import RandomizedSearchCV
K=[1, 5, 10, 50]
min_samples_split= [5, 10, 100, 500]
#C = uniform(0.00001,10000)
#parameters = {'alpha':uniform(0.00001,10000)}
parameters = dict(max_depth=K, min_samples_split=min_samples_split)
clf = RandomizedSearchCV(DecisionTreeClassifier(), parameters, cv=3, scoring='ro
c_auc',return_train_score=True)
clf.fit(X_tr, y_train)
results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_max_depth'])
train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K = results['param_max_depth']
plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4 084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train _auc_st
d,alpha=0.2,color='darkblue')
plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4 084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alp ha=0.2,c
olor='darkorange')
plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points') #plt.xscale('log')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
results.head()
```



Out[150]:

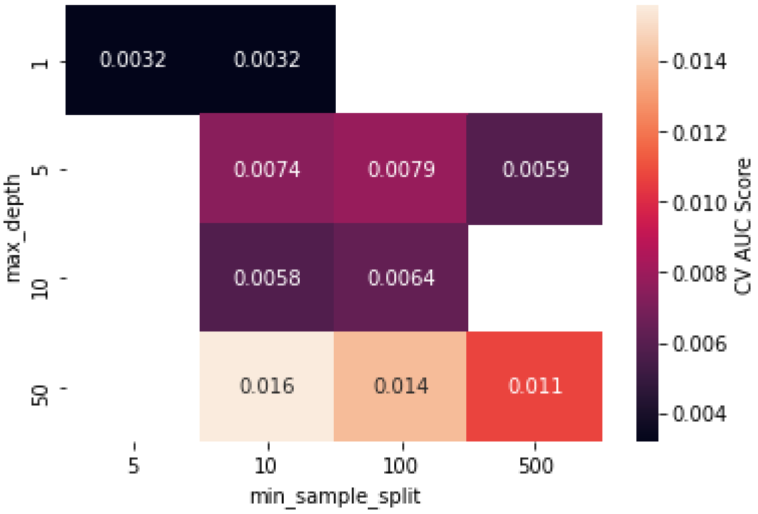
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_split	pa
0	0.327779	0.004640	0.012290	0.001432	10	
5	0.321918	0.001566	0.011024	0.000099	5	
1	1.504600	0.029637	0.011337	0.000137	100	
7	1.412528	0.008292	0.011675	0.000154	500	
9	1.535199	0.037339	0.011266	0.000287	10	

In [151]:

```
#copied from https://stackoverflow.com/questions/45470882/x-y-z-array-data-to-heatmap
x = results['param_min_samples_split'].values
y = results['param_max_depth'].values
z = results['std_test_score'].values

df = pd.DataFrame.from_dict(np.array([x,y,z]).T)
X_value = 'min_sample_split'
Y_value =  'max_depth'
Z_value =  'AUC Scorre'
df.columns = [X_value,Y_value,Z_value]
df[Z_value] = pd.to_numeric(df[Z_value])
pivotted= df.pivot(Y_value,X_value,Z_value)
sns.heatmap(pivotted, annot=True, cbar_kws={'label': 'CV AUC Score'})
```

Out[151]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2386e61450>
```

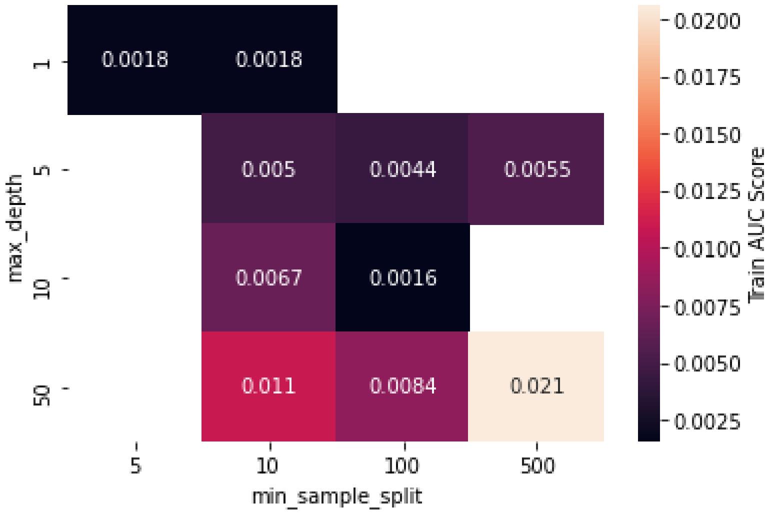


In [152]:

```
#copied from https://stackoverflow.com/questions/45470882/x-y-z-array-data-to-he
atmap
x = results['param_min_samples_split'].values
y = results['param_max_depth'].values
z = results['std_train_score'].values
df = pd.DataFrame.from_dict(np.array([x,y,z]).T)
X_value = 'min_sample_split'
Y_value =  'max_depth'
Z_value =  'AUC Scorre'
df.columns = [X_value,Y_value,Z_value]
df[Z_value] = pd.to_numeric(df[Z_value])
pivotted= df.pivot(Y_value,X_value,Z_value)
sns.heatmap(pivotted, annot=True,cbar_kws={'label': 'Train AUC Score'})
```

Out[152]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2388525fd0>
```



In [153]:

```
print(clf.best_params_)
```

```
{'min_samples_split': 100, 'max_depth': 10}
```

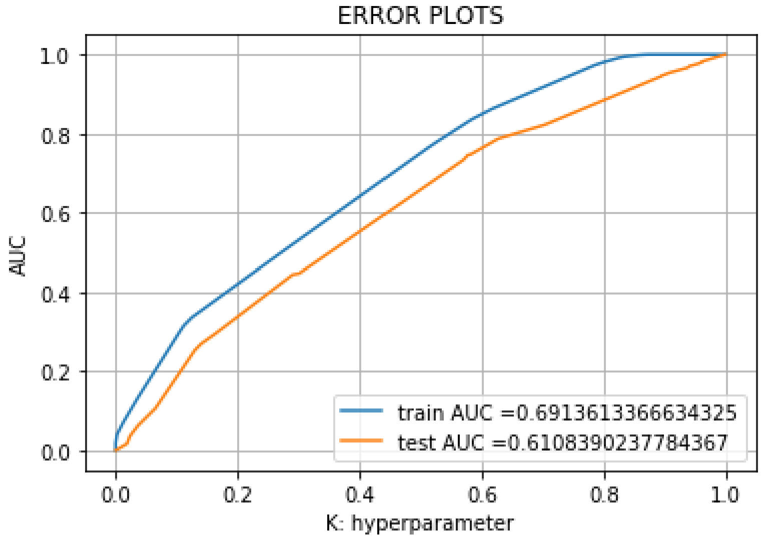
In [155]:

```
best_depth = 10
```

Apply DT using best parameters

In [156]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc _curve.h
tml#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
clf = DecisionTreeClassifier(max_depth=best_depth)
clf.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probabilit y estima
tes of the positive class
# not the predicted outputs
y_train_pred = batch_predict(clf, X_tr)
y_test_pred = batch_predict(clf, X_te)
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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [158]:

```
# we are writing our own function for predict, with defined thresold
# 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
```

In [159]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================
================================
the maximum value of tpr*(1-fpr) 0.372879957795789‍14 for threshold
0.852
Train confusion matrix
[[ 1755  1840]
 [ 4452 14398]]
Test confusion matrix
[[ 1137  1505]
 [ 3703 10155]]
```

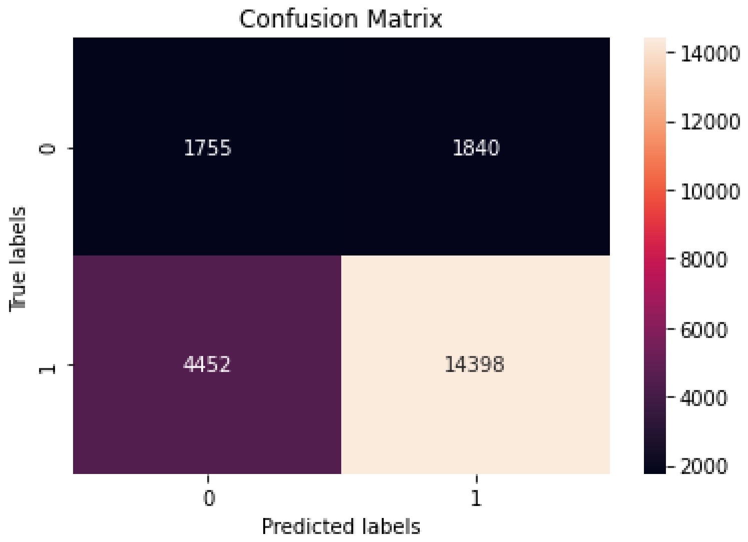
Train Confusion Matrix

In [160]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t
)), annot = True, ax = ax, fmt = 'g')

# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
```

Out[160]:

```
Text(0.5, 1.0, 'Confusion Matrix')
```



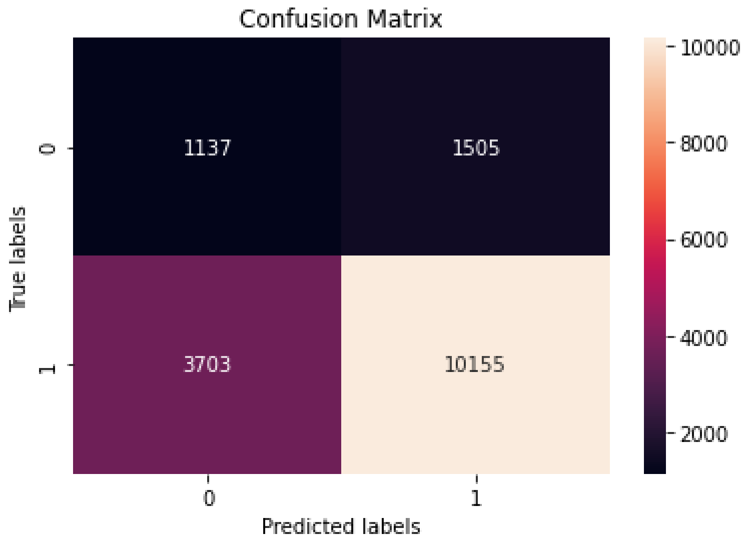
Test Confusion Matrix

In [161]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)),
annot=True, ax = ax,fmt='g')

# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
```

Out[161]:

```
Text(0.5, 1.0, 'Confusion Matrix')
```



Apply DT with TFIDF W2V

In [162]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-t
o-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('/content/drive/MyDrive/DT/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [163]:

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

In [164]:

```python
# average Word2Vec
# compute average word2vec for each review.
X_train_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_essay_tfidf_w2v_vectors.append(vector)

X_cv_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_cv_essay_tfidf_w2v_vectors.append(vector)

X_test_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_essay_tfidf_w2v_vectors.append(vector)

#print(len(tfidf_w2v_vectors))
#print(len(tfidf_w2v_vectors[0]))
```

```
100%|██████████| 22445/22445 [00:53<00:00, 420.73it/s]
100%|██████████| 11055/11055 [00:25<00:00, 428.43it/s]
100%|██████████| 16500/16500 [00:38<00:00, 425.96it/s]
```

In [165]:

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

print("="*100)
```

```
(22445, 8) (22445,)
(11055, 8) (11055,)
(16500, 8) (16500,)
====================================================================
================================
```

Concatinate TFIDF features

In [166]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfidf_w2v_vectors, X_train_state_ohe, X_train_teach
er_ohe, X_train_grade_ohe,X_train_clean_categories_ohe,X_train_clean_subcategori
es_ohe, X_train_price_norm,X_train_teacher_number_of_previously_posted_projects_
norm)).tocsr()
X_cr = hstack((X_cv_essay_tfidf_w2v_vectors, X_cv_state_ohe, X_cv_teacher_ohe, X
_cv_grade_ohe, X_cv_clean_categories_ohe,X_cv_clean_subcategories_ohe,X_cv_price
_norm,X_cv_teacher_number_of_previously_posted_projects_norm)).tocsr()
X_te = hstack((X_test_essay_tfidf_w2v_vectors, X_test_state_ohe, X_test_teacher_
ohe, X_test_grade_ohe,X_test_clean_categories_ohe,X_test_clean_subcategories_ohe
, X_test_price_norm,X_test_teacher_number_of_previously_posted_projects_norm)).t
ocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 397) (22445,)
(11055, 397) (11055,)
(16500, 397) (16500,)
====================================================================
================================
```

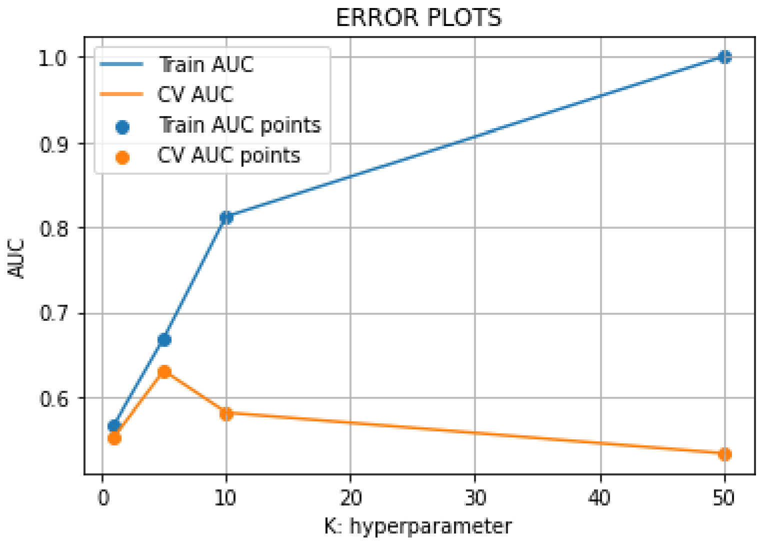
Hyperparameter Tunning

1: Simple loop method

In [167]:

```
train_auc = []
cv_auc = []
K=[1, 5, 10, 50]
min_samples_split= [5, 10, 100, 500]

for i in tqdm(K):
    clf = DecisionTreeClassifier(max_depth=i)
    clf.fit(X_tr, y_train)

    y_train_pred = batch_predict(clf, X_tr)
    y_cv_pred = batch_predict(clf, X_cr)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability est
imates of the 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.xscale('log')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
100%|██████████| 4/4 [01:10<00:00, 17.73s/it]
```



In [168]:

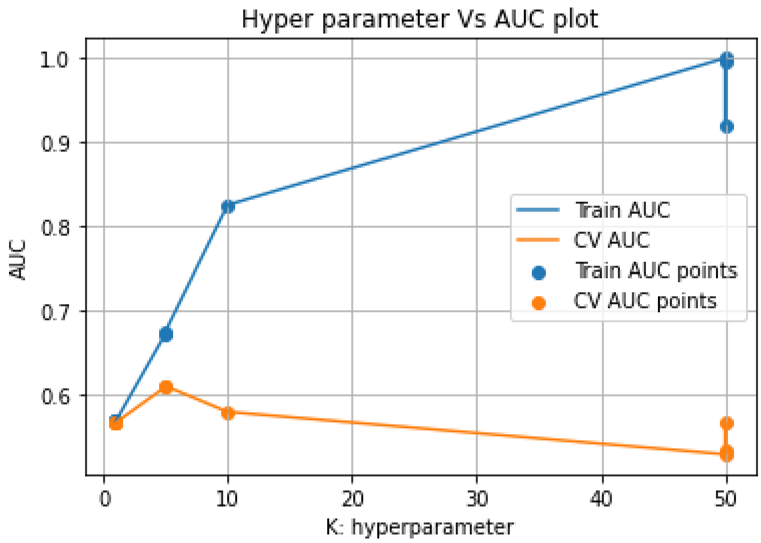
```
score_t_cv = [x for x in cv_auc]
opt_t_cv = K[score_t_cv.index(max(score_t_cv))]
print("Maximum AUC score of cv is:" + ' ' + str(max(score_t_cv)))
print("Corresponding alpha value of cv is:",opt_t_cv, '\n')
best_alp=opt_t_cv
print(best_alp)
```

```
Maximum AUC score of cv is: 0.6307121625609619
Corresponding alpha value of cv is: 5

5
```

2: Random Search

In [169]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier

K=[1, 5, 10, 50]
min_samples_split= [5, 10, 100, 500]
#C = uniform(0.00001,10000)
#parameters = {'alpha':uniform(0.00001,10000)}
parameters = dict(max_depth=K,min_samples_split=min_samples_split)

clf = RandomizedSearchCV(DecisionTreeClassifier(), parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf.fit(X_tr, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_max_depth'])


train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_max_depth']


plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



Out[169]:

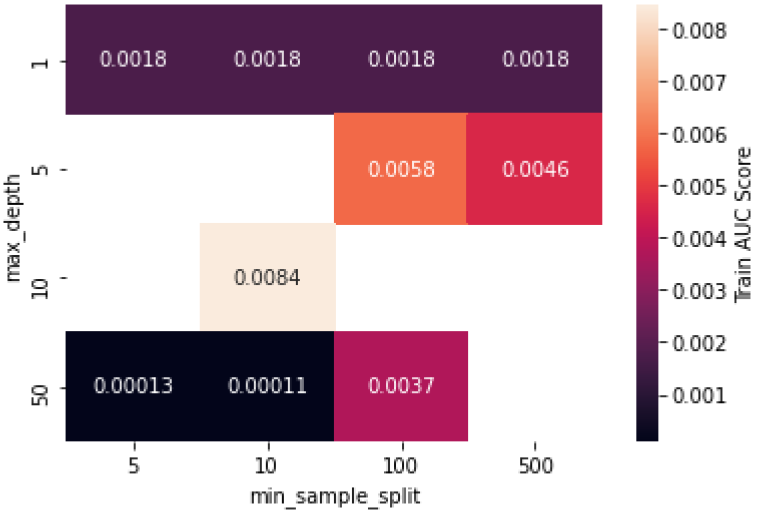
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_split	pa
0	0.807806	0.011675	0.017736	0.000772	100	
1	0.796574	0.004971	0.016894	0.000534	5	
6	0.790915	0.001034	0.016768	0.000856	500	
9	0.799738	0.002487	0.017998	0.001913	10	
2	3.740514	0.021095	0.017043	0.000369	500	

In [170]:

```
#copied from https://stackoverflow.com/questions/45470882/x-y-z-array-data-to-he
atmap
x = results['param_min_samples_split'].values
y = results['param_max_depth'].values
z = results['std_train_score'].values

df = pd.DataFrame.from_dict(np.array([x,y,z]).T)
X_value = 'min_sample_split'
Y_value =  'max_depth'
Z_value =  'AUC Scorre'
df.columns = [X_value,Y_value,Z_value]
df[Z_value] = pd.to_numeric(df[Z_value])

pivotted= df.pivot(Y_value,X_value,Z_value)
sns.heatmap(pivotted, annot=True,cbar_kws={'label': 'Train AUC Score'})
```

Out[170]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f238948d0d0>
```

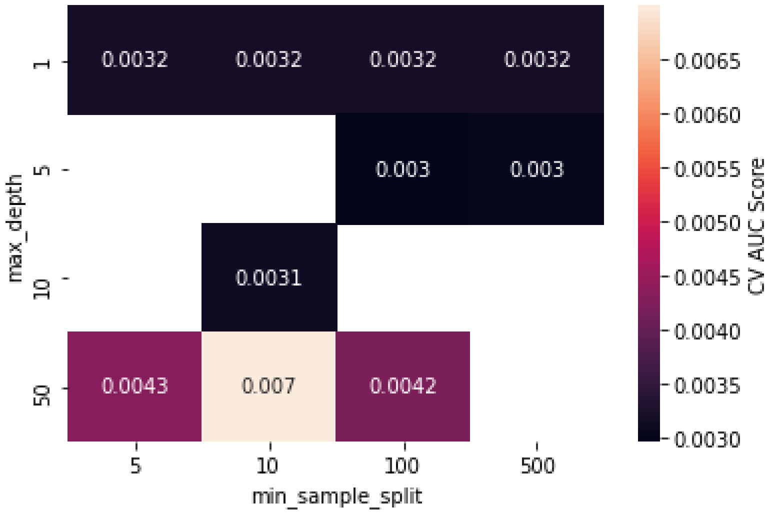


In [171]:

```
#copied from https://stackoverflow.com/questions/45470882/x-y-z-array-data-to-he
atmap
x = results['param_min_samples_split'].values
y = results['param_max_depth'].values
z = results['std_test_score'].values

df = pd.DataFrame.from_dict(np.array([x,y,z]).T)
X_value = 'min_sample_split'
Y_value =  'max_depth'
Z_value =  'AUC Scorre'
df.columns = [X_value,Y_value,Z_value]
df[Z_value] = pd.to_numeric(df[Z_value])

pivotted= df.pivot(Y_value,X_value,Z_value)
sns.heatmap(pivotted, annot=True,cbar_kws={'label': 'CV AUC Score'})
```

Out[171]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f238977bb50>
```



In [172]:

```
print(clf.best_params_)
```

```
{'min_samples_split': 100, 'max_depth': 5}
```

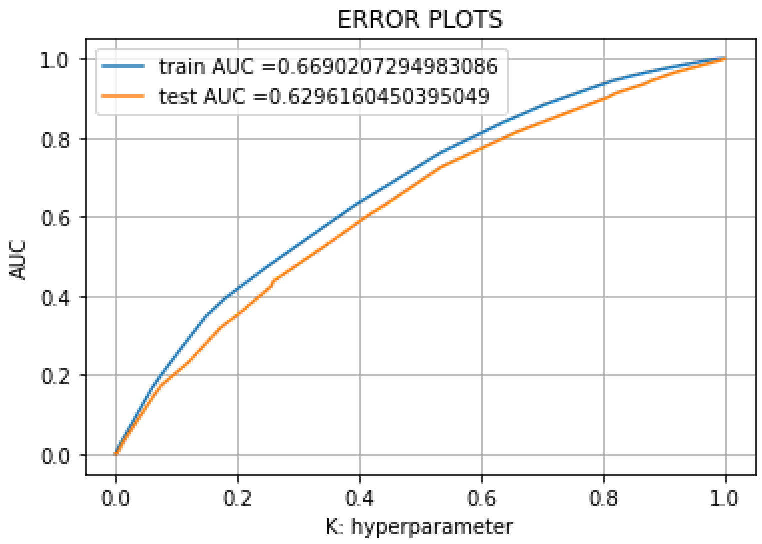
In [173]:

```
best_depth = 5
```

Testing the performance of ROC curve using test data

In [174]:

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


clf = DecisionTreeClassifier(max_depth=best_depth)
clf.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimat
es of the positive class
# not the predicted outputs

y_train_pred = batch_predict(clf, X_tr)
y_test_pred = batch_predict(clf, X_te)

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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Representation of results

In [175]:

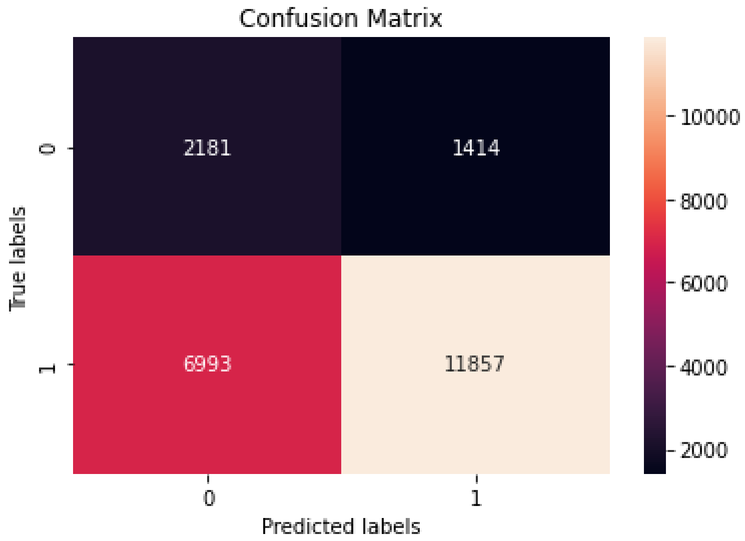
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t )))
```

```
====================================================================
================================
the maximum value of tpr*(1-fpr) 0.3816104300476273 for threshold 0.
849
Train confusion matrix
[[ 2181  1414]
 [ 6993 11857]]
Test confusion matrix
[[1529 1113]
 [5409 8449]]
```

Train COnfusion Matrix

In [176]:

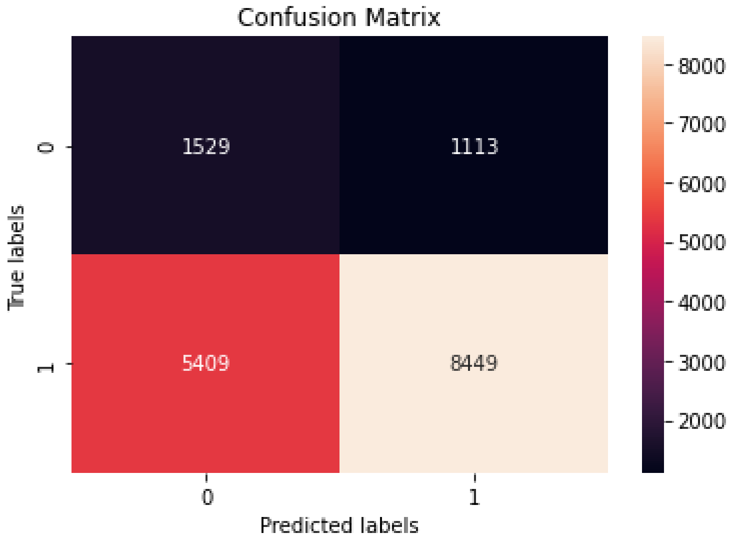
```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t
)), annot=True, ax = ax,fmt='g');

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



Test COnfusion Matrix

In [177]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix
import seaborn as sns
import matplotlib.pyplot as plt
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)),
annot=True, ax = ax,fmt='g');

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



Feature Importance

In [178]:

```
from sklearn.metrics import accuracy_score
from scipy.sparse import csr_matrix
count_vect = CountVectorizer()
count_vect.fit_transform(X_train['essay'].values)

depth_best =5

clf = DecisionTreeClassifier(max_depth=depth_best)
clf = clf.fit(X_tr, y_train)

feat_importance = clf.tree_.compute_feature_importances(normalize=True)
print("feat importance = " + str(feat_importance))
```

```
feat importance = [0.         0.04508087 0.         0.         0.111
68435 0.
 0.03730294 0.         0.         0.         0.01691372 0.01329597
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.04455224 0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.01936384
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.03403704
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.02055573 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.02569561 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.038767   0.
 0.         0.04252611 0.         0.         0.01255976 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.09052367 0.         0.         0.         0.
 0.         0.04888974 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.02572537 0.         0.
 0.         0.         0.         0.         0.02814606 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.061939
 0.         0.         0.         0.         0.         0.
 0.         0.         0.00875642 0.         0.         0.
 0.0196818  0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.00698437 0.         0.         0.         0.         0.02081233
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.01785451
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.20835156
 0.        ]
```

In [179]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer","Model","HYPER PARAMETER-depth","AUC"]

x.add_row(["TFIDF","DT-RandomSearch",10,0.6090948282418943])
x.add_row(["TFIDF W2V","DT-Auto",5,0.6307121625609619])


print(x)
```

```
+------------+-----------------+-----------------------+------------
--------+
| Vectorizer |      Model      | HYPER PARAMETER-depth |         AUC
|
+------------+-----------------+-----------------------+------------
--------+
|   TFIDF    | DT-RandomSearch |           10          | 0.609094828
2418943 |
| TFIDF W2V  |     DT-Auto     |           5           | 0.630712162
5609619 |
+------------+-----------------+-----------------------+------------
--------+
```