Assignment 9: GBDT

Response Coding: Example

Train Data

State	class
A	0
B	1
C	1
A	0
A	1
B	1
A	0
A	1
C	1
C	0

Resonse table(only from train)

State	Class=0	Class=1
A	3	2
B	0	2
C	1	2

Encoded Train Data

State_0	State_1	class
3/5	2/5	0
0/2	2/2	1
1/3	2/3	1
3/5	2/5	0
3/5	2/5	1
0/2	2/2	1
3/5	2/5	0
3/5	2/5	1
1/3	2/3	1
1/3	2/3	0

Test Data

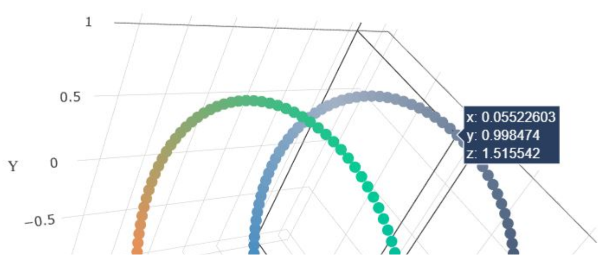
State
A
C
D
C
B
E

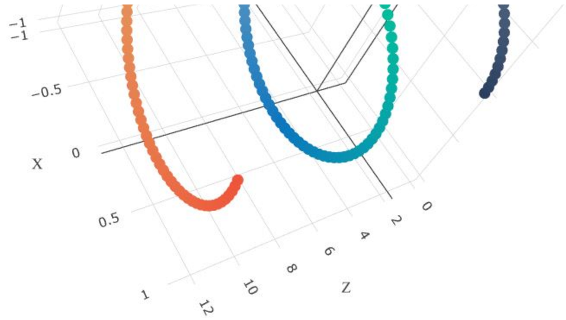
Encoded Test Data

State_0	State_1
3/5	2/5
1/3	2/3
1/2	1/2
1/3	2/3
0/2	2/2
1/2	1/2

> The response tabel is built only on train dataset. For a category which is not there in train data and present in test data, we will encode them with default values Ex: in our test data if have State: D then we encode it as [0.5, 0.05]

1. **Apply GBDT on these feature sets**
   - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check the bellow example, include all 4 values as 4 features)
   - Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (Consider any two hyper parameters)**
   - Find the best hyper parameter which will give the maximum AUC value
   - find the best hyper paramter using k-fold cross validation/simple cross validation data
   - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task
3. **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

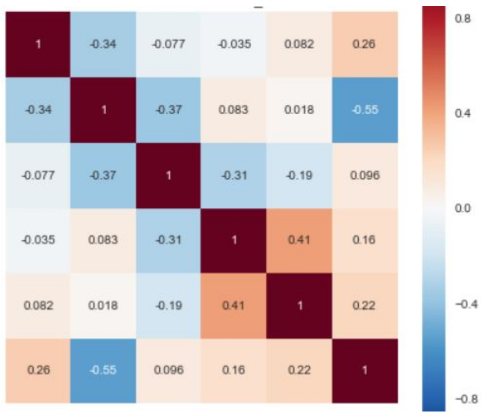




with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

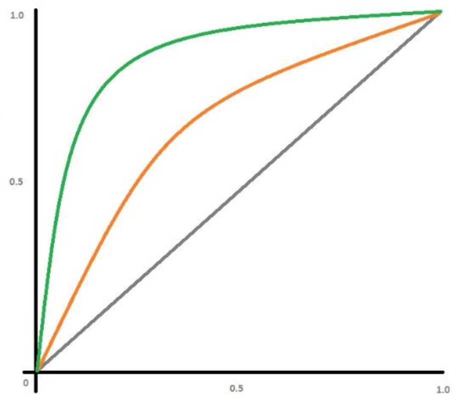
or

- 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](#) with rows as **n_estimators**, 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](#) with predicted and original labels of test data points

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

4. You need to summarize the results at the end of the notebook, summarize it in the table format

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79

TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

1. GBDT (xgboost/lightgbm)

In [1]:

```
import pickle
#please use below code to load glove vectors
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [3]:

```
# Imports

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

import pandas as pd
import numpy as np
import collections
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import Normalizer
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_curve, auc, roc_auc_score
from tqdm import tqdm
from prettytable import PrettyTable
from pandas_ml import ConfusionMatrix
from sklearn.model_selection import GridSearchCV
import seaborn as sns

import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

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

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\hp\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[3]:

```
True
```

1.1 Loading Data

In [4]:

```
import pandas
data = pandas.read_csv('preprocessed_data.csv', nrows = 50000)

# Seprate X and y

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

Out[4]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories
0	ca	mrs	grades_prek_2	53	math_science

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

In [5]:

```
# Split into trainig and test set and again split the training set to train and cv

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

1.3 Make Data Model Ready: Sentiment Score

In [6]:

```
# Function to be applied with indevidual train, cv and test data
def getSentimentVector(essays):
    """This function gives the sentiment measures for text data"""

    sid = SentimentIntensityAnalyzer()
    output = dict()
    # SentimentIntensityAnalyzer results values for below keys
    columns = ['neg', 'neu', 'pos', 'compound']
    neg = []
    neu = []
    pos = []
    compound = []

    # Get scores for each essays
    for essay in essays:
        ss = sid.polarity_scores(essay)
        values = list(ss.values())
        neg.append(values[0])
        neu.append(values[1])
        pos.append(values[2])
        compound.append(values[3])

    # combine all the data and return
    data = []
    for value in list(zip(neg, neu, pos, compound)):
        data.append(list(value))

    output['columns'] = columns
    output['values'] = np.array(data)
    return output
```

1.4 Make Data Model Ready: encoding eassay

IFIDF vectorizer for text(essay) vectorization

In [7]:

```
def encodeEssayTFIDF(trainText, testText):
    """This function returns the encoded vectors for train, cv and test data with
TfidfVectorizer"""
```

```
    output = dict()
    vectorizer = TfidfVectorizer(min_df = 10, max_features = 5000)

    # Fit the vectorizer with train data and trasnform all train, cv and test texts
    train_vec = vectorizer.fit_transform(trainText)
    output['columns'] = vectorizer.get_feature_names()
    test_vec = vectorizer.transform(testText)

    # Add all enocoded vectorized data and bows into a dict to return it
    output['train_vec'] = train_vec
    output['test_vec'] = test_vec
    output['columns'] = vectorizer.get_feature_names()

    return output
```

TFIDF_W2V vectorization of text(essay)

In [8]:

```
from scipy.sparse import coo_matrix
def getW2C(texts, tfidf_words, dictionary):
    """This function returns W2V response for given TFIDF data and texts"""

    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(texts): # 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
                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 coo_matrix(tfidf_w2v_vectors)
```

In [9]:

```
def encodeEssayW2VTFIDF(trainText, testText):
    """This function returns the encoded vectors for train, cv and test data with
TfidfVectorizer"""

    output = dict()

    # Got TFIDF model and use it with W2V
    tfidf = TfidfVectorizer()
    tfidf.fit(trainText)
    dictionary = dict(zip(tfidf.get_feature_names(), list(tfidf.idf_)))
    tfidf_words = set(tfidf.get_feature_names())

    # Get w2V vectors for train, test and cv
    train_vec = getW2C(trainText, tfidf_words, dictionary)
    test_vec = getW2C(testText, tfidf_words, dictionary)

    # As w2V doesnt give the feature names as it only gives d-dim vector for a word, we are
manually adding columns
    columns = []
    for i in range(300):
        c = 'w' + str(i + 1)
        columns.append(c)

    # Retun the results in dict format
    output['columns'] = columns
    output['train_vec'] = train_vec
    output['test_vec'] = test_vec

    return output
```

1.5 Make Data Model Ready: encoding numerical, categorical features

Apply Probability numerical vectorizing for categorical values

In [10]:

```
def encodeCategoricalValues(train_cat, test_cat, y_train):
    """This function encodes the categorical values to a vector containing probability values for
classes"""

    # Maintain a dictionary of unique categories and its counts in training data
    category_counts = [(k,v) for k,v in collections.Counter(train_cat).items()]
    # Collect unique classes
    classes = np.unique(y_train)

    # Create a dataframe using x and y (use to check for category having some class level)
    df = pd.DataFrame({'x': train_cat, 'y': y_train})

    # Initialize vectors of categories (maintain categories as keys and list of probabilities of
classes as value)
    vec_cat = dict()

    # Check for each category
    for k, v in category_counts:
        vec = []
        # Check for each classes
        for cls in classes:
            cls_count = df[(df['x'] == k) & (df['y'] == cls)].shape[0]
            vec.append(round((cls_count / v), 3))
        vec_cat[k] = vec

    # Create columns for the vector
    columns = ['class_' + str(cls) for cls in classes]

    # Vectorize training data
    train_vec = []
    for value in train_cat:
        train_vec.append(vec_cat[value])

    # Vectorize test data
    test_vec = []
    for value in test_cat:
        if (vec_cat.get(value) == None):
            test_vec.append([0.5, 0.5])
        else:
            test_vec.append(vec_cat[value])

    # Return the vectors
    result = dict()
    result['columns'] = columns
    result['train_vec'] = np.array(train_vec)
    result['test_vec'] = np.array(test_vec)

    return result
```

Normalize the numerical data

In [11]:

```
def numericNormalizer(train_val, test_val, column):
    """This function normalizes numerical values"""
    normalizer = Normalizer()
    output = dict()

    normalizer.fit(train_val.reshape(-1,1))
    x_train_val = normalizer.transform(train_val.reshape(-1,1))
    x_test_val = normalizer.transform(test_val.reshape(-1,1))

    # Add all enocoded vectorized data and bows into a dict to return it
    output['train_val'] = x_train_val
    output['test_val'] = x_test_val
    output['column'] = column
```

```
    return output
```

In [12]:

```
from scipy.sparse import hstack

# Transform data (text, categotical and numerical data)
def vectorizeDataset(X_train, X_test, y_train, setType):
    """This function vectorizes the feature values"""

    columns = []

    trainEssay = X_train['essay'].values
    testEssay = X_test['essay'].values
    trainState = X_train['school_state'].values
    testState = X_test['school_state'].values
    trainPrefix = X_train['teacher_prefix'].values
    testPrefix = X_test['teacher_prefix'].values
    trainGrade = X_train['project_grade_category'].values
    testGrade = X_test['project_grade_category'].values
    trainCategory = X_train['clean_categories'].values
    testCategory = X_test['clean_categories'].values
    trainSubCategories = X_train['clean_subcategories'].values
    testSubCategories = X_test['clean_subcategories'].values
    trainPrevProjects = X_train['teacher_number_of_previously_posted_projects'].values
    testPrevProjects = X_test['teacher_number_of_previously_posted_projects'].values
    trainPrice = X_train['price'].values
    testPrice = X_test['price'].values

    # vectorize essays
    output_essay = dict()

    if (setType == 1):
        output_essay = encodeEssayTFIDF(trainEssay, testEssay)
    elif (setType == 2):
        output_essay = encodeEssayW2VTFIDF(trainEssay, testEssay)


    x_train_essay = output_essay['train_vec']
    x_test_essay = output_essay['test_vec']
    columns += output_essay['columns']

    # Get sentiment scores for essays
    output_sentiment = getSentimentVector(trainEssay)
    x_train_sentiment = output_sentiment['values']
    x_test_sentiment = getSentimentVector(testEssay)['values']
    columns += output_sentiment['columns']



    # One hot encode for school state
    output_state = encodeCategoricalValues(trainState, testState, y_train)
    x_train_state = output_state['train_vec']
    x_test_state = output_state['test_vec']
    columns += output_state['columns']

    # One hot encode for teacher prefix
    output_prefix = encodeCategoricalValues(trainPrefix, testPrefix, y_train)
    x_train_prefix = output_prefix['train_vec']
    x_test_prefix = output_prefix['test_vec']
    columns += output_prefix['columns']

    # One hot encode fot project grades
    output_grade = encodeCategoricalValues(trainGrade, testGrade, y_train)
    x_train_grade = output_grade['train_vec']
    x_test_grade = output_grade['test_vec']
    columns += output_grade['columns']

    # One hot encode fot project categories
    output_Category = encodeCategoricalValues(trainCategory, testCategory, y_train)
    x_train_category = output_Category['train_vec']
    x_test_category = output_Category['test_vec']
    columns += output_Category['columns']

    # One hot encode fot project sub categories
    output_subCategory = encodeCategoricalValues(trainSubCategories, testSubCategories, y_train)
```

```
    x_train_subCategory = output_subCategory['train_vec']
    x_test_subCategory = output_subCategory['test_vec']
    columns += output_subCategory['columns']

    # Normalize previous project numbers
    output_projectNumber = numericNormalizer(trainPrice, testPrice,
'teacher_number_of_previously_posted_projects')
    x_train_number = output_projectNumber['train_val']
    x_test_number = output_projectNumber['test_val']
    columns.append(output_projectNumber['column'])

    # Normalize project price
    output_price = numericNormalizer(trainPrice, testPrice, 'price')
    x_train_price = output_price['train_val']
    x_test_price = output_price['test_val']
    columns.append(output_price['column'])

    #Combine all vectorized features and return final train, cv and test sets and columns

    X_train = hstack((x_train_essay, x_train_sentiment, x_train_state, x_train_prefix, x_train_grad
e, x_train_category, x_train_subCategory,\
                      x_train_number, x_train_price)).tocsr()
    X_test = hstack((x_test_essay, x_test_sentiment, x_test_state, x_test_prefix, x_test_grade, x_t
est_category, x_test_subCategory, \
                     x_test_number, x_test_price)).tocsr()

    print("Final Data matrix with train, cv and test")
    print(X_train.shape, Y_train.shape)
    print(X_test.shape, Y_test.shape)
    print(len(columns))

    return X_train, X_test, columns
```

In [13]:

```
X_Train_set1, X_Test_set1, columns_set1 = vectorizeDataset(X_train, X_test, Y_train, 1)
```

```
Final Data matrix with train, cv and test
(33500, 5016) (33500,)
(16500, 5016) (16500,)
5016
```

In [14]:

```
X_Train_set2, X_Test_set2, columns_set2 = vectorizeDataset(X_train, X_test, Y_train, 2)
```

```
100%|██████████████████████████████████████████████████| 33500/33500 [01:
38<00:00, 338.40it/s]
100%|██████████████████████████████████████████████████| 16500/16500 [00:
47<00:00, 350.21it/s]
```

```
Final Data matrix with train, cv and test
(33500, 316) (33500,)
(16500, 316) (16500,)
316
```

1.5 Appling Models on different kind of featurization as mentioned in the instructions

Apply GBDT on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

In [15]:

```
def getBestModelGBDT(X_train, Y_train):

    gbdt = GradientBoostingClassifier()
    # Hyper parameters max_depth and n_estimators
```

```
    parameters = {'n_estimators': [25, 50, 75, 100], 'max_depth': [1, 3, 5, 7]}

    # Apply GridSearchCV to get auc scores for train and cv data with different hyper parameter va
lues
    clf = GridSearchCV(gbdt, parameters, cv=3, scoring='roc_auc', n_jobs=-1)
    clf.fit(X_train, Y_train)

    # Get the result and plot in 3D (parameters and auc score)
    results = pd.DataFrame.from_dict(clf.cv_results_)
    results = results.sort_values(['param_max_depth', 'param_n_estimators'])
    train_auc= results['mean_train_score']
    cv_auc = results['mean_test_score']
    n_estimators =  results['param_n_estimators']
    max_depth = results['param_max_depth']

    train_auc_plot = go.Scatter3d(x = max_depth, y = n_estimators, z = train_auc, name = 'train auc
')
    cv_auc_plot = go.Scatter3d(x = max_depth, y = n_estimators, z = cv_auc, name = 'cv auc')
    data = [train_auc_plot, cv_auc_plot]

    layout = go.Layout(scene = dict(
            xaxis = dict(title='max_depth'),
            yaxis = dict(title='n_estimators'),
            zaxis = dict(title='AUC'),))

    fig = go.Figure(data=data, layout=layout)
    offline.iplot(fig, filename='3d-scatter-colorscale')
```

In [16]:

```
# For SET-1 data

getBestModelGBDT(X_Train_set1, Y_train)
```

In [17]:

```
# For SET-2 data

getBestModelGBDT(X_Train_set2, Y_train)
```

From the roc-auc scores, we can see the maximum auc of CV having less difference to auc of train is with max_depth as 3 and n-estimators as 25.

Train and model with best hyper parameter

In [18]:

```
# In ROC_AUC scores, there are number of threshold values.
# Among them best one has to be chosed to predict the class levels from the probability scores.
def get_best_threshold(threshoulds, fpr, tpr):
    """This function takes threshould values with TPR and FPR values and calculates best threshoul
d"""
    # Choose best threshould such that TPR is more and FPR is less
    threshould = threshoulds[np.argmax(tpr*(1-fpr))]
    print("best threshould",threshould)
    return threshould

# Predict the class levels with best threshould
def predict_class_levels(proba, threshould):
    """This function takes best threshould value and probability scores and predict class
levels"""
    predicted_class_levels = []
    for i in proba:
        if i>=threshould:
            predicted_class_levels.append(1)
        else:
            predicted_class_levels.append(0)
    return predicted_class_levels
```

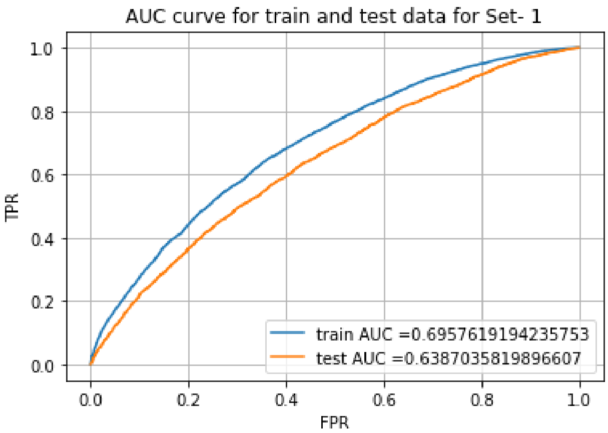
In [19]:

```
# Predict y-test values with best hyper parameter

def bestParamClassificatationAndAUC(X_train, X_test, Y_train, Y_test, max_depth, n_estimators, setT
ype):
    """This function does train and test on a model with best values of hyper paramters,
    and given ROC_AUC curve and value. It returns best threshould and predicted probability values
for train and test data"""

    # Create and train Decision tree with best values of hyper parameters
    gbdt = GradientBoostingClassifier(max_depth = max_depth, n_estimators = n_estimators)
    gbdt.fit(X_train, Y_train)

    # Predict class level probabilities for train and test
    y_proba_train = gbdt.predict_proba(X_train)[:, 1]
```

```
    y_proba_test = gbdt.predict_proba(X_test)[:,1]

    # Get TPR, FPR and threshold values for train and test probability values
    train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_proba_train)
    test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_proba_test)

    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("AUC curve for train and test data for Set- {}".format(setType))
    plt.grid()
    plt.show()

    threshould = get_best_threshold(tr_thresholds, train_fpr, train_tpr)
    y_pred = predict_class_levels(y_proba_test, threshould)

    result = dict()
    result["y_pred"] = y_pred
    result['auc'] = auc(test_fpr, test_tpr)

    return result
```

In [20]:

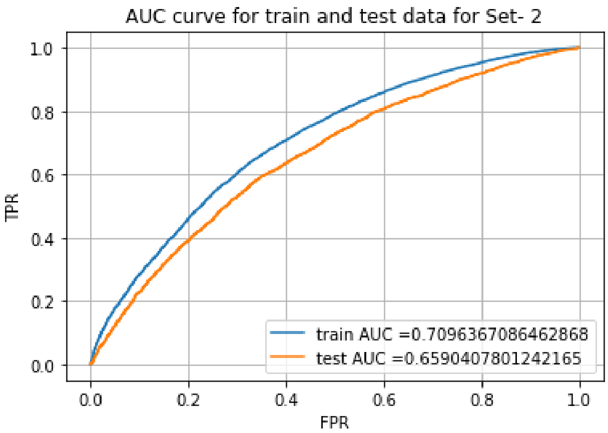
```
# Get classification result with best hyper parameter for SET1

result_set1 = bestParamClassificatationAndAUC(X_Train_set1, X_Test_set1, Y_train, Y_test, 3, 25, 1)
```




```
best threshould 0.8411432916833502
```

In [21]:

```
# Get classification result with best hyper parameter for SET2

result_set2 = bestParamClassificatationAndAUC(X_Train_set2, X_Test_set2, Y_train, Y_test, 3, 25, 2)
```



```
best threshould 0.8328411753683362
```

Train and test auc is more in case of W2V. Also the difference is less as compared to TFIDF.

Confusion matrxix

In [22]:

```
# For SET1

y_pred_set1 =  result_set1['y_pred']
cmSet1 = pd.DataFrame(confusion_matrix(Y_test, y_pred_set1), columns = ['Y_pred-0', 'Y_pred-1'])
cmSet1.index = ['Y_actual-0', 'Y_actual-1']
cmSet1
```

Out[22]:

	Y_pred-0	Y_pred-1
Y_actual-0	1492	1150
Y_actual-1	5070	8788

In [23]:

```
# For SET2

y_pred_set2 =  result_set2['y_pred']
cmSet2 = pd.DataFrame(confusion_matrix(Y_test, y_pred_set2), columns = ['Y_pred-0', 'Y_pred-1'])
cmSet2.index = ['Y_actual-0', 'Y_actual-1']
cmSet2
```

Out[23]:

	Y_pred-0	Y_pred-1
Y_actual-0	1534	1108
Y_actual-1	4798	9060

In [24]:

```
table = PrettyTable()
table.field_names = ["vectorizer", "Model", "Hyper parameter", "AUC"]
table.add_row(["TFIDF", "GBDT", 'max_depth = 3, n-estimators = 25', round(result_set1['auc'],3)])
table.add_row(["TFIDF_W2V", "GBDT", 'max_depth = 3, n-estimators = 25', round(result_set2['auc'],3
)])
print(table)
```

```
+------------+-------+-----------------------------------+-------+
| vectorizer | Model |          Hyper parameter          |  AUC  |
+------------+-------+-----------------------------------+-------+
|   TFIDF    |  GBDT | max_depth = 3, n-estimators = 25 | 0.639 |
| TFIDF_W2V  |  GBDT | max_depth = 3, n-estimators = 25 | 0.659 |
+------------+-------+-----------------------------------+-------+
```

3. Summary

as mentioned in the step 4 of instructions

Here best hyper parameter are max_depth as 3 and n-estimators as 25. We have more auc value in w2v TFIDF case. Its also more as compared to SVM and decision tree. This is the power of ensemeble models.

The base models are of low high bias as we are taking max_depth as 3 but used 25 base models to lower the bais. Here models are having less variance having less variance.