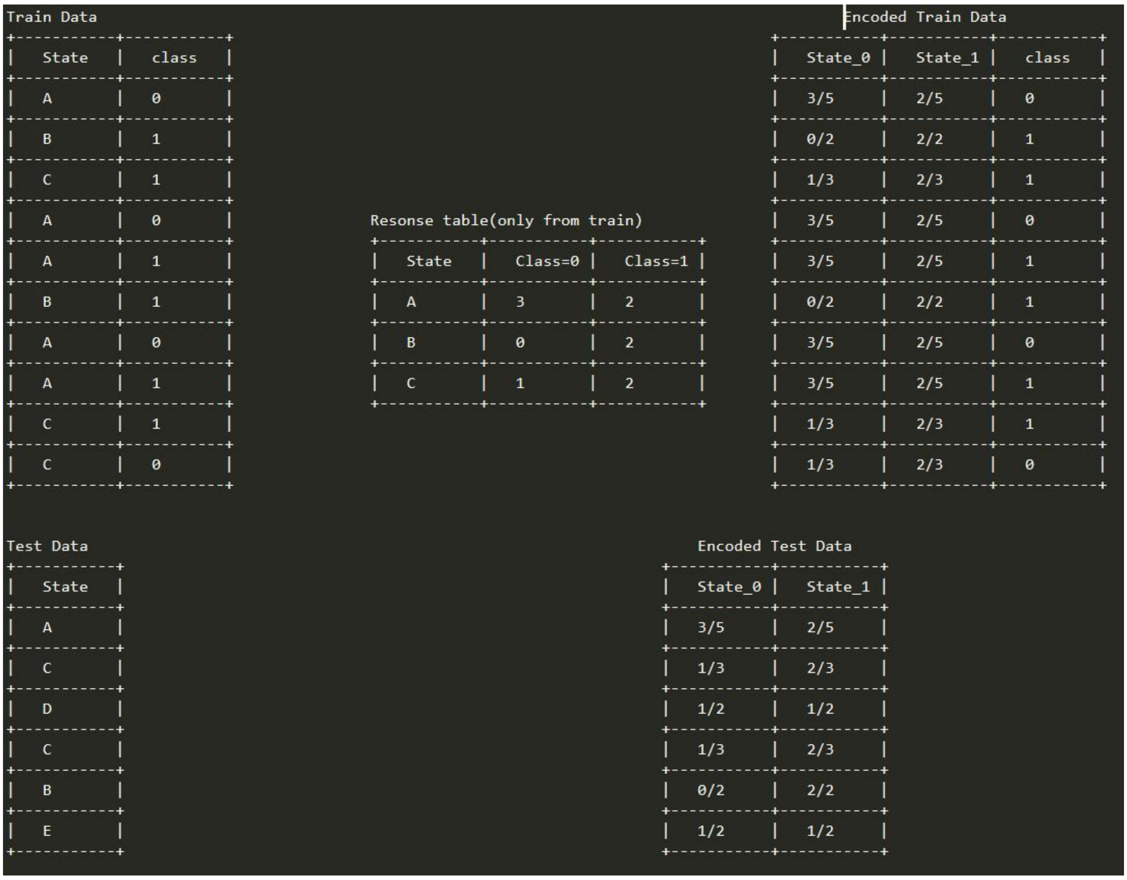
Assignment 9: GBDT

Response Coding: Example

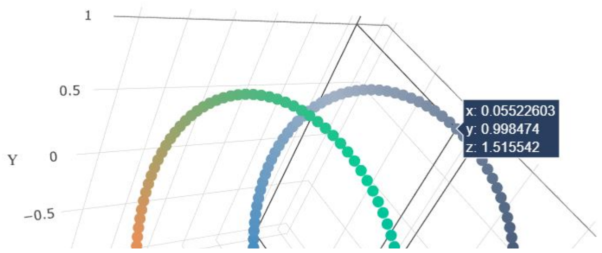


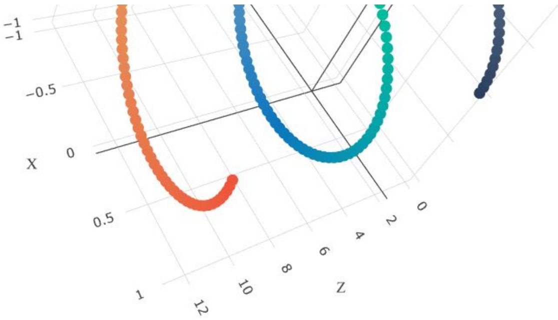
```
Train Data                                                             Encoded Train Data
+----------+----------+                                                +---------+---------+---------+
|  State   |  class   |                                                | State_0 | State_1 |  class  |
+----------+----------+                                                +---------+---------+---------+
|  A       |  0       |                                                |  3/5    |  2/5    |  0      |
+----------+----------+                                                +---------+---------+---------+
|  B       |  1       |                                                |  0/2    |  2/2    |  1      |
+----------+----------+                                                +---------+---------+---------+
|  C       |  1       |                                                |  1/3    |  2/3    |  1      |
+----------+----------+                                                +---------+---------+---------+
|  A       |  0       |     Resonse table(only from train)             |  3/5    |  2/5    |  0      |
+----------+----------+     +------------+------------+------------+   +---------+---------+---------+
|  A       |  1       |     |  State     |  Class=0   |  Class=1   |   |  3/5    |  2/5    |  1      |
+----------+----------+     +------------+------------+------------+   +---------+---------+---------+
|  B       |  1       |     |  A         |  3         |  2         |   |  0/2    |  2/2    |  1      |
+----------+----------+     +------------+------------+------------+   +---------+---------+---------+
|  A       |  0       |     |  B         |  0         |  2         |   |  3/5    |  2/5    |  0      |
+----------+----------+     +------------+------------+------------+   +---------+---------+---------+
|  A       |  1       |     |  C         |  1         |  2         |   |  3/5    |  2/5    |  1      |
+----------+----------+     +------------+------------+------------+   +---------+---------+---------+
|  C       |  1       |                                                |  1/3    |  2/3    |  1      |
+----------+----------+                                                +---------+---------+---------+
|  C       |  0       |                                                |  1/3    |  2/3    |  0      |
+----------+----------+                                                +---------+---------+---------+

Test Data                                                  Encoded Test Data
+----------+                                                +---------+---------+
|  State   |                                                | State_0 | State_1 |
+----------+                                                +---------+---------+
|  A       |                                                |  3/5    |  2/5    |
+----------+                                                +---------+---------+
|  C       |                                                |  1/3    |  2/3    |
+----------+                                                +---------+---------+
|  D       |                                                |  1/2    |  1/2    |
+----------+                                                +---------+---------+
|  C       |                                                |  1/3    |  2/3    |
+----------+                                                +---------+---------+
|  B       |                                                |  0/2    |  2/2    |
+----------+                                                +---------+---------+
|  E       |                                                |  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

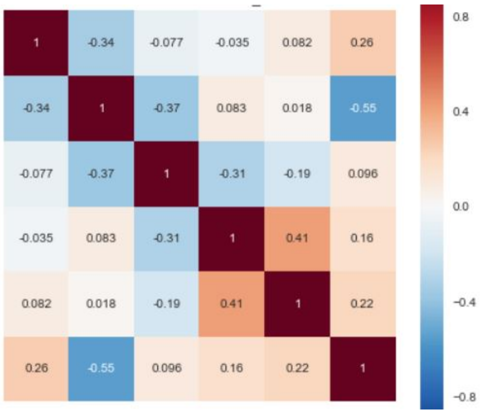




with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

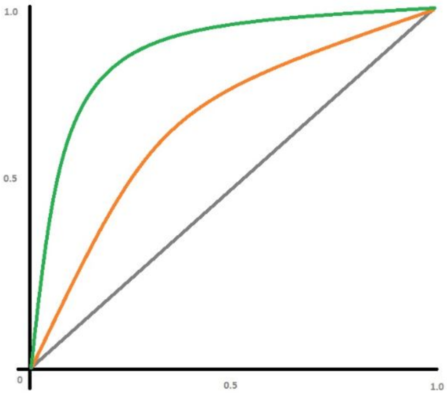
or

- 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

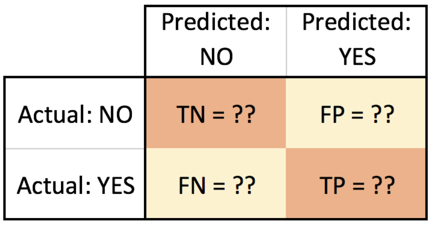


seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**

- 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.

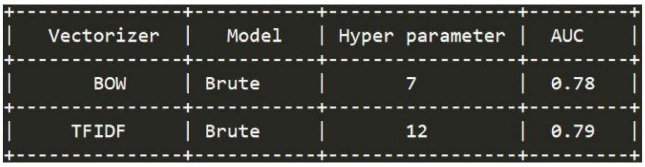


- Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

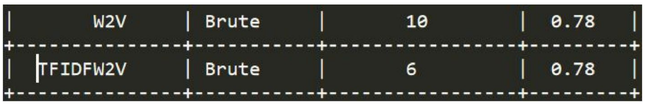


	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

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  |
+-----------------+------------+------------------+--------+
```



```
|     W2V      | Brute    |        10        |  0.78  |
+--------------+----------+------------------+--------+
|  TFIDFW2V    | Brute    |        6         |  0.78  |
+--------------+----------+------------------+--------+
```

1. GBDT (xgboost/lightgbm)

In [52]:

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

In [170]:

```
with open(r'E:\assignment\donar case\glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

1.1 Loading Data

In [171]:

```
prep_data = pd.read_csv(r'E:\assignment\donar case\preprocessed_data.csv')
prep_data.head(2)
```

Out[171]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_cate
0	ca	mrs	grades_prek_2	53	1	math_s
1	ut	ms	grades_3_5	4	1	special

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

In [172]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()
neg=[];pos=[];neu=[]; compound = []
for i in range(len(prep_data['essay'])):
    sentiment_scores = analyzer.polarity_scores(prep_data['essay'][i])
    neg.append(sentiment_scores['neg'])
    pos.append(sentiment_scores['pos'])
    neu.append(sentiment_scores['neu'])
    compound.append(sentiment_scores['compound'])
```

In [173]:

```
#new columns indicating the sentiment score of each project essay
prep_data['neg'] = neg
prep_data['neu'] = neu
prep_data['pos'] = pos
prep_data['compound'] = compound
```

In [174]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(prep_data, prep_data['project_is_approved'], te
st_size=0.33, stratify = prep_data['project_is_approved'])
```

1.3 Make Data Model Ready: encoding eassay, and project_title

In [175]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_essay = TfidfVectorizer(min_df=10)
vectorizer_tfidf_essay.fit(X_train['essay']) #Fitting has to be onTrain data
train_essay_tfidf = vectorizer_tfidf_essay.transform(X_train['essay'].values)
test_essay_tfidf =vectorizer_tfidf_essay.transform(X_test['essay'].values)
print("Shape of train data matrix after one hot encoding ",train_essay_tfidf.shape)
print("Shape of test data matrix after one hot encoding ",test_essay_tfidf.shape)
```

```
Shape of train data matrix after one hot encoding  (73196, 14184)
Shape of test data matrix after one hot encoding  (36052, 14184)
```

In [176]:

```
# average Word2Vec
# compute average word2vec for each review.
train_avg_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
train_avg_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in X_train['essay']: # 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
    train_avg_w2v_essays.append(vector)
print(len(train_avg_w2v_essays))
print(len(train_avg_w2v_essays[0]))
```

```
141
300
```

In [177]:

```
# average Word2Vec
# compute average word2vec for each review.
test_avg_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in X_test['essay']: # 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
    test_avg_w2v_essays.append(vector)
print(len(test_avg_w2v_essays))
print(len(test_avg_w2v_essays[0]))
```

```
36052
300
```

In [178]:

```
# 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 [179]:

```
#average Word2Vec
# compute average word2vec for each review.
train_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in X_train['essay']: # 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
tfvalue((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2vtf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_tfidf_w2v_essays.append(vector)
print(len(train_tfidf_w2v_essays))
print(len(train_tfidf_w2v_essays[0]))
```

```
73196
300
```

In [180]:

```
#average Word2Vec
# compute average word2vec for each review.
test_tfidf_w2v_essays = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in X_test['essay']: # 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 tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2vtf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_tfidf_w2v_essays.append(vector)
print(len(test_tfidf_w2v_essays))
print(len(test_tfidf_w2v_essays[0]))
```

```
36052
300
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [181]:

```
#Response Coding

def mask(df, key, value):
    return df[df[key] == value]
def get_response(data,data_label):
    cat_values = np.unique(data).tolist()
    d=""
    d=data.name
    cat_test_values=np.unique(X_test[d]).tolist()
    df = pd.DataFrame({'feature':data.values.tolist(),'label':data_label.values.tolist()})
    pd.DataFrame.mask = mask
    accep = {};reject={};prob_neg = {};prob_pos={}
    for i in cat_values:
        if i in cat_test_values:
            count_0 = len(df.mask('feature', i).mask('label', 0))
            count_1 = len(df.mask('feature', i).mask('label', 1))
            total = count_0 + count_1
            prob_0 = count_0/total
            prob_1 = count_1/total
            accep[i] = count_1
            reject[i] = count_0
            prob_neg[i] = prob_0
            prob_pos[i] = prob_1
        else:
            prob_neg[i] = 0.5
            prob_pos[i] = 0.5

    return prob_neg,prob_pos
```

In [182]:

```
cat_0_train = get_response(X_train['clean_categories'],y_train)[0]
cat_1_train = get_response(X_train['clean_categories'],y_train)[1]
```

In [183]:

```
subcat_0_train = get_response(X_train['clean_subcategories'],y_train)[0]
subcat_1_train = get_response(X_train['clean_subcategories'],y_train)[1]
```

In [184]:

```
state_0_train = get_response(X_train['school_state'],y_train)[0]
state_1_train = get_response(X_train['school_state'],y_train)[1]
prefix_0_train = get_response(X_train['teacher_prefix'],y_train)[0]
prefix_1_train = get_response(X_train['teacher_prefix'],y_train)[1]
grad_cat_0_train = get_response(X_train['project_grade_category'],y_train)[0]
grad_cat_1_train = get_response(X_train['project_grade_category'],y_train)[1]
cat_0_test = get_response(X_test['clean_categories'],y_test)[0]
cat_1_test = get_response(X_test['clean_categories'],y_test)[1]
subcat_0_test = get_response(X_test['clean_subcategories'],y_test)[0]
subcat_1_test = get_response(X_test['clean_subcategories'],y_test)[1]
state_0_test = get_response(X_test['school_state'],y_test)[0]
state_1_test = get_response(X_test['school_state'],y_test)[1]
prefix_0_test = get_response(X_test['teacher_prefix'],y_test)[0]
prefix_1_test = get_response(X_test['teacher_prefix'],y_test)[1]
grad_cat_0_test = get_response(X_test['project_grade_category'],y_test)[0]
grad_cat_1_test = get_response(X_test['project_grade_category'],y_test)[1]
```

In [185]:

In [185]:

```
cat_neg_train = []
cat_pos_train = []
for i in X_train['clean_categories']:
    cat_neg_train.append(cat_0_train[i])
    cat_pos_train.append(cat_1_train[i])
X_train['cat_0'] = cat_neg_train
X_train['cat_1'] = cat_pos_train
print(X_train['cat_0'].shape)
print(X_train['cat_1'].shape)
```

```
(73196,)
(73196,)
```

In [186]:

```
cat_neg_test = []
cat_pos_test = []
for i in X_test['clean_categories']:
    cat_neg_test.append(cat_0_test[i])
    cat_pos_test.append(cat_1_test[i])
X_test['cat_0'] = cat_neg_test
X_test['cat_1'] = cat_pos_test
print(X_test['cat_0'].shape)
print(X_test['cat_1'].shape)
```

```
(36052,)
(36052,)
```

In [187]:

```
subcat_neg_train = []
subcat_pos_train = []
for i in X_train['clean_subcategories']:
    subcat_neg_train.append(subcat_0_train[i])
    subcat_pos_train.append(subcat_1_train[i])
X_train['subcat_0'] = subcat_neg_train
X_train['subcat_1'] = subcat_pos_train
print(X_train['subcat_0'].shape)
print(X_train['subcat_1'].shape)
```

```
(73196,)
(73196,)
```

In [188]:

```
subcat_neg_test = []
subcat_pos_test = []
for i in X_test['clean_subcategories']:
    subcat_neg_test.append(subcat_0_test[i])
    subcat_pos_test.append(subcat_1_test[i])
X_test['subcat_0'] = subcat_neg_test
X_test['subcat_1'] = subcat_pos_test
print(X_test['subcat_0'].shape)
print(X_test['subcat_1'].shape)
```

```
(36052,)
(36052,)
```

In [189]:

```
state_neg_train = []
state_pos_train = []
for i in X_train['school_state']:
    state_neg_train.append(state_0_train[i])
    state_pos_train.append(state_1_train[i])
X_train['state_0'] = state_neg_train
X_train['state_1'] = state_pos_train
print(X_train['state_0'].shape)
print(X_train['state_1'].shape)
```

```
(73196,)
(73196,)
```

In [190]:

```
state_neg_test = []
state_pos_test = []
for i in X_test['school_state']:
    state_neg_test.append(state_0_test[i])
    state_pos_test.append(state_1_test[i])
X_test['state_0'] = state_neg_test
X_test['state_1'] = state_pos_test
print(X_test['state_0'].shape)
print(X_test['state_1'].shape)
```

```
(36052,)
(36052,)
```

In [191]:

```
prefix_neg_train = []
prefix_pos_train = []
for i in X_train['teacher_prefix']:
    prefix_neg_train.append(prefix_0_train[i])
    prefix_pos_train.append(prefix_1_train[i])
X_train['prefix_0'] = prefix_neg_train
X_train['prefix_1'] = prefix_pos_train
print(X_train['prefix_0'].shape)
print(X_train['prefix_1'].shape)
```

```
(73196,)
(73196,)
```

In [192]:

```
prefix_neg_test = []
prefix_pos_test = []
for i in X_test['teacher_prefix']:
    prefix_neg_test.append(prefix_0_test[i])
    prefix_pos_test.append(prefix_1_test[i])
X_test['prefix_0'] = prefix_neg_test
X_test['prefix_1'] = prefix_pos_test
print(X_test['prefix_0'].shape)
print(X_test['prefix_1'].shape)
```

```
(36052,)
(36052,)
```

In [193]:

```
grade_neg_train = []
grade_pos_train = []
for i in X_train['project_grade_category']:
    grade_neg_train.append(grad_cat_0_train[i])
    grade_pos_train.append(grad_cat_1_train[i])
X_train['grade_0'] = grade_neg_train
X_train['grade_1'] = grade_pos_train
print(X_train['grade_0'].shape)
print(X_train['grade_1'].shape)
```

```
(73196,)
(73196,)
```

In [194]:

```
grade_neg_test = []
grade_pos_test = []
for i in X_test['project_grade_category']:
    grade_neg_test.append(grad_cat_0_test[i])
    grade_pos_test.append(grad_cat_1_test[i])
X_test['grade_0'] = grade_neg_test
X_test['grade_1'] = grade_pos_test
print(X_test['grade_0'].shape)
print(X_test['grade_1'].shape)
```

```
(36052,)
(36052,)
```

In [79]:

```
#Concatenating all TFIDF Features
```

In [80]:

```
from scipy.sparse import hstack
X_train = hstack((X_train["cat_0"].values.reshape(-1,1), X_train["cat_1"].values.reshape(-1,1), X_t
rain["subcat_0"].values.reshape(-1,1),
                  X_train["subcat_1"].values.reshape(-1,1), X_train["state_0"].values.reshape(-1,1)
X_train["state_1"].values.reshape(-1,1),
                  X_train["grade_0"].values.reshape(-1,1), X_train["grade_1"].values.reshape(-1,1),
X_train["prefix_0"].values.reshape(-1,1),
                  X_train["prefix_1"].values.reshape(-1,1),X_train['price'].values.reshape(-1,1),X_
rain['teacher_number_of_previously_posted_projects'].values.reshape(-1,1),
                  X_train['pos'].values.reshape(-1,1), X_train['neu'].values.reshape(-1,1), X_train
'neg'].values.reshape(-1,1), X_train['compound'].values.reshape(-1,1),train_essay_tfidf)).tocsr()

X_test = hstack((X_test["cat_0"].values.reshape(-1,1), X_test["cat_1"].values.reshape(-1,1), X_test
["subcat_0"].values.reshape(-1,1),
                  X_test["subcat_1"].values.reshape(-1,1), X_test["state_0"].values.reshape(-1,1), X
_test["state_1"].values.reshape(-1,1),
                  X_test["grade_0"].values.reshape(-1,1), X_test["grade_1"].values.reshape(-1,1), X_
test["prefix_0"].values.reshape(-1,1),
                  X_test["prefix_1"].values.reshape(-1,1),X_test['price'].values.reshape(-1,1),X_te
t['teacher_number_of_previously_posted_projects'].values.reshape(-1,1),
                  X_test['pos'].values.reshape(-1,1), X_test['neu'].values.reshape(-1,1), X_test['ne
g'].values.reshape(-1,1), X_test['compound'].values.reshape(-1,1),test_essay_tfidf)).tocsr()
print("Final TFIDF Data matrix")
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)
```

```
Final TFIDF Data matrix
(73196, 14232) (73196,)
(36052, 14232) (36052,)
=============================================================================================
```

In [81]:

```
#Applying XGBOOST ON TFIDF
```

In [82]:

```
from scipy.stats import randint as sp_randint
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
gbdt = XGBClassifier()
grid_params = {'n_estimators': [10, 50, 100], 'max_depth':[2,5,10]}
gs = GridSearchCV(gbdt,grid_params ,cv=3, scoring='roc_auc',return_train_score=True,n_jobs=-1)
gs.fit(X_train, y_train)
```

Out[82]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constrai...
                                     num_parallel_tree=None,
                                     objective='binary:logistic',
                                     random_state=None, reg_alpha=None,
                                     reg_lambda=None, scale_pos_weight=None,
                                     subsample=None, tree_method=None,
                                     validate_parameters=False,
                                     verbosity=None),
             iid='warn', n_jobs=-1,
             param_grid={'max_depth': [2, 5, 10],
                         'n_estimators': [10, 50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

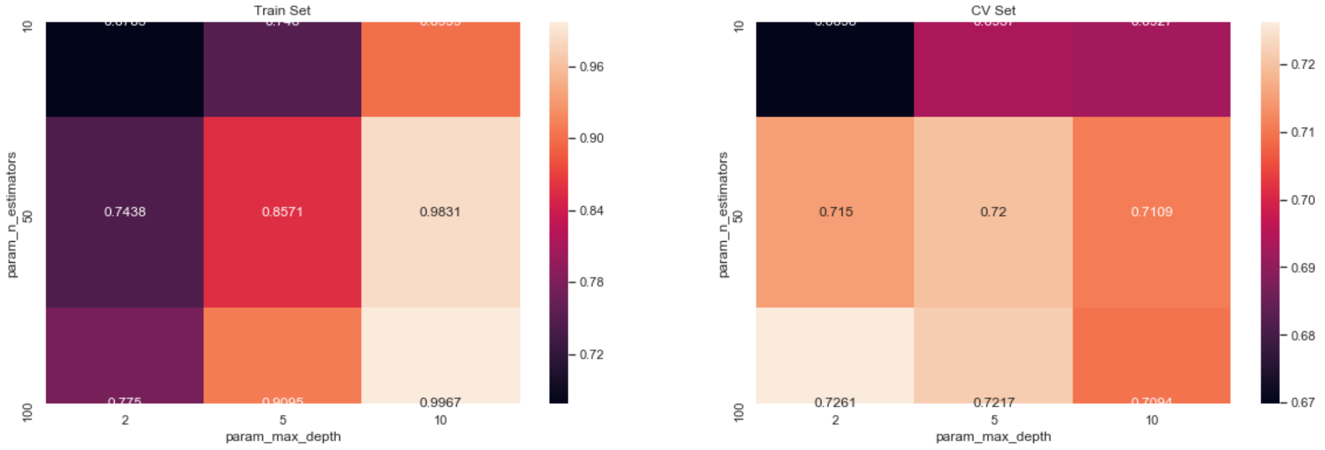
In [83]:

```
print('Best score: ',gs.best_score_)
print('k value with best score: ',gs.best_params_)
print('='*75)
print('Train AUC scores')
print(gs.cv_results_['mean_train_score'])
print('CV AUC scores')
print(gs.cv_results_['mean_test_score'])
```

```
Best score:  0.7261385332980901
k value with best score:  {'max_depth': 2, 'n_estimators': 100}
===========================================================================
Train AUC scores
[0.6783479  0.74382595 0.77496039 0.74799896 0.85705737 0.90950417
 0.89988699 0.98307372 0.99671967]
CV AUC scores
[0.66984183 0.71499992 0.72613853 0.69371443 0.71995857 0.72169263
 0.69268183 0.71088139 0.70943098]
```

In [84]:

```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(gs.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max(
).unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [87]:

```
gs.best_params_
```

Out[87]:

```
{'max_depth': 2, 'n_estimators': 100}
```

In [86]:

```
max_d = gs.best_params_['max_depth']
n_est = gs.best_params_['n_estimators']
```

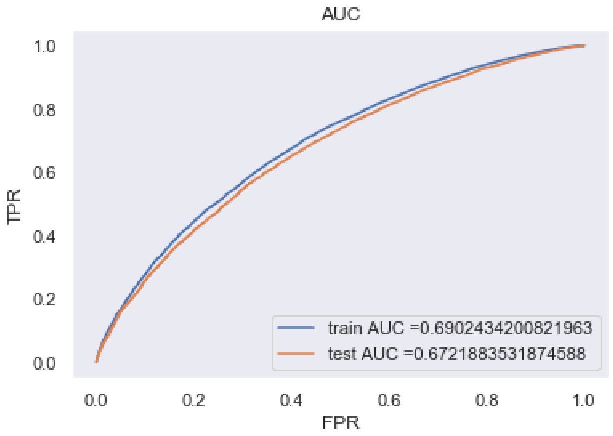
In [88]:

```
def pred_prob(clf, data):
    y_pred = []
    y_pred = clf.predict_proba(data)[:,1]
    return y_pred

def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    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, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [89]:

```
#
https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.ro
ve
from sklearn.metrics import roc_curve, auc
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(max_depth = max_d, n_estimators = n_est)
model.fit(X_train,y_train)
y_train_pred = pred_prob(model,X_train)
y_test_pred = pred_prob(model,X_test)
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.close
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")
plt.grid()
plt.show()
```



In [168]:

```
#Train Confusion Matrix of TFIDF
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
```

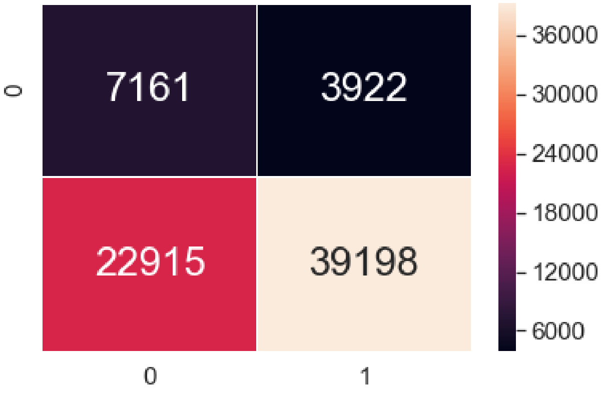
```
the maximum value of tpr*(1-fpr) 0.40775354295248856 for threshold 0.848
```

In [161]:

```
print("Train confusion matrix")
co=confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(co, annot=True, annot_kws={"size": 25},fmt="d",linewidths=.5,yticklabels=2)
```

Out[161]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7212ec6a48>
```



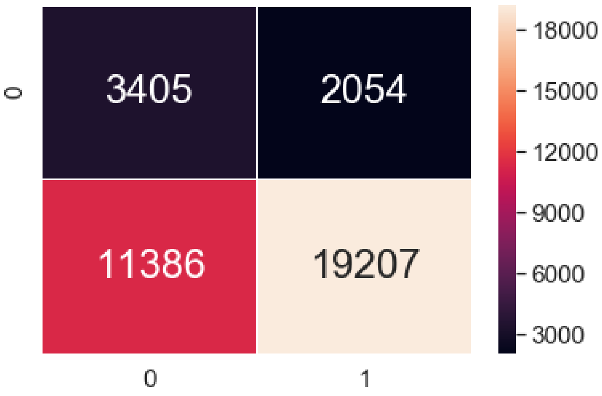
In [162]:

```
#Test Confusion Matrix of TFIDF
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), annot=True, annot_k
ws={"size": 25},fmt="d",linewidths=.5,yticklabels=2)
```

```
Test confusion matrix
```

Out[162]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7212f918c8>
```



In [163]:

```
train_tfidf_w2v_essays_np = np.array(train_tfidf_w2v_essays)
test_tfidf_w2v_essays_np = np.array(test_tfidf_w2v_essays)
```

In [196]:

```
from scipy.sparse import coo_matrix, hstack
tr1 = coo_matrix(X_train["cat_0"].values.reshape(-1,1))
tr2 = coo_matrix(X_train["cat_1"].values.reshape(-1,1))
tr3 = coo_matrix(X_train["subcat_0"].values.reshape(-1,1))
tr4 = coo_matrix(X_train["subcat_1"].values.reshape(-1,1))
tr5 = coo_matrix(X_train["state_0"].values.reshape(-1,1))
```

```
tr6 = coo_matrix(X_train["state_1"].values.reshape(-1,1))
tr7 = coo_matrix(X_train["grade_0"].values.reshape(-1,1))
tr8 = coo_matrix(X_train["grade_1"].values.reshape(-1,1))
tr9 = coo_matrix(X_train["prefix_0"].values.reshape(-1,1))
tr10 = coo_matrix(X_train["prefix_1"].values.reshape(-1,1))
tr11 = coo_matrix(X_train["price"].values.reshape(-1,1))
tr11 = coo_matrix(X_train["teacher_number_of_previously_posted_projects"].values.reshape(-1,1))
tr12 = coo_matrix(X_train["pos"].values.reshape(-1,1))
tr13 = coo_matrix(X_train["neg"].values.reshape(-1,1))
tr14 = coo_matrix(X_train["neu"].values.reshape(-1,1))
tr15 = coo_matrix(X_train["compound"].values.reshape(-1,1))
tr16 = coo_matrix(train_tfidf_w2v_essays_np)
```

In [197]:

```
X_train = hstack([tr1,tr2,tr3,tr4,tr5,tr6,tr7,tr8,tr9,tr10,tr11,tr12,tr13,tr14,tr15,tr16]).tocsr()
```

In [198]:

```
from scipy.sparse import coo_matrix, hstack
te1 = coo_matrix(X_test["cat_0"].values.reshape(-1,1))
te2 = coo_matrix(X_test["cat_1"].values.reshape(-1,1))
te3 = coo_matrix(X_test["subcat_0"].values.reshape(-1,1))
te4 = coo_matrix(X_test["subcat_1"].values.reshape(-1,1))
te5 = coo_matrix(X_test["state_0"].values.reshape(-1,1))
te6 = coo_matrix(X_test["state_1"].values.reshape(-1,1))
te7 = coo_matrix(X_test["grade_0"].values.reshape(-1,1))
te8 = coo_matrix(X_test["grade_1"].values.reshape(-1,1))
te9 = coo_matrix(X_test["prefix_0"].values.reshape(-1,1))
te10 = coo_matrix(X_test["prefix_1"].values.reshape(-1,1))
te11 = coo_matrix(X_test["price"].values.reshape(-1,1))
te11 = coo_matrix(X_test["teacher_number_of_previously_posted_projects"].values.reshape(-1,1))
te12 = coo_matrix(X_test["pos"].values.reshape(-1,1))
te13 = coo_matrix(X_test["neg"].values.reshape(-1,1))
te14 = coo_matrix(X_test["neu"].values.reshape(-1,1))
te15 = coo_matrix(X_test["compound"].values.reshape(-1,1))
te16 = coo_matrix(test_tfidf_w2v_essays_np)
```

In [199]:

```
X_test = hstack([te1,te2,te3,te4,te5,te6,te7,te8,te9,te10,te11,te12,te13,te14,te15,te16]).tocsr()
```

In [200]:

```
#Applying XGBOOST On TFIDF -W2V
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier
gbdt  = XGBClassifier()
grid_params = {'n_estimators': [10, 50, 100], 'max_depth':[2,5,10]}
rs = RandomizedSearchCV(gbdt,grid_params ,cv=3, scoring='roc_auc',return_train_score=True,n_jobs=-1
)
rs.fit(X_train, y_train)
```

Out[200]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None, gamma=None,
                                           gpu_id=None, importance_type='gain',
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=None,
                                           min_child_weight=None, missing=nan,
                                           monotone_co...
                                           random_state=None, reg_alpha=None,
                                           reg_lambda=None,
                                           scale_pos_weight=None,
                                           subsample=None, tree_method=None,
                                           validate_parameters=False,
                                           verbosity=None),
                   iid='warn', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': [2, 5, 10],
                                        'n_estimators': [10, 50, 100]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=True, scoring='roc_auc', verbose=0)
```

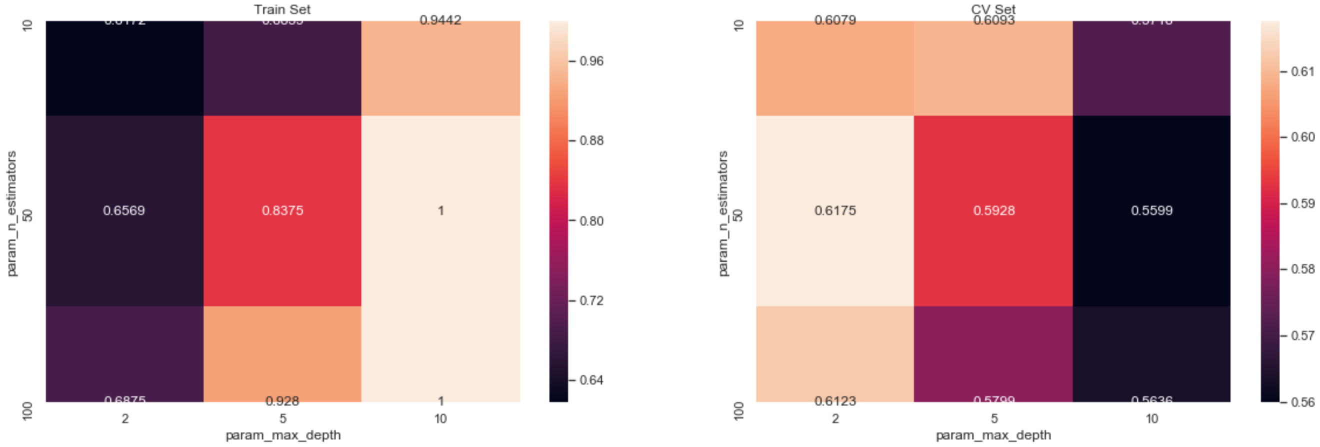
In [201]:

```
print('Best score: ',rs.best_score_)
print('k value with best score: ',rs.best_params_)
print('='*75)
print('Train AUC scores')
print(rs.cv_results_['mean_train_score'])
print('CV AUC scores')
print(rs.cv_results_['mean_test_score'])
```

```
Best score:  0.6174958760137832
k value with best score:  {'n_estimators': 50, 'max_depth': 2}
===========================================================================
Train AUC scores
[0.6171561  0.65694083 0.68747866 0.68388587 0.83754251 0.92801416
 0.94418432 0.99999969 1.        ]
CV AUC scores
[0.607904   0.61749588 0.61233596 0.60930948 0.59282055 0.57987553
 0.57181274 0.55986971 0.56357041]
```

In [202]:

```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(rs.cv_results_).groupby(['param_n_estimators', 'param_max_depth']).max(
).unstack()[['mean_test_score', 'mean_train_score']]
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



In [203]:

```
#Best Parameters for TFIDF-W2V
rs.best_params_
```

Out[203]:

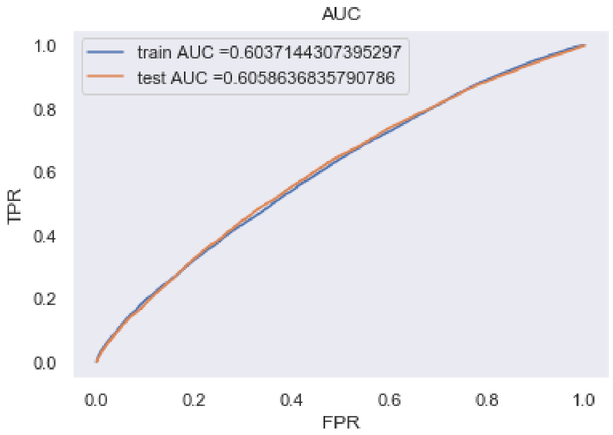
```
{'n_estimators': 50, 'max_depth': 2}
```

In [204]:

```
max_d = rs.best_params_['max_depth']
n_est = rs.best_params_['n_estimators']
```

In [205]:

```
#
https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc
ve
from sklearn.metrics import roc_curve, auc
model = RandomForestClassifier(max_depth = max_d, n_estimators = n_est)
model.fit(X_train,y_train)
y_train_pred = pred_prob(model,X_train)
y_test_pred = pred_prob(model,X_test)
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.close
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")
plt.grid()
plt.show()
```



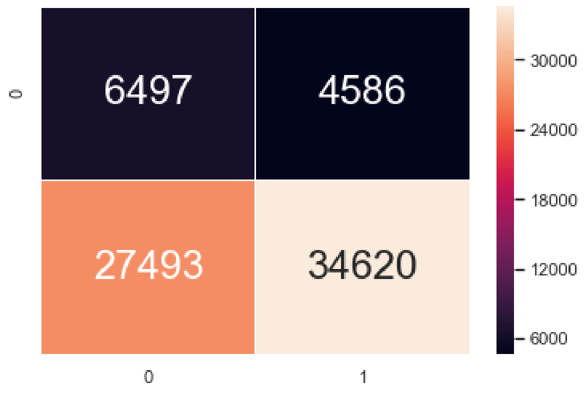
In [206]:

```
#Train TFIDF-W2V Confusion Matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
co=confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(co, annot=True, annot_kws={"size": 25},fmt="d",linewidths=.5,yticklabels=2)
```

```
the maximum value of tpr*(1-fpr) 0.32673833475136643 for threshold 0.848
Train confusion matrix
```

Out[206]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7230868ac8>
```



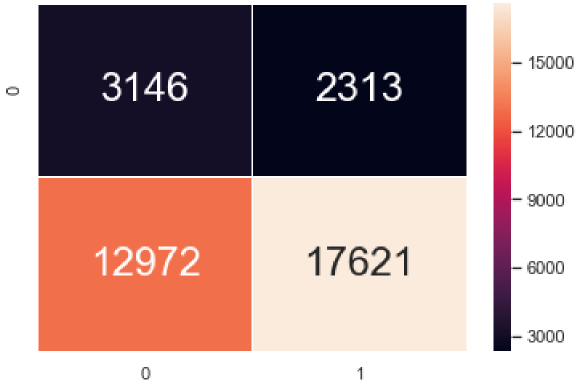
In [207]:

```
#Test TFIDF-W2V Confusion Matrix
print("Test confusion matrix")
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)), annot=True, annot_k
ws={"size": 25},fmt="d",linewidths=.5,yticklabels=2)
```

```
Test confusion matrix
```

Out[207]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x723082e548>
```



3. Summary

In [1]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameters(n_estimators,max_depth)", "Test AUC"]
x.add_row(["TFIDF", "GBDT", "(100, 10)", 0.672])
x.add_row(["TFIDF W2V", "GBDT", "(100, 5)", 0.605])
print(x)
```

```
+------------+-------+-----------------------------------------+----------+
| Vectorizer | Model | Hyperparameters(n_estimators,max_depth) | Test AUC |
+------------+-------+-----------------------------------------+----------+
|   TFIDF    |  GBDT |                (100, 10)                |  0.672   |
| TFIDF W2V  |  GBDT |                 (100, 5)                |  0.605   |
+------------+-------+-----------------------------------------+----------+
```

In [ ]: