Donors Choose Decision Tree

April 28, 2021

[ ]:

```
```

1 Assignment : DT

Please check below video before attempting this assignment

TF-IDFW2V

Tfidf w2v (w1,w2..) = (tfidf(w1) * w2v(w1) + tfidf(w2) * w2v(w2) + …) / (tfidf(w1) + tfidf(w2) + …)

(Optional) Please check course video on AVgw2V and TF-IDFW2Vfor more details.

Glove vectors

In this assignment you will be working with glove vectors , please check this and this for more details.

Download glove vectors from this link

or else , you can use below code

2 Task - 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)

```
</li>
<li><strong>The hyper paramter tuning (best `depth` in range [1, 5, 10, 50], and the best `min_
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>find the best hyper paramter using k-fold cross validation(use gridsearch cv or randomsearc
    </ul>
</li>
<li>
<strong>Representation of results</strong>
    <ul>
```

```
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='https://i.imgur.com/Gp2DQmh.jpg' width=500px> with X-axis as <strong>min_sample_spli
        <p style="text-align:center;font-size:30px;color:red;"><strong>or</strong></p> <br>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='https://i.imgur.com/fgN9aUP.jpg' width=300px> <a href='https://seaborn.pydata.org/gen
<li>You choose either of the plotting techniques out of 3d plot or heat map</li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='https://i.imgur.com/wMQDTFe.jpg' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='https://i.imgur.com/IdN5Ctv.png' width=300px></li>
<li>Once after you plot the confusion matrix with the test data, get all the `false positive da
    <ul>
        <li> Plot the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) wi
        <li> Plot the box plot with the `price` of these `false positive data points`</li>
        <li> Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `fal
    </ul>
    </ul>
</li>
```

2.1 Importing necessary packages

[1]:
```python
import numpy as np
import pandas as pd
import re
from nltk.corpus import stopwords
import pickle
from tqdm import tqdm
import nltk
from tqdm import tqdm

from sklearn.model_selection import train_test_split
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 collections import Counter
from sklearn.preprocessing import Normalizer
from scipy.sparse import hstack
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from wordcloud import WordCloud, STOPWORDS
import math
```

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
```

2.2 Importing Data

```
[2]: data = pd.read_csv('preprocessed_data.csv')
```

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

```
[3]:   school_state teacher_prefix project_grade_category  \
     0           ca            mrs          grades_prek_2

        teacher_number_of_previously_posted_projects clean_categories  \
     0                                           53     math_science

                        clean_subcategories  \
     0  appliedsciences health_lifescience

                                                    essay   price
     0  i fortunate enough use fairy tale stem kits cl…  725.05
```

2.3 Splitting Data

```
[4]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
      ↪stratify=y)
     X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
      ↪33, stratify=y_train)
```

2.4 Applying BoW on Essay feature

```
[5]: print(X_train.shape, y_train.shape)
     print(X_cv.shape, y_cv.shape)
     print(X_test.shape, y_test.shape)

     print("="*100)

     vectorizer_bow = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
```

```
vectorizer_bow.fit(X_train['essay'].values)

X_train_essay_bow = vectorizer_bow.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer_bow.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer_bow.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
```

```
(49041, 8) (49041,)
(24155, 8) (24155,)
(36052, 8) (36052,)
================================================================================
====================
After vectorizations
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
================================================================================
====================
```

2.5 Applying TFIDF on Essay Feature

[6]:

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

print("="*100)

vectorizer_tfidf = TfidfVectorizer(min_df=10,ngram_range=(1,4),␣
 ↪max_features=5000)
vectorizer_tfidf.fit(X_train['essay'].values)

X_train_essay_tfidf = vectorizer_tfidf.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer_tfidf.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer_tfidf.transform(X_test['essay'].values)

print("After vectorizations")
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)
```

```
(49041, 8) (49041,)
```

```
(24155, 8) (24155,)
(36052, 8) (36052,)
================================================================================
====================
After vectorizations
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
================================================================================
====================
```

2.6 TFIDF W2V on Essay Feature

[7]:

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

[8]:

```
def get_tfidfw2v(tfidf_model, preprocessed_essays):
    dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.
↪idf_)))
    tfidf_words = set(tfidf_model.get_feature_names())
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in␣
↪this list
    for sentence in tqdm(preprocessed_essays): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence/
↪review
        for word in sentence.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words):
                vec = model[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and the␣
↪tf value((sentence.count(word)/len(sentence.split())))
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.
↪split())) # getting the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)
    print(len(tfidf_w2v_vectors))
    print(len(tfidf_w2v_vectors[0]))
    return np.array(tfidf_w2v_vectors)
```

[9]:

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

```python
print("="*100)

vectorizer_tfidf = TfidfVectorizer(min_df=10,ngram_range=(1,4),␣
 ↪max_features=5000)
vectorizer_tfidf.fit(X_train['essay'].values)

X_train_essay_tfidf_w2v = get_tfidfw2v(vectorizer_tfidf, X_train['essay'].
 ↪values)
X_cv_essay_tfidf_w2v = get_tfidfw2v(vectorizer_tfidf, X_cv['essay'].values)
X_test_essay_tfidf_w2v = get_tfidfw2v(vectorizer_tfidf, X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf_w2v.shape, y_train.shape)
print(X_cv_essay_tfidf_w2v.shape, y_cv.shape)
print(X_test_essay_tfidf_w2v.shape, y_test.shape)
print("="*100)
```

```
(49041, 8) (49041,)
(24155, 8) (24155,)
(36052, 8) (36052,)
================================================================================
====================

100%|     | 49041/49041 [01:05<00:00, 745.74it/s]
  1%|          | 146/24155 [00:00<00:32, 729.84it/s]

49041
300

100%|     | 24155/24155 [00:32<00:00, 732.94it/s]
  0%|          | 163/36052 [00:00<00:44, 801.31it/s]

24155
300

100%|     | 36052/36052 [00:48<00:00, 747.42it/s]

36052
300
After vectorizations
(49041, 300) (49041,)
(24155, 300) (24155,)
(36052, 300) (36052,)
================================================================================
====================
```

2.7 One Hot Encoding the State Feature

[10]:

```
vectorizer_state = CountVectorizer()
vectorizer_state.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_state.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer_state.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer_state.transform(X_test['school_state'].values)

print("After vectorizations")
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_state.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 51) (49041,)
(24155, 51) (24155,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia',
'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms',
'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa',
'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
================================================================================
====================
```

2.8 One Hot Encoding the Project Category Feature

[11]:

```
vectorizer_grade = CountVectorizer()
vectorizer_grade.fit(X_train['project_grade_category'].values) # fit has to␣
 ↪happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_category_ohe = vectorizer_grade.
 ↪transform(X_train['project_grade_category'].values)
X_cv_project_category_ohe = vectorizer_grade.
 ↪transform(X_cv['project_grade_category'].values)
X_test_project_category_ohe = vectorizer_grade.
 ↪transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_project_category_ohe.shape, y_train.shape)
print(X_cv_project_category_ohe.shape, y_cv.shape)
print(X_test_project_category_ohe.shape, y_test.shape)
print(vectorizer_grade.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 4) (49041,)
(24155, 4) (24155,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
================================================================================
====================
```

2.9 One Hot Encoding the Teacher Prefix Feature

[12]:
```
vectorizer_teacher_prefix = CountVectorizer()
vectorizer_teacher_prefix.fit(X_train['teacher_prefix'].values) # fit has to␣
 ↪happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix_ohe = vectorizer_teacher_prefix.
 ↪transform(X_train['teacher_prefix'].values)
X_cv_teacher_prefix_ohe = vectorizer_teacher_prefix.
 ↪transform(X_cv['teacher_prefix'].values)
X_test_teacher_prefix_ohe = vectorizer_teacher_prefix.
 ↪transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_prefix_ohe.shape, y_train.shape)
print(X_cv_teacher_prefix_ohe.shape, y_cv.shape)
print(X_test_teacher_prefix_ohe.shape, y_test.shape)
print(vectorizer_teacher_prefix.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 5) (49041,)
(24155, 5) (24155,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
================================================================================
====================
```

2.10 one Hot Encoding the Category Feature

[13]:
```
vectorizer_cat = CountVectorizer()
vectorizer_cat.fit(X_train['clean_categories'].values) # fit has to happen only␣
 ↪on train data

# we use the fitted CountVectorizer to convert the text to vector
```

```
X_train_clean_categories_ohe = vectorizer_cat.
 ↪transform(X_train['clean_categories'].values)
X_cv_clean_categories_ohe = vectorizer_cat.transform(X_cv['clean_categories'].
 ↪values)
X_test_clean_categories_ohe = vectorizer_cat.
 ↪transform(X_test['clean_categories'].values)

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_cat.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 9) (49041,)
(24155, 9) (24155,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics',
'literacy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
================================================================================
====================
```

2.11 One Hot Encoding the Sub Category Feature

[14]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategories_ohe = vectorizer_subcat.
 ↪transform(X_train['clean_subcategories'].values)
X_cv_clean_subcategories_ohe = vectorizer_subcat.
 ↪transform(X_cv['clean_subcategories'].values)
X_test_clean_subcategories_ohe = vectorizer_subcat.
 ↪transform(X_test['clean_subcategories'].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_subcat.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 30) (49041,)
(24155, 30) (24155,)
```

```
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics',
'environmentalscience', 'esl', 'extracurricular', 'financialliteracy',
'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness',
'history_geography', 'literacy', 'literature_writing', 'mathematics', 'music',
'nutritioneducation', 'other', 'parentinvolvement', 'performingarts',
'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
================================================================================
====================
```

2.12 Normalizing the Price Feature

[15]:

```
normalizer = Normalizer()
normalizer.fit(X_train['price'].values.reshape(-1,1))
# normalizer.fit(X_train['price'])

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
================================================================================
====================
```

2.13 Normalizing the Previous Projects Feature

[16]:

```
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(-1,1))

X_train_prev_projects_norm = normalizer.
 ↪transform(X_train['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(-1,1))
X_cv_prev_projects_norm = normalizer.
 ↪transform(X_cv['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(-1,1))
```

```
X_test_prev_projects_norm = normalizer.
 ↪transform(X_test['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(-1,1))


print("After vectorizations")
print(X_train_prev_projects_norm.shape, y_train.shape)
print(X_cv_prev_projects_norm.shape, y_cv.shape)
print(X_test_prev_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
================================================================================
====================
```

2.14 Calculating the Sentiment Scores of Pre Processed Essays

[17]:
```
def get_sentiment_scores(essays):
    sid = SentimentIntensityAnalyzer()

    scores = np.zeros(shape=(len(essays),4))

    for i in tqdm(range(len(essays))):
        essay = essays.iloc[i]
        ss = sid.polarity_scores(essay)
        sentscores = [ss['neg'], ss['neu'], ss['pos'], ss['compound']]
        scores[i] = sentscores

    print(scores.shape)

    return scores
```

[18]:
```
from nltk.sentiment.vader import SentimentIntensityAnalyzer

X_train_sent_scores = get_sentiment_scores(X_train['essay'])
X_cv_sent_scores = get_sentiment_scores(X_cv['essay'])
X_test_sent_scores = get_sentiment_scores(X_test['essay'])

print("Sentiment Scores Shapes")
print(X_train_sent_scores.shape)
print(X_cv_sent_scores.shape)
print(X_test_sent_scores.shape)
print("="*100)
```

```
100%|      | 49041/49041 [01:16<00:00, 639.48it/s]
  0%|          | 67/24155 [00:00<00:36, 667.43it/s]

(49041, 4)

100%|      | 24155/24155 [00:36<00:00, 660.17it/s]
  0%|          | 67/36052 [00:00<00:54, 662.99it/s]

(24155, 4)

100%|      | 36052/36052 [00:59<00:00, 610.62it/s]

(36052, 4)
Sentiment Scores Shapes
(49041, 4)
(24155, 4)
(36052, 4)
================================================================================
====================
```

2.15 Stacking all vectorized features into one dataset

[19]:

```
X_tr_bow = hstack((X_train_state_ohe, X_train_project_category_ohe,␣
 ↪X_train_teacher_prefix_ohe, X_train_clean_categories_ohe,␣
 ↪X_train_clean_subcategories_ohe, X_train_price_norm,␣
 ↪X_train_prev_projects_norm, X_train_essay_bow, X_train_sent_scores)).tocsr()
X_cr_bow = hstack((X_cv_state_ohe, X_cv_project_category_ohe,␣
 ↪X_cv_teacher_prefix_ohe, X_cv_clean_categories_ohe,␣
 ↪X_cv_clean_subcategories_ohe, X_cv_price_norm, X_cv_prev_projects_norm,␣
 ↪X_cv_essay_bow, X_cv_sent_scores)).tocsr()
X_te_bow = hstack((X_test_state_ohe, X_test_project_category_ohe,␣
 ↪X_test_teacher_prefix_ohe, X_test_clean_categories_ohe,␣
 ↪X_test_clean_subcategories_ohe, X_test_price_norm,␣
 ↪X_test_prev_projects_norm, X_test_essay_bow, X_test_sent_scores)).tocsr()

X_tr_tfidf = hstack((X_train_state_ohe, X_train_project_category_ohe,␣
 ↪X_train_teacher_prefix_ohe, X_train_clean_categories_ohe,␣
 ↪X_train_clean_subcategories_ohe, X_train_price_norm,␣
 ↪X_train_prev_projects_norm, X_train_essay_tfidf, X_train_sent_scores)).
 ↪tocsr()
X_cr_tfidf = hstack((X_cv_state_ohe, X_cv_project_category_ohe,␣
 ↪X_cv_teacher_prefix_ohe, X_cv_clean_categories_ohe,␣
 ↪X_cv_clean_subcategories_ohe, X_cv_price_norm, X_cv_prev_projects_norm,␣
 ↪X_cv_essay_tfidf, X_cv_sent_scores)).tocsr()
X_te_tfidf = hstack((X_test_state_ohe, X_test_project_category_ohe,␣
 ↪X_test_teacher_prefix_ohe, X_test_clean_categories_ohe,␣
 ↪X_test_clean_subcategories_ohe, X_test_price_norm,␣
 ↪X_test_prev_projects_norm, X_test_essay_tfidf, X_test_sent_scores)).tocsr()
```

```
X_tr_tfidf_w2v = hstack((X_train_state_ohe, X_train_project_category_ohe,␣
 ↪X_train_teacher_prefix_ohe, X_train_clean_categories_ohe,␣
 ↪X_train_clean_subcategories_ohe, X_train_price_norm,␣
 ↪X_train_prev_projects_norm, X_train_essay_tfidf_w2v, X_train_sent_scores)).
 ↪tocsr()
X_cr_tfidf_w2v = hstack((X_cv_state_ohe, X_cv_project_category_ohe,␣
 ↪X_cv_teacher_prefix_ohe, X_cv_clean_categories_ohe,␣
 ↪X_cv_clean_subcategories_ohe, X_cv_price_norm, X_cv_prev_projects_norm,␣
 ↪X_cv_essay_tfidf_w2v, X_cv_sent_scores)).tocsr()
X_te_tfidf_w2v = hstack((X_test_state_ohe, X_test_project_category_ohe,␣
 ↪X_test_teacher_prefix_ohe, X_test_clean_categories_ohe,␣
 ↪X_test_clean_subcategories_ohe, X_test_price_norm,␣
 ↪X_test_prev_projects_norm, X_test_essay_tfidf_w2v, X_test_sent_scores)).
 ↪tocsr()

print("Final Data matrix: BoW")
print(X_tr_bow.shape, y_train.shape)
print(X_cr_bow.shape, y_cv.shape)
print(X_te_bow.shape, y_test.shape)

print("="*100)

print("Final Data matrix: TFIDF")
print(X_tr_tfidf.shape, y_train.shape)
print(X_cr_tfidf.shape, y_cv.shape)
print(X_te_tfidf.shape, y_test.shape)

print("="*100)

print("Final Data matrix: TFIDF W2V")
print(X_tr_tfidf_w2v.shape, y_train.shape)
print(X_cr_tfidf_w2v.shape, y_cv.shape)
print(X_te_tfidf_w2v.shape, y_test.shape)
```

```
Final Data matrix: BoW
(49041, 5105) (49041,)
(24155, 5105) (24155,)
(36052, 5105) (36052,)
================================================================================
====================
Final Data matrix: TFIDF
(49041, 5105) (49041,)
(24155, 5105) (24155,)
(36052, 5105) (36052,)
================================================================================
====================
Final Data matrix: TFIDF W2V
(49041, 405) (49041,)
(24155, 405) (24155,)
(36052, 405) (36052,)
```

3 Bag of Words

3.1 Cross Validation to get the best hyperparameters

[20]:
```
depths = [1,5,10,50]
splits = [5,10,100,500]
```

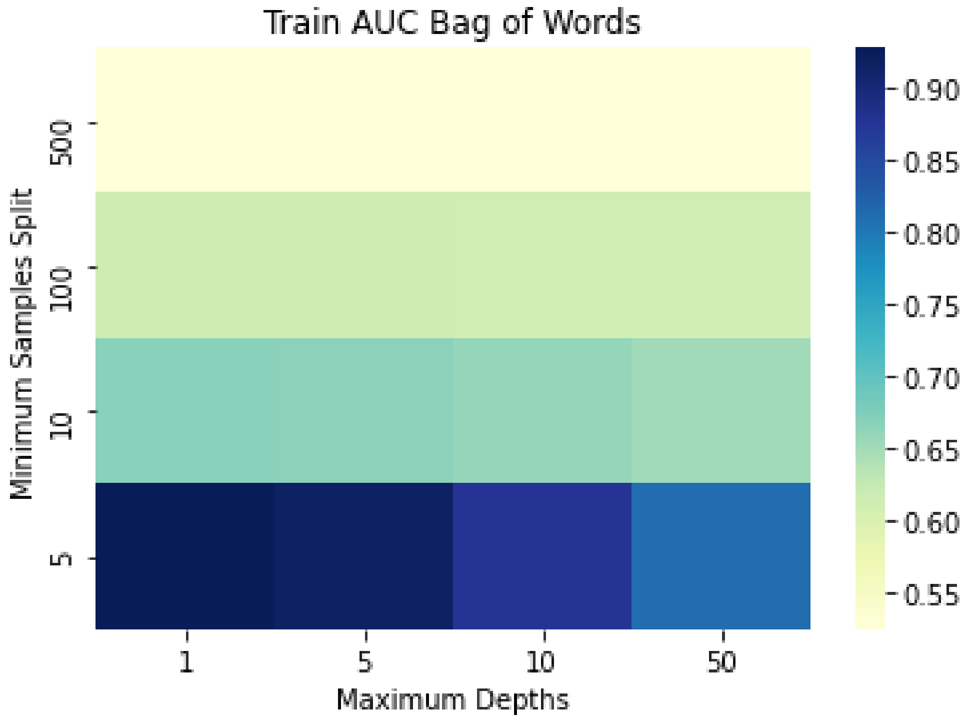
[ ]:
```
dt_bow = DecisionTreeClassifier()

parameters = {'max_depth':[1,5,10,50], 'min_samples_split':[5,10,100,500]}

clf_bow = GridSearchCV(dt_bow, parameters, cv= 5,␣
 ↪scoring='roc_auc',return_train_score=True,verbose=2)

clf_bow.fit(X_tr_bow, y_train)

train_auc_bow = clf_bow.cv_results_['mean_train_score']
train_auc_std_bow = clf_bow.cv_results_['std_train_score']
cv_auc_bow = clf_bow.cv_results_['mean_test_score']
cv_auc_std_bow = clf_bow.cv_results_['std_test_score']
```

[22]:
```
clf_bow.best_params_
```

[22]:
```
{'max_depth': 10, 'min_samples_split': 500}
```

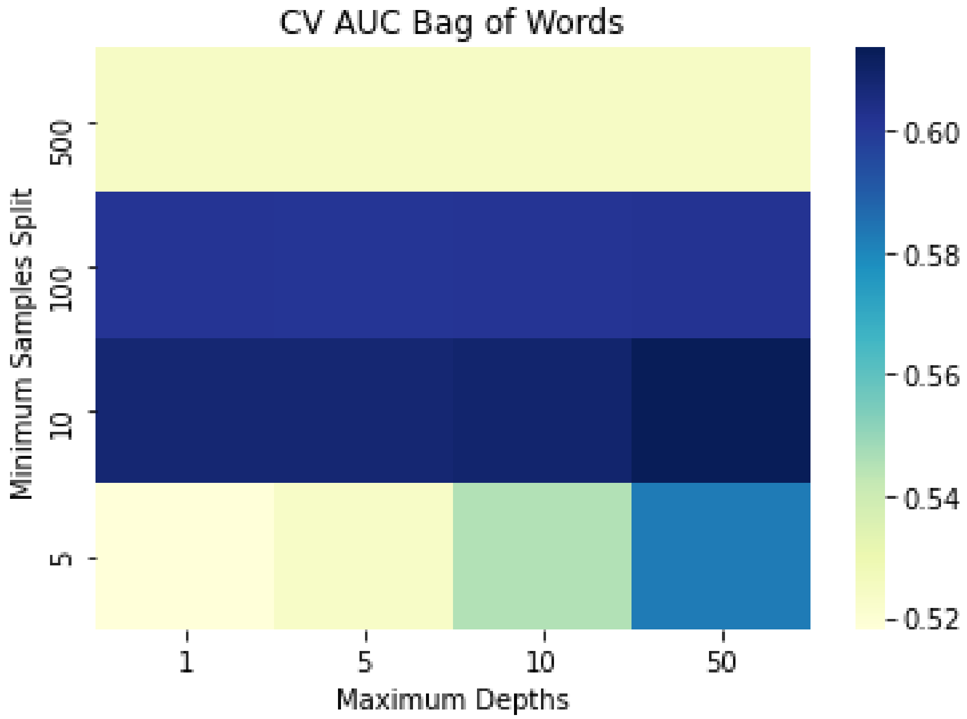
[23]:
```
def get_broken_arrays(row_arr):
    arr = np.zeros(shape=(4,4))
    for i in range(4):
        arr[i] = row_arr[i*4:(i*4)+4]
    return arr
```

[24]:
```
hm = sns.heatmap(get_broken_arrays(train_auc_bow), vmin=np.amin(train_auc_bow),␣
 ↪vmax=np.amax(train_auc_bow), xticklabels=depths, yticklabels=splits[::-1],␣
 ↪cmap="YlGnBu")
plt.title("Train AUC Bag of Words")
plt.xlabel("Maximum Depths")
plt.ylabel("Minimum Samples Split")
plt.show()
```



[25]:

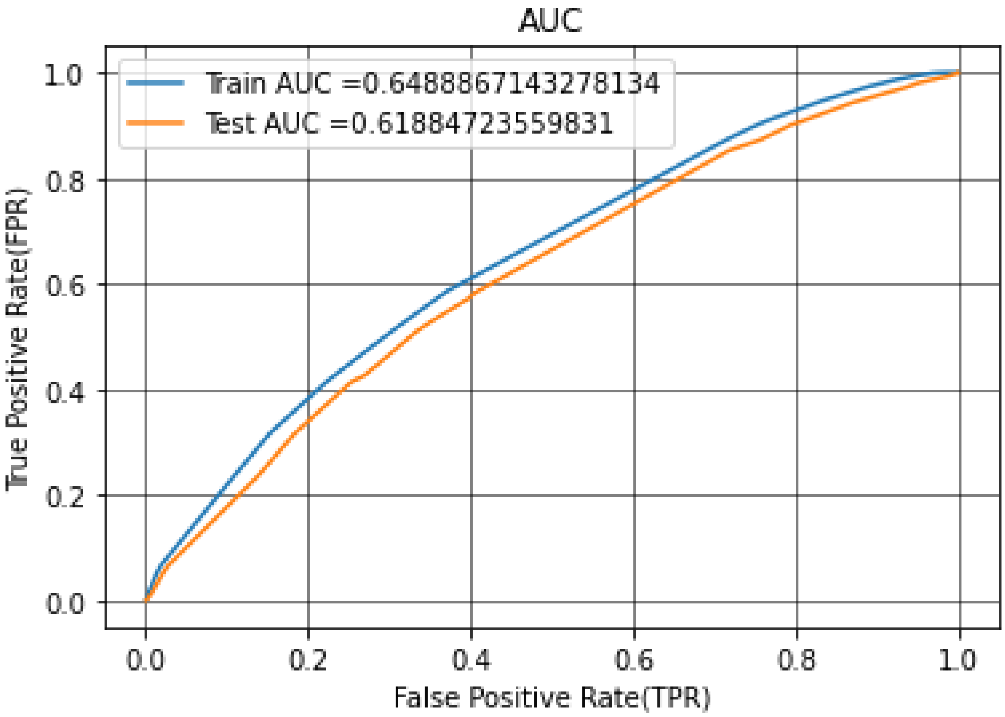
```
hm = sns.heatmap(get_broken_arrays(cv_auc_bow), vmin=np.amin(cv_auc_bow),␣
 ↪vmax=np.amax(cv_auc_bow), xticklabels=depths, yticklabels=splits[::-1],␣
 ↪cmap="YlGnBu")
plt.title("CV AUC Bag of Words")
plt.xlabel("Maximum Depths")
plt.ylabel("Minimum Samples Split")
plt.show()
```



3.2 Applying the Decision Tree Classifier with the best parameters

[26]:

```
dt_bow = DecisionTreeClassifier(max_depth = 10, min_samples_split = 500)
dt_bow.fit(X_tr_bow, y_train)

y_train_pred_bow = dt_bow.predict_proba(X_tr_bow)
y_test_pred_bow = dt_bow.predict_proba(X_te_bow)

train_fpr_bow, train_tpr_bow, tr_thresholds_bow = roc_curve(y_train,␣
 ↪y_train_pred_bow[:,1])
test_fpr_bow, test_tpr_bow, te_thresholds_bow = roc_curve(y_test,␣
 ↪y_test_pred_bow[:,1])

plt.plot(train_fpr_bow, train_tpr_bow, label="Train AUC␣
 ↪="+str(auc(train_fpr_bow, train_tpr_bow)))
plt.plot(test_fpr_bow, test_tpr_bow, label="Test AUC ="+str(auc(test_fpr_bow,␣
 ↪test_tpr_bow)))
plt.legend()
plt.xlabel("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

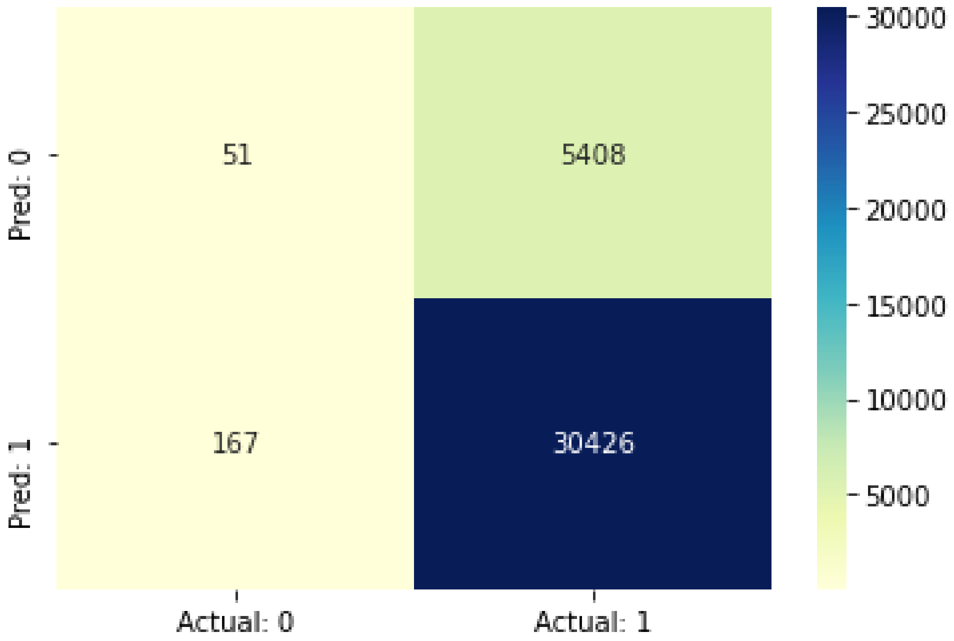


3.3 Displaying the Confusion Matrix

```
[27]: y_preds_bow = dt_bow.predict(X_te_bow)
      conf_mat_bow = confusion_matrix(y_test, y_preds_bow)
```

```
[28]: df_cm = pd.DataFrame(conf_mat_bow, index = ["Pred: 0", "Pred: 1"], columns =␣
       ↪["Actual: 0", "Actual: 1"])
      plt.figure(figsize = (6,4))
      sns.heatmap(df_cm, annot=True, fmt='g', cmap="YlGnBu")
```

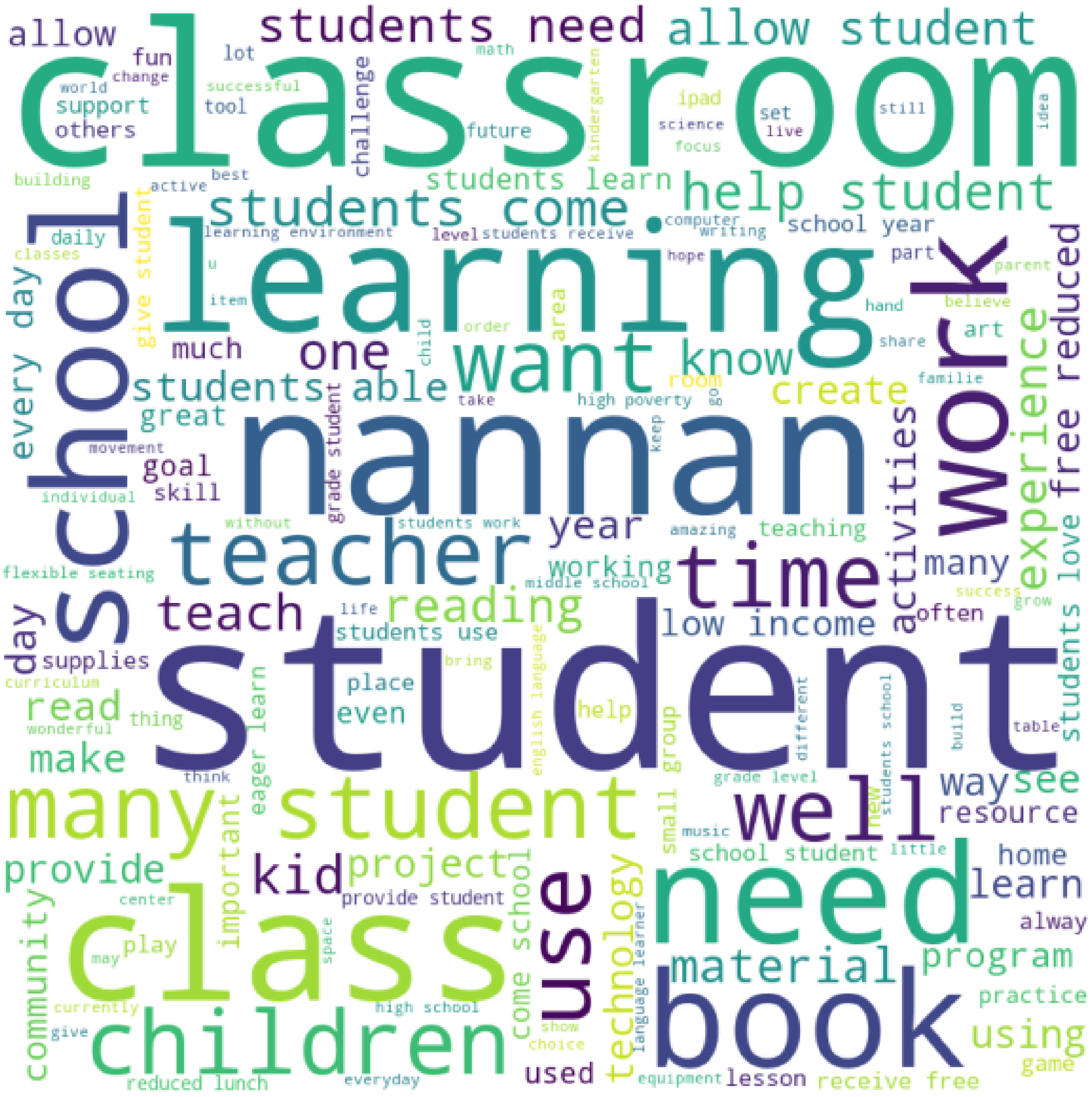
```
[28]: <AxesSubplot:>
```



[29]:
```
fp_indices = []
for i in range(len(y_preds_bow)):
    if(y_test[i] == 0 and y_preds_bow[i] == 1):
        fp_indices.append(i)
```

3.4 Showing the wordcloud of the words in False Positives

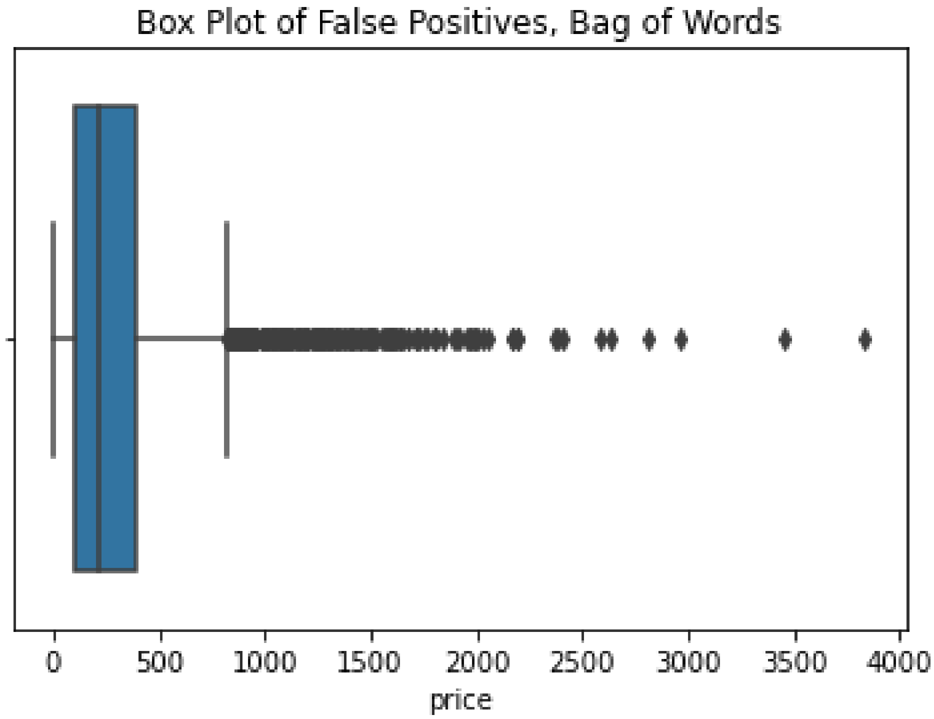
[30]:
```
word_list = ""
stopwords = set(STOPWORDS)
for i in fp_indices:
    essay = data.iloc[i]['essay']
    tokens = essay.split()
    for j in range(len(tokens)):
        tokens[j] = tokens[j].lower()
    word_list += " ".join(tokens)+ " "

wordcloud = WordCloud(width = 800, height = 800, background_color = 'white',
                        stopwords = stopwords, min_font_size = 10).
 ↪generate(word_list)
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



3.5 Boxplot of Prices of False Positives

```
[31]: ax = sns.boxplot(data.iloc[fp_indices]['price'])
      ax.set(title="Box Plot of False Positives, Bag of Words")
```

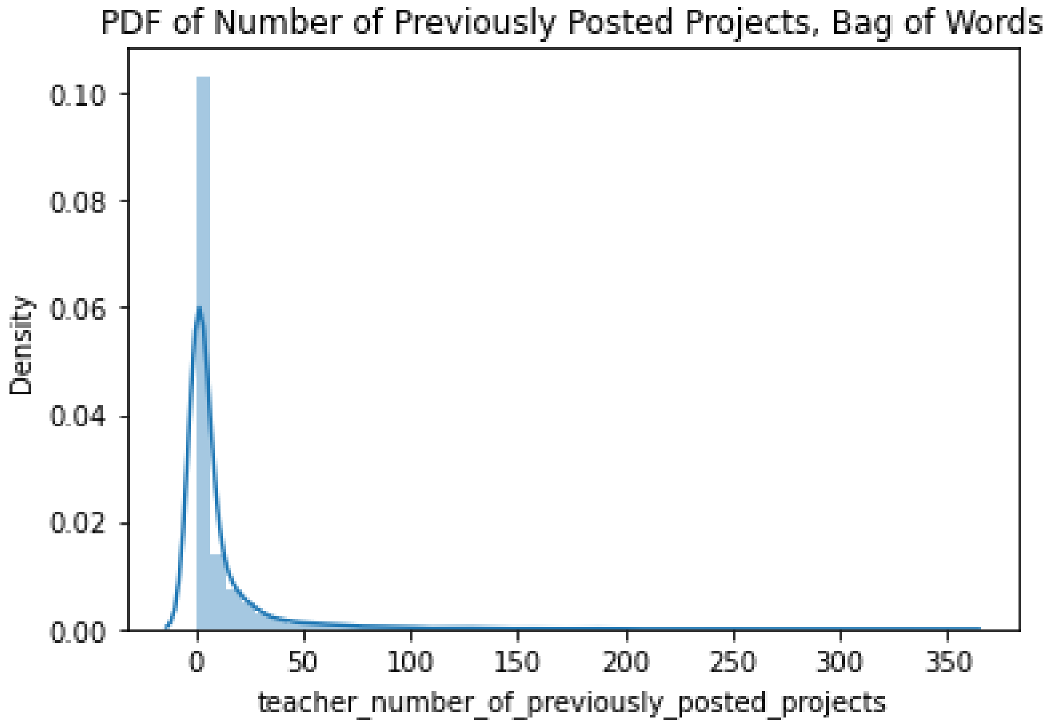
```
[31]: [Text(0.5, 1.0, 'Box Plot of False Positives, Bag of Words')]
```



3.6 PDF of Previous Posts of False Positives

```
[32]: ax = sns.distplot(data.
       ↪iloc[fp_indices]['teacher_number_of_previously_posted_projects'])
      ax.set(title="PDF of Number of Previously Posted Projects, Bag of Words")
```

```
[32]: [Text(0.5, 1.0, 'PDF of Number of Previously Posted Projects, Bag of Words')]
```



4 TFIDF

4.1 Cross Validation to get the best parameters

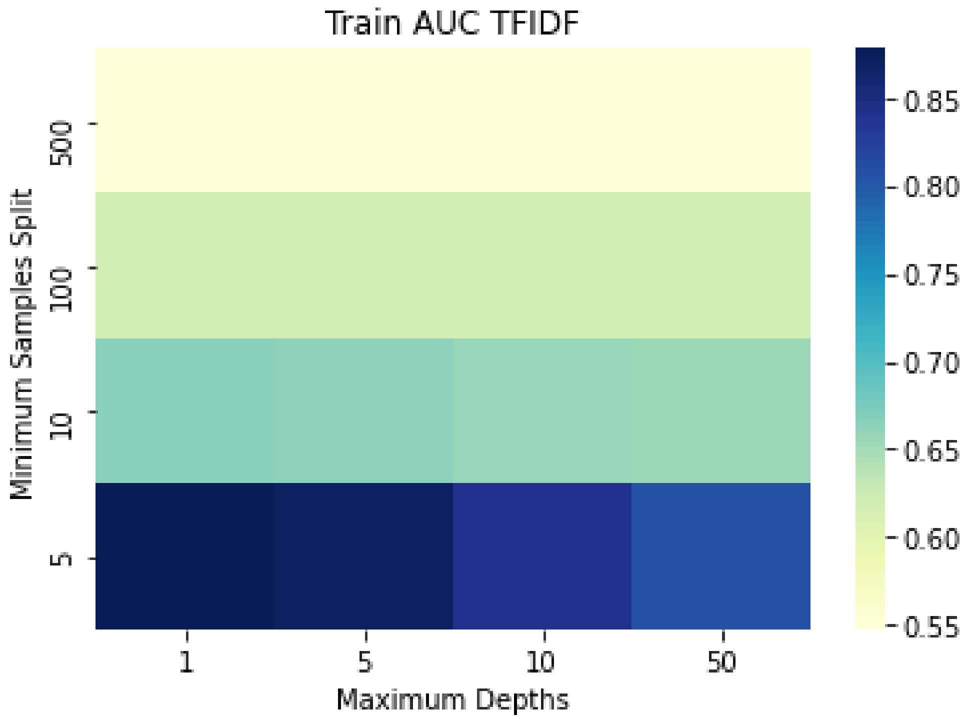
```
[ ]: dt_tfidf = DecisionTreeClassifier()

     parameters = {'max_depth':[1,5,10,50], 'min_samples_split':[5,10,100,500]}

     clf_tfidf = GridSearchCV(dt_tfidf, parameters, cv= 5,␣
      ↪scoring='roc_auc',return_train_score=True,verbose=2)

     clf_tfidf.fit(X_tr_tfidf, y_train)

     train_auc_tfidf = clf_tfidf.cv_results_['mean_train_score']
     train_auc_std_tfidf = clf_tfidf.cv_results_['std_train_score']
     cv_auc_tfidf = clf_tfidf.cv_results_['mean_test_score']
     cv_auc_std_tfidf = clf_tfidf.cv_results_['std_test_score']
```

```
[34]: clf_tfidf.best_params_
```

```
[34]: {'max_depth': 10, 'min_samples_split': 500}
```

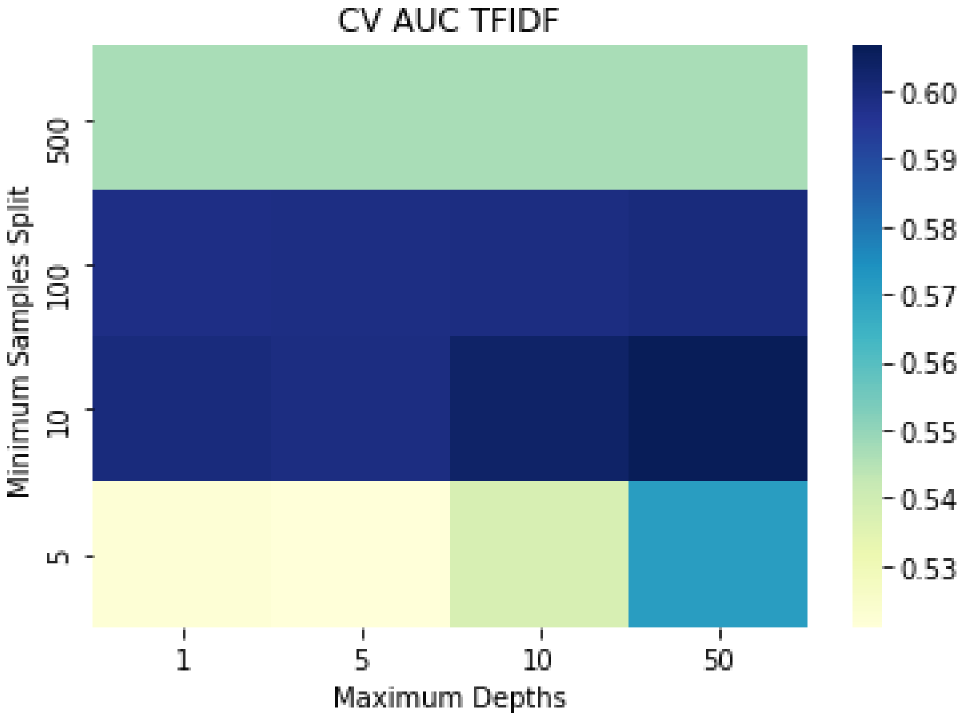
[35]:

```
hm = sns.heatmap(get_broken_arrays(train_auc_tfidf), vmin=np.
 ↪amin(train_auc_tfidf), vmax=np.amax(train_auc_tfidf), xticklabels=depths,␣
 ↪yticklabels=splits[::-1], cmap="YlGnBu")
plt.title("Train AUC TFIDF")
plt.xlabel("Maximum Depths")
plt.ylabel("Minimum Samples Split")
plt.show()
```



[36]:

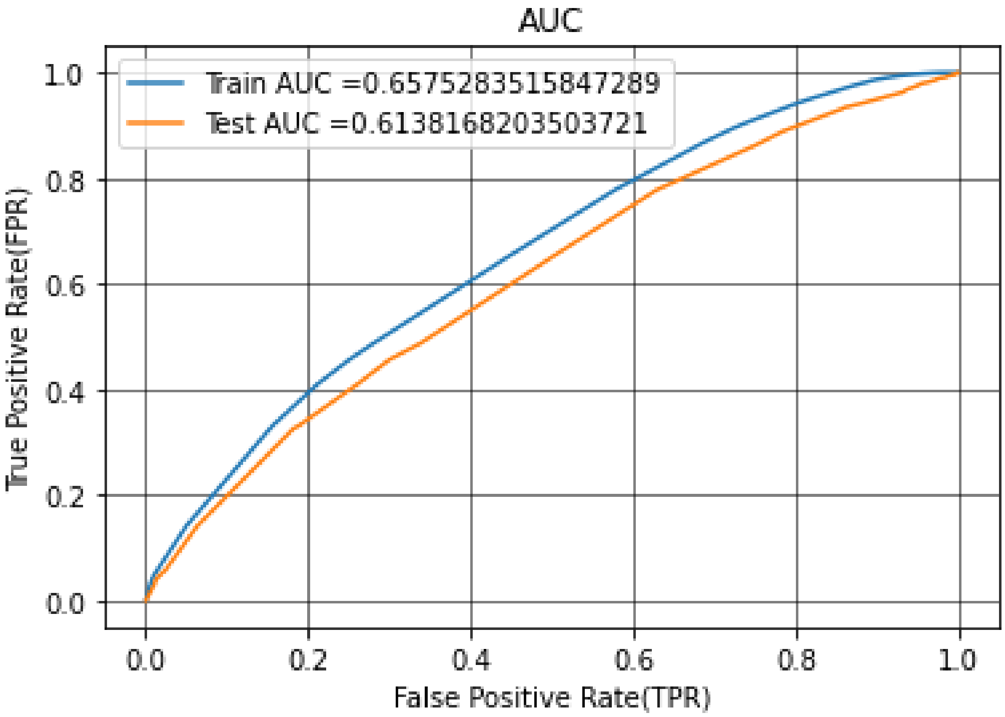
```
hm = sns.heatmap(get_broken_arrays(cv_auc_tfidf), vmin=np.amin(cv_auc_tfidf),␣
 ↪vmax=np.amax(cv_auc_tfidf), xticklabels=depths, yticklabels=splits[::-1],␣
 ↪cmap="YlGnBu")
plt.title("CV AUC TFIDF")
plt.xlabel("Maximum Depths")
plt.ylabel("Minimum Samples Split")
plt.show()
```



4.2 Applying Decision Tree Classifier with the best parameters

[37]:

```
dt_tfidf = DecisionTreeClassifier(max_depth = 10, min_samples_split = 500)
dt_tfidf.fit(X_tr_tfidf, y_train)

y_train_pred_tfidf = dt_tfidf.predict_proba(X_tr_tfidf)
y_test_pred_tfidf = dt_tfidf.predict_proba(X_te_tfidf)

train_fpr_tfidf, train_tpr_tfidf, tr_thresholds_tfidf = roc_curve(y_train,␣
 ↪y_train_pred_tfidf[:,1])
test_fpr_tfidf, test_tpr_tfidf, te_thresholds_tfidf = roc_curve(y_test,␣
 ↪y_test_pred_tfidf[:,1])

plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="Train AUC␣
 ↪="+str(auc(train_fpr_tfidf, train_tpr_tfidf)))
plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="Test AUC␣
 ↪="+str(auc(test_fpr_tfidf, test_tpr_tfidf)))
plt.legend()
plt.xlabel("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

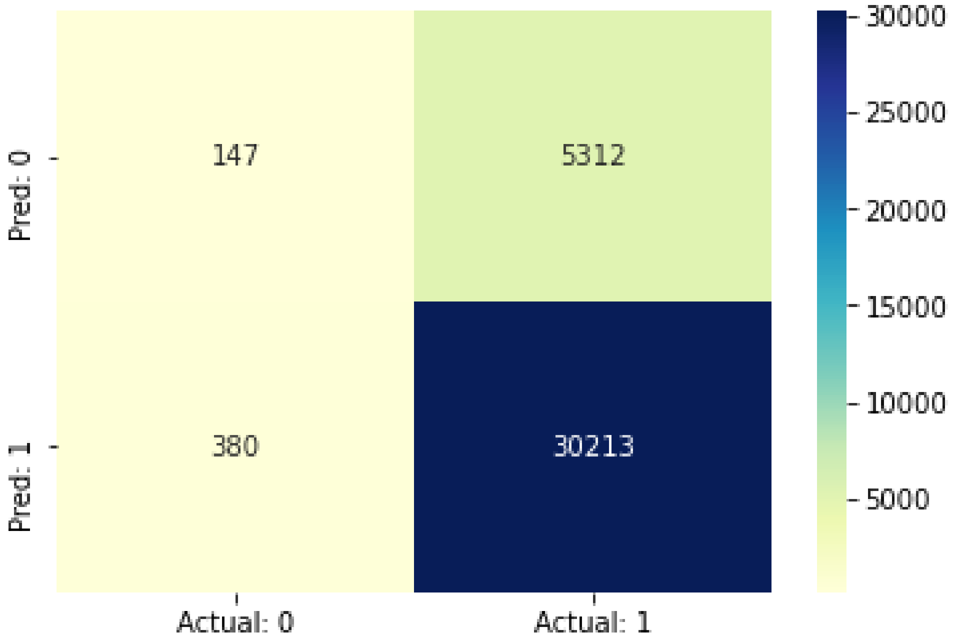


4.3 Displaying the Confusion Matrix

[38]:
```
y_preds_tfidf = dt_tfidf.predict(X_te_tfidf)
conf_mat_tfidf = confusion_matrix(y_test, y_preds_tfidf)
```

[39]:
```
df_cm = pd.DataFrame(conf_mat_tfidf, index = ["Pred: 0", "Pred: 1"], columns =␣
 ↪["Actual: 0", "Actual: 1"])
plt.figure(figsize = (6,4))
sns.heatmap(df_cm, annot=True, fmt='g', cmap="YlGnBu")
```

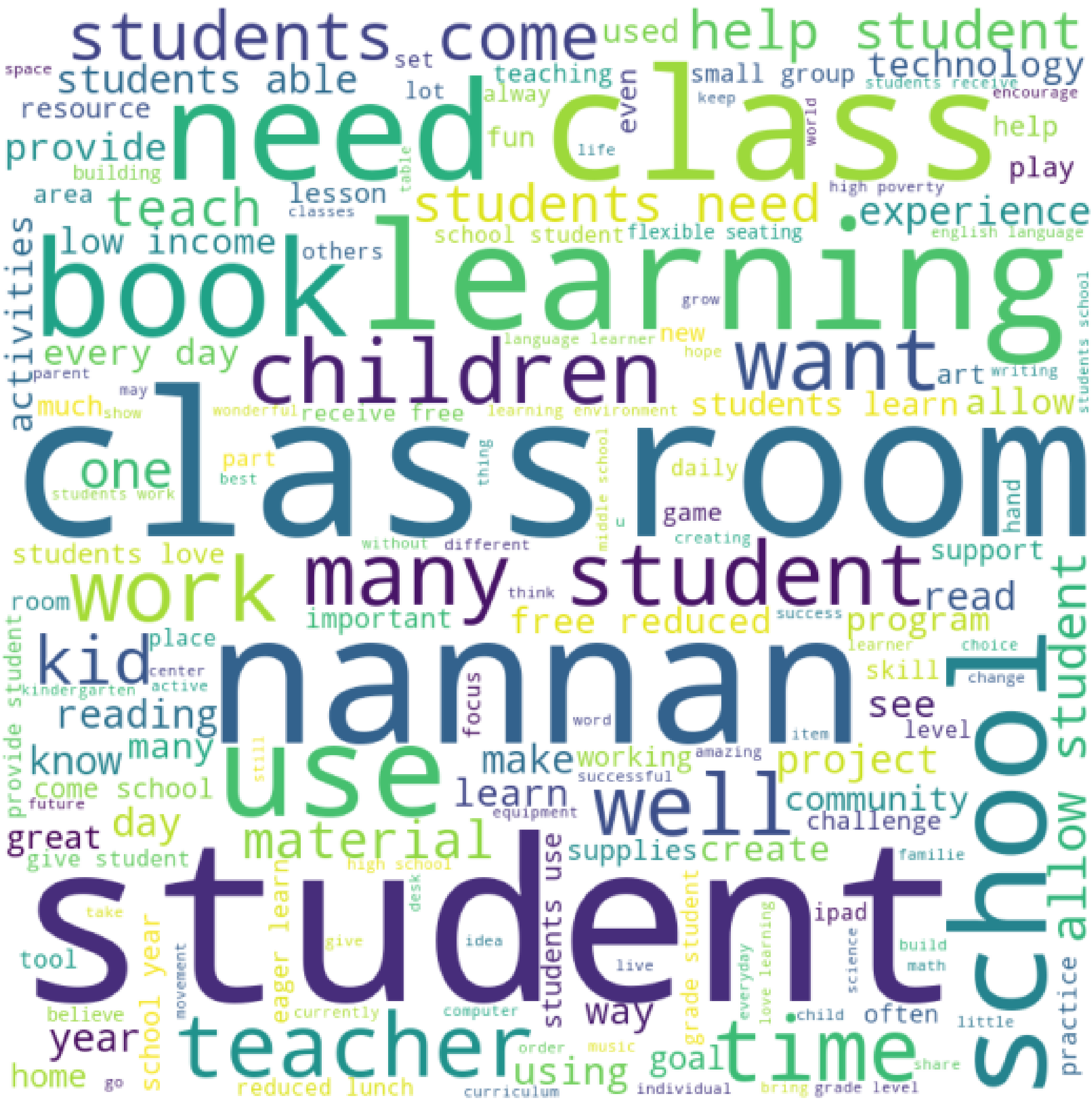
[39]:
```
<AxesSubplot:>
```



4.4 Displaying the wordcloud of essays of False Positives

[40]:
```
fp_indices = []
for i in range(len(y_preds_tfidf)):
    if(y_test[i] == 0 and y_preds_tfidf[i] == 1):
        fp_indices.append(i)
```

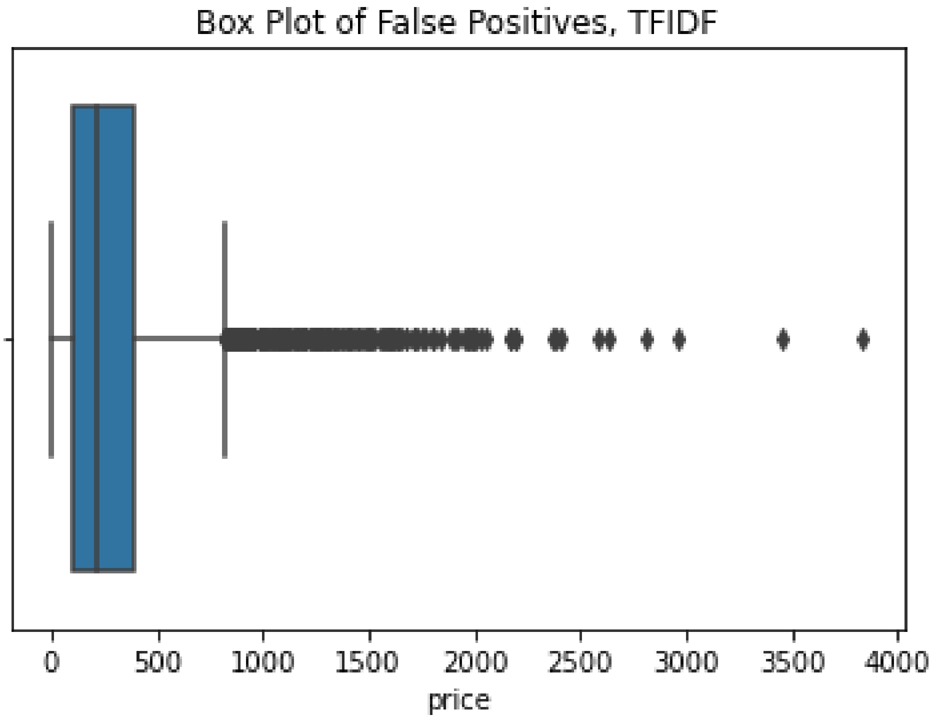
[41]:
```
word_list = ""
stopwords = set(STOPWORDS)
for i in fp_indices:
    essay = data.iloc[i]['essay']
    tokens = essay.split()
    for j in range(len(tokens)):
        tokens[j] = tokens[j].lower()
    word_list += " ".join(tokens)+ " "

wordcloud = WordCloud(width = 800, height = 800, background_color = 'white',
                        stopwords = stopwords, min_font_size = 10).
 ↪generate(word_list)
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



4.5 Box Plot of Prices of False Positives

```
[42]: ax = sns.boxplot(data.iloc[fp_indices]['price'])
      ax.set(title="Box Plot of False Positives, TFIDF")
```

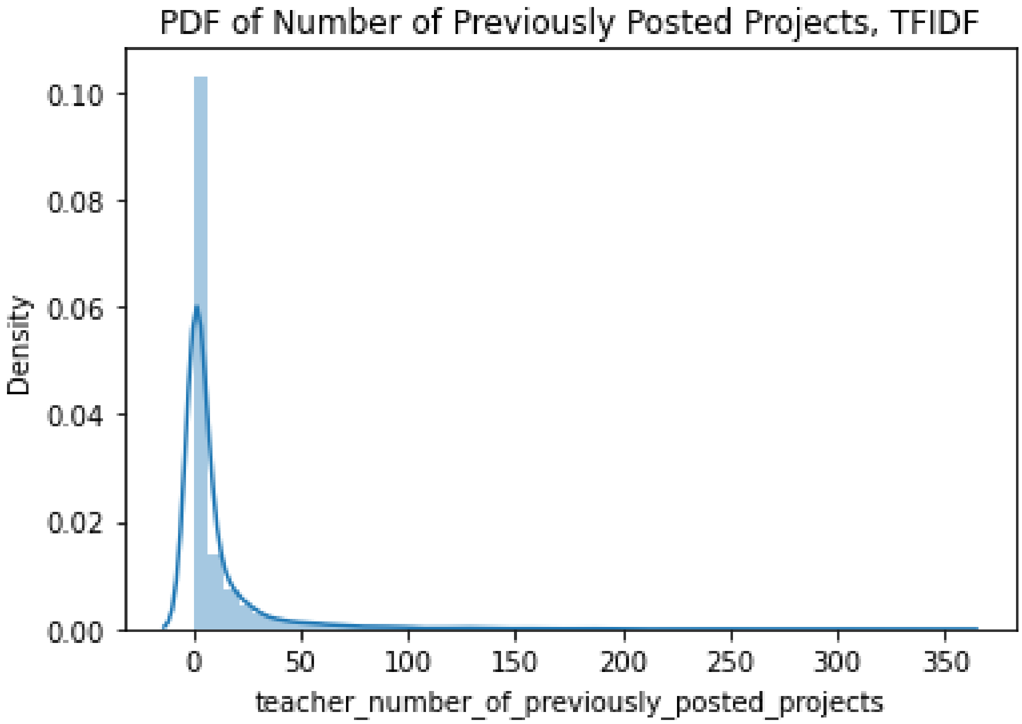
```
[42]: [Text(0.5, 1.0, 'Box Plot of False Positives, TFIDF')]
```



4.6 PDF of Previous Posts of False Postives

```
[43]: ax = sns.distplot(data.
       ↪iloc[fp_indices]['teacher_number_of_previously_posted_projects'])
      ax.set(title="PDF of Number of Previously Posted Projects, TFIDF")
```

```
[43]: [Text(0.5, 1.0, 'PDF of Number of Previously Posted Projects, TFIDF')]
```



5 TFIDF W2v

5.1 Cross Validation to get the best parameters

[ ]:

```
dt_tfidf_w2v = DecisionTreeClassifier()

parameters = {'max_depth':[1,5,10,50], 'min_samples_split':[5,10,100,500]}

clf_tfidf_w2v = GridSearchCV(dt_tfidf_w2v, parameters, cv= 5,␣
 ↪scoring='roc_auc',return_train_score=True,verbose=2)

clf_tfidf_w2v.fit(X_tr_tfidf_w2v, y_train)

train_auc_tfidf_w2v = clf_tfidf_w2v.cv_results_['mean_train_score']
train_auc_std_tfidf_w2v = clf_tfidf_w2v.cv_results_['std_train_score']
cv_auc_tfidf_w2v = clf_tfidf_w2v.cv_results_['mean_test_score']
cv_auc_std_tfidf_w2v = clf_tfidf_w2v.cv_results_['std_test_score']
```

[45]:

```
clf_tfidf_w2v.best_params_
```

[45]:

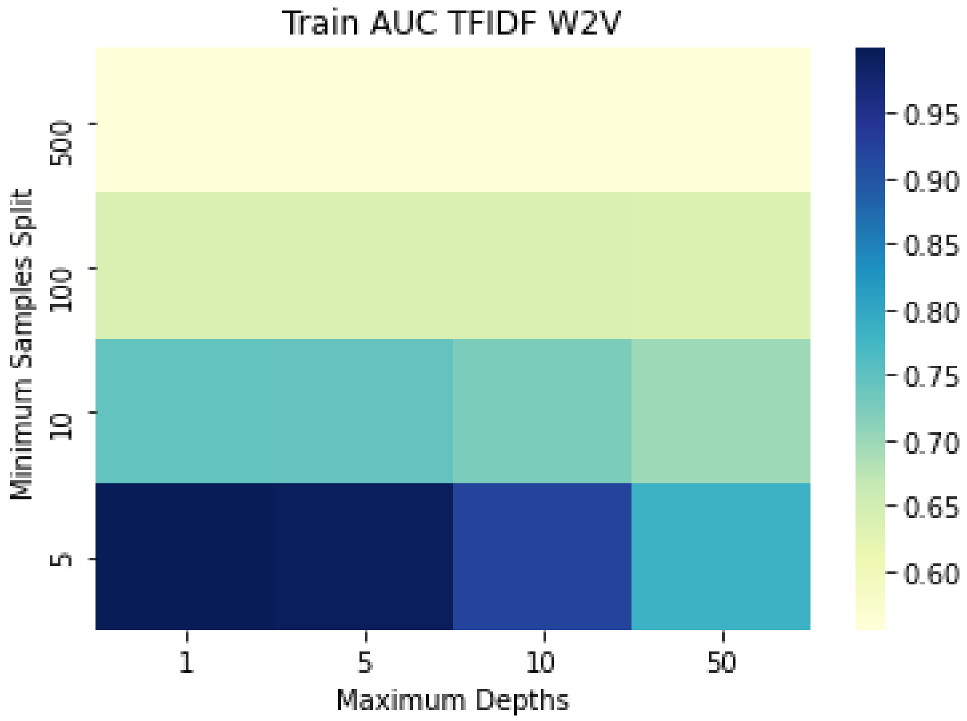
```
{'max_depth': 5, 'min_samples_split': 500}
```

[46]:

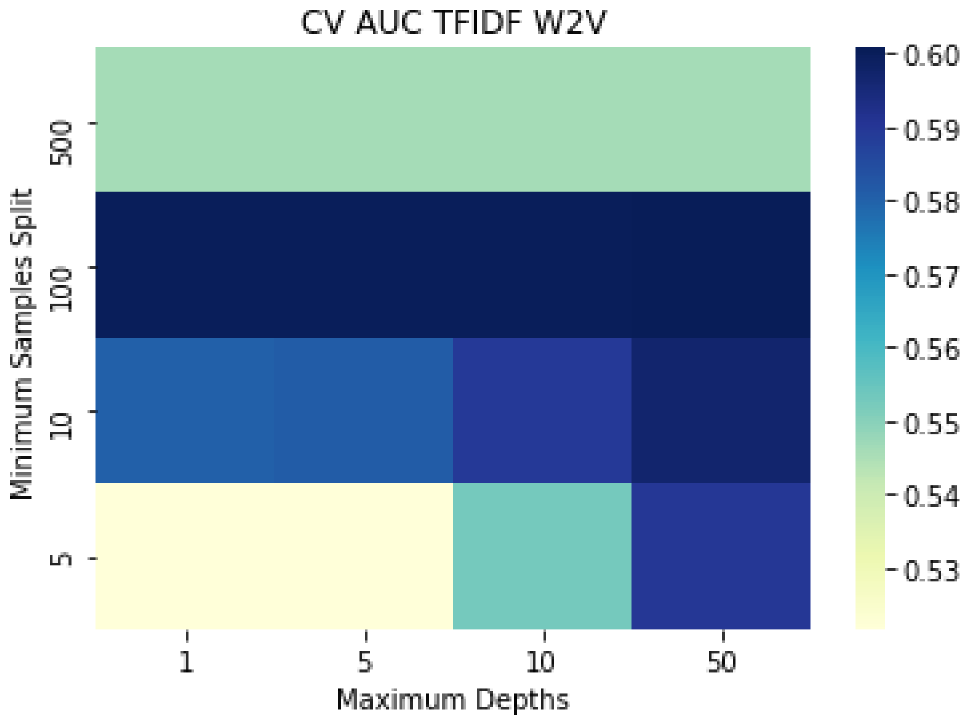
```
clf_tfidf_w2v.best_params_
```

[46]: {'max_depth': 5, 'min_samples_split': 500}

[47]:
```
hm = sns.heatmap(get_broken_arrays(train_auc_tfidf_w2v), vmin=np.
 ↪amin(train_auc_tfidf_w2v), vmax=np.amax(train_auc_tfidf_w2v),␣
 ↪xticklabels=depths, yticklabels=splits[::-1], cmap="YlGnBu")
plt.title("Train AUC TFIDF W2V")
plt.xlabel("Maximum Depths")
plt.ylabel("Minimum Samples Split")
plt.show()
```

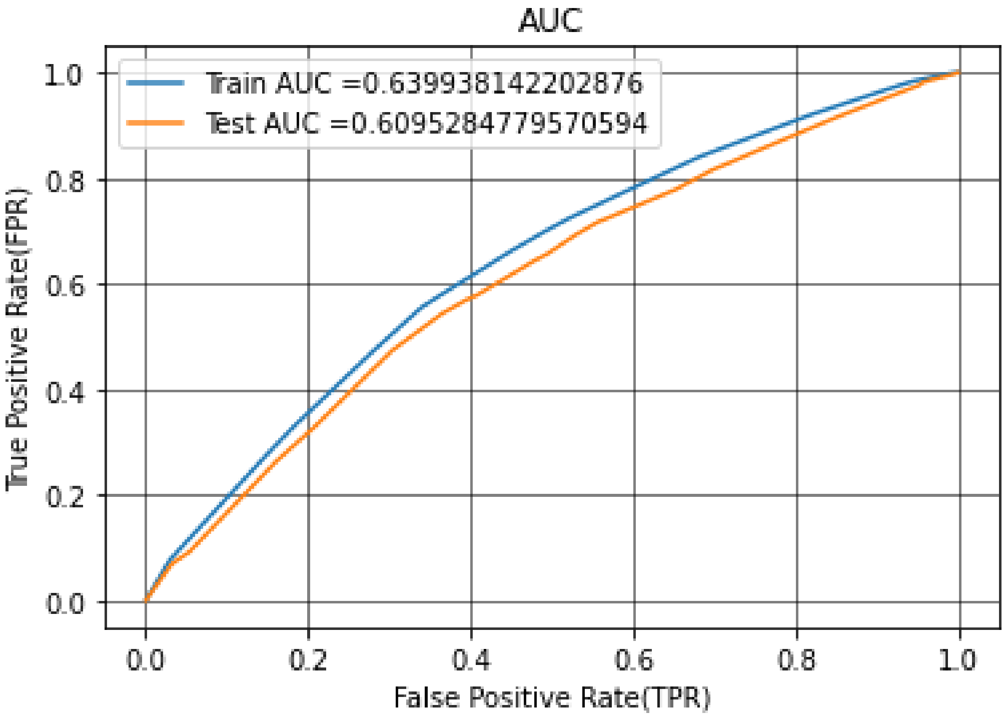


[48]:
```
hm = sns.heatmap(get_broken_arrays(cv_auc_tfidf_w2v), vmin=np.
 ↪amin(cv_auc_tfidf_w2v), vmax=np.amax(cv_auc_tfidf_w2v), xticklabels=depths,␣
 ↪yticklabels=splits[::-1], cmap="YlGnBu")
plt.title("CV AUC TFIDF W2V")
plt.xlabel("Maximum Depths")
plt.ylabel("Minimum Samples Split")
plt.show()
```



5.2 Applying Decision Tree Classifier with the best parameters

[49]:
```
dt_tfidf_w2v = DecisionTreeClassifier(max_depth = 5, min_samples_split = 100)
dt_tfidf_w2v.fit(X_tr_tfidf_w2v, y_train)

y_train_pred_tfidf_w2v = dt_tfidf_w2v.predict_proba(X_tr_tfidf_w2v)
y_test_pred_tfidf_w2v = dt_tfidf_w2v.predict_proba(X_te_tfidf_w2v)

train_fpr_tfidf_w2v, train_tpr_tfidf_w2v, tr_thresholds_tfidf_w2v =␣
 ↪roc_curve(y_train, y_train_pred_tfidf_w2v[:,1])
test_fpr_tfidf_w2v, test_tpr_tfidf_w2v, te_thresholds_tfidf_w2v =␣
 ↪roc_curve(y_test, y_test_pred_tfidf_w2v[:,1])

plt.plot(train_fpr_tfidf_w2v, train_tpr_tfidf_w2v, label="Train AUC␣
 ↪="+str(auc(train_fpr_tfidf_w2v, train_tpr_tfidf_w2v)))
plt.plot(test_fpr_tfidf_w2v, test_tpr_tfidf_w2v, label="Test AUC␣
 ↪="+str(auc(test_fpr_tfidf_w2v, test_tpr_tfidf_w2v)))
plt.legend()
plt.xlabel("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```

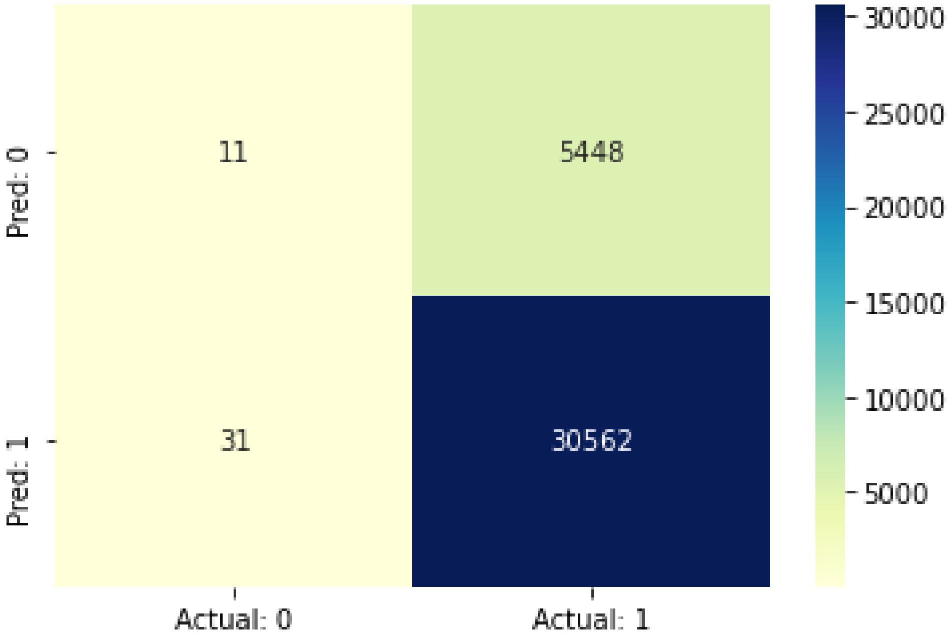


5.3 Displaying the Confusion Matrix

```
[50]: y_preds_tfidf_w2v = dt_tfidf_w2v.predict(X_te_tfidf_w2v)
      conf_mat_tfidf_w2v = confusion_matrix(y_test, y_preds_tfidf_w2v)
```

```
[51]: df_cm = pd.DataFrame(conf_mat_tfidf_w2v, index = ["Pred: 0", "Pred: 1"],␣
       ↪columns = ["Actual: 0", "Actual: 1"])
      plt.figure(figsize = (6,4))
      sns.heatmap(df_cm, annot=True, fmt='g', cmap="YlGnBu")
```

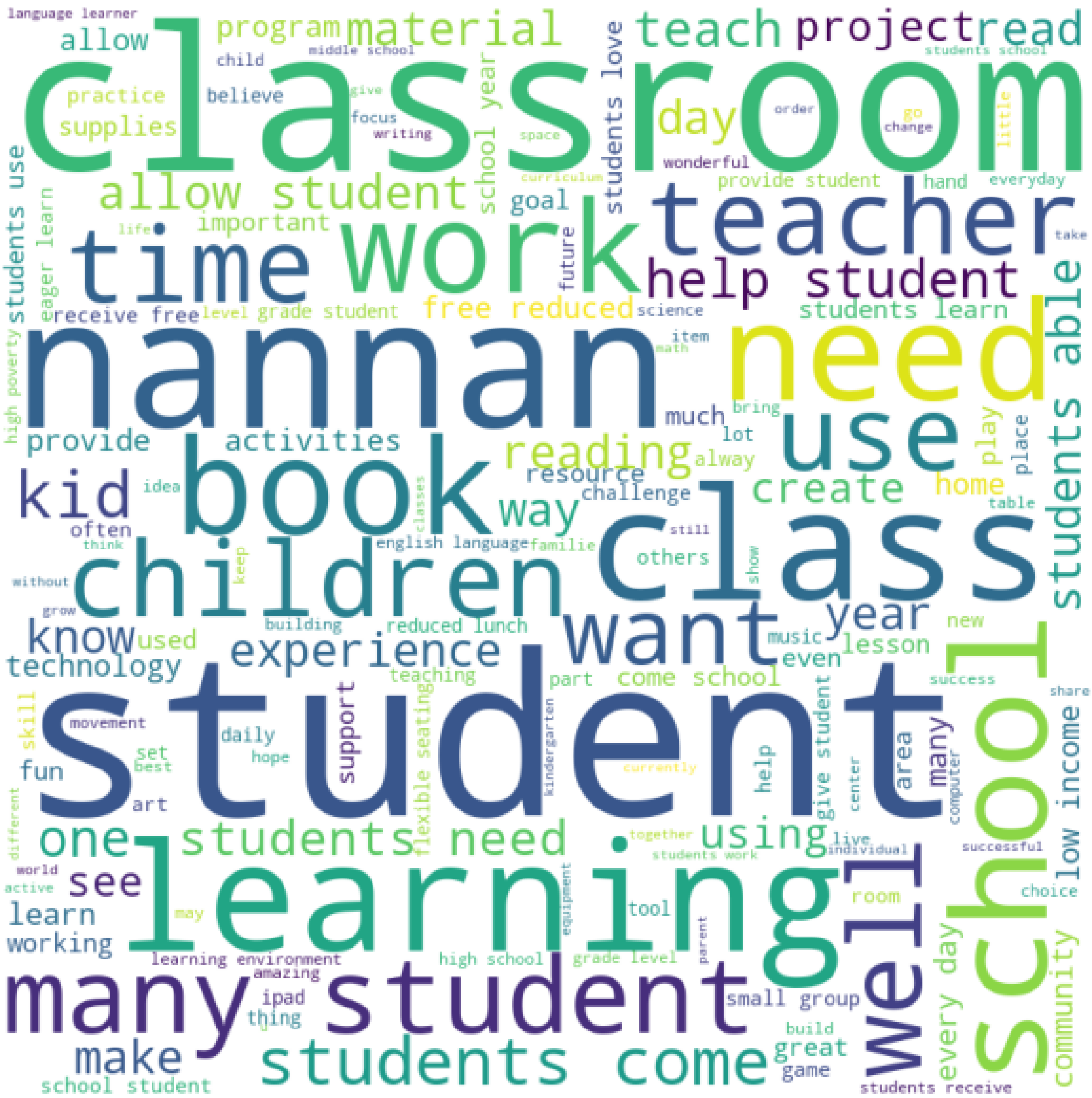
```
[51]: <AxesSubplot:>
```



5.4 Displaying the wordcloud of essays in False Positives

[52]:
```
fp_indices = []
for i in range(len(y_preds_tfidf_w2v)):
    if(y_test[i] == 0 and y_preds_tfidf_w2v[i] == 1):
        fp_indices.append(i)
```

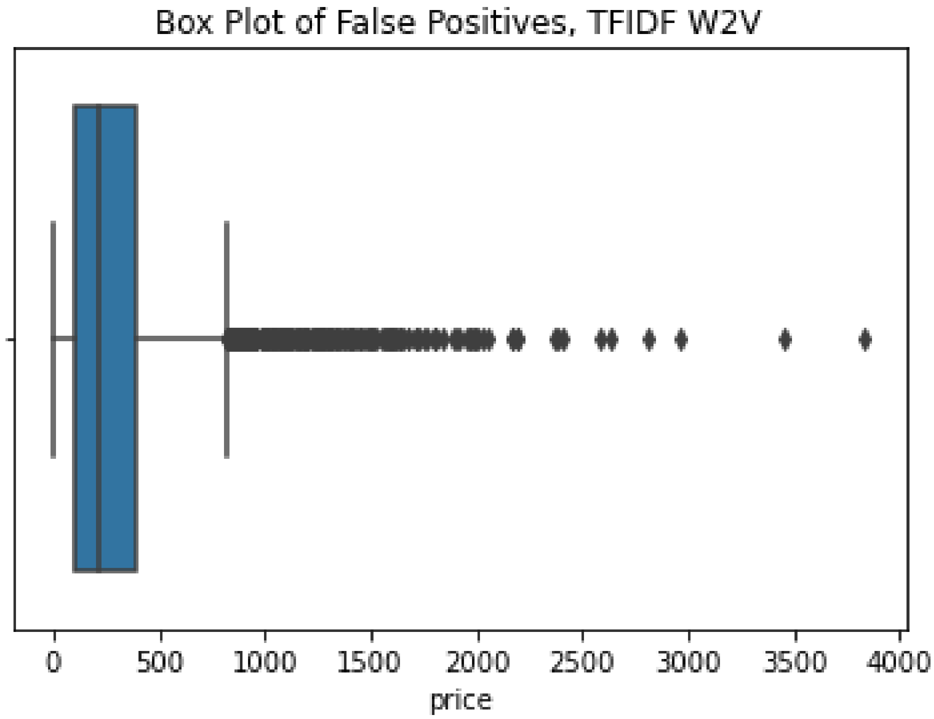
[53]:
```
word_list = ""
stopwords = set(STOPWORDS)
for i in fp_indices:
    essay = data.iloc[i]['essay']
    tokens = essay.split()
    for j in range(len(tokens)):
        tokens[j] = tokens[j].lower()
    word_list += " ".join(tokens)+ " "

wordcloud = WordCloud(width = 800, height = 800, background_color = 'white',
                        stopwords = stopwords, min_font_size = 10).
 ↪generate(word_list)
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



5.5 Displaying the BoxPlot of prices of False Positives

```
[54]: ax = sns.boxplot(data.iloc[fp_indices]['price'])
      ax.set(title="Box Plot of False Positives, TFIDF W2V")
```

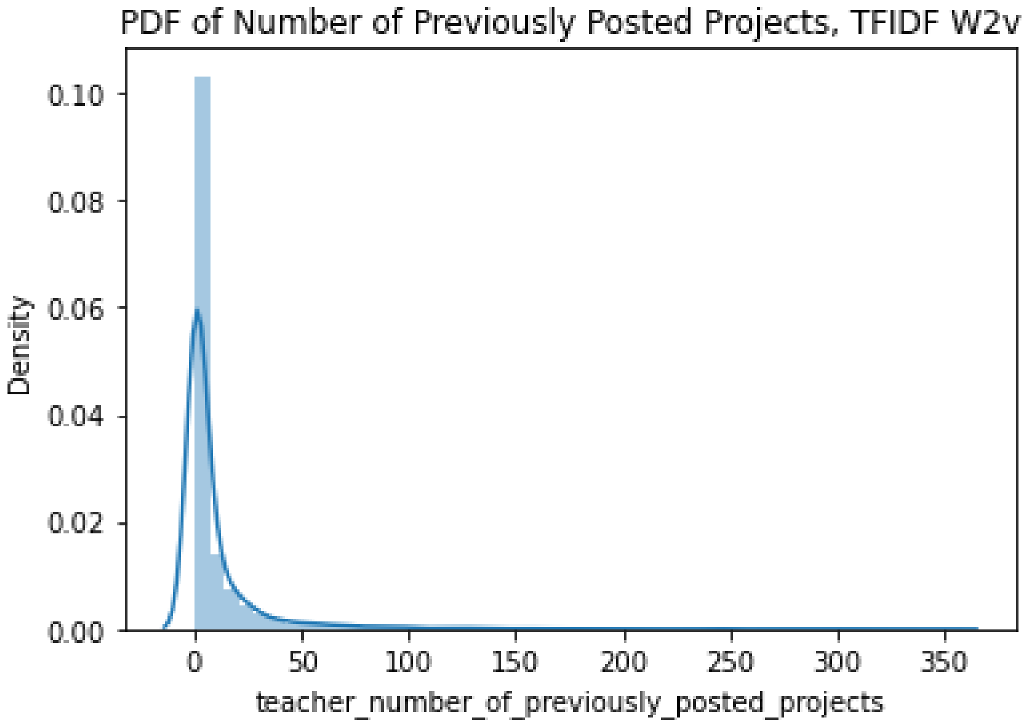
```
[54]: [Text(0.5, 1.0, 'Box Plot of False Positives, TFIDF W2V')]
```



5.6 PDF of Previous Posts of False Positives

```
[55]: ax = sns.distplot(data.
       ↪iloc[fp_indices]['teacher_number_of_previously_posted_projects'])
      ax.set(title="PDF of Number of Previously Posted Projects, TFIDF W2v")
```

```
[55]: [Text(0.5, 1.0, 'PDF of Number of Previously Posted Projects, TFIDF W2v')]
```



6 Task - 2

For this task consider **set-1** features.

- Select all the features which are having non-zero feature importance.You can get the feature importance using 'feature_importances_' (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html), discard the all other remaining features and then apply any of the model of you choice i.e. (Dession tree, Logistic Regression, Linear SVM).
- You need to do hyperparameter tuning corresponding to the model you selected and procedure in step 2 and step 3 **Note**: when you want to find the feature importance make sure you don't use max_depth parameter keep it None.
  You need to summarize the results at the end of the notebook, summarize it in the table format

6.1 Training Model on BOW Data without Max Depth

```
[71]: if_dt = DecisionTreeClassifier(min_samples_split = 500)
      if_dt.fit(X_tr_bow, y_train)
```

```
[71]: DecisionTreeClassifier(min_samples_split=500)
```

6.2 Getting the important features

```
[72]: imp_features = np.argwhere(if_dt.feature_importances_ != 0)
      imp_features
```

```
[72]: array([[   0],
              [   6],
              [  16],
              …,
              [5102],
              [5103],
              [5104]])
```

```
[73]: imp_features = [imp_features[i][0] for i in range(imp_features.shape[0])]
```

```
[74]: imp_features_train  = X_tr_bow[:, imp_features]
      imp_features_cv = X_cr_bow[:, imp_features]
      imp_features_test = X_te_bow[:, imp_features]
```

6.3 Training on only the important features

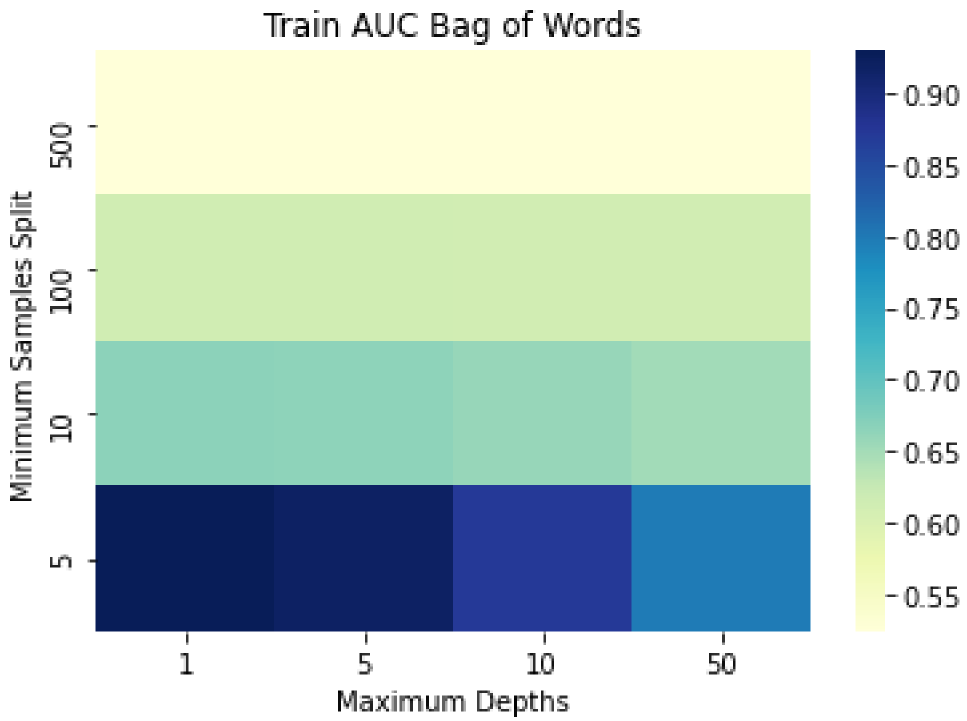
```
[ ]: if_bow = DecisionTreeClassifier()

     parameters = {'max_depth':[1,5,10,50], 'min_samples_split':[5,10,100,500]}

     if_clf_bow = GridSearchCV(if_bow, parameters, cv= 5,␣
      ↪scoring='roc_auc',return_train_score=True,verbose=2)

     if_clf_bow.fit(imp_features_train, y_train)

     if_train_auc_bow = if_clf_bow.cv_results_['mean_train_score']
     if_train_auc_std_bow = if_clf_bow.cv_results_['std_train_score']
     if_cv_auc_bow = if_clf_bow.cv_results_['mean_test_score']
     if_cv_auc_std_bow = if_clf_bow.cv_results_['std_test_score']
```

```
[76]: if_clf_bow.best_params_
```

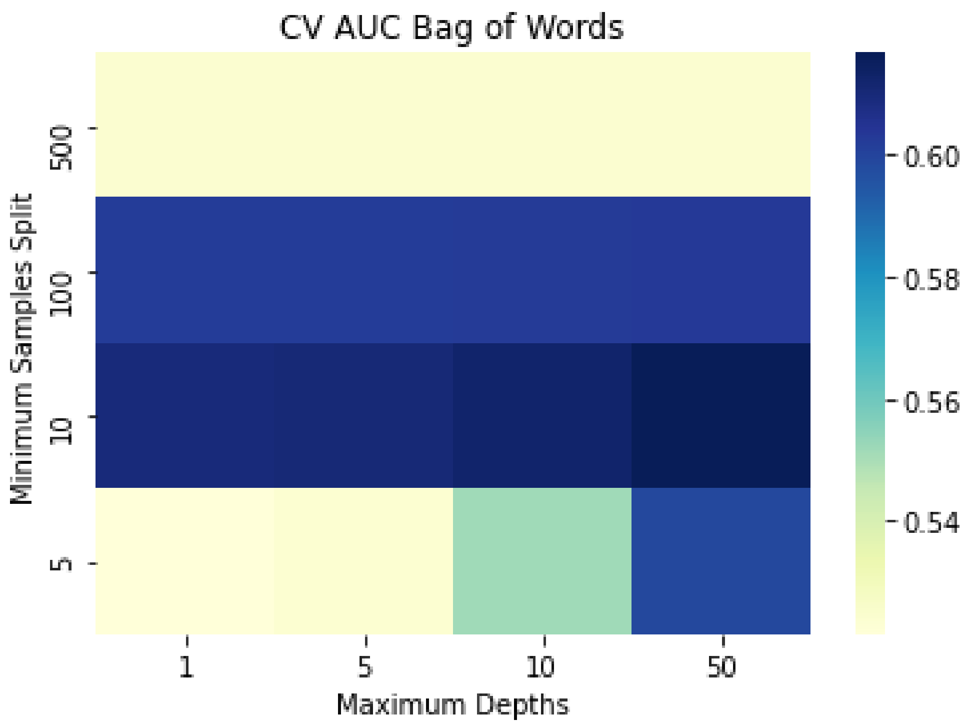
```
[76]: {'max_depth': 10, 'min_samples_split': 500}
```

6.4 Plotting the Heatmaps

```
[77]: hm = sns.heatmap(get_broken_arrays(if_train_auc_bow), vmin=np.
       ↪amin(if_train_auc_bow), vmax=np.amax(if_train_auc_bow), xticklabels=depths,␣
       ↪yticklabels=splits[::-1], cmap="YlGnBu")
      plt.title("Train AUC Bag of Words")
      plt.xlabel("Maximum Depths")
      plt.ylabel("Minimum Samples Split")
plt.show()
```

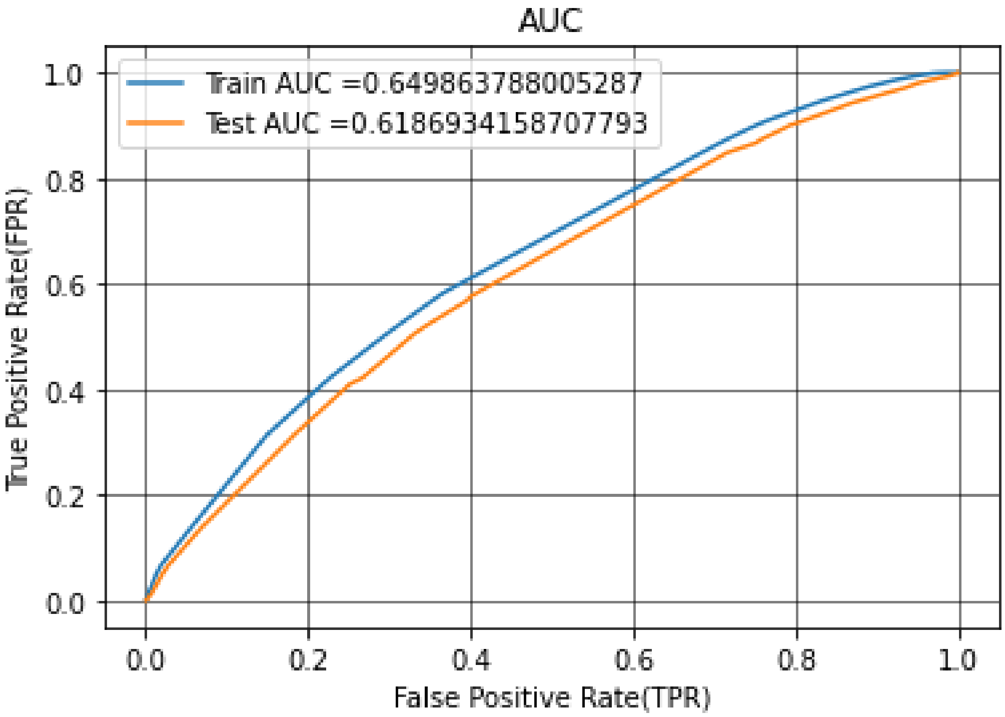


[78]:
```
hm = sns.heatmap(get_broken_arrays(if_cv_auc_bow), vmin=np.amin(if_cv_auc_bow),␣
 ↪vmax=np.amax(if_cv_auc_bow), xticklabels=depths, yticklabels=splits[::-1],␣
 ↪cmap="YlGnBu")
plt.title("CV AUC Bag of Words")
plt.xlabel("Maximum Depths")
plt.ylabel("Minimum Samples Split")
plt.show()
```



6.5 Training on the best parameters and plotting AUC Curve

```
[79]: if_dt_bow = DecisionTreeClassifier(max_depth = 10, min_samples_split = 500)
      if_dt_bow.fit(imp_features_train, y_train)

      if_y_train_pred_bow = if_dt_bow.predict_proba(imp_features_train)
      if_y_test_pred_bow = if_dt_bow.predict_proba(imp_features_test)

      if_train_fpr_bow, if_train_tpr_bow, if_tr_thresholds_bow = roc_curve(y_train,␣
       ↪if_y_train_pred_bow[:,1])
      if_test_fpr_bow, if_test_tpr_bow, if_te_thresholds_bow = roc_curve(y_test,␣
       ↪if_y_test_pred_bow[:,1])

      plt.plot(if_train_fpr_bow, if_train_tpr_bow, label="Train AUC␣
       ↪="+str(auc(if_train_fpr_bow, if_train_tpr_bow)))
      plt.plot(if_test_fpr_bow, if_test_tpr_bow, label="Test AUC␣
       ↪="+str(auc(if_test_fpr_bow, if_test_tpr_bow)))
      plt.legend()
      plt.xlabel("False Positive Rate(TPR)")
      plt.ylabel("True Positive Rate(FPR)")
      plt.title("AUC")
      plt.grid(color='black', linestyle='-', linewidth=0.5)
      plt.show()
```

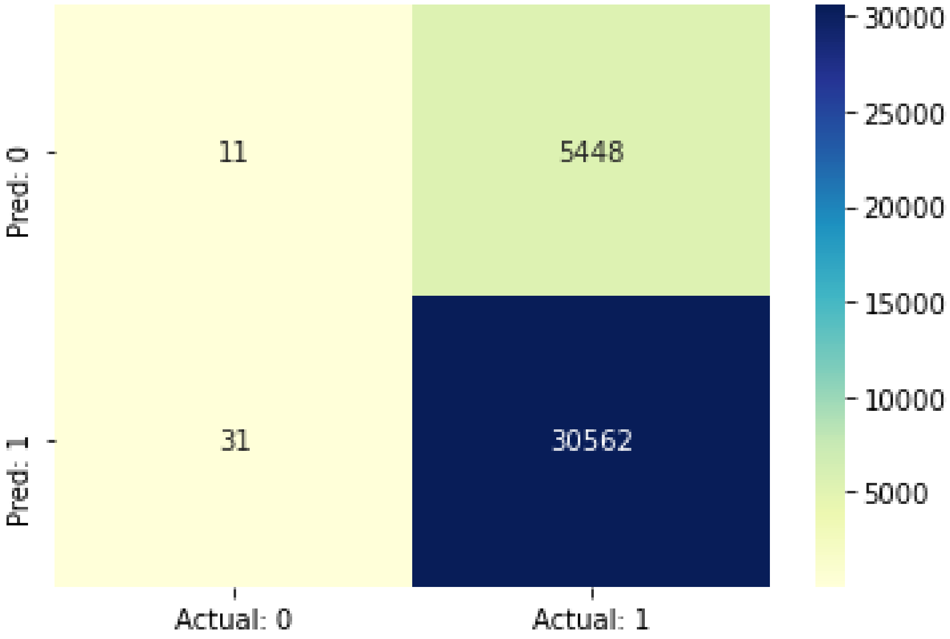


```
[80]: if_y_preds_bow = if_dt_bow.predict(imp_features_test)
      if_conf_mat_bow = confusion_matrix(y_test, if_y_preds_bow)
```

6.6 Displaying the Confusion Matrix

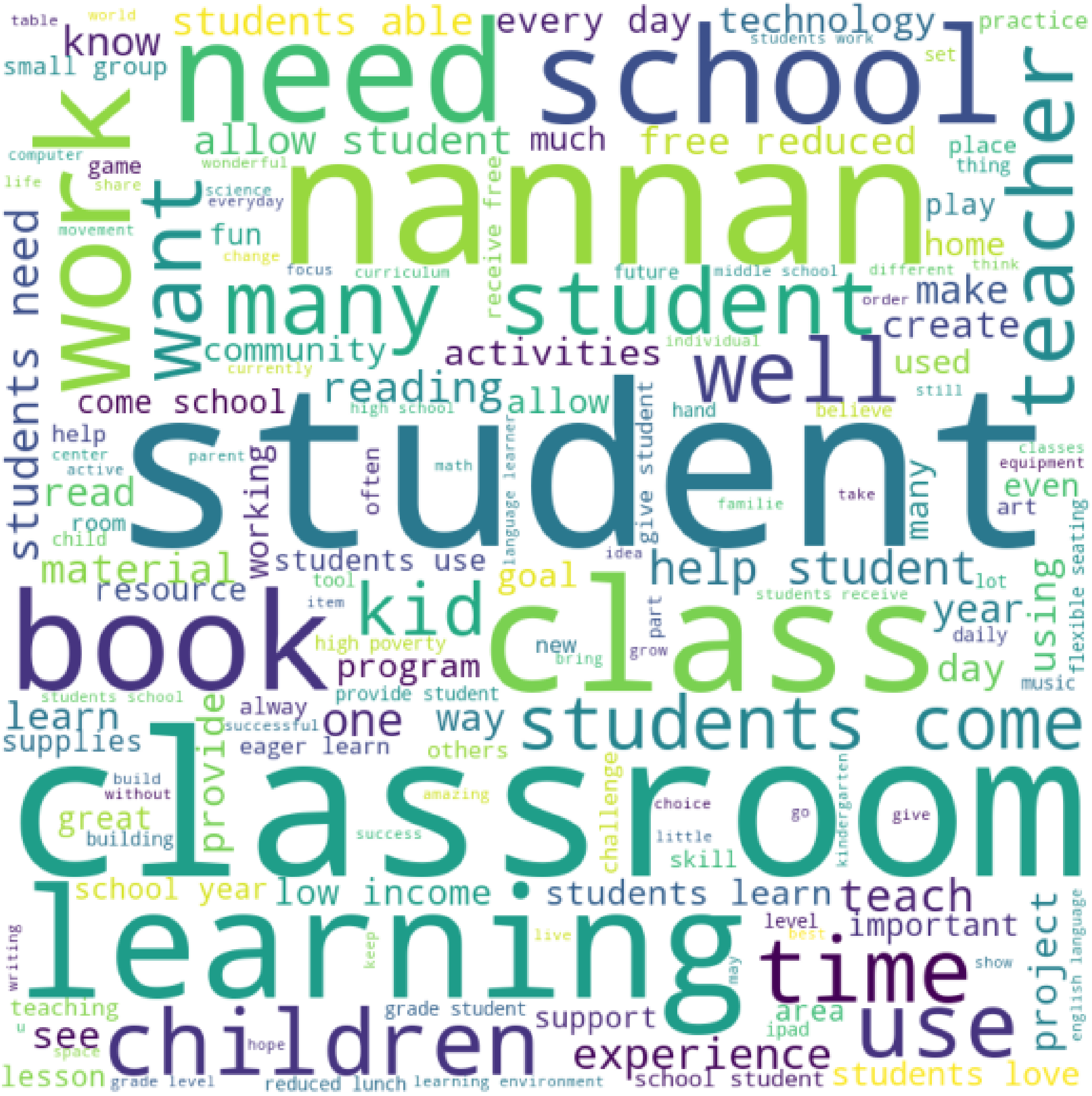
```
[81]: if_df_cm = pd.DataFrame(if_conf_mat_bow, index = ["Pred: 0", "Pred: 1"],␣
       ↪columns = ["Actual: 0", "Actual: 1"])
      plt.figure(figsize = (6,4))
      sns.heatmap(df_cm, annot=True, fmt='g', cmap="YlGnBu")
```

```
[81]: <AxesSubplot:>
```



[82]:
```
fp_indices = []
for i in range(len(if_y_preds_bow)):
    if(y_test[i] == 0 and if_y_preds_bow[i] == 1):
        fp_indices.append(i)
```

[83]:
```
word_list = ""
stopwords = set(STOPWORDS)
for i in fp_indices:
    essay = data.iloc[i]['essay']
    tokens = essay.split()
    for j in range(len(tokens)):
        tokens[j] = tokens[j].lower()
    word_list += " ".join(tokens)+ " "

wordcloud = WordCloud(width = 800, height = 800, background_color = 'white',
                        stopwords = stopwords, min_font_size = 10).
 ↪generate(word_list)
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



7 Displaying the final results

```
[85]: df = pd.DataFrame([["Bag of Words", "Decision Tree", 10, 500, 0.65359, 0.
       ↪62339], ["TFIDf", "Decision Tree", 10, 500, 0.65747, 0.62319], ["TFIDF W2V",␣
       ↪"Decision Tree", 5, 100, 0.63984, 0.60992], ["Bag of Words with Important␣
       ↪Features", "Decision Tree", 10, 500, 0.65359, 0.62339]], columns =␣
       ↪["Vectorizer", "Model", "Max Depth", "Minimum Splits", "Train AUC", "Test␣
       ↪AUC"])
      df
```

```
[85]:                             Vectorizer          Model  Max Depth  \
      0                         Bag of Words  Decision Tree         10
```

```
1                                 TFIDf  Decision Tree         10
2                             TFIDF W2V  Decision Tree          5
3  Bag of Words with Important Features  Decision Tree         10

   Minimum Splits  Train AUC  Test AUC
0             500    0.65359   0.62339
1             500    0.65747   0.62319
2             100    0.63984   0.60992
3             500    0.65359   0.62339
```