RAMAIAH UNIVERSITY OF APPLIED SCIENCES

Assignment

Course Code	CSC402A
Course Name	Data Mining
Programme	B.Tech
Department	CSE
Faculty	FET

Name of the Student	Satyajit Ghana
Reg. No.	17ETCS002159
Semester/Year	07/2021
Course Leader(s)	Prof. Mohan Kumar

Declaration Sheet

Student Name	Satyajit Ghana		
Reg. No	17ETCS002159		
Programme	B.Tech	Semester/Year	07/2021
Course Code	CSC402A		
Course Title	Data Mining		
Course Date		to	
Course Leader	Prof. Mohan Kumar		

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Contents

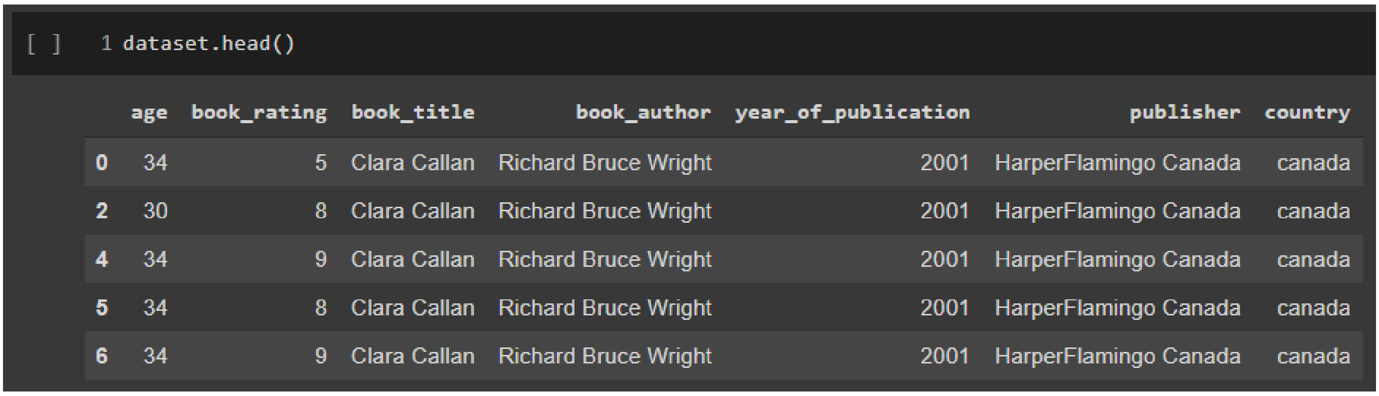
1 Question 1

Solution to Question No. 1 Part B

1.1 Introduction

A few more pre-processing steps were performed on the dataset, to clean it, which will then be used by the learning algorithms.

Columns like [user_id, isbn, city, state] are dropped, since they don't contribute to the rating classification, and they don't really tell a lot about rating of the book (not much of information gain), also something to note (since we are considering the country) column, that most of the users (~70%) are from usa, which may/may not contribute to lot to the accuracy of the classification.



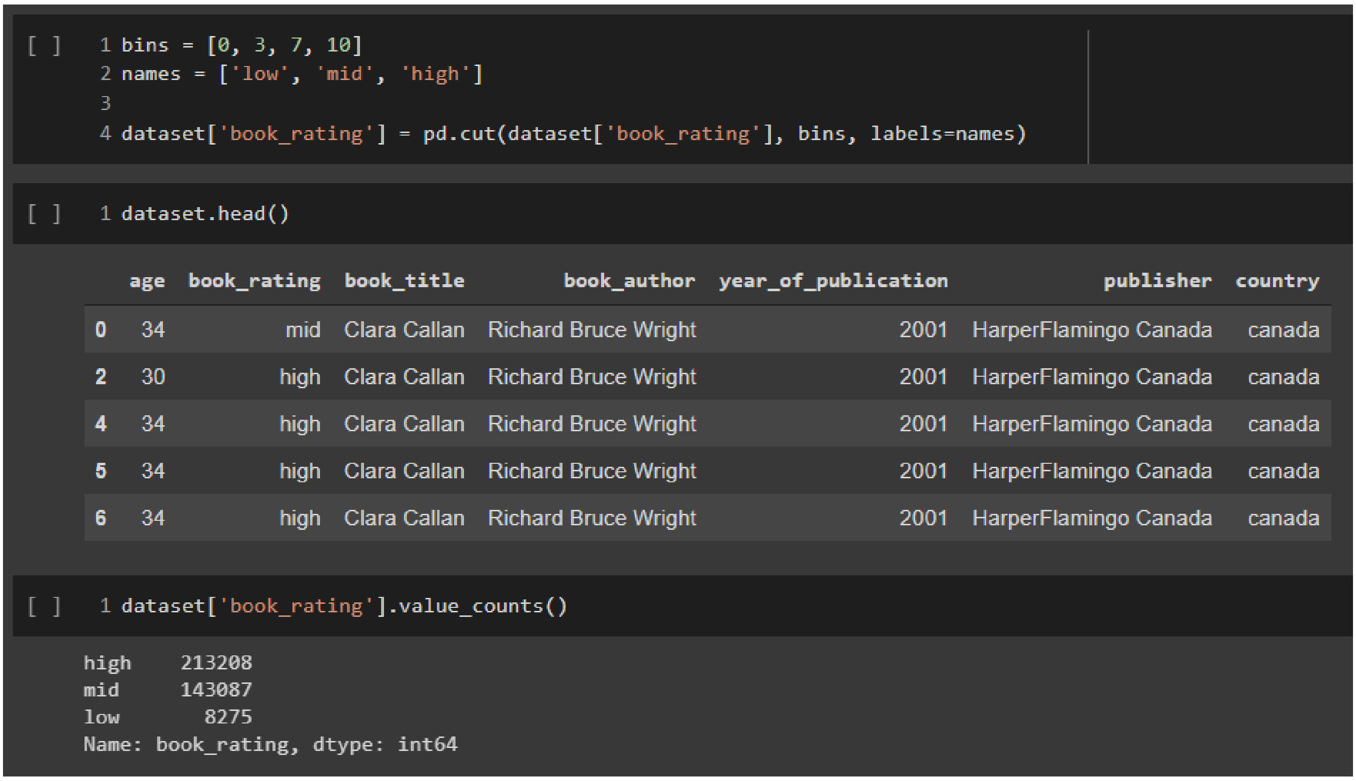
```
[ ]  dataset.head()
```

	age	book_rating	book_title	book_author	year_of_publication	publisher	country
0	34	5	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	canada
2	30	8	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	canada
4	34	9	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	canada
5	34	8	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	canada
6	34	9	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	canada

The target column is `book_rating`, now we have 10 classes for `book_rating`, but to make the task a little simpler we will be labelling the `book_rating` into 3 classes, namely `[low, mid, high]`.

The stats for them are,

high	mid	low
58.482%	39.248%	2.269%



```
bins = [0, 3, 7, 10]
names = ['low', 'mid', 'high']

dataset['book_rating'] = pd.cut(dataset['book_rating'], bins, labels=names)
```

```
dataset.head()
```

	age	book_rating	book_title	book_author	year_of_publication	publisher	country
0	34	mid	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	canada
2	30	high	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	canada
4	34	high	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	canada
5	34	high	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	canada
6	34	high	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	canada

```
dataset['book_rating'].value_counts()
```

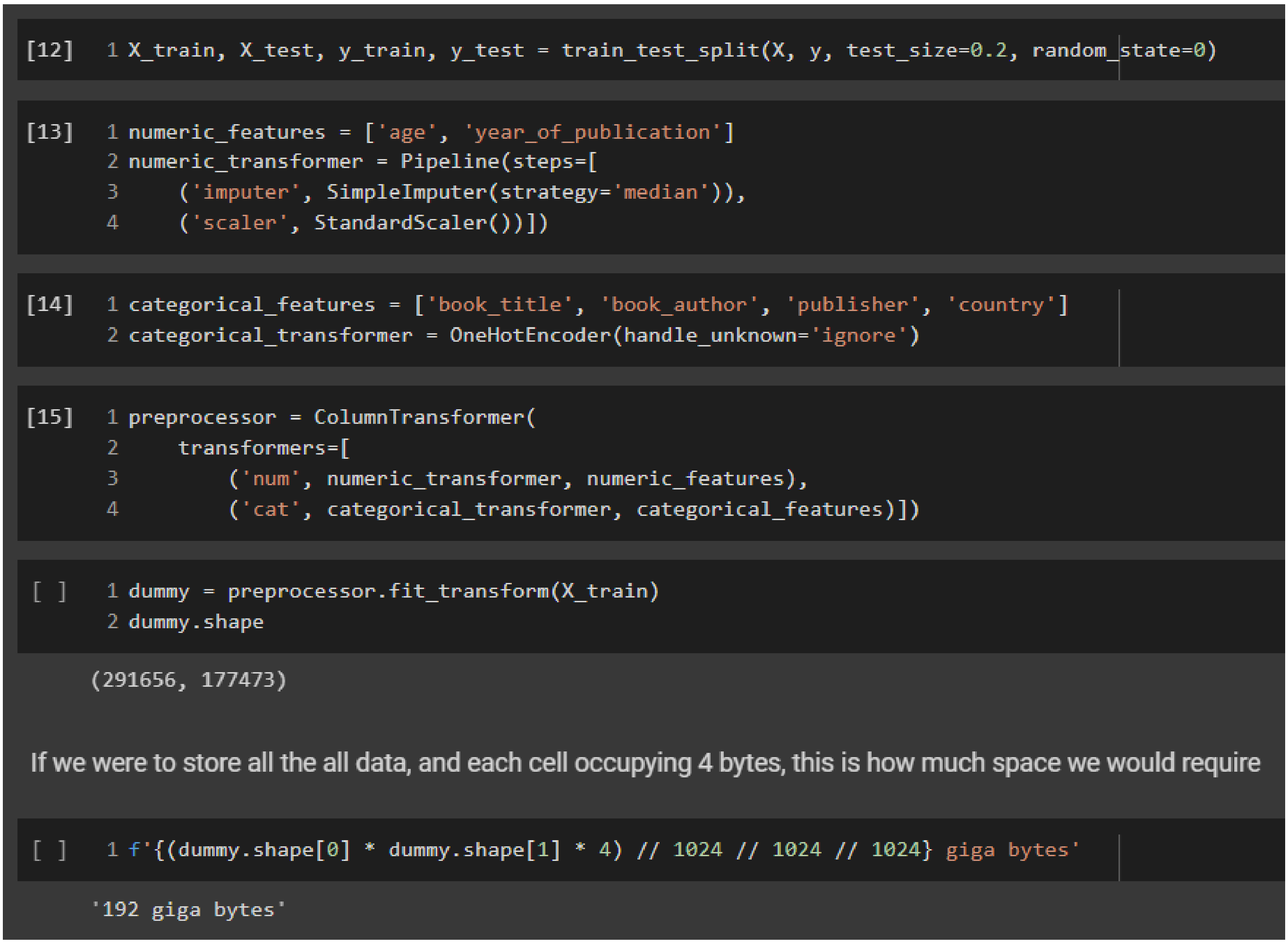
```
high    213208
mid     143087
low       8275
Name: book_rating, dtype: int64
```

To perform any kind of learning on this dataset we would have to convert the categorical values into numerical, to do this there are two options

- OneHotEncoder
- LabelEncoder

Label Encoder is used for those categorical values which have a “ordering” associated with them, for example the temperature can be cold, warm or hot, we can assign the values 0, 1, 2 to them, that makes sense right? Yes! But now let’s say the categorical values are Satyajit, Mohan, Ram how can a number be assigned to them? We cannot assign 0, 1, 2 to them, because we don’t how to order them, to deal with this problem we use OneHotEncoder, which simply assigns them `[1, 0, 0]`, `[0, 1, 0]` and `[0, 0, 1]` which are basically indicating presence, of `[Satyajit, Mohan, Ram]` and there is no ordering involved here.

But looking at our dataset, we have a lot of data, and the since we are considering the book name, author, publisher all as categorical value, they will have quite a lot of categories, which means that when we do one hot encoding, there will be a lot of columns generating, which will be very sparse ofcourse, but this means that we will be restricted by memory consumption.



```
[12]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
[13]  numeric_features = ['age', 'year_of_publication']
      numeric_transformer = Pipeline(steps=[
          ('imputer', SimpleImputer(strategy='median')),
          ('scaler', StandardScaler())])
```

```
[14]  categorical_features = ['book_title', 'book_author', 'publisher', 'country']
      categorical_transformer = OneHotEncoder(handle_unknown='ignore')
```

```
[15]  preprocessor = ColumnTransformer(
          transformers=[
              ('num', numeric_transformer, numeric_features),
              ('cat', categorical_transformer, categorical_features)])
```

```
[ ]   dummy = preprocessor.fit_transform(X_train)
      dummy.shape

(291656, 177473)
```

If we were to store all the all data, and each cell occupying 4 bytes, this is how much space we would require

```
[ ]   f'{(dummy.shape[0] * dummy.shape[1] * 4) // 1024 // 1024 // 1024} giga bytes'

'192 giga bytes'
```

From the above illustration, we can see if we try to save this data, we would approximately require 192 GB, and that is just the training set. (I did try it on a 128GB RAM machine, and I got out-of-memory error). We'll later discuss some of the solutions to this.

1.2 Supervised Learning

Supervised Learning was performed on the cleaned dataset, with OneHotEncoding and StandardScaler with the following algorithms

- Decision Tree
- SVC (Support Vector Classification)
- Passive Aggressive
- Ridge Classifier
- AdaBoost
- LinearSVC
- MLP (Multi Later Perceptron)
- RandomForest

Refer to the notebook attached at the end for more information of accuracy, confusion matrix, f1 score, etc., here we will compare the accuracies and time taken to train for all the models.

Algorithm	**F1 Score**	**Balanced Accuracy**	**Avg Precision**	**Avg Recall**	**Time Taken (s)**
DecisionTree	0.60	0.36	0.58	0.60	146
SVC	NULL	NULL	NULL	NULL	>1000
PassiveAggresive	0.55	0.37	0.55	0.55	**8**
Ridge	0.58	0.37	0.56	0.59	22
AdaBoost	0.59	0.34	0.57	0.59	50
LinearSVC	0.58	0.37	0.56	0.59	200
MLP	**0.61**	**0.38**	**0.60**	**0.61**	630
RandomForest	0.58	0.33	0.60	0.59	537

All of the above models were trained by using Imputation, and Standard Scaler, but we can see and vary the scaler to see how it affects the accuracy, to do that we consider the following scalers, and since Passive Aggressive algorithm was the faster, we will stick with that,

- StandardScaler
- MinMaxScaler
- RobustScaler
- MaxAbsScaler
- PowerTransformer
- QuantileTransfomer
- Normalizer
- UnScaled (Original Data)

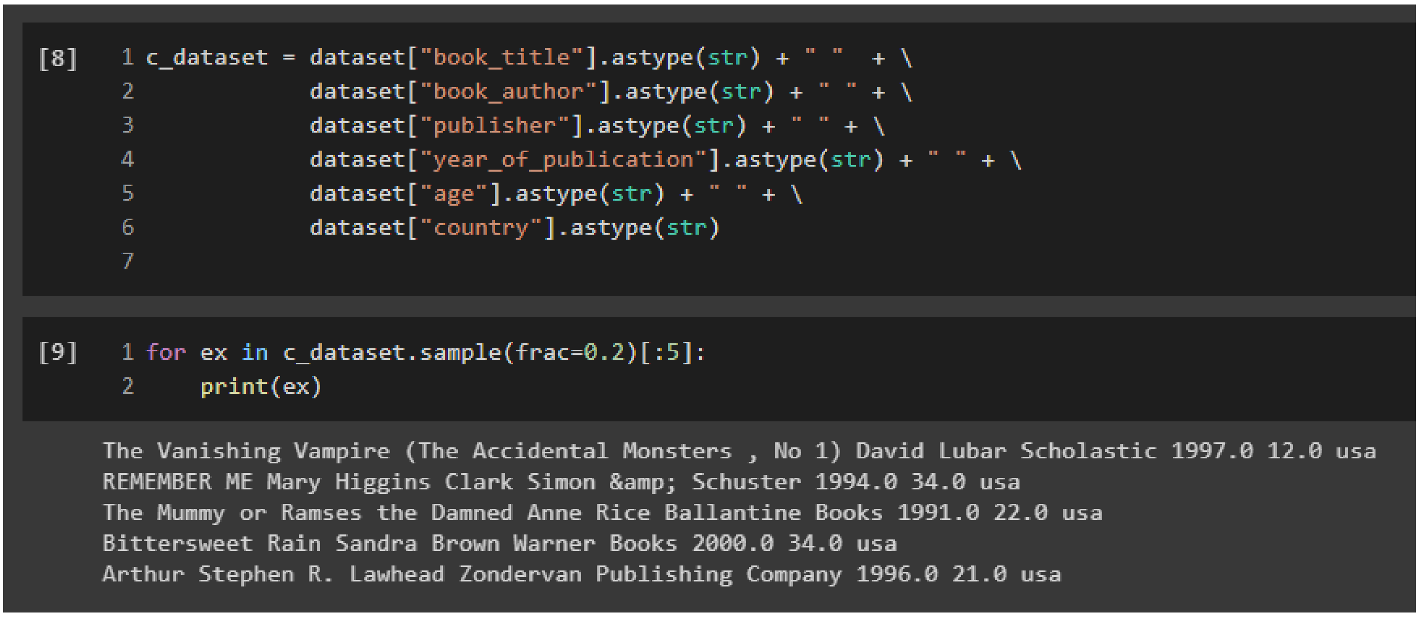
Scaler	**F1 Score**	**Balanced Accuracy**	**Avg Precision**	**Avg Recall**
StandardScaler	0.561	**0.378**	0.55	0.56
MinMaxScaler	0.541	0.369	0.55	0.54
MaxAbsScaler	**0.588**	0.359	**0.55**	**0.59**
RobustScaler	0.576	0.366	0.54	0.58
PowerTransform (Yeo-Johnson)	0.525	0.374	0.55	0.53
QuantileTransform	0.519	0.371	0.54	0.52
Normalizer (L2)	0.570	0.366	0.54	0.57
UnScaled	0.583	0.364	0.55	0.58

1.3 Unsupervised Learning

In the case of unsupervised algorithms, most of them crashed on the original data, since the data was very sparse, and a lot of memory was required.

Learning from the problems faced in supervised learning, one solution to deal with text data is to perform NLP (Natural Language Processing) on the text itself, i.e. we can use pretrained word embeddings to encode the book names, authors, publishers, country into a dense matrix, which can be easily done by using spacy language models.

All of the columns in the dataset was concatenated to form a single sentence, we will later encode this sentence and try to apply clustering algorithm to predict rating.



```
c_dataset = dataset["book_title"].astype(str) + " "  + \
            dataset["book_author"].astype(str) + " " + \
            dataset["publisher"].astype(str) + " " + \
            dataset["year_of_publication"].astype(str) + " " + \
            dataset["age"].astype(str) + " " + \
            dataset["country"].astype(str)

```

```
for ex in c_dataset.sample(frac=0.2)[:5]:
    print(ex)
```

```
The Vanishing Vampire (The Accidental Monsters , No 1) David Lubar Scholastic 1997.0 12.0 usa
REMEMBER ME Mary Higgins Clark Simon &amp; Schuster 1994.0 34.0 usa
The Mummy or Ramses the Damned Anne Rice Ballantine Books 1991.0 22.0 usa
Bittersweet Rain Sandra Brown Warner Books 2000.0 34.0 usa
Arthur Stephen R. Lawhead Zondervan Publishing Company 1996.0 21.0 usa
```

Also learning from the fact that I don’t have enough time to fit a model on this large (300,000) data, we can simplify evaluation of algorithms by sampling a fraction of the original dataset. Even on a really good machine (i7-5930K, 128GB RAM), the model takes a lot of time to fit.

The following algorithms were used,

- KMedoids Clustering

- Agglomerative Clustering
- DBSCAN
- KMeans
- HDBSCAN
- MiniBatchKMeans

Algorithm	**F1 Score**	**Balanced Accuracy**	**Avg Precision**	**Avg Recall**	**Time Taken (s)**
KMedoids	0.30	0.31	0.50	0.31	223
Agglomerative	0.27	0.30	0.48	0.27	119
DBSCAN	**0.58**	**0.33**	0.42	**0.58**	**116**
KMeans	0.31	0.33	0.50	0.32	222
HDBSCAN	**0.58**	**0.33**	**0.54**	**0.58**	121
MiniBatch KMeans	0.33	0.32	0.49	0.34	217

To see the classification report, and the confusion matrix, see appendix.

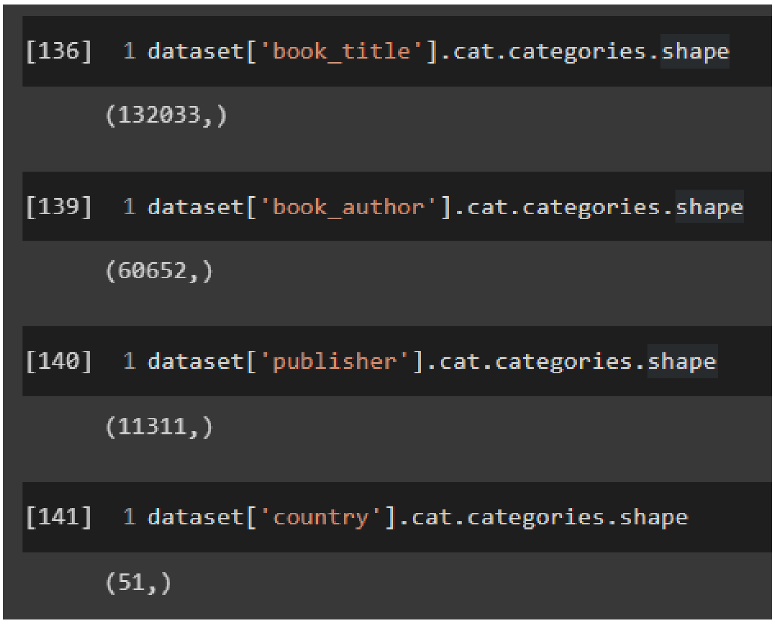
1.4 Comparative Analysis and Conclusion

If we were to compare the supervised and unsupervised models, they have pretty much comparable accuracies (~58-61%), but one thing that cannot be compared is the execution time, since we used clustering algorithms, they require heavy memory usage, and a lot of processing time.

The highest accuracy in Supervised Learning was achieved by MLP (Multi-Layer Perceptron) of 61%, and in Unsupervised Learning, highest was achieved by HDBSCAN of 58%. And in terms of Scaler, it was observed that MaxAbsScaler gave the highest accuracy compared to other scalers.

Something even more important to address is why the accuracy is so low, the `"high"` class of `book_rating` has 58% total records, and this means that if we were to classify any input as `"high"` we have a probability of `0.58` to be correct, this is exactly like a `OneR` learner, but the reason why we are observing such bad accuracy is because there isn't much correlation between the attributes and the target value, this dataset was never meant to do something like this, it was more towards recommending books to a new user with a given past book purchase history, predicting rating based on country, book title, publisher just does not make sense, just based on the book title, how could a book be rated better? One thing that could make sense is the author, there are certain authors that have a good reputation and they have a higher chance of getting a good rating. But in this assignment, we have considered all the attributes, which could have been a mistake, one lesson learnt is that, grabbing all the available data can have negative effects to the accuracy.

But there is another issue, if we see the number of categorical values for each of the categorical columns,



```
[136]  dataset['book_title'].cat.categories.shape

    (132033,)

[139]  dataset['book_author'].cat.categories.shape

    (60652,)

[140]  dataset['publisher'].cat.categories.shape

    (11311,)

[141]  dataset['country'].cat.categories.shape

    (51,)
```

Something we can clearly observe is that, that is a lot of categories, for specially the book_title, this means that if we use OneHotEncoding, we will get those many columns added to the dataset. This does make it really difficult to train a model, scipy has a csr representation (Compressed Sparse Row Matrix) which does help to a lot of extend, but still this is the biggest issue that has to be addressed.

We can take inspiration from text classification models and NLP (Natural Language Processing), what is done for it, is that each of the words in dictionary is a sparse matrix, and then a model is used to convert this sparse matrix into a dense matrix, for example lets say the word `"the"` is represented as `[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]`, if we use a word embedding model, such as from spacy, we would get something like `[0.45673, 0.67395, 0.98729]` which is a much better representation as before, this matrix can then be used to train. This inspiration was tried in the clustering algorithms in 1.3, and it quite did not produce good results. But I would blame the dataset for that. On top of giving the dense matrix, we can further reduce the dimensionality by using SVD (Singular Value Decomposition).

I had a hunch that if I were to use GPT-2 or BERT, the model would have been really great (although it might overfit, but still), but due to time constrains, is wasn't possible, but it would be a good exploration domain.

1.4.1 Conclusion

Never rely on SciKit Learn, the algorithms are really slow (except K-Mean) and are not quite optimized enough, instead `C++` backend algorithms give much better speed.

Also, when a classification model has to be made, in which there is a lot of attributes in text in them, it would be better, to just convert those specific attributes to a dense matrix using a pretrained word embedding model, or an even better improvement can be to use `n-gram` models.

To conclude, we were not really successful in creating a good classification model for the given dataset, but we were definitely able to compare different supervised models, unsupervised models and numeric scaling techniques, overall, the outcome of this course was covered.

2 Appendix

BookReview-Classification-Clean

January 20, 2021

[ ]:
```
import gdown

url = 'https://drive.google.com/uc?id=1PL13wgXLfXcsrkKNuVIaNJdGXrIqv2mv'
output = 'book_crossing.cleaned.csv'
gdown.download(url, output, quiet=False)
```

```
Downloading...
From: https://drive.google.com/uc?id=1PL13wgXLfXcsrkKNuVIaNJdGXrIqv2mv
To: /content/book_crossing.cleaned.csv
44.9MB [00:01, 40.7MB/s]
```

[ ]:
```
'book_crossing.cleaned.csv'
```

[ ]:
```
%matplotlib inline

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

sns.set()
palette = sns.color_palette("icefire")

plt.style.use('ggplot')

sns.set_context("talk")
```

1 BookCrossing - Cleaning

[ ]:
```
dataset = pd.read_csv('book_crossing.cleaned.csv')
```

[ ]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 383849 entries, 0 to 383848
```

```
Data columns (total 11 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   user_id              383849 non-null  int64
 1   age                  383849 non-null  int64
 2   isbn                 383849 non-null  object
 3   book_rating          383849 non-null  int64
 4   book_title           383849 non-null  object
 5   book_author          383849 non-null  object
 6   year_of_publication  383849 non-null  int64
 7   publisher            383849 non-null  object
 8   city                 375223 non-null  object
 9   state                371248 non-null  object
 10  country              366406 non-null  object
dtypes: int64(4), object(7)
memory usage: 32.2+ MB
```

We won't be considering city, state, because they don't really tell a lot of the rating of a book, but also most of the users (~70%) are from usa (which may not contribute a lot to accuracy of classification, but we'll consider it), and the location is realated to the user, and not the book directly, we'll also be dropping isbn, user_id, since they dont contribute to classification of rating

```
[ ]: dataset = dataset.drop(['user_id', 'isbn', 'city', 'state'], axis=1)
```

```
[ ]: f'Dataset Shape : {dataset.shape}'
```

```
[ ]: 'Dataset Shape : (383849, 7)'
```

```
[ ]: dataset.dropna(inplace=True)
```

```
[ ]: f'Dataset Shape after dropping NA: {dataset.shape}'
```

```
[ ]: 'Dataset Shape after dropping NA: (366406, 7)'
```

```
[ ]: dataset.head()
```

```
[ ]:    age  book_rating  ...              publisher country
     0   34            5  ...  HarperFlamingo Canada  canada
     2   30            8  ...  HarperFlamingo Canada  canada
     4   34            9  ...  HarperFlamingo Canada  canada
     5   34            8  ...  HarperFlamingo Canada  canada
     6   34            9  ...  HarperFlamingo Canada  canada

     [5 rows x 7 columns]
```

```
[ ]: dataset.describe().T
```

```
[ ]:                         count       mean        std  ...    50%    75%
     max
     age                 366406.0  35.860998  10.448608  ...   34.0   40.0
     100.0
     book_rating         366406.0   7.635975   1.836354  ...    8.0    9.0
     10.0
```

```
year_of_publication  366406.0  1995.670314   7.397156  ...  1997.0  2001.0
2006.0

[3 rows x 8 columns]
```

We'll remove the rows which have a country which has value count <= 50

[ ]:
```python
dataset = dataset.groupby('country').filter(lambda x: len(x) > 50)
```

[ ]:
```python
dataset.describe().T
```

[ ]:
```
                        count         mean        std  ...     50%     75%
max
age                  364570.0    35.867227  10.447887  ...    34.0    40.0
100.0
book_rating          364570.0     7.636709   1.835857  ...     8.0     9.0
10.0
year_of_publication  364570.0  1995.667164   7.400552  ...  1997.0  2001.0
2006.0

[3 rows x 8 columns]
```

[ ]:
```python
f'Dataset Shape : {dataset.shape}'
```

[ ]:
```
'Dataset Shape : (364570, 7)'
```

[ ]:
```python
f'Column Names: {dataset.columns.to_list()}'
```

[ ]:
```
"Column Names: ['age', 'book_rating', 'book_title', 'book_author',
'year_of_publication', 'publisher', 'country']"
```

[ ]:
```python
dataset['book_rating'].value_counts()
```

[ ]:
```
8     87090
10    68038
7     63036
9     58080
5     42988
6     29943
4      7120
3      4746
2      2198
1      1331
Name: book_rating, dtype: int64
```

We'll now convert the rating into classification categories

[ ]:
```python
bins = [0, 3, 7, 10]
names = ['low', 'mid', 'high']

dataset['book_rating'] = pd.cut(dataset['book_rating'], bins, labels=names)
```

[ ]:
```python
dataset.head()
```

[ ]:
```
   age book_rating  ...              publisher country
0   34         mid  ...  HarperFlamingo Canada  canada
2   30        high  ...  HarperFlamingo Canada  canada
4   34        high  ...  HarperFlamingo Canada  canada
5   34        high  ...  HarperFlamingo Canada  canada
6   34        high  ...  HarperFlamingo Canada  canada

[5 rows x 7 columns]
```

[ ]:
```
dataset['book_rating'].value_counts()
```

[ ]:
```
high    213208
mid     143087
low       8275
Name: book_rating, dtype: int64
```

[ ]:
```
dataset.to_csv('book_crossing.classification.cleaned.csv', index=False)
```

[ ]:
```

```

BookReview-Classification-Supervised

January 20, 2021

[ ]:
```
! pip install --upgrade scikit-learn
```

1 Book Crossing - Classification

[1]:
```
%matplotlib inline

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

sns.set()
palette = sns.color_palette("icefire")

plt.style.use('ggplot')

sns.set_context("talk")
```

[2]:
```
dataset = pd.read_csv('book_crossing.classification.cleaned.csv')
```

[3]:
```
dataset['age'] = dataset['age'].astype(np.float64)
dataset['book_rating'] = dataset['book_rating'].astype('category')
dataset['book_title'] = dataset['book_title'].astype('category')
dataset['book_author'] = dataset['book_author'].astype('category')
dataset['year_of_publication'] = dataset['year_of_publication'].astype(np.
 ↪float64)
dataset['publisher'] = dataset['publisher'].astype('category')
dataset['country'] = dataset['country'].astype('category')
```

[4]:
```
dataset.head()
```

[4]:
```
    age book_rating  ...                publisher country
0  34.0         mid  ...  HarperFlamingo Canada  canada
1  30.0        high  ...  HarperFlamingo Canada  canada
2  34.0        high  ...  HarperFlamingo Canada  canada
3  34.0        high  ...  HarperFlamingo Canada  canada
```

```
4  34.0         high  ...  HarperFlamingo Canada   canada

[5 rows x 7 columns]
```

[5]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 364570 entries, 0 to 364569
Data columns (total 7 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   age                  364570 non-null  float64
 1   book_rating          364570 non-null  category
 2   book_title           364570 non-null  category
 3   book_author          364570 non-null  category
 4   year_of_publication  364570 non-null  float64
 5   publisher            364570 non-null  category
 6   country              364570 non-null  category
dtypes: category(5), float64(2)
memory usage: 19.1 MB
```

[6]:

```
from sklearn import set_config
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder,␣
 ↪MinMaxScaler
from sklearn.svm import SVC, LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, plot_confusion_matrix,␣
 ↪confusion_matrix, accuracy_score, balanced_accuracy_score
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.linear_model import RidgeClassifier, PassiveAggressiveClassifier

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import minmax_scale
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import QuantileTransformer
from sklearn.preprocessing import PowerTransformer

set_config(display='diagram')
```

[7]:

```
X, y = dataset.drop('book_rating', axis=1), dataset['book_rating']
```

[8]:

```
target_names = ['low', 'mid', 'high']
```

[9]:

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 364570 entries, 0 to 364569
Data columns (total 6 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   age                  364570 non-null  float64
 1   book_title           364570 non-null  category
 2   book_author          364570 non-null  category
 3   year_of_publication  364570 non-null  float64
 4   publisher            364570 non-null  category
 5   country              364570 non-null  category
dtypes: category(4), float64(2)
memory usage: 18.8 MB
```

[10]:

```
X.head()
```

[10]:

```
    age    book_title  ...              publisher  country
0  34.0  Clara Callan  ...  HarperFlamingo Canada   canada
1  30.0  Clara Callan  ...  HarperFlamingo Canada   canada
2  34.0  Clara Callan  ...  HarperFlamingo Canada   canada
3  34.0  Clara Callan  ...  HarperFlamingo Canada   canada
4  34.0  Clara Callan  ...  HarperFlamingo Canada   canada

[5 rows x 6 columns]
```

[11]:

```
y.head()
```

[11]:

```
0     mid
1    high
2    high
3    high
4    high
Name: book_rating, dtype: category
Categories (3, object): ['high', 'low', 'mid']
```

[12]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)
```

[13]:

```
numeric_features = ['age', 'year_of_publication']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())])
```

[14]:

```
categorical_features = ['book_title', 'book_author', 'publisher', 'country']
categorical_transformer = OneHotEncoder(handle_unknown='ignore')
```

[15]:
```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])
```

[24]:
```
dummy = preprocessor.fit_transform(X_train)
dummy.shape
```

[24]:
```
(291656, 177473)
```

If we were to store all the all data, and each cell occupying 4 bytes, this is how much space we would require

[30]:
```
f'{(dummy.shape[0] * dummy.shape[1] * 4) // 1024 // 1024 // 1024} giga bytes'
```

[30]:
```
'192 giga bytes'
```

1.1 Supervised Models

- DecisionTreeClassifier
- SVC
- LinearSVC
- KNeighborsClassifier
- PassiveAggressiveClassifier
- AdaBoostClassifier
- GaussianProcessClassifier

[16]:
```
from time import time

def fit_model(algorithm, data, preprocessor):

    t1 = time()

    X_train, X_test, y_train, y_test = data

    clf = Pipeline(steps=[('preprocessor', preprocessor),
                          ('classifier', algorithm)])

    print(f'\nStarted Training {algorithm.__class__.__name__} on X_train:␣
 ↪{X_train.shape} y_train: {y_train.shape}')

    # train the model
    clf.fit(X_train, y_train)

    print(f"\nEvaluating model on X_test: {X_test.shape} y_test: {y_test.
 ↪shape}")

    # test the model
    y_true = y_test.copy()
    y_pred = clf.predict(X_test)
    # get the classification report
    print(f"\nClassification Report for {algorithm.__class__.__name__}")
    print(classification_report(y_true, y_pred, target_names=target_names,␣
 ↪labels=target_names))

    acc_score = accuracy_score(y_true, y_pred)
    bal_score = balanced_accuracy_score(y_true, y_pred)

    print(f"\nAccuracy Score: {acc_score}")
    print(f"Balanced Accuracy Score: {bal_score}")

    print()
    # show the confusion matrix
    cmmat_table = pd.DataFrame({'y_true': y_true, 'y_pred': y_pred})
    conmat = pd.crosstab(cmmat_table.y_true, cmmat_table.y_pred,␣
 ↪rownames=['Actual'], colnames=['Predicted'], margins=True, normalize='all')
    ax = plt.axes()
    sns.set(rc={'figure.figsize':(9, 7)})
    sns.heatmap(conmat, annot=True, ax=ax)
    ax.set_title(f'{algorithm.__class__.__name__}')
    plt.show()
    print()

    t2 = time()

    print(f'Trained {algorithm.__class__.__name__} in {(t2 - t1)}s')

    return clf
```

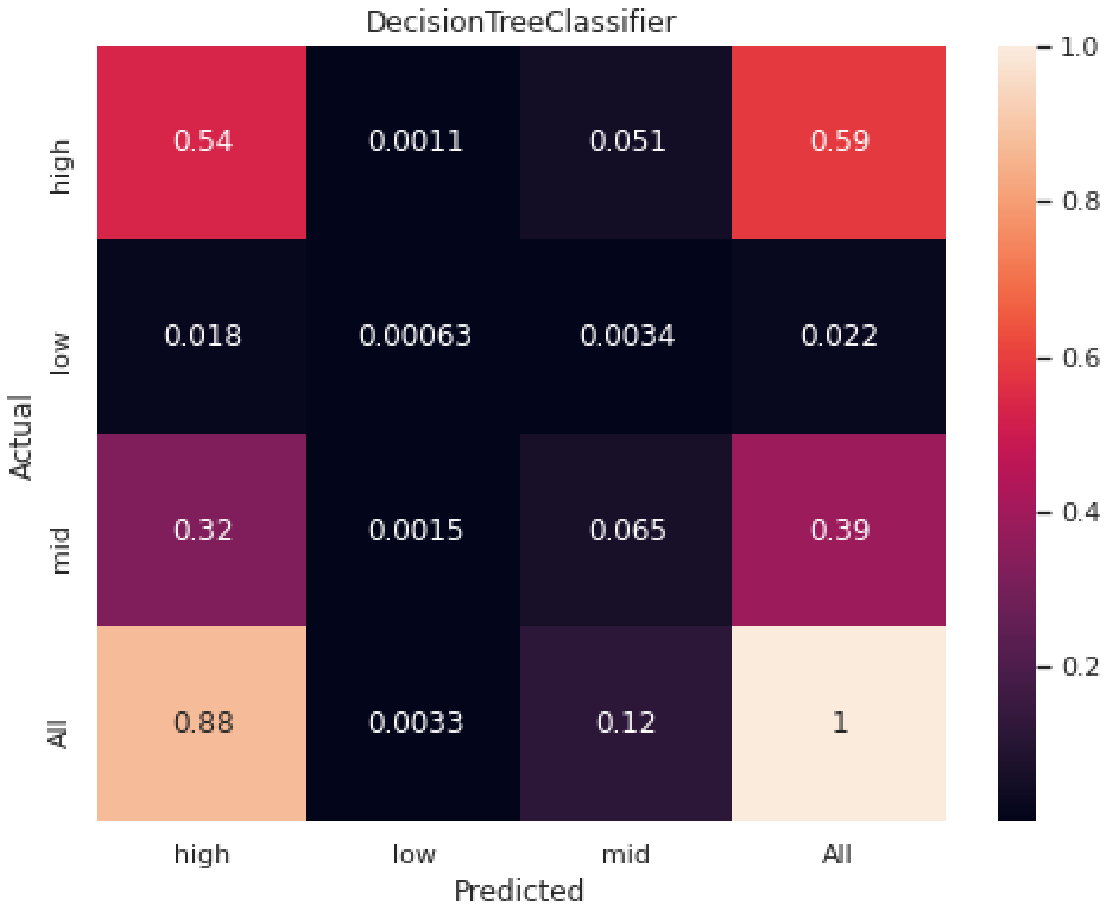
1.2 DecisionTree Classifier

[33]:
```
dtc = DecisionTreeClassifier(max_depth=100)
```

[34]:
```
clf = fit_model(algorithm=dtc, data=(X_train, X_test, y_train, y_test),␣
 ↪preprocessor=preprocessor)
```

```
Started Training DecisionTreeClassifier on X_train: (291656, 6) y_train:
(291656,)

Evaluating model on X_test: (72914, 6) y_test: (72914,)

Classification Report for DecisionTreeClassifier
              precision    recall  f1-score   support
```

```
         low       0.19      0.03      0.05      1603
         mid       0.55      0.17      0.25     28483
        high       0.61      0.91      0.73     42828

    accuracy                           0.60     72914
   macro avg       0.45      0.37      0.35     72914
weighted avg       0.58      0.60      0.53     72914


Accuracy Score: 0.6010368379186439
Balanced Accuracy Score: 0.36884514138535285
```



```
Trained DecisionTreeClassifier in 146.20573687553406s
```

[35]:
```
clf
```

[35]:

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['age',
                                                   'year_of_publication']),
                                                 ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['book_title', 'book_author',
                                                   'publisher', 'country'])])),
                ('classifier', DecisionTreeClassifier(max_depth=100))])
```

1.3 SVC (Support Vector Classification)

[31]:

```
svc = SVC(gamma='auto', C=1.0, kernel='rbf')
```

[ ]:

```
clf = fit_model(algorithm=svc, data=(X_train, X_test, y_train, y_test),␣
 ↪preprocessor=preprocessor)
```

```
Started Training SVC on X_train: (291656, 6) y_train: (291656,)
```

[ ]:

```
clf
```

1.4 Passive Aggressive Classifier

[17]:

```
pac = PassiveAggressiveClassifier(early_stopping=True, verbose=1, n_jobs=-1)
```

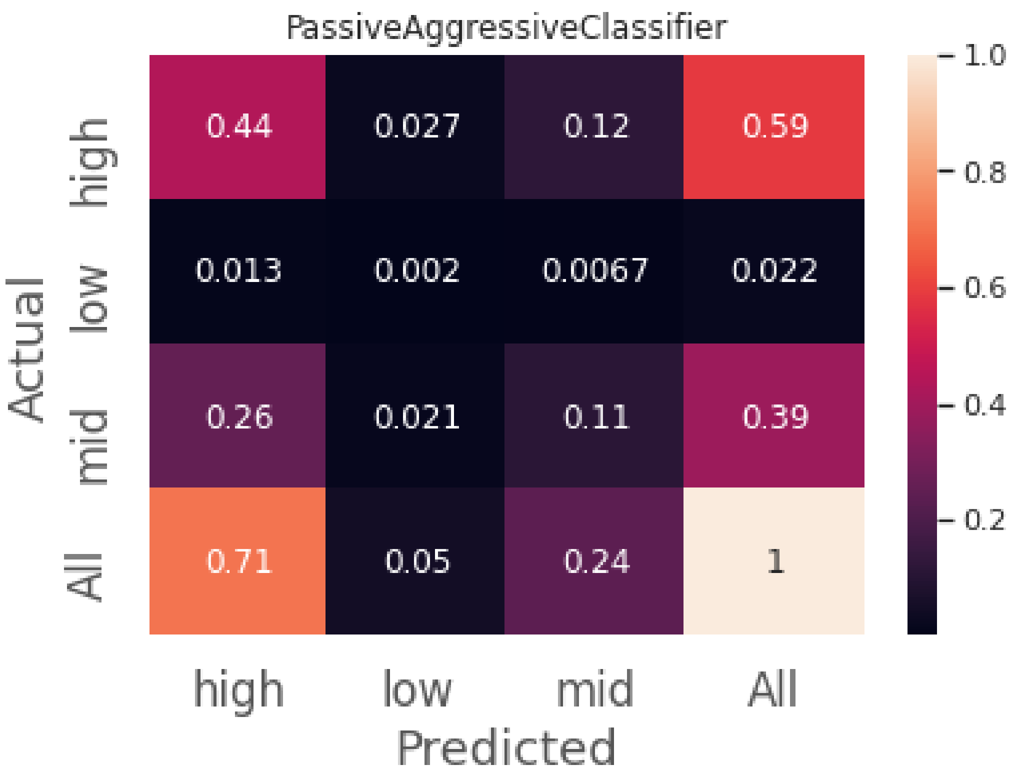
[20]:

```
clf = fit_model(algorithm=pac, data=(X_train, X_test, y_train, y_test),␣
 ↪preprocessor=preprocessor)
```

```
Started Training PassiveAggressiveClassifier on X_train: (291656, 6) y_train:
(291656,)

[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent workers.

-- Epoch 1-- Epoch 1

Norm: 46.21, NNZs: 53157, Bias: -0.504273, T: 262490, Avg. loss: 0.105049
Total training time: 0.11 seconds.
-- Epoch 2
Norm: 119.01, NNZs: 149844, Bias: 0.084396, T: 262490, Avg. loss: 0.905595
Total training time: 0.18 seconds.
-- Epoch 2
```

```
Norm: 66.08, NNZs: 70363, Bias: -0.521754, T: 524980, Avg. loss: 0.086252
Total training time: 0.29 seconds.
-- Epoch 3
Norm: 164.90, NNZs: 161910, Bias: 0.072078, T: 524980, Avg. loss: 0.789480
Total training time: 0.36 seconds.
-- Epoch 3
Norm: 79.79, NNZs: 79611, Bias: -0.583871, T: 787470, Avg. loss: 0.077301
Total training time: 0.46 seconds.
-- Epoch 4
Norm: 198.89, NNZs: 164254, Bias: 0.097328, T: 787470, Avg. loss: 0.731145
Total training time: 0.54 seconds.
-- Epoch 4
Norm: 90.21, NNZs: 85297, Bias: -0.646251, T: 1049960, Avg. loss: 0.072339
Total training time: 0.63 seconds.
-- Epoch 5
Norm: 225.85, NNZs: 164894, Bias: 0.119736, T: 1049960, Avg. loss: 0.696331
Total training time: 0.72 seconds.
-- Epoch 5
Norm: 98.61, NNZs: 89244, Bias: -0.634484, T: 1312450, Avg. loss: 0.070211
Total training time: 0.80 seconds.
-- Epoch 6
Norm: 248.75, NNZs: 165157, Bias: 0.069723, T: 1312450, Avg. loss: 0.674325
Total training time: 0.89 seconds.
-- Epoch 6
Norm: 105.81, NNZs: 92113, Bias: -0.705083, T: 1574940, Avg. loss: 0.067957
Total training time: 0.99 seconds.
Convergence after 6 epochs took 1.04 seconds
-- Epoch 1
Norm: 268.11, NNZs: 165276, Bias: 0.086452, T: 1574940, Avg. loss: 0.659386
Total training time: 1.10 seconds.
Convergence after 6 epochs took 1.16 seconds
Norm: 117.95, NNZs: 148735, Bias: -0.110863, T: 262490, Avg. loss: 0.897951
Total training time: 0.15 seconds.
-- Epoch 2
Norm: 163.59, NNZs: 161578, Bias: -0.108644, T: 524980, Avg. loss: 0.784290
Total training time: 0.30 seconds.
-- Epoch 3
Norm: 197.51, NNZs: 164105, Bias: -0.107356, T: 787470, Avg. loss: 0.725686
Total training time: 0.45 seconds.
-- Epoch 4
Norm: 224.64, NNZs: 164845, Bias: -0.101451, T: 1049960, Avg. loss: 0.692328
Total training time: 0.59 seconds.
-- Epoch 5
Norm: 247.11, NNZs: 165150, Bias: -0.105241, T: 1312450, Avg. loss: 0.670131
Total training time: 0.73 seconds.
-- Epoch 6
Norm: 266.48, NNZs: 165274, Bias: -0.121605, T: 1574940, Avg. loss: 0.654409
Total training time: 0.89 seconds.
```

```
-- Epoch 7
Norm: 283.55, NNZs: 165349, Bias: -0.141001, T: 1837430, Avg. loss: 0.642702
Total training time: 1.03 seconds.
-- Epoch 8
Norm: 298.48, NNZs: 165405, Bias: -0.106583, T: 2099920, Avg. loss: 0.634828
Total training time: 1.18 seconds.
-- Epoch 9
Norm: 312.23, NNZs: 165423, Bias: -0.145596, T: 2362410, Avg. loss: 0.626431
Total training time: 1.33 seconds.
Convergence after 9 epochs took 1.38 seconds

[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:    2.5s finished


Evaluating model on X_test: (72914, 6) y_test: (72914,)

Classification Report for PassiveAggressiveClassifier
              precision    recall  f1-score   support

         low       0.04      0.09      0.05      1603
         mid       0.47      0.29      0.36     28483
        high       0.62      0.75      0.68     42828

    accuracy                           0.55     72914
   macro avg       0.38      0.37      0.36     72914
weighted avg       0.55      0.55      0.54     72914


Accuracy Score: 0.5531722302987081
Balanced Accuracy Score: 0.37477427261674867
```



```
Trained PassiveAggressiveClassifier in 8.378920316696167s
```

[21]:
```
clf
```

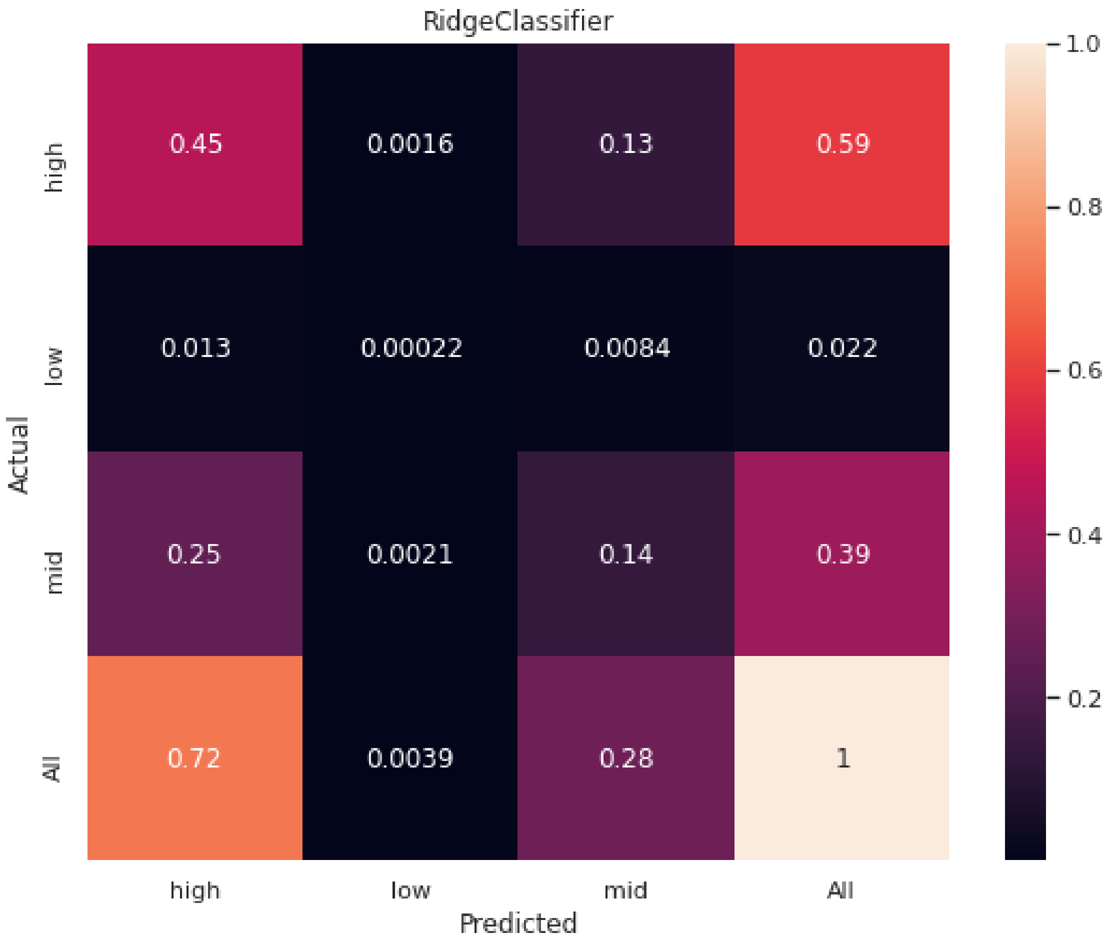
[21]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['age',
                                                   'year_of_publication']),
                                                 ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['book_title', 'book_author',
                                                   'publisher', 'country'])])),
                ('classifier',
                 PassiveAggressiveClassifier(early_stopping=True, n_jobs=-1,
                                             verbose=1))])
```

1.5 Ridge Classifier

[26]:
```
rc = RidgeClassifier(tol=1e-2, solver="auto")
```

[27]:
```
clf = fit_model(algorithm=rc, data=(X_train, X_test, y_train, y_test),␣
 ↪preprocessor=preprocessor)
```

```
Started Training RidgeClassifier on X_train: (291656, 6) y_train: (291656,)

Evaluating model on X_test: (72914, 6) y_test: (72914,)

Classification Report for RidgeClassifier
              precision    recall  f1-score   support

         low       0.06      0.01      0.02      1603
         mid       0.49      0.35      0.41     28483
        high       0.63      0.77      0.69     42828

    accuracy                           0.59     72914
   macro avg       0.39      0.38      0.37     72914
weighted avg       0.56      0.59      0.57     72914


Accuracy Score: 0.5888855363853307
Balanced Accuracy Score: 0.3765274882659582
```



```
Trained RidgeClassifier in 22.259966611862183s
```

[28]:

```
clf
```

[28]:

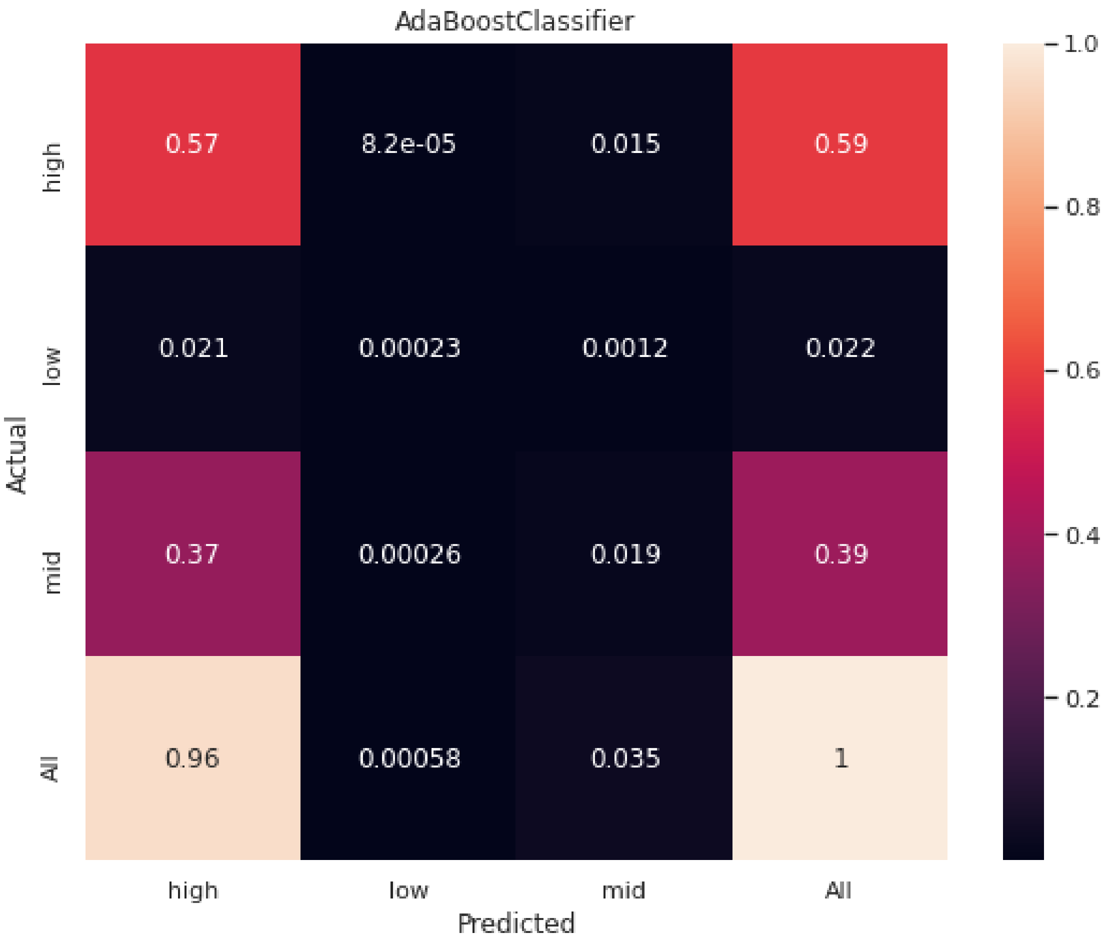
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['age',
                                                   'year_of_publication']),
                                                 ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['book_title', 'book_author',
                                                   'publisher', 'country'])])),
                ('classifier', RidgeClassifier(tol=0.01))])
```

1.6 AdaBoost Classifier

[29]:
```
abc = AdaBoostClassifier()
```

[30]:
```
clf = fit_model(algorithm=abc, data=(X_train, X_test, y_train, y_test),␣
 ↪preprocessor=preprocessor)
```

```
Started Training AdaBoostClassifier on X_train: (291656, 6) y_train: (291656,)

Evaluating model on X_test: (72914, 6) y_test: (72914,)

Classification Report for AdaBoostClassifier
              precision    recall  f1-score   support

         low       0.40      0.01      0.02      1603
         mid       0.55      0.05      0.09     28483
        high       0.59      0.97      0.74     42828

    accuracy                           0.59     72914
   macro avg       0.51      0.34      0.28     72914
weighted avg       0.57      0.59      0.47     72914


Accuracy Score: 0.5919165043750172
Balanced Accuracy Score: 0.34478823998711056
```



```
Trained AdaBoostClassifier in 50.97685408592224s
```

[31]:

```
clf
```

[31]:

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['age',
                                                   'year_of_publication']),
                                                 ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['book_title', 'book_author',
                                                   'publisher', 'country'])])),
                ('classifier', AdaBoostClassifier())])
```

1.7 Linear SVC

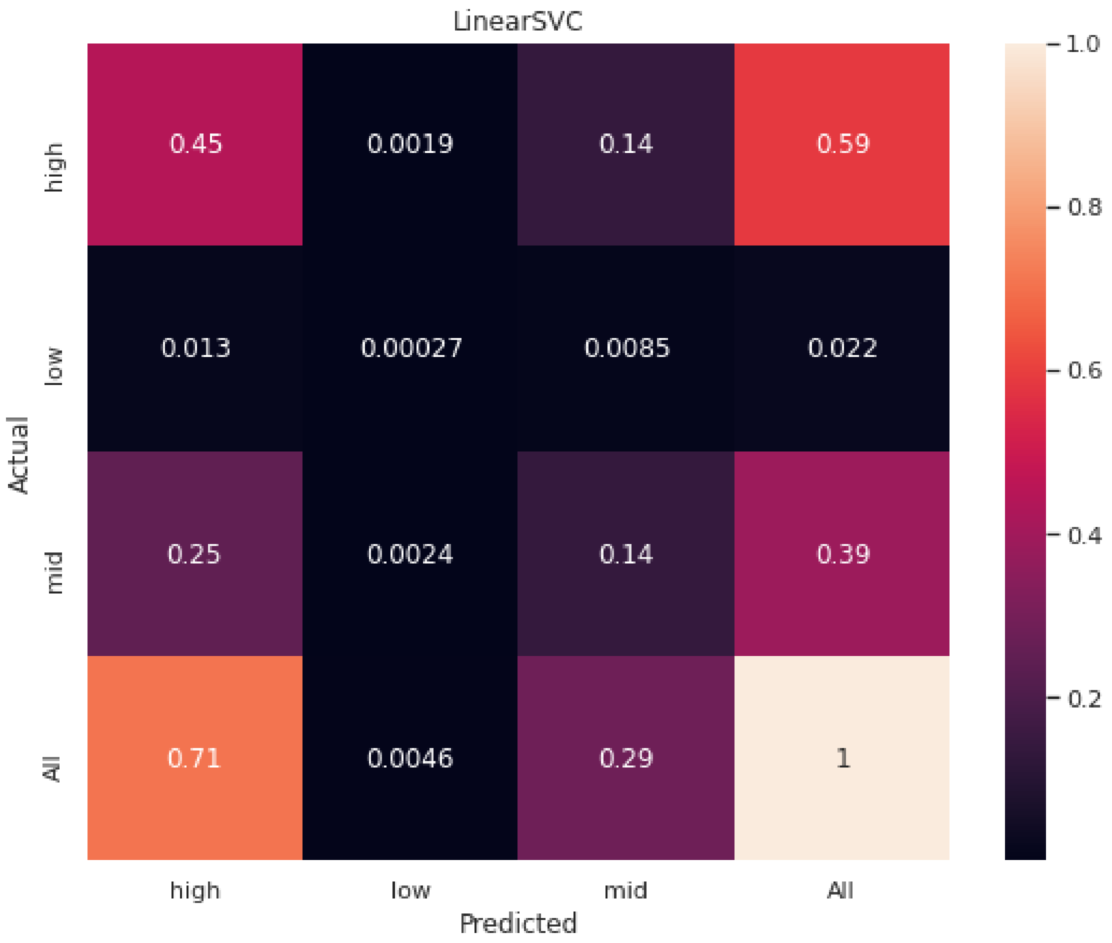
[28]:

```
lsvc = LinearSVC(verbose=1)
```

[29]:

```
clf = fit_model(algorithm=lsvc, data=(X_train, X_test, y_train, y_test),␣
 ↪preprocessor=preprocessor)
```

```
Started Training LinearSVC on X_train: (291656, 6) y_train: (291656,)

/usr/local/lib/python3.6/dist-packages/sklearn/svm/_base.py:986:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)


Evaluating model on X_test: (72914, 6) y_test: (72914,)

Classification Report for LinearSVC
              precision    recall  f1-score   support

         low       0.06      0.01      0.02      1603
         mid       0.49      0.35      0.41     28483
        high       0.63      0.76      0.69     42828

    accuracy                           0.59     72914
   macro avg       0.39      0.38      0.37     72914
weighted avg       0.56      0.59      0.57     72914


Accuracy Score: 0.5862660120141536
Balanced Accuracy Score: 0.3763315404262985
```



```
Trained LinearSVC in 199.0021414756775s
```

[30]:
```
clf
```

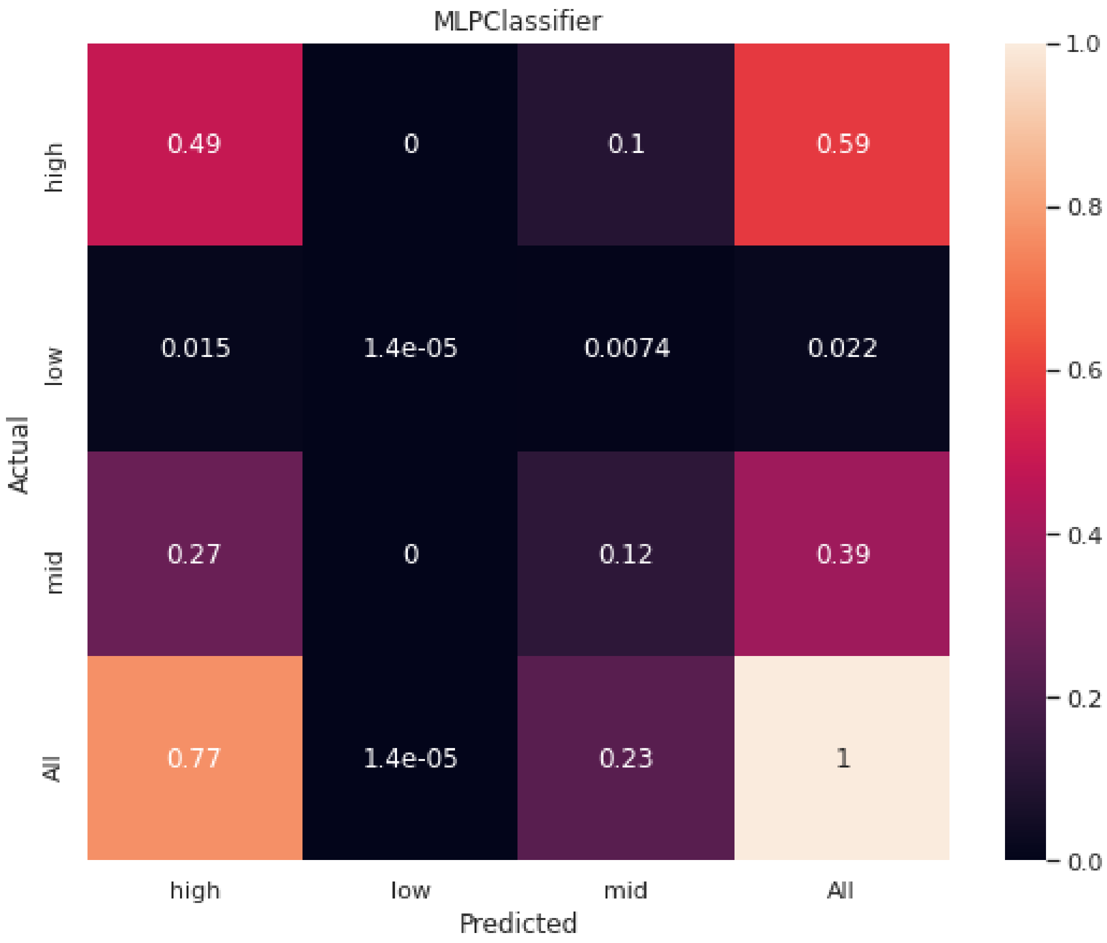
[30]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['age',
                                                   'year_of_publication']),
                                                 ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['book_title', 'book_author',
                                                   'publisher', 'country'])])),
                ('classifier', LinearSVC(verbose=1))])
```

1.8 MLP Classifier

[26]:
```
mlp = MLPClassifier(alpha=0.001, max_iter=1, verbose=1)
```

[27]:
```
clf = fit_model(algorithm=mlp, data=(X_train, X_test, y_train, y_test),␣
 ↪preprocessor=preprocessor)
```

```
Started Training MLPClassifier on X_train: (291656, 6) y_train: (291656,)
Iteration 1, loss = 0.75835927

Evaluating model on X_test: (72914, 6) y_test: (72914,)

/usr/local/lib/python3.6/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:617:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1) reached and the
optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)


Classification Report for MLPClassifier
              precision    recall  f1-score   support

         low       1.00      0.00      0.00      1603
         mid       0.53      0.31      0.39     28483
        high       0.63      0.83      0.72     42828

    accuracy                           0.61     72914
   macro avg       0.72      0.38      0.37     72914
weighted avg       0.60      0.61      0.57     72914


Accuracy Score: 0.6084153934772472
Balanced Accuracy Score: 0.3799475125685053
```



```
Trained MLPClassifier in 630.3555474281311s
```

[19]:
```
clf
```

[19]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['age',
                                                   'year_of_publication']),
                                                 ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['book_title', 'book_author',
                                                   'publisher', 'country'])])),
                ('classifier',
                    MLPClassifier(alpha=0.001, max_iter=2, verbose=1))])
```

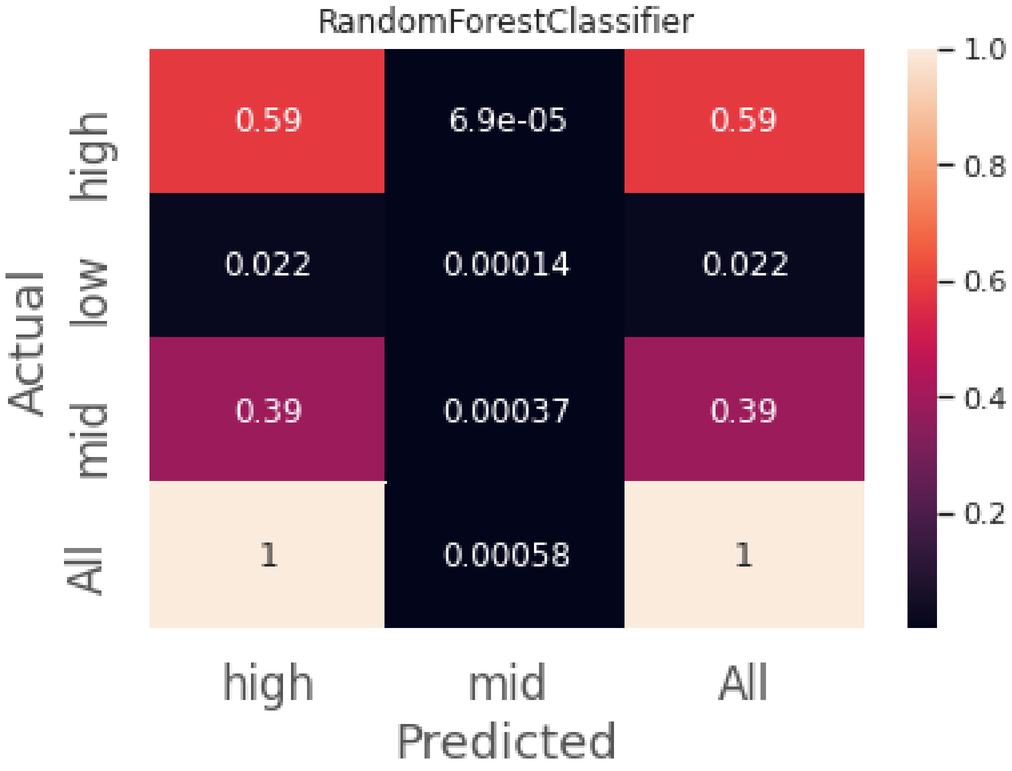
1.9 RandomForest Classifier

[23]:
```
rfc = RandomForestClassifier(max_depth=50, verbose=1, n_jobs=-1)
```

[24]:
```
clf = fit_model(algorithm=rfc, data=(X_train, X_test, y_train, y_test),␣
 ↪preprocessor=preprocessor)
```

```
Started Training RandomForestClassifier on X_train: (291656, 6) y_train:
(291656,)

[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  8.8min finished


Evaluating model on X_test: (72914, 6) y_test: (72914,)

[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.6s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    1.2s finished


Classification Report for RandomForestClassifier

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1245:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1245:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1245:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

              precision    recall  f1-score   support

         low       0.00      0.00      0.00      1603
         mid       0.64      0.00      0.00     28483
        high       0.59      1.00      0.74     42828
```

```
    accuracy                           0.59     72914
   macro avg       0.41      0.33      0.25     72914
weighted avg       0.60      0.59      0.44     72914


Accuracy Score: 0.5876786351043696
Balanced Accuracy Score: 0.33361039593376624
```



```
Trained RandomForestClassifier in 537.0094470977783s
```

[25]:
```
clf
```

[25]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['age',
                                                    'year_of_publication']),
                                                  ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                   ['book_title', 'book_author',
                                                    'publisher', 'country'])])),
                ('classifier',
                 RandomForestClassifier(max_depth=50, n_jobs=-1, verbose=1))])
```

1.10 Comparing Scalers

- StandardScaler
- MinMaxScaler
- RobustScaler
- MaxAbsScaler
- PowerTransformer
- QuantileTransformer
- Normalizer

[43]:
```python
from time import time

def fit_model_unscaled(algorithm, data):
    categorical_transformer = OneHotEncoder(handle_unknown='ignore')

    preprocessor = ColumnTransformer(
        transformers=[
            ('cat', categorical_transformer, categorical_features)
        ]
    )

    t1 = time()

    X_train, X_test, y_train, y_test = data

    clf = Pipeline(steps=[('preprocessor', preprocessor),
                          ('classifier', algorithm)])


    # train the model
    clf.fit(X_train, y_train)

    # test the model
    y_true = y_test.copy()
    y_pred = clf.predict(X_test)

    # get the classification report
    print(f"\nClassification Report for {algorithm.__class__.__name__}␣
 ↪Unscaled")
```

```
    print(classification_report(y_true, y_pred, target_names=target_names,␣
↪labels=target_names))

    acc_score = accuracy_score(y_true, y_pred)
    bal_score = balanced_accuracy_score(y_true, y_pred)

    print(f"\nAccuracy Score: {acc_score}")
    print(f"Balanced Accuracy Score: {bal_score}")

    print()

    t2 = time()

    print(f'Trained {algorithm.__class__.__name__} in {(t2 - t1)}s')

    return clf
```

[25]:

```
from time import time

def fit_model_scaler(algorithm, data, scaler):

    numeric_transformer = Pipeline(
        steps=[
            ('imputer', SimpleImputer(strategy='median')),
            ('scaler', scaler)]
        )

    categorical_transformer = OneHotEncoder(handle_unknown='ignore')

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_transformer, numeric_features),
            ('cat', categorical_transformer, categorical_features)
        ]
    )

    t1 = time()

    X_train, X_test, y_train, y_test = data

    clf = Pipeline(steps=[('preprocessor', preprocessor),
                          ('classifier', algorithm)])


    # train the model
    clf.fit(X_train, y_train)
    # test the model
    y_true = y_test.copy()
    y_pred = clf.predict(X_test)

    # get the classification report
    print(f"\nClassification Report for {algorithm.__class__.__name__} with␣
↪{scaler.__class__.__name__}")
    print(classification_report(y_true, y_pred, target_names=target_names,␣
↪labels=target_names))

    acc_score = accuracy_score(y_true, y_pred)
    bal_score = balanced_accuracy_score(y_true, y_pred)

    print(f"\nAccuracy Score: {acc_score}")
    print(f"Balanced Accuracy Score: {bal_score}")

    print()

    t2 = time()

    print(f'Trained {algorithm.__class__.__name__} in {(t2 - t1)}s')

    return clf
```

[26]:

```
algo = PassiveAggressiveClassifier(early_stopping=True, n_jobs=-1)
```

1.10.1 Standard Scaler

[27]:

```
std_scaler = StandardScaler()
```

[28]:

```
fit_model_scaler(algo, data=(X_train, X_test, y_train, y_test),␣
↪scaler=std_scaler)
```

```

Classification Report for PassiveAggressiveClassifier with StandardScaler
              precision    recall  f1-score   support

         low       0.06      0.06      0.06      1603
         mid       0.46      0.36      0.40     28483
        high       0.62      0.72      0.67     42828

    accuracy                           0.56     72914
   macro avg       0.38      0.38      0.38     72914
weighted avg       0.55      0.56      0.55     72914


Accuracy Score: 0.5615656801163014
Balanced Accuracy Score: 0.3785619997072959
```

```
Trained PassiveAggressiveClassifier in 7.8669610023498535s
```

```
[28]: Pipeline(steps=[('preprocessor',
                       ColumnTransformer(transformers=[('num',
                                                        Pipeline(steps=[('imputer',
      SimpleImputer(strategy='median')),
                                                                         ('scaler',
      StandardScaler())]),
                                                        ['age',
                                                         'year_of_publication']),
                                                       ('cat',
      OneHotEncoder(handle_unknown='ignore'),
                                                        ['book_title', 'book_author',
                                                         'publisher', 'country'])])),
                      ('classifier',
                       PassiveAggressiveClassifier(early_stopping=True, n_jobs=-1))])
```

1.10.2 MinMax Scaler

```
[29]: minmax_scaler =  MinMaxScaler()
```

```
[30]: fit_model_scaler(algo, data=(X_train, X_test, y_train, y_test),␣
       ↪scaler=minmax_scaler)
```

```
Classification Report for PassiveAggressiveClassifier with MinMaxScaler
              precision    recall  f1-score   support

         low       0.05      0.00      0.01      1603
         mid       0.44      0.55      0.49     28483
        high       0.64      0.55      0.59     42828

    accuracy                           0.54     72914
   macro avg       0.37      0.37      0.36     72914
weighted avg       0.55      0.54      0.54     72914


Accuracy Score: 0.5409249252544093
Balanced Accuracy Score: 0.3694473636392875

Trained PassiveAggressiveClassifier in 7.548543214797974s
```

```
[30]: Pipeline(steps=[('preprocessor',
                       ColumnTransformer(transformers=[('num',
                                                        Pipeline(steps=[('imputer',
      SimpleImputer(strategy='median')),
                                                                        ('scaler',
      MinMaxScaler())]),
                                                        ['age',
                                                         'year_of_publication']),
                                                       ('cat',
      OneHotEncoder(handle_unknown='ignore'),
                                                        ['book_title', 'book_author',
                                                         'publisher', 'country'])])),
                      ('classifier',
                       PassiveAggressiveClassifier(early_stopping=True, n_jobs=-1))])
```

1.10.3 MaxAbs Scaler

[31]:
```
maxabs_scaler = MaxAbsScaler()
```

[32]:
```
fit_model_scaler(algo, data=(X_train, X_test, y_train, y_test),␣
 ↪scaler=maxabs_scaler)
```

```
Classification Report for PassiveAggressiveClassifier with MaxAbsScaler
              precision    recall  f1-score   support

         low       0.07      0.01      0.02      1603
         mid       0.49      0.19      0.27     28483
        high       0.61      0.87      0.72     42828

    accuracy                           0.59     72914
   macro avg       0.39      0.36      0.34     72914
weighted avg       0.55      0.59      0.53     72914


Accuracy Score: 0.5882820857448501
Balanced Accuracy Score: 0.3598257153815126

Trained PassiveAggressiveClassifier in 7.605406761169434s
```

[32]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
MaxAbsScaler())]),
                                                  ['age',
                                                   'year_of_publication']),
                                                 ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['book_title', 'book_author',
                                                   'publisher', 'country'])])),
                ('classifier',
                 PassiveAggressiveClassifier(early_stopping=True, n_jobs=-1))])
```

1.10.4 Robust Scaler

[33]:
```
robust_scaler = RobustScaler(quantile_range=(25, 75))
```

[34]:
```
fit_model_scaler(algo, data=(X_train, X_test, y_train, y_test),␣
 ↪scaler=robust_scaler)
```

```
Classification Report for PassiveAggressiveClassifier with RobustScaler
              precision    recall  f1-score   support

         low       0.08      0.03      0.05      1603
         mid       0.47      0.26      0.33     28483
        high       0.61      0.81      0.70     42828

    accuracy                           0.58     72914
   macro avg       0.39      0.37      0.36     72914
weighted avg       0.54      0.58      0.54     72914


Accuracy Score: 0.5769811010231232
Balanced Accuracy Score: 0.36651556542661007

Trained PassiveAggressiveClassifier in 7.649526119232178s
```

[34]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
RobustScaler(quantile_range=(25,
                75)))]),
                                                  ['age',
                                                   'year_of_publication']),
                                                 ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['book_title', 'book_author',
                                                   'publisher', 'country'])])),
                ('classifier',
                 PassiveAggressiveClassifier(early_stopping=True, n_jobs=-1))])
```

1.10.5 PowerTransform (Yeo-Johnson)

[35]:
```
power_trans = PowerTransformer(method='yeo-johnson')
```

[36]:
```
fit_model_scaler(algo, data=(X_train, X_test, y_train, y_test),␣
 ↪scaler=power_trans)
```

```
/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:205:
RuntimeWarning: overflow encountered in multiply
  x = um.multiply(x, x, out=x)
/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:216:
RuntimeWarning: overflow encountered in reduce
  ret = umr_sum(x, axis, dtype, out, keepdims)


Classification Report for PassiveAggressiveClassifier with PowerTransformer
              precision    recall  f1-score   support

         low       0.06      0.04      0.05      1603
         mid       0.43      0.58      0.50     28483
        high       0.64      0.51      0.57     42828

    accuracy                           0.53     72914
   macro avg       0.38      0.37      0.37     72914
weighted avg       0.55      0.53      0.53     72914


Accuracy Score: 0.5252214938146309
Balanced Accuracy Score: 0.37488500156296717

Trained PassiveAggressiveClassifier in 10.008253574371338s
```

[36]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
PowerTransformer())]),
                                                  ['age',
                                                   'year_of_publication']),
                                                 ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['book_title', 'book_author',
                                                   'publisher', 'country'])])),
                ('classifier',
                 PassiveAggressiveClassifier(early_stopping=True, n_jobs=-1))])
```

1.10.6 Quantile Transform

[37]:
```
quant_trans = QuantileTransformer(output_distribution='normal')
```

[38]:
```
fit_model_scaler(algo, data=(X_train, X_test, y_train, y_test),␣
 ↪scaler=quant_trans)
```

```
Classification Report for PassiveAggressiveClassifier with QuantileTransformer
              precision    recall  f1-score   support

         low       0.08      0.03      0.05      1603
         mid       0.43      0.59      0.50     28483
        high       0.64      0.49      0.56     42828

    accuracy                           0.52     72914
   macro avg       0.38      0.37      0.37     72914
weighted avg       0.54      0.52      0.52     72914


Accuracy Score: 0.5197492936884549
Balanced Accuracy Score: 0.37149925184802246

Trained PassiveAggressiveClassifier in 8.048646688461304s
```

[38]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
QuantileTransformer(output_distribution='normal'))]),
                                                  ['age',
                                                   'year_of_publication']),
                                                 ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['book_title', 'book_author',
                                                   'publisher', 'country'])])),
                ('classifier',
                 PassiveAggressiveClassifier(early_stopping=True, n_jobs=-1))])
```

1.10.7 Normalizer (L2)

[39]:
```
norm_scaler = Normalizer()
```

[40]:
```
fit_model_scaler(algo, data=(X_train, X_test, y_train, y_test),␣
 ↪scaler=norm_scaler)
```

```
Classification Report for PassiveAggressiveClassifier with Normalizer
              precision    recall  f1-score   support

         low       0.05      0.03      0.04      1603
         mid       0.46      0.30      0.36     28483
        high       0.62      0.77      0.69     42828

    accuracy                           0.57     72914
   macro avg       0.38      0.37      0.36     72914
weighted avg       0.54      0.57      0.55     72914


Accuracy Score: 0.5706722988726445
Balanced Accuracy Score: 0.36646952962846085

Trained PassiveAggressiveClassifier in 8.04029631614685s
```

[40]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
Normalizer())]),
                                                  ['age',
                                                   'year_of_publication']),
                                                 ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['book_title', 'book_author',
                                                   'publisher', 'country'])])),
                ('classifier',
                 PassiveAggressiveClassifier(early_stopping=True, n_jobs=-1))])
```

1.10.8 Unscaled

[44]:
```
fit_model_unscaled(algo, data=(X_train, X_test, y_train, y_test))
```

```
Classification Report for PassiveAggressiveClassifier Unscaled
              precision    recall  f1-score   support

         low       0.09      0.02      0.03      1603
         mid       0.48      0.24      0.32     28483
        high       0.61      0.83      0.70     42828

    accuracy                           0.58     72914
   macro avg       0.39      0.36      0.35     72914
```

```
weighted avg       0.55      0.58      0.54     72914


Accuracy Score: 0.5831938996626163
Balanced Accuracy Score: 0.36462720452376884

Trained PassiveAggressiveClassifier in 7.465895652770996s
```

[44]:

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['book_title', 'book_author',
                                                   'publisher', 'country'])])),
                ('classifier',
                 PassiveAggressiveClassifier(early_stopping=True, n_jobs=-1))])
```

[ ]:

BookReview-Classification-Unsupervised

January 20, 2021

[ ]:

```
! pip install --upgrade scikit-learn
```

[ ]:

```
! pip install scikit-learn-extra
```

[ ]:

```
! pip install hdbscan
```

[ ]:

```
! python -m spacy download en_core_web_sm
```

---

1 Book Crossing - Classification (Unsupervised)

[3]:

```
%matplotlib inline

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

sns.set()
palette = sns.color_palette("icefire")

plt.style.use('ggplot')

sns.set_context("talk")
```

[127]:

```
dataset = pd.read_csv('book_crossing.classification.cleaned.csv')
```

[128]:

```
dataset['age'] = dataset['age'].astype(np.float64)
dataset['book_rating'] = dataset['book_rating'].astype('category')
dataset['book_title'] = dataset['book_title'].astype('category')
dataset['book_author'] = dataset['book_author'].astype('category')
dataset['year_of_publication'] = dataset['year_of_publication'].astype(np.
 ↪float64)
dataset['publisher'] = dataset['publisher'].astype('category')
dataset['country'] = dataset['country'].astype('category')
```

[136]:
```
dataset['book_title'].cat.categories.shape
```

[136]:
```
(132033,)
```

[139]:
```
dataset['book_author'].cat.categories.shape
```

[139]:
```
(60652,)
```

[140]:
```
dataset['publisher'].cat.categories.shape
```

[140]:
```
(11311,)
```

[141]:
```
dataset['country'].cat.categories.shape
```

[141]:
```
(51,)
```

[6]:
```
dataset.head()
```

[6]:
```
    age book_rating  ...              publisher country
0  34.0         mid  ...  HarperFlamingo Canada  canada
1  30.0        high  ...  HarperFlamingo Canada  canada
2  34.0        high  ...  HarperFlamingo Canada  canada
3  34.0        high  ...  HarperFlamingo Canada  canada
4  34.0        high  ...  HarperFlamingo Canada  canada

[5 rows x 7 columns]
```

[7]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 364570 entries, 0 to 364569
Data columns (total 7 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   age                  364570 non-null  float64
 1   book_rating          364570 non-null  category
 2   book_title           364570 non-null  category
 3   book_author          364570 non-null  category
 4   year_of_publication  364570 non-null  float64
 5   publisher            364570 non-null  category
 6   country              364570 non-null  category
dtypes: category(5), float64(2)
memory usage: 19.1 MB
```

[8]:
```
c_dataset = dataset["book_title"].astype(str) + " "  + \
            dataset["book_author"].astype(str) + " " + \
            dataset["publisher"].astype(str) + " " + \
            dataset["year_of_publication"].astype(str) + " " + \
            dataset["age"].astype(str) + " " + \
            dataset["country"].astype(str)
```

[9]:
```
for ex in c_dataset.sample(frac=0.2)[:5]:
    print(ex)
```

```
The Vanishing Vampire (The Accidental Monsters , No 1) David Lubar Scholastic
1997.0 12.0 usa
REMEMBER ME Mary Higgins Clark Simon &amp; Schuster 1994.0 34.0 usa
The Mummy or Ramses the Damned Anne Rice Ballantine Books 1991.0 22.0 usa
Bittersweet Rain Sandra Brown Warner Books 2000.0 34.0 usa
Arthur Stephen R. Lawhead Zondervan Publishing Company 1996.0 21.0 usa
```

[92]:

```python
small_dataset = dataset.copy().sample(frac=0.03)
```

[93]:

```python
small_dataset.shape
```

[93]:

```
(10937, 7)
```

[94]:

```python
c_dataset_small = small_dataset["book_title"].astype(str) + " "  + \
                  small_dataset["book_author"].astype(str) + " " + \
                  small_dataset["publisher"].astype(str) + " " + \
                  small_dataset["year_of_publication"].astype(str) + " " + \
                  small_dataset["age"].astype(str) + " " + \
                  small_dataset["country"].astype(str)
```

[95]:

```python
import spacy
from sklearn.base import BaseEstimator, TransformerMixin
import en_core_web_sm

nlp = en_core_web_sm.load()
```

[96]:

```python
from sklearn import set_config
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, plot_confusion_matrix,␣
 ↪confusion_matrix, accuracy_score, balanced_accuracy_score
from sklearn.cluster import KMeans, DBSCAN, Birch, MiniBatchKMeans,␣
 ↪SpectralClustering, AgglomerativeClustering, MeanShift, AffinityPropagation,␣
 ↪OPTICS
from sklearn.decomposition import TruncatedSVD

set_config(display='diagram')
```

[97]:

```python
class SpacyVectorTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, nlp):
        self.nlp = nlp
        self.dim = 300

    def fit(self, X, y):
        return self

    def transform(self, X):
        # Doc.vector defaults to an average of the token vectors.
        # https://spacy.io/api/doc#vector
        return [self.nlp(text).vector for text in X]
```

[98]:

```
X, y = c_dataset, dataset['book_rating']
```

[99]:

```
X_small, y_small = c_dataset_small, small_dataset['book_rating']
```

[100]:

```
target_names = ['low', 'mid', 'high']
```

1.1 Unsupervised Models

- KMedoids
- AgglomerativeClustering
- DBSCAN
- KMeansClustering
- HDBSCAN
- MiniBatchKMeans

[102]:

```
"""
km = KMeans(n_clusters=3) # works (needs remap of output)
dbscan = DBSCAN()
birch = Birch() # crash
mbkm = MiniBatchKMeans() # works (needs remap of output)
sc = SpectralClustering() # crash
ac = AgglomerativeClustering() # sparse not supported
ms = MeanShift(bandwidth=3) # sparse not supported
ap = AffinityPropagation() # crash
oo = OPTICS() # sparse not supported
"""
```

[102]:

```
'\nkm = KMeans(n_clusters=3) # works (needs remap of output)\ndbscan =
DBSCAN()\nbirch = Birch() # crash\nmbkm = MiniBatchKMeans() # works (needs remap
of output)\nsc = SpectralClustering() # crash\nac = AgglomerativeClustering() #
sparse not supported\nms = MeanShift(bandwidth=3) # sparse not supported\nap =
AffinityPropagation() # crash\noo = OPTICS() # sparse not supported\n'
```

[103]:

```
from time import time

def fit_model(algorithm, data):

    t1 = time()

    X, y = data

    print(f'\nStarted Training {algorithm.__class__.__name__} on X: {X.shape} y:
 ↪ {y.shape}')

    embeddings_pipeline = Pipeline(
        steps=[
            ("mean_embeddings", SpacyVectorTransformer(nlp=nlp)),
            ("reduce_dim", TruncatedSVD(50)),
            ("clusterer", algorithm),
        ]
    )

    # train the model
    embeddings_pipeline.fit(X, y)

    print(f"\nEvaluating model on X_test: {X.shape} y_test: {y.shape}")

    # test the model
    y_true = y.copy()

    if isinstance(embeddings_pipeline['clusterer'], (AgglomerativeClustering,␣
↪DBSCAN, OPTICS, HDBSCAN)):
        y_pred = embeddings_pipeline['clusterer'].labels_
    else:
        y_pred = embeddings_pipeline.predict(X)

    y_pred = np.array(list(map(lambda x: "low" if x == 0 else "mid" if x == 1␣
↪else "high", y_pred)))

    # get the classification report
    print(f"\nClassification Report for {algorithm.__class__.__name__}")
    print(classification_report(y_true, y_pred, target_names=target_names,␣
↪labels=target_names))

    acc_score = accuracy_score(y_true, y_pred)
    bal_score = balanced_accuracy_score(y_true, y_pred)

    print(f"\nAccuracy Score: {acc_score}")
    print(f"Balanced Accuracy Score: {bal_score}")

    print()
    # show the confusion matrix
    cmmat_table = pd.DataFrame({'y_true': y_true, 'y_pred': y_pred})
    conmat = pd.crosstab(cmmat_table.y_true, cmmat_table.y_pred,␣
↪rownames=['Actual'], colnames=['Predicted'], margins=True, normalize='all')
    ax = plt.axes()
    sns.set(rc={'figure.figsize':(9, 7)})
    sns.heatmap(conmat, annot=True, ax=ax)
    ax.set_title(f'{algorithm.__class__.__name__}')
    plt.show()
    print()

    t2 = time()
    print(f'Trained {algorithm.__class__.__name__} in {(t2 - t1)}s')

    return embeddings_pipeline
```

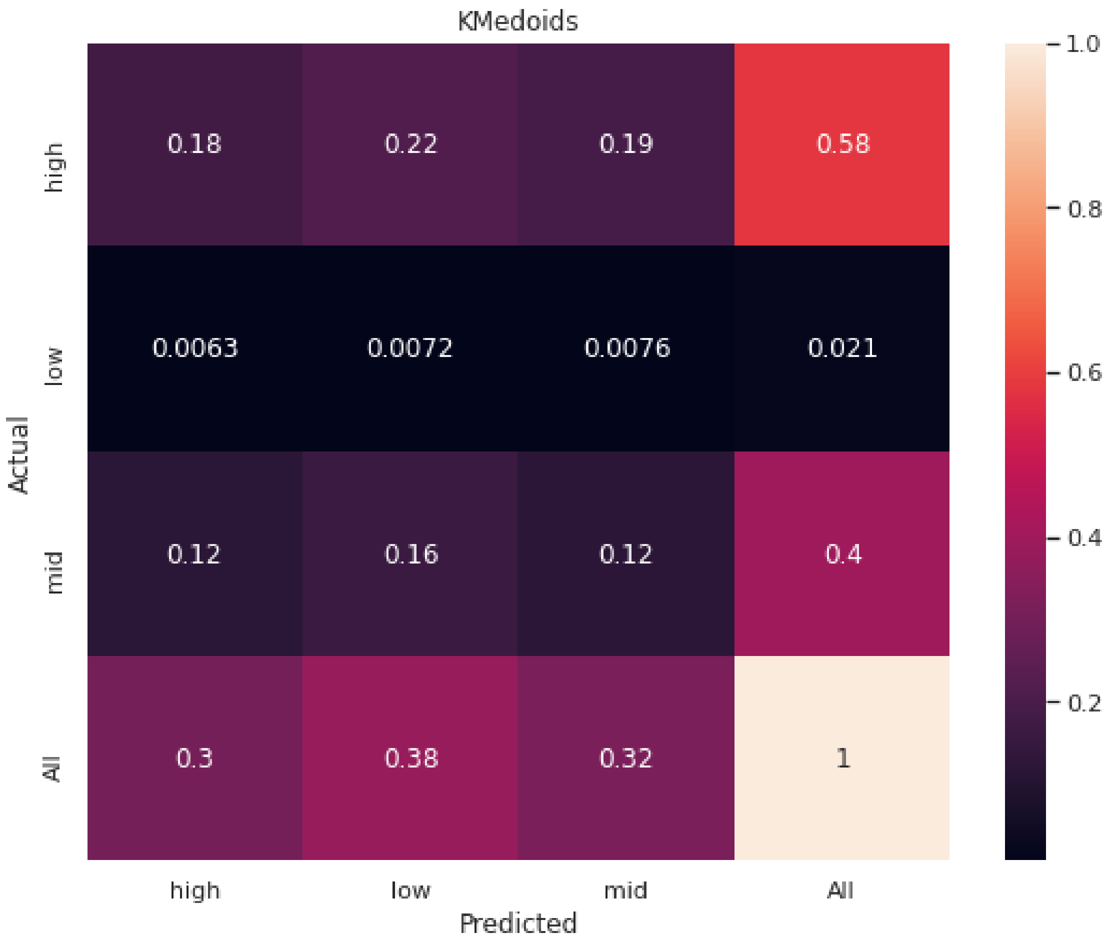
1.2 KMedoids Clustering

[107]:
```
from sklearn_extra.cluster import KMedoids
```

[108]:
```
kmedoids = KMedoids(n_clusters=3, max_iter=1)
```

[109]:
```
clf = fit_model(algorithm=kmedoids, data=(X_small, y_small))
```

```
Started Training KMedoids on X: (10937,) y: (10937,)

Evaluating model on X_test: (10937,) y_test: (10937,)

Classification Report for KMedoids
              precision    recall  f1-score   support

         low       0.02      0.34      0.04       231
         mid       0.38      0.30      0.34      4333
        high       0.60      0.31      0.41      6373

    accuracy                           0.31     10937
   macro avg       0.33      0.32      0.26     10937
weighted avg       0.50      0.31      0.37     10937


Accuracy Score: 0.30785407332906645
Balanced Accuracy Score: 0.3184037656596693
```



```
Trained KMedoids in 223.7921495437622s
```

[110]:
```
clf
```

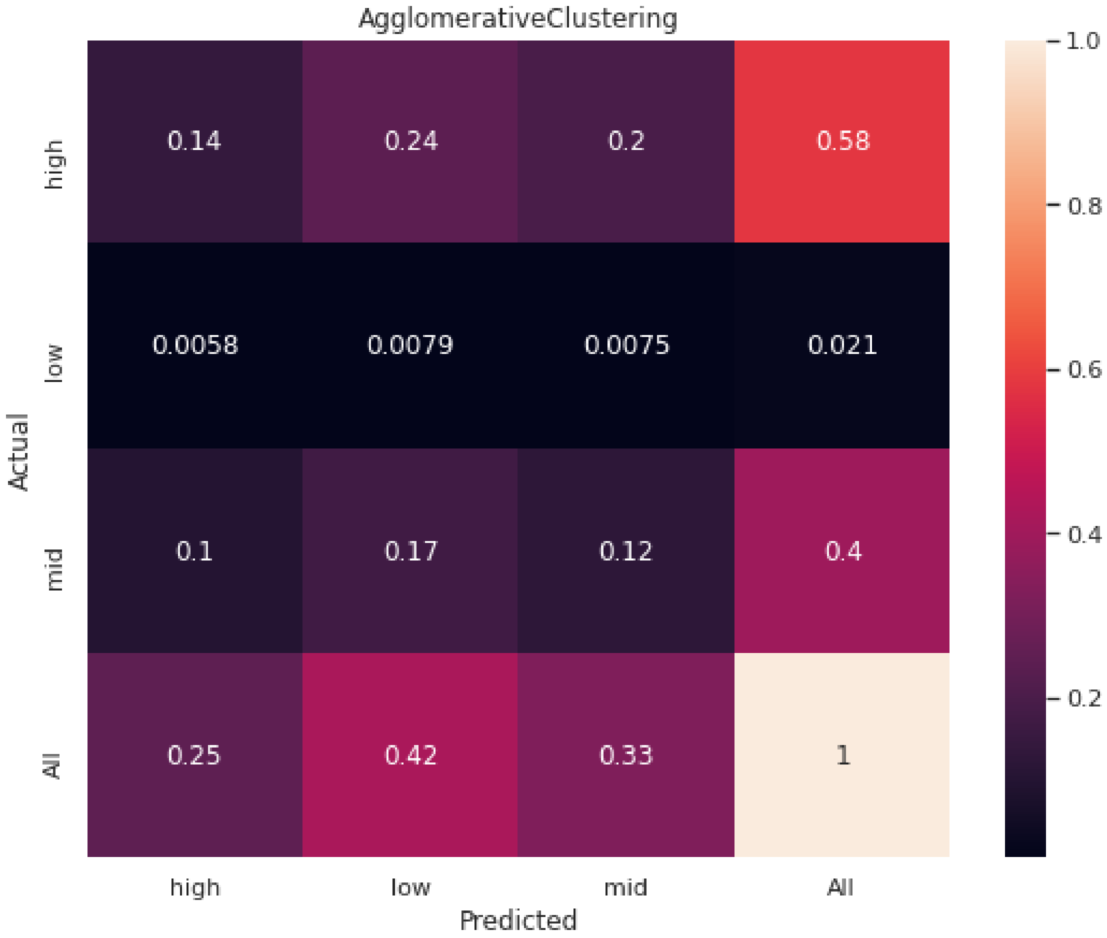
[110]:
```
Pipeline(steps=[('mean_embeddings',
                  SpacyVectorTransformer(nlp=<spacy.lang.en.English object at
0x7ff253d3b6d8>)),
                 ('reduce_dim', TruncatedSVD(n_components=50)),
                 ('clusterer', KMedoids(max_iter=1, n_clusters=3))])
```

1.3 Agglomerative Clustering

[111]:
```
aggc = AgglomerativeClustering(n_clusters=3)
```

[112]:
```
clf = fit_model(algorithm=aggc, data=(X_small, y_small))
```

```
Started Training AgglomerativeClustering on X: (10937,) y: (10937,)

Evaluating model on X_test: (10937,) y_test: (10937,)

Classification Report for AgglomerativeClustering
              precision    recall  f1-score   support

         low       0.02      0.37      0.04       231
         mid       0.37      0.31      0.34      4333
        high       0.57      0.24      0.34      6373

    accuracy                           0.27     10937
   macro avg       0.32      0.31      0.24     10937
weighted avg       0.48      0.27      0.33     10937


Accuracy Score: 0.2729267623662796
Balanced Accuracy Score: 0.30877414843520845
```



```
Trained AgglomerativeClustering in 119.64335346221924s
```

[113]:
```
clf
```

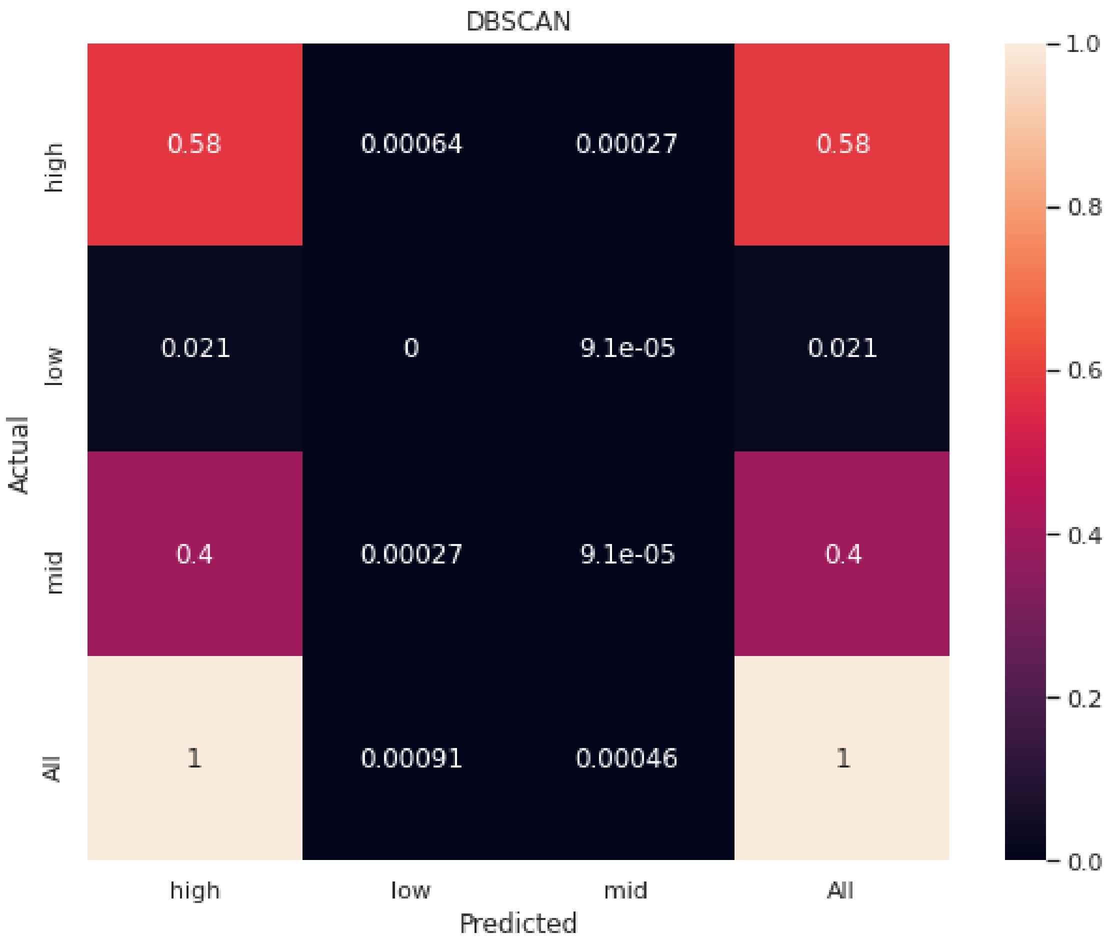
[113]:
```
Pipeline(steps=[('mean_embeddings',
                 SpacyVectorTransformer(nlp=<spacy.lang.en.English object at
0x7ff253d3b6d8>)),
                ('reduce_dim', TruncatedSVD(n_components=50)),
                ('clusterer', AgglomerativeClustering(n_clusters=3))])
```

1.4 DBSCAN

[114]:
```
dbscan = DBSCAN(n_jobs=-1)
```

[115]:
```
clf = fit_model(algorithm=dbscan, data=(X_small, y_small))
```

```
Started Training DBSCAN on X: (10937,) y: (10937,)

Evaluating model on X_test: (10937,) y_test: (10937,)

Classification Report for DBSCAN
              precision    recall  f1-score   support

         low       0.00      0.00      0.00       231
         mid       0.20      0.00      0.00      4333
        high       0.58      1.00      0.74      6373

    accuracy                           0.58     10937
   macro avg       0.26      0.33      0.25     10937
weighted avg       0.42      0.58      0.43     10937


Accuracy Score: 0.5818780287098839
Balanced Accuracy Score: 0.3328872224199264
```



```
Trained DBSCAN in 116.67007970809937s
```

[116]:
```
clf
```

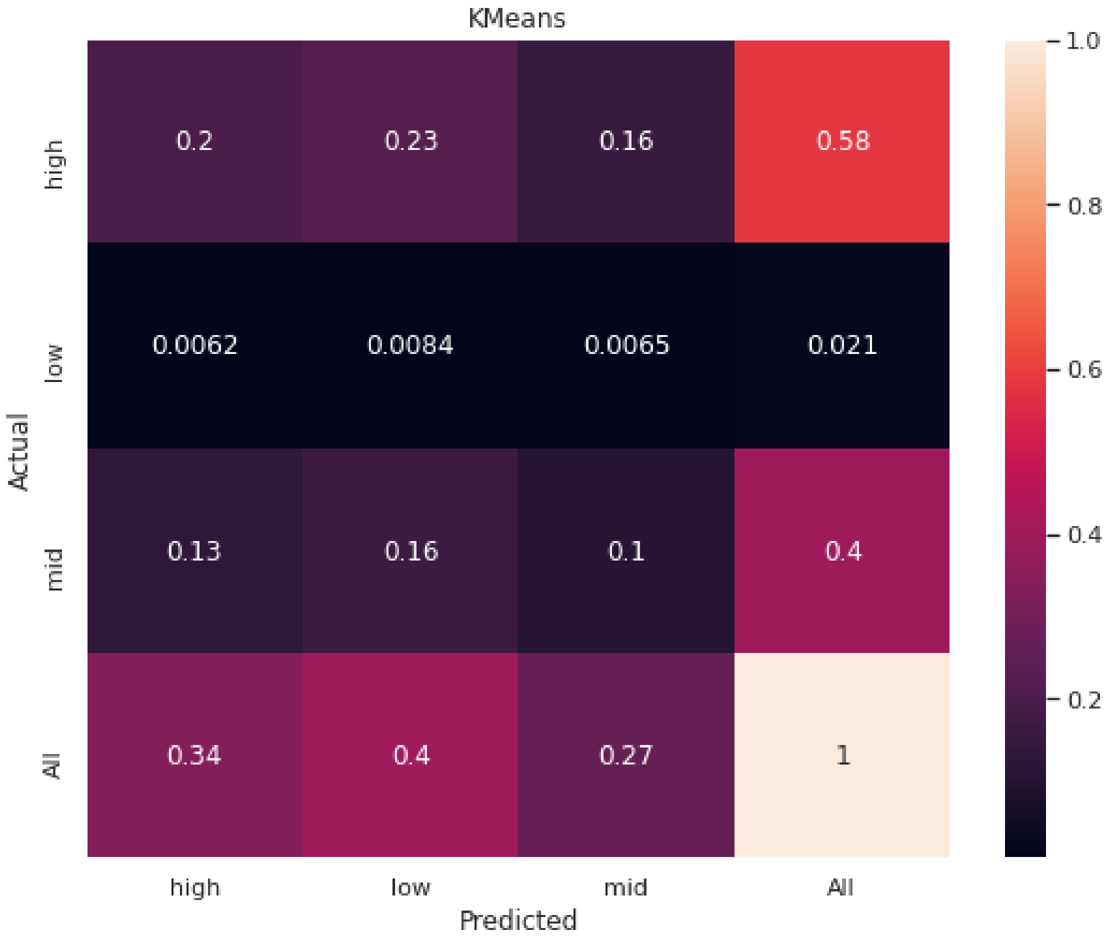
[116]:
```
Pipeline(steps=[('mean_embeddings',
                 SpacyVectorTransformer(nlp=<spacy.lang.en.English object at
0x7ff253d3b6d8>)),
                ('reduce_dim', TruncatedSVD(n_components=50)),
                ('clusterer', DBSCAN(n_jobs=-1))])
```

1.5 KMeans Clustering

[117]:
```
kmeans = KMeans(n_clusters=3, n_init=3)
```

[118]:
```
clf = fit_model(algorithm=kmeans, data=(X_small, y_small))
```

```
Started Training KMeans on X: (10937,) y: (10937,)

Evaluating model on X_test: (10937,) y_test: (10937,)

Classification Report for KMeans
              precision    recall  f1-score   support

         low       0.02      0.40      0.04       231
         mid       0.39      0.26      0.32      4333
        high       0.60      0.35      0.44      6373

    accuracy                           0.32     10937
   macro avg       0.34      0.34      0.27     10937
weighted avg       0.50      0.32      0.38     10937


Accuracy Score: 0.31535155892840816
Balanced Accuracy Score: 0.33658550914710594
```



```
Trained KMeans in 222.76584196090698s
```

[119]:
```
clf
```

[119]:
```
Pipeline(steps=[('mean_embeddings',
                 SpacyVectorTransformer(nlp=<spacy.lang.en.English object at
0x7ff253d3b6d8>)),
                ('reduce_dim', TruncatedSVD(n_components=50)),
                ('clusterer', KMeans(n_clusters=3, n_init=3))])
```

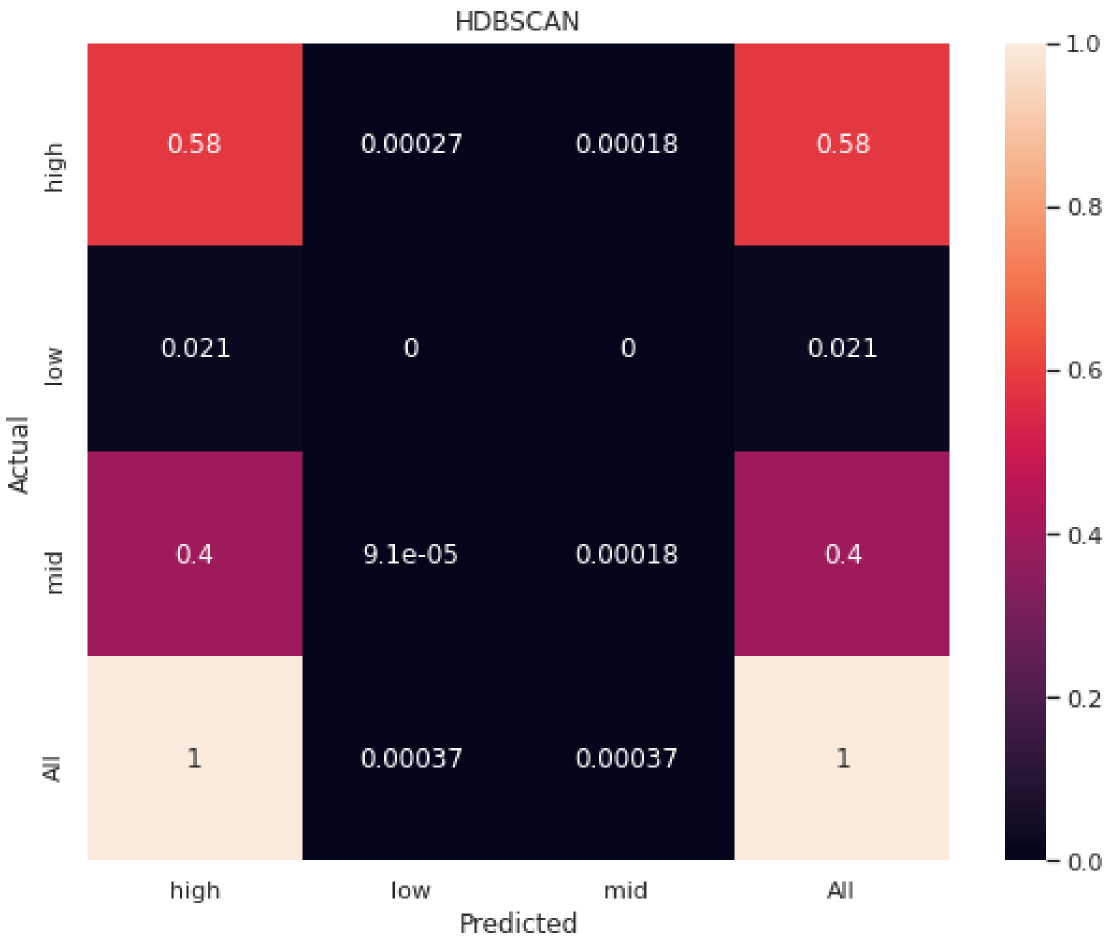
1.6 HDBSCAN

[120]:
```
from hdbscan import HDBSCAN
```

[121]:
```
hscan = HDBSCAN(min_cluster_size=3)
```

[122]:
```
clf = fit_model(algorithm=hscan, data=(X_small, y_small))
```

```
Started Training HDBSCAN on X: (10937,) y: (10937,)

Evaluating model on X_test: (10937,) y_test: (10937,)

Classification Report for HDBSCAN
              precision    recall  f1-score   support

         low       0.00      0.00      0.00       231
         mid       0.50      0.00      0.00      4333
        high       0.58      1.00      0.74      6373

    accuracy                           0.58     10937
   macro avg       0.36      0.33      0.25     10937
weighted avg       0.54      0.58      0.43     10937


Accuracy Score: 0.5824266252171528
Balanced Accuracy Score: 0.3332256713684369
```



```
Trained HDBSCAN in 121.39391756057739s
```

[123]:
```
clf
```

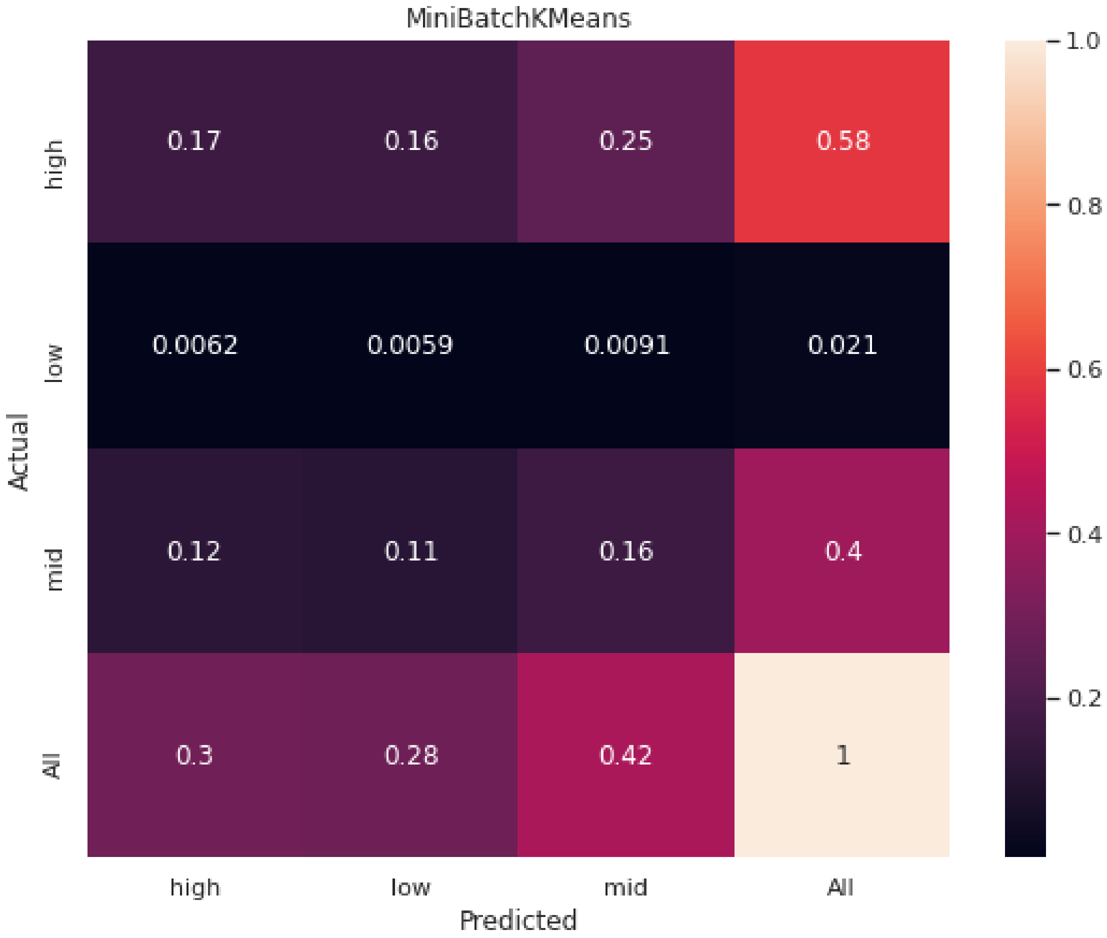
[123]:
```
Pipeline(steps=[('mean_embeddings',
                 SpacyVectorTransformer(nlp=<spacy.lang.en.English object at
0x7ff253d3b6d8>)),
                ('reduce_dim', TruncatedSVD(n_components=50)),
                ('clusterer', HDBSCAN(min_cluster_size=3))])
```

1.7 Mini Batch KMeans

[124]:
```
mb_kmeans = MiniBatchKMeans(n_clusters=3, max_iter=1)
```

[125]:
```
clf = fit_model(algorithm=mb_kmeans, data=(X_small, y_small))
```

```
Started Training MiniBatchKMeans on X: (10937,) y: (10937,)

Evaluating model on X_test: (10937,) y_test: (10937,)

Classification Report for MiniBatchKMeans
              precision    recall  f1-score   support

         low       0.02      0.28      0.04       231
         mid       0.39      0.41      0.40      4333
        high       0.57      0.29      0.38      6373

    accuracy                           0.34     10937
   macro avg       0.33      0.33      0.27     10937
weighted avg       0.49      0.34      0.38     10937


Accuracy Score: 0.3378440157264332
Balanced Accuracy Score: 0.32632201462411303
```



```
Trained MiniBatchKMeans in 217.56095910072327s
```

[126]:
```
clf
```

[126]:
```
Pipeline(steps=[('mean_embeddings',
                 SpacyVectorTransformer(nlp=<spacy.lang.en.English object at
0x7ff253d3b6d8>)),
                ('reduce_dim', TruncatedSVD(n_components=50)),
                ('clusterer', MiniBatchKMeans(max_iter=1, n_clusters=3))])
```

[ ]:
```

```