**MA 5851: Assessment 1: NLP recommendation Engine**
**Paul Kirkwood**

# 1. Introduction

The purpose of this report is to describe the construction of two book recommender systems using different neural linguistic processing techniques and machine learning approaches.
The report details the key processes of extracting, augmenting and pre-processing data, using the provided base data set. The report will describe the search and acquisition of external data from a variety of web-based datasets using python. Each source is tested and assessed for suitability before the final dataset and corpus are prepared.
Two recommender engines are developed in python, a content-based system and a separate collaboration-based system. In each case, different machine learning algorithms are explored to compare performance.
All coding was undertaken in Juypter notebooks within the Pycharm IDE.

## 2. Data Generation

The base dataset 'MA5851_SP86_2021_A1.xlsx' was imported into python using pandas read_excel. An initial integrity check was carried out to check for duplicates and null entries, before the dataset was converted to a dataframe for further work.

The loaded dataset had comprised a dataframe of five variables [School_ID, State, Year, Subject & ISBN] with 1804 observations or rows.

The data, aside from the subject, does not provide sufficient information to enable recommendations to be made for similar books, subject matter, topic, genre, etc. Augmentation of the data is required and this is to be obtained from web based repositories using webcalls or API's.

Data Preparation

The API calls each used the ISBN as the unique search parameter for each book. This variable was extracted from the imported excel dataset and as a list for use in API calls. Given the size of the data (1804 observations) a subset of 10 ISBN was prepared for testing purposes, this subset was used to validate and refine the API calls before the full search was conducted. All API requests used the python 'requests' base function.

The following repositories were tested, function code (where appropriate) for each is contained within the code appendix to this report.

- Googlebooks.

  The call passed the ISBN number as a general query and after testing for a valid response, extracted the following terms.

| **Variable** | **Description** |
|---|---|
| title | Book title by ISBN |
| publishedDate | String containing date(s) of publication, this also included range dates for multiple publications and mixed date formats |
| decription | String containing a brief book description. |
| authors | String listing authors by name |
| ratingsCount | Integer returned denoting the number of ratings on record. |
| averageRating | Floating point variable returned denoting the average value of ratings. |

- ISBND

  A similar approach to Googlebooks, the extraction process from the returned JSON file is also identical. The dataset was more limited and only the following could eb extracted:

| **Variable** | **Description** |
|---|---|

| | |
|---|---|
| title | Book title by ISBN |
| synopsys | String containing a brief book description. |
| authors | String listing authors by name |

- Trove

  Trove required a slightly different approach as the repository required an API key to be obtained, but as specific 'ISBN' search parameter to be used rather than a general search. The call returned a JSON variable.

- Openlibrary.org

  Open library accepted API calls using the Google format, however, I was unable to return valid search results using the 13 digit ISBN. Conversion to 10 digit also failed but conversion to LCCN book identifiers did produce a comprehensive response.

- Goodreads

  Although widely referenced, the Goodreads API was deprecated in 2020 and could not be accessed.

- Amazon

  API calls could only be made via the Amazon partner programme which the author did not have permission to access.

API Performance

An assessment of the successful API calls is as follows:

| **API** | **Assessment** |
|---|---|
| Goodreads | The API required an interim conversion of ISBN to LCCN to be effective. While this is not insurmountable and may be due to a lack experience with the API call, this additional step requires additional time and further API searches and was therefore rejected. |
| GoogleBooks | The GoogleBooks API responses were largely complete. An assessment of the final dataset confirmed that 79 of the 1804 calls provided no book title data. The dataset was therefore reasonably complete and provided sufficient information to build the book recommender.<br>GoogleBooks however, has a call quota of 100 calls in 100 seconds. Exceeding this quota caused the get function to fail. While batching is an option for larger datasets, in this instance, a delay timer was built into the get function that paused the process when reaching the quota limit. |
| Trove | Trove returned comprehensive data, however the author was unable to extract data from multiple layers of nesting within the JSON() variable. |

| | |
|---|---|
| ISBND | ISBND performed well and is noticeably faster than GoogleBooks which may be attributable to traffic management being used by Google. However, the returned JSON data was sparce and samples provide less than 50% of requested data. |

Based on the above, the GoogleBooks API was selected for data augmentation.

The GoogleBooks API code is as follows:

```
#%% GOOGLEBOOKS API

endpoint = "https://www.googleapis.com/books/v1/volumes"

def get_info_from_google_book(isbn):
  params = {"q":"isbn="+str(isbn), "maxResults": 1}
  response = requests.get(endpoint, params=params).json()
  #print("Fetching detail for ISBN:"+str(isbn))
  print(response)
#extract the key variables, testing for missing items
  if (response['totalItems']>0):
    for book in response["items"]:
        volume = book["volumeInfo"]
        title = volume["title"]
        if 'publishedDate' in volume.keys():
            published = volume["publishedDate"]
        else: published = ""
        if "description" in volume.keys():
            description = volume["description"]
        else:
            description = ""
        if "authors" in volume.keys():
             author = volume["authors"]
        else:
            author = ""
        if 'ratingsCount' in volume.keys():
            ratings = volume["ratingsCount"]
        else:
            ratings = 0
        if 'averageRating' in volume.keys():
            score = volume["averageRating"]
        else:
            score = 0

        #print(f"{title} ({published}) | {description}")
    return {'isbn':isbn, 'title':title, 'authors': author, 'published':published,'rating':ratings,
'score':score, 'description':description}
  else:
    return {'isbn':isbn, 'title':'', 'authors':'', 'published':'','rating':'', 'score':'',
'description':''}
#Build the raw corpus/dataset
info2 = []
timer = 0
for isbn in isbn_list:
    timer = timer + 1 #introducing delay function to overcome timing quotas
    info2.append(get_info_from_google_book(isbn))
    if timer > 90:
        time.sleep(60)
        timer = 0
    else:
        continue
```

## Data Augmentation

Data from the API call was appended to the original imported dataset using the ISBN from both as the key. The final dataset comprised 11 variables [‘School_ID, State, Year, Subject, ISBN, Title, Authors, Published, Rating, Score & Description’] with the 1804 observations remaining. A copy of the final dataset was saved as ‘results.csv’

## 3. Data Wrangling

The augmented dataset comprised raw data that required cleaning and pre-processing before use. The following steps were carried out:

Date Format

The date variable contained multiple formats (including days/months and year ranges). For simplicity, a single year of publication was extracted as the alternative would require each publication to be treated as a separate book entry. The Google returned date as YYYY(first)-YYYY(last), DD, MM. Extracting the first four characters of the string provided the years of first publication.

```
result_df['published'] = result_df['published'].str[:4]
```

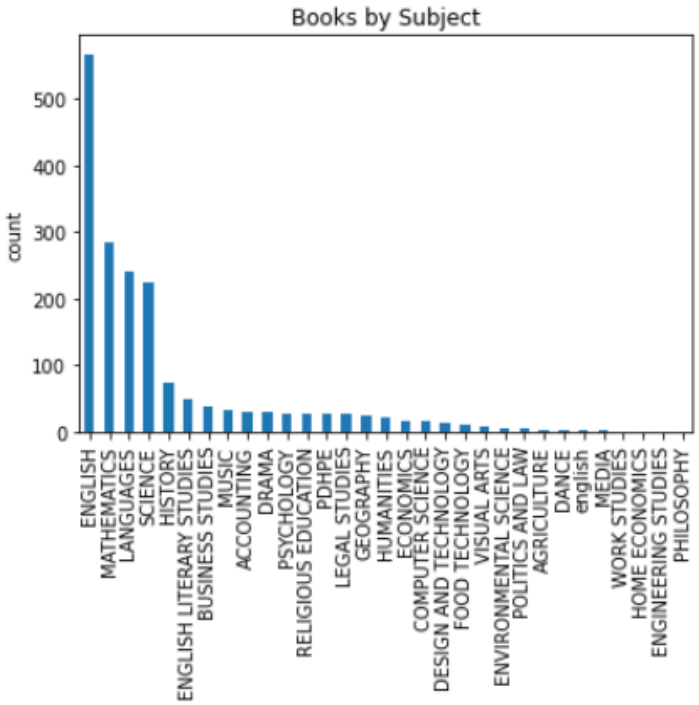
Removing Unwanted Characters from Strings.

Both the Authors and Description variables contained punctuation, whitespace and non-text characters that needed to be removed.

```
#remove unwanted characters from columns
result_df['authors'] = result_df['authors'].str.replace(r'[^\w\s]',' ')
result_df['description'] = result_df['description'].str.replace(r'[^\w\s]+',' ')
```

Exploring Data Distribution

A summary plot of the subjects noted that the data is heavily skewed to the key subjects of English, maths, languages and science.



This is not unexpected as the prevalence of these subjects in every school contrast with niche subjects that may be taught in few schools or year groups This suggests, however, that modelling books by subject may be difficult, particularly for rule-based algorithms as they may default to the 'high probability' outcomes.

ISBN is Not a Parameter

The ISBN value is not a parameter as it is unique to each book and cannot be used as a recommender value.

Rating Metrics

The rating metrics obtained from Google are sparse, with only 291 entries containing a rating value >0. A count of scores given returned 292 ratings >0. Care needs to be taken with these metrics as 0 may be a score given or possibly an average.

The final corpus created as a basis for the modelling is as follows:

| | School_ID | State | Year | Subject | isbn | title | authors | published | rating | score | description |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 8 | VIC | 0 | ENGLISH | 9781741250879 | Victorian Targeting Handwriting | Tricia Dearborn Jo Ryan Stephen Michae... | 2004 | 0.0 | 0.0 | Targeting Handwriting Victoria Year 6 Student ... |
| **1** | 9 | NSW | 0 | ENGLISH | 9780648237327 | Kluwell My Home Reading Yellow Level | Stan Kluzek Andrew Coldwell | 2019 | 0.0 | 0.0 | Suitable for lower primary students |
| **2** | 15 | NSW | 0 | ENGLISH | 9781742990682 | Wilfrid Gordon McDonald Partridge | Mem Fox Julie Vivas | 1987 | 58.0 | 4.5 | The elderly Miss Nancy is helped by Wilfrid to... |

Data types will be set depending upon the model and algorithm being used. The ISBN for instance may be treated as both an integer or string depending upon it's use.

## 4. Pre-processing for Recommender Systems.

As outlined in the introduction, two recommender models are to be create, one content based using information contained within the books metadata and the second based upon collaborative filtering that in this case will utilise the rating scores.

Both models require distinct datasets; sample subsets are extracted from the corpus created in the earlier steps.

Testing and training subsets will be created in the model development sections below.

Content Based System: Data Preparation

As noted earlier, the content for this form of recommender must be drawn from the metadata. Content, however, is not limited to subject matter alone and must consider other factors such as the age group, author and year of publication.

The proposal is therefore to create a content subset that combines each of these variables into a single subset that can be that can be parsed to use in a machine learning system.

A data subset is created using the subject, title, authors, published and description variables, these are then combined to create a single string variable.

```
combined_features['combined'] = combined_features['Subject']+' '+\
                                combined_features['title']+' '+\
                                combined_features['authors']+' '+\
                                combined_features['description']

combined_features['combined'] = combined_features['combined'].apply(str)
```

This string is then simplified using NLP techniques to first remove unhelpful or unnecessary words such as ‘the, is, a,’ etc. using the English stopwords from the NLTK dictionary.

The corpus is then ‘cleaned’ using a function that steps through each observation to remove non-characters, punctuation and excess whitespace. The corpus was also converted to lowercase to remove potential differentiation.

It is noted at this point that the corpus will need to be split into word terms for analysis to prevent searching on a letter-by-letter basis, this step will be carried out in the transformation step of the modelling.

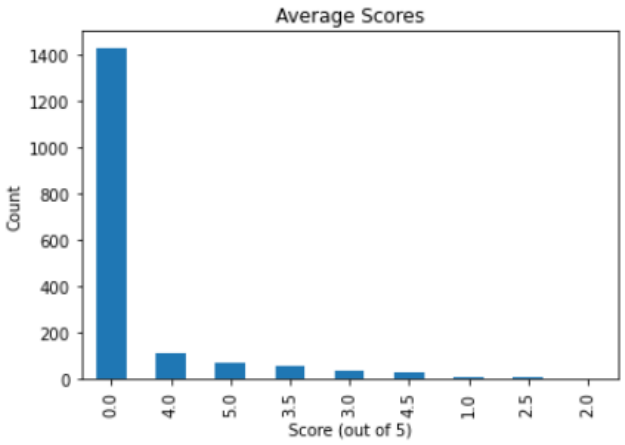
```
combined_features['cleaned'].head()
```

```
0    english victorian targeting handwriting tricia...
1    english kluwell my home reading yellow level s...
2    english wilfrid gordon mcdonald partridge mem ...
3    english imaths mary serenc chris linthorne len...
4                                                  nan
Name: cleaned, dtype: object
```

A final check to remove nan’s was carried out.

Collaboration Based System: Data Preparation

The collaboration-based engine will use the rating information extracted from Googlebooks. Two variables are available: the number of reviews and the average rating given. An analysis of the score data confirms that the dataset is very sparce.




```
0.0    1433
4.0     106
5.0      65
3.5      54
3.0      31
4.5      28
1.0       4
2.5       3
2.0       1
Name: score, dtype: int64
```

For this model, the approach selected is to use the counts of ratings as the metric for the model. The reasoning being that the spread of average scores is wide (0 to 5) with very low results for values below 3 (8 scores from 1804 books). By selecting the count, the variance is effectively binary, is there a score > 0, yes or no. While remaining sparce, the number of scores >0 is 292 (16.1%).

```
train_set = collab_dataset[['School_ID', 'isbn', 'rating']]
train_set.head()
```

| | School_ID | isbn | rating |
|---|---|---|---|
| 0 | 8 | 9781741250879 | 0.0 |
| 1 | 9 | 9780648237327 | 0.0 |
| 2 | 15 | 9781742990682 | 58.0 |
| 3 | 15 | 9781741351750 | 0.0 |
| 4 | 15 | 9781742152196 | 0.0 |

The sample set is created and retains the School_ID as a user ID variable that may be required by some algorithms. The data extracted from Google does not identifiy the reviewer and hence we cannot directly link to the google review to the school. In this instance, an assumption is being made that the likelihood is that a school would rate a book at a similar level to google.

The issue of sparce reviews in developing a collaborative model is addressed in the modelling sections below. Cold-starting recommender models is a recognised issue, the

sparsity of reviews act as a skew to the data, with recommenders being at risk of only suggesting previously reviewed books which in turn generates a bias is the way future reviews are generated predominantly on previously reviewed books.

# 5. Recommender System #1: Content based Recommender.

The content-based model proposed uses the combined feature corpus developed previously in the report. The EDA carried out had noted the skewness of the data, over representing certain book subjects, English, for instance. While omission may be an option, given the subject remains a critical component of selection for suitability, the skewness must be overcome through normalisation or weighting of features or terms within the dataset.

It is proposed to use a term frequency weighting using inverse document frequency (TFIDF) on the corpus. This approach to weighting assigns importance to the keywords or terms within the corpus, applying increased weighting to lower frequency words. In practice this may for instance, provide a higher weighting for the authors name, but reduce weighting for 'English' providing greater prominence to key subject matter keywords and thus overcome the skewed data observed earlier.

Model Preparation

The model will be built using the TfidVectorizer function from sklearn to create a frequency array that can be passed to a distance function from sklearn. The objective being to calculate a distance metric between each array vector that corresponds with a book. The distance function used is the cosine similarity function from sklearn.

```
#%% COSINE SIMILARITY
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
tfidf_rev = tfidf_vectorizer.fit_transform((combined_features['cleaned']))

from sklearn.metrics.pairwise import cosine_similarity
book_tfidf = tfidf_vectorizer.transform(test_book)
cos_similarity_tfidf = map(lambda x: cosine_similarity(book_tfidf, x), tfidf_rev)
output = list(cos_similarity_tfidf)
```

The resultant array is sorted to provide a list of cosine distances, with those closer to 1 being most similar. The array however, contained a number of results where the distance = 1 which is not unexpected as there are several books of the same title but with different publication dates.

```
[array([[0.0242549]]),
 array([[1.]]),
 array([[0.0065155]]),
 array([[0.01019116]]),
 array([[1.]]),
 array([[0.03134761]]),
 array([[0.0242549]]),
 array([[0.02232594]]),
 array([[0.0242549]]),
 array([[0.00850266]]),
 array([[1.]]),
 array([[1.]]),
 array([[0.00588555]]),
 array([[0.00445573]]),
 array([[0.]]),
 array([[0.]]),
 array([[1.]]),
```

*Sample from cosine distance matrix.*

## Testing the Model

A random book was selected

```
6    english kluwell my home reading yellow level s.
```

And passed to the model to calculate. The resultant output array was sorted to identify the closest matches and these values used to extract the matching books from the original dataset.

```
9780648237327 Kluwell My Home Reading Yellow Level
9781742152196 Targeting Maths
9781741352795 Writing Time: Queensland beginner's alphabet
9781922246165 ORIGO Stepping Stones Student Journal Year F
9781743305614 Strategies to Achieve Reading Success
9781741351828 IMaths
9781741353501 Sound Waves Foundation Student Book
9781741353358 Think Mentals Student Book 1
9780593098431 Are We There Yet?
9780648237327 Kluwell My Home Reading Yellow Level
9781922259011 NǏ Hăo
9780648237327 Kluwell My Home Reading Yellow Level
9781925487459 My Wellbeing Journal Years 1 And 2
9781925114300 Maths Mate 10
9780170416917 PM Handwriting for NSW K
9780224025720 Matilda BL
```

The resulting output mixed. While there are notably similar books within the top 15, there are anomalies, ‘Targeting maths’ for example would not appear to be a good fit.

## Understanding Accuracy

To assess accuracy a test model was created using a train/test dataset

The corpus of terms is converted to a feature array using MultilabelBinarizer(). In managing the transformation, the .split(‘ ‘) argument is added to ensure that each feature being counted is based upon a single word term and not per character.

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer
tfidf_vec = TfidfVectorizer(max_df=0.7, max_features=10000)
m_bin = MultiLabelBinarizer()
m_bin.fit(combined_features['cleaned'])
y = m_bin.transform(combined_features['cleaned'].str.split(' '))
```

The resultant data is split into train and test datasets in a 70/30 split.

```
xtrain, xval, ytrain, yval = train_test_split(combined_features['cleaned'],y,
                                                test_size=0.3, random_state=123)
```

and the split data transformed into weighted arrays using Tf-IDF.

```
xtrain_tfidf = tfidf_vec.fit_transform(xtrain)
xval_tfidf = tfidf_vec.transform(xval)
```

The an extract of the resultant array provides the following:

```
(0, 8025)     0.05786582276240948
(0, 5840)     0.07160303839686313
(0, 7554)     0.06750457415748229
(0, 193)      0.10453276575077902
(0, 2497)     0.07306157829995806
(0, 6958)     0.06460825874749544
(0, 1066)     0.08683325688204233
(0, 7856)     0.08844389581526353
(0, 7887)     0.06274498623204022
(0, 8012)     0.13334127978322036
```

The book recommender model is in practical terms a classifier, whereby, the transformed book data is classified across one of many categories. In this instance, a Logistic Regression is used which must be passed through sklearn's OnevRest Classifier to enable the multicategory data to be passed to the binary regression function.

```
lr = LogisticRegression()
clf = OneVsRestClassifier(lr)
clf.fit(xtrain_tfidf, ytrain)
```

The hyperparameter function of the OneVsRestClassifier were left as default, with the model limited to 100 iterations and set to assume a cold start.

The resulting model was used with the test dataset:

```
y_pred = clf.predict(xval_tfidf)
```

This test returned an F1 score as below.

```
f1_score(yval, y_pred, average='micro')
```

```
0.0962962962962963
```

This suggests a very low level of accuracy, given the F score is ranged from 0 (bad) to 1 (good). A score of less than 10% suggests the model is a poor predictor.

<u>Analysis</u>

The low accuracy score the an visible errors within the top 15 recommended books confirms the model is not performing correctly. My assumption is that the error lies in the test used for the content analysis from several aspects:

- The description is relatively short. A single sentence summary that may provide information by inference to an informed reader, but which is insufficient for a frequency model to derive enough data to correctly weight keywords. A solution may be to use the introductory chapters from each book where the likelihood of forming a theme is higher.
- Presence of low value words. Although the NLTK stopword dictionary was used, further analysis of the corpus is required to establish whether this went far enough in removing unnecessary vocabulary.

# 6. Recommender System #1: Collaboration based Recommender.

As outlined in the data wrangling and pre-processing section earlier, the collaboration-based model relies upon the ratings information extracted from Google. Books with higher ratings are weighted better against books without ratings or lower ratings.

The 'collaboration' of users in providing review or rating data improves model performance as the weight will tend towards higher review values.

In this instance, however, due to the relative sparsity of review data, the model will be built upon review counts rather than ratings.

Approach

The sample set created during pre-processing was split into test and training datasets on a 30/70 split.

```
df_train = train_set.sample(frac=0.7, random_state=123)
df_test = train_set.drop(df_train.index)
df_train.dropna(inplace=True)
df_test.dropna(inplace=True)
X_train, ydf_train = train_set.sample(frac=0.7, random_state=123)
df_test = train_set.drop(df_train.index)
```

Several algorithms were tested (see code attached) including K_nearest_Neighbours, SVC and Linear Regression from the sklearn package, with Linear Regression appearing the easier to implement given the numeric values within the rating data.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

The output created from the model was sorted and used to extract recommended books against a sample book from the results dataset.

```
10, 9780375842207, 2495.0, 4.5 'The Book Thief'
```

For testing care was taken to select a sample that had book ratings (2495 ratings at an average of 4.5):

```
9781741351811 0.0 IMaths
9780198603276 2.0 The Pocket Oxford Greek Dictionary
9781741353501 0.0 Sound Waves Foundation Student Book
9780190303488 0.0 Oxford Australian Integrated School Dictionary and Thesaurus
9781473221628 0.0 Dune: House Corrino
9780143505730 0.0 Natural Born Loser
9781925945065 0.0 ATAR Notes Text Guide: Burial Rites
9781741353389 0.0 Think Mentals Student Book 3
9781743305751 0.0 Strategies to Achieve Reading Success
9781488611261 0.0 Heinemann Physics 11 Enhanced
9781488611247 0.0 Heinemann Chemistry 1
9781741351811 0.0 IMaths
9780195508697 3.0 Biology
9780170393720 0.0 Tapis Volant 1 4th Edition Student Book
```

This output does not appear to be correct; the expectation was that the recommender would select as a minimum the higher rated books which it has failed to do.

Analysis of Performance

Performance metrics were extracted using sklearn.metrics and produced the following:

```
from sklearn.metrics import r2_score, mean_squared_error
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print(f'R-Squared',r2)
print(f'MSE', mse)
```

```
R-Squared -0.01818128485161008
MSE 145560.9671814672
```

The values produced confirmed that the model does not perform, a negative $R^2$ would suggest the model performs worse than average and the MSE is extremely high.

The recommender based upon user ratings appears to be non-functional in this instance. The reasoning behind this may be:

- The sparsity of reviews provides limited data against which to model.
- There is likely a structural issue with the model. Given the sparsity of data, the model was built using ratings from across the entire dataset. In reality this approach was incorrect at it would not differentiate between subject matter i.e. a high ranked English text could be recommended as equally to a similar ranked maths book. Creating User_ID categories based upon Subject and year group would overcome this issue, the challenge being that for niche subjects, the sparsity of reviews would make the model ineffective.

# 7. Conclusions & Recommendations

The assignment has comprised of two separate phases, data acquisition and augmentation and the development of two recommender systems from the developed data.

Data acquisition while technically straightforward, was challenged by the subject matter. The availability of additional content and metadata is less than would be expected from popular fiction, for instance, where established data sources are well populated.

In a commercial approach, being contractually engaged by major repositories such as Amazon would provide far greater depth of information and potentially access to book content.

Although several book data repositories were sampled, the freely available data sources were limited, sparce or required significant data wrangling and management to extract meaningful content.

The content recommender system struggled for content, the limited data and metadata obtained was insufficient. That said, a more finely filtered corpus may increase the frequency of key words and assist further. The content approach selected is appropriate and would, I expect, perform much better given a larger corpus of keywords.

An alternate approach may be to implement some form of tagging, allowing new metadata to be created and attached to existing books to close the distance in a cosine similarity matrix and provide better recommendations.

The collaboration engine, as performed, did not function. There was insufficient data to support the machine learning algorithm together with a structural issue in the approach namely the classification of subjects and year groups.

A separate search of Amazon did highlight greater numbers of reviews, and this would assist, but would be subject to commercial arrangements.