Supervised and Unsupervised Approaches to Content-Based Recommendation Systems

Ryan Brown

January 2021

1 Summary

People are spending an increasing amount of time consuming media during the COVID-19 pandemic, whether it is in the form of movies, shows, or books. About 80% of U.S. consumers now subscribe to at least one paid video streaming service[1], and the amount of time spent reading has nearly doubled for adults in the UK[2]. More content consumption means more content recommendations for users, usually accomplished through collaborative filtering, where a system uses the tastes of similar users and combines them to create a ranked list of recommendations. Platforms such as Netflix and Goodreads use collaborative filtering; however, this paper aims to explore the usage, potential, and limitations of content-based filtering for book recommendations. After testing unsupervised and supervised machine learning models on a corpus of books derived from books I have read or want to read, this paper shows that there is merit in looking further into a content-based approach for media recommendation.

2 Data Collection

The primary input for both the supervised and unsupervised models was book text. This is the case because content-based filtering, as its name suggests, analyzes the actual content of a media and uses various techniques to determine what other media are most similar to one the user liked. Goodreads book blurbs were also collected from `Goodreads.com` to diversify text input and model classifications.

2.1 Book Text Corpus

The book text corpus began with the first 52 books that I finished reading this year. For each book I indicated the author, genre, and whether I liked the book. Since I liked most of the books that I finished (46/52), I added 14 books that I started but did not finish (and therefore disliked) to balance the classes.

The genre distribution of the corpus at this point was remarkably undiverse, with 49 science fiction and 16 self-help books, as well as 1 mystery book. To help balance this, top titles from other distinctive genres were added as well, including 10 self-help books, 24 mystery books, and 26 history books from the top 100 books most shelved as that genre on Goodreads.

The .epub or .pdf version of each book was converted to a .txt file, where the first 900,000 characters of each file were read into the corpus, starting at the 50,000th character. The total length is limited to 900,000 to stay under spaCy's default max length for NLP parsing, and the first 50,000 characters are skipped to bypass any forewords or table of contents. 50,000 was chosen because it equates to 16 full pages of text, which was the largest amount of text seen before the book's content began.

	author	title	text	genre	started	finished	label
89	Brandon Sanderson	The Final Empire	than Kelsier, and he had a squarish\nface that...	Science Fiction	True	True	good
44	Stieg Larsson	The Girl with the Dragon Tattoo	care of the court costs.\nWith prudent budget...	Mystery	False	False	NaN
3	Ron Chernow	Alexander Hamilton	ry plume themselves on ancestry."17\nIn 1685, ...	History	False	False	NaN
95	N. K. Jemisin	The Obelisk Gate	the earth beneath your feet. No. Wrong direct...	Science Fiction	True	True	good
57	Iain M Banks	Consider Phlebas	e was to the enemy.\n'Well, whatever, Balveda,...	Science Fiction	True	False	bad

Figure 1: First five records in the book_data dataframe

[1] Deloitte. (2020, June). Digital Media Trends 14th Edition.
[2] Nielsen Book. (2020, May). Monthly Consumer Research Survey.

This corpus will be referred to as **book_data** for the rest of the paper.

2.2 Goodreads Corpus

To increase the sample size and diversify the number of genres, data was collected from Goodreads. 5 distinct genres were chosen: history, mystery, romance, science fiction, and self-help. For each of those genres, web crawling was used to collect the title, blurb, and top shelved genres for the 100 books most frequently shelved as that genre.

Books that appeared twice due to being in the top 100 for more than one genre were removed, as well as books that had one of the other four classified genres in their "top shelved genres" list. These books were removed because it's unclear what genre they should fall under, and the model is being trained to classify a single genre. The resulting corpus had 418 records.

	title	blurb	genre	goodreads_genres
101	Hyperion	On the world called Hyperion, beyond the law o...	Science Fiction	['Science Fiction', 'Fiction', 'Fantasy', 'Spa...
80	Along Came a Spider	A missing little girl named Maggie Rose	Mystery	['Mystery', 'Fiction', 'Thriller', 'Mystery > ...
320	A Court of Wings and Ruin	FEYRE WILL BRING VENGEANCE.\n\nShe has left th...	Romance	['Fantasy', 'Romance', 'Young Adult', 'New Adu...
340	The Guns of August	Historian and Pulitzer Prize-winning author Ba...	History	['History', 'Nonfiction', 'War', 'War > World ...
54	The Secret History	Under the influence of their charismatic class...	Mystery	['Fiction', 'Mystery', 'Contemporary', 'Litera...

Figure 2: First five records in the goodreads_data dataframe

This corpus will be referred to as **goodreads_data** for the rest of the paper.

3 Data Cleaning and Preprocessing

The text for both the books and the Goodreads blurbs required cleaning and preprocessing before the models were trained. The primary objective of these processes was to improve data quality and represent the data numerically so that scikit-learn models could use it as input. Data cleaning consisted of GloVe vocab matching, punctuation removal, and stopword removal; data preprocessing consisted of tokenization and vectorization.

3.1 GloVe

To initially clean the text of contractions, uncommon special characters, and typos, I looked at what portion of the text was recognized by GloVe (Global Vectors for Word Representation)[3]. GloVe is trained on over 2 million words of vocab, so words in the books not recognized by GloVe might need cleaning to improve data quality.

For the book text, the majority of frequent out-of-vocab words—words in the text that weren't recognized by GloVe—were words with punctuation at the beginning or end. In the case of both **book_data** and **goodreads_data**, because the punctuation provides useful context for how words were used, cleaning was not needed. Re.sub was used to clean all other out-of-vocab words with more than 50 occurrences, which were mostly contractions, typos, or unrecognized special characters.

Finally, to confirm that the text was effectively cleaned, I looked at what the GloVe coverage would be if words with punctuation attached to them had been cleaned.

```
Original Text:
GloVe Embeddings cover 24.96% of vocabulary and 84.64% of text in book_data

Post Cleaning:
GloVe Embeddings cover 29.82% of vocabulary and 88.27% of text in book_data

With Punctuation Cleaned:
GloVe Embeddings cover 78.56% of vocabulary and 99.16% of text in book_data
```

Figure 3: GloVe coverages pre- and post-cleaning for book_data

[3] https://nlp.stanford.edu/projects/glove/

For **goodreads_data**, the initial uncleaned text was already largely covered by GloVe, so coverage improvements from cleaning were minimal.

```
Original Text:
GloVe Embeddings cover 64.64% of vocabulary and 87.46% of text in goodreads_data

Post Cleaning:
GloVe Embeddings cover 65.88% of vocabulary and 88.88% of text in goodreads_data

With Punctuation Cleaned:
GloVe Embeddings cover 98.31% of vocabulary and 99.72% of text in goodreads_data
```

Figure 4: GloVe coverages pre- and post-cleaning for goodreads_data

3.2 Tokenization

spaCy's pretrained core_web_md[4] model powered the tokenizer, first turning the text into a spaCy document. Using the parts of speech tagging, spaCy can determine the lemma (e.g. walked walk) for each token. The tokenizer simultaneously lemmatized and tokenized the docs, while also considering the POS tag of each token.

There are two versions of the tokenizer. The first includes tokens for all parts of speech, while the second only uses the following subset of tags: adjectives, adverbs, nouns, proper nouns, and verbs. Tokens with these tags are the most significant in the sentence, so removing all other tokens should help to reduce noise in the data. Finally, the tokenizer removed stop words and punctuation before returning the list of tokens.

```
from spacy.lang.en.stop_words import STOP_WORDS
from string import punctuation
import en_core_web_md
nlp = en_core_web_md.load()
nlp.disable_pipes("ner", "parser")

desired_pos = ["ADJ", "ADV", "NOUN", "PROPN", "VERB"]

def pos_tokenizer(text):
    doc = nlp(text)
    tokens = [ word.lemma_.lower().strip() for word in doc if word.pos_ in desired_pos ]
    tokens = [ word for word in tokens if word not in STOP_WORDS and word not in
    punctuations ]

    return tokens
```

Listing 1: Code for pos_tokenizer

3.3 Vectorization

For each modeling task, two different vector representations were tested: bag-of-words (BoW) and term frequency inverse document frequency (TF-IDF). Each representation was tested using both tokenizers, for a total of four possible preprocessing combinations. Both representations were implemented using scikit-learn vectorizers.

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

bow_vector = CountVectorizer(tokenizer = tokenizer)
bow_vector_pos = CountVectorizer(tokenizer = pos_tokenizer)
tfidf_vector = TfidfVectorizer(tokenizer = tokenizer)
tfidf_vector_pos = TfidfVectorizer(tokenizer = pos_tokenizer)
```

Listing 2: Code for vectorizers

[4] https://spacy.io/models/en/

4 Unsupervised Modeling

Unsupervised learning is the golden standard for content-based filtering. The user gives the model a book that they liked (hereafter referred to as a "given book"), and the model compares the content of that book to the content of all the other books it has in its repository. The books that it finds to be the most similar (based on some similarity metric) are recommended to the user.

There are various ways to determine the similarity between two texts, four of which were tested in this paper (including the two vectorizers mentioned in Section 3.3):

1. Scikit-learn's CountVectorizer + cosine similarity
2. Scikit-learn's TfidfVectorizer + cosine similarity
3. spaCy's embedding-based text similarity system
4. Scikit-learn's KNearestNeighbors model

4.1 Evaluation

There's no concrete way to evaluate the accuracy of an unsupervised model. However, there are still ways to estimate how effective an unsupervised model is. In this case, there are a couple of features that the top recommendations should share with the given book, such as genre and series.

Because I can better validate the effectiveness of recommendations for books that I have read, all models were tested on two different datasets: the entire **book_data** corpus, and the subset of **book_data** where I started and finished the book, which will be referred to as **finished_books**. For each dataset, I examined the model's recommendations for every possible given book and awarded points based on the following factors:

book_data

- *Genre:* The recommendations should be the same genre as the given book, since all genres present are distinct. A linear amount from 0-1 was added to the model's overall score based on the number of the top 10 recommendations that are the same genre as the given book.

finished_books

- *Series:* If the given book is part of a series, the other entries in the series should be top recommendations, since the reader is likely to enjoy other books in the same series. A linear score from 0-1 was added to the model's overall point total based on the recommendation position of the other series entries. If the given book is not part of a series, a score of 0.5 was added.
- *Topic:* Topics exist within a genre, such as post-apocalyptic, space exploration, and quantum physics being topics within science fiction. Recommendations should ideally have topics similar to that of the given book. Topic is not a feature of the corpus, so I used discretion from having read the books to add a score from 0-1 to the model's overall point total based on the topic similarity of the recommendations.

Series was not included as a scoring factor for **book_data** because books that were not from the same series were intentionally chosen to diversify the dataset. Genre was not included as a scoring factor for **finished_books** as to not double count books present in both datasets.

There were 130 books in the entire corpus and 52 in the subset (with 16 of those 52 not in a series), meaning that the maximum overall point total for a model was (130 * 1) + (36 * 2) + (16 * 1.5) = 226.

This scoring system is not perfect, as it scores based on book characteristics that are easily accessible without a model and does not look at the real usage of a recommendation. However, it evenly evaluates the quality of the unsupervised recommendations across a large number of books. Thus, it is a proxy for determining which of the four unsupervised models is the best, as well as evaluating the general effectiveness of unsupervised-based recommendation systems as a whole.

4.2 CountVectorizer and cosine similarity

Cosine similarity is the most commonly used similarity function and was used in this paper because it measures how similar each text is irrespective of size. It does this by measuring the cosine of the angle between two vectors that are projected in a multi-dimensional space.

The similarity matrix was created using the following code:

```
from sklearn.metrics.pairwise import cosine_similarity

bow_vectorized = bow_vector_pos.fit_transform(book_data["text"])
bow_matrix = cosine_similarity(bow_vectorized, bow_vectorized)
```

Listing 3: Code to create similarity matrix

To get the recommendations for a particular book, the recommendation function indexed to the corresponding row of the matrix, created a dataframe, and sorted it in descending order by similarity. Below are sample recommendations for the given book *Foundation* by Isaac Asimov.

	title	genre	similarity
0	Foundation and Empire	Science Fiction	0.707825
1	Second Foundation	Science Fiction	0.675715
2	How Long 'til Black Future Month	Science Fiction	0.580531
3	The Worthing Saga	Science Fiction	0.546309
4	The Fifth Season	Science Fiction	0.542625
5	The Time Machine	Science Fiction	0.533045
6	The Stone Sky	Science Fiction	0.529593
7	Shades of Grey	Science Fiction	0.528265
8	Red Mars	Science Fiction	0.522145
9	The Obelisk Gate	Science Fiction	0.519826

Figure 5: Top 10 recommendations for *Foundation* by Isaac Asimov

The model scores for CountVectorizer + cosine similarity are as follows:

Table 1: Model scores for CountVectorizer + cosine similarity

Genre	Series	Topic	**Total**
88.40 / 130	43.25 / 44	28.20 / 52	**159.85 / 226**

4.3 TfidfVectorizer and cosine similarity

Following an identical process as in Section 4.2, the model scores for TfidfVectorizer + cosine similarity are as follows:

Table 2: Model scores for TfidfVectorizer + cosine similarity

Genre	Series	Topic	**Total**
64.80 / 130	40.50 / 44	23.00 / 52	**128.30 / 226**

4.4 K-Nearest Neighbors

An unsupervised K-Nearest Neighbors model was trained on the TF-IDF vectorized text using the following code:

```
from sklearn.neighbors import NearestNeighbors

KNN = NearestNeighbors(n_neighbors=11, p=2)
KNN.fit(tfidf_vectorized)
```

Listing 4: Code for K-Nearest Neighbors model

The neighbors count was set to 11 because the first neighbor of any point is the point itself, and the unsupervised scoring uses the top 10 recommendations for each given book. The model scores for K-Nearest Neighbors are as follows:

Table 3: Model scores for K-Nearest Neighbors

Genre	Series	Topic	**Total**
65.00 / 130	40.50 / 44	23.00 / 52	**128.50 / 226**

4.5 spaCy document similarity

When a text is represented as a spaCy document, its similarity to another spaCy document can be computed by taking the average of the word vectors that compose the document. To get the preprocessed book text as a spaCy doc, the text was converted twice: once in **pos_tokenizer**, and again to get the final list of documents. The similarity matrix was created using the following code:

```
texts = list(finished_books.text)
tokens_list = [pos_tokenizer(text) for text in texts]
doc_list = [nlp(' '.join(tokens)) for tokens in token_list]

spacy_matrix = []
for i in range(len(doc_list)):
    row = [doc_list[i].similarity(doc_list[j]) for j in range(len(doc_list))]
    spacy_matrix.append(row)
```

Listing 5: Code for spaCy document similarity matrix

The model scores for spaCy document similarity are as follows:

Table 4: Model scores for spaCy document similarity

Genre	Series	Topic	**Total**
110.00 / 130	43.75 / 44	35.60 / 52	**189.35 / 226**

4.6 Comparison

Below is a table comparing the scores of all four unsupervised models.

Table 5: Comparison of scores for all four models

Model	Genre	Series	Topic	**Total**
BoW	88.40 / 130	43.25 / 44	28.20 / 52	**159.85 / 226**
TF-IDF	64.80 / 130	40.50 / 44	23.00 / 52	**128.30 / 226**
KNN	65.00 / 130	40.50 / 44	23.00 / 52	**128.50 / 226**
spaCy	110.00 / 130	43.75 / 44	35.60 / 52	**189.35 / 226**

5 Supervised Modeling

Supervised modeling is not usually used in recommendation systems, because the recommendation system does not have the labels required to train a supervised machine learning model. However, in this paper I explore a unique case in that I have read a large number of books this year and know my taste for those books.

Scikit-learn classification models were trained to classify whether or not I liked a particular book based on its text. In production, this kind of supervised recommendation system functions differently than an unsupervised model; instead of being given a book that the reader liked as input and giving recommended books as output, it is given a book that the reader has not read yet as input and outputs a prediction as to whether or not they will like the book. This would ideally prevent them from wasting time starting books that they won't enjoy, as I did with 14 books this year.

The models were also trained to classify more concrete attributes about the books based on their text, such as genre and author. These models would not be used directly in production, but they demonstrate the potential of what supervised machine learning models can accomplish using media content.

For each task, 6 different scikit-learn machine learning models known to perform well with text classification were spot-checked:

```
from sklearn.linear_model import LogisticRegression, RidgeClassifier, SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC

models = {"Logistic Regression": LogisticRegression(solver="liblinear", max_iter=300),
          "SVM": SVC(C=1.0, kernel="linear", degree=3, gamma="auto"),
          "Naive Bayes": MultinomialNB(),
          "KNeighbors": KNeighborsClassifier(),
          "Ridge": RidgeClassifier(),
          "SGD": SGDClassifier(loss="hinge", penalty="l2", alpha=1e-3)}
```

Listing 6: Code for six scikit-learn supervised classifiers

5.1 Evaluation

The accuracy of each model was estimated using either 2-fold or 5-fold stratified cross-validation, depending on the size of each class. Stratified cross-validation was used because class distributions were not always even, and the small sample size meant that without stratification, a randomly selected fold could have a highly imbalanced class distribution that would skew results.

```
from sklearn.model_selection import StratifiedKFold, cross_val_score

num_folds = 5
cv = StratifiedKFold(num_folds)
accuracy = cross_val_score(estimator=pipe, X=X, y=y, cv=cv, scoring="accuracy")
```

Listing 7: Cross-validation code

The models were trained using all four preprocessing combinations (bag-of-words vs. TF-IDF, all PoS tags vs. subset of tags). I used the mean and standard deviation of accuracy across all folds for the best-performing preprocessing combination to determine the best model.

After determining the most effective model, it was evaluated against a baseline zero rule classifier (implemented using scikit-learn's DummyClassifier with a most_frequent strategy) using a 5 x 2-fold cross-validation paired t-test, proposed by Thomas Dietterich in 1998. This test helps determine whether the model performance is significant, as a model that does not perform better than the baseline is not useful. Accordingly, the relevant question is whether the trained models perform significantly better than the baseline, not whether their performance is significantly different in either direction. Thus, the prediction is that they will perform better than the baseline, and a one-tailed rather than a two-tailed t-test is appropriate. The MLxtend implementation of the paired t-test automatically gives the two-tailed p-value, so the resulting p-value was divided by two to get the one-tailed p-values for this paper.

An alpha of 0.1 as opposed to 0.05 was used, because the datasets used in this paper are small, so the standard errors will be larger—a larger alpha helps capture significant differences in light of these standard errors[5].

[5]Lavrakas, P. J. (2008). Encyclopedia of survey research methods (Vols. 1-0). Thousand Oaks, CA: Sage Publications, Inc. doi: 10.4135/9781412963947

5.2 Good vs. bad

5.2.1 Precision as a metric

Since good vs. bad is an imbalanced binary classification problem, just looking at accuracy does not tell the full story. Precision was also measured, with "good" being the positive class. This decision was made because the consequence of a false positive (incorrectly classifying a bad book as "good") and a false negative (incorrectly classifying a good book as "bad") are not the same.

To better understand this disparity, consider how the model would function in production. The user gives the model a book as input, and the model classifies that book as being either "good" or "bad". With a false negative, the user incorrectly misses out on a good book, but there are millions of other books to choose from. A false positive, on the other hand, results in the user wasting time reading a book that they don't like. The reader's time, rather than the number of potential books, is the limiting factor; thus, measuring precision with "good" as the positive class helps with understanding the model's performance.

5.2.2 Results

The bag-of-words representation gave superior performance for classifying good vs. bad books; the mean and standard deviation of accuracy and precision for each model are shown below.

	model	accuracy_mean	accuracy_std	precision_mean	precision_std
0	Logistic Regression	0.7571	0.0330	0.7623	0.0399
1	SVM	0.7571	0.0763	0.7747	0.0746
2	Naive Bayes	0.7879	0.1321	0.8953	0.0936
3	KNeighbors	0.7110	0.0922	0.7296	0.0584
4	Ridge	0.7571	0.0763	0.7747	0.0746
5	SGD	0.7110	0.1583	0.7620	0.0346

Figure 6: Model performance for good vs. bad book classification

Below is the confusion matrix for the most accurate and precise model, Naive Bayes

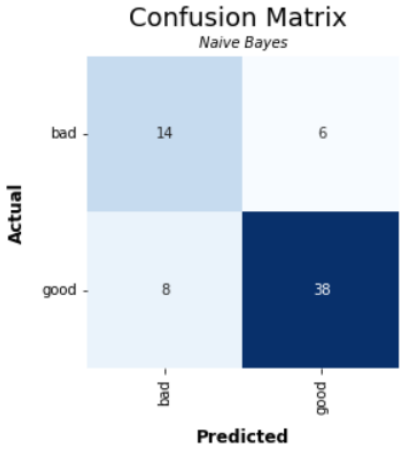

Figure 7: Confusion matrix for Naive Bayes on good vs. bad book classification

The one-tailed p-value for accuracy was 0.2436, which is greater than the alpha of 0.1. Thus, I failed to reject the null hypothesis that the Naive Bayes classifier and the baseline zero rule classifier have the same accuracy. However, the one-tailed p-value for precision was 0.0919, which is less than the alpha of 0.1. Thus, the null hypothesis is rejected in favor of the alternative hypothesis, which states that the Naive Bayes classifier is more precise than the baseline zero rule classifier.

5.3 Author

A subset of **started_books** was used for this task, only including books by authors that had multiple books present in the dataset. This subset, named **author_data**, had 42 records. Because books with authors that have as few as 2 books are included in **author_data**, 2-fold cross-validation was used for this task.

The bag-of-words representation gave superior performance for classifying the author of the book based on the book's text; the accuracy's mean and standard deviation for each model are shown below.

	model	accuracy_mean	accuracy_std
0	Logistic Regression	0.8310	0.0690
1	SVM	0.8060	0.0440
2	Naive Bayes	0.9762	0.0238
3	KNeighbors	0.1476	0.0524
4	Ridge	0.8548	0.0452
5	SGD	0.7810	0.0190

Figure 8: Model performance for author classification

Below is the confusion matrix for the most accurate model, Naive Bayes.

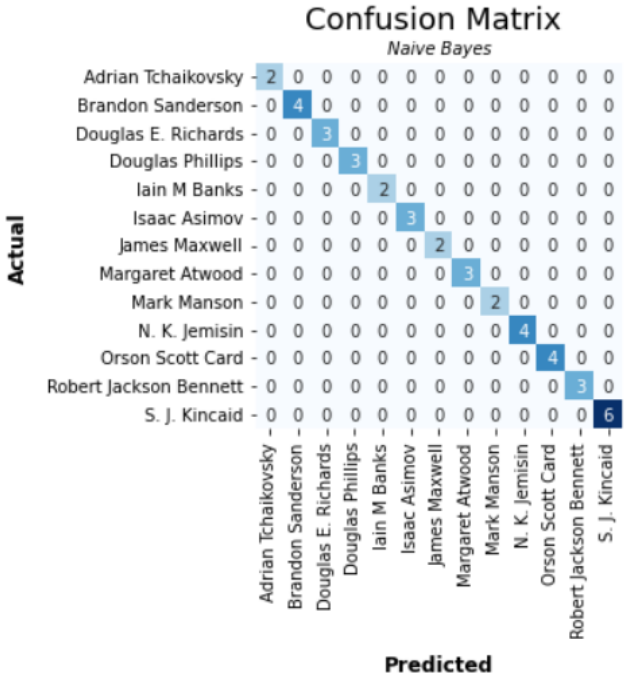

Figure 9: Confusion matrix for Naive Bayes on author classification

The one-tailed p-value for accuracy was 0.002, which is less than the alpha of 0.1. Thus, the null hypothesis is rejected in favor of the alternative hypothesis, which states that the Naive Bayes classifier is more accurate than the baseline zero rule classifier.

5.4 Genre (Goodreads)

The TF-IDF representation gave superior performance for classifying different genres based on the book's Goodreads blurb; the accuracy's mean and standard deviation for each model are shown below.

	model	accuracy_mean	accuracy_std
0	Logistic Regression	0.8540	0.0093
1	SVM	0.9020	0.0239
2	Naive Bayes	0.8205	0.0083
3	KNeighbors	0.8157	0.0212
4	Ridge	0.8996	0.0242
5	SGD	0.9116	0.0243

Figure 10: Model performance for Goodreads blurb genre classification

Below is the confusion matrix for the most accurate model, SGD (stochastic gradient descent).

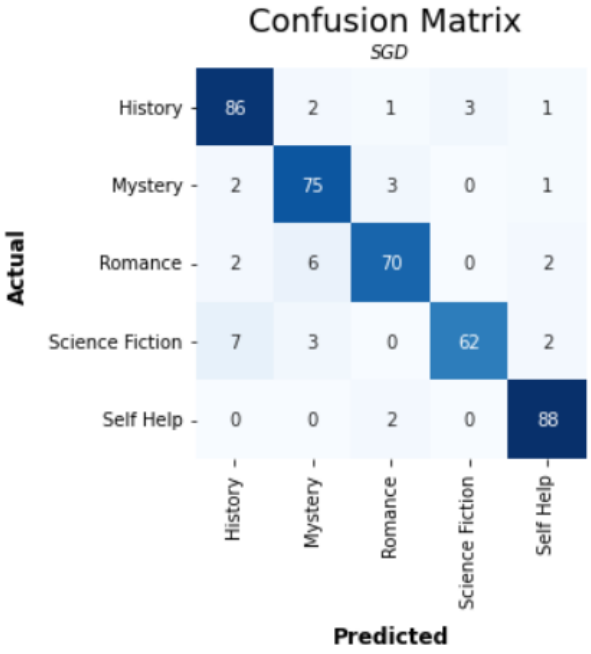

Figure 11: Confusion matrix for SGD on Goodreads blurb genre classification

The one-tailed p-value for accuracy was 0.000004, which is less than the alpha of 0.1 for this task. Thus, the null hypothesis is rejected in favor of the alternative hypothesis, which states that the SGD classifier is more accurate than the baseline zero rule classifier.

5.5 Genre (book text)

The bag-of-words representation gave superior performance for classifying different genres based on the book's text; the accuracy's mean and standard deviation for each model are shown below.

	model	accuracy_mean	accuracy_std
0	Logistic Regression	0.9000	0.0625
1	SVM	0.9385	0.0188
2	Naive Bayes	0.9308	0.0615
3	KNeighbors	0.6538	0.1264
4	Ridge	0.8846	0.0644
5	SGD	0.9308	0.0288

Figure 12: Model performance for book text genre classification

Below is the confusion matrix for the most accurate model, SVM (support vector machine).

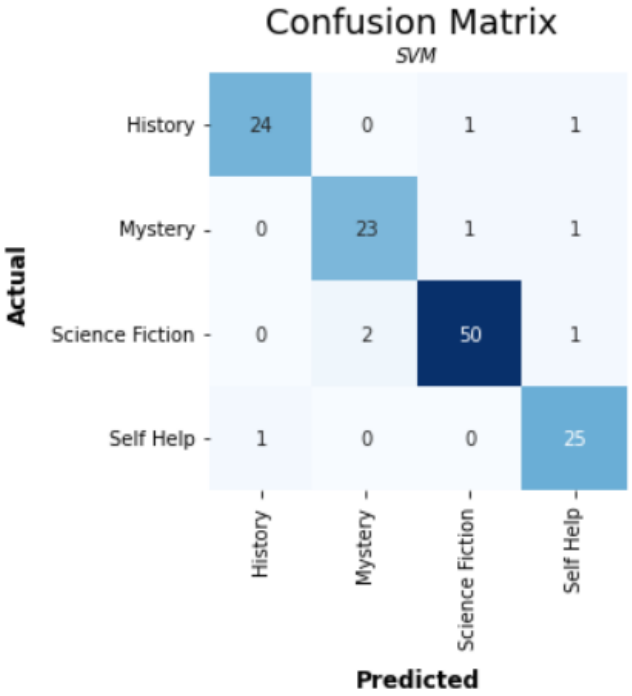

Figure 13: Confusion matrix for SGD on book text genre classification

The one-tailed p-value for accuracy was 0.00095, which is less than the alpha of 0.05. Thus, the null hypothesis is rejected in favor of the alternative hypothesis, which states that the SVM classifier is more accurate than the baseline zero rule classifier.

6 Discussion

6.1 Unsupervised Modeling

Of the four unsupervised models, the spaCy document similarity model performed the best, with a total score of 189.35 / 226. It had higher scores than the other three models in all three scoring categories, but the largest variation was in the Genre score. The spaCy model had a genre score percentage of 84.6% (110/130), an average of about 8.5 books out of the 10 recommended being the same genre as the given book. The BoW model was in the middle with a genre score percentage of 68% (88.4/130), or 7 books out of 10, and the TF-IDF and KNN models were at the bottom, with a genre score percentage of 50% (65/130), or 5 books out of 10. While recommending books of the same genre is a good proxy for the performance of these models, it is not very useful on its own. It would be very simple to filter the recommendations by genre of the given book, rendering genre discrimination nearly useless.

The TF-IDF and KNN models had almost identical scores in all three categories, likely due to the fact that the KNN model was trained on a TF-IDF vector representation of the text. A potential reason for the poor performance of these models is that the usage of inverse document frequency in their vectorization resulted in certain "super recommendations"; books that often showed up in the top 10, regardless of the given book. The super recommendations often did not match the genre or topic of the given book, bringing down the score of both of these models.

The series and topic scores had much less variability than genre. Regardless of the quality of their other recommendations, all four models had same-series books in their top 1-2 recommendations. This is not surprising, since same-series books are written by the same author in a near-identical prose and have the same proper nouns (characters, places, objects, etc). While the high scores in this category are promising, they are not very insightful. A reader can easily find the next books in their series without a recommendation system; thus, the recommendations beyond those same-series books are much more important.

The topic score saw some variation but not nearly as much as genre. spaCy outperformed the other models in this area, but was still far from a perfect score. A potential reason for this imperfection is that the topic scoring was done ad-hoc; I did not go through before the scoring to determine how many potential "similar topic" candidates each book had. Thus, the low scores across the board could be a result of suboptimal score scaling rather than poor model performance. Despite this, the spaCy model's score of 35.6 / 52 was impressive in what is the most significant score category. Unlike genre and series, the "topic" of a book is not clearly defined, so the utility of an unsupervised recommendation system for finding similar-topic books is quite high.

Overall, the superior performance of the spaCy model makes sense; the document similarity function considers the meaning of the words by comparing word vectors, rather than just looking at term frequencies like the other models. For most given books in **finished_books**, KNN and TF-IDF had identical recommendations, BoW had an improved version of those recommendations, and then spaCy had an entirely different set of recommendations that were much better. These recommendations sometimes surprised me, as I had not thought about the similarities between the two books on my own but could see them clearly now that the book was recommended to me.

The results of these unsupervised models are promising, especially considering the small scale this paper uses. Given a large dataset of books, a content-based recommendation system powered by unsupervised models could certainly be useful in practice. For example, Goodreads currently uses a collaborative filtering approach to book recommendations. I have found them to be hit or miss; sometimes the suggested books are similar and interesting, but sometimes they're pretty far off from the book I am currently reading. This dissimilarity is made possible by the fact that it's basing the recommendations solely on the tastes of other users. Combining their current system with an unsupervised, content-based approach could help improve recommendation consistency.

6.2 Supervised Modeling

All four supervised modeling tasks had performances that were significant according to the 5 x 2-fold cross validation paired t-tests. The BoW vector representation yielded better performances in three out of the four modeling tasks, while the TF-IDF representation was superior for the only task that used the shorter Goodreads blurbs as opposed to the longer book text. There's no clear reason for this distinction, but one factor to consider is that the book text contains frequent usage of proper nouns (character names, common locations) that are fictional and have low document frequency. TF-IDF therefore alters the weight of these words in the vector representation, whereas BoW does not. Additionally, the length and

diversity of the book text means that common words had a very high document frequency, while those same words could have a medium or low document frequency in the Goodreads data because the blurbs are so short.

Of the six models tested for each task, Naive Bayes had the highest performance on both the good vs. bad classification task and the author classification task. This makes sense, because Naive Bayes is known to have especially strong performance in NLP tasks compared to the other already strong models. Additionally, both of the aforementioned tasks used a smaller subset of the data, and Naive Bayes performs well with small amounts of data. This is the case because of its Naive nature, where every word in a sentence or text is assumed to be independent of the others.

The other model with notable performance was the support vector machine (SVM). It had the highest performance on the book text genre task and the second highest performance on the Goodreads genre task. It makes sense that SVM was able to effectively group and classify distinct texts based on their genre, since it finds a separating hyperplane to distinguish classes. Finally, the very poor performance of K-Nearest Neighbors is explained by its dependence on a TF-IDF vector representation as compared to BoW.

The most important of the four tasks in the context of a usable recommendation system is the classification of good vs. bad books, which predicts whether or not the reader will enjoy a book based on past books that they have read. While the accuracy for this task was not significant compared to the zero-rule, the precision was. Especially when considering the use of a model like this in production, where there would be thousands of books to draw from, the lower accuracy becomes negligible in the face of high precision.

Because supervised models require the user to input a book, though, a model trained for this task would not be very useful on its own. However, when combined with a traditional recommendation system, it becomes much more powerful. The traditional system creates the initial list of book recommendations based on a book that the user has read, then the supervised model pares that list down to books that it predicts the user will like. A platform like Goodreads has easy access to the books the user has read and their rating of those books, so implementation is feasible.

The other three tasks, while they show strong performance for the ability of a supervised model to discern between different aspects that define a book, are not useful in practice. As mentioned in Section 6.1, the genre and author of a book are clearly defined and easily accessed by a model, so there's no need for classification of these features. However, expanding model classification potential to other less accessible book traits, such as theme or topic, could aid in recommendation efforts. The hurdle for more niche traits would be labeling a large number of books, but this could be overcome with the usage of a smaller, manually-labeled subset and a semi-supervised model.

6.3 Overall Implications

There is a lot of potential in content-based recommendation systems that virtually no media platforms are taking advantage of. It's been shown in this paper that both unsupervised content-based recommendation systems and user-specific supervised models can be used to give effective user recommendations for books. While they need not replace collaborative filtering, using them in tandem with existing systems could help improve the quality and consistency of recommendations that users receive.

7 Next Steps

A key next step to validate the results discussed in this paper is to use a larger dataset for the supervised models that allows a test set to be held back. No test sets were used in this paper to help evaluate the models' performance because the datasets were small, so their results are subject to overfitting. To better understand supervised model potential in production-level recommendation systems, having a test set is crucial. Additionally, using nearly a million characters of text per book would likely be computationally inefficient for a large number of books. The vectorization of the text is expensive, and the vector representation for all books needs updating every time a book is added to account for new vocabulary. Smaller portions of the text that maintain the content for modeling purposes likely exist, and should be explored.

Finally, there are various extensions hinted at in this paper that begin to reap the benefits of alternative recommendation systems. These include combining unsupervised and supervised models as well as training semi-supervised models on less accessible traits of a book, such as topic or theme. Alternative recommendation approaches should also be explored in other media, such as movies and TV shows.