GOODREADS BOOK RECOMMENDATIONS

Mark Durrenberger

Springboard Data Science Intensive
-Capstone Project Report-

1) Introduction

The client here is Goodreads. Goodreads is a user-based review site where users can review/rate books they've read as well as perform other actions like add books to their "shelf" to keep track of books they would like to read later and track books they have read.

When it comes to rating books, there are two methods for doing so. A user can leave an integer rating - from 1 to 5 based on how well they liked the book. Or they can leave a full text review about how they thought of the book.

Unlike some ratings systems there are no "half stars" given. The book must be rated either 1, 2, 3, 4 or 5.

Goodreads recommends thinking of the ratings along these lines:
1 - "didn't like it"
2 - "just ok"
3 - "liked it"
4 - "really liked it"
5 - "it was amazing"

1.1) The Problem Statement

Most significantly though for our purposes is that Goodreads also has a recommendation engine for users. The workings of this recommendation system is proprietary, but I am setting out to create a version that would work well to both recommend books a user would love (i.e. provide a 5 star rating to), as well as for a user to see how they would like a potential book (i.e. what rating they would likely give to a specific book).

Therefore, the goal of this project is to create a ratings prediction model for Goodreads's readers.

2) My approach

Because the goal is to create a model that will not just recommend books a user would love (i.e. give a 5 star review to) but also to be helpful in predicting how likely a user would be to like a specific book (i.e. what the user would rate that book), my approach was to treat this as a multi-class classification problem.

The goal of this model is to predict the rating (class) which a user would give to a specific book.

2.1) The Raw Data

Goodreads book and review data was available through Kaggle from past competitions. However, this data was already preprocessed to some extent. In being cleaned up some decisions were already made as to how to process the data.

A key part of the data science process is the first step gathering raw data and decisions made around how to process it. I wanted to find more “pure” data.

Fortunately, I was able to find raw Goodreads review data through the work done by Mengting Wan and others (see citations at end of report). This data, and more about their work related to this dataset can be found here: https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home

Based on this raw data, one of my initial findings was that, as there were nearly 2.4 million different book, over 228 million user-book interactions, including over 104 million ratings given and nearly 16 million text reviews, my personal computer’s processing abilities would be a limitation on the amount of data I would be able to work with.

2.1.1) Limitations on Scope of Data

Some reduction in the scope or sample size of this project would be necessary. One of the steps taken by Wan and associates in collecting this data was to break it out into subsets classified by genre. For example, books and reviews of books in the historical genre was separated into their own datasets. Other genres broken out were comic books, fantasy, mystery, and others.

Due to my personal experience with comic books, and an assumption that comic books would be the most “isolated” genre, I decided to limit my initial model to one that would predict comic book ratings and used the data specific to comic books and reviews based on comics.

2.2) Data Cleaning & Wrangling

For this project I planned to use the list of books and the list of user reviews in the comics genre. I explored and cleaned up each set of data separately, before joining them on book_id (explained below).

2.2.1) Books dataset

The books data had the following columns:

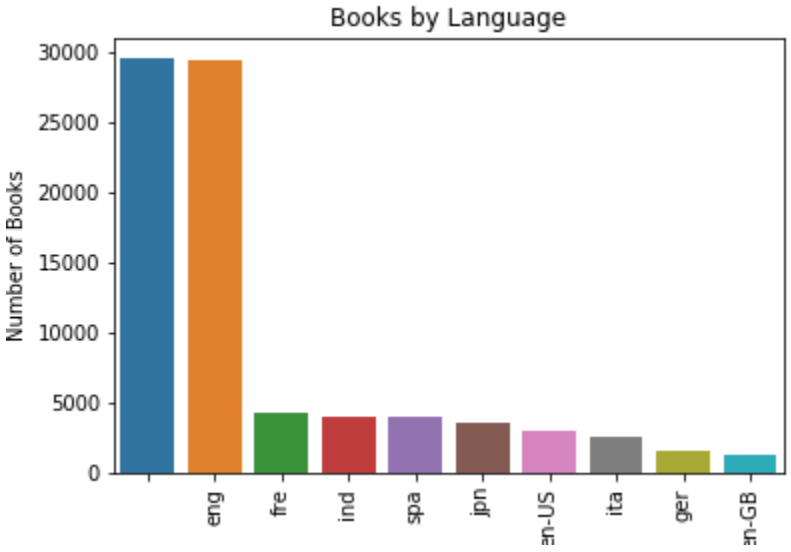
0	isbn	89411 non-null	object
1	text_reviews_count	89411 non-null	int64
2	series	89411 non-null	object
3	country_code	89411 non-null	object
4	language_code	89411 non-null	object
5	popular_shelves	89411 non-null	object

```
 6   asin                89411 non-null  object
 7   is_ebook            89411 non-null  object
 8   average_rating      89411 non-null  float64
 9   kindle_asin         89411 non-null  object
 10  similar_books       89411 non-null  object
 11  description         89411 non-null  object
 12  format              89411 non-null  object
 13  link                89411 non-null  object
 14  authors             89411 non-null  object
 15  publisher           89411 non-null  object
 16  num_pages           89411 non-null  object
 17  publication_day     89411 non-null  object
 18  isbn13              89411 non-null  object
 19  publication_month   89411 non-null  object
 20  edition_information 89411 non-null  object
 21  publication_year    89411 non-null  object
 22  url                 89411 non-null  object
 23  image_url           89411 non-null  object
 24  book_id             89411 non-null  int64
 25  ratings_count       89411 non-null  int64
 26  work_id             89411 non-null  int64
 27  title               89411 non-null  object
 28  title_without_series 89411 non-null  object
```

To better understand how to utilize the data for my purposes I examined each attribute of the dataset.

Language

The first that stood out was languages - there was a wide variety of languages included in the books. Say a few things about the visual below:



Being US-based myself (and, again, with somewhat limited processing power), I decided to focus on English language books. I noticed several different formats for “English” - such as ‘eng’, ‘en-US’, ‘en-GB’, ‘en-CA’ - keeping all of these left me with a data set of nearly 34,000 comic books.

Country Code

Now that I’ve kept only English-language comics, I was curious as to what the different values of ‘country_code’ column could be.

```
# Now that only english, what different countries are there?
print(books.country_code.value_counts())
```

```
US    33925
Name: country_code, dtype: int64
```

So all of these books are from the US. No point in keeping this column for our model then.

Attributes to Drop

After this reduction, upon further examination of the columns there were some that stood out immediately as extra information that could also be dropped from our data set:

- **ISBN/ASIN columns** - each of these numbers are simply different ways of identifying books. Since this dataset also has a ‘book_id’ column that uniquely identifies each book, we will drop these other identifying columns and use ‘book_id’ for this purpose.
- **Link/URL/Image_url** - these columns provide links to the relevant
- **Language_code** - since I kept only English-language comics, this column also becomes no longer necessary.
- **Country_code** - as mentioned above, these are all ‘US’ for the comics remaining in our data set. So this column can also be dropped.

Attributes for Further Exploration

- **Publication Day, Month, Year** - dropped day and month, but kept year.
- **Publisher** - This lists the publisher of the comic. It seems like it could potentially be relevant at first glance. If someone likes a lot of Marvel comics, for example, they would likely be more inclined to like another Marvel comic in the future.
    - But this information is probably encapsulated better at the title level - i.e. the fact that someone likes a Marvel Captain America comic could be gathered in more detail by the title "Captain America" rather than the publisher "Marvel"
    - *Decision: drop this column*
- **Edition information** - this column is mostly blank with nearly 90% of the columns having no value.
    - *Decision: drop this column*

- **Is_ebook -** This column is a Boolean of whether or not the comic in question is an electronic version. If it's not an ebook, there is no other information gained from this column.
    - However, there is also a 'format' column that also reflects whether a given book is an ebook, but also contains other information it it's not (i.e. weather its paperback, hardcover, etc).
    - Upon further inspection, there are nearly 150 different formats, with many being only a single instance (for example: "The Walking Dead - Single Issues #146 is a specified format).
    - There is also a large chunk (over 8,000) that are blank.
    - *Decision: drop both "is_ebook" and "format" columns*
- **Title Columns *("title" & "title_without_series")* -** These are nearly identical. In most cases the title is just the title of that comic and has no additional information.
    - *Decision: Drop title_without_series & just use title*
- **Authors -** This column contains dictionaries with an author_id number and then a string for their role in the work. Roles in this instances would be things like "author", "illustrator", "collaborator", etc.
    - I extracted the author names, but discarded the roles as a huge proportion of those were blank or overly specific (i.e. "penciler" or "inker"). Those don't seem like they would factor much into someone's preference for a comic.
    - *Decision: extract author name and keep; discard role information*
- **Series -** Based on the external documentation from Mengteng's website, this column refers to a separate data table that is outside the scope of this project.
    - *Decision: drop this column*
- **Popular Shelves -** This column refers to actions users can take with books. I will be using a separate dataset with user reviews/ratings to incorporate this type of information instead.
    - *Decision: drop this column*

2.2.2) Reviews dataset

The other dataset I will use is a set of user reviews and ratings. This dataset has some extraneous information around dates, other users. For the purposes of this project I'll keep the following:

- **User_id / Book_id -** These columns will be kept and used as a multi-level index for identification of each row of data. Ultimately this dataset will be merged with the books information on book_id.
- **Review_id -** This will be set as the index for now for ease of use.
- **Review_text -** This column contains the string text of the users review. This information will be used to create a sentiment feature in the preprocessing stage.
- **Rating -** This will be used as the target variable for our modelling purposes. See additional exploration of this column in EDA section (below).

3) Exploratory Data Analysis (EDA)

At this stage I performed more in depth exploratory data analysis.

3.1) Books Dataset

For the books dataset, the first step was to use *books.describe().T* to get an high-level synopsis of all the numeric columns and their ranges/spread.

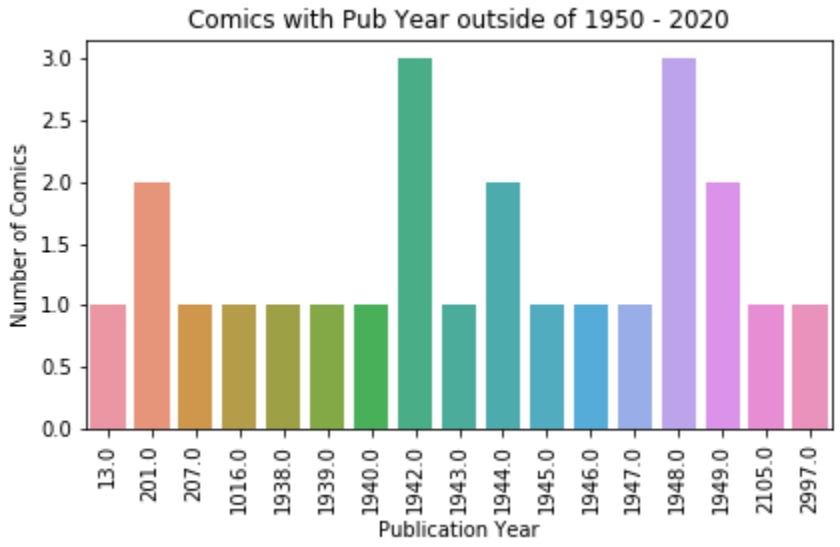
Copy of this output here:

	count	mean	std	min	25%	50%	75%	max
text_reviews_count	33925.0	38.330671	188.273209	0.0	3.00	8.00	23.00	10057.0
average_rating	33925.0	3.915122	0.414071	0.0	3.67	3.97	4.22	5.0
num_pages	33925.0	169.851113	143.721228	0.0	128.00	160.00	192.00	8124.0
publication_year	27521.0	2009.722903	25.023938	13.0	2008.00	2012.00	2015.00	2997.0
ratings_count	33925.0	796.750273	5095.773964	0.0	25.00	82.00	319.00	406669.0

3.1.1) Publication Year outliers

One of the first things that stood out from this table was some unusual extremes in the publication year. A minimum of 13 and maximum of 2997 don't make any sense. Further exploration of this column was necessary.

Exploring the ones that are outside of what I would expect.



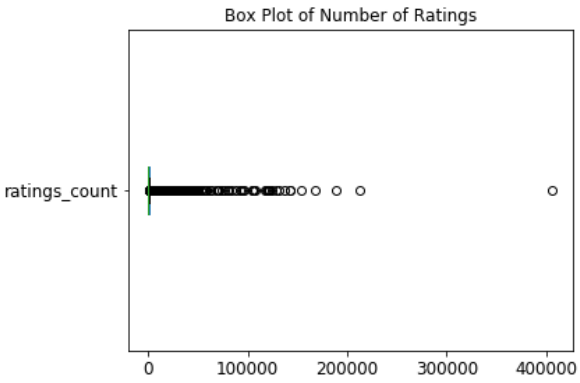
Since the non-conforming numbers were a manageable amount, I was able to verify them and correct by hand. For example -

The 2105 was a typo. The year of that comic was 2015. 2997 was also a typo. That was corrected to 1997. Replaced the other early ones with median year.

3.1.2) Ratings Count max

Another unusual number was the ratings_count max of 406,669. The median number of ratings for a comic was only 82, so it seems a bit extreme to have a book that was rated that many times. Considering out review data contains 542,338 reviews, that is a very large number of ratings. (Note - ratings can be given without a full review, so it does not necessarily follow that nearly 75% of reviewers have rated this book).

Looking at it visually, that max is a very extreme outlier.

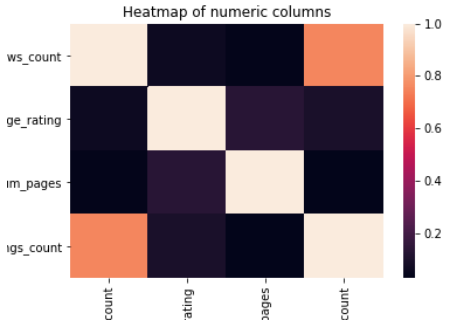


Upon further explanation, this comic is one that has received many awards. It was originally printed in the 80s and reprinted several times, with the most recent being in 2005. If there was one comic to receive that huge share of ratings, this one would seem like it.

The average rating of this comic is 4.35, which is above the 75% percentile of 4.22, so clearly it is a very popular book. This value seems like it belongs in the dataset.

3.1.3) Correlation

Another check is to highly correlated features. From exploring this it's clear that ratings_count and text_reviews_count are highly correlated.

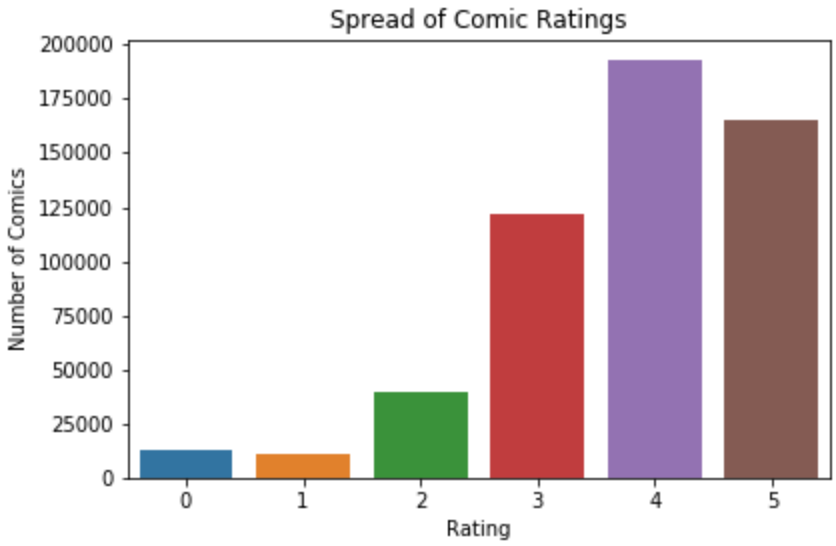


3.2) Reviews dataset:

Most of the columns from this dataset will be used for merging with the books dataset. In the case of the review_text column, that will be used to create a sentiment feature.

3.2.1) Ratings attribute – My Target Variable

The key column to explore of this dataset is the rating column. This will be our target variable so we want to get an idea of the spread and anything else relevant.



Clearly the ratings are skewed towards the high end. This frequently tends to be the case with ratings.

Rating	% of Overall Ratings*
0	2%
1	2%
2	7%
3	22%
4	35%
5	30%

**add to 98% due to rounding*

4) Feature Engineering & Preprocessing

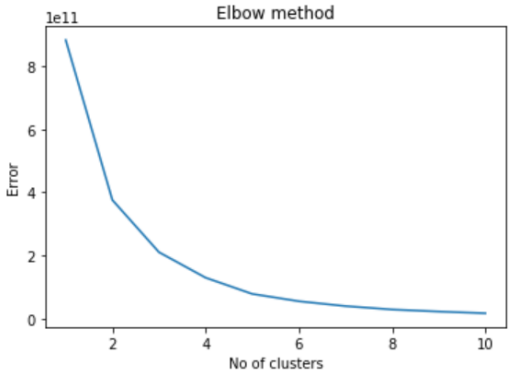
In addition to the existing features, I created some additional features. These were:

4.1) Book Features

4.1.1) Book clusters

I used the K-Means clustering unsupervised learning technique to group books into similar clusters and then assign those cluster numbers to each book as an additional feature of the dataset.

Using the elbow method, I determined that 5 clusters would be appropriate.



4.1.2 Average rating of most similar books

Another feature I thought would be useful was to find the books most similar to a specific book, and then find the average rating given to those books by all users. The thought here being that books that are quite similar to very highly-rated books may be more likely to be highly-rated themselves.

To create this feature required a few steps:

A) Cosine Similarity Matrix

The first step was to create a cosine similarity matrix for all of the books, based on each book's text description. This is done by using sklearn's TfidVecorizer to create an array of all the words in the description fields for each book.

From here, I used linear_kernal to create a pairwise similarity score for each book in our dataset. This gives us relative ratings of how similar a book is to each other book in our dataset, based on the description text.

B) Create similar books rankings

The next step was to create a function that would sort the similarity scores for all other books and return the top 3 most similar. Since the similarity scores in the above matrix were sorted by index, NOT by similarity score, the first step of this fucntion was to take all scores for the given book, sort them in descending order, and then return numbers 2, 3, and 4 in that ranking.

The reason for ignoring number 1 is that each book is most similar to itself, so the top ranking is simply the original book.

C) Find average rating for each of the three most-similar books

With the book_ids for the top 3 most similar books in hand, it was a simple matter to create a function to look up and return the average ratings for each of those books.

The final result for this feature is as follows:

most_similar_book_avg_rtg	2nd_similar_book_avg_rtg	3rd_similar_book_avg_rtg
4.28	4.28	4.32
4.03	4.46	3.98
4.00	4.21	4.50

4.2) Review Features

In addition to the two book features created above, two additional features for each review were incorporated as well.

4.2.1) Review Sentiment

For the text of each review, I utilized Natural Language Processing (NLP) to create a sentiment rating for the given review. To do this, I utilized the Python package nltk – specifically using the Vader SentimentIntensityAnalyzer.

The distribution of these ratings is as follows:

```
positive    385222
negative     91619
neutral      65174
```

This feature was then turned into a categorical variable and dummy variables created using one-hot encoding.

4.2.2) Average Rating of books in cluster by reader

The last feature I created was an average rating of books by cluster for each reader. This was done by merging the cluster column from the books dataset with the reviews dataset, based on the corresponding book_id column in common.

This resulted in the reviews dataset having a cluster attribute. I was able to then groupby user_id and book_cluster to find the mean rating assigned by each reader to the books in the given cluster.

4.3) Final Dataset & Scaling

The final step in preprocessing was to combine the books and reviews dataset into one. This was done by merging on the book_id column.

This dataset was then standarized using skelarn's StandardScaler and sorted into a testing and training set by using sklearn's train_test_split function.

5) Modelling

For modelling, I utilized sklearn's Python module as well. Specifically I created models using the K-Nearest Neighbors, Decision Tree and Random Forest algorithms. Each of these models was then tuned using sklearn's GridSearchCV.

After optimizing each of the models chosen, the best results came from the Random Forest model.

Here I will briefly examine each model individually before comparing their performance.

5.1) K-Nearest Neighbors

K-Nearest Neighbors (KNN) is one of the simplest machine learning algorithms, and therefore often makes a good place to start for modelling purposes. Training consists only of storing the dataset.

Hyperparameter Tuning

For KNN, the hyperparameters I chose to optimize for were leaf_size, n_neighbors and p, which specifices whether to use Manhatten distance (p=1) or Euclidean (p=2). From the set of parameters I searched, the best parameters were:

- Leaf_size = 3
- N_neighbors = 13
- P = 2 (Euclidean distance)

5.2) Decision Tree

The second algorithm chosen was a Decision Tree Classifier. A big advantage of decision trees in this situation is the interpretability of feature importance. In a business situation like this, it helps to be able to understand clearly which features were most important. Compare a Decision Tree to something like an SVM model, which can be extremely accurate but hard to translate into meaningful impacts on business decisions.

Hyperparameter Tuning

For the basic decision tree, the key parameters to tune are the criterion and max_depth. Without limiting max depth, a decision tree tends to build an overly complex model which very easily overfits. The optimal parameters from my tuning of the decision tree were:

- Criterion = ‘entropy’
- Max_depth = 9
- Min_samples_leaf = 13

5.3) Random Forest

The final algorithm I chose was the Random Forest Classifier. A Random Forest is an ensemble method, built out of a series of individual Decision Trees. Random Forest models help limit the tendency of an individual decision tree to overfit. By building a series of decision trees, each of which may overfit individually, we can then aggregate all of the trees to help minimize the overall overfitting.

Hyperparameter Tuning

For a random forest model, parameters to tune are similar to those of a decision tree with the additional parameter of n_estimators, which is simply how many individual trees will be fit within the random forest. The best parameters from my tuning were:

- N_estimators = 300
- Max_depth = 15
- Min_samples_leaf = 5

6) My findings

Comparing the performance of the models showed that the Random Forest model performed the best after optimization. The results of the various models are shown here:

K-Nearest Neighbors

Best model performance:

	precision	recall	f1-score	support
0	0.87	0.56	0.68	2275
1	0.50	0.11	0.18	1805
2	0.42	0.20	0.27	6702
3	0.44	0.46	0.45	20799
4	0.51	0.66	0.57	34421
5	0.74	0.60	0.67	30388
accuracy			0.55	96390
macro avg	0.58	0.43	0.47	96390
weighted avg	0.57	0.55	0.55	96390

Decision Tree

Best model performance:

	precision	recall	f1-score	support
0	0.86	0.59	0.70	2275
1	0.63	0.16	0.25	1805
2	0.51	0.20	0.28	6702
3	0.46	0.46	0.46	20799
4	0.51	0.68	0.59	34421
5	0.75	0.62	0.68	30388
accuracy			0.57	96390
macro avg	0.62	0.45	0.49	96390
weighted avg	0.59	0.57	0.56	96390

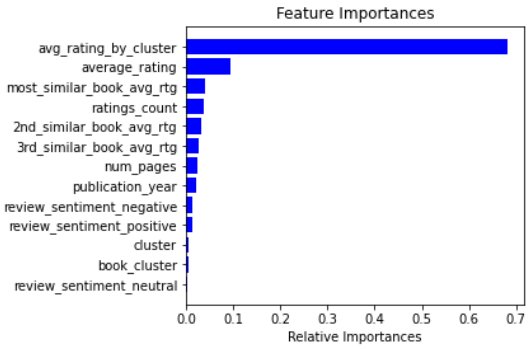
Random Forest

Best model performance:

	precision	recall	f1-score	support
0	0.86	0.58	0.70	2275
1	0.66	0.09	0.16	1805
2	0.52	0.17	0.26	6702
3	0.46	0.46	0.46	20799
4	0.51	0.72	0.60	34421
5	0.78	0.60	0.68	30388
accuracy			0.57	96390
macro avg	0.63	0.44	0.48	96390
weighted avg	0.60	0.57	0.56	96390

6.1) Client Recommendations

By far the most important predictive feature was the average rating that reader gave to other books in the same cluster as a given book. This means that we can really focus our efforts on improving and expanding upon our clustering methods. The more clusters we can develop, the more refined we can get with our targeting.



This is an excellent way for us to maximize the impact of our efforts.

6.2) Further Research Recommendations

This model gives us some guidance on how to focus on efforts, but could easily be expanded to provide more insights. Some ways of extending this research would be to:

- Expand this model across all genres of books
- Incorporate other user interactions in addition to the ratings and reviews used here
- Incorporating additional data with implicit features such as clicks, page views, etc.
- Expand to all languages
- With comics – incorporate visual features such as cover art

---

Citations

1) Mengting Wan and Julian McAuley. 2018. Item Recommendation on Mono- tonic Behavior Chains. In Twelfth ACM Conference on Recommender Systems (RecSys '18), October 2–7, 2018, Vancouver, BC, Canada. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3240323.3240369

2) Mengting Wan, Rishabh Misra, Ndapa Nakashole, Julian McAuley, "Fine-Grained Spoiler Detection from Large-Scale Review Corpora", in ACL'19.