Brigham Young University

Building A Crosswalk:

Matching Books Across Datasets

Econ 484

Professor Frandsen

By Seth Peterson, McCord Lethco, Isaac Smith, and Pōhai Chandler

Introduction:

In the initial stages of this paper, we sought to determine the causal effect of race on the New York Times' bestseller-status of a publication. In doing so, we began to construct a dataset of relevant covariates aside from race, namely, title of the publication, author, publisher, and publication date, as well as other interesting variables, such as a variety of popularity metrics. We gathered data from the New York Times Bestseller list[1], a Goodreads web-scraped dataset[2], which, aside from book characteristics, included metrics such as average rating and number of reviews, and the Seattle Public Library dataset[3] with all check-out data from over 15 years to accomplish this task. However, during our exploratory analysis of the dataset, we encountered a major roadblock to our project: the data were not matching correctly across the different datasets. Upon researching the issue, we uncovered a massive setback: a single book could have many ISBNs; so many such that they were nearly useless as a tool on which to merge datasets. Upon consultation, we decided to make this book-matching problem the focus of our project. In this paper, we present how we use machine learning to match a book in one dataset to another dataset, using the characteristics of title, author, publisher, and publication year.

A Deep Dive into Data Difficulties:

We encountered a couple of key obstacles that made data matching near impossible. A natural approach would be to match books based on ISBNs, International Standard Book Numbers, but we quickly learned that a unique ISBN is generated for each version of a book (paperback, hardback, 50th anniversary, illustrated, with foreword, etc.) For a popular book like Harry Potter and the Sorcerer's Stone, that means more than 50 distinct ISBNs, all representing the same creative work. We also considered an approach that concatenated a combination of column values but learned that the nearly infinite number of formatting differences made a case-by-case exception method *nigh impossible*. Ordering of first and last name, inclusion of middle initials, additional punctuation and articles are just a few of the dozens of formatting differences that we noted as we tried to match books across datasets. It was at this point that we determined that machine learning may provide us with the best book matching solution.

In order to tackle this problem, we create a machine learning book-matching algorithm, capable of bypassing ISBNs entirely, using a minimum of two of four identifying characteristics: author, title, publisher, and publication year. By utilizing this algorithm, we will be able to build a crosswalk between SPL dataset and the GoodReads dataset, thus generating new data. This new ID will act as the unifying identifier for each book across datasets. With unique ID generation, we are then able to match information from a range of book datasets with a variety of formatting differences.

Data:

This project utilizes data from three primary sources from the New York Times Best Seller List, Goodreads, and the Seattle Public Library. The New York best sellers list contains 5,000 different book titles, representing books in various genres such as fiction and nonfiction literature. While Goodreads obviously contains a massive cache of more than 3.5 million books, for our analysis we borrow a subset of 10,000 titles available on Kaggle[2]. The dataset tends to emphasize popular titles and highly rated books which increases our likelihood of finding matches between titles. Our GoodReads set includes unique information on book reviews and ratings as well as author profiles that clusters books according to author. Book sales data are closely guarded and very difficult to obtain. Thus, we use library checkout data provided by the Seattle Public Library as a proxy for book sales. Seattle Public Library is a major library district with several branches, who take seriously their accounting of checkouts. In our subset, we have roughly *50,000* *observations* with detailed information regarding checkout counts for all book formats. Each of the three datasets is comprehensive and contains important book characteristics including author name, book titles, publication dates, publishing house, and of course, ISBN identifier. A significant proportion of the work to develop our model centers on cleaning the data and converting string variables to numeric values. The raw data has many inconsistencies with regards to spellings, spacing, and formatting, making data cleaning extremely difficult. Our solution was to implement different string metric approaches, which we discuss in our paper.

Tables 6-8 illustrate the relationship between the observations in our data with the various string metrics. We analyze this relationship between these methods to understand the effects of matching and non-matching scenarios. Each table provides the summary statistics of these data combinations and the number of matches according to various percentiles, including the number of observations. We are able to see how successful each metric with our datasets. We can see, at the 75% percentile, we are able to maximize the number of matches and better utilize these metrics.

Methods:

In order to create a book matching algorithm, we first make a training set that is labeled with the four identifying characteristics we are interested in matching on. To accomplish this, we manually matched over 900 books from the New York Times bestseller list to a random sample of 10,000 books pulled from Goodreads books and the Seattle Public Library databases. Our hand-collected data resulted in 328 matches from the 984 NYT bestseller subset.

With 984 observations in our comparison set and 10,000 observations in our master sheet, we create a unique permutation for each book to book comparison in the two sets and end up with a set of 9,840,000 permutations. When performing these exact match classifications, we encounter a unique problem. If we simply used a naive estimator that simply answers "Not a match" for every book to book comparison it encounters, we would have a training accuracy of 99.96%. While this may seem like an impressive outcome, it is entirely useless if the estimator cannot find any matches in the master list. We need to reduce the number of "Not a match" results in our dataset.

A method we used to decrease the number of "Not a match" results is random selection. We randomly select half of the items from the dataset, bringing down the number of "Not a match" to 400,000. However, even with this reduction, the accuracy of the naive estimator remains at 99.92%, decreasing by only a marginal amount. This is not ideal because the accuracy rate is still showing inefficiencies in the matching process. The sheer size of the dataset and the complexity of the matching process make it clear that we need sophisticated algorithms and techniques to achieve high levels of accuracy.

Using our reduced training set, we clean and transform the data from string variables to numeric values. Publication year is already an integer, but every other characteristic requires transformation. Our ultimate goal is to identify whether two books are the same using a crosswalk model. Our model is fed a number of raw characteristics like book title, author name, publisher, and publication date. We convert these raw characteristics into a set of string similarity features that aim to measure comparability and similarity of text strings. We begin with seven separate string comparison algorithms to create our dataset features. The Levenshtein distance computed the number of single-character edits needed to get two strings to match. The Damerau-Levenshtein method adds a slight twist to the original Levenshtein by allowing for transpositions in addition to inserts, substitutions, and deletions. Jaro Similarity has a similar goal to the Levenshtein but rather than looking at simply the number of edits, Jaro considers string length, matching characters, and distance between characters. These features are fed into a calculation that generates a score between 0 and 1. The Jaro-Winkler adds upon the base Jaro algorithm by giving preference to strings that begin with a common prefix. As the name suggests, Longest Common Subsequence (LCS) Our ultimate analysis takes place within a dataset where each book is compared to every other book in the dataset and thus each observation is a book-to-book comparison rather than an individual book. Edit distance performs similarly to the Levenshtein by quantifying dissimilarity as the minimum number of changes needed to convert one string to another. Hamming distances are particularly optimized for two strings of the same length. However, to significant overlap in the string similarity metrics and increased computing complexity, we ultimately decided to discard the Jaro-Winkler and edit distance comparators in our feature set. We apply the remaining five similarity comparison algorithms to each matching book characteristic to generate our vector of features.

We use the generated string similarity features to train and test our model. Because we seek to solve a classification problem, we test the effectiveness of random forests, neural networks, and boosting. We use GridSearch for five-fold cross-validation to find the optimal parameters for each model type. Activation is the means by which input nodes are converted to output nodes. Layer size, as the name implies, represents the number of node layers in the neural network. Solver is a method for finding local minimums in the calculus of the computations. Alpha represents a penalty parameter for added complexity. Learning rate determines how much to change a model in response to error feedback. Tables 1-5 provide the selected parameters for the neural network for each model variant. Using those optimized parameters, we find that each of the methods performed similarly and produced a test set accuracy above 98%. However, the neural network method slightly edges out the competition and ultimately performs best.

After training the model using all features, we implement code to consider the robustness of our overall result. More specifically, we test the effectiveness and accuracy of our model if certain features are excluded. This robustness test helps us feel greater confidence in the real life usefulness and applicability of our tool. For example, how effective is the matching if we do not have author or publisher information? We retrain the neural network for performance with fewer inputs by systematically dropping a column each time and then recomputing our cross-validated parameters. We find that the results hold and do not differ significantly from our main result.

Results:

After running the training and test set, we find a training accuracy of 99.932% and a test accuracy of 99.727%. These results were very promising. With our algorithms tuned and trained, we are now ready to create our crosswalk between books. To do this, we'll create a master dataset, formed between our GoodReads and our Seattle Public Library dataset for the most comprehensive approach. Once again, we just use the top 10,000 checked-out books, found in the Seattle Public Library datasets. With around 11,000 observations in the GoodReads dataset, we reason that 110,000,000 permutations are too many to reasonably. Thus, we halve the GoodReads set, and compute 54,000,000 permutations. Upon completion, we then predict the match status, creating the crosswalk between ISBNs.

Using our completed crosswalk, we found that the model successfully identified *403 books matches*. As a reference point, when we merge the halved Goodreads dataset (5,400 obs) on the SPL checkouts dataset (10,000 obs) using only ISBNs, we yield just 86 matches. Therefore, our model is around *500% as effective* as raw merging on ISBN. To better understand our successful match rate, it is important to recognize that not all books have a corresponding pair in the other dataset. We do not know the true rate of overlap in the merge between our Goodreads and Seattle Public Library random sample subsets.

As shown in figure 1, we are able to visualize the neural network that our model creates. The circles represent matches made based on a simple ISBN merge, while the lines represent new connections our model was able to make. Our model clearly is superior in finding matching between the different datasets. This figure illustrates that ISBN numbers alone would not be an efficient identifier to conduct matching between the different datasets. The creation of new data proved essential in our machine learning approach to match data across datasets.

We recognize that our matches can include false positives and false negatives, affecting our matching estimations. Hence, we test for this specifically. We found, in our confusion matrix, 4 false positives and 1 false negative in the test set. Although we attempted our best effort to eliminate these errors, we were effectively able to minimize these occurrences by filtering our data, typically removing observations beyond the 50th characteristics, conditional on not being a match.

Using our confusion matrix (table 9), we can compute our false positive, false negative, true positive, and true positive rates. A simple calculation results in a false positive rate of just 0.0021, and a false negative rate of 0.025. Thus, in our crosswalk, given the confusion matrix of our neural network, we expect there to be just under one (.84) false positive, though we are likely leaving many false negatives. This is certainly an area to improve on, yet we reason that these results show our model to be effective in what we sought out to do, and thus, we consider our crosswalk to be a resounding success.

We also acknowledge that our model excels in robustness because of its ability to maintain significant results with the elimination of different columns. This means that even excluding certain parameters, the model still works more efficiently and effectively than an IBSN-matching approach. We are pleased with these results because the model was built around four criteria, but it can still work without including certain criteria. Essentially, we can still input books with missing observations, a recurring problem, and still match them successfully across datasets.

Conclusion:

In conclusion, our team is successful in creating a "Book Comparer" algorithm. With the new data we created, we are better able to adeptly match books across datasets. With this specific machine learning method, we are able to create a model to help match books to their respective ISBNs by utilizing a pre-trained Neural Network. This robust tool provides us with enhanced matching methods, while accommodating various combinations of available information, including title, author, publisher, and publication date. Our model will provide researchers a more effective tool to clean through book datasets and analyze them more efficiently.

While our current model primarily excels at classifying books within the scope of the New York Times and GoodReads data, we do recognize the limitations of the data that was available to us. To enhance its performance and applicability, we propose several future extensions, including training the model on a more diverse dataset and incorporating additional matching scenarios. By applying these changes to our model, it will create a more dynamic, constantly updated resource, with the potential to increase its matching potential.

By continuing to advance the capabilities of our tool, we envision its valuable application in the realm of record linking and other areas where accurate book identification is essential.. We recognize that common problems in economic research surrounds datasets inaccessibility. As we continue refining and expanding the "Book Comparer," our contribution will hopefully prove beneficial in creating and examining data in the future.

Tables and Figures

Table 1

Cross Validated Neural Network (With Title String Metrics)	
Hidden Layer Size	10 x 10
Activation	Identity
Solver	LBFGS
Alpha	0.0001
Learning Rate	Constant
Accuracy on Test Set	.99318

Table 2

Cross Validated Neural Network (With Title and Author String Metrics)	
Hidden Layer Size	5 x 5
Activation	Logistic
Solver	LBFGS
Alpha	0.0001
Learning Rate	Constant
Accuracy on Test Set	.99523

Table 3

Cross Validated Neural Network (With Title and Publisher String Metrics)	
Hidden Layer Size	5 x 5
Activation	Logistic
Solver	LBFGS
Alpha	0.0001
Learning Rate	Constant
Accuracy on Test Set	.99795

Table 4

Cross Validated Neural Network (With Publisher and Author String Metrics)	
Hidden Layer Size	5 x 5
Activation	Logistic
Solver	LBFGS
Alpha	0.0001
Learning Rate	Invscaling
Accuracy on Test Set	. 98499

Table 5

Cross Validated Neural Network (With All String Metrics)	
Hidden Layer Size	5 x 5
Activation	Logistic
Solver	LBFGS
Alpha	0.0001
Learning Rate	Adaptive
Accuracy on Test Set	.99727

String Metrics Conditional Upon Match[1] (Table 6)

	Hamming Metric Title	Levenshtein Metric Title	Jaro Metric Title	Hamming Metric Publisher	Levenshtein Metric Publisher	Jaro Metric Publisher	Hamming Metric Author	Jaro Metric Author	Damerau Metric Author
Count	328	328	328	328	328	328	328	328	328
Mean	21.557	18.493	0.852	10.155	8.719	0.737	2.484	0.946	1.801
Standard Deviation	31.548	27.867	0.149	9.774	8.356	0.223	5.457	0.126	4.167
Min	0	0	0.405	0	0	0	0	0.435	0
25 Percentile	0	0	0.763	0	0	0.529	0	1	0
50 Percentile	12	12	0.854	9	7	0.788	0	1	0
75 Percentile	25.5	21	1	14	12	1	0	1	0
Max	151	138	1	66	53	1	41	1	34

String Metrics Conditional Upon Not Match[2] (Table 7)

	Hamming Metric Title	Levenshtein Metric Title	Jaro Metric Title	Hamming Metric Publisher	Levenshtein Metric Publisher	Jaro Metric Publisher	Hamming Metric Author	Jaro Metric Author	Damerau Metric Author
Count	400000	400000	400000	400000	400000	400000	400000	400000	400000
Mean	40.460	35.329	0.510	15.912	13.841	0.490	13.315	0.480	11.899
Standard Deviation	29.074	26.157	0.082	7.261	6.255	0.126	3.75312	0.112	3.181
Min	0	0	0	0	0	0	0	0	0
25 Percentile	21	19	0.467	11	10	0.438	11	0.435	10
50 Percentile	32	28	0.517	14	13	0.500	13	0.490	12
75 Percentile	48	41	0.561	19	17	0.553	15	0.543	13
Max	169	163	1	90	86	1	45	1	45

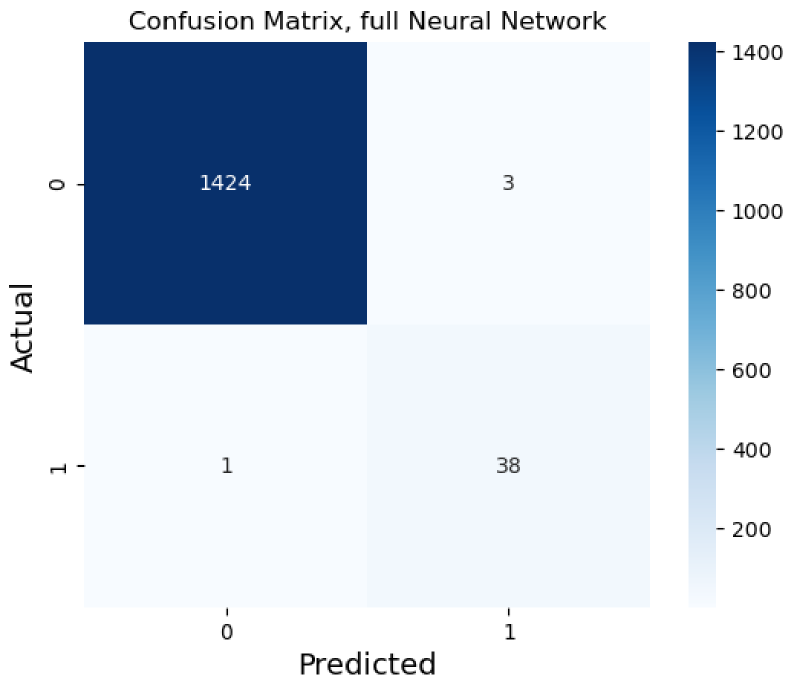
[1] In order to decrease the training set, which would have slowed down the machine learning training and decreased accuracy, we considered the how different the individual string metrics could be and still be a match. This table shows, conditional that the book is a match, summary statistics for string metrics for the title of the book. We often used a little beyond the 75th percentile to keep the training set small enough to be accurate.

[2] This table serves as a comparison to the previous table. It shows summary statistics of string metrics for the title, conditional upon the book pair not matching. For example, the max of the Hamming Metric Title if the book is a match is 151, while for not being a match is 169.

Filtered Training Set[3] (Table 8)

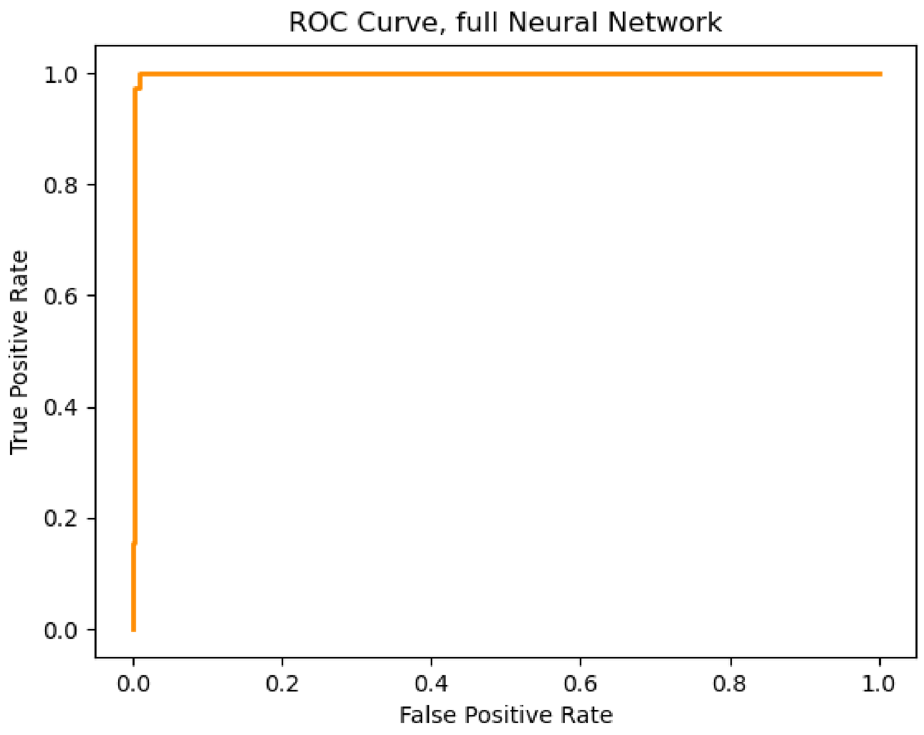
	Book Matches	Publish Year Delta	Levenshtein Metric Author	Damerau Metric Author	Hamming Metric Author	Jaro Metric Author	LCS Seq Length Author	Edit Distance Author	Levenshtein Metric Title
Count	5863	5863	5863	5863	5863	5863	5863	5863	5863
Mean	0.022	8.034	7.782	7.757	8.903	0.591	3.991	7.782	23.000
Standard Deviation	0.148	7.535	1.751	1.741	2.075	0.094	1.850	1.751	8.439
Min	0	0	0	0	0	0.5	1	0	0
25 Percentile	0	3	7	7	8	0.533	3	7	17
50 Percentile	0	6	8	8	9	0.567	4	8	23
75 Percentile	0	11	9	9	10	0.614	4	9	28
Max	1	90	10	9	11	1	17	10	53

Confusion Matrix[4] (Table 9)



[3] These are some of the summary statistics of the filtered, cleaned data that was used to train the model. The filters were derived from summary statistics in the String Metrics Conditional Upon Match table.
[4] The confusion matrix is computed for the Neural Network, using the training parameters described in Table 5.

ROC Curve of Full Neural Network[5] (Figure 1)



[5] The ROC Curve is computed for the Neural Network, whose tuning parameters are found in Table 5.

Figure 2
Network Graph of Crosswalk Data,
showing all new connections