"THE THREE MUSKETEERS": THE INSPIRING STORY OF HOW THREE FRIENDS, AGAINST ALL ODDS, BANDED TOGETHER TO BUILD A CONTENT-BASED RECOMMENDER SYSTEM FOR BOOKS

By: James LeDoux, Aniket Saoji, Drew Hoo

"There is no book so bad...that it does not have something good in it." - Don Quixote de la Mancha

Abstract

The problem we wanted to solve was the standard problem with collaborative-filtering-based recommendation systems: that exaggerated social effects have the tendency to bury books that are lacking in social clout, causing a disproportionate representation of well-publicized books and a poor performance in facilitating new book discovery. Building a recommendation system based on the content of the book is meaningful because it allows for an effective cold-start for the recommender system, and is also solely focused on the literary and semantic features that make a book enjoyable to its reader. Moreover, a person's attraction to a book is unique. It stems not from social suggestions but from content attributes including complexity, diction, subject matter and style. For this reason, our goal was straightforward: let's build a book recommendation system based on content and nothing else.

Our Data:

But alas, with such a lofty goal came many unexpected twists, turns, ups and downs. The first of which was the massive amounts of data we were sifting through and the second of which was the fact that our data was absolutely, completely unsupervised. We had a basic notion of the data we wanted to use: the 55,000 free English e-books comprising of 40 compressed gigabytes from Project Gutenberg (PG), which is the oldest and largest online library of full literary texts, containing almost every book in the public domain, as well as various manuals, guides, dictionaries, encyclopedias, and miscellaneous texts.

Initially we considered downloading a mirror of the entire PG corpus, but this idea was discarded due to its excessively large size (over 650gb due to its many duplicate e-books stored in differing formats). After finding a small 1.21gb set of 3,036 books from the University of Michigan, we decided to utilize this data to test our code locally, and then stream data directly from PG to AWS.

In the end, the approach we pursued was successful, resulting in the attainment of 23,795 e-books, representing roughly 44% of PG's English language library. We manually distributed our computing workload for pulling and unzipping these files from the directory using the Fulton Distributed File System™ (patent pending), in which we partitioned the data four ways and had the computers in the computer science lab run a wget script overnight. We decided to end our script prematurely at 24,000 books out of courtesy for the other students hoping to use the lab, and also due to the joint decision that the 9gb of plain text data this represented would be sufficiently large to build a recommender system upon. Finally, we applied machine learning algorithms on our data to provide content-based literary recommendations.

Preprocessing and Feature Extraction

With our large text library intact, we then turned to preprocessing and feature extraction in order to clean the data and create features that would later be used for approximating a user's literary taste profile. The extracted features include: author, total words, average words per sentence, sentiment analysis, relative punctuation frequencies, and topic clusters obtained by running k-means clustering on books' TF-IDF scores.

Our pre-processing approach was computationally expensive due to the need to pass through books on a character-by-character basis in order to extract punctuation data. It got the job done, however, again thanks to the Fulton Distributed File System (FDFS). In total, the process took three separate operations. In the first run through, we combed over the first block of lines of each book's metadata to locate the book's title and author. After this initial shallow pass, the first full run through then parsed through each character of each book. In this pass, we collected punctuation data, average sentence sizes, and average word length. In the second full run through, we then used Natural Language Toolkit's Vader sentiment analyzer to conduct sentiment analysis. Vader ran through each sentence of each book and outputted the book's positive, negative, neutral, and compound sentiment scores. The third pass used Spark's mllib TF-IDF library and generated a TF-IDF score for each word in each book. Using these scores, we would later run k-means clustering to assign content clusters to the books. Finally, we modified our author and cluster membership features to become binary variables in order to use them in our machine learning algorithms. Our feature set was as follows:

Title, Author, Word Count, Avg. Sentence Length, !, #, “”, &, $, &, (), +, *, /, ., ;. :, @,], negative score, neutral score, positive score, compound score, cluster ID where k = 50, cluster ID where k = 155, cluster ID where k = 480

To complete the feature manipulation stage, we then standardized our features to [0,1] scales. The purpose of this was to prevent any particular feature from dominating our results during the prediction stage. This would have been an issue because some of our features, such as word count, had high variances, which would have dominated the results of algorithms such as k-nearest neighbors which rely on measures of euclidean distance.

Machine Learning

We applied machine learning methodologies for both feature extraction and for serving recommendations. The algorithms we employ are k-means clustering on TF-IDF scores, k-nearest neighbors for recommending books similar to those enjoyed by a reader, and ordinary least squares regression to predict a user's ratings for books based on a given user profile.

K-Means

K-Means is a clustering algorithm that takes n-dimensional data and assigns its observations categorical values. This unsupervised learning algorithm lends itself particularly well to problems of recommendation, since it can serve two important purposes: first, that its topic-based outputs can serve as crude recommendations in and of themselves, and second, that their document categorizations are crucial features for any broader-reaching recommendation algorithm that aims to capture content as a feature.

At a high level, k-means randomly initializes k starting points, which are n-dimensional coordinate sets, and then passes over the n-dimensional data, assigning each observation to the cluster centroid with the closest euclidean distance. The cluster centroids then update by moving to the mean of the points assigned to them, and the process repeats until the algorithm converges and the centroids no longer move significant amounts in new iterations.

To capture the meaning of our texts, we sought to find a way to apply k-means to the words themselves that make up our books. We first considered using a bag-of-words approach, using sparse vectors of word-counts as the features to cluster upon, but then realized that this data would be severely skewed by the length of the individual texts. While looking into normalizing this by text length, we came across the TF-IDF measurement, which appeared to be a much better solution, so this is what we applied.

TF-IDF stands for term frequency x inverse document frequency. It measures the importance of a word to an individual document, relative to its importance in a larger corpus of documents. The result is a measurement that rewards words that are distinctive to a text by being disproportionately important to a particular document in the corpus, making this metric useful for seeking out unique content categories. The exact calculation is:

(occurrences of a term in a document / total number of terms in the document) x ln(total number of documents / number of documents containing this term)

Or, more simply, TF x IDF.

K-Nearest-Neighbors

K-nearest neighbors is a simple algorithm that proves intuitive and useful for recommendation tasks. The algorithm takes a set of feature vectors (one for each book), as well as a target vector (the book we are seeking to find similar recommendations to), and then calculates the euclidean distance between the target vector and each of the others in the data set. These distances are then sorted, and the algorithm returns the k books with the shortest distances to the target observation. The result is a set of recommended texts that are most similar to a selected text according to the features we have extracted. A weakness of this algorithm is that it weighs each feature equally, while some may, in reality, be more important than others. Its quickness of computation and intuitive application to recommendation, however, makes it a useful tool for our task.

Ordinary Least Squares

The last recommendation algorithm we applied was ordinary least squares regression. This algorithm required us to obtain user ratings data, since OLS is a supervised algorithm that requires target values. In our application of OLS, we ask a user for his rating of 20 books on a 0-10 scale, and then apply the algorithm to the feature vectors in order to predict how the user would then rate all of the 23,000+ other texts in our data. The highest of these predicted ratings are then served as our highest recommendations.

OLS takes a user's ratings as target values, and then calculates the weights attributed to the data's features that best approximate the given test data. This is accomplished by minimizing the sum of squared residuals, where

$$\widehat{p} = \beta_0 + \sum \beta_j x_{ij}$$

and

$$SSR = \sum_{i=1}^{n} (rating_i - \widehat{p}_i)^2 .$$

The benefit of this model is that it attempts to learn which features affect user ratings, and to what extent they do. This presents a potential solution to the problem of not knowing which features are most important in the data. To test these results, we hid a small subset of the user's ratings from the algorithm to compare the predicted ratings to those observed in reality.

Processing Challenges

Additional to the time-expensiveness of the feature extraction work that required slow passes over the full data set, some of our algorithms required greater processing power than our local machines and the Fulton Distributed File System could provide. For this reason, we turned to Amazon Web Services' S3 storage system and Elastic Mapreduce service to compute our TF-IDF calculation and K-Means algorithm at scale. This required our code to be written in a distributed, memory-efficient manner since the feature vectors of TFIDF scores were far too large for any single node to handle in their dense format, even on AWS.

We were able to run k-means on a cluster of one master and three core nodes. With Amazon's superior processing power and our Spark-distributed code, the TFIDF and k-means calculations were able to run to completion in roughly two hours.

Individual Contributions

In order to maintain revision control while having three people simultaneously working on the project, we used github and had our entire repository online. It was extremely simple to see when somebody committed changes and to pull the updated file every single iteration.

On top of that, we utilized the idea of pair programming at its finest, and cranked out a lot of the code by looking over each other's shoulders. That being said, each of us were responsible for a few specific parts. Aniket was responsible for many of the preprocessing tasks. He wrote the code to parse through the texts and collect different punctuation, different sentence sizes, the word count and the title of the book. Drew, on the other hand, helped to put the "Big" in "Big Data." Drew found a better way to download the data from PG using their robots URL, and helped us transition from using the Michigan dataset (3,036 books) to approximately half of the english PG corpus (23,795). He also researched how to use TF-IDF and co-wrote the TF-IDF / Kmeans / KNN clustering scripts with James. On top of helping Aniket and Drew collect the preprocessing data, James wrote three different prediction algorithms to figure out which was the best. He wrote a simple KNN algorithm, OLS algorithm, and a KMeans algorithm to predict books the user would enjoy, and also helped to run the KMeans algorithm on AWS.

Results

K Means on TFIDF Clusters

A naive way to provide recommendations would be to recommend solely based on k-means clustering outputs. This would ignore our stylistic features collected during preprocessing, but be a content-based recommendation nonetheless. To demonstrate how this form of recommendation would work, below we show examples from our three clustering tiers. We ran k-means three times, with k values of 50, 155, and 480, resulting in content clusters ranging from general to highly specific groupings. We found interesting patterns ranging from genre-level clusterings in the small k cluster, to highly specific groupings in the large k cluster. A recommendation based solely on k-means would take a book that the user enjoyed as input, and return a selection of additional books from that book's cluster as recommendations. Below, you can see how some of these recommendations might look using a small, medium, and large value of k.

Examples:

Small k (most general): 50 clusters

Cluster ID	**Apparent Category**	**Titles in Cluster**
32	Old English Plays	A Collection Of Old English Plays, Vol. IV, The Works of John Dryden, Vol. II, A Select Collection of Old English Plays, Vol. VII (4th edition), A Select Collection of Old English Plays, Vol. VIII (4th edition), A Select Collection of Old English Plays, Vol. IX
30	History	The Roman Poets of the Augustan Age: Virgil, The French Revolution, Benjamin Franklin, History of the Warfare of Science with Theology in Christendom, The German Classics of The Nineteenth and Twentieth Centuries, Mahabharata of Krishna-Dwaipayana Vyasa Bk. 3 Pt. 1
24	Religion (general)	The Divine Comedy, The Psalms of David, Paradise Lost, Prayers and Meditations, The Odyssey of Homer, The Aeneid

Medium K: 155 clusters

Cluster ID	**Apparent Category**	**Titles in Cluster**
2	Country Westerns	The Texan Scouts, The Prospector, The American Baron, Wild Western Scenes, Sappers and Miners
99	War Stories	The Memoirs of General Ulysses S. Grant, The Song of Roland, Real Soldiers of Fortune, Our Campaign around Gettysburg
152	Sexual Psychology	Studies in the Psychology of Sex Volumes I - IV, The Sexual Life of the Child, The Sexual Question, Studies in the Psychology of Sex

Large K (most specific): 480 clusters

Cluster ID	**Apparent Category**	**Titles in Cluster**
57	Arctic Exploration	South!, Glaciers in the Alps, Farthest North, The North Pole, Ice-Caves of France and Switzerland
247	Bibles (specific)	The New Testament, Bible Readings for the Home Circle, The New Testament of our Lord and Savior Jesus Christ, A Translation of the New Testament from the original Greek, An Examination of the Testimony of the Four Evangelists, Companion to the Bible, Jesus the Christ
314	Chemistry Textbooks	Elements of Chemistry, Nitro-Explosives: A Practical Treatise, The Handbook of Soap Manufacture, The Chemistry of Plant Life, The Phase Rule and Its Applications, An Introductory Course of Quantitative Chemical Analysis, Soap-Making Manual

As can be seen in the above selections from our k-means output, this algorithm appears to have worked quite well. Whether you were interested in recommending a book within the same genre (small k) or the same precise subject matter (large k) as a given target book, the k-means outputs appear to be able to do so successfully.

One trick we applied that is worth mentioning was the limiting the size of our TFIDF feature sets before running the algorithm. Our initial clustering outputs showed very little variance, and offered nothing useful by way of recommendation. After limiting the TFIDF vectors for each book to its top 1500 values, however, the results improved dramatically, resulting in those shown above. This improvement was due to reducing the difficulties caused by high-dimensional data, where, by reducing the data's dimensionality, the clustering algorithm was able to find better distinctions between the observations.

K Nearest Neighbors

The next recommendation algorithm we applied was KNN. This marked an improvement from k-means recommendations by including the stylistic features of punctuation, author, book length, and so-on to the features we recommended upon, moving us closer to our original goal of capturing a user's full literary taste profile. These results were better than originally anticipated, providing recommendations that made intuitive sense and picked up upon subtle features in the text. Below are shown the nearest-neighbor recommendations for a selection of the books we tested.

Title, Author	**Synopsis of Target Book**	**Nearest-Neighbors Recommendations**
South! By Ernest Shackleton	A book about antarctic exploration, telling the tale of the *Endurance* and one of the leading figures of the Heroic Age of Arctic Exploration	Farthest North, The Voyages and Adventures of Captain Hatteras, Camp Life in the Woods, Pole to Pole, an Artilleryman's Diary
On the Origin of Species by Charles Darwin	A book on evolutionary biology, discussing plants, animals, and their natural survival mechanisms in detail	The Descent of Man and Selection in Relation to Sex, The Mushroom: Edible and Otherwise, Fungi: Their Nature and Uses, the Letters of Charles Darwin, North American Slime-Moulds, Darwinism
The Odyssey, by Homer	Homer's ancient epic poem of Odysseus' journey home after the battle of Troy. The first and most influential epic poem in the Western canon.	The Lusiad, the Aeneid, Egyptian Literature, Hebrew Literature, Kalevala: the Epic Poem of Finland, Sacred Books of the East, the Poetical Works of John Milton, Bible Stories and Religious Classics
A Doll's House, by Henrik Ibsen	A realist, modernist play about a woman's struggles with domesticity and traditional women's marital roles	Ghosts (Ibsen), an Enemy of the People (Ibsen), Dr. Jonathan (a play), Roses: One-Act Plays, Representative Plays by American Dramatists

These results show that the combination of stylistic and content-based features allowed the algorithm to make intelligent insights into the data. Starting with the first example, the system was able to recognize content similarity to a high level of precision, recommending arctic exploration stories and memoirs similar to the target text of an explorer sharing his account of a journey to the South Pole. Example two carries this further, showing a capability to understand broad categories as well, recommending general ecology and natural philosophy texts in relation to Darwin's *Origin of Species*.

The final two examples, however, are more intriguing, picking up on subtler features of the texts. The recommendations for Homer's *Odyssey* show that the algorithms were able to learn what epic poetry looks like from the features in our data. Epic poetry is a difficult genre to capture due to its broad-reaching subject matter and wide range of cultural and stylistic influences. It was exciting to see, then, that the nearest neighbors of *The Odyssey* were primarily epic poems from various cultures, including Milton's Christian epic *Paradise Lost*, the epic poem of Portuguese discovery *The Lusiad*, and even the traditional epics of Egypt, the Hebrew culture, and Finland.

Last, the recommendations for *A Doll's House* show that this system was able to pick up on non-novel styles of text. The nearest neighbors to this were all 20th century dramas, showing a precision in both genre and time period. This is an example of the value of the punctuation features in our data, where the prevalence of semicolons and quotation marks were likely influential in helping the system to discern the differences between drama and traditional literature.

With these results in mind, the benefit of KNN over k-means was that the algorithm was able to take the strengths of each of the k-means outputs, combine them, and consider them in relation to stylistic data, essentially capturing all important aspects of a text.

Similar to with k-means, this algorithm, too, appeared to be doomed to poor performance upon its initial application. After scaling our features to a [0,1] scale, however, the results improved considerably. This scaling marked a significant improvement because, due to our features being on different scales, KNN's measurements of euclidian distance were originally being dominated by the high variance of the book-length feature. This feature, differing by thousands of units between books, rendered differences in punctuation frequencies and cluster memberships essentially useless, since these other features only varied by a maximum of one unit between observations. Scaling the data set, then, evened out the influence of the many features and allowed them to each influence recommendations in meaningful ways.

It was also a slight surprise that the dimensionality of our data did not harm the quality of this algorithm's outputs. With cluster IDs from our three runs of the k-means algorithm and each author's name all expanded into binary variables, the feature set contained 10,498 dimensions. Typically, this high of a dimensionality leads to meaningless measures of distance. Due to the scaling of our data and the sparsity of our features (most of the binary variables were zeros for any given observation), however, the nearest neighbors' distances were small and consistent enough to provide recommendations that made sense.

Ordinary Least Squares

We fit the OLS models using manually gathered ratings data, hiding a small subset of ratings in order to evaluate model performance. The data tested was 20-book ratings lists gathered by Aniket and James. The results included some recommendations that showed promise, and others that did not make intuitive sense. The primary challenge of evaluating this model was that we could not generate large enough amounts of user data to make the model reliable. Because of our small testing sets, the OLS model would pick up on spurious correlations between various features and our ratings, leading to models that placed too much importance on unimportant features.

Samples of the training data and model outputs are shown below:

User	Training Data	Highest Predicted Ratings	Lowest Predicted Ratings	MSE on Withheld Data
Aniket	Rated Beowulf, Huck Finn, Dr. Jekyll and Mr. Hyde, The Picture of Dorian Gray	The Tragedy of Dido Queen of Carthage, Medea, the Trojan Women,	CIA World Factbook, Historic Court Memoirs, The Complete Historical Romances of George Ebers	16.273
James	Rated Don Quixote, Hamlet, Henry V, Ulysses, Heart of Darkness	The Merchant of Venice, A Portrait of the Artist as a Young Man, Don Quixote pt. 2,	The Practical High School Speller, the CIA World Factbook, Encyclopedia Britannica, Alaska Indian Dictionary	0.396

The end result of OLS was that we had too little ratings data to make this a useful model. The model is nice in that it is the only of those tested that provides personalized ratings, but, in the absence of large-scale social data, this algorithm will not be able to consistently provide useful recommendations. As is seen in the MSEs listed on the table above, the model performed well in predicting James' literary tastes, but poorly in predicting Aniket's. It is our prediction that these MSEs would fall in the presence of larger-scale ratings profiles.

Additional Challenges

The chief difficulties we encountered were found in the data collection and the scaling of our code to work on larger data sets.

The data collection proved difficult because of both the size of the library we were attempting to collect, and the restrictions put in place by PG. With our original goal being to obtain the full library of 55,000 english texts, we quickly realized that this would be a lengthy collection process. We were highly advised to use two-second delays in our code for downloading the zip files via Wget, due to PG's restrictions, and quick detection of IP addresses sending too many requests to their servers. Due to the time restriction that this creates on a large scale, we had to terminate our data collection early at 24,000 books. Additionally, we faced issues with PG's dowloading restrictions, attempting unsuccessfully to pull the data from the website by various means, only to be blocked by the restrictions, and even temporarily banned from the website. In the end, this resulted in a data set that was sufficiently large for our project goals, but not quite up to par with what we had hoped for.

We also faced challenges due to the scale of our data. Our initial code was written for and tested on the smaller University of Michigan data set. This presented the problem of our data being in a different format once we began updating our code to work on the parsed data, requiring us to collect features in slightly different ways.

More importantly, however, transitioning to the large data set required us to take a close look at the efficiency of our code. Slow and inefficient approaches to feature extraction worked without issue on the smaller scale of our original test sets, but proved to be significant threats to our project's completion on the scale of 24,000 documents. To remedy this, we removed several passes from our preprocessing code, and also moved our machine learning algorithms to Spark's mllib, which uses a hashing trick to speed up TFIDF, and a special parallelized variant of k-means called k-means||. We also ran into difficulties with running our code on AWS. After several failed clusters, however, we got our code to work on AWS.

Conclusion

In conclusion, our recommender system proved more successful than originally anticipated, providing useful recommendations based solely on content. KNN appeared to be the most successful of our recommendation algorithms, followed by K-Means and then OLS. While content-based recommendation may not be quite up to par with the collaborative filtering approaches taken by companies such as Amazon, this experiment shows that a content-only approach can, in fact, still be useful for both cold-starting a recommender system and for helping users to discover non-mainstream books that fit their preference profiles without necessarily having the social clout to be shown by a collaborative system. Looking forward, we could expand upon this research by obtaining the rest of the PG library, experimenting further with K values for optimal clustering outputs, discovering additional stylistic features to test upon, and building web UI around the project that allows real-world users to interact with the recommendation system and discover new books.

Bibliography

1. Lahiri, Shibamouli (2013). Gutenberg Dataset. Text corpus. The University of Michigan EECS.
2. Project Gutenberg (2016). Text library. Web.