Amazon Top 50 Bestselling Books 2009-2019

EECE2300: COMPUTATIONAL METHODS FOR DATA ANALYTICS

AUTHORS: JOLIE BERNER, ANGERICA FITZMAURICE, RYAN HOFFMAN

I. Abstract

The problem this project attempted to solve was to figure out how to make a book that is most likely to be popular simply based off of attributes like the title, author, genre, price, and release year. We used a list of 550 books from the Amazon bestseller list from 2009 to 2019 that was published to Kaggle in order to accomplish this, which required a significant amount of preprocessing and preparation to run both regression, classification, and clustering algorithms to predict two features: user rating (which is the average review out of 5) and total stars, which is that average rating multiplied by the number of people who reviewed the book. In the end, we elected to use linear regression, lasso regression, logistic regression, random forest regression, k nearest neighbors classification, decision trees classification, and random forest classification to predict the popularity of the books using the two metrics, while we used KMeans to cluster the books based off of genre. After running all of the models, we came to the conclusion that the random forest regressor is the best regression model for predicting the popularity of a book, while decision trees are the best at doing the same for the classification models, as well as KMeans clearly showing clusters by genre.

II. Introduction

This project is about the Amazon Top Bestselling Books 2009 – 2019, from Kaggle. The dataset contains 550 books. Data has been categorized into fiction and non-fiction using Goodreads. The analysis of this dataset will allow us to have a deep understanding of the book market trends over the past decade. This dataset includes seven categories such as book's name, book's author, Amazon user rating, number of written reviews on Amazon, book's price, year or years the book's ranked as the bestseller, and the genre (fiction or non-fiction).

The problem we are trying to solve is how to write a book that is mostly likely to be popular. The problem from a computational perspective is to predict an outcome based on different features. The inputs are Name, Author, User Rating, Reviews, Price, Year, and Genre. We would like to create a model that will predict if a book will have a high user rating and high number of reviews. This problem is interesting because it shows what consumers look for most in a book and books would, theoretically, be a best seller as well as popular based off rating and simple categorization.

For our project we will be using both regression algorithms to predict either the total stars or user rating, and classification algorithms to determine what makes a popular book and an unpopular book. We will also be using KMeans to cluster the books into two genres, fiction or non-fiction. In order to implement these algorithms, as we mentioned beforehand, we will be using Scikit-learn package which is the most useful library for machine learning in Python.

III. Background

The regression algorithms we used are linear regression and lasso regression random forest regressor. The classification algorithms we used are logistic regression, decision tree classifier, random forest classifier, and k nearest neighbors classifier. These algorithms will be explained more in depth in the project description portion of this report.

In addition to regression and classification algorithms, we also implemented clustering algorithm, we used KMeans. Clustering is quite literally the clustering or grouping up of data according to the similarity of data points and data patterns. The aim of this is to separate similar categories of data and differentiate them into localized regions. This way, when a new data point arrives, we can easily identify which group or cluster it belongs to. This is done unstructured datasets where it is up to the machine to figure out the categories.

In order to understand our results, we will need to define our performance metrics.

To measure the performance of our regression algorithms we will use the RMSE and R^2 scores. The RMSE is the root mean square error. This is the square root of the variance (or mean squared error), and equivalent to the standard deviation. This measure is in the same units as the original data and tells you the average deviation of the predicted values from the real values. It is ideal to have the lowest RMSE possible to lower the error.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

The R^2 score takes into account both the sum of squares of residuals and the total sum of squares. This is a good measure of how well a model will predict new points. In an ideal model, the line of best fit would perfectly predict the target values, making the sum of squares of residuals 0, and R^2 will be 1.

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

To measure the performance of our classification algorithms, we will use accuracy, recall, precision, and the F1 score. The accuracy is how many times the model predicted the correct result out of all the samples. The recall is how many times the model correctly identified the sample as positive out of all of the true positive samples. The precision is how many times the model correctly identified a sample as positive compared to the total amount of times it predicted a sample to be positive. Having a low recall means that there are many false negatives. False negatives are when the sample is predicted to be negative but it is actually positive. Low precision indicates a high number of false positives. False positives are when the model will predict the sample to be positive when it is actually negative. The F1 score will combine the recall and precision score.

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Since it is important to get a balance of both good precision and recall, the F1 score is a good indication of how well a model will predict test data.

IV. Related work

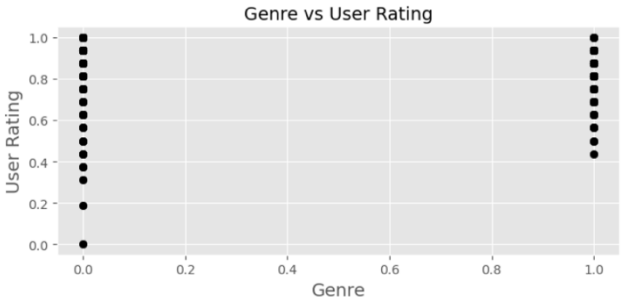
Other people have used similar algorithms to solve this problem by predicting the genre of a book using TensorFlow, which is similar in that predicting genre of a popular book is important, although we don't care about predicting the genre of every book. Another person compared books over the years based on categories, looking at things like ratings, prices, and genre to observe which consistently gets a spot on the best seller list.

V. **Project description**

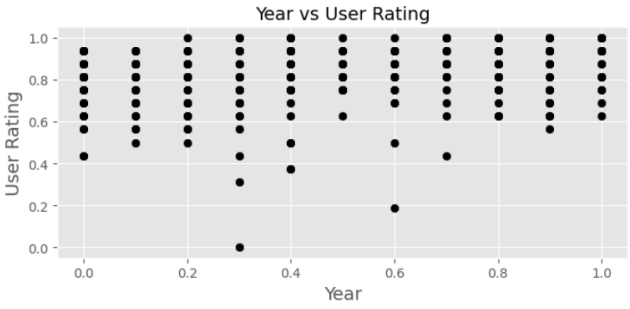
Exploratory Graphs:

In the scatter plot below, we can note that there are only a few books towards the higher price range, but all of the books towards a higher price range were towards the higher end of user rating.

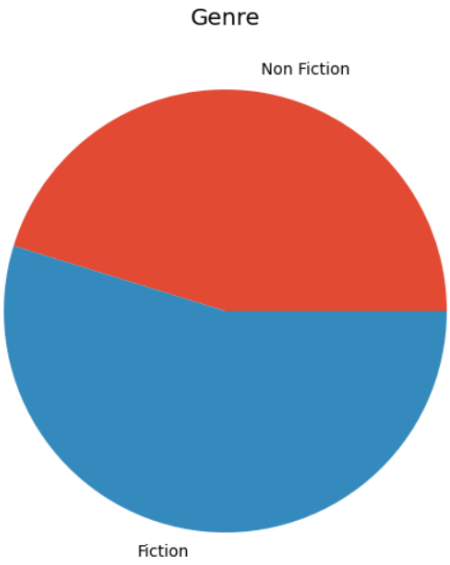
In the scatter plot below, we can note that the books on the lower end of the user rating are all genre 0, which is fiction, while non-fiction (genre 1) books tend not to get the very low user ratings.



In this scatter plot we can see that for all the years, user rating is pretty evenly distributed. There seems to be no correlation between user rating and year.



In this pie chart we can note that there is a pretty even split between fiction and non-fiction books, where there only slightly more fiction than non-fiction



Preprocessing:

The data from Kaggle, as seen in the figure below, started off with just book title, author name,user rating, number of reviews, price (in dollars), year it was released, and the genre.

	Name	Author	User Rating	Reviews	Price	Year	Genre
0	10-Day Green Smoothie Cleanse	JJ Smith	4.7	17350	8	2016	Non Fiction
1	11/22/63: A Novel	Stephen King	4.6	2052	22	2011	Fiction
2	12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson	4.7	18979	15	2018	Non Fiction
3	1984 (Signet Classics)	George Orwell	4.7	21424	6	2017	Fiction
4	5,000 Awesome Facts (About Everything!) (Natio...	National Geographic Kids	4.8	7665	12	2019	Non Fiction
...	...	...	...	...	...	...	...
545	Wrecking Ball (Diary of a Wimpy Kid Book 14)	Jeff Kinney	4.9	9413	8	2019	Fiction
546	You Are a Badass: How to Stop Doubting Your Gr...	Jen Sincero	4.7	14331	8	2016	Non Fiction
547	You Are a Badass: How to Stop Doubting Your Gr...	Jen Sincero	4.7	14331	8	2017	Non Fiction
548	You Are a Badass: How to Stop Doubting Your Gr...	Jen Sincero	4.7	14331	8	2018	Non Fiction
549	You Are a Badass: How to Stop Doubting Your Gr...	Jen Sincero	4.7	14331	8	2019	Non Fiction

Right from the start, the title, author, and genre features needed to be modified, while further investigation showed that there were multiple occurrences of certain books, as the list covered 10 years' worth of bestsellers and some books remain on the list year after year, as well as some missing prices as well as $0 prices, with the former being unusable and the latter likely resulting in the data being thrown off as the book didn't actually retail for $0.

The first step we took to preprocess our data was to fix up the prices, removing all NaN or 0 price values and replacing them with the average price for that author, or dropping the title if the author either didn't have any other books on the list or all the books were still $0. We decided to manually pick out and change which books had a 0 or NaN value for the price as there were relatively few on the list, first by finding titles where the price was $0, then computing the average price for that

author when the price wasn't $0. This needed to be done for 5 authors, followed by the price being set to the average price for each respective author when the title's price was $0. We decided to then check if there were any remaining NaN values, with 4 still remaining. We concluded that 4 is relatively small to the 550 books in total that we had in the data, so we dropped them as their absence likely wouldn't have affected the results significantly.

Following fixing the price, we decided to both count the number of times each book appeared in the list as well as drop any duplicate appearances by creating a dictionary with the book names and number of occurrences that had the occurrences increase if the book name was found in a list of the names multiple times, as well as add the title to a list of names to drop if it had already occurred at least once, while if it was only found once it would simply have an occurrence of 1 and wouldn't be dropped. The list of titles to drop was then applied to the dataframe to drop all repetitions as well as a new occurrence feature being added, both of which can be seen in the figure below.

	Name	Author	User Rating	Reviews	Price	Year	Genre	Occurences
0	10-Day Green Smoothie Cleanse	JJ Smith	4.7	17350	8.0	2016	Non Fiction	1
1	11/22/63: A Novel	Stephen King	4.6	2052	22.0	2011	Fiction	1
2	12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson	4.7	18979	15.0	2018	Non Fiction	1
3	1984 (Signet Classics)	George Orwell	4.7	21424	6.0	2017	Fiction	1
4	5,000 Awesome Facts (About Everything!) (Natio...	National Geographic Kids	4.8	7665	12.0	2019	Non Fiction	1
...	...	...	...	...	...	...	...	...
342	Winter of the World: Book Two of the Century T...	Ken Follett	4.5	10760	15.0	2012	Fiction	1
343	Women Food and God: An Unexpected Path to Almo...	Geneen Roth	4.2	1302	11.0	2010	Non Fiction	1
344	Wonder	R. J. Palacio	4.8	21625	9.0	2013	Fiction	5
345	Wrecking Ball (Diary of a Wimpy Kid Book 14)	Jeff Kinney	4.9	9413	8.0	2019	Fiction	1
346	You Are a Badass: How to Stop Doubting Your Gr...	Jen Sincero	4.7	14331	8.0	2016	Non Fiction	4

We then moved onto creating a workable feature for the book title and author name by performing TF-IDF vectorization on a new combined name and author feature, which was made by simply combining the "name" column with the "author" column and putting a space in between. We then created a function that, for each combined words cell, would tokenize the words, stem them using a porter stemmer, lemmatize the words, convert all the words to lowercase, and remove all the stop words for each title and author combination, turning them into what we called the clean name. In order to not mess with the remaining features that were already numeric, the clean names were separated out from the original dataframe and TF-IDF vectorization was performed, now turning each word into its own feature, as can be seen below. The words were then added back into the original dataframe, adding an additional 1384 features.

	000	10	100	11	111	12	13	14	140	150	...	write	year	york	young	zelda	zhi	zinczenko	zone	zoo	zusak
0	0.000	0.342	0.0	0.000	0.0	0.000	0.0	0.000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.000	0.000	0.0	0.426	0.0	0.000	0.0	0.000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.000	0.000	0.0	0.000	0.0	0.404	0.0	0.000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.000	0.000	0.0	0.000	0.0	0.000	0.0	0.000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.297	0.000	0.0	0.000	0.0	0.000	0.0	0.000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
342	0.000	0.000	0.0	0.000	0.0	0.000	0.0	0.000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
343	0.000	0.000	0.0	0.000	0.0	0.000	0.0	0.000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
344	0.000	0.000	0.0	0.000	0.0	0.000	0.0	0.000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
345	0.000	0.000	0.0	0.000	0.0	0.000	0.0	0.429	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
346	0.000	0.000	0.0	0.000	0.0	0.000	0.0	0.000	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

We then got rid of the “fiction” and “non-fiction” values for the genre feature by performing a simple label encoding, converting all the non-fiction values to a 1 and the fiction values to a 0. In addition, we removed all of the remaining features that contained words, which were now the name, author, name and author, and clean name features, leaving behind only numeric values for every feature. To finish up the pre-processing, we created a new feature called “Total Stars,” which was the user rating for each title multiplied by the number of reviews it got. The goal of doing this was to account for the fact that one book, for example, might have a 5 star rating but only got 10 reviews, while another book can have a 4.5 star rating with 500 reviews. On the surface, that 5 star rating may look like the better book, but there were less reviews impacting how many stars it got, while the total stars feature takes the number of reviews into account as well. Furthermore, if a book has more reviews, it’s likely that it had been purchased a lot and a lot of people were reading it, which could be another way of considering a book to be popular. All of the features were then scaled using a min-max scaler from 0 to 1 as negative values that might come from using something like a standard scaler could cause issues with classification algorithms. The final data set, with 1391 features and 347 entries compared to the original 7 features and 550 entries can be seen below.

	User Rating	Review	Price	Year	Genre	Occurences	Total Stars	000	10	100	...	write	year	york	young	zelda	zhi	zinczenko	zone	zoo
0	0.8750	0.197178	0.067308	0.7	1.0	0.000000	0.193075	0.0	0.914439	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.8125	0.022949	0.201923	0.2	0.0	0.000000	0.021992	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.8750	0.215730	0.134615	0.9	1.0	0.000000	0.211241	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.8750	0.243577	0.048077	0.8	0.0	0.000000	0.238507	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.9375	0.086875	0.105769	1.0	1.0	0.000000	0.086891	1.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
342	0.7500	0.122124	0.134615	0.3	0.0	0.000000	0.114481	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
343	0.5625	0.014407	0.096154	0.1	1.0	0.000000	0.012571	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
344	0.9375	0.245866	0.076923	0.4	0.0	0.444444	0.245879	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
345	1.0000	0.106783	0.067308	1.0	0.0	0.000000	0.109032	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
346	0.8750	0.162794	0.067308	0.7	1.0	0.333333	0.159409	0.0	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

For choosing our target values, we decided to work with both user rating and total stars, primarily to see which might be better for the algorithms to decide whether a book is popular or unpopular, where popular doesn’t just mean it sells well, but rather than it sells well and is highly regarded. As previously mentioned, we had the idea that the total stars feature might be better at determining popularity for a few reasons, although at the same time it was also possible that the feature could just give books with more reviews the appearance that they’re better than they really are. In other words, a book that might have 4000 reviews but a 3 star rating would significantly more popular than a book with 2000 reviews and a 4 star rating. Again, popularity means both number of buyers

and how well the book is received by the public, so the total star metric works well enough for both, but the user rating metric works very well for the how well the book is received by the public. Furthermore, in practice, running the algorithms using both total stars and user rating as target features wouldn't have resulted in much more time spent waiting for the algorithms to run, as the data was small enough that no matter what it ran relatively quickly, although it should be noted that it was still large enough that we had faith that there were enough samples for the algorithms to work as intended. Using both features as targets also enabled us to think about what really makes a book popular: the average user rating or how many people bought the book and at least liked it a little. If we assume that the classes of popular and unpopular were set using the proper thresholds, as in we labeled each book in the test and training sets properly ourselves, the algorithms should help to tell us what matters more when deciding a book is popular.

Since our goal is to predict a "popular" book versus an "unpopular" book, we decided to use both supervised learning techniques regression and classification. For both, we will use targets of total stars and user rating. When we are using regression, the values for these to targets are both continuous so we will just be predicting the decimal value. When we are using classification, we needed to have two distinct classes of "popular" vs "not popular". In this case, we split the data so that 80% was "not popular" and 20% was popular for both total stars and user rating. For total stars, 20% of the data had total stars > 0.17. And for user rating, 20% of the data had user rating > 0.8.

Algorithms:

Linear Regression:

The first regression algorithm we tried is linear regression with targets total stars and user rating. The linear regression algorithm will train a model so that the target will have a linear relationship between the feature values. In this case we are using multivariate linear regression because there are many features. The model can be represented with this mathematical equation. Where each x represents a different feature.

$$h(\mathbf{x}) = w_0 + w_1 x_1 + \ldots + w_p x_p = w_0 + \sum_{j=1}^{p} w_j x_j$$

To implement this algorithm, we used Scikit-Learn's LinearRegression class. This class will solve the normal equation and learn the coefficients and intercept of the line of best fit. The normal equation is the equation that minimizes the loss function as explained in lecture 14. Here, w* is the closed form solution that minimizes the loss function.

$$Loss(\mathbf{w}) = \sum_{i=1}^{n} (y_i - h(\mathbf{x}_i))^2 = \sum_{i=1}^{n} \left(y_i - \sum_{j=0}^{p} w_j x_{ij} \right)^2$$

$$\mathbf{w}^* = (X^T X)^{-1} X^T \mathbf{y}$$

Lasso Regression:

We also used lasso regression to train models to predict total stars and user rating. Lasso regression is a form of linear regression with the L1 regularization term. Using a regularization term prevents overfitting to the training data by penalizing features with large weights. The loss function for lasso regression can be represented in a mathematical expression as shown in lecture 14. We can see that

the term alpha times the sum of the absolute values of the weights is added to the mean squared error function. This makes sense, because if we are trying to minimize the function, adding the absolute values of the weights will cause the result to be much higher depending on if the weight is higher.

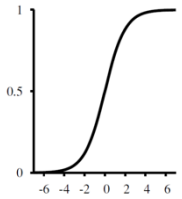
$$J_{Lasso}(\mathbf{w}) = \mathrm{MSE}(\mathbf{w}) + \alpha \sum_{i=1}^{p} | w_i |$$

We used Scikit-Learn's Lasso class to implement this algorithm. This class will perform lasso regression using a closed form solution. In this case, we set alpha to be .001 for user rating and .0001 for total stars. A larger alpha value will penalize larger weights more heavily. A large alpha value will force more weights of features to zero.

Logistic Regression

One of the classification algorithms we used is logistic regression. We predicted a "popular" and "unpopular" book based on the thresholds of total stars and user rating mentioned above. Logistic regression is very similar to linear regression, but instead of predicting the continuous target variable, it will predict the probability that each sample belongs to a certain class. It uses the logistic function as a threshold to determine the probability of the sample belonging to positive class. The logistic function and its plot is shown below, as explained in lecture 14.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Logistic regression can be shown in this mathematical equation, from lecture 14, where "w" represents the weight given to each feature, "x".

$$h(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

As opposed to linear regression, which minimizes the loss function to solve the equation, logistic regression uses the log loss function. This is because the loss function (mean squared error) is not convex when using logistic regression. It is bad to minimize a function that is not convex because there is not a guaranteed global minimum. In this case, the log loss function (shown below from lecture 14) is a convex function that can be minimized.

$$J(\mathbf{w}) = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i) \right]$$

To implement this algorithm, we used Scikit-Learn's class LogisticRegression.

K – Nearest Neighbor (KNN):

The K-nearest neighbors algorithm, commonly referred to as KNN, uses, as the name implies, the classification of the k nearest neighbors to determine the classification of the item that's being looked at. In simplest terms, each item is classified based on what a majority of its k nearest neighbors are, where k is a number that is set by the person running the model.

The algorithm itself is relatively simple, as it just uses the mode of the nearest neighbors to determine what each item should be classified as, with most complexity lying in how distance is calculated. Various options exist, including, but not limited to, the Euclidean and Manhattan distances, each with their own pros and cons in what they say is closer than the others. The distance theory also applies when there are multiple attributes being looked at, with a good example being for Euclidean distance where the total distance is just the square root of the sum of the distances between the neighbor and the item being classified squared. The distance formula in the figure below, for example, would be for two attributes, with more being represented by adding another square of the difference under the square root.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Complexity also exists for the person running the model in choosing the right number of neighbors that will be used in classifying the item. While in some other algorithms larger samples might always seem better, when the k value is too large then the algorithm starts to look at samples that are way too far away, while if the k value is too small then the algorithm might pick up on outliers that just so happened to be near the item that's being labeled. For most cases, using a k value equal to the square root of n, the number of samples being looked at, usually works very well.

When looking at the book dataset, we decided to use a grid search for the number of neighbors between 1 and 200, and we used Scikit-Learn's KNeighborsClassifier class to implement it in predicting the classifications for total stars and user rating. Each feature other than the selected target was used in order to use the classifier, resulting in many factors lying inside the distance equation.

Decision Trees:

Decision trees are another type of classification algorithm that, at their core, are simply a series of decisions that you follow to, in the end, reach your classification for the test data. The tree in the name comes from the fact that a lot of the parts of the algorithm can be modeled by tree terms, with the first decision being the root, and each decision leading into a new branch, creating subtrees with their own roots and branches. Each branch is formed by asking a new question, or, in the case of numerical data, creating a range for which the value should be.

Most often for numerical data the decisions are made by asking if one of the features is above or below a certain value and then deciding whether the answer is true or false, branching off into another question until eventually the sample is classified.

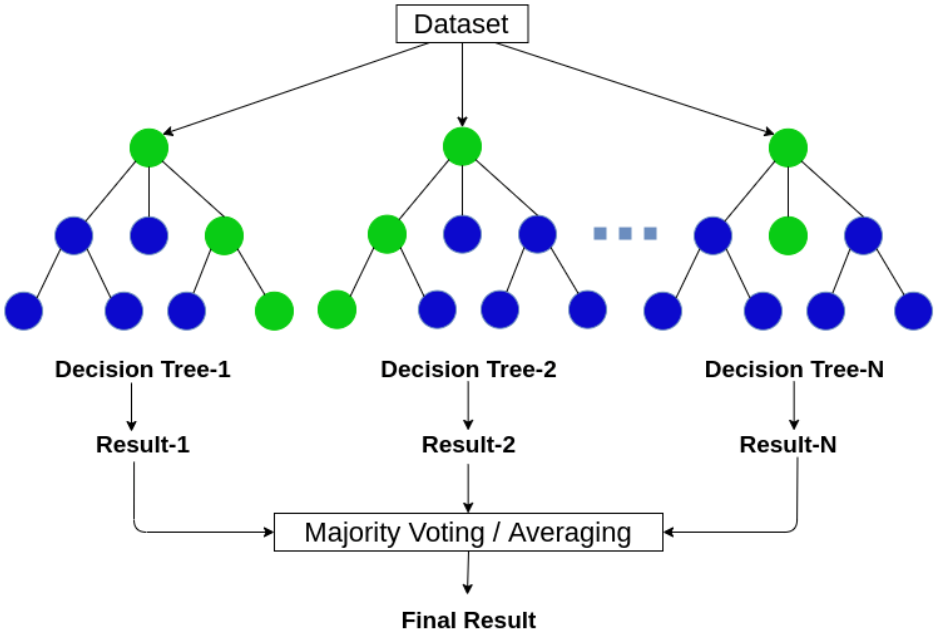
In practice, there could be an extremely large number of branches that go on for a very long time, which could result in the data being extremely overfit and the test data not doing nearly as well as the training. Changing the maximum number of branches and depth is referred to as, pruning, another tree term.

For our data, we used Scikit-Learn's DecisionTreeClassifier class to predict the total stars and user rating classifications, as well as a grid search to determine what the best parameters are for the maximum depth and maximum features used in the tree, as well as whether to use the Gini impurity or entropy, both of which are essentially the same measure with some minor mathematical differences, in determining the attributes.

Random Forest Classifier:

Random forest classifiers are essentially just a more advanced decision tree algorithm, which use multiple randomly generated decision trees, each with their own varying choice at attributes, branches, and depth to classify each sample multiple times, and then performing a majority voting rule on the classification, just as the algorithm did in the KNN algorithm, to decide what the sample should be classified as.

In practice, the algorithm works mostly the same as the decision tree algorithm, just being performed multiple times to find what is the best classification for the sample. The main advantage over the ordinary decision tree algorithm is that different tree compositions can result in different classifications, where some might be better than the others. One decision tree might do a decent job at classifying the sample, but, arguably, if there are more decision trees working together there are higher odds that the sample will be classified properly. The image below shows a good example of how a random forest classifier works.

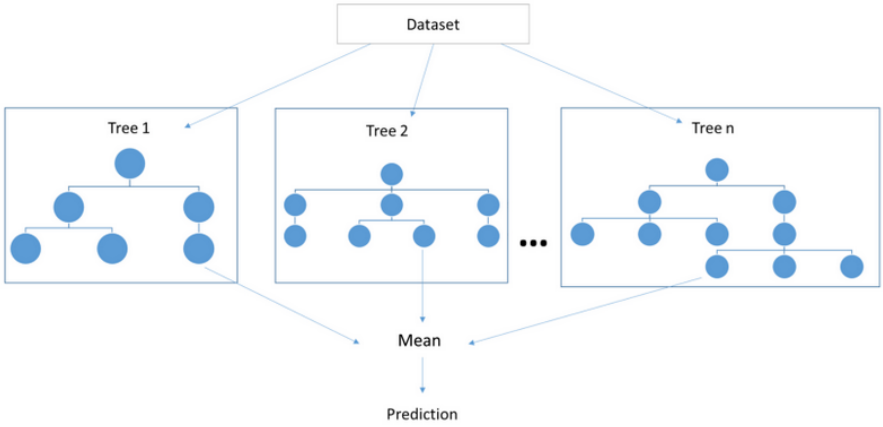


That being said, the same issues that arise from overfitting exist in the random forest classifier as they do in the decision tree classifier, which makes pruning equally important in ensuring that the trees don't overfit.

In our case, we used Scikit-Learn's RandomForestClassifier class to predict the classifications for total stars and user rating.

Random Forest Regression:

Similar to random forest classifier, random forest regressors are derived from multiple randomly generated decision trees with an average taken for the final result. Again, there are different parameters that are able to be tuned such as the number of estimators, number of samples, max depth, and max features. The only difference is that the random forest regressor will perform regression isntead of classification, meaning that instead of estimating the class of a sample, it will estimate a value. This results in a new parameter for choosing either “mse” or “mae” (mean squares error or mean aboslute error). The function chosed will measure the quality of the split. Again, similarly to random forest classifiers, this model has the potential to overfit, if there are too many branches of the tree, which again the solution could be to use early stopping, or pruning the completed tree model. Since the random forest regressor is identical to the random forest classifier, except the type of prediction, it is appropriate to view the model similarly to the classifier. As we can see, the only difference is taking the mean of the trees, instead of majority vote.



For implementing this mode, we used Scikit-Learn's RandomForestClassifier class to predict the values of total stars and user rating.

KMeans:

The cluster algorithm we used is KMeans that will allow us to cluster our data into 2 groups, namely Fiction and Nonfiction. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters fixed apriori. The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed, and an early group age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step. After we have these k new centroids, a new binding must be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop, we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more.

Finally, this algorithm aims at minimizing an objective function know as squared error function given by:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} \left(\left\| x_i - v_j \right\| \right)^2$$

where,

'$||x_i - v_j||$' is the Euclidean distance between x_i and v_j.

'c_i' is the number of data points in i^{th} cluster.

'c' is the number of cluster centers.

VI. Empirical Results

Algorithms:

Regressors

Linear Regression:

```
Results for User Rating target
RMSE on the training set:  6.309740391678008e-16
R^2 score on training set:  1.0
RMSE on the test set:  0.11972828191974981
R^2 score on test set:  0.05561862311373267

Results for Total Stars target
RMSE on the training set:  4.533970327581221e-16
R^2 score on training set:  1.0
RMSE on the test set:  0.059383507310479665
R^2 score on test set:  0.8336769704397591
```

Lasso Regression:

Alpha = 0.001

```
Results for User Rating target
RMSE on the training set:  0.10275409629852032
R^2 score on training set:  0.5143331590752129
RMSE on the test set:  0.11715251152876123
R^2 score on test set:  0.0958153701636536
```

Alpha = 0.0001

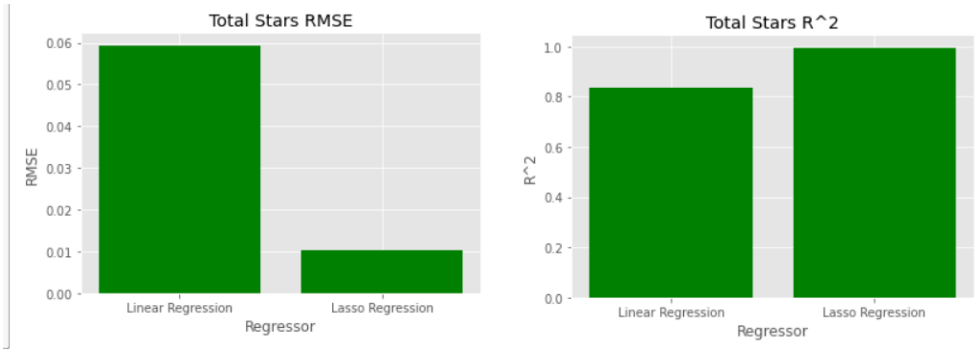
```
Results for Total Stars target
RMSE on the training set:  0.006429366004956142
R^2 score on training set:  0.9963148985424308
RMSE on the test set:  0.010150867234712 42
R^2 score on test set:  0.9951401037994316
```

Linear Regression vs. Lasso Regression for Training Set

We can see here when using Lasso Regression compared to Linear Regression, the RMSE on the training set will increase. This is due to the regularization in Lasso Regression, preventing the model from overfitting to the training set.

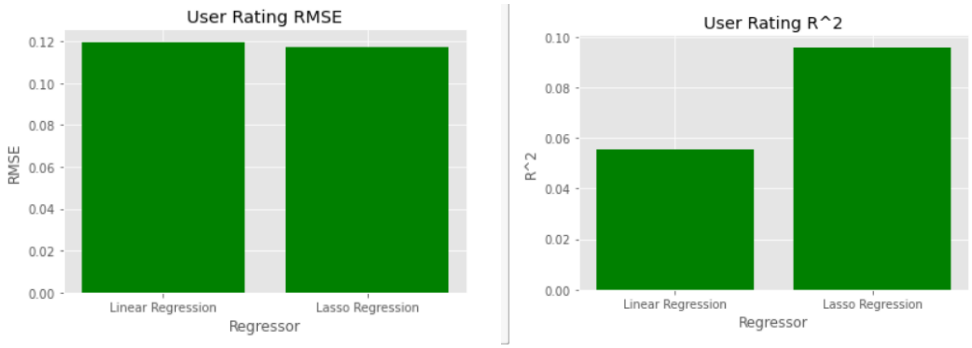
Linear Regression vs. Lasso Regression for Test Set

Total Stars:



We can see for when the target is total stars, the RMSE is significantly smaller using lasso regression compared to linear regression. This is ideal, since we want the smallest error on the test set, keeping in mind this data was scaled so the max value is 1. We can see for the R^2 value, lasso regression has a value much closer to our idea value, 1. lasso regression performed much better when predicting total stars as compared to linear regression.

User Rating:



We can see for when the target is user rating, the RMSE is very similar using lasso regression compared to linear regression. This error isn't awful when understanding that the data is in range 0 to 1, but it is a worse RMSE compared to total stars above. We can see for the R^2 value, lasso regression has a value higher than linear regression braising the value from around .055 to around .095. We can see both of these values are not close to the ideal R^2 value of 1. A low R^2 value

means that the data is not very close to the fitted regression line. For this case, the RMSE is good, but the data is not close to the regression line created by the model.

Random Forest Regression:

```
User Rating
RMSE on the training set:  0.011032064973320323
R^2 score on training set:  0.9922068780781766
RMSE on the test set:  0.00760678217097436
R^2 score on test set:  0.9927924553495077


Total Stars
RMSE on the training set:  0.013577477744992938
R^2 score on training set:  0.9933047276005343
RMSE on the test set:  0.0056460627662849915
R^2 score on test set:  0.9934289806176718
```

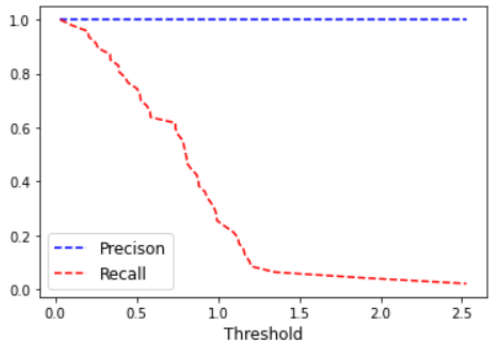
Random forest regression performed very well on both using user rating and total stars as targets. We can see that the RMSE was less than .01 for both cases. This is really good results since the data is from 0 to 1. Also, the R^2 score for both cases is .99 which is very close to the ideal score of 1. This is using the default parameters, as we did not have to use grid search because the results were very good.

Classifiers:

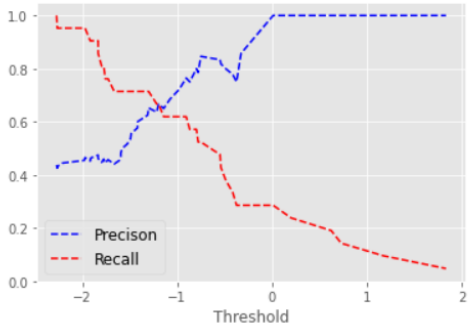
Logistic Regression

Total Stars:

```
Accuracy on train
1.0
Train Precision
1.0
Train Recall
1.0
Train F1
1.0
```




```
Accuracy on Test
0.8275862068965517
Test Precision
1.0
Test Recall
0.2857142857142857
Test F1
0.4444444444444445
```



Here we can see that logistic regression with total stars as the target performed perfectly on the training set, but not as good on the test set. The precision on the test set is perfect but the recall is very low. When we adjust the threshold, it will increase out recall, and lower our precision, this

will ultimately increase our F1 score. Since the intersection of precision and recall are around 0.6, we used the threshold of when recall was above 0.6

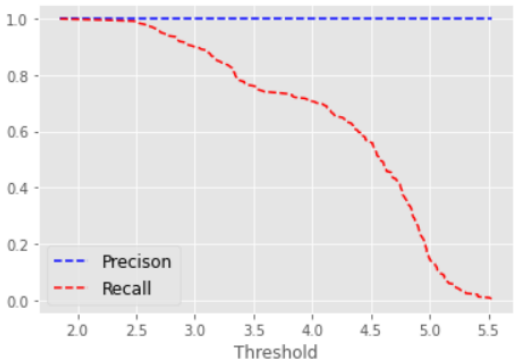
```
For using threshold where recall > 60
Precison
0.75
Recall
0.5714285714285714
F1 Score
0.6486486486486486
```

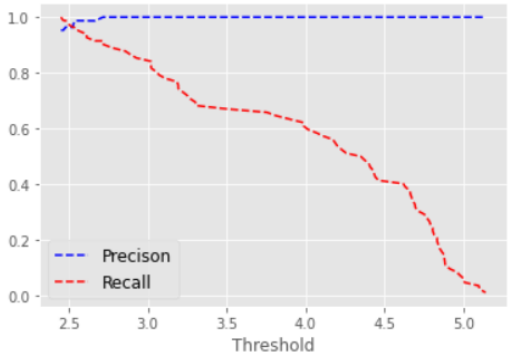
We can observe that precision went from 1 to .75 and recall went from .28 to .57. This resulted in an F1 score increase from .44 to .64. Although we were able to increase the results, an F1 score of .64 still is not very good.

User Rating:

```
Accuracy on train
0.9730769230769231
Train Precision
0.9728682170542635
Train Recall
1.0
Train F1
0.9862475442043221
```

```
Accuracy on Test
0.9425287356321839
Test Precision
0.9425287356321839
Test Recall
1.0
Test F1
0.9704142011834319
```





Here we can see that logistic regression with user rating as the target performed very well on both the training and test set. The recall on both the test and training sets were perfect and the precision was only slightly lower. In this case, we do not need to use thresholding to improve our results since they are already good.

K – Nearest Neighbor (KNN):

Total Stars:

```
Stars Scores

-Train
--Accuracy
0.8538461538461538
--Precision
0.8579881656804734
--Recall
0.9119496855345912
--F1 Score
0.8841463414634145

-Test
--Accuracy
0.8390804597701149
--Precision
0.9206349206349206
--Recall
0.8656716417910447
--F1 Score
0.8923076923076922
```

The total stars data yielded relatively high scores for all metrics, with none of them dipping below 80% and most sitting in the high 80s to low 90s, all of which are pretty good. The training set was surprisingly low in scores, however, with the accuracy actually the lowest of all the scores in training and the second lowest score overall, right above the test data accuracy. The grid search for the best number of neighbors was 3, which is much smaller than the square root of the number of samples, which is about 19.

User Rating:

```
User Rating

-Train
--Accuracy
0.6615384615384615
--Precision
0.6808510638297872
--Recall
0.9248554913294798
--F1 Score
0.784313725490196

-Test
--Accuracy
0.7241379310344828
--Precision
0.7763157894736842
--Recall
0.8939393939393939
--F1 Score
0.8309859154929577
```

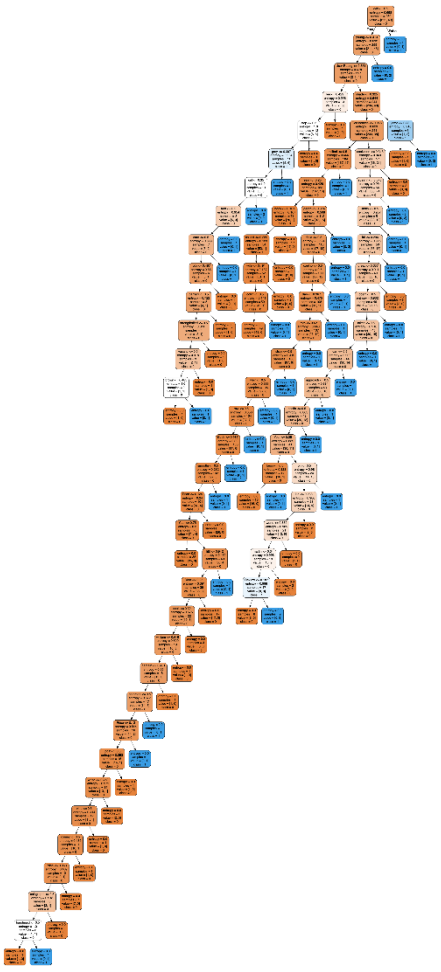
Overall, the user rating scores were significantly lower than the total stars scores using the KNN classifier, indicating that the user rating popular and non-popular classifications aren't grouped together nearly as well as the total stars. This serves to be even more true when looking at the grid search, where the search found 61 neighbors to be the best parameter, which, unlike in the total stars model, was much higher than the square root of the number of samples, 19 neighbors. While the scores aren't terrible, they could definitely be much better, indicating that a KNN algorithm isn't the best for classifying popular books based on their rating on Amazon.

Decision Trees:

Total Stars:

```
Stars Scores

-Train
--Accuracy
1.0
--Precision
1.0
--Recall
1.0
--F1 Score
1.0

-Test
--Accuracy
1.0
--Precision
1.0
--Recall
1.0
--F1 Score
1.0
```

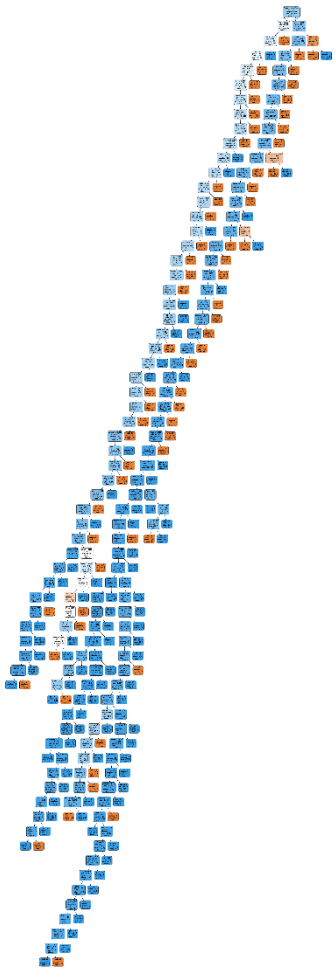
The decision tree classifier did amazingly well on the total stars yielding perfect scores for everything, both training and test sets included. Simply put, the decision tree classifier seemed to neither overfit not underfit for the total stars feature, with the perfect results being a clear indication that there is a strong correlation between the rest of the features and total stars. It managed to yield these results by using the Gini criteria, 150 for the maximum depth, and 95 for the maximum number of features in the grid search. The actual tree that was used is shown below.



User Rating:

```
Ratings Scores

-Train
--Accuracy
1.0
--Precision
1.0
--Recall
1.0
--F1 Score
1.0

-Test
--Accuracy
0.6206896551724138
--Precision
0.7121212121212122
--Recall
0.7704918032786885
--F1 Score
0.7401574803149605
```

The user rating scores were perfect for the test data, just as in the total stars, although just like in the KNN classifier, the test data yielded significantly worse results. The results are actually worse than the KNN results, indicating that the decision tree likely overfit the to the training data, even with the grid search attempting to prune the tree, again using Gini criteria, although this time with a max depth of 196 and max features of 79. A key takeaway from the decision tree is that while it works well for the total stars metric, it does the exact opposite with the user rating metric. Again, the tree itself is shown below.

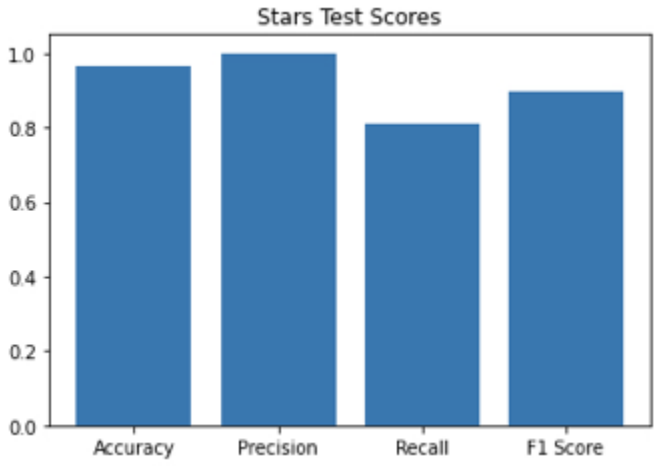


Random Forest Classifier:

Total Stars:

```
Stars Scores

-Train
--Accuracy
1.0
--Precision
1.0
--Recall
1.0
--F1 Score
1.0

-Test
--Accuracy
0.9655172413793104
--Precision
1.0
--Recall
0.8125
--F1 Score
0.896551724137931
```

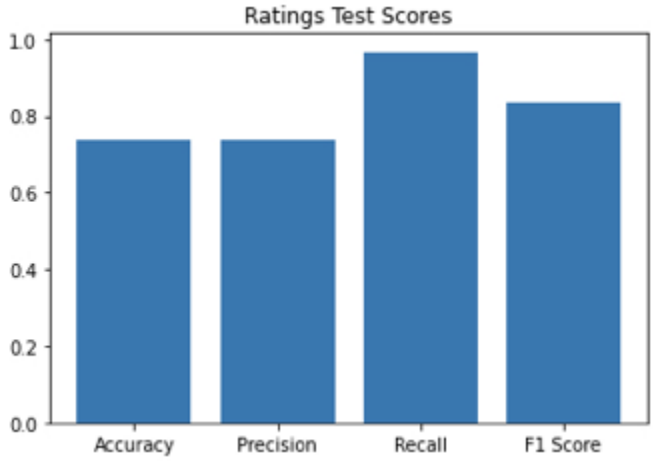
The random forest classifier, which, again, is just an expanded version of the decision tree classifier, actually did worse at classifying the test data for the total stars metric than the regular decision tree classifier (although the training data did yield perfect scores). That being said, the precision was still perfect, and the accuracy of 0.966 was very close to 1, but the recall of 0.813, while still good, fails in comparison to the perfect recall of the decision tree classifier. While the random forest classifier isn't terrible by any means, and it is better than the results from the KNN classifier, the decision tree classifier appears to be better than the random forest at classifying based on total stars. Below is a bar graph of all four scores.



User Rating:

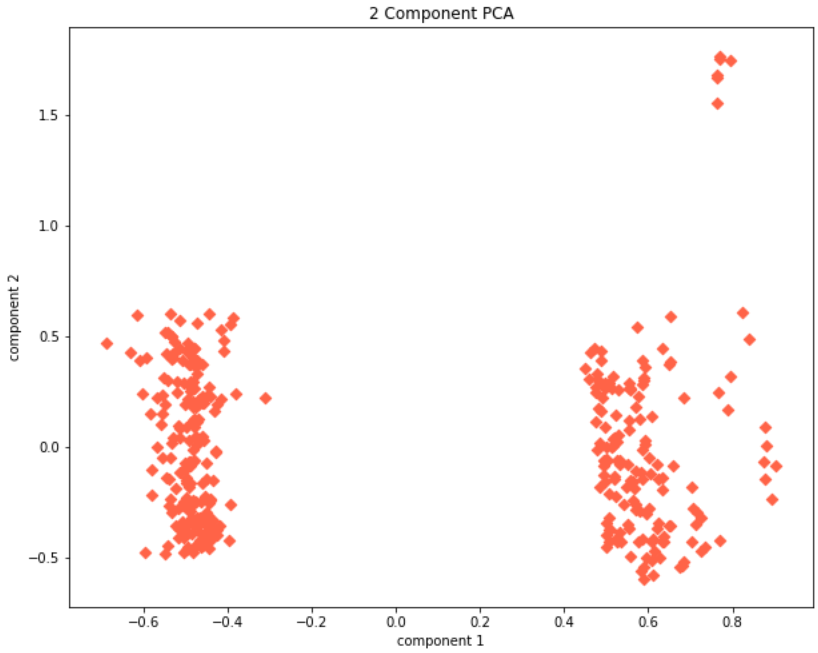
```
Ratings Scores

-Train
--Accuracy
1.0
--Precision
1.0
--Recall
1.0
--F1 Score
1.0

-Test
--Accuracy
0.735632183908046
--Precision
0.7375
--Recall
0.9672131147540983
--F1 Score
0.8368794326241135
```

Again, the training data yielded perfect scores for the user ratings data, although the quality of the scores in the test data appeared to flip from the total stars data, with the recall now being high and the remaining scores relatively lower in comparison. At the same time, all of the scores for the test set were higher in the random forest classifier, although not by much except in the case of recall. A bar graph with the scores for the test set is shown below.



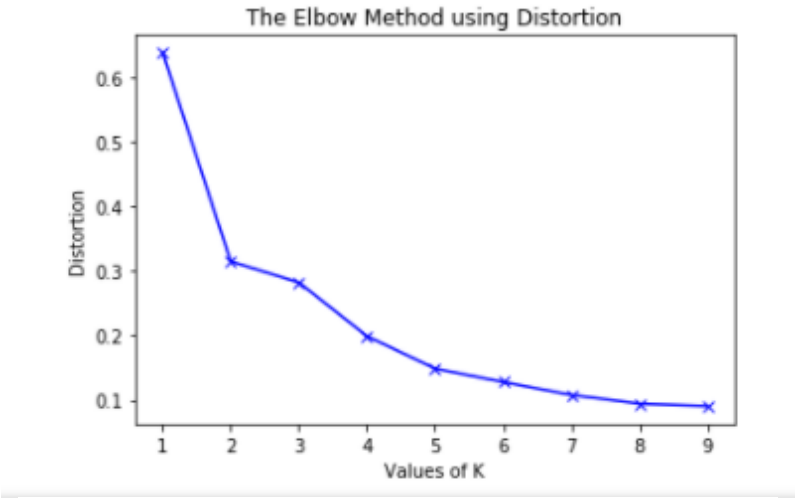
KMeans:

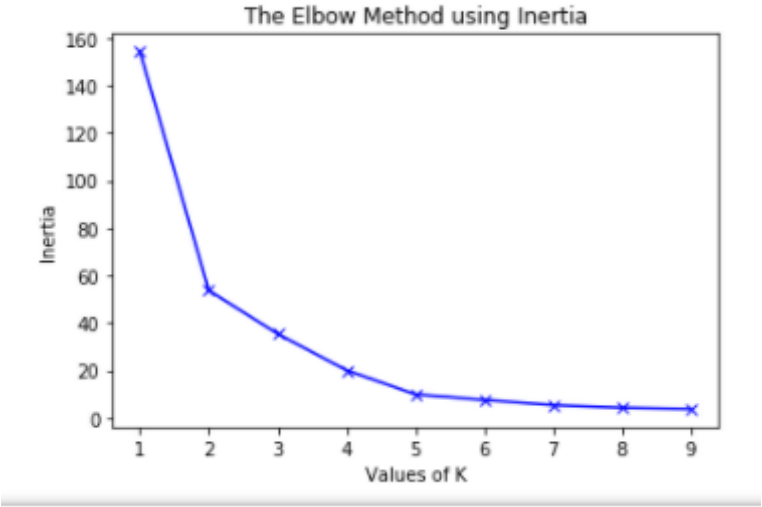
The basic idea behind KMeans consists of defining k clusters such that total within-cluster variation or error is minimum. In order to determine the optimal number of clusters into which our data may be clustered, we used elbow method, which calculates the “Within-Cluster-Sum of Squared Errors (WSS)” for different values of k, and choose the k for which WSS becomes first starts to diminish. In the plot of WSS vs k, this is visible as an elbow.



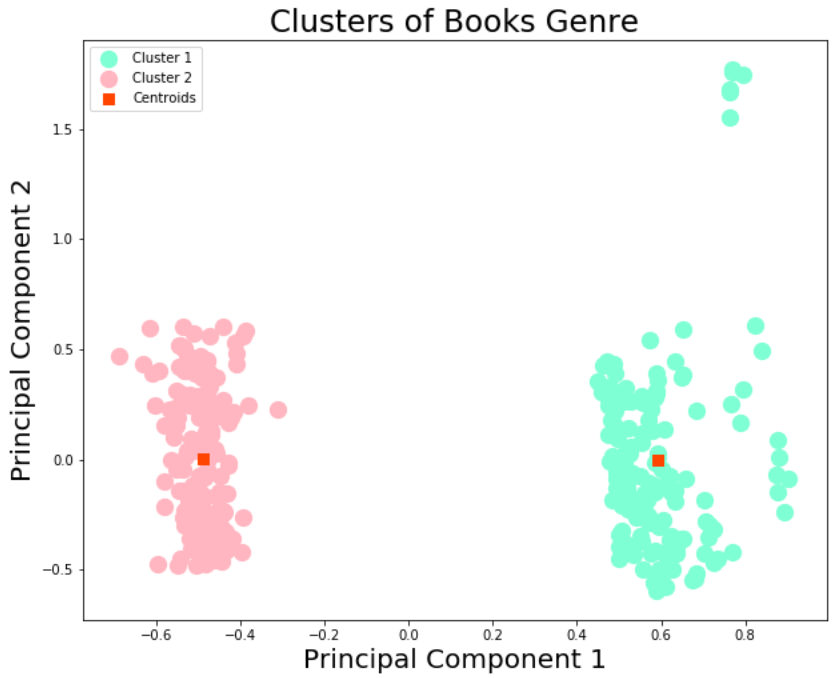
From the above graph, we clearly see that we clustered the data into two groups. However, visualizing alone cannot always give the right answer. Therefore, we demonstrated defining the distortion and Inertia to determine the optimal value of k.

For definition of distortion, it calculates as the average of the squared distances from the cluster centers of the respective clusters. Typically, the Euclidean distance metric used. Meanwhile, inertia is the sum of squared distances of samples to their closest cluster center. We iterated the values of k from 1 to 9 and calculate the values of distortions for each value of k and calculate the distortion and inertia for each value of k in the given range.





To determine the optimal number of clusters, we have to select the value of k at the elbow, which the point after which the distortion or inertia start decreasing in a linear fashion. Thus, for the given data, we concluded that the optimal number of clusters for the data is 2 and to visualize the two clusters that were formed with optimal k. We can clearly see two clusters in the graph below, each cluster represented by a different color.



VII. Conclusions

Random forest regressor was our best regressor since it performed very well on both the total stars and user rating targets. lasso regression (which performed better then linear regression) performed very well on total stars but had a slightly worse RMSE for user rating.

We decided that decision trees classifier was our best classifier since it performed perfectly on the test set for total stars. Although it performed slightly worse on the test data for user rating, in comparison to the random forest classifier, the perfect score on the total stars data leads us to believe it is the best classifier. We have eliminated the logistic regression, even though it did well on user rating, because it performed worse on total stars. Using our best models, we would be able to predict if a new book will be popular among readers after collecting the necessary data.

Looking at the classifiers, overall there seemed to be much higher scores when looking at the total stars models when compared to the user ratings models. Assuming that we classified all of the samples properly for both model types, we can come to the conclusion that there seems to be a stronger relationship between the total stars metric and whether or not a book is popular than the user rating metric, or, in other words, using total stars as the target will yield a better prediction as to if a new book will be popular.

For KMeans, we found the optimal k by using elbow method and we concluded that the data was clustered into two, which were categorized into two genres, fiction and nonfiction.

In this project we learned how to take a real dataset, preprocess the data to be suitable for our chosen algorithms, and interpret our results using performance metrics.

If we had more time to work on this project, it would have been nice to perform cross validation on our data. This would have helped us tune models to avoid overfitting and not overfit to the test data. Also, we could have tried our more algorithms to compare them to our current results.

Future EECE2300 students should be wary of how much time running some of these models can take, as well as how much time it takes to properly preprocess datasets that are already published. Getting late starts on either process can throw off the timeline for the remainder of the project, resulting in a lot of stressing over how to get things done in time. Building off of that, future students should also take the time to learn everything they can about how they plan to run their models before they actually start to run them, as having the wrong format for data (such as negative values when scaling) can throw off certain models and require you to go back and re-preprocess the data.