

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | Develop an end-to-end Machine Learning Pipeline (Use case: Book rating) | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | Philippe FOGOUMDavid BEGARINGuillaume NONYSaïd HAMDI |  | | | |
|  | | | | 25/08/2023—ML with Python Labs—Prof : Hanna Abi Akl |  | | | |
|  | | |  | | |  | | |



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | |  |  | | |  |
|  | INTRODUCTION | | | | | | |  |
|  |  | | |  |  | | |  |
|  |  | |  | | |  | |  |
|  |  |  | Various books are referenced on Goodreads website which also allow a 5-point rating system.  The purpose of this Machine learning project is to train a model (using a provided dataset books.csv) that predicts a book’s rating.  The present report documents the approach, various steps followed and results obtained with regards to:   * Data analysis * Feature selection * Model training & evaluation   **GitHub Repository is available at:**  <https://github.com/PhilippeJacques/Book-Rating.git> | | |  |  |  |
|  | | | | |
|  |  |  |  |
|  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | |  |  |  |
|  | | Approach & Results | | |
|  | |  |  |  |
|  | Problem definition: **#About the data:**   * We do have a dataset “books.csv” (11127 books, 12 features) * The data analysis step shall ensure consistency and quality * Question : how to predict the average rating of a book (numeric value) ? * The problem is deemed to be a ***regression*** one (enough data)  Data analysis **#Data Processing / cleaning**  Summary of actions on “books.csv” dataset (for more details please Ref. notebook v6.1):  - fixing formatting issues like:  > ‘unnamed ‘ column with *NaN* entries due to extra ‘commas’ in ‘authors’ column (fixed after commas manual removal)  > unwanted spaces before certain column names (removed)  - importing dataset as a pandas dataframe & remove inconsistent parameters  - removing all data having ratings count equal to zero as the rating count is our most leading feature for the decision process  - removing all books with num\_pages equal to zero (deemed as having no content)  Remark : variables type is “numeric” and “string”.  **#Exploratory Analysis (& plotting)**  - Check potential correlations on most interesting features (average\_rating, num\_pages, ratings\_count, text\_reviews\_count)  - From heat map & scatter plot:  > almost no correlation with average\_rating.  > Text\_reviews\_count highly correlated to ratings\_count.  Need some data processing to create a stronger correlation.  > ratings\_count or text\_reviews\_count vs average\_rating is curve-like,  exponential to some degree (log function could create a greater correlation between these two parameters and straighten the curve)    Visualization of distributions using boxplots:  > “*average\_rating*” boxplot: ~normal distribution, with a mean ~4.  Also, data imbalance between high ratings vs low ratings (below 3).  > other features significantly skewed  **Feature Selection**  **#Feature engineering & pruning**  “*Publication\_Date*”: converted to pandas datetime format for easier processing as such feature is deemed significant for our objective.  Language encoding: “*is\_eng*” category created to classify the book as english or others: 1 for ‘english’ or 0 for ‘not english’ (then, ‘*language\_code*’ is dropped).  Removing Audiobooks: assuming an audiobook has a text-based version with less than 30 pages, or with ‘*publisher’* having "Audio" in its name: such books removed from dataset  Removing ‘*isbn’ and ‘isbn13’* of the book, for it has no good correlation with our findings.  Creating new variables with in-depth detailed information about dataset:  - duration feature: how long it is since the book was published, based on the assumed date of extraction (2022-12-31) and the ‘*publication\_date’*  - From the above information, we determine the average frequency of ratings and reviews (number of ratings or reviews per unit of time).  - Other ratios creation: the average rating\_count and text\_reviews\_count per page of each book (including different combinations for ‘log values’ of the newly generated variables)  Note: We also remove all ‘string’ values for we wish to evaluate a books rating which is generally expressed in numbers (‘string’ features deemed not influent)  Then a check is done to see if all our remaining values are numeric: books with non-numeric values in the dataframe are removed from it.  We now generate correlation plot(s) to get an understanding of our newly added variables.  At this stage, there are many highly correlated variables, although still none of them are highly correlated with the response variable.  Yet, we have created some variables that are more correlated with the response variable than some of those we had initially.  For example *log\_num\_pages* and *log\_r\_count\_per\_log\_day*, some of our “new variables” have higher correlation than some of the old ones.  **Model training & evaluation**  **#Model types training (and screening):**  Here, we use *regression models* because our response variable is numeric.  - We will now find out what model is the best and the most stable by running it through a series of random distributions of training and test data.  - The ‘scores’ (R² and MSE) for each model are stored in a list to be analyzed afterwards.  - After finding the model with the best score, we will take a closer look at that model and see if there are some insignificant explanatory variables that can be removed.  We can already assume that this will be the case, as we can see from the correlation plot that some of the variables we created by transforming other variables are highly correlated with each other, but not that correlated with the response variable.  We start by defining our models (as shown in the extract below):  def linmod(X\_train, X\_test, y\_train, y\_test) # linear regression  def polmod(X\_train, X\_test, y\_train, y\_test) # polynomial regression  def svrmod(X\_train, X\_test, y\_train, y\_test) # SVR model  def rafmod(X\_train, X\_test, y\_train, y\_test) # Random forest regression  def ridmod(X\_train, X\_test, y\_train, y\_test) # Ridge model  def lasmod(X\_train, X\_test, y\_train, y\_test) # Lasso model  def baymod(X\_train, X\_test, y\_train, y\_test) # Bayesian Ridge model  def lgbmod(X\_train, X\_test, y\_train, y\_test) # LightGBM model  We then distinguish the training and response features and import necessary libraries.  Next we define the train and test of all the previous models  We found that polynomial model, svr model and the random forest model showed the lowest score, so we remove them and focus on the other models.  We will look for the ***adjusted R²*** values now because, this measures the variation of a range of regression models  For the R² we will start by looking at the *mean values of the R² list* and then the *mean values of the MSE list*   * ***LightGBM model has the highest R² value***   We look at it for the latest *random\_state* distribution seed for the training data and test data distributions. This can be done as we don't know how this seed compares with the other ones for this model, so our analysis of the model in this case is unbiased.  ***Using p-value statistical indicator*** :  # Calculate p-values using statsmodels (as shown by the extract below):  P-values sorted from largest to smallest:  log\_t\_count\_per\_year 9.977301e-01  log\_t\_count\_per\_day 9.977301e-01  log\_num\_pages 9.601149e-01  days\_since\_published 6.968271e-01  ratings\_count\_per\_num\_pages 6.874249e-01  log\_r\_count\_per\_log\_num\_pages 6.272965e-01  ratings\_count 6.183681e-01  Name: P>|t|, dtype: float64  We see that *log\_t\_count\_per\_year* has the highest p-value, meaning it is the least significant variable for our model. We will nor try to simplify the model until all variables have a significance level of 90%  As the variables are correlated with each other to some degree, we can only remove one at a time as the p-values will potentially change for the other variables when one variable is removed.  Example:  X = X.drop(['log\_t\_count\_per\_year'], axis=1)  Then re-run train\_test\_split 🡪 R2 and MSE lgbmod estimation   * *# Calculate p-values using statsmodels*   And so on… We continue the procedure of removing the variable with the highest p-value and then re-calculating the p-values  After a number of trials, we reach a number of remaining features = 9  Extract:  P-values sorted from largest to smallest:  log\_r\_count 2.088402e-02  log\_t\_count\_per\_log\_num\_pages 8.519018e-03  text\_reviews\_count 1.319774e-03  log\_r\_count\_per\_log\_day 9.373036e-04  log\_t\_count 2.052018e-04  log\_t\_count\_per\_log\_day 4.901342e-05  is\_eng 8.751255e-08  log\_r\_count\_per\_num\_pages 5.571895e-12  num\_pages 5.984906e-49  const 0.000000e+00  Name: P>|t|, dtype: float64  We see here that the p-value decreases relatively much while at the same time, some of the remaining variables become less significant. It is therefore natural to stop here, so to not simplify the model into one that is not underfitted. So we keep the rest of the variables remaining and plot their correlation.  If we compare this with the correlation plot, we see that we have some more variables with a relatively high correlation with average\_rating than what we had in the beginning, ref. the correlation plot of df[["average\_rating", "num\_pages", "ratings\_count", "text\_reviews\_count"]]  We will continue by training and testing this latest obtained correlated dataset to get the MSE and R². For a better visualization of outliers, the distribution and the count of the observations of the R², we will draw a histogram (same for MSE)    **#Exporting using the pickle and joblib library**  The process of using pickle is called “Pickling”: a process of serializing a python object structure by converting the underlying object hierarchy into a byte stream. What’s not so great about pickling is that the resulting bytestream is hard to inspect unless unpickled (or generated using the oldest Protocol, v0). It also represents a potential security risk as a pickle could contain malicious code, and an untrusted pickle file opened without precautions could lead to naughty code being arbitrarily executed.  joblib extends pickle by supporting compression helping serialize objects a bit more efficiently. SO we will proceed by using Joblib.    **# Deployment with Pickle** | | | | |
|  |  | | | | |