Course: Python Machine Learning Labs

Project: Develop an end-to-end Machine Learning Pipeline

Project Team:

- Thanh-Cong
- Arabi TEFFAHI
- Eya MIDOUNI
- Maryam TEYMOURI
- Valentina RODRIGUEZ

Project Github:

https://github.com/thanhvo-uparis/ML_pipeline

Introduction

In today's digital age, books continue to be a significant source of knowledge, entertainment, and exploration. With the proliferation of online platforms and e-commerce websites dedicated to books, understanding the factors that contribute to a book's success and popularity has become a topic of great interest. This project aims to leverage the power of machine learning to predict a book's rating based on a range of relevant features, thereby providing insights into the elements that influence a book's reception by readers.

Objective

The primary goal of this project is to develop a predictive model that estimates the rating of a book based on features. By analyzing a dataset containing information about various books and their corresponding ratings, we aim to uncover patterns and relationships that influence how books are perceived by readers.

Dataset

The foundation of this project lies in a comprehensive dataset collected from Goodreads platforms. This dataset includes a variety of attributes for each book.

- variable target : average rating
- rows and columns: 11123, 12
- variable types: qualitative: 5 , quantitative : 7
- Missing values analysis : 0

Data Cleaning

One of the issues we faced involved four specific lines within our data-frame where a comma couldn't be used as the proper delimiter for separating data. To address this problem, we used the parameter **error_bad_lines=False** which allowed us to skip these problematic lines during data loading.

After examining our dataset, we identified several features that require cleaning, such as, for example: publisher, publication date, authors, language code.

Publisher: After investigating the "publisher" column, it has come to our attention that there is a notable issue of high cardinality, referring to a large number of unique values in this column. This situation requires mitigation to facilitate effective data processing.

In the upcoming preprocessing phase, we will address this high cardinality concern in order to streamline the data.

Publication date: After a brief investigation of the data, it was discovered that there were errors in the publication dates of certain books. Specifically, two of the problematic dates was " '11/31/2000': '12/01/2001', '6/31/1982': '07/01/1982' ". These inconsistencies were addressed by identifying and correcting such erroneous dates. This quality control process ensured that the dataset's publication dates accurately represented valid calendar dates.

Language code: In the analysis of book data, we explored language codes using a series of steps. We started by creating a list of unique language codes present in the dataset. This step highlighted similarities among certain languages, leading to the creation of a mapping dictionary to group similar codes. For instance, different English variations were grouped under the common code 'Eng'. Next, we aggregated data based on these grouped language codes, calculating median ratings and associated book counts. We visualized this information using a bar chart, where the color of each bar corresponded to the median rating. Finally, we suggested a feature engineering technique to merge languages with few books into a single category, potentially enhancing data quality and future analyses.

Exploratory Data Analysis (EDA)

Analyses numerical data

Upon checking the correlation matrix, a strong correlation of 0.87 was observed between the "rating count" and "text review count" variables. This correlation was also highlighted by examining the histograms of both these variables, revealing a similar distribution shape in both cases.

For the histogram of average rating, the majority of the data is concentrated around 4. This concentration of data around 4 explains why, upon examining the statistics of our dataset, the average rating is calculated to be 3.934919. We found out that the average rating is mostly positive.

The number of pages has a correlation of 0.15 with the average rating. Additionally, it's worth noting that 75% of the data has a page count of up to 416 pages, while approximately 25% of the data falls within the range of 416 to 752 pages.

```
books_clean.describe()
```

	book_id	average_rating	isbn13	num_pages	ratings_count	text_reviews_count	day	month	year	numberOfAutho
count	10985.000000	10985.000000	1.098500e+04	10985.000000	1.098500e+04	10985.000000	10985.000000	10985.000000	10985.000000	10985.0000(
mean	21256.653437	3.934919	9.759615e+12	336.598999	1.811101e+04	547.355394	11.271825	6.546017	2000.154483	1.5720!
std	13105.889815	0.348383	4.457435e+11	241.372953	1.131827e+05	2592.066785	10.286820	3.411521	8.269247	0.8476(
min	1.000000	0.000000	8.987060e+09	0.000000	0.000000e+00	0.000000	1.000000	1.000000	1900.000000	1.0000(
25%	10206.000000	3.780000	9.780345e+12	192.000000	1.050000e+02	9.000000	1.000000	4.000000	1998.000000	1.0000(
50%	20051.000000	3.960000	9.780571e+12	300.000000	7.520000e+02	47.000000	8.000000	7.000000	2003.000000	1.0000(
75%	32078.000000	4.130000	9.780872e+12	416.000000	5.052000e+03	242.000000	20.000000	9.000000	2005.000000	2.0000(
max	45641.000000	5.000000	9.790008e+12	6576.000000	4.597666e+06	94265.000000	31.000000	12.000000	2020.000000	6.0000(

Analyses qualitative data

It can be observed that the English language constitutes 95.4 percent of all languages in the dataset, and there are 10 languages that are represented by only one book each.

The distribution cardinality of the authors is significant.

Upon examining the "publisher" column, a significant challenge arises due to its high cardinality, indicating a substantial number of distinct values. This issue needs to be addressed to enhance data processing efficiency. In the forthcoming preprocessing stage, we will tackle this high cardinality problem to optimize data handling.

Relation between numerical variables and target

The relationship between page numbers and average ratings is evident from the plotted data, showcasing shifts between Gaussian curves for each review type.

An interesting finding from the ratings counts plot is that reviews rated as "Poor" tend to have significantly lower or almost negligible rating counts compared to those rated as "Excellent."

The first day of each month stands out with a prominent peak in the Gaussian-like curve, suggesting potential links to increased consumer activity following salary receipt.

The peak observed in the ninth month can be attributed to strategic book releases aligning with the autumn season and year-end holidays. This temporal alignment leads to a surge in positive reviews, capitalizing on festive readership and new literary works' availability. This phenomenon underscores the interplay between seasonal releases and reader engagement patterns.

Preprocessing and featuring engineering

Publication_date: converts the 'publication_date' column to a datetime format and then extracts and assigns the day, month, and year components into separate columns .

BookType: We added a new variable to distinguish between individual books and series. For title adjustments that preserve page count outliers, we utilized the **check_book_type** function. This function examines title patterns and keywords related to multi-book titles, assigning '1' to the 'bookType' if such indicators are found, and '0' if not.

authors*:* we found that many times the column was a combination of author + editor or author + contributor. Therefore, we extracted the name of the first author since we considered him to be the most important. At the same time, we created new columns to find out if it was a single author or if it was a collaboration and from how many people.

language code: Languages containing fewer than 15 books in our dataset were classified under the label "Other." Additionally, we created a new binary variable for each language. This transformation of a categorical variable into binary format enhances the utility of our model.

Substituting Zero Page Counts with Medians for Accuracy in Page Numbers.

Add the Genre Column: extracting valuable information by processing a JSON file containing book genre data. The code reads a Gzip-compressed JSON file, decodes each line as UTF-8, loads the JSON content, and compiles it into a list. This list of JSON objects is then transformed into a pandas DataFrame, enabling us to analyze and manipulate the genre-related information effectively.

Authors: We introduced a new binary variable named "collaboration" to indicate whether a book was written by a single author (denoted as 0) or if it's a collaboration (denoted as 1).

Add the publisher_class: We calculated the number of books per publisher, assigned classification labels ('Rare', 'Medium', 'Famous') based on the counts, and mapped these labels to publisher names in the 'publisher_class' column.
Remove unnecessary columns: publisher, book_id, title, authors, isbn, isbn13, language_code, publication_date, numberOfAut.

Motivation for the feature selection

Considering all the data exploration we did, we found that the best feature selection is these features.

```
Index(['average_rating', 'num_pages', 'ratings_count', 'text_reviews_count',
       'day', 'month', 'year', 'bookType', 'lang_code_new', 'type',
       'collaboration', 'publisher_class'],
      dtype='object')
```

First we wanted to reduce all the complexity for the model, the complexity started with the categorical variables with high cardinality so we tried to make it simple and catch the maximum of the patterns that are related to the average rating column which is our target variable. our objective is to make a model that is clean as much as possible and anyone can use it as we saw in the data exploration that we don't have a lot of variability between categorical variables and our target variables like type, publisher class , language code, beside there is no really linear relationship between the numerical variables and our target variable here we starterd thinking about a model that is not linear so we thought about lasso regression, xgboost and randomforest.

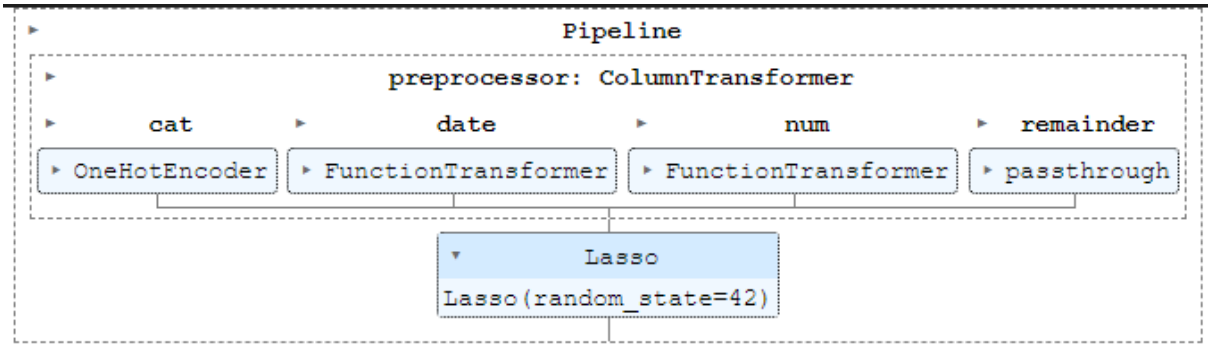
Model strategy

First, we used an a Scikit-learn pipeline so we can make it easy for everyone to train again the model and make the changes according to our needs. The pipeline is to one hot encode every categorical variable, simpler impute the day month and year and then scale and transform the numerical variables. We add a parameters grid for all the models.

Then, we split the train set into train, validation, test set and also we use the cross validation to get the best of the parameters (hyperparameter tunning).

Finally, we use three models to compare (xgboost, randomForest and lasso regression).

This is an example of the pipeline.

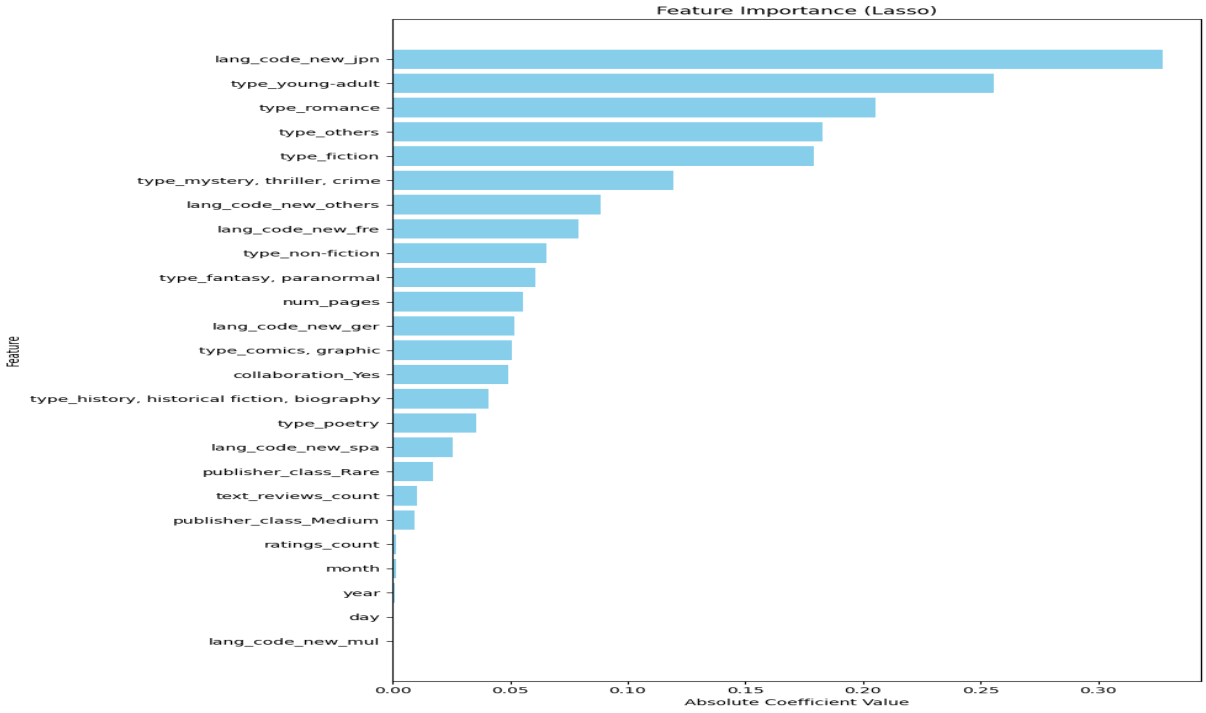


Model selection

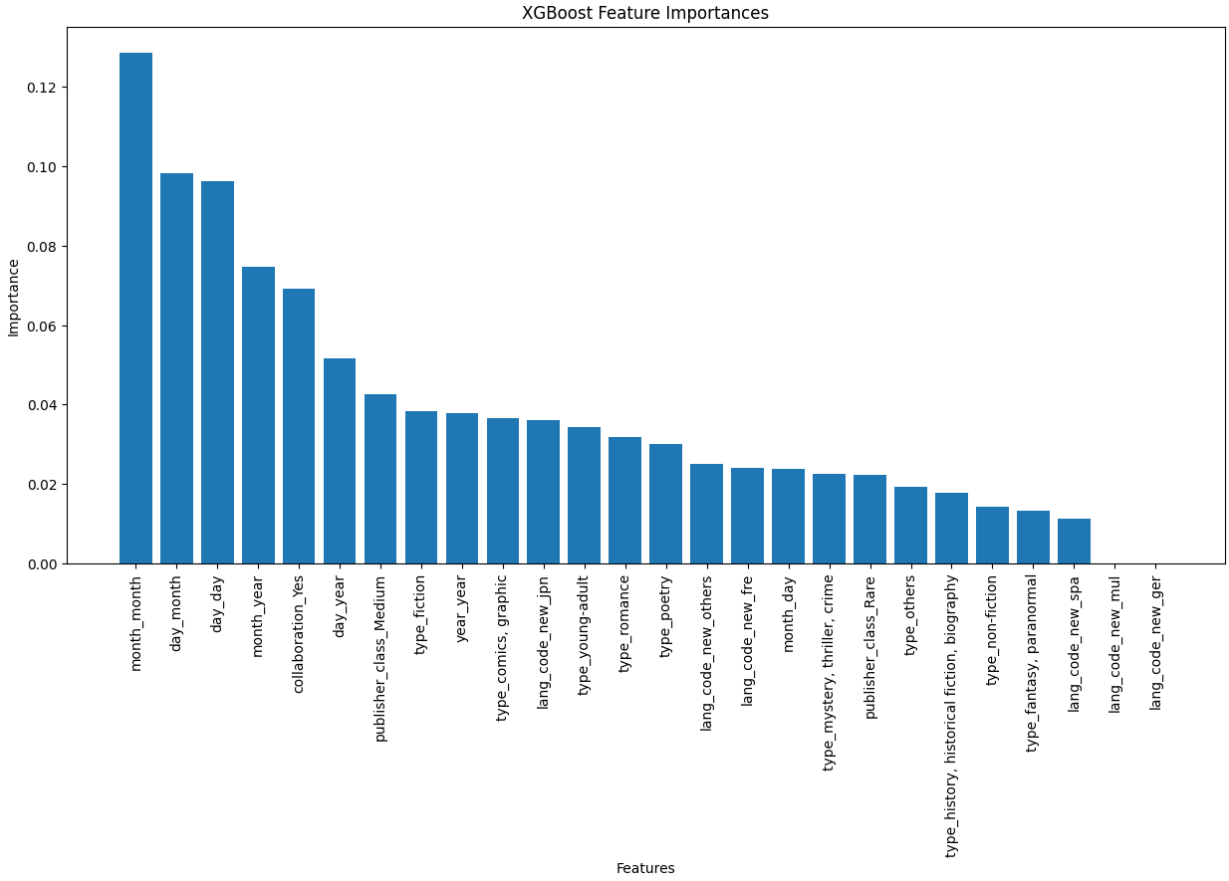
We tested the three models on different parameters and lasso is always the champ model, proving our first logic was right with 0,108 MSE for the test set.

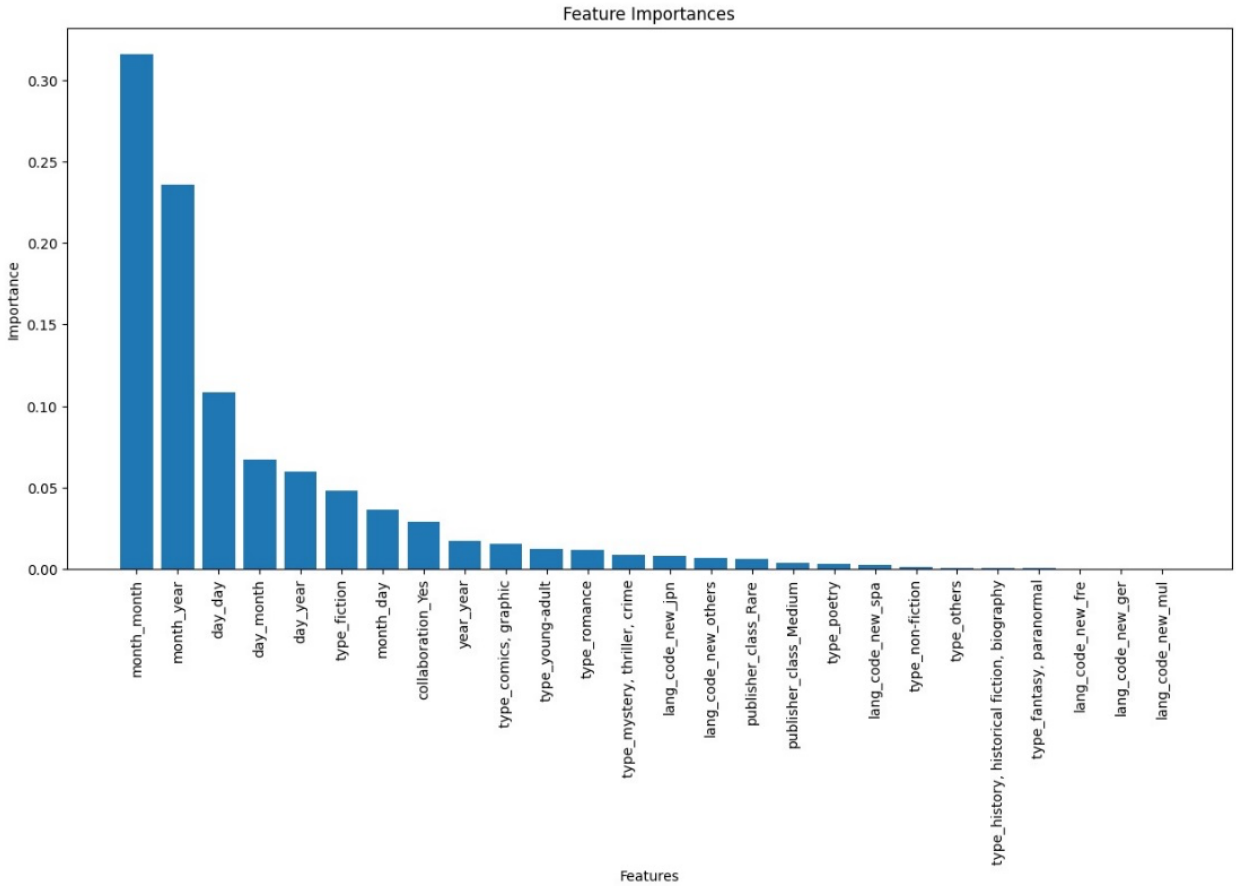
```
0.0s

Test MSE for lasso: 0.10865071921479616
Test MAE for lasso: 0.217752589930287
```

Here it is the feature importance plot:



We have choose the lasso regression, short for "Least Absolute Shrinkage and Selection Operator, because first it have the most important metrics besides the features importance are more logic than the other two xgb and random forest here are the feature importance of both models.


XGBoost Feature Importances
Importance
0.00
0.02
0.04
0.06
0.08
0.10
0.12
month_month
day_month
day_day
month_year
collaboration_Yes
day_year
publisher_class_Medium
type_fiction
year_year
type_comics, graphic
lang_code_new_jpn
type_young-adult
type_romance
type_poetry
lang_code_new_others
lang_code_new_fre
month_day
type_mystery, thriller, crime
publisher_class_Rare
type_others
type_history, historical fiction, biography
type_non-fiction
type_fantasy, paranormal
lang_code_new_spa
lang_code_new_mul
lang_code_new_ger
Features


Feature Importances
Importance
0.00
0.05
0.10
0.15
0.20
0.25
0.30
month_month
month_year
day_day
day_month
day_year
type_fiction
month_day
collaboration_Yes
year_year
type_comics, graphic
type_young-adult
type_romance
type_mystery, thriller, crime
lang_code_new_jpn
lang_code_new_others
publisher_class_Rare
publisher_class_Medium
type_poetry
lang_code_new_spa
type_non-fiction
type_others
type_history, historical fiction, biography
type_fantasy, paranormal
lang_code_new_fre
lang_code_new_ger
lang_code_new_mul
Features

Conclusion

From the data we have we can conclude that we need to have more variables to predict exactly the average rating for every book, more lines and more variables will add more variability to the average rating depending on the other variables which can make more models to capture more of the variability and the patterns of the target variable average rating.

Recommendation

It's better to use spectral embedding between the authors like this we can captures the combination of authors that explain the average rating.

We suggest trying more hyper parameters ant test more models that are not linear models.

Project hosting and deployment

The provided code snippet implements a Flask API that serves as an interface for making predictions using a machine learning model stored in a pickled file. The Flask app exposes an endpoint '/predict' that accepts POST requests containing JSON data. Upon receiving a request, the app loads the pre-trained machine learning model, processes the incoming data, and returns a prediction as a JSON response.

Functionalities of the code:

1. **Importing Libraries:** The necessary libraries are imported at the beginning of the code, including Flask for creating the API, pandas for data manipulation, pickle for loading the model, and requests for making HTTP POST requests.
2. **Defining Flask App:** An instance of the Flask app is created using app = Flask(__name__).
3. **Loading Model and Defining Functions:**
    - The machine learning model is loaded from the 'best_model.pkl' file using the pickle module.
    - Two functions are defined: simple_imputer for filling missing values with the median and scale for standardizing data. These functions are not used in the provided code, but they seem to be intended for data preprocessing.
4. **'/predict' Endpoint:**
    - The endpoint '/predict' is defined as a route that accepts only POST requests.
    - The do_prediction() function is defined to handle the prediction process.

- Within this function, JSON data is obtained from the request using request.get_json().
- The JSON data is converted into a pandas DataFrame, and the loaded machine learning model is used to make a prediction on this data.
- The prediction result is returned in a JSON format as a response.

5. **Error Handling:**
   - The code includes a try-except block to handle potential exceptions that might occur during the prediction process. If an exception occurs, an error message is returned in a JSON response.
6. **Running the Flask App:**
   - The Flask app is run with app.run(host='0.0.0.0', port=port), making it accessible at the specified host and port.
7. **Making a Prediction Request:**
   - An example of making a prediction request is provided using the requests library. A sample data point from a DataFrame (train.iloc[0].to_dict()) is converted into a dictionary and sent as JSON data in a POST request to the '/predict' endpoint.
   - The response from the server is printed.