Artificial Intelligence Master Degree

Big Data Course

Classical and PySpark-ML Recommendation Systems

Enrico Zorzi (VR495467), Gaetano Alberto Caporusso (VR489760)

1 Introduzione

1.1 What is a Recommendation System?

Recommendation systems are filtering systems used to suggest products or services to customers based on their previous choices and the preferences of similar users. Major companies like Amazon, YouTube, and Spotify use those systems to propose movies, books, music, and other items. The concept behind these algorithms is to find patterns in consumer behavior or similar consumers' behavior towards a service or product.

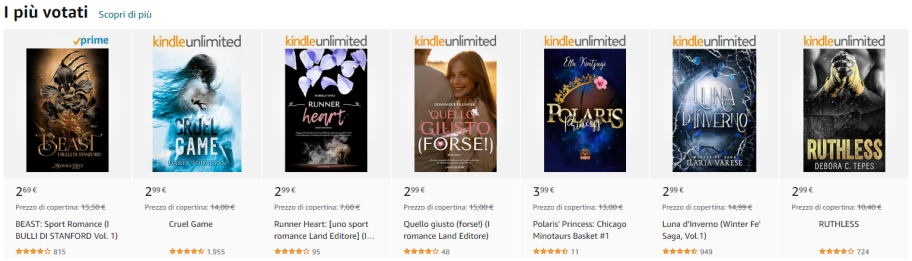


Figure 1: Most voted book on amazon

1.2 Benefits of Recommendation Systems

Recommendation systems offer various benefits to sellers by keeping users engaged with their website, increasing sales, and acting as a powerful marketing strategy. They provide users with a customized experience, increasing satisfaction and loyalty.

2 Pipeline

The development of a recommendation system involves several common steps:

1. Collect data, such as reviews on Amazon and Booking or likes and views on YouTube.
2. Filter the data, removing troublesome values.
3. Choose and fit the model or algorithm.
4. Evaluate and test the model, and eventually tune it based on performances and metrics.
5. Deploy the model.

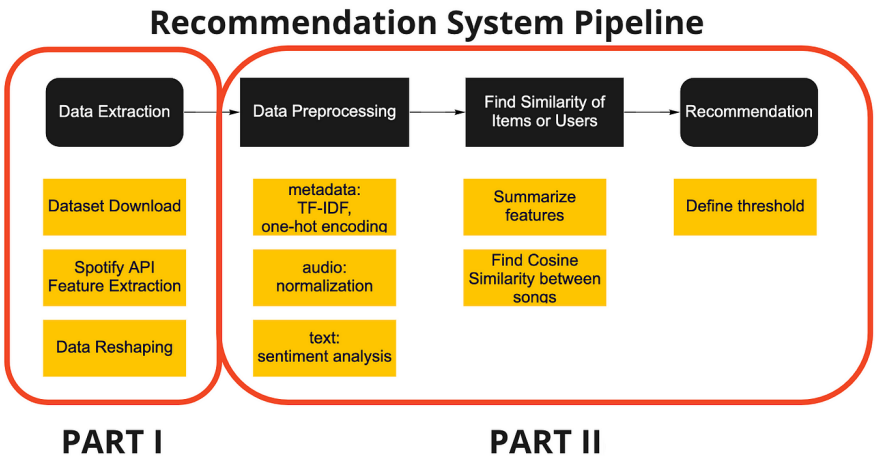


Figure 2: Pipeline

3 Types of Recommendation Systems

3.1 Collaborative Filtering

This approach is based on the preferences and behaviors of similar users to predict future interactions. Consumers who share an overlap of similar interests will more than likely be interested in other similar products, joining a cluster of other people with similar tastes and seeing content based on historic choices.

3.2 Content Filtering

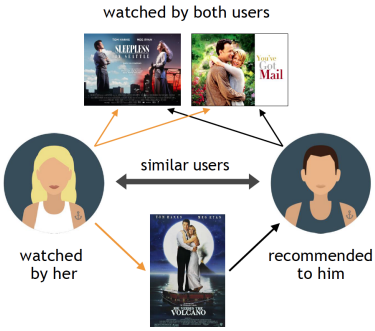
Content filtering is based on item features, suggesting products with similar attributes to items you previously liked or bought. For example, if the last three watched movies included the comedy genre, the system will recommend other similar comedy movies or shows.

3.3 Hybrid Filtering

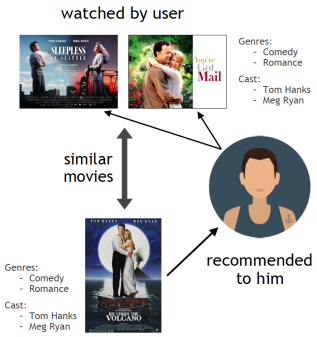
Hybrid filtering combines both Collaborative and Content Filtering advantages, creating a more complex but accurate system.

3.4 Context Filtering

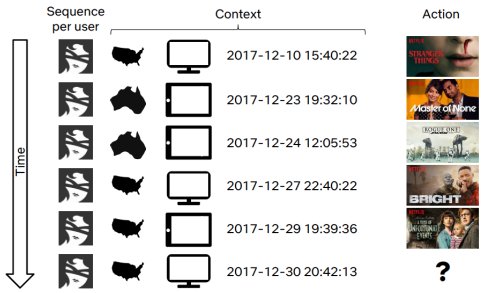
Context filtering considers users' contextual information such as device, country, date, and time, training a model to predict future choices.



(a) Collaborative filtering



(b) Content filtering



(c) Context filtering

4 Spark

Apache Spark is a strong open-source analytics engine built to process large-scale data. It allows programming entire clusters with implicit data parallelism and fault tolerance. One of the primary features of Spark is its incredible speed in processing huge volumes of data by utilizing in-memory computing as well as a rich set of libraries for SQL, machine learning, graph computation, and stream processing.

PySpark stands for Python Spark, allowing Python developers to use Spark to its full potential. It gives a way of running Spark on Python modules such as pandas, NumPy, and scikit-learn seamlessly among other libraries provided by Python.

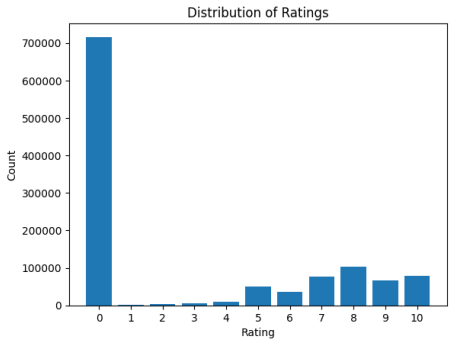
PySparkML is a library that integrates Spark's high processing capabilities with ML algorithms and tools. It allows the building, tuning, and deployment of complex machine learning models, crucial for big data analysts.

Spark and PySpark collectively offer an efficient and scalable data processing framework for data engineering and data science with Python's simplicity and Spark's speed.

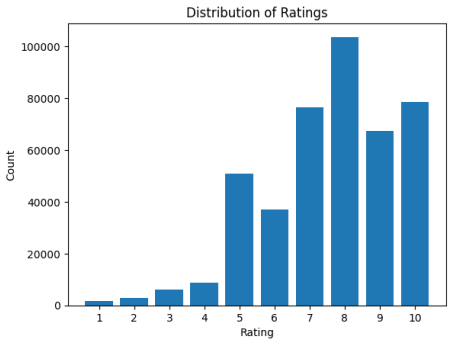
5 Dataset

Our goal is to compare two recommendation systems, one is based on classical techniques and the other on sparkML library. We use a books and ratings dataset from amazon reviews, which contains explicit ratings, users

provide ratings or direct feedback on items, explicitly indicating their preferences, and implicit ratings, collected from user's implicit behavior, such as clicks, purchases, time spent on pages, viewing history. These behaviors can indicate the user's interest in an item, even if there is no direct rating or feedback. Implicit interactions may not provide enough detail about the user's specific preferences, making recommendations less personalized. Implicit ratings are represented by the zeros in our dataset. Since those compose the majority of the datas, they can badly affect our final recommendations, so we decided to delete them all, keeping the explicit ratings only.



(a) Implicit + Explicit



(b) Explicit

The dataset is composed by two files:

- Book (340556 items)
  - ISBN
  - Title
  - Author
  - Year of Publication
  - Publisher
- Books-Ratings (1149780 ratings)
  - User-ID
  - ISBN
  - Book-Rating (0 to 10)

ISBN	Book-Title	Book-Author	Year-Of-Publication	Publisher
0195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University...
0002005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Ca...

(a) Book

User-ID	ISBN	Book-Rating
276725	034545104X	0
276726	0155061224	5

(b) Books-Ratings

6 Classic Recommendation System

We first built a user-book matrix which, with a full size dataset, requires the allocation of 72 GB of RAM. This is not achievable on our devices, so we need to apply some preprocessing to our data. We have removed the implicit ratings, ensuring the consistency of data keeping all the explicit ratings of books which are in the books' file also. Furthermore, we have kept 200k ratings only and removed the books with less than 2 ratings for memory purposes in the generation of a user-book matrix. Cosine similarity is computed on this matrix and then the item score for each user is obtained multiplying his interaction row of the matrix with cosine similarity. Now we have a score for each book and we can order those to obtain the top user recommendations. In the end, we compute the precision of the algorithm, considering a certain tolerance between prediction and ground-truth.

6.1 Cosine similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space based on the cosine of the angle between them, resulting in a value between -1 and 1. The value -1 means that the vectors are opposite, 0 represents orthogonal vectors, and value 1 signifies similar vectors.

$$similarity = cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \, \|\mathbf{B}\|}$$

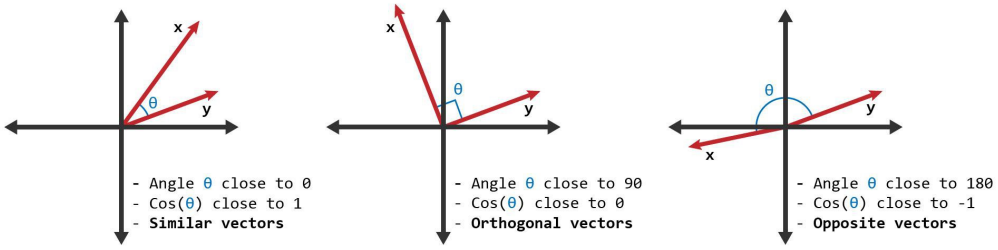


Figure 6: Cosine similarity graph

Overall, cosine similarity is preferred because it effectively measures similarity in a way that handles sparse data, normalizes for different user behaviors, is computationally efficient, and is easy to interpret.

7 PySparkML Recommendation System

We have realized a recommendation system using PySparkML. As we said before, PySpark is built to process a huge amount of data, so we kept most of the records, except for the implicit ratings, to have the same kind of data as the previous system. We have used an ALS(Alternating Least Square) model, randomly splitting the dataset in 80% train set and 20% test set. ALS models accept 3 parameters: rank, maxiter, regParam. A CrossValidator with a ParamGridBuilder has been used to fine-tune the ALS model, specifying a grid of hyperparameters and using cross-validation(3-fold) to determine the best model. After fitting the model and obtaining the predictions, we have evaluated it with RMSE (Root Mean Squared Error). The precision is computed in the same way as the previous system.

7.1 ALS (Alternating Least Square)

Is an algorithm widely used in the implementation of recommender systems based on Collaborative Filtering. It approaches the recommendation problem as a factorization matrix, where the ratings given by users to different items are represented as a sparse matrix. it realizes a User-Books matrix, like the one used for the classical system, with the objective of finding two smaller matrices whose product is as close as possible to the original matrix. The best results are found minimizing the sum of the squares of the error between the predictions and the actual evaluations.

$$Loss = \sum_{(i,j)\in\Omega} (r_{ij} - \hat{r}_{ij})^2$$

The hyperparameters used are:

- **Rank**, which specifies the number of features extracted, indicating the complexity of patterns captured by the model
- **Maxiter**, numbers of ALS iterations
- **RegParam**, regularizations parameters applied to the matrix to avoid overfitting

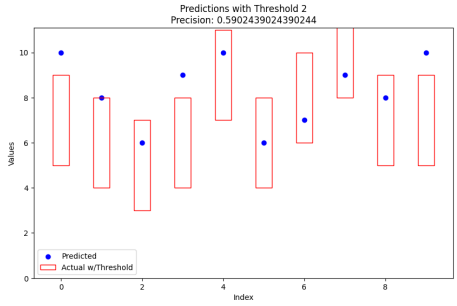
Furthermore, to build the ALS model we needed to convert the ISBN to integer values, using the function StringIndexer, obtaining for each ISBN a unique index.

8 Comparison between systems

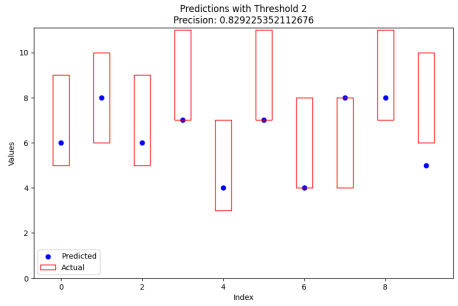
We decided to compare our systems based on precision, computational time and memory allocated.

8.1 Precision

We calculated the precision as the percentage of correct predictions.Since the ratings have a range from 0 to 10, we are able to compute the precision given some tolerance too. For example, if the prediction is 6 and the tolerance is 1, it will be considered as correct even if the ground-truth is 5 or 7. We can see below the precision of the two systems (classical on the left and ML one on the right) with the predictions of 10 random samples.



(a) Classic precision



(b) PySparkML precision

Setting the threshold(tolerance) equal to 2, the Classical System's precision is 0.59 and the PySparkML one is 0.82.

8.2 Computational Time

We have checked for the most significant differences in the execution time of the most time-consuming part of both systems' code. The classical method takes about 20 seconds for building the user-item matrix, 49 seconds for computing the cosine similarity and just 1 millisecond for a single user recommendation. The only relevant time consuming step in the PySparkML one, it's the training time, which takes about 1 minute and it's done on the whole dataset without the implicit ratings, so on about 433'000 rows. Instead, the Classical System is computed on 131'000 ratings.

8.3 Memory and CPU usage

We have checked the RAM allocation and CPU use in both systems. In the classic system, the 2 csv files of the dataset occupy 100 MB on the RAM, after the preprocessing phase. Instead, the user-book matrix occupies 6 GB. The computation of the cosine similarity is mostly a CPU-based operation, in fact, the CPU usage goes from 3% to 87% during the execution(1 minute) and the RAM usage goes from 12 GB to 17 GB. We can't check the RAM usage properly because during the execution of the code there are other processes running on the device. All these datas have been collected using the libraries psutils, for RAM and CPU, and a sys' method for the dataframes sizes. On the PySparkML system, the dataset and the model occupy about 25 MB of RAM each. We have monitored the RAM and CPU variations for the whole execution of the code, we haven't noticed any relevant increment of those like in the classic system. We have used

the Spark UI to check the RDDs size in the memory and a parallel python process to have real time RAM and CPU usage bars.

9 Conclusion

At the end of our project, we can easily conclude saying that the use of PySpark, with machine learning methods, is way more efficient then the classic one, considering precision of the model, computational time (PySparkML model has been executed on bigger dataset), allocation of the memory and CPU usage. A future improvement of our project could be comparing machine learning methods with deep learning ones on recommendation systems, trying to mainly improve the precision of the model.