Big Data Project Report

Master ADEO 2

Title:

Movies Average Vote Prediction

BY

Azza Kamoun

Taught by

Dr.Madan Radhakrishnan

Table of Contents

0- Abstract **3**
1- Introduction **4**
1.1- Problem Statement 4
1.2- Approach 4
1.3- Dataset and its Attributes 6
1.4- Libraries Imported 6
2- Exploratory Data Analysis (EDA) **8**
2.1- Data Preprocessing 8
2.1.1-Movies_metadata Table 8
2.1.2- Credits Table 9
2.2- EDA 9
3-Data Preprocessing (For Modeling) **10**
1.1- Movies_metadata Table 10
1.2- Credits Table 11
1.3- Keywords Table 11
1.4- Final Joined Table: 12
4-Data Modeling **13**
4.1- Data Reading 13
4.2- One-Hot Encoding 13
4.3- Correlation Analysis 13
4.4- K-means Clustering using Elbow Method 15
4.5- Random Sampling (Test & Train) 17
4.6- Regression Models 18

0- Abstract

Our project aims to predict the average vote of movies based on different relevant attributes. This can help movie production companies predict how well their film will perform based on its features, and it can make decisions backed by data even before starting the project itself. We trained our models on a dataset of movie attributes such as budget, revenue, popularity, runtime, and so on. Before starting our regression, we performed an exploratory data analysis (EDA) on the MovieLens dataset to provide insights into the different features and characteristics of the movies. The **EDA** involved analyzing various attributes such as the number of movies released per year, the distribution of movie ratings, the most common genres, directors, and keywords.
After understanding the most common patterns and stories from our data, we prepared it for the regression model using **data cleaning** and **preprocessing techniques** (feature engineering and selection, duplicates handling, missing values handling…). We also verified if there is any need for **clustering** it to choose the adequate way to split it into training and test sets (**random sampling** or **stratified sampling**). This was performed using **K-means clustering** technique while optimizing the optimal K using the Elbow method.
We used **Simple Linear Regression, Generalized Linear Regression, Random Forest, Gradient-Boosted Trees** and **Decision Trees** to model our data. We then evaluated their performances using various metrics like **R squared (R^2),Mean Absolute Error (MAE),** and **Root Mean Squared Error (RMSE)**. We finally ended the work with some manual **fine-tuning** of the hyperparameters to optimize the different metrics. It is worth noting that bothe EDA and Regression works were performed using the libraries available in **Pyspark** and the codes were run on Databricks.

Keywords:
Average vote of movies, exploratory data analysis (EDA),data cleaning, preprocessing techniques ,clustering , random sampling ,stratified sampling, Simple Linear Regression, Generalized Linear Regression, Random Forest, Gradient-Boosted Trees , Decision Trees, R squared (R^2),Mean Absolute Error (MAE), Root Mean Squared Error (RMSE),Pyspark ,Databricks, K-means clustering, Elbow method

1- Introduction

1.1- Problem Statement

Predicting the potential success of a movie is very important for the producers as well as the investors; and the whole team. Two main indicators are likely to determine whether or not a film is successful: the revenue, and the average ratings. The first is a crucial metric for the team itself, while the second is important for both the team, and the consumer. The latter have the tendency to look up how well a movie is rated before watching it to avoid any potential disappointment.
After further analysis, I found out that the revenue data is extremely noisy, and barely predictable, on the contrary of the average rating, hence our choice of the target variable. As a result, our work will be aiming to predict the average vote for movies.

1.2- Approach

The project approach can be structured as follows:

1- Data Collection and Preparation:

- Identify and collect the relevant dataset for the project.
- Load the dataset into the workspace using pandas library.
- Explore the dataset using descriptive statistics.

2- Exploratory data analysis (EDA) :

- Analyzing around 20 KPIs to track various factors such as the number of films released each year/ month, the distribution of film ratings, the most popular genres, directors, actors and keywords, in order to create a proper story from our data.

3- Data Preprocessing:

- Handle missing values by either dropping the rows or imputing them with a suitable value.
- Outlier Detection and Removal: This step entails identifying and eliminating any data points that differ significantly from the rest of the data. Outliers in the budget and revenue columns are identified and removed from this dataset using the interquartile range (IQR) method.
- Converting categorical data into numerical data using one-hot encoding.
- Data Scaling: In this step, numerical data is scaled to ensure that all features are on the same scale. This is done to prevent any feature from dominating the analysis. The budget and revenue columns in this dataset are scaled using the scikit-learn library's MinMaxScaler.

- Data Transformation: This step entails converting data into a more suitable format for analysis. Some columns containing JSON objects, for example, the cast and crew information in the credits.csv file, are flattened into separate columns. Additionally, some categorical variables, such as the genres column in the movies metadata.csv file, are encoded using one-hot encoding.
- Feature Engineering is the process of developing new features that may be useful for analysis. In this dataset, for example, the release date column is separated into columns for year, month, and day. The budget and revenue columns have also been converted from strings to numerical values.
- Check if clustering the data can be useful (K-means Clustering optimized using Elbow Method)
- Split the data into training and testing datasets (Random Sampling).

4- Model Development:

- Choose a suitable algorithm for the movie rating prediction task (Simple Linear Regression, Generalized Linear Regression, Random Forest, Gradient-Boosted Trees and Decision Trees).
- Train the model using the training dataset.
- Fine-tune the model by adjusting the hyperparameters.
- Evaluate the model's performance using various evaluation metrics (R squared (R^2),Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE)).

1.3- Dataset and its Attributes

The Full MovieLens Dataset is extracted from Kaggle, and contains metadata for 45,000 movies released on or before July 2017/ The dataset contains data on cast, crew, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts, and vote averages. Furthermore, the dataset includes files containing 26 million ratings from 270,000 users for all 45,000 movies. The ratings are on a scale of 1-5 and were obtained from the GroupLens website.

The dataset also consists of the following files:

- **movies_metadata.csv:** The main Movies Metadata file. It contains information on 45,000 movies featured in the Full MovieLens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.

- **keywords.csv:** Contains the movie plot keywords for the MovieLens movies. Available in the form of a stringified JSON Object.
- **credits.csv:** Consists of Cast and Crew Information for all movies in the dataset. Available in the form of a stringified JSON Object.
- **links.csv:** The file that contains the TMDB and IMDB IDs of all the movies featured in the Full MovieLens dataset.
- **links_small.csv:** Contains the TMDB and IMDB IDs of a small subset of 9,000 movies of the Full Dataset.
- **ratings_small.csv:** The subset of 100,000 ratings from 700 users on 9,000 movies.

In our project, we used the first 3 files to perform our analysis. Those csv files were imported into /FileStore/tables/ directory in databricks, which allowed us to read them later on.

1.4- Libraries Imported

The Pyspark package was used in our model pipeline to create a linear regression model for predicting the average vote of movies.

- **numpy**: a Python library that adds support for large, multi-dimensional arrays and matrices, as well as a large collection of high-level mathematical functions to operate on these arrays.
- **Pandas** is an open source data analysis and manipulation tool that is fast, powerful, flexible, and simple to use. It is built on top of the Python programming language.
- **matplotlib.pyplot**: a set of functions that makes matplotlib behave like MATLAB.
- **datetime**: is a Python module that contains several classes for working with dates and times. The module includes date, time, datetime… classes for working with dates and times.
- **plotly.express**: It is a Python high-level data visualization library that allows you to easily and quickly plot various types of graphs.
- **pyspark.sql.functions**: This module contains column-related functions.
- **plotly.graph objects:** is a package that contains classes for creating plots.
- matplotlib.ticker.
- **FuncFormatter** is a class in the matplotlib library that allows you to customize the tick values on an axis.
- **pyspark.sql.types.MapType** is a class that stores a key-value pair.
- **pyspark.sql.types.StructType, pyspark.sql.types.StructField, pyspark.sql.types.IntegerType, pyspark.sql.types.StringType,**

pyspark.sql.types.ArrayType : In Pyspark, these classes are used to represent the schema of the dataframe.

- **pyspark.ml.feature**: This module contains many feature transformation methods used in Machine Learning models, such as Imputer, StringIndexer, OneHotEncoder, VectorAssembler, StandardScaler, Bucketizer, PCA, and QuantileDiscretizer.
- **pyspark.ml**: This module contains Machine Learning algorithms such as regression, classification, clustering, and collaborative filtering.
- **pyspark.sql.functions.sum:** computes the sum of the values in a column in a PySpark DataFrame.
- **pyspark.sql.functions.col :** Using the col() function to select a column from a DataFrame.
- **pyspark.sql.functions.from_unixtime:** This function is used to convert a UNIX timestamp into standard date format.

2- Exploratory Data Analysis (EDA)

2.1- Data Preprocessing

2.1.1-Movies_metadata Table

➔ Missing Values

- **The function count missing values(df)** counts the number of missing values in each column of the input dataframe df. This function is then used to count the number of missing values in the ratings, links, credits, and movies metadata columns of various tables.
  Rows with missing values in the credits table's column 'crew' are dropped using the na.drop(subset=["crew"]) function. The columns 'production countries,' 'production companies,' and 'genres' in the movies metadata table contain data in Json format. There are many rows with the value '[]' in these columns. To avoid problems with Json parsing, empty values in these columns are replaced with default values via a user-defined function.
  Default values for columns 'production_countries','production_companies','genres' are: '[{'iso_3166_1': 'Unknown', 'name': 'Unknown'}]', '[{'name': 'Unknown', 'id': 0}]', and "[{'id': 0, 'name': 'Unknown'}]" respectively.

➔ Duplicates Values

- **The function count** duplicates(df) is used to count the number of duplicate values in the input dataframe df. This function is then used to count the number of duplicate values in the ratings, links, credits, and movies metadata tables.
- Rows with duplicate values in the credits table's columns 'cast','crew','id' and the movies metadata table's columns 'imdb id','title','release date','overview' are dropped using the drop duplicates function.

➔ Type Update

- Using the withColumn function, the data type of column 'rating' in the ratings table is changed to 'float'.
- Using the withColumn function, the data type of columns 'budget,' 'popularity,' and'revenue' in the movies metadata table is changed to 'integer' and 'float'.

➔ Parsing JSON fields

- The movies metadata table's Json object column 'belongs to collection' is parsed using a user-defined function. The Json object column is exploded, and the Json object's distinct keys are retrieved. The Json object's key values are retrieved, and a new column is created for each key value.
- The movies metadata table's columns 'production companies,' 'production countries,' and 'genres' have Json array values. To convert the list to column-separated values, a user-defined function is used. Because the data is in a Json array, extracting values based on Json keys yields a list.

2.1.2- Credits Table

- After defining the schema for the cast data and creating a user-defined function to update the data, we extract the cast data and convert it into a Spark DataFrame using the defined schema. The director information is also extracted from the crew data.

2.2- EDA

➔ **Analysis of Genres and Directors**

We analyzed the genres and directors in the dataset. The dataset's genres and directors are extracted, and a new dataframe is created to store this information. The dataframe is then used to generate two bar charts displaying the directors with the most and fewest films.

➔ **Actors and Movies Analysis**

We also analyzed the actors and films in the dataset. The actor data is extracted and exploded from the credits dataset**. After that, the movies are combined with the exploded actor data to form a new dataframe.** The new dataframe is then used to generate a bar chart displaying the actors with the greatest number of votes.

➔ **Gender Analysis**

The gender data from the actor dataframe is extracted, and a pie chart is created to show the distribution of male, female, and gender-unspecified characters in the dataset.

➔ **Successful / Unsuccessful Films Analysis**

We looked at the genres and countries with the most successful and unsuccessful films. The stat production countries dataframe, for example, is created by grouping the dataset by production countries and aggregating the sum of revenue and budget, renaming the sum of revenue and budget total revenue and total budget, and then adding a column profit that is the difference between total revenue and total budget. The table is then constructed by joining the stat production countries dataframe and the df prodcntry dataframe on the production countries column. We also created two word clouds depicting the top countries with the most successful films and the top countries with the most unsuccessful films.

➔ **Descriptive statistics**

We performed descriptive statistics on the input dataframe using the summary function. The selected columns for analysis are runtime, budget, revenue, vote_average, vote_count, and popularity. The function displays the count, minimum, 25th percentile, 75th percentile, and maximum values for each column.

3-Data Preprocessing (For Modeling)

This section will describe PySpark code for preprocessing data from the credits, keywords, and movies metadata tables. The code extracts pertinent information from these tables and joins it to form a new table containing a subset of the original data. It is made up of several PySpark DataFrame transformations and is designed to run on an Apache Spark cluster.

1.1- Movies_metadata Table

Using the Pyspark framework, we were able to retrieve pertinent data from the movies metadata table, which contains information about each movie.

- **Parsing JSON array column 'Belongs to Collection', 'production companies', 'production countries', and 'genre':** We parsed the different JSON object columns and retrieved the JSON object's distinct keys. We then converted the key collection object to a list and retrieved values into separate columns based on keys or converted the JSON column into string columns and separated by a comma.

- **Aggregating Data:** The revenue and budget columns are aggregated to determine the average, minimum, maximum, and total values.

1.2- Credits Table

The credits table is addressed in the first section of the code. The credits table contains information about each film's cast and crew. For each film, the code extracts the director's name and the top five actors.

- **Schema for Casting:** The cast field is a nested array of structures containing information about each cast member. The first step is to define this field's schema as an array of structs, with each struct containing the following fields: cast id, character, credit id, gender, id, name, order, and profile path.

- **Updating UDF Casting:** Some of the profile path values are "None" instead of "null" due to a bug in the data so we created a user-defined function (UDF) that replaces those fields for all profile path parameters.

- **Extracting Cast and Crew Information:** The "from json()" function is used to convert the cast field into a PySpark struct type that conforms to the schema defined in step 1. Each struct's name field is extracted using the getField() function. Using a regular expression, the regexp extract() function **extracts the director's name** from the crew field.

- **Extract Top Five Actors:** The explode() function is used to add a new row to the cast array for each element. To add a row number column for each group of id values, use the row number() function. Only the first 5 rows for each group of id values are selected using the filter() function. The function groupBy() is used to pivot the resulting actor. The resulting columns are renamed to actor_name1, actor_name2, actor_name3, actor_name4, and actor_name5.

1.3- Keywords Table

The keywords table contains data on the keywords associated with each film. The code retrieves the top five keywords for each film.

- **Keywords Schema**: The keywords field is a nested array of structures containing information about each keyword. The first step is to define this field's schema as an array of structs, with each struct containing the following fields: id, name.

- **Extract Keyword Information:** The from json() function is used to convert the keywords field into a PySpark struct type that conforms to the schema defined in step 1. The explode() function is used to add a new row to the keywords array for each element. To add a row number column for each group of id values, use the row number() function. Same as the actor names, only the **first 5 rows** for each group of id values are selected using the filter() function. The groupBy() function divides the resulting keyword.name values into 5 columns.

1.4- Final Joined Table:

→ **Final Table for Modeling :**

The tables are generally treated separately , or an occasional jointure between 2 of the tables is made for better analysis purposes.

The code's final section joins the preprocessed data from the credits and keywords tables with the data from the movies metadata table. The resulting table includes a subset of the movies metadata table's columns, as well as the top 5 actors and keywords for each movie.

The code first renames the "id" column in the movies metadata table to "movie id" before joining the tables. This is required because the "id" field in the credits and keywords tables represents the movie id, and we do not want any naming conflicts.

The code then joins the movies metadata and credits tables using the "movie id" field as the join key. This creates a new table with all of the columns from the movies metadata table, as well as some additional columns. The code then joins this new table to the keywords table, with the "movie id" field serving as the join key. This produces a final table containing all of the columns from the movies metadata table, as well as the top 5 actors and keywords for each film. The resulting table is then saved to a Parquet file for further investigation. The choice of

separating the preprocessing and the modeling of the data is intentional since too many rows in the notebook is likely to deteriorate its result rendering.

Joined table columns	Table
kw1 (keyword1)	Keywords.csv
kw2 (keyword2)	
Drama	movies_metadata.csv
Fantasy	
Family	
Foreign	
Action	
Unknown	
Documentary	
Animation	
Romance	
Thriller	
Western	
Science Fiction	
TV Movie	
Comedy	
Adventure	
War	
Music	
Horror	
Crime	
History	
Mystery	
revenue	
budget	
original_language	
release_date	
runtime	
video	
cntry_name	
vote_average	
vote_count	
actor_name1	credits.csv
actor_name2	
actor_name3	
director	

4-Data Modeling

4.1- Data Reading

After joining the tables from preprocessed data, the table is read from a csv file located at the path "/FileStore/tables/joined_data.csv" using Databricks spark csv reader. The following options are set:

- "multiline" option is set to True, which allows multiline records to be read.
- "header" option is set to True, which considers the first line of the file as a header.
- "escape" option is set to "" to escape the double quotes in the data.
- "inferschema" option is set to True, which infers the schema of the data.

4.2- One-Hot Encoding

'kw1', 'kw2', 'director', 'cntry name', 'actor name1', 'actor name2', 'actor name3', 'original language' are all encoded in one pass.

For each of these columns, StringIndexer and OneHotEncoder objects are created and used to create a pipeline object. The data is then transformed using the pipeline object.

4.3- Correlation Analysis

The corr() function of the pyspark.sql.functions module is used to calculate the correlation between the columns of the transformed data with average rating (target variable).
This will help select the variables which are highly correlated to the target variable.
The table below shows the correlation and absolute correlation values between various columns/features of a dataset. In this table, the column "corr" shows the correlation values between the column in the row and the "vote_average" column, while the column "corr_abs" shows the absolute value of these correlation values. The first features selected were those with correlation >5%. Then, as part of the fine tuning work, we added and deleted some of those columns from the training and test sets, which led to refining the list to the columns having a correlation >7%. Another alternative was also tested with starting from 10% and it also made my predictions worse. ID is also not included in the list of features as they are unique per movie.

col	corr	corr_abs
vote_average	1.000000	1.000000
id	-0.205556	0.205556
vote_count	0.176343	0.176343
Drama	0.150065	0.150065
runtime	0.126636	0.126636
original_language_index	0.124967	0.124967
Science Fiction	-0.119954	0.119954
Documentary	0.113859	0.113859
Horror	-0.106239	0.106239
Animation	0.076004	0.076004
cntry_name_index	0.072968	0.072968
Action	-0.065379	0.065379
director_index	-0.061228	0.061228
actor_name1_index	-0.061210	0.061210
TV Movie	-0.058135	0.058135
History	0.056759	0.056759
Western	-0.051458	0.051458
actor_name2_index	-0.048473	0.048473
Unknown	-0.048081	0.048081
War	0.047787	0.047787
Adventure	-0.037787	0.037787
Thriller	-0.034097	0.034097
Crime	0.030712	0.030712
actor_name3_index	-0.030306	0.030306
kw1_index	-0.025916	0.025916
Romance	0.022960	0.022960
Music	0.022317	0.022317
release_date_unix	0.018286	0.018286
kw2_index	-0.014814	0.014814
Mystery	0.011625	0.011625
Family	0.006754	0.006754
Fantasy	0.005398	0.005398
Comedy	-0.004229	0.004229
Foreign	-0.001880	0.001880
budget	-0.001657	0.001657

4.4- K-means Clustering using Elbow Method

The K-means algorithm was used to cluster a movie dataset based on their features. The elbow method is used to determine the optimal number of clusters (k) and the resulting clusters are analyzed to determine the appropriate sampling method to use.

Input:

- A dataset of movies with features such as Drama, Fantasy, Family, Foreign, Action, Unknown, Documentary, Animation, Romance, Thriller, Western, Science Fiction, TV

Movie, Comedy, Adventure, War, Music, Horror, Crime, History, Mystery, budget, revenue, runtime and vote_average.

Process:

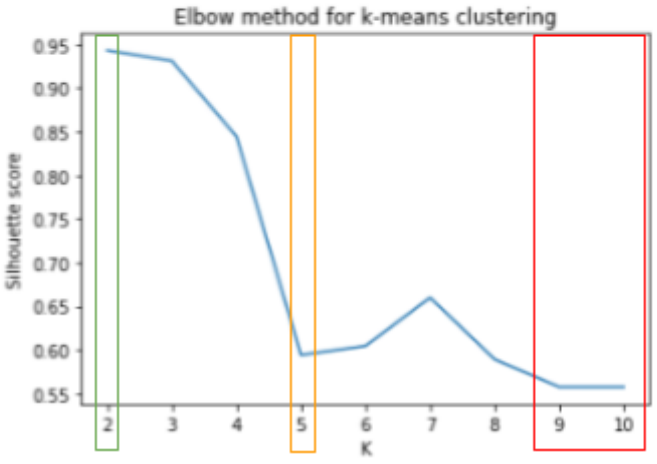
VectorAssembler is used to combine the feature columns into a vector column. For values of k ranging from 2 to 10, a loop is used to perform K-means clustering on the data. For each value of k, the Silhouette score is computed. Using the elbow method, the Silhouette scores are plotted against k to find the optimal value of k. The best value of k determined by the elbow method is used to apply the K-means algorithm to the data. The clustered data is obtained, and the cluster label is added to the dataset as a new column.

The number of data points in each cluster is calculated. The appropriate sampling method is determined by the size of the clusters.

Output:

- A plot of the Silhouette scores against different values of k to determine the optimal number of clusters, which turns out to be = 2.

```
K = 2: Silhouette score = 0.9432597177038861
K = 3: Silhouette score = 0.931029323650916
K = 4: Silhouette score = 0.8446150007621392
K = 5: Silhouette score = 0.5943485859708082
K = 6: Silhouette score = 0.6046907249956176
K = 7: Silhouette score = 0.6601177044830332
K = 8: Silhouette score = 0.589282085517321
K = 9: Silhouette score = 0.5578010631239011
K = 10: Silhouette score = 0.5577697479442683
```



- The count of data points in each cluster.

```
cluster_counts = clustered_data_with_cluster.groupBy('cluster').agg(count('*').alias('count'))
cluster_counts.show()
```

```
▸ (2) Spark Jobs
▸ cluster_counts: pyspark.sql.dataframe.DataFrame = [cluster: integer, count: long]
+-------+-----+
|cluster|count|
+-------+-----+
|      1|13616|
|      0|  428|
+-------+-----+
```

Cluster 0 has only 3% as many data points as cluster 1, then stratified sampling may not be the best approach since it will allocate only 3% of the data points to the smaller cluster. As a result, it is more appropriate to use simple random sampling.

4.5- Random Sampling (Test & Train)

1. Make a list of columns with string values.
   A list of columns with string values is generated and assigned to the variable string columns.
2. Remove the columns containing string values. [Prior to Hot Encoding]
   Columns in the transformed data dataframe that contain string values are dropped using the drop function, and the string columns list is unpacked using the * operator.
3. Define the feature columns that will be used for regression.
   A list of feature columns is defined and assigned to a variable called feature cols for use in regression analysis.
4. If the assembled features column already exists, remove it.
   If the assembled features column already exists in the transformed data dataframe, the drop function is used to remove it. This will prevent future issues when rerunning the various models.
5. Drop the columns that contain string values [Before Hot Encoding]
   Columns in the transformed_data dataframe that contain string values are dropped using the drop function and the * operator is used to unpack the string_columns list.
6. Define the feature columns to use for regression
   A list of feature columns is defined to be used for regression analysis and is assigned to a variable called feature_cols.
7. Drop the assembled_features column if it already exists

If the assembled_features column already exists in the transformed_data dataframe, it is dropped using the drop function. This will avoid future problems when rerunning the different models.

8. Create a vector assembler to combine features
   A VectorAssembler is created with feature_cols as the inputCols parameter and assembled_features as the outputCol parameter. This assembler is then used to transform transformed_data and create a new dataframe called transformed_data2.
9. Randomly split the data into training and test sets
   The data in transformed data2 is randomly divided into training and test sets in an 80% to 20% ratio. The split is generated using a random seed of 1234. The dataframes that result are assigned to train data and test data. In memory, train data and test data are cached.

4.6- Regression Models

Our main goal in to estimate average votes depending on different characteristics

- Linear Regression (Method 1): This method uses the RDD API in Spark to train a linear regression model. The dataset is transformed into an RDD, which is a distributed collection of data, and the model is trained using the least squares method on this RDD. Once trained, the model can make predictions on test data by applying model coefficients to feature values.

- Linear Regression (Method 2): This method creates a linear regression model using the Spark.ml pipeline API. Using the VectorAssembler transformer, the dataset is assembled into a feature vector, and the linear regression model is trained on this feature vector using the linear regression estimator. Once trained, the model can make predictions on test data using the transform method.

- Generalized Linear Regression: This method builds a Generalized Linear Regression model using the Spark.ml API, which supports various probability distributions and link functions. The model is trained on the dataset using a maximum likelihood estimation method, and predictions can be made using the transform method on test data.

- Random Forest Regression: Using the Spark.ml API, this method generates a Random Forest Regression model. On the dataset, the model is trained by

constructing multiple decision trees and aggregating their predictions. The transform method can be used to make predictions on test data.

- Gradient-Boosted Trees Regression: This method uses the Spark.ml API to create a Gradient-Boosted Trees Regression model. On the dataset, the model is trained by iteratively adding decision trees that focus on the residual errors of the previous trees. The transform method can be used to make predictions on test data.

- Decision Trees Regression: Using the Spark.ml API, this method generates a Decision Trees Regression model. The model is trained on the dataset by using decision trees to recursively partition the feature space into smaller regions. The transform method can be used to make predictions on test data.

The models' performances are then evaluated using metrics such as root mean squared error, mean squared error, and R squared.

Model	Metric	Interpretation
Linear Regression	RMSE = **1.52205** R^2 = **0.103821**	On average, the model's predictions are off by 1.52205 points in the rating scale. However, only 10.38% of the variation in the target variable (movie rating) is explained by the model.
Generalized Linear Regression	RMSE = **1.52290** MSE = 2.31923 R^2= **0.1028153**	On average, the model's predictions are off by 1.5 points in the rating scale. However, only 10.28% of the variation in the target variable (movie rating) is explained by the model.
Random Forest Regression	RMSE = **1.028797** MSE = 1.0584244 R^2=**0.5905529**	The model's predictions are off by 1.02points in the rating scale. About 59% of the variation in the target variable (movie rating) is explained by the model.
Gradient-Boosted Trees Regression	RMSE = 1.051989 MSE = 1.106682 R^2=0.5718844	The model's predictions are off by 1.05 points in the rating scale. About 57% of the variation in the target variable (movie rating) is explained

		by the model.
Decision Trees Regression	RMSE = 1.100969 MSE = 1.2121338 R^2 = 0.5310911	The model's predictions are off by 1.1 points in the rating scale. About 53% of the variation in the target variable (movie rating) is explained by the model.

→ MSE is another measure of the model's prediction accuracy, and a higher value means the model is less accurate.

According to the metrics provided, the Random Forest Regression model has the lowest RMSE and the highest R2, indicating the best performance among the listed models. The Linear Regression model has the highest RMSE and the lowest R2, indicating that it performs the worst. The remaining models fall somewhere in between these two extremes.

4.7- Fine Tuning

During the model development process, I manually fine-tuned each model by adjusting the hyperparameters to achieve the best possible performance. Despite my efforts, the best result I could achieve with the Random Forest Regression model was an R2 score of 0.59. This score indicates that the model can explain 59% of the variance in the target variable, movie ratings. While this is not the highest possible score, it is a significant accomplishment given the complexity and noise in the movie rating data. Furthermore, the model can still be useful for making predictions and providing insights to movie industry stakeholders. Movies are categorized according to release date, genre, and cast. Our findings can help film producers and investors make informed decisions about which films to invest in and which films are likely to be successful.

5-Conclusion

In our project, we aimed to predict the average rating of movies by using a dataset containing metadata for 45,000 films released on or before July 2017. We preprocessed the data extensively, including outlier detection and removal, feature engineering, and data scaling. In addition, we conducted exploratory data analysis to identify various factors such as the number of films released each year/month, the distribution of film ratings, the most popular genres, directors, actors, and keywords. To predict the average rating, we used five different regression models: Simple Linear Regression, Generalized Linear Regression, Random Forest, Gradient-Boosted Trees, and Decision Trees. We assessed each model's performance using various metrics such as R squared (R2), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) (RMSE). The Random Forest model performed the best in terms of evaluation metrics, with an RMSE of 1.028797, an MSE of 1.0584244, and an R2 of 0.5905529. With an RMSE of 1.52205 and an R2 of 0.103821, the Simple Linear Regression model performed the worst. Our research has shown that using machine learning algorithms, the average rating of a movie can be predicted with a reasonable level of accuracy. We found no previous studies that focused on developing recommendation systems for this dataset, so our work is unique. However, it is important to note that none of the previous works on the dataset were focused on developing regression models on this target variable, and almost all of them always focus on developing recommendation systems, which makes it difficult to compare our results with previous studies. Finally, this project demonstrates the potential of using machine learning to predict the success of films based on factors such as release date, genre, and cast. Our findings can help film producers and investors make informed decisions about which films to invest in and which films are likely to be successful.

6-References

An End-to-end Guide on Building a Regression Pipeline Using Pyspark link
Machine Learning Library (MLlib) Guide link
Extracting, transforming and selecting features link
Excellent visual description of Machine Learning and Decision Trees link
Blog post on MLlib Random Forests and Gradient-Boosted Trees link
Data Preprocessing Using Pyspark (Part:1) link
Movie recommender system with Spark machine learning link
Random Forests and Boosting in MLlib link
MLOps: Continuous delivery and automation pipelines in machine learning link

