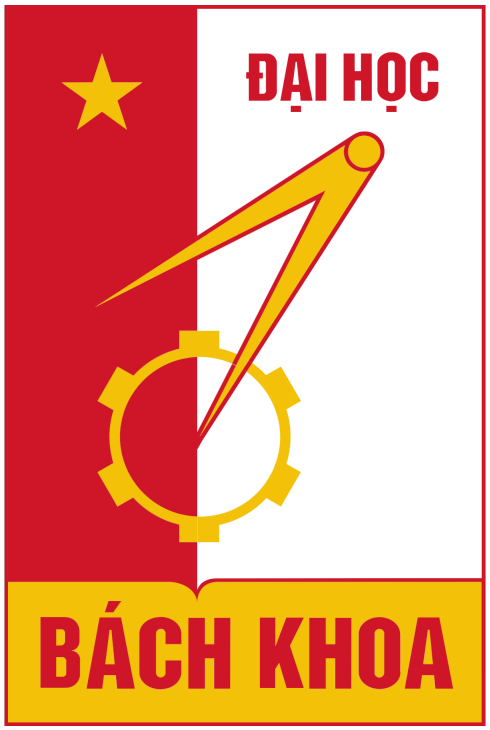
Movie Recommender

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY



Group members:

Nguyen Trong Phuong Bach 20225473

Vu Minh Hieu 20225494

Bui Nguyen Khai 20225501

Tran Ngoc Quang 20225523

Nguyen Viet Tien 20225533

Guided by:

Associate Professor Than Quang Khoat

Course:

IT3190E - 2023.2

Abstract

Recommender systems play a crucial role in various applications by suggesting relevant items to users. This report explores regression-based recommendation systems using two main techniques: collaborative filtering and content-based filtering. The report details the data analysis process, including feature selection using techniques like Singular Value Decomposition (SVD). We then delve into various regression algorithms suitable for recommender systems, including Matrix Factorization, Stacked Generalization, Support Vector Regression (SVR), Random Forest, and LightGBM (Gradient Boosted Decision Trees). The implementation section outlines how these techniques are applied for content-based and collaborative filtering approaches. We explore model tuning strategies for different algorithms and finally analyze the results obtained from the implemented recommendation system.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

In the ever-evolving domain of recommender systems, accurately predicting user preferences and delivering personalized recommendations have become critical components for enhancing user satisfaction and engagement. This report explores regression-based movie prediction methods, leveraging both collaborative filtering and content-based filtering techniques. The MovieLens dataset, comprising extensive user ratings and movie metadata, serves as an excellent benchmark for developing and evaluating such systems.

1.2 Objectives

The primary objective of this report is to compare and analyze the effectiveness of collaborative filtering and content-based filtering in predicting user ratings for movies. By employing regression techniques, we aim to develop predictive models that can accurately estimate user ratings, thus enhancing the recommendation quality. Additionally, we seek to understand the strengths and limitations of each approach and identify scenarios where one might outperform the other.

1.3 Collaborative Filtering

Collaborative filtering models leverage the collective ratings provided by numerous users to generate recommendations. The core concept of collaborative filtering is that missing ratings can be estimated because the ratings given by different users and items tend to be highly correlated. For instance, if two users, Bach and Tien, have similar tastes and their provided ratings are very similar, the algorithm can identify this similarity. Consequently, it is likely that ratings specified by only one of them will also be similar. This identified similarity can then be used to infer missing ratings. [1]

1.4 Content-based

In content-based recommender systems, recommendations are made using the descriptive attributes of items, referred to as "content." These methods combine user ratings and purchasing behavior with the item's descriptive information. For instance, if Quang has

rated the movie Terminator highly but we lack ratings from other users, collaborative filtering is not an option. However, because Terminator shares genre keywords with other science fiction movies like Alien and Predator, these can be recommended to Quang. Content-based methods use item descriptions labeled with ratings as training data to create a user-specific classification or regression model. For each user, the training data consists of descriptions of items they have bought or rated, with the ratings serving as the dependent variable. This user-specific model predicts whether the user will like an item for which their rating is unknown. One key advantage of content-based methods is their effectiveness in recommending new items that lack sufficient rating data. Since the active user may have rated other items with similar attributes, the model can use these ratings along with the item attributes to make informed recommendations even in the absence of prior ratings for that specific item. [1]

Chapter 2

Data analyzing and Feature selection

2.1 Dataset

2.1.1 MoviesLens

The MovieLens latest dataset describes 5-star rating and from MovieLens, a movie recommendation service. It contains 33,832,162 ratings and 2328315 tag applications across 86,537 movies. These data were created by 330975 users between January 09, 1995 and July 20, 2023. This dataset was generated on July 20, 2023. Users were selected at random for inclusion. All selected users had rated at least 1 movies. No demographic information is included. Each user is represented by an id, and no other information is provided. All ratings are contained in the file *ratings.csv*. The content of the file includes: userId, movieId, rating, timestamp. Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars). Movie information is contained in the file *movies.csv*. The content of the file includes: movieId, title, genres. Genres are a pipe-separated list. Identifiers that can be used to link to other sources of movie data are contained in the file *links.csv*. The content of the file includes: movieId,imdbId,tmdbId. are identifiers for movies used by movielens.com, imdb.com, themoviedb.com. [2]

2.1.2 IMDb

Since MovieLens dataset is lacking genres and years information, IMDb datasets are used to fill in missing data, and get important movies' features such as: release years, directors, top actors, IMDb ratings. The dataset used are: *title.basics.tsv.gz* (genres, years), *title.ratings.tsv.gz* (IMDb ratings), *title.crew.tsv.gz* (directors' ID), *title.principals.tsv.gz* (actors' ID).

2.2 Data analysis

2.2.1 User ratings

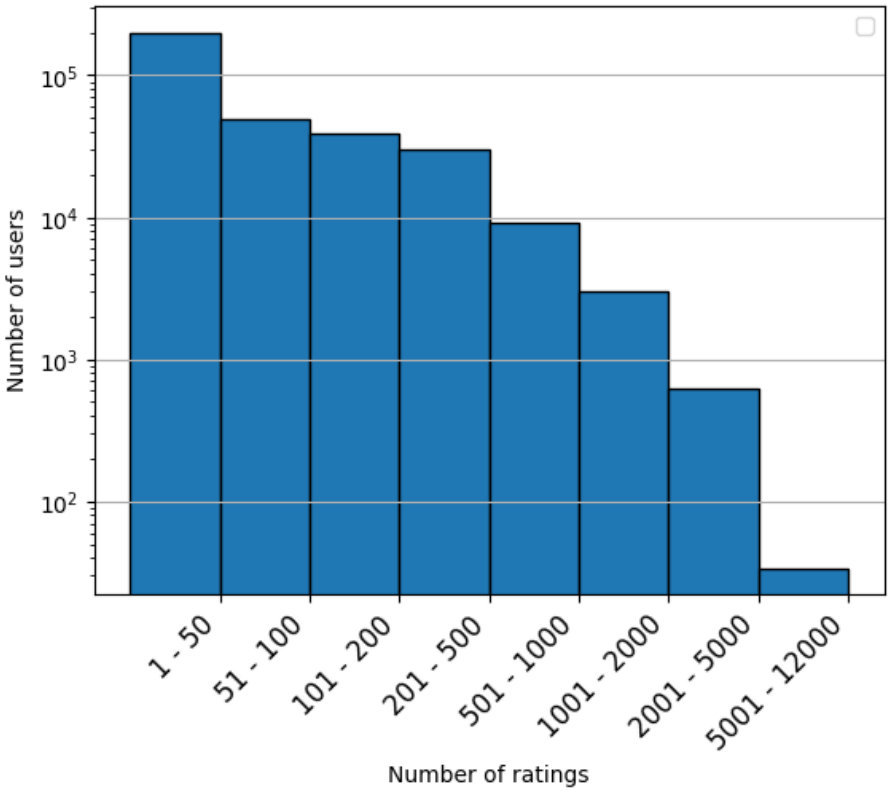


Fig 2.1. Number of users per rating groups

The figure is a histogram showing the distribution of rating counts. The x-axis represents the number of ratings, and the y-axis represents the number of users who gave that many ratings. The most common number of ratings is between 0 and 50. There are progressively fewer users as the number of ratings increases.

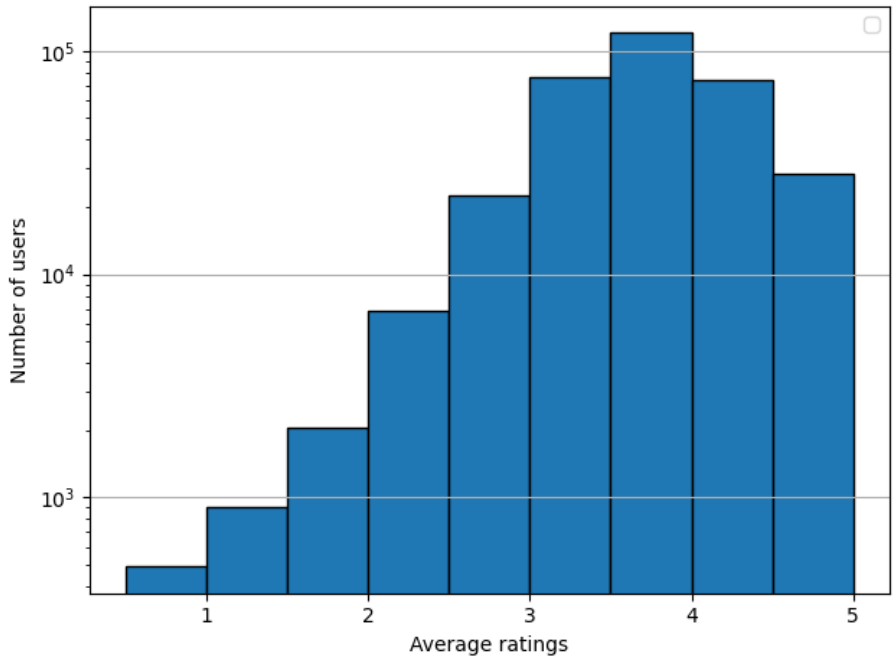


Fig 2.2. Average ratings per users

Overall, the histogram suggests that there are more users who tend to rate movies with an average rating between 3.5 and 4, compared to the number of users who rate movies with very high or very low average ratings.

2.2.2 Movies

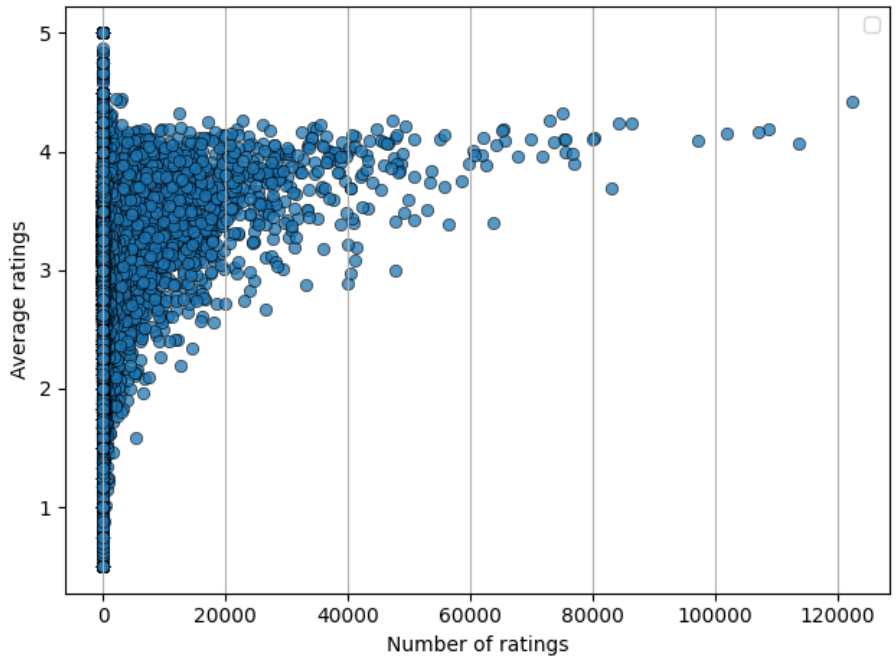


Fig 2.3. Number of ratings to average ratings

The figure appears to be right-skewed, meaning there are more users who tend to give movies higher ratings than lower ratings. The most popular movies are usually highly-rated.

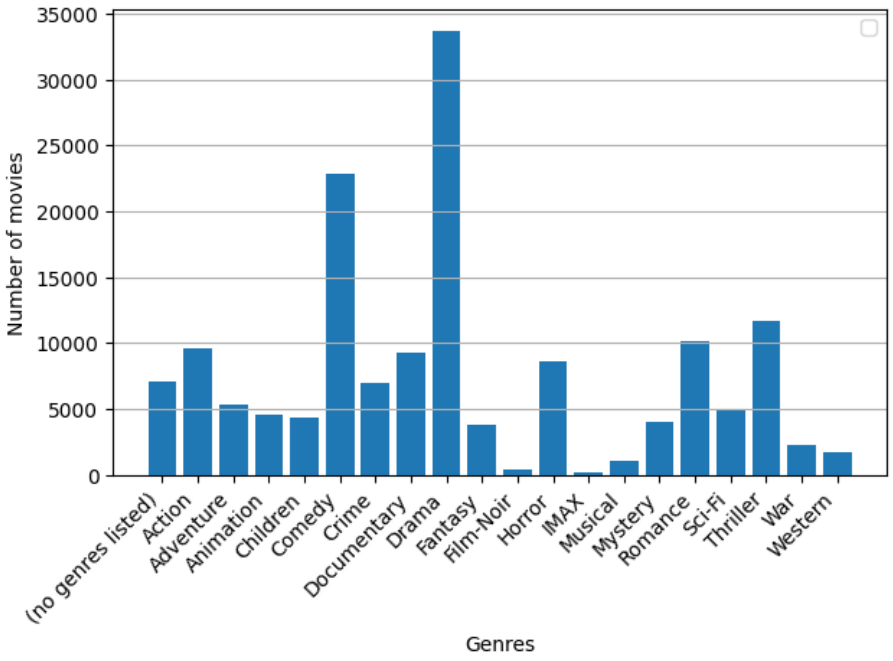


Fig 2.4. Movies' genres count

The figure shows that Drama and Comedy are clear frontrunners, with Drama leading the pack with almost 35000 movies. Meanwhile several genres have a relatively small number of movies, including Film-Noir, Musical and Western. However, there are still over 5000 movies that has no genres listed on the dataset.

2.3 Feature selection

2.3.1 Singular value decomposition

The sparse user-item matrix is very large and grows exponentially with every user and movie added, while only contain little information since any single user is likely to have rated only a small percentage of possible items. Fitting this matrix directly to machine learning models would result in inefficient and inaccuracy. Singular value decomposition helps combat this problem and aids the data processing for many algorithm.

The rating data collected can be represent by a $2D$ matrix where each column represents an user, each row represents a movie and the entries of the matrix is the rating of an user to the corresponding movie. The goal is to predict the rating of an user to a movie that they haven't seen yet, in other word, the value off an empty entries in the matrix.

Consider an user item matrix $A_{m\times n}$, it is always possible to decompose this matrix in to three matrix: $A = U\sigma V$, where U in an $m \times m$ matrix, σ is a $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal $\{\sigma_1, \sigma_2, ...\sigma_n\}$ such that $\sigma_1 \geqslant \sigma_2 \geqslant ... \geqslant \sigma_n$ and V is an $n \times n$ matrix.

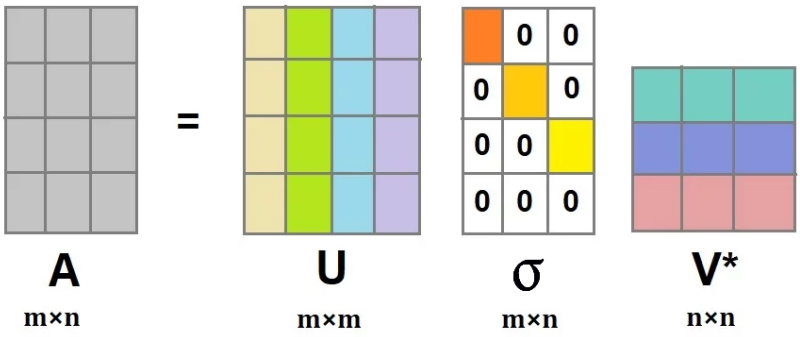


Fig 2.5. Singular value decomposition

The rating of user u for movie i is recorded in the $a_u i$ entry of A can be calculated by the equation: $A_u i = u_i^T \sigma v_i^T$ where U_u is the u^{th} column of U and v_i is the i^{th} row of V. The values of U_u can be represent as the interest of user u to that corresponding latent factor, and the values of v_i measure the extent to which the item possesses those factors.

2.3.2 Truncated SVD

Since σ is a rectangle diagonal matrix with decreasing entries, we can truncated the small values of σ from the matrix, thus only keeping the high r energy value that approximate the original A matrix, as well as removing the corresponding $(m-r)$ columns of U and $(n-r)$ rows of V. We call this three new matrix $\tilde{U}$, $\tilde{V}$ and $\tilde{\sigma}$, its product is the best r-rank approximation to the original matrix A measured by the Frobenius norm:

$$\tilde{A} = \tilde{U}\tilde{\sigma}\tilde{V} = arg \min_{\text{rank}(\tilde{A}) \leqslant r} \left\| A - \tilde{A} \right\|_{\mathbf{F}}$$

This allows for reducing the overall size of the dataset, while maintain the required accuracy.

2.3.3 Content-based

Users who have rated fewer than 50 movies and movies that have been rated by fewer than 50 users have been removed before the training process for several reason: Sparse data can lead to biased recommendations because there may not be enough information to accurately evaluate these users or movies. It also can negatively impact the performance of machine learning models by introducing noise or overfitting.

By focusing on users and movies with a sufficient number of ratings, the system can better identify patterns and similarities among items or users, leading to more reliable recommendations.

Genres is one of the main attributes for content-based filtering. Knowing a user's preferred genres allows the system to suggest movies that align with their tastes. For instance, if a user enjoys science fiction, the system can recommend similar sci-fi films. Furthermore, users can discover movies within their favorite genres that they may not

have come across otherwise. This is particularly useful in exploring lesser-known films within a genre.

Movies often reflect the cultural, social, and technological contexts of the times in which they were made. By categorizing movies by decade, the system can better match users with movies that have the aesthetic, themes, and production styles they are likely to appreciate based on their preferences.

Mean average ratings for directors and top 3 actors can serve as indicators of historical success and quality. If a director or actor consistently receives high ratings, it's likely they have a track record of creating or participating in well-received films, which can be a strong predictor of a movie's potential appeal. Actors and directors with high average ratings often have a strong fan base. By incorporating their ratings into the recommendation system, you can tap into the star power that attracts audiences, making recommendations more aligned with user preferences.

2.3.4 Collaborative filtering

Collaborative Filtering recommends items based on similarity measures between users and items. The basic assumption behind the algorithm is that users with similar interests have common preferences. For example if user B likes movie X; Z and user A likes movie X; Y; Z. Then it is likely that user B will also be interested in the movie Y.

New user that we haven't collected any data on the platform will either be asked to rate some movies they have watch, or recommended high average rated movie.

Chapter 3

Algorithm

3.1 K-nearest neighbours

The idea of K-NN (in this case regression) is that we calculate the average target values of its k- nearest neighbors. For predictions, the algorithm computes the distance/similarity between the target point and all the other points in the dataset, usually by Euclidean distance. Then, k nearest points are selected and their averages are taken into consideration for the prediction of the average of the target value, which is given by the following formula:

$$y = \frac{1}{|NB_k(\mathbf{x})|} \sum_{i \in \mathcal{NB}_k(\mathbf{x})} y_i$$

where y is the predicted average value of the target, y_i is the average of neighbor i, $NB_k(x)$ is the set of all the nearest neighbors we choose(indicating that its cardinality is k) [3]

3.2 Ridge

Ridge is essentially an improvement of regular linear regression, where the use of ordinary least squares only focuses on optimization for the training set and can lead to overfitting. This model aims to find and minimize the following function:

$$f^* = \arg\min_{f \in H} RSS(f) + \lambda \|\mathbf{w}\|_2^2$$

Where H is the space we are exploring to find the best approximate function(In this case the linear space), RSS is the residual sum of squares with respect to a certain function f in H,λ is the penalty coefficient $(>= 0)$ and $||\mathbf{w}||^2$ is the L2 norm of vector w. [3]

3.3 Matrix Factorization

3.3.1 Matrix Factorization Model

A Matrix Factorization Model [4] factor user-item matrix generated from the data into two matrix: the user-factor P and thee item-factor matrix Q such that the estimated rating of user u for movie i, denoted by $\hat{r}_{ui}$ is equal to $p_u^T \ q_i$.

$$\hat{r}_{ui} = p_u^T \ q_i \tag{1}$$

To learn the factor vectors (p_u and q_i), the system minimizes the regularized squared error on the set of known ratings:

$$\min \sum_{(u,i)\in \mathbf{K}} (r_{ui} - q_i^T p_u)^2 \; + \; \lambda(|| \; q_i \; ||^2 \; + \; || \; p_u \; ||^2) \tag{2}$$

where K is the set of the (u, i) pairs for which r_{ui} is known (the training set). The model simultaneously learn the function and prevent over fitting using the learning parameter λ.

3.3.2 Learning Algorithm

To minimize the above minimization problem, the algorithm use stochastic gradient descent optimization method. For each given training case, the system predicts r_{ui} and computes the associated prediction error

$$e_{ui} = r_{ui} - q_i^T p_u$$

Then it modifies the parameters by a magnitude proportional to γ in the opposite direction of the gradient, yielding:

$$p_u \leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u)$$
$$q_i \leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i)$$

3.3.3 Adding biases

The majority of data observed contains variation that is due to the effect of *bias*. Two types of bias considered in this algorithm is the item bias -the bias result in fluctuation in the rating of a particular movie and the user bias - the bias result in higher or lower mean rating of an user compare to the global mean rating.

We model the bias of a rating as follow:

$$r_{ui} = \mu + b_i + b_u + q_i^T p_u \tag{3}$$

Where μ is the global average rating of all user b_i is the item bias, b_u is the user bias. This is illustrated in the following example:
Gordon is a critical user and the algorithm predicts the rating Gordon would give to the movie “Iron man 3”. The average rating is 2,4 , “Iron man 3” is from a famous studio so people tend to rate this movie 0.5 higher than average. However, Gordon usually rate movies 0,2 lower than the average user, so the rating of Gordon for this movie would be: $2,4 + 0,5 - 0,2 + q_i^T p_u$ Substituting (2) into (3), the system learns by minimizing the squared error function:

$$\min_{q*,p*,b*} \sum_{(u,i)\in K} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2 \right)$$

3.4 Stacked Generalization (Stacking)

Stacked Generalization for regression is an ensemble learning technique to combine multiple regression models via a meta-regressor. The individual regression models are trained based on the complete training set; then, the meta-regressor is fitted based on the outputs (meta-features) of the individual regression models in the ensemble. The idea is based on different types of models, which can perform well when dealing with some part of the problem, but not the whole space of the problem.

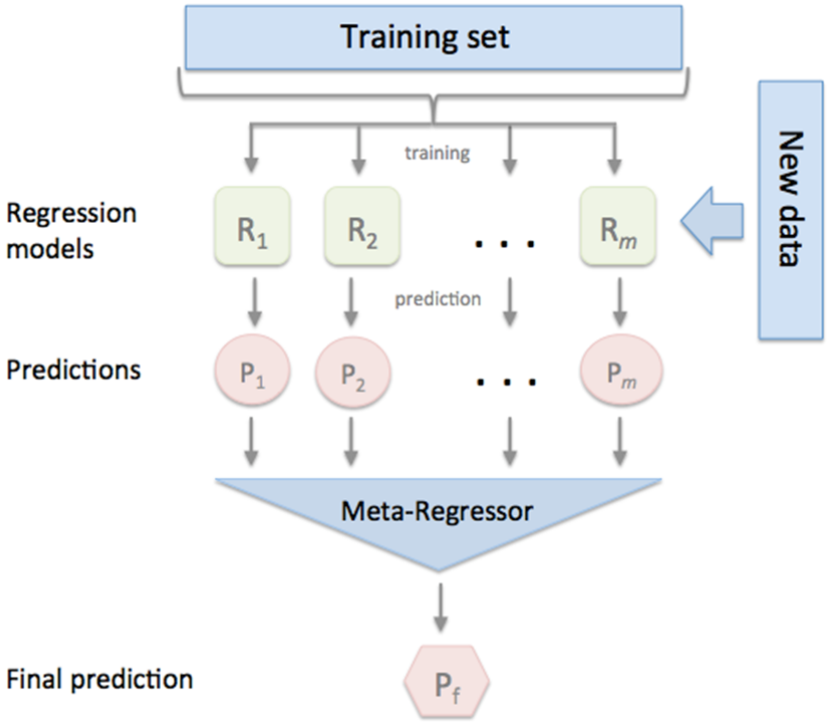


Fig 3.1 How Stacked Generalization works.

First, the data is split into k-folds, just like in k-fold cross-validation, and a test set. Then, several base models are fitted on the k-1 part of a training dataset, and predictions are made for k^{th} part. These base models can be different algorithms, such as decision trees, neural networks, or support vector machines. The process is done on every part of the training data. Every base model is then fitted on the whole train data set to calculate its performance on the test set.

After that, overall predictions from the train set are used as features for training the meta-model, which is used to make the final prediction on the test set. The meta-regressor is often simple, providing a smooth interpretation of the predictions made by the base models. As such, linear models are often used as the meta-model, such as linear regression, ridge regression or LASSO.

3.5 Support Vector Regression (SVR)

Support Vector Regression (SVR) is an extension of the popular Support Vector Machine (SVM) algorithm for regression tasks. It seeks to find a hyperplane that best fits the

data points in a continuous space. This is achieved by mapping the input variables to a high-dimensional feature space and finding the hyperplane that maximizes the margin (distance) between the hyperplane and the closest data points, while also minimizing the prediction error. The goal here is to minimize empirical error term R_{emp} by Vapnik's ϵ-insensitivity loss function

$$R^{\epsilon}_{emp}(\mathbf{W}, b) = \frac{1}{l}\sum_{i=1}^{l} |y_i - \mathbf{w}^T\mathbf{x}_i - b|$$

and $||\mathbf{w}||^2 =< \mathbf{w}, \mathbf{w} >$, simultaneously. Thus, we construct a linear regression hyperplane $f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x} + b$ by minimizing

$$R = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l} |y_i - f(\mathbf{x}_i, \mathbf{w}|_{\epsilon}$$

which can be done with the help of optimization techniques such as Lagrange multipliers, Karush–Kuhn–Tucker theorem ...

For non-linear regression, the idea is to transform the input into another space, which often has higher dimensions, so that the projection of data is linearly separable, and linear SVR can be used in the new space. In this case, nonlinear regression function is created by using the weighting vector $\mathbf{v}_0$ and the kernel (Grammian) matrix G:

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{G}\mathbf{v}_0 + b$$

[5]

3.6 Random Forest

3.6.1 Decision Tree

A decision tree is a tree-like structure where each node represents an attribute to test for a sample, each branch from a node corresponds to a possible value of the attribute associated with that node, and each leaf node represents a class or a final prediction. The decision tree's goal is to create a model to predict the value of the target variable by learning simple IF-THEN decision rules derived from the data attributes.

A learned decision tree predicts a sample by traversing the tree from the root node to a leaf node, where the class label or value associated with that leaf node is used for the prediction. Thus, a tree represents some classification or regression function. In the case of our problem, we use decision trees where the target variable takes on continuous values (usually real numbers) which are called regression trees. [6]

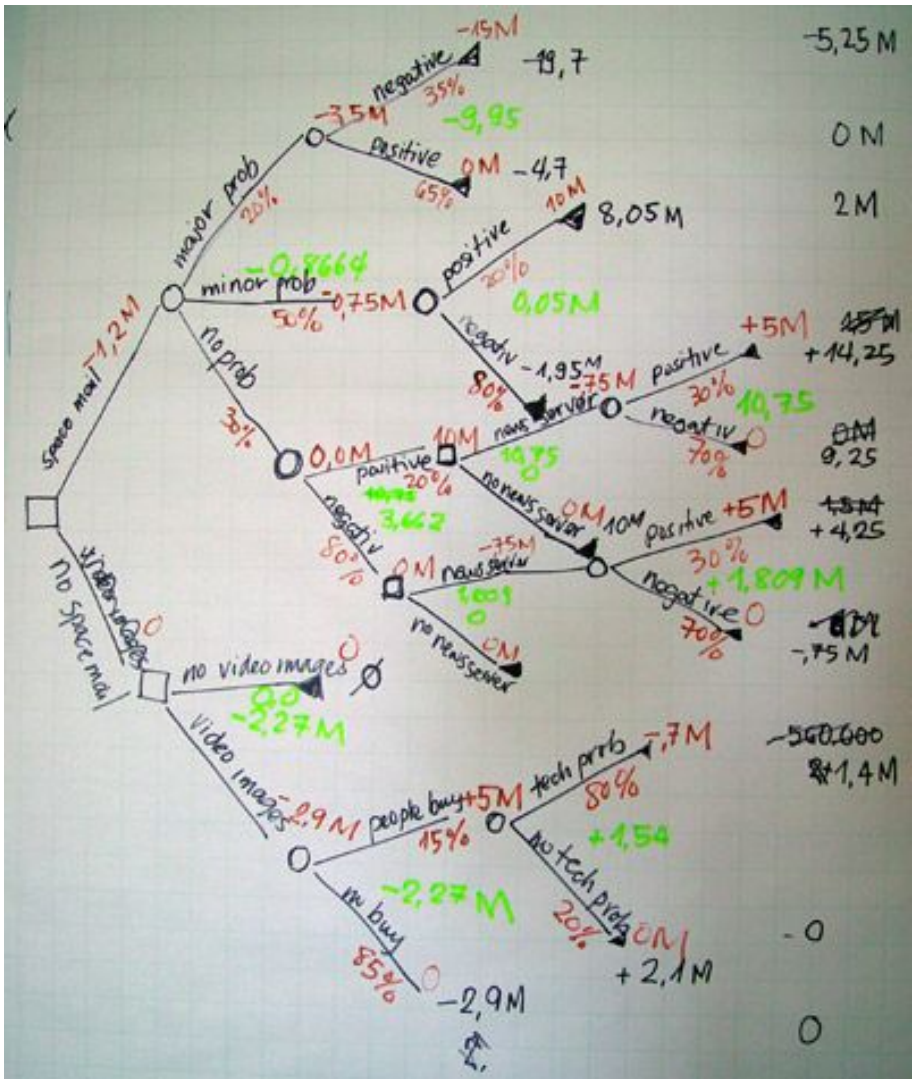


Fig 3.1 A manually drawn decision tree diagram [7]

3.6.2 Random Forest

When constructing a decision tree, if the depth is left unrestricted, the tree will classify all the data in the training set correctly, which can lead to overfitting. The Random Forest algorithm consists of multiple decision trees, and each decision tree incorporates random elements:

- Randomly sampling data to construct the decision tree.
- Randomly selecting features to place at a node.

Since each decision tree in the Random Forest algorithm does not use all the training data, nor does it use all the features of the data to construct the tree, each tree may not perform well individually. As a result, each decision tree model is not overfitted but may be underfitted, or in other words, the model has high bias. However, the final result of the Random Forest algorithm is an ensemble of many decision trees, so the information from each tree complements the others, leading to a model with low bias and low variance, or in other words, a model with good predictive performance.

Random Forest Algorithm

Assume the training dataset D consists of n data samples, and each sample has d attributes. We learn K decision trees as follows:

1 Create K trees, with each tree generated as follows:

   a Build a subset D_i by randomly sampling (with replacement) from D.

   b Train the i^{th} tree from D_i.

   c At each node during the tree construction process:

      i Randomly select a subset of attributes.

      ii Split the tree based on this subset of attributes.

   d This tree will be grown to its maximum size without pruning.

2 Each subsequent prediction is obtained by averaging the predictions from all the trees.

To construct a subset D_i from D, we randomly sample n data points from the training dataset using Bootstrapping. This means that once a data sample is selected, it is not removed from the original dataset but remains available for further sampling until n samples have been selected. The Random Forest algorithm consists of multiple decision trees, each built using the Decision Tree algorithm on different subsets of data and different subsets of attributes. The final prediction of the Random Forest algorithm is then aggregated from the predictions of all the constructed decision trees.

3.7 LightGBM

3.7.1 Gradient Boosted Decision Trees

Gradient Boosting is an ensemble learning technique that builds a model in a stage-wise fashion by sequentially adding predictors to an ensemble, each one correcting its predecessor. The core idea is to combine the strengths of multiple weak learners (typically Decision Trees) to create a strong overall model. How the algorithm works:

(1) Start with an initial model (Decision Tree) $F_0(x)$.

(2) Calculate the residuals, which are the negative gradient of the loss function with respect to the model's predictions. These residuals represent the errors the model needs to correct in the next iteration.

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

Where r_{im} is the residual for the i^{th} instance at the m^{th} iteration. y_i is the actual target value, and $f_{m-1}(x_i)$ is the prediction from the model at the previous iteration.

(3) For $m = 1$ to M (number of trees): Fit a new tree $h_m(x)$ to the residuals from the previous step.

(4) Add the new tree to the model. The contribution of this new tree is scaled by a shrinkage v, which is a hyperparameter that controls how fast the strong model is learning,

$$F_m(x) = F_{m-1}(x) + vh_m(x)$$

where $h_m(x)$ is the prediction of the new tree and $Fm_(x)$ is the updated model.

(5) Repeat steps 2 to 4 for a predefined number of iterations or until the model's performance stops improving.

Chapter 4

Implementation

4.1 Content-based

For content-based, after processing the information of the movies, the attributes used for the model are: IMDb ratings, directors' mean ratings, top 3 actors' mean ratings, binary-encoded decades, binary-encoded genres. Normalization of these attributes ensures that all features contribute equally to the similarity measures and model training. This prevents features with larger ranges from dominating the results and allows the model to treat all attributes with equal importance, leading to more balanced recommendations. To handle sparsity, **TruncatedSVD** by scikit-learn will be used to transform the sparse matrix into a dense representation, where latent factors capture the underlying patterns in the data, thus improving the ability to make accurate predictions. The input to the regression model are features vector of movies that the users have rated. The output will be prediction ratings of all movies in the dataset by using regression models such as: k-Nearest Neighbors, Ridge Regression, Random Forest, LightGBM (Gradient Boosted Decision Trees)

4.2 Collaborative filtering

For collaborative filtering, the rating data collected is projected into a 2D user-item matrix, with users as column index and movies as row index. This matrix is the input to the models namely: Matrix Factorization, k-Nearest Neighbors, Support Vector Regression, Stacked Generalization.

Chapter 5

Model Tuning

5.1 Model tuning by groups

Grouping users by their number of ratings can help improve the tuning and accuracy of recommendation systems. Users with different numbers of ratings may show different behaviors and preferences. Grouping them allows for more tailored model tuning, such as adjusting hyperparameters that best fit the data density and variability within each group. Moreover, users with a small number of ratings may introduce more noise, as their preferences are less clearly defined. Grouping and handling them separately can prevent this noise from affecting the overall model.

5.2 k-Nearest Neighbors

- weights: Weights determine how the influence of neighbors is factored into the prediction. There are two common weighting strategies:
  - Uniform Weights: All neighbors are treated equally.
  - Distance Weights: Closer neighbors have a higher influence on the prediction than distant ones.
- n_neighbors (k): Critical for balancing bias and variance, capturing local data patterns, and determining computational complexity.

5.3 Ridge

- solver: The solver is the algorithm used to optimize the cost function in Ridge regression. Different solvers can affect the efficiency and accuracy of the model.
  - 'auto': Automatically selects the solver based on the data.
  - 'svd': Uses Singular Value Decomposition, suitable for small datasets.
  - 'cholesky': Uses the Cholesky decomposition, faster for well-conditioned and smaller datasets.
  - 'lsqr': Uses the least-squares method, efficient for large datasets.
  - 'sparse_cg': Uses the conjugate gradient method, efficient for large and sparse datasets.

  - 'sag': Uses Stochastic Average Gradient descent, suitable for large datasets with many features.
  - 'saga': Similar to 'sag' but supports elastic-net regularization, effective for very large datasets.

- alpha: is the regularization parameter that controls the strength of the penalty applied to the coefficients. By adding a penalty to large coefficients, alpha reduces the model complexity, thus preventing overfitting. A higher alpha increases bias but reduces variance, while a lower alpha decreases bias but increases variance. Proper tuning of alpha helps in finding the right balance.

5.4 Random Forests

n_estimators: specifies the number of individual decision trees to be included in the ensemble. This parameter plays a crucial role in the performance and behavior of the Random Forest.

5.5 LightGBM

- num_leaves: determines the maximum number of leaves in one tree in the LightGBM model. It is a key parameter that directly affects the complexity and capacity of the model. Higher num_leaves allows for more complex models that can capture intricate patterns in the data, while lower num_leaves results in simpler models that are less prone to overfitting but might not capture all the nuances of the data, potentially leading to underfitting.
- min_data_in_leaf: specifies the minimum number of data points that a leaf must have. It acts as a regularization parameter to prevent overfitting and control the granularity of the tree splits. Higher
  min_data_in_leaf ensures that each leaf has enough data points, reducing the likelihood of capturing noise in the data. This helps in creating a more generalized model that performs better on unseen data. Lower min_data_in_leaf allows leaves to have fewer data points, which can capture more detailed patterns but increases the risk of overfitting, especially if the data is noisy.

5.6 Stacked Generalization

- final_estimator: The meta-regressor is often simple, providing a smooth interpretation of the predictions made by the base models. As such, linear models are often used as the meta-model: linear regression, ridge regression or LASSO.

[3]

5.7 Matrix Factorization (SVD)

- n_factors: The number of factors in the latent vector. Higher value will retain more information and higher accuracy, at the expense of larger data size and more com-

putational power required.

- n_epochs: The number of iteration of the Stochastic Gradient procedure. This control the accuracy of the algorithm.
- lr_all: The learning rate of the algorithm.
- reg_all: The regularization term of the algorithm.

5.8 Support Vector Regression (SVR)

- kernel: Specifies the kernel type to be used in the algorithm:
  - 'linear': Linear kernel.
  - 'poly': Polynomial kernel.
  - 'rbf': Radial basis function (RBF) kernel.
  - 'sigmoid': Sigmoid kernel.
  - 'precomputed': Use a precomputed kernel matrix (if provided).
- C: Regularization parameter with L2 penalty. The strength of the regularization is inversely proportional to C, which plays a crucial role in determining the trade-off between training error and margin.

Chapter 6

Results

RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error) are used for evaluating regression models. Both metrics provide different insights into the model's performance, which helps in understanding and improving the prediction quality.

- RMSE: RMSE squares the prediction errors before averaging them. This means that larger errors will have significant impact on the RMSE value, making the metric sensitive to outliers. RMSE emphasizes the importance of accurate predictions for all instances, especially where large errors are unacceptable.
- MAE: MAE calculates the average absolute difference between predicted and actual values, making it straightforward and easy to interpret. The error is in the same unit as the predicted values, which helps in understanding the performance directly. Since MAE treats all errors equally, it provides a more balanced view of model performance without disproportionately penalizing larger errors. Unlike RMSE, MAE is less sensitive to outliers because it does not square the error terms. This can be beneficial when the dataset contains outliers or when the impact of large errors should not dominate the evaluation.

6.1 Content-Based

Model	Hyperparameters	Train MAE	Train RMSE	Test MAE	Test RMSE
Group: 50-100					
Ridge Regression	$\alpha = 90$	0.690	0.856	0.693	0.860
KNN	$n = 16$	0.699	0.870	0.705	0.875
Random Forest	max_depth=20	0.710	0.887	0.712	0.895
LGBM	lr=0.01, m_d_i_l=5	0.703	0.873	0.692	0.856
Group: 101-200					
Ridge Regression	$\alpha = 80$	0.664	0.836	0.667	0.842
KNN	$n = 22$	0.685	0.863	0.690	0.867
Random Forest	max_depth=20	0.682	0.865	0.707	0.884
LGBM	lr=0.01, m_d_i_l=5	0.680	0.849	0.680	0.855
Group: 201-500					
Ridge Regression	$\alpha = 38$	0.636	0.807	0.638	0.809
KNN	$n = 26$	0.672	0.851	0.672	0.851
Random Forest	max_depth=20	0.672	0.854	0.672	0.850
LGBM	lr=0.01, m_d_i_l=5	0.672	0.847	0.679	0.857
Group: 501-1000					
Ridge Regression	$\alpha = 19$	0.607	0.772	0.610	0.776
KNN	$n = 26$	0.657	0.831	0.660	0.835
Random Forest	max_depth=20	0.652	0.830	0.654	0.830
LGBM	lr=0.01, m_d_i_l=5	0.659	0.829	0.660	0.831
Group: 1001-12000					
Ridge Regression	$\alpha = 8$	0.586	0.744	0.585	0.743
KNN	$n = 30$	0.633	0.801	0.632	0.801
Random Forest	max_depth=10	0.623	0.792	0.619	0.789
LGBM	lr=0.01, m_d_i_l=5	0.6384	0.803	0.641	0.806

Table 6.1: Performance metrics for content-based models and their respective hyperparameters, grouped by ranges

6.2 Collaborative Filtering

n_factors	Avg RMSE	Avg MAE
50	0.835	0.677
100	0.809	0.603
150	0.789	0.584
200	0.768	0.551

Table 6.2: Avegare RMSE and MAE with respect to different n_factors of Matrix Factorization Model

Increase in n_factors result in highest decrease rate of MAE and RMSE. This suggest a strong dependent between the numbers of latent factor and the overall accuracy of the prediction.

param_n_epochs	Avg RMSE	Avg MAE
40	0.812	0.602
50	0.811	0.589
60	0.810	0.589

Table 6.3: Avegare RMSE and MAE with respect to different param_n_epochs of Matrix Factorization Model

param_lr_all	Avg RMSE	Avg MAE
0.02	0.769	0.562
0.04	0.768	0.561
0.06	0.770	0.562
0.08	0.771	0.565
0.10	0.779	0.569

Table 6.4: Avegare RMSE and MAE with respect to different param_lr_all of Matrix Factorization Model

param_reg_all	Avg RMSE	Avg MAE
0.01	0.816	0.596
0.02	0.799	0.579
0.03	0.776	0.567

Table 6.5: Avegare RMSE and MAE with respect to different param_reg_all of Matrix Factorization Model

The remaining parameters has less significant contribute to the result, with an exception of regularization value decrease the error greatly when set to 0.02. It is evident from the result that we should invest in the number of latent factor for good approximation, and sacrifices others parameter to reduce computational power.

Model	RMSE	MAE
Group: 50-100		
SVR	0.8381	0.6365
KNN	0.8332	0.6551
Stacking	0.8349	0.6676
Group: 100-200		
SVR	0.8051	0.6095
KNN	0.8068	0.6296
Stacking	0.8063	0.6329
Group: 200-500		
SVR	0.8024	0.6168
KNN	0.7679	0.5950
Stacking	0.7629	0.5946
Group: 501-1000		
SVR	0.7672	0.5898
KNN	0.7670	0.5960
Stacking	0.7348	0.5735
Group: 1001-12000		
SVR	0.7048	0.5347
KNN	0.7237	0.5582
Stacking	0.7588	0.5949

Table 6.6: Performance metrics for collaborative filtering models.

Chapter 7

Discussion

7.1 Overview

The results from our experiments on movie recommendation using various regression techniques and ensemble methods provide a comprehensive understanding of the strengths and limitations of each approach. Our primary focus was on comparing the performance of Ridge Regression, k-Nearest Neighbors (KNN), SVR, Matrix Factorization, Stacking, Random Forests, and LightGBM, all of which were evaluated using mean absolute error (MAE) and root mean squared error (RMSE) as the performance metrics. The evaluation was conducted across different dataset sizes, categorized into five distinct groups to analyze the scalability and robustness of each model.

7.2 Key Findings

7.2.1 Content-based

Ridge Regression: Ridge Regression demonstrated relatively consistent performance across all dataset sizes. It achieved the best results in the largest group (1001-12000) with a Test MAE of 0.5847 and Test RMSE of 0.7429.

The model's performance declined with smaller datasets, as evident from the Test MAE and RMSE values in the 50-100 group. This suggests that Ridge Regression benefits from larger datasets, where it can leverage more information to regularize the model effectively.

k-Nearest Neighbors (KNN): KNN's performance was generally poorer compared to Ridge Regression, particularly in smaller groups. For instance, in the 50-100 group, it had a Test MAE of 0.7050 and Test RMSE of 0.8751.

The model performed relatively better in larger datasets (1001-12000), with a Test MAE of 0.6321 and Test RMSE of 0.8006. This indicates that KNN may require a significant amount of data to capture the underlying patterns effectively.

Random Forests: Random Forests showed a reasonable balance between bias and variance, with its performance being quite competitive across different dataset sizes. For example, in the 1001-12000 group, it achieved a Test MAE of 0.6195 and Test RMSE of 0.7887.

However, the performance drop in smaller datasets was less pronounced compared to KNN, making Random Forests a more robust choice for varying data sizes.

LightGBM: LightGBM exhibited strong performance across all dataset sizes, closely trailing Ridge Regression in the largest group with a Test MAE of 0.6408 and Test RMSE of 0.8064.

The model's ability to handle large datasets efficiently and its gradient boosting framework's inherent feature selection contributed to its strong performance.

7.2.2 Collaborative Filtering

KNN: The model shows fairly good results for users with small to medium number of movies watched, and average results for big datasets.

SVR: SVR work best for users with less than 200 movies or more than 1000 movies watched, but is not suitable for medium datasets.

Stacking: Stacking does not show good results for very small or very big datasets. However, in terms of users with medium number of movies watched, it performs best among the models used, especially with users in group 501-1000.

Matrix Factorization: Matrix Factorization have fast computation time and less strain on the hardware of the recommender. However, its linearity result in less accurate prediction compare to other algorithm. Matrix Factorization also relies heavy on pass rating of user so it will not perform well on new user.

7.3 Future Work

Future research could explore hybrid models that combine the strengths of collaborative filtering and content-based models. Additionally, incorporating deep learning techniques and leveraging embeddings from neural networks could further enhance the recommendation system's accuracy and personalization capabilities.

Furthermore, experimenting with advanced hyperparameter tuning methods and ensemble strategies like stacking could provide deeper insights and potentially better-performing models. Lastly, considering user feedback and real-time adaptation mechanisms would be valuable for developing more dynamic and responsive recommendation systems.

Bibliography

[1] C. C Aggarwal. *Recommender Systems.* Cham, Switzerland: Springer International Publishing, 2016.

[2] grouplens. *MovieLens*. URL: https://grouplens.org/datasets/movielens/.

[3] *lightgbm.LGBMRegressor*. URL: https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMRegressor.html.

[4] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: (2009).

[5] Than Quang Khoat et al. *Nhap Mon Hoc May Va Khai Pha Du Lieu*. Dai Hoc Bach Khoa Ha Noi, 2024.

[6] T. Hastie, R. Tibshirani, and J. H Friedman. *"10. Boosting and Additive Trees". The Elements of Statistical Learning (2nd ed.).* New York: Springer., 2009.

[7] Wikimedia Commons. *Traditionally, decision trees have been created manually.* 2007. URL: https://en.wikipedia.org/wiki/Decision_tree.