DICKINSON COLLEGE

PROJECT REPORT

Freelancer Rehire Rate Estimation in Gig Economy

Course: Statistical and Machine Learning
Supervisor: Assoc. Prof. Lulu Wang

Authors:	Student ID:
Dac Kien Nguyen	900970120

Carlisle, May 2025

ABSTRACT

As digital labor platforms scale to accommodate a growing freelancer workforce, understanding and anticipating key behavioral metrics has become central to platform optimization. Among these metrics, rehire rate — defined as the frequency with which a freelancer is selected for subsequent engagements—serves as a proxy for client satisfaction, freelancer reliability, and long-term marketplace value. In this work, I present a machine learning framework for predicting freelancer rehire rate using structured tabular data. Our dataset consists of approximately 2,000 freelancer instances, each represented by a heterogeneous set of numerical and categorical features encompassing project history, economic signals, and client interaction attributes. We formulate the task as a supervised regression problem and construct a modeling pipeline that includes feature normalization, one-hot encoding, and domain-informed feature engineering. Multiple regression models are benchmarked, including linear regression, regularized baselines, and decision-tree-based architectures.

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

Freelance labor platforms now account for a significant share of global employment, offering scalable and flexible work arrangements to millions of users. As participation grows, so does the need for data-driven infrastructure to assess freelancer performance and forecast engagement outcomes. Unlike static resume-like profiles, modern platforms generate fine-grained behavioral logs, enabling predictive modeling of key freelancer metrics. One such metric is rehire rate, which quantifies the likelihood that a freelancer will be selected again by the same or similar clients. High rehire rates are indicative of both platform and freelancer success, and serve as useful targets for retention systems, recommendation algorithms, and labor quality assessment.

Predicting rehire rate is non-trivial. The task involves modeling over a high-dimensional, mixed-type feature space with incomplete and noisy labels. Standard rule-based or heuristic strategies fail to capture the nuanced dependencies between variables such as job completion rate, client satisfaction, marketing effort, and earnings trajectory. Furthermore, overfitting and target leakage are recurrent challenges, especially in domains where engineered features can inadvertently encode the target variable.

This project formulates rehire rate prediction as a supervised regression problem and proposes a pipeline-based solution with interpretable, low-bias models. By carefully constructing input features and benchmarking across multiple regression algorithms, we aim to assess the viability of real-time rehire forecasting in production-grade systems. Our ultimate goal is to contribute a machine learning framework for intelligent freelancer ranking, churn mitigation, and talent lifecycle management within gig economy.

CHAPTER 2. FEATURE ANALYSIS

For this project, I use dataset Freelancer Earnings Job Trends [1]. This dataset provides comprehensive information about freelancer earnings and job trends across various industries and skill categories. It aims to help job seekers, freelancers, and researchers understand compensation patterns and demand in the gig economy.

2.1 List of Features

I begin by listing all the features present in the dataset. These features are assumed to be extracted from the raw data and may include both numerical and categorical variables.

- Feature 1: Job Category
- Feature 2: Platform
- Feature 3: Experience Level
- Feature 4: Client Region
- Feature 5: Payment method
- Feature 6: Job Completed
- Feature 7: Earnings USD
- Feature 8: Hourly Rate
- Feature 9: Job Successful Rate
- Feature 10: Client Rating
- Feature 11: Job Duration Days
- Feature 12: Project Type
- Feature 13: Marketing Spend

Those features could be divided into 2 main categories:

- **Categorical Features**: Job Category, Platform, Experience Level, Client Region, Payment Method, Project Type
- **Numerical Features**: Job Completed, Hourly Rate, Job Successful Rate, Client Rating, Job Duration Days, Rehire Rate

I figure out that the distribution of each feature is fairly even. Below, I show some distributions of some features:

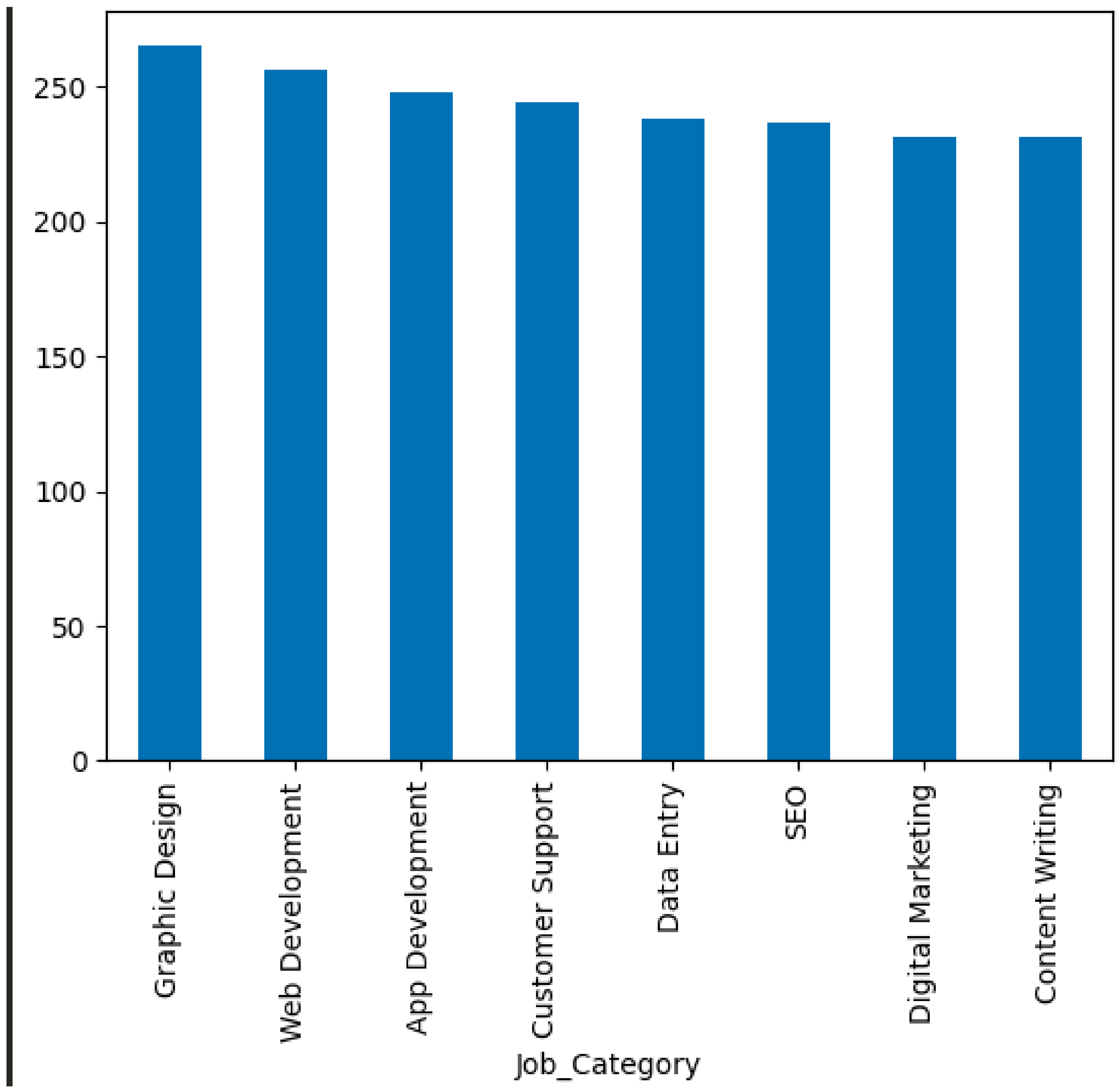

Figure 2.1: Value distribution of Job Category feature

2.2 Correlation Among Features

To understand the relationships between features, we compute the Pearson correlation coefficient ρ for each pair of numerical features:

$$\rho_{X,Y} = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y}$$

Below is the correlation figure, before I applied feature engineering technique:

2.3 Feature Engineering and Transformation

To improve the quality and predictive power of the dataset, we apply both manual feature engineering and automated transformations.

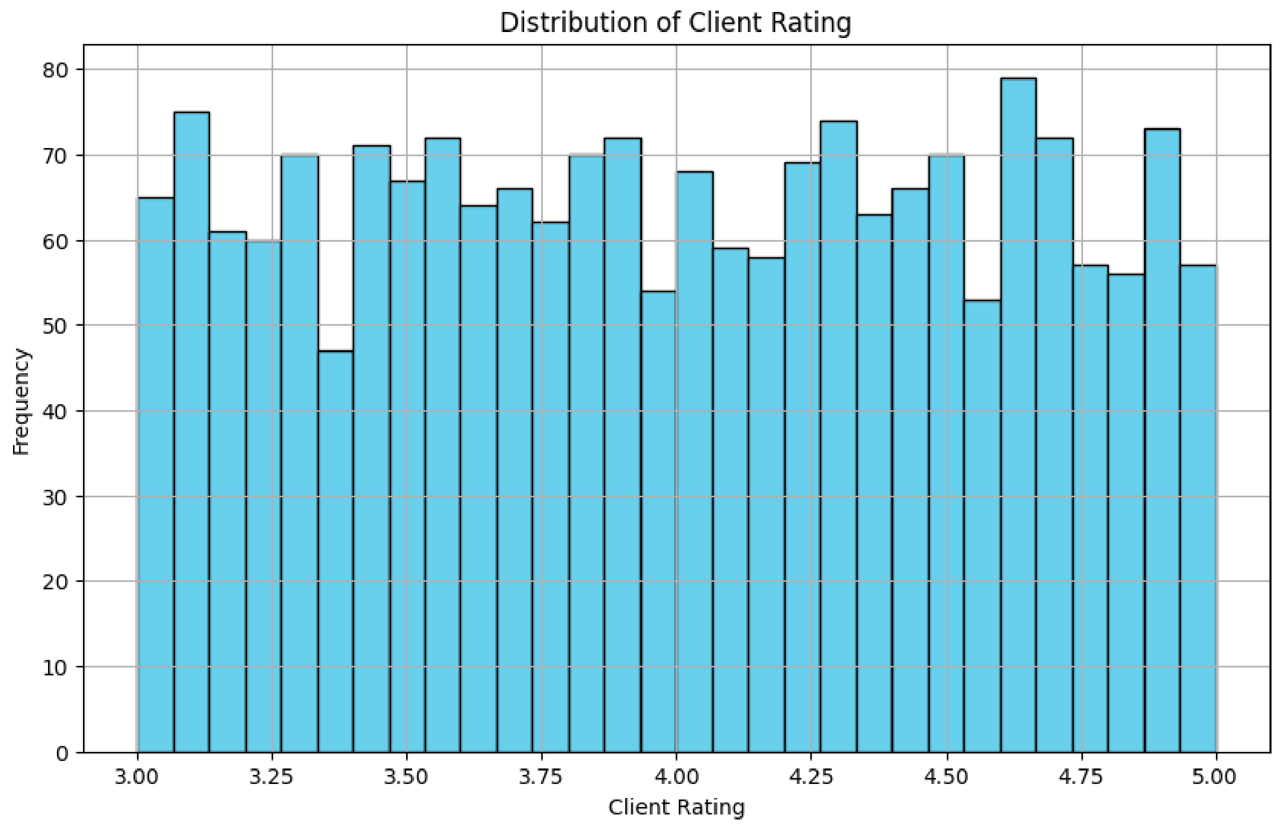

Figure 2.2: Value distribution of Client Rating feature

New Engineered Features

The following new features are created to capture important interactions and efficiency metrics:

- **Earnings per Job:** Measures average earnings per completed job to normalize across freelancers.

$$\text{Earnings\_per\_Job} = \frac{\text{Earnings\_USD}}{\text{Job\_Completed} + 1}$$

- **Earnings per Day:** Reflects efficiency by showing how much a freelancer earns per day of job duration.

$$\text{Earnings\_per\_Day} = \frac{\text{Earnings\_USD}}{\text{Job\_Duration\_Days} + 1}$$

- **Marketing Efficiency:** Quantifies return on marketing investment.

$$\text{Marketing\_Efficiency} = \frac{\text{Earnings\_USD}}{\text{Marketing\_Spend} + 1}$$

- **Success $\times$ Rehire Rate:** Captures combined quality and client retention.

$$\text{Success\_x\_Rehire} = \text{Job\_Success\_Rate} \times \text{Rehire\_Rate}$$

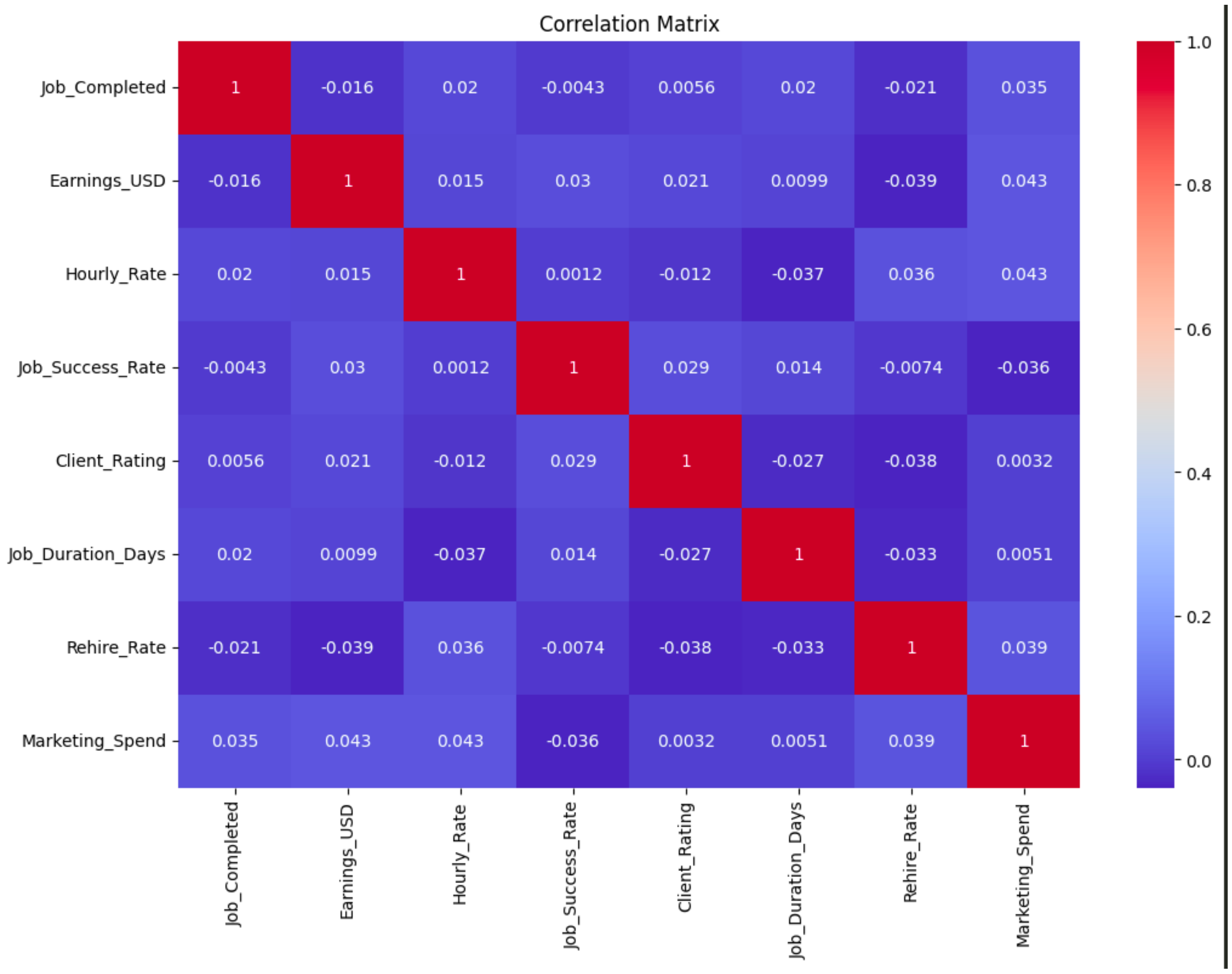

Figure 2.3: Correlation of feature before engineering

- **Log Earnings:** Applies logarithmic transformation to reduce skewness in earnings distribution.

$$\text{Log\_Earnings\_USD} = \log(1 + \text{Earnings\_USD})$$

Data Transformation Pipeline

We construct a preprocessing pipeline using the `ColumnTransformer` to handle numerical and categorical variables differently:

- **Numerical Features:**
  1. **Standard Scaling:** Normalize the distribution using `StandardScaler`.
  2. **Polynomial Features:** Add second-order interaction terms via `PolynomialFeatures` $(\text{degree} = 2)$ without bias.
  3. **Gaussian Noise Injection:** Slightly perturb data to improve model robustness, using a small noise standard deviation of 0.001.
- **Categorical Features:**

  - **One-Hot Encoding:** Convert categorical variables to binary format with `OneHotEncoder`, while handling unknown categories gracefully.

This preprocessing ensures all features are well-scaled, expressive, and robust for downstream modeling.

3.1 Linear Regression

Linear Regression is a supervised learning algorithm that models the relationship between a scalar dependent variable y and one or more independent variables denoted by $\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top$. The objective is to find a linear function that best fits the data, usually by minimizing the residual sum of squares between the observed targets and the targets predicted by the linear approximation [2].

The hypothesis function for linear regression can be written as:

$$\hat{y} = \mathbf{w}^\top \mathbf{x} + b \tag{3.1}$$

where $\mathbf{w} \in \mathbb{R}^n$ is the vector of weights, and $b \in \mathbb{R}$ is the bias term.

Given a dataset $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^m$, where m is the number of training examples, the cost function (Mean Squared Error) is defined as:

$$J(\mathbf{w}, b) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)^2 = \frac{1}{2m} \sum_{i=1}^{m} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)} \right)^2 \tag{3.2}$$

The optimal parameters $\mathbf{w}$ and b are typically found by minimizing this cost function using analytical methods (e.g., Normal Equation) or iterative methods (e.g., Gradient Descent).

The Normal Equation provides a closed-form solution:

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \tag{3.3}$$

assuming $\mathbf{X}^\top \mathbf{X}$ is invertible, where $\mathbf{X}$ is the design matrix and $\mathbf{y}$ is the vector of target values.

Assumptions

Linear regression relies on several key assumptions:

- Linearity: The relationship between inputs and output is linear.
- Independence: Observations are independent.
- Homoscedasticity: Constant variance of errors.
- Normality: Errors are normally distributed.

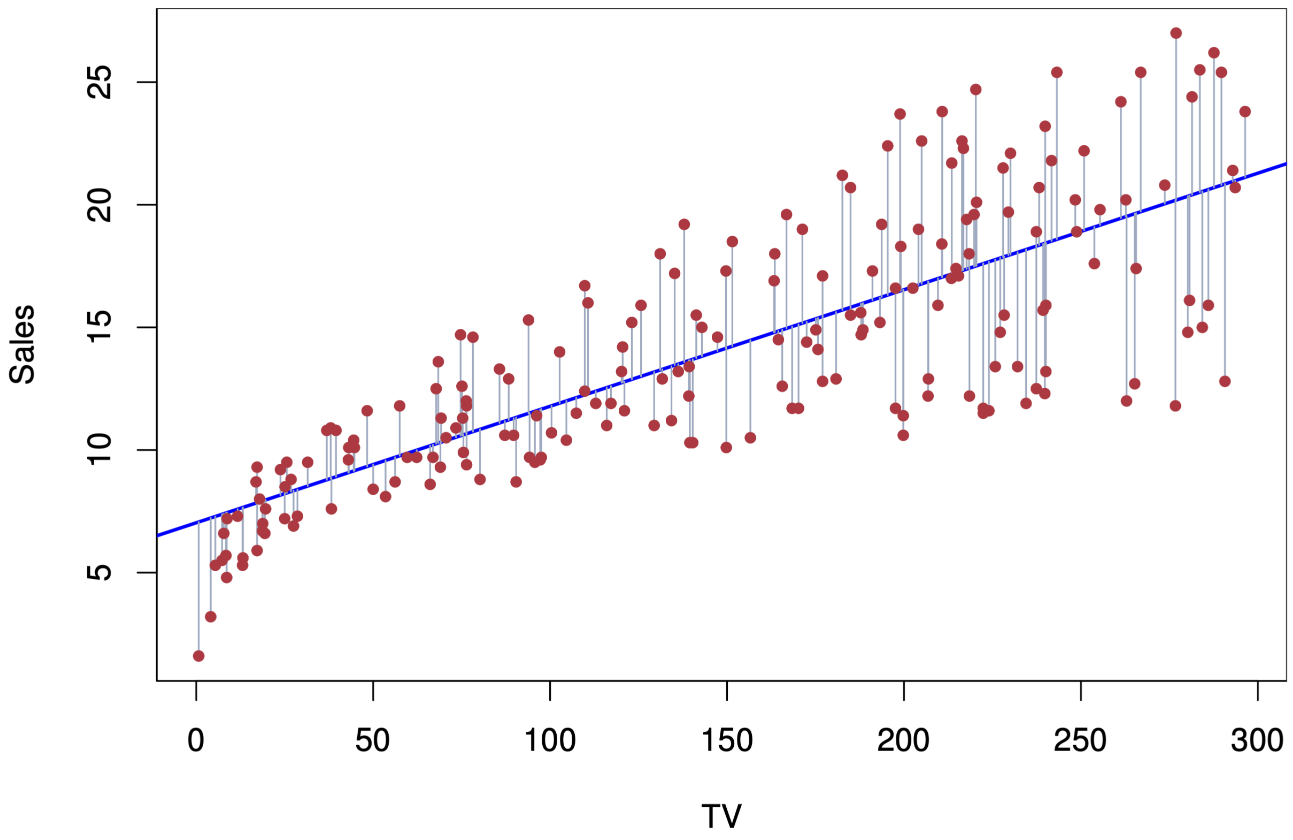

Figure 3.1: Linear Regression

3.2 Decision Tree and Random Forest

3.2.1 Decision Tree

A Decision Tree is a supervised learning algorithm used for both classification and regression tasks. It splits the input space recursively into regions with homogeneous labels. The algorithm selects the best feature and threshold to minimize an impurity measure such as Gini index or entropy [3].

Impurity Measures

Given a dataset D with K classes, the Gini index is defined as:

$$Gini(D) = 1 - \sum_{k=1}^{K} p_k^2$$

where p_k is the proportion of class k in D.

Entropy is defined as:

$$Entropy(D) = -\sum_{k=1}^{K} p_k \log_2 p_k$$

Recursive Splitting Rule

Let A be the set of attributes. At each node, the best split (a^*, t^*) is selected as:

$$(a^*, t^*) = \arg \min_{a \in A, t} \left[\frac{|D_{left}|}{|D|} \cdot I(D_{left}) + \frac{|D_{right}|}{|D|} \cdot I(D_{right}) \right]$$

where $I(D)$ is the impurity (e.g., Gini or Entropy), and D_{left}, D_{right} are the partitions of D induced by threshold t on feature a.

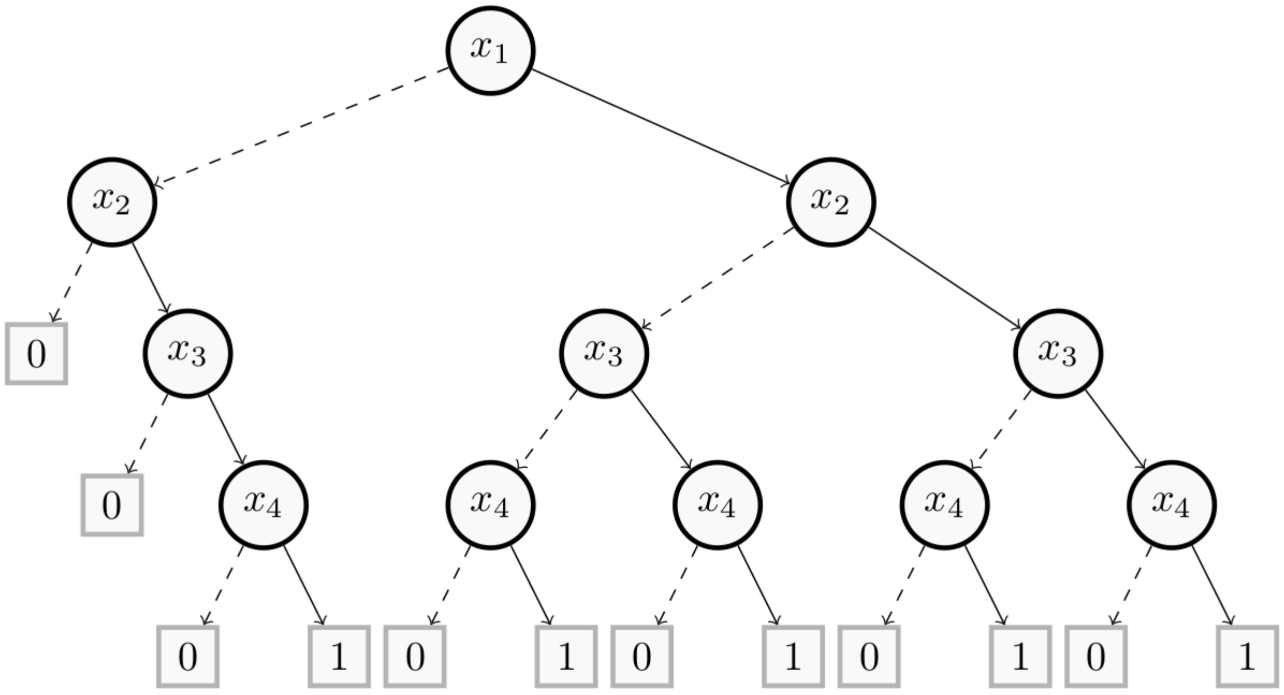

Figure 3.2: Decision Tree

3.2.2 Random Forest

A Random Forest is an ensemble method that constructs a multitude of decision trees during training and outputs the mode (for classification) or mean (for regression) of the predictions [4].

Algorithm Overview

- For B iterations:
  1. Sample a bootstrap dataset D_b from D.
  2. Train a decision tree T_b on D_b using a random subset of features at each split.
- Output prediction:

$$\hat{y} = \text{mode}(\{T_b(x)\}_{b=1}^{B}) \quad \text{(classification)}$$

or

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} T_b(x) \quad \text{(regression)}$$

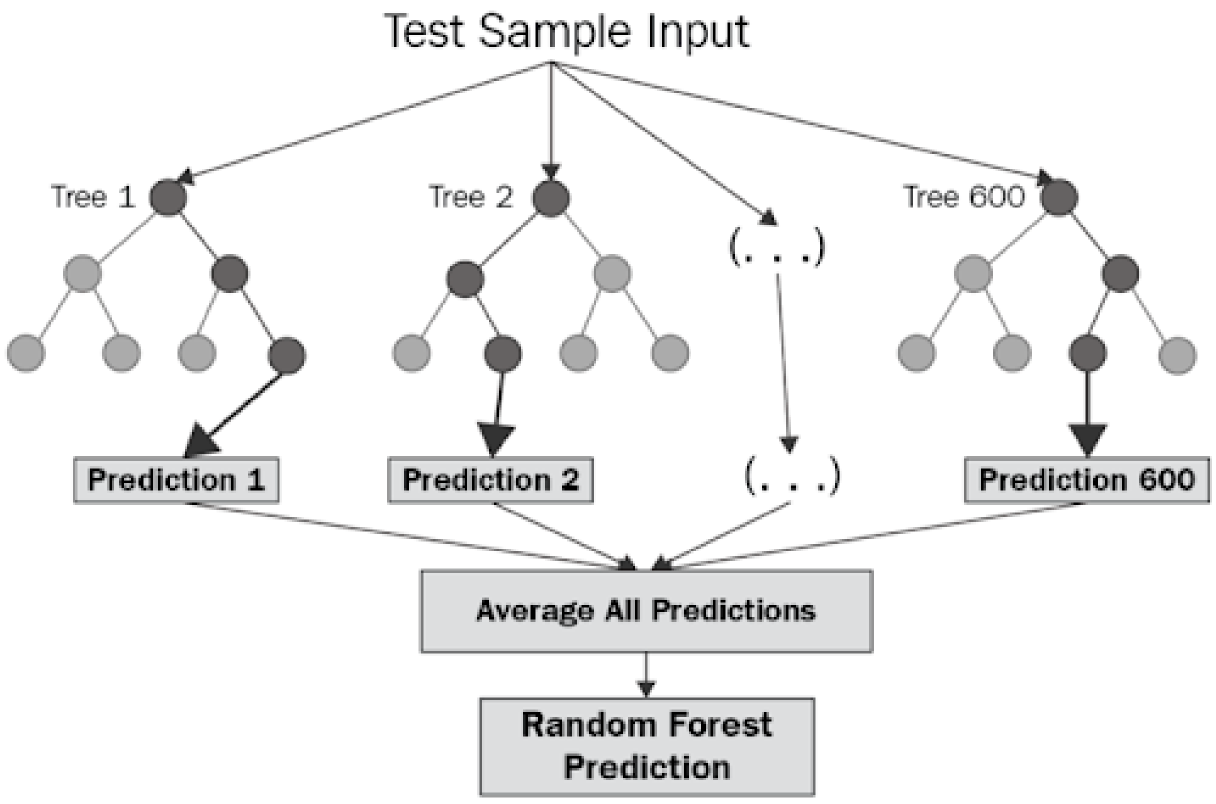

Figure 3.3: Random Forest

3.3 Gradient Boosting

Gradient Boosting is an ensemble learning technique that builds a strong predictive model by sequentially adding weak learners, typically decision trees. Each new model is trained to correct the residual errors of the combined ensemble of previous models [5].

3.3.1 Algorithm Overview

Given a dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$, and a loss function $L(y, \hat{y})$, Gradient Boosting minimizes the expected value of the loss function using gradient descent in function space.

Initialization

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

Iterative Procedure

For $m = 1$ to M (number of boosting rounds):

1. Compute the pseudo-residuals:

$$r_i^{(m)} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$$

2. Fit a weak learner (e.g., decision tree) $h_m(x)$ to the residuals:

$$h_m(x) \approx r^{(m)}$$

3. Compute a multiplier γ_m:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$$

4. Update the model:

$$F_m(x) = F_{m-1}(x) + \eta\gamma_m h_m(x)$$

where $\eta \in (0, 1]$ is the learning rate.

Final Model

$$F_M(x) = \sum_{m=1}^{M} \eta\gamma_m h_m(x)$$

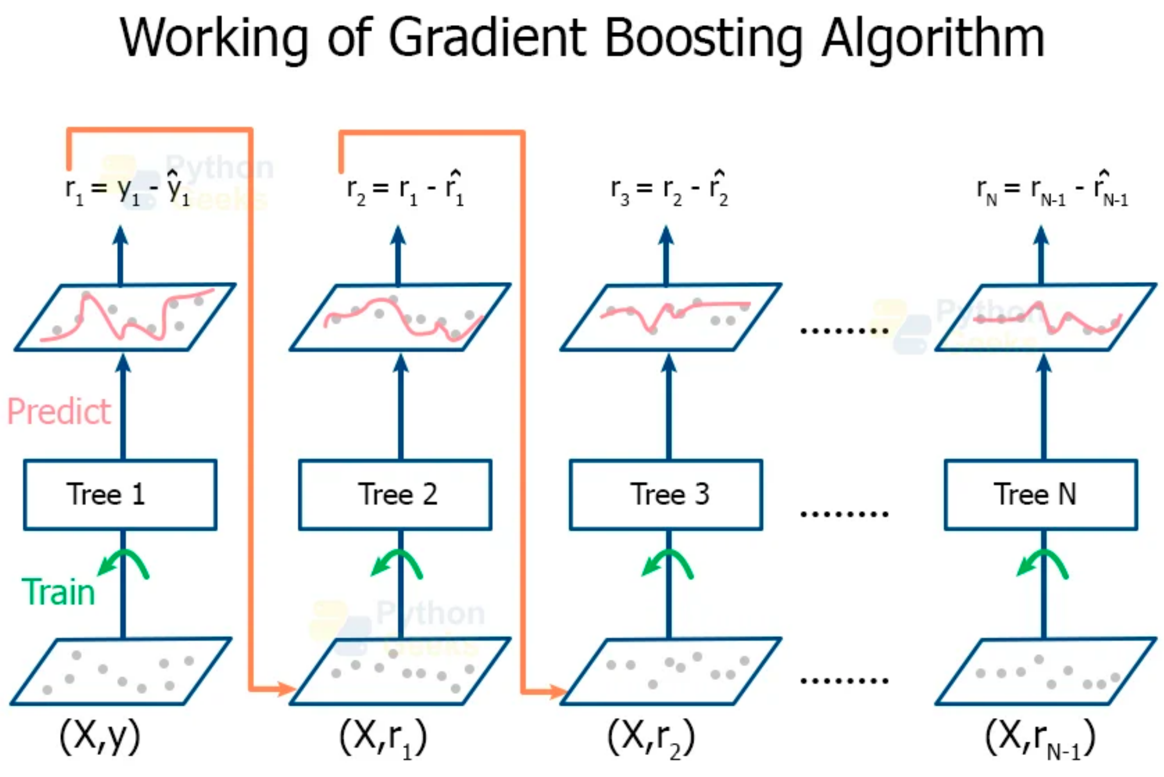

Figure 3.4: Gradient Boosting

3.4 Multilayer Perceptron

A **Multilayer Perceptron** (MLP) is a class of feedforward artificial neural networks consisting of an input layer, one or more hidden layers, and an output layer. Each layer is fully connected to the next one [6].

3.4.1 Architecture

An MLP maps an input vector $x \in \mathbb{R}^d$ to an output $y \in \mathbb{R}^k$ through a series of linear transformations and nonlinear activations.

$$\begin{aligned} h^{(1)} &= \sigma(W^{(1)}x + b^{(1)}) \\ h^{(2)} &= \sigma(W^{(2)}h^{(1)} + b^{(2)}) \\ &\vdots \\ \hat{y} &= f(W^{(L)}h^{(L-1)} + b^{(L)}) \end{aligned}$$

where:

- $W^{(l)}$, $b^{(l)}$ are the weight matrix and bias vector at layer l
- σ is a nonlinear activation function (e.g., ReLU, sigmoid, tanh)
- f is the output activation (e.g., softmax for classification)

3.4.2 Activation Functions

- **ReLU:** $\sigma(z) = \max(0, z)$
- **Sigmoid:** $\sigma(z) = \frac{1}{1+e^{-z}}$
- **Tanh:** $\sigma(z) = \tanh(z)$

3.4.3 Loss Function and Training

The network is trained using gradient descent to minimize a loss function $L(y, \hat{y})$, such as:

$$\text{Cross-Entropy: } L(y, \hat{y}) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$$

$$\text{MSE: } L(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Gradients are computed via the **backpropagation algorithm**, using the chain rule to propagate error backward through the network.

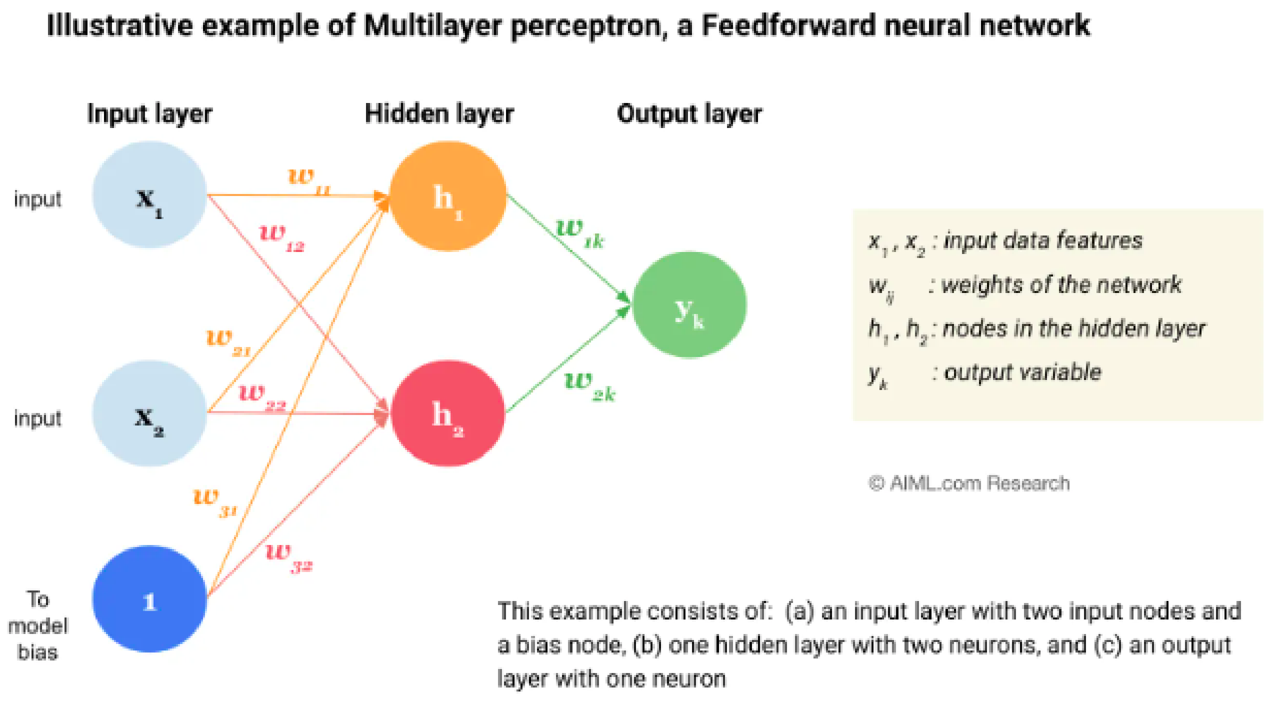
Illustrative example of Multilayer perceptron, a Feedforward neural network
Input layer
Hidden layer
Output layer
input
input
To model bias
x_1
x_2
1
h_1
h_2
y_k
w_{11}
w_{12}
w_{21}
w_{22}
w_{31}
w_{32}
w_{1k}
w_{2k}
x_1, x_2 : input data features
w_{ij} : weights of the network
h_1, h_2 : nodes in the hidden layer
y_k : output variable
© AIML.com Research
This example consists of: (a) an input layer with two input nodes and a bias node, (b) one hidden layer with two neurons, and (c) an output layer with one neuron

Figure 3.5: Multilayer Perceptron

CHAPTER 4. MODEL SELECTION AND RESULT

4.1 Model Performance Summary

I evaluated multiple regression models, as mentioned before in theoretical section,to predict Rehire_Rate, all yielding similar poor performance with negative R^2 scores (worse than a baseline mean predictor) and RMSE values around 20-21:

- **Linear Regression**: RMSE ≈ 21.17, $R^2 \approx -0.058$
- **Decision Tree**: RMSE ≈ 28.3, $R^2 \approx -0.884$ (extreme overfitting)
- **Random Forest**: RMSE ≈ 20.65, $R^2 \approx -0.037$ (best performer, marginally)
- **Gradient Boosting**: RMSE ≈ 21.86, $R^2 \approx -0.127$
- **HistGradientBoosting**: RMSE ≈ 20.91, $R^2 \approx -0.032$
- **Neural Network (MLP)**: Failed to converge, poor R^2

4.2 Why All Models Underperform

The consistent negative R^2 values across all models indicate that the dataset lacks predictive signal for Rehire_Rate:

1. **Lack of signal in features**: Near-zero correlations between features and target ($|\text{corr}| < 0.04$)
2. **Model limitations**: Complex models overfit noise while simple models underfit
3. **Metric analysis**: RMSE approximately equals target's standard deviation ($\sim$20.2), confirming near-total unexplained variance

4.3 Dataset Diagnostics

- **Target distribution**: Roughly uniform between 10-80 (mean $\sim$44.6), no obvious outliers
- **Feature correlations**: Negligible linear relationships with target
- **Multicollinearity**: Not an issue (all $p < 0.05$)
- **Feature quality**: Engineered variables provide minimal improvement

CHAPTER 5. CONCLUSION

This project aimed to predict freelancer rehire rates using multiple regression models, including Linear Regression, Decision Trees, Random Forests, Gradient Boosting, and Neural Networks. Our comprehensive analysis revealed significant challenges in developing an accurate predictive model for this target variable.

The consistent negative R^2 values across all models (ranging from -0.032 to -0.884) and high RMSE values (approximately 20-21) demonstrate that the current feature set lacks sufficient predictive power for rehire rates. Even sophisticated ensemble methods and deep learning approaches could not extract meaningful patterns from the available data. This suggests that freelancer rehire rates may be influenced primarily by factors not captured in our dataset, such as client-specific preferences, project-specific requirements, or qualitative aspects of freelancer performance that are difficult to quantify.

Our analysis of feature importance and correlation revealed minimal relationships between the available predictors and rehire rates. The failure of polynomial features, feature engineering, and various model architectures to improve performance further confirms that the signal-to-noise ratio in the current dataset is too low for effective prediction.

In practical terms, these findings suggest that businesses and platforms seeking to predict freelancer rehire rates should approach purely quantitative models with caution. The current state of modeling indicates that rehire decisions may be more complex and nuanced than can be captured by the available metrics.

5.1 Future Development

While our current models failed to achieve satisfactory predictive performance, several promising directions for future development exist:

5.1.1 Data Enrichment

- **Client-side variables**: Incorporate features related to clients' historical hiring patterns, industry, company size, and project requirements
- **Communication metrics**: Include measures of communication frequency, response times, and sentiment analysis of client-freelancer interactions
- **Qualitative feedback**: Convert qualitative feedback into quantitative features using natural language processing techniques
- **Temporal features**: Develop features that capture timing aspects, such as market conditions, seasonality, or trending skills

5.1.2 Advanced Modeling Approaches

- **Multi-task learning**: Simultaneously predict related outputs (e.g., client satisfaction, project completion success) alongside rehire rate
- **Domain adaptation**: Develop transfer learning techniques to leverage patterns across different freelancer categories or platforms
- **Survival analysis**: Reframe the problem as time-to-rehire prediction using survival analysis techniques
- **Causal inference**: Apply causal inference methods to understand factors that *cause* rehiring rather than just correlate with it

5.1.3 Alternative Problem Formulations

- **Classification approach**: Convert rehire rate into categorical outcomes (e.g., low, medium, high) to simplify the prediction task
- **Recommendation system**: Develop a freelancer-client matching system based on compatibility rather than direct rehire prediction
- **Cluster analysis**: Identify distinct freelancer profiles that exhibit different rehiring patterns and develop segment-specific models
- **Explanatory modeling**: Shift focus from prediction to explanation, identifying even small effects that might inform business understanding

5.1.4 Practical Applications

Even with limited predictive power, insights from this analysis can guide platform development:

- **Transparent metrics**: Develop more transparent metrics that correlate better with rehiring decisions
- **Feature importance monitoring**: Track which features show even slight correlations with rehiring over time
- **User education**: Educate freelancers about factors that might improve their rehire potential based on domain knowledge
- **Client behavior analysis**: Study client decision patterns to understand the human factors behind rehiring decisions

In conclusion, while the current dataset doesn't support accurate rehire rate prediction, this project establishes a foundation for more sophisticated approaches. By combining additional data sources, advanced modeling techniques, and alternative problem formulations, future work may uncover previously hidden patterns in freelancer rehire dynamics.

REFERENCES

Bibliography

[1] S. P. Shohan, *Freelancer earnings and job trends*, https://www.kaggle.com/datasets/shohinurpervezshohan/freelancer-earnings-and-job-trends, Accessed: 2025-05-11, 2025.

[2] A. Shrestha, A. Gogoi **and** A. K. Chaubey, "Using linear regression machine learning algorithm for the prediction of real estate," *ScienceOpen*, 2020. **url**: https://www.scienceopen.com/hosted-document?doi=10.14293/S2199-1006.1.SOR-.PP6RJWG.v1.

[3] K. J. Cios, W. Pedrycz, R. W. Swiniarski **and** L. A. Kurgan, "Supervised learning: Decision trees, rule algorithms, and their hybrids," *Data Mining*, **pages** 134–167, 2007. DOI: 10.1007/978-0-387-36795-8_12. **url**: https://link.springer.com/chapter/10.1007/978-0-387-36795-8_12.

[4] L. Breiman, "Random forests," *Machine Learning*, **jourvol** 45, **number** 1, **pages** 5–32, 2001. DOI: 10.1023/A:1010933404324. **url**: https://link.springer.com/article/10.1023/A%3A1010933404324.

[5] A. Natekin **and** A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in Neurorobotics*, **jourvol** 7, **page** 21, 2013. DOI: 10.3389/fnins.2013.00021. **url**: https://www.frontiersin.org/articles/10.3389/fnins.2013.00021/full.

[6] S. Haykin, *Neural Networks: A Comprehensive Foundation.* Prentice Hall PTR, 1994, ISBN: 9780132733502.