Titanic: Machine Learning from Disaster

System Analysis Project

Julián Carvajal Garnica
20242020024

Andrés Mauricio Cepeda Villanueva
20242020010

Jhonatan David Moreno Barragán
20201020094

Andrés Camilo Ramos Rojas
20242020005

School of Engineering, Universidad Distrital Francisco José de Caldas
System Analysis Course
Teacher: Carlos Andrés Sierra
Bogotá D.C.
2025

Part 2 of the System Analysis Project

1. Review of Workshop #1 Findings

The analytical work from Workshop #1 provided a comprehensive understanding of the Titanic competition as a dynamic and interdependent system. Rather than treating it as a simple prediction problem, the study highlighted structural dependencies, sensitivity points, and chaotic behaviors that influence model performance and reliability.

System Overview and Core Elements

The system's main objective is to predict passenger survival using multiple features — such as class, sex, age, and family relations — extracted from `train.csv`, `test.csv`, and `gender_submission.csv`. Each dataset interacts with the others through defined roles in the machine learning pipeline: data ingestion, preprocessing, model training, and evaluation within Kaggle's controlled environment.

Data Characteristics and Constraints

Several structural constraints were identified in the datasets:

- Missing or incomplete values in `Age`, `Cabin`, and `Embarked`
- Class imbalance biasing model learning
- Limited dataset size (approximately 1,300 samples), increasing risk of overfitting

Sensitivity and Complexity

The system displays high sensitivity to key predictors (`Sex`, `Pclass`, `Age`). Removing or misprocessing them decreases accuracy. Nonlinear relationships such as `Sex` $\times$ `Age` and `Pclass` $\times$ `Fare` create complex interactions requiring feature engineering and regularization.

Chaos and Randomness

Even structured datasets like Titanic's contain unpredictability. Passengers with similar profiles sometimes had opposite outcomes due to hidden variables. Family-related variables (`SibSp`, `Parch`) introduce feedback loops, reflecting emergent nonlinear behavior—an echo of chaos theory in data.

Implications for System Design

These findings guide the system design foundations: modularity, robustness, and controlled randomness. Each stage—from data ingestion to model evaluation—must minimize instability and maximize reproducibility.

2. System Requirements Definition

Functional Requirements

ID	Requirement	Description
FR-1	Data Ingestion Module	Load and validate input datasets (`train.csv`, `test.csv`, and `gender_submission.csv`).
FR-2	Preprocessing and Cleaning	Handle missing or null values in variables such as `Age`, `Cabin`, and `Embarked`.
FR-3	Feature Engineering	Apply transformations (e.g., one-hot encoding for categorical variables).
FR-4	Model Training	Train a supervised learning model (e.g., Random Forest) to predict `Survived`.
FR-5	Evaluation and Metrics	Compute model accuracy and generate submission file.
FR-6	Submission Output	Export predictions to `submission.csv` following Kaggle's structure.

Table 1: Functional Requirements

Non-Functional Requirements

ID	Requirement	Description
NFR-1	Performance	Process datasets (1,300 records) in less than 5 seconds.
NFR-2	Scalability	Allow easy addition of features without refactoring.
NFR-3	Reproducibility	Use fixed random seeds and documented dependencies.
NFR-4	Maintainability	Modular architecture separating stages.
NFR-5	Usability	Provide clear workflow and outputs.
NFR-6	Reliability	Handle corrupted input files gracefully.

Table 2: Non-Functional Requirements

Sensitivity-Driven Requirements

- **SR-1 – Controlled Randomness:** Use fixed random seeds in training.
- **SR-2 – Robust Feature Selection:** Include key predictors (`Sex`, `Pclass`, `Age`).
- **SR-3 – Missing Data Resilience:** Apply imputation strategies.
- **SR-4 – Monitoring and Feedback:** Log performance metrics to detect anomalies.

User-Centric Requirements

- **UR-1 – Simplicity:** Execute workflow in one notebook sequence.
- **UR-2 – Interpretability:** Provide explainable feature relationships.
- **UR-3 – Security:** Prevent exposure of private Kaggle data.

These requirements translate the *Workshop 1* insights into engineering goals that ensure an **accurate, reproducible, and resilient** predictive system.

3. High-Level Architecture

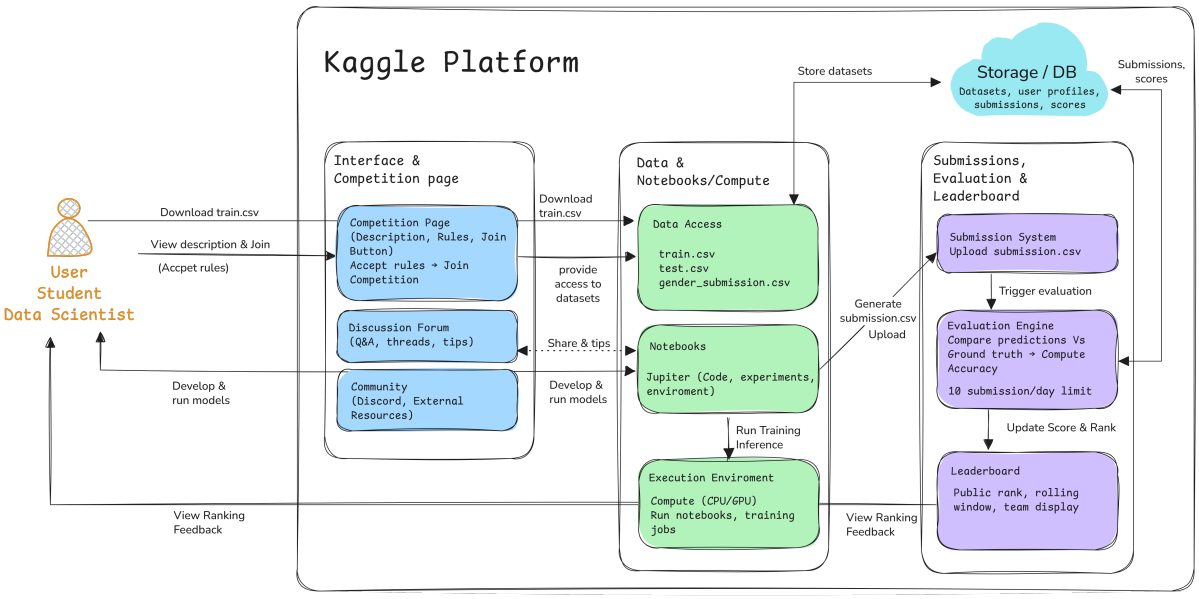

Figure 1: Architecture of the Kaggle competition environment.

Figure 1 illustrates the architecture of the Kaggle competition environment. Competitors access the competition page, download datasets, train models in notebooks, and submit predictions for evaluation. The system computes accuracy, updates leaderboards, and integrates community interaction via forums and repositories.

- **User:** The user will be an ordinary person regardless of their occupation; in reality, they only need to have an interest in the world of systems analysis, with access to practically the entire system except for certain interactions that will be carried out internally by the system.
- **Interface & competition page:** This module allows the user to interact directly with the competency environment, meaning that they are allowed to access the competency and view all the necessary information to understand the environment. Additionally, there are community environments that allow users to ask questions about the competencies.
- **Data & Notebooks/Compute:** This module allows the user to dive into the competition, accessing the data that will be used, the testing environment, and resources such as the notebooks provided by Kaggle for a better understanding of the competition.
- **Storage/DB:** This module is not directly accessible to the user but interacts with them indirectly by sending information that will be stored for later viewing, as in the leaderboard or in the different saved files.

- **Submissions, Evolution & Leaderboard:** This module allows the user to finalize their attempt in the competition by submitting the presented model to improve the prediction algorithm, as well as assigning a ranking by comparing it with the other submitted solutions.

4. Addressing Sensitivity and Chaos

It is inevitable to deal with random factors. In the case of the Titanic, we can imagine damaged gates, someone being far from the escape routes, falling awkwardly into the water and being unable to stay afloat. There are many factors that can have an impact. Likewise, we find nonlinear relationships which will be somewhat random since there is nothing that can describe them accurately, as a linear function would. For all the above reasons, a robust and well-divided system is proposed in which it is possible to identify the implications of the elements among themselves more clearly, aiming for a better approximation of who has the capacity to survive and who does not. This is evidenced through the tests provided by the competition, where cases of people with very similar characteristics but opposite outcomes are found, although we also understand that it could be due to a lack of consideration of important variables.
It would be extremely positive to start taking into account those values that we find to be random for a more exhaustive review and to find out which other variables could affect both sensitivity, as well as an unexpected variable or others.

5. Technical Stack and Implementation Sketch

The system will be created using Python, primarily due to its extensive range of libraries for data analysis and machine learning, and its compatibility with Kaggle's environment. This language facilitates a structured workflow and simplifies future maintenance and updates of the system.

During the data ingestion and preprocessing phases, the libraries Pandas and NumPy will facilitate the loading, cleaning, and transformation of the datasets (train.csv, test.csv, and *gender_submission.csv*). Imputation techniques will be applied to handle missing values in variables like Age, Cabin, and Embarked, while one-hot encoding will be used for processing categorical data. To gain a clearer insight into the data distribution and identify correlations or outliers, visualization tools such as Matplotlib and Seaborn will be utilized.

Scikit-learn will serve as the primary framework for model training and evaluation, as it offers various classification algorithms like Random Forest and Logistic Regression, which are suitable for this issue. Cross-validation will be applied to verify model reliability

and minimize overfitting. The system will assess performance using metrics like accuracy and F1-score to guarantee the model's dependability.

Ultimately, the deployment stage will utilize Flask to develop an API that can produce survival predictions using fresh passenger information. In the future, the system might be containerized with Docker to enhance scalability and simplify environment replication

The execution will adopt a pipeline framework, segmenting the system into structured and self-sufficient modules that can collaborate effectively. This modular structure simplifies workflow maintenance and enables rapid changes without impacting the whole system.

The procedure begins with data ingestion, during which all input files are loaded and checked for accuracy. Following that is data preprocessing, which sanitizes the data, substitutes missing values, and encodes categorical variables. Subsequently, feature engineering is utilized to generate new variables that enhance model performance. Subsequently, model training occurs, during which various algorithms are assessed and confirmed using cross-validation. The assessment stage gauges the model's effectiveness and records all pertinent metrics. Ultimately, in the deployment phase, the trained model is made available via an API that can deliver predictions automatically.

This method enhances the system's maintainability while also bolstering its capacity to handle unexpected or chaotic fluctuations in the data. Monitoring tools will assist in identifying these fluctuations and initiate updates or retraining if needed, guaranteeing that the model stays consistent and dependable over time. This implementation plan operationalizes the system architecture described earlier, ensuring that each module—from data ingestion to deployment—fulfills the requirements established in the design phase.