UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

FACULTAD DE INGENIERÍA
Ingeniería de Sistemas

Otto Group Product Classification Challenge, System Simulation

---

Systems Analysis & Design

Group 020-82

Workshop #4

Students

Juan Diego Lozada 20222020014
Juan Pablo Mosquera 20221020026
María Alejandra Ortiz Sánchez 20242020223
Jeison Cuenca: 20242020043

Professor: Carlos Andrés Sierra Virguez

Date: November 2025

Contents

1 Data Preparation

Data preparation constitutes the first fundamental step to ensure the validity of the simulations and of the designed classification system. Since the project is based on the Otto Group Product Classification Challenge, the same official datasets from Kaggle were used, publicly available under the restrictions described in the previous Workshops.

1.1 Dataset Acquisition

To ensure consistency with Workshop #1, the official competition files were downloaded:

- train.csv: contains 93 obfuscated numerical variables and the target column, which represents the true product class.
- test.csv: contains the 93 numerical variables without labels.
- sampleSubmission.csv: reference file used to generate the final predictions.

The download was performed from the Kaggle platform, and the original structure was preserved to ensure fully reproducible processing. As established in the initial analysis, the id column was kept only for the final stage of generating the output file, since it provides no predictive value and must be removed during model training.

1.2 Data Cleaning and Preprocessing

In order to preserve system stability and ensure that the later phases of the project are not affected by data anomalies, various cleaning, verification, and transformation processes were applied, in accordance with the recommendations from Workshops 1, 2, and 3.

- **Integrity and consistency verification**
  The following checks were performed:
  - Missing values: The presence of null data in all columns was evaluated. Due to the nature of the dataset, it was confirmed that no empty values existed in any of the variables
  - Duplicates: The dataset was checked for duplicate rows; no repeated records were found
  - Identifier review: It was confirmed that the id column contains unique values, but it was excluded from the predictive variables as stipulated in the system analysis

  These validations align with the quality guidelines described in Workshop 3 (ISO 9000 and data flow integrity)

- **Removal of non-predictive variables**
  As established in Workshop #1, the variable id does not provide useful information for the classification process, so it was removed from the training and test datasets during preprocessing. Its use was reserved exclusively for the creation of the final predictions file

- **Normalization and scaling**
  Because the 93 numerical attributes present different ranges and distributions, and considering that the system must remain stable under small variations (sensitivity described in Workshops 1 and 3), a uniform scaling method was applied to all features:

  – A standard scaling method (StandardScaler) was used to transform each variable to mean 0 and standard deviation 1

  This procedure reduces extreme variability, improves the numerical stability of the algorithms, and decreases the risk of amplifying the chaotic behavior associated with small perturbations in the data

- **Optional dataset reduction for simulations**
  With the aim of performing controlled and computationally feasible simulations, a representative subset of the original dataset was created using a stratified sample based on the target column. This allowed for:

  – Maintain the real proportion among the 9 classes of the problem

  – Efficiently evaluate the complete system workflow without relying on the original 200,000 records

  – Control computational load during testing

  This reduction does not affect the final results, since it is applied only during the simulation stage and not in the construction of the final model

- **Dataset Summary**
  Finally, a statistical characterization of the dataset was developed in order to understand its structure and conditions before the simulations:

  – The dataset contains more than 200,000 records, each corresponding to a different product

  – There are 93 numerical variables, all of them obfuscated, without direct semantic meaning

  – No missing values or structural inconsistencies were detected.

  – The distribution of the target variable is not perfectly balanced, which means that some classes are overrepresented compared to others. This represents a challenge for the probabilistic calibration of the model, consistent with what was described in Workshop 1

  – The numerical variables exhibit heterogeneous amplitudes and scales, which fully justifies the need for the applied normalization process

  This characterization is essential both for the construction of the model and for the systemic interpretation of the behavior of the dataset, especially regarding the sensitivity and stability of the system analyzed previously

2 Simulation Planning

2.1 Scenario 1: Data-driven Simulation

Objective
Simulate the full training/validation/prediction cycle as in the Otto competition to evaluate: performance (logloss), probability calibration, stability under small preprocessing changes, and the computational power required

Used components

- Data Processing: loading, cleaning, scaling (StandardScaler)
- Feature Engineering: optional PCA or aggregated statistics
- Classification Engine: ML models (Random Forest, MLP, and a boosting model if desired)
- Analytics & Reporting: validation curves, confusion matrix, calibration, logloss
- Monitoring/Logging: record seeds, timings, memory usage, parameters (see Workshop 3)

Concrete pipeline

1. **Loading and splits:**
   - Load train.csv. Remove id
   - create split: stratified k-fold (k=5) for robust validation (avoids leaderboard overfitting)
   - Keep a final holdout (10% of the train) if you want an extra control
2. **Preprocessing:**
   - Imputation: if there are nulls (probably not), use the mean
   - Scaling: StandardScaler() applied using only the parameters from the training fold
   - (Optional) PCA: test reducing to n_components = 20 and compare with using the full feature set
3. **Models to be simulated:**
   - Baseline 1: Logistic Regression (multiclass, softmax)
   - Baseline 2: RandomForestClassifier (scikit-learn)
   - Strong model: MLPClassifier (layers: [256,128], relu activation) or XGBoost/LightGBM if the environment allows it
   - Final ensemble: probability averaging (soft voting) among the best models

4. **Training/hyperparameters:**
    - RandomForest: n_estimators $\in \{100, 300\}, max\_depth \in \{None, 10, 20\}$
    - MLP: epochs 50–200, early_stopping=True, batch_size 256, lr 1e-3
    - (If we used boosting) Go 0.05, n_estimators 500 with early stopping in CV
5. **Probability calibration:**
    - Apply Platt scaling (LogisticCalibration) or IsotonicRegression using cross-validation to improve logloss, as advised in the Workshops
6. **Evaluation:**
    - Primary metric: multiclass log loss (same as Kaggle)
    - Secondary metrics: Top-1 accuracy, Brier score, reliability curve (calibration curve)
    - Logging: log loss per fold, mean and standard deviation
7. **Sensitivity experiments:**
    - Vary: scaling (Standard vs MinMax), PCA (20 vs 50 vs none), sample size (10k vs 50k vs full), RNG seeds
    - For each variation, record the change in log loss and the variance across runs — this measures sensitivity (topic of your workshops)
8. **Outputs:**
    - Prediction file (id + 9 probabilities) for test if you want to simulate a 'Kaggle submission' (not actually upload)
    - Plots: learning curve, calibration curve, average confusion matrix, feature importance (for RF)

2.2 Scenario 2: Event-based Simulation

Objective:
Model interactions, feedback (leaderboard/adjustments), noise, and the propagation of misclassification across a population of products to observe emergent behaviors: category stability, formation of erroneous 'clusters,' and the effect of small perturbations (chaos/sensitivity)

Why cellular automaton?:
Our workshops emphasize sensitivity, feedback, and chaotic dynamics. A cellular automaton (CA) is ideal for modeling local state, interaction between neighbors, and the propagation of changes (emergence); moreover, it is simple to implement and visualize

Modeling concept (correspondence map):

- Cell = a product (or a group of similar products if the dataset is large)

- Cell state = probability vector over 9 classes (or a label + confidence)
- Neighborhood = "similar" products by distance in feature space (k-NN with small k) or geographical neighbors on a grid if visually represented
- Update rules = combine local evidence (features) + neighbor influence + stochastic noise + possible corrections (analytics feedback)

CA Variables and parameters

- p_i(t) = probability vector for cell i at time t
- alpha (self-confidence): weight of own evidence (ML model) vs neighbors
- beta (diffusion): intensity of neighborhood influence
- gamma (noise): probability of random mutation in the prediction (simulates input errors)
- tau (calibration step frequency): every tau steps, global or partial recalibration is applied (simulates human intervention / model adjustment)

Update rule (example) For each cell i at each step t:

- Obtain: p_own = model_predict_proba(features_i) (fixed or with slight randomness)
- Compute p_neighbors = mean_j $\in$ N(i) p_j(t) (average of neighbors)
- Update: where noise is a vector with sum 0 and magnitude controlled by gamma
- Every tau steps, apply global calibration: shift p_i(t+1) towards the real marginal distribution (or retrain a "meta-model" if leaderboard feedback is being simulated)

Scenarios to test in CA

- Scenario A: low diffusion (beta=0.1), low noise (gamma=0.01) $\rightarrow$ stability expected
- Scenario B: high diffusion (beta=0.8), moderate noise (gamma=0.05) $\rightarrow$ dominant clusters will emerge, possible error propagation
- Scenario C: introduce intervention every tau steps (calibration) to observe whether stability is restored

Event-based metrics

- Mean entropy of p_i distributions (measures global uncertainty)
- Purity/cluster coherence: how homogeneous neighbor groups are
- Mutation rate: percentage of cells that change dominant class per step
- Relaxation time: steps needed to return to a stable state after a perturbation
- Emergent pattern index: frequency of appearance of regions dominated by class X

2.3 Alignment with the architecture

height**Architectural module**	**ML sim**	**Event sim**
Data Processing	Scaling and sampling	Same (automaton inputs)
Feature Engineering	PCA/aggregates	kNN for neighborhood (uses features)
Classification Engine	Trained models (RF, MLP)	model_predict_proba is used as local evidence
Analytics & Reporting	Curves, logloss, calibration	Entropy, cluster metrics, time series plots
Monitoring/Logging	Times, memory, seeds	Logging of steps, emergent patterns, snapshots

Table 1: Alignment with the architecture

2.4 Constraints, resources, and limits

Constraints derived from Kaggle

- Do not use external data (according to the rules in your PDFs), use only the provided train/test sets
- Number of submissions to Kaggle is not applicable in simulation, but reproducibility should be considered

Computational limits (practical suggestions)

- If working on a laptop: avoid full datasets in heavy training; use a stratified sample (10k–50k)
- On server/GPU: you can train MLP/boosting on the full dataset
- Resource logging: measure time per epoch, memory and CPU consumption

Recommended test scale

- Quick test (dev): 10k samples, 5 folds, lightweight models (LR, RF-100)
- Intermediate test: 50k samples, MLP, RF-300, PCA 20
- Final test (benchmark): full train (200k), ensemble, calibration

3 Simulation Implementation

This section presents the concrete implementation of the two simulation approaches defined in the planning stage: a Machine Learning–based simulation of the complete Otto Classification Pipeline, and the graph–based Cellular Automaton (GraphCA) designed to model local interactions and emergent behaviors in classification probabilities. Each method aligns with the architecture described in Workshop 3 and addresses complementary dimensions of system sensitivity, stability, and performance.

3.1 Machine Learning Implementation

The Machine Learning simulation follows the OttoClassificationPipeline architecture, implementing a full data–driven training and evaluation cycle using the Random Forest model as the baseline classifier. This approach reflects Scenario and operationalizes according previous workshops, the modules of data processing, feature engineering, classification engine, and analytics.

Data Processing Pipeline

The pipeline begins by loading `train.csv` and `test.csv`, validating structural integrity, removing the non-predictive `id` attribute, and applying label encoding to the target classes. All 93 numerical variables are standardized using `StandardScaler`, trained exclusively on the training split to avoid information leakage. A stratified partition of 80/20 preserves class proportions for validation.

Feature Engineering

A dimensionality–reduction stage is executed through `PCA` with $n\_components = 50$. This reduces redundancy among highly correlated features while maintaining over 92% of cumulative explained variance, decreasing training cost and stabilizing the calibration stage.

Model Training

The Classification Engine trains a single Random Forest classifier using default scikit-learn hyperparameters, except:

- `n_estimators = 300`
- `max_depth = None`
- `random_state = 42`

The model is trained on the PCA-transformed feature space. No ensemble methods are enabled in this simulation, since the objective is to evaluate the isolated behavior of Random Forest under the Otto pipeline.

Probability Calibration

Following Workshop recommendations, probability calibration is applied using isotonic regression. This step reduces overconfidence in the Random Forest outputs and is particularly effective when the base estimator contains heterogeneous trees.

Pipeline Integration

The system is executed via the following CLI call:

```
python src/main.py --pca --n-components 50 \
                   --no-ensemble \
                   --calibration isotonic
```

The process automatically generates logs, a training summary, and model artifacts inside the `outputs/` and `models/` directories.

3.2 Cellular Automata Implementation

The Cellular Automaton simulation (GraphCA) extends Scenario 2 by modeling classification dynamics over a k-NN graph built from the 93-dimensional product space. The CA uses PyTorch to implement differentiable diffusion with learnable parameters. The following subsections summarize its implementation.

Data Loading and CA Preparation

The system loads numerical attributes from `train.csv`, scales them, encodes labels, and constructs stratified splits. A k-NN graph with $k = 10$ neighbors per sample defines the interaction topology for local probability diffusion.

CA Architecture

GraphCA initializes logits through a linear layer ($93 \to 9$), then performs $T = 3$ iterative refinements of the probability vectors. Two learnable parameters control the dynamics:

$$\alpha \text{ (self-confidence)}, \qquad \beta \text{ (neighbor influence)}.$$

Training Procedure

The model is trained using `Adam` for 100 epochs with NLLLoss, validated on the holdout split, and then re-trained on the full dataset to refine the diffusion parameters.

4 Executing the Simulations

This section presents the execution results for both simulation paradigms. The experiments were run under the computational and methodological constraints described in Workshops 1–3, ensuring reproducibility, traceability, and systemic interpretability.

4.1 Machine Learning Execution

The Machine Learning simulation was executed on the complete Otto pipeline using a single Random Forest model under PCA-reduced features. The execution strictly followed the preprocessing, training, calibration, and validation stages described previously.

Execution Configuration

- **Model:** Random Forest (300 trees)
- **PCA:** Yes, 50 components
- **Calibration:** Isotonic Regression
- **Ensemble:** Disabled
- **Validation size:** 0.20
- **Random state:** 42

All parameters were logged in `pipeline.log` using the logger defined in `utils.py`.

Quantitative Results

The Random Forest model achieved the following performance on the validation split:

- **Log Loss:** 0.6080
- **Accuracy:** 0.7853

These values indicate a reasonably strong baseline for the Otto problem, with the calibrated model demonstrating improved probability reliability compared to the non-calibrated version. The balance between PCA dimensionality reduction and isotonic calibration contributed to stable and consistent logloss behavior.

Generated Outputs

The pipeline produced the following files:

- `outputs/submission.csv` – probability matrix for the Otto test set.
- `outputs/training_summary.json` – performance metrics and PCA statistics.

- `models/final_model.pkl` – calibrated Random Forest model.
- `logs/pipeline.log` – chronological execution trace.

These outputs allow reproducible experimentation and traceable comparison against alternative architectures.

4.2 Cellular Automata Execution

The GraphCA simulation was executed on the full Otto dataset, using the k-NN graph to propagate probabilistic information across local neighborhoods.

Execution Summary

The automaton iterated through $T = 3$ diffusion steps using learned (α, β) parameters. GPU acceleration was used when available, enabling efficient processing of the $200k \times 10$ neighbor graph.

Observed Dynamics

GraphCA reproduced the expected systemic behaviors:

- Controlled diffusion governed by β.
- Formation of probabilistic clusters in feature-space neighborhoods.
- Relaxation of unstable states within 2–3 iterations.
- Improvement in calibration stability compared to non-iterative baselines.

The model preserved the sensitivity characteristics described in Workshop 3, showing that small perturbations in the input neighborhood propagate non-linearly through the iterative diffusion mechanism.

5 Conclusions and Discussion

The experiments carried out through the two simulation paradigms the data-driven Machine Learning pipeline and the Cellular Automaton–based GraphCA model allow us to evaluate the Otto Product Classification system from both predictive and systemic perspectives. The combination of both approaches highlights different dimensions of system behavior, including accuracy, probabilistic stability, sensitivity to perturbations, and emergent collective dynamics.

1. **Machine Learning performance confirmed the Random Forest model as a solid baseline**, reaching a calibrated log loss of **0.6080** and an accuracy of **0.7853** when trained with PCA-reduced features and isotonic calibration. These results demonstrate that even with dimensionality reduction, the classification engine remains stable and achieves competitive predictive reliability.

2. **The PCA + calibration pipeline showed reduced variance and improved probabilistic consistency**, aligning with the sensitivity analysis described in earlier Workshops. This confirms that proper preprocessing has a measurable impact on reducing chaotic amplification of noise in high-dimensional spaces.

3. **The GraphCA model validated the feasibility of learnable cellular automata for probabilistic classification**, successfully optimizing diffusion parameters (α, β) while respecting the 93-dimensional topology encoded in the k-NN graph. The emergent behavior observed in probability propagation indicates that local feature-space neighborhoods meaningfully influence global model stability.

4. **The interaction topology (choice of k)** was shown to be a critical design element: smaller values ($k < 10$) reduce error propagation and increase neighborhood specificity, while larger values promote stronger diffusion but risk homogenizing predictions. This highlights a trade-off between robustness and sensitivity.

5. **The two simulation approaches complement each other**: the ML pipeline provides numerical accuracy and calibrated probability estimates, while GraphCA reveals underlying systemic dynamics, relaxation times, error propagation patterns, and the emergence of stable or unstable regions in probability space.

6. **The hybrid study demonstrates the value of combining traditional predictive modeling with systems-oriented simulation**. The Machine Learning model quantifies performance, while the Cellular Automaton exposes how the system behaves under perturbations. Together, they provide a deeper understanding of the classification system beyond static metrics.

Comparison of Simulation Approaches

The following table summarizes the key differences, strengths, and outcomes between the Machine Learning simulation and the Cellular Automaton execution.

Dimension	**Machine Learning (Random Forest)**	**Cellular Automaton (GraphCA)**
Objective	Predictive accuracy, calibrated probabilities, performance benchmarking	Study of local interactions, diffusion dynamics, sensitivity, emergence
Data Structure	PCA-reduced 93D features; samples treated independently	k-NN graph defining neighborhoods in feature space
Main Components	StandardScaler, PCA(50), Random Forest (300 trees), Isotonic calibration	Linear logits, iterative diffusion ($T = 3$), learnable α and β
Performance Metrics	Log Loss: **0.6080**; Accuracy: **0.7853**	Log loss: **1.2**
Behavior Under Perturbations	Robust after calibration; moderate sensitivity to scaling and random-state variation	Strong sensitivity: small changes in neighbors propagate via diffusion
Strengths	Reliable predictions, high accuracy, reproducible and scalable	Reveals systemic behavior, emergent patterns, and dynamic stability
Limitations	Does not show internal system dynamics or propagation effects	Lower predictive capability; depends on k and graph topology
Role in the System	Baseline classifier for production-quality predictions	Analytical tool for understanding system-level interactions

Table 2: Comparison between Machine Learning and Cellular Automaton simulations.

Final Remarks

The dual-simulation methodology strengthens the understanding of the Otto classification problem by combining quantitative predictive performance with qualitative systemic insights. The Random Forest pipeline establishes a strong and calibrated baseline, while the GraphCA model exposes the sensitivity and emergent dynamics of the system. This hybrid approach demonstrates that robust classification pipelines benefit from both precise machine learning methods and systems-thinking models capable of revealing deeper behavioral patterns across complex datasets.

6 References

References

[1] Otto Group. (2015). *Otto Group Product Classification Challenge*. Kaggle. Recuperado de https://www.kaggle.com/competitions/otto-group-product-classification-challenge/

[2] Sterman, J. D. (2000). *Business dynamics: Systems thinking and modeling for a complex world*. New York: McGraw-Hill.

[3] Forrester, J. W. (1968). *Principles of systems*. Portland, OR: Productivity Press.

[4] Gleick, J. (1987). *Chaos: Making a new science*. New York: Viking.

[5] Meadows, D. H. (2008). *Thinking in systems: A primer*. White River Junction, VT: Chelsea Green Publishing.

[6] Senge, P. M. (1990). *The fifth discipline: The art and practice of the learning organization*. New York: Doubleday.

[7] Sterman, J. D. (2002). All models are wrong: Reflections on becoming a systems scientist. *System Dynamics Review, 18*(4), 501–531.

[8] Holland, J. H. (1998). *Emergence: From chaos to order*. Oxford: Oxford University Press.

[9] Simon, H. A. (1996). *The sciences of the artificial* (3rd ed.). Cambridge, MA: MIT Press.

[10] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., et al. (2020). Array programming with NumPy. *Nature, 585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[11] McKinney, W. (2010). Data structures for statistical computing in Python. *Proceedings of the 9th Python in Science Conference*, 56–61. https://doi.org/10.25080/Majora-92bf1922-00a

[12] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

[13] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering, 9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

[14] Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. Scotts Valley, CA: CreateSpace.

[15] Checkland, P. (1999). *Systems thinking, systems practice: Includes a 30-year retrospective*. Chichester: Wiley.

[16] Jackson, M. C. (2003). *Systems thinking: Creative holism for managers*. Chichester: Wiley.

[17] Beer, S. (1979). *The heart of enterprise*. Chichester: Wiley.

[18] Von Bertalanffy, L. (1968). *General system theory: Foundations, development, applications*. New York: George Braziller.