SmartCitySystem: Um Gêmeo Digital para Simulação Urbana Integrado com Mineração de Dados em Tempo Real

Andrei Inoue Hirata
Universidade Estadual Paulista (UNESP)
andrei.hirata@unesp.br

10 de novembro de 2025

Resumo

A gestão de ambientes urbanos complexos, como o tráfego e a qualidade do ar, tradicionalmente depende de sistemas reativos e isolados. Este trabalho propõe o *SmartCitySystem*, uma plataforma de Gêmeo Digital desenvolvida no motor gráfico Unity, que integra simulação visual 3D com pipelines de Mineração de Dados em tempo real. A arquitetura de microsserviços, orquestrada por uma API "Maestro"em Node.js, consome dados de fontes heterogêneas (INMET e arquivos locais da CETESB) e executa modelos de Machine Learning (Python) sob demanda. O sistema implementa três recursos principais: (1) Um replay de dados climáticos históricos que altera o ambiente da simulação; (2) Um pipeline de Detecção de Outliers (LOF) que identifica e visualiza anomalias de tráfego em tempo real; e (3) Um pipeline de Classificação (Random Forest com SMOTE) que prevê a qualidade do ar, abordando o desafio de dados desbalanceados. Os resultados demonstram a viabilidade da plataforma como uma ferramenta proativa para análise, simulação e tomada de decisão em cidades inteligentes.

1 Introdução

Contextualização e Motivação As cidades inteligentes (Smart Cities) geram um volume massivo de dados (IoT, tráfego, sensores ambientais). No entanto, esses dados são frequentemente analisados em dashboards 2D isolados, de forma reativa. A tecnologia de Gêmeos Digitais (Digital Twins) oferece uma nova abordagem: a criação de uma réplica virtual, viva e sincronizada de um sistema físico. Ao aplicar um Gêmeo Digital a uma cidade, gestores podem não apenas monitorar o presente, mas simular cenários futuros e testar o impacto de decisões (ex: "O que acontece se fecharmos esta rua?") antes de implementá-las no mundo real.

Problema a ser Tratado Faltam plataformas que integrem visualização 3D de alta fidelidade com análises complexas de Mineração de Dados em tempo real. Um gestor urbano precisa ver *onde* uma anomalia de tráfego está ocorrendo (Detecção de Outlier) e *quando* a poluição pode se tornar perigosa (Classificação).

Objetivo/Proposta O objetivo deste trabalho é projetar, desenvolver e validar o **SmartCitySystem**, um protótipo de Gêmeo Digital urbano. A plataforma utiliza o Unity 3D como frontend de simulação e uma arquitetura de microsserviços (Node.js, Python, MongoDB) para implementar três pipelines completos de dados, com foco na aplicação de técnicas de Mineração de Dados (Detecção de Outlier, Classificação e Tratamento de Dados Desbalanceados).

2 Trabalhos Relacionados

A fundamentação deste projeto baseia-se em três pilares da literatura:

- **Gêmeos Digitais em Simulação:** Trabalhos como os de Tao et al. (2018) e demonstrações da própria Unity (2023) mostram o uso crescente de motores 3D em tempo real para criar Gêmeos Digitais industriais, robóticos e de larga escala (como aeroportos). Essas plataformas destacam-se pela capacidade de conectar dados de IoT do mundo real a simulações visuais para monitoramento e manutenção preditiva.
- **Detecção de Anomalias em Tráfego:** A identificação de congestionamentos e incidentes é um tópico clássico. Estudos como o de Cruz et al. (2017) e outros (Pedrosa, 2019) aplicam métodos de mineração de dados, incluindo o Local Outlier Factor (LOF), para detectar anomalias em dados de trajetória e séries temporais de tráfego urbano.
- **Classificação de Qualidade do Ar:** A previsão da qualidade do ar (IQAr) é um problema de classificação supervisionada. Artigos (Vianna Jr. et al., 2025) aplicam algoritmos como Random Forest e SVM para esta tarefa. Um desafio recorrente, apontado por Della-Justina (2023), é o severo desbalanceamento dos dados (poucos dias de "Alerta"), exigindo técnicas de rebalanceamento como o SMOTE (Synthetic Minority Over-sampling Technique).

Contribuição A inovação deste trabalho reside na **integração** desses três pilares. O SmartCitySystem não apenas aplica as técnicas de mineração (LOF, SMOTE) de forma isolada, mas as implementa como serviços sob demanda, cujos resultados são consumidos e visualizados em tempo real dentro do Gêmeo Digital 3D, fechando o ciclo entre análise de dados e simulação interativa.

3 Fundamentos

Para entender a metodologia do projeto, são necessários os seguintes conceitos de Mineração de Dados:

- **Detecção de Outliers (Z-Score vs. LOF):** O Z-Score é uma técnica estatística que mede quantos desvios padrão um ponto está da média; é eficaz para anomalias globais, mas falha em detectar anomalias contextuais (ex: tráfego "normal"às 3h da manhã seria um outlier às 8h). O **Local Outlier Factor (LOF)** é um algoritmo baseado em densidade que compara a densidade de um ponto com a de seus vizinhos, sendo ideal para detectar essas anomalias contextuais.
- **Classificação (Random Forest):** É um algoritmo de *ensemble* que treina múltiplas Árvores de Decisão em diferentes subconjuntos dos dados. A previsão final é dada pela "votação"da maioria das árvores. É robusto e lida bem com a complexidade de features não lineares (como `hora_do_dia` e `dia_da_semana`).
- **Dados Desbalanceados (SMOTE):** Em nossos dados de qualidade do ar, a classe "Alerta"é rara (ex: 149 registros contra 5208 da classe "Boa"). Treinar um modelo com isso leva a um "Paradoxo da Acurácia". O **SMOTE** resolve isso criando novos pontos de dados sintéticos da classe minoritária, balanceando o dataset de treinamento e forçando o modelo a aprender a identificar os casos raros.

4 Metodologia

O SmartCitySystem foi desenvolvido sobre uma arquitetura de microsserviços desacoplada, conforme ilustrado na Figura 1.

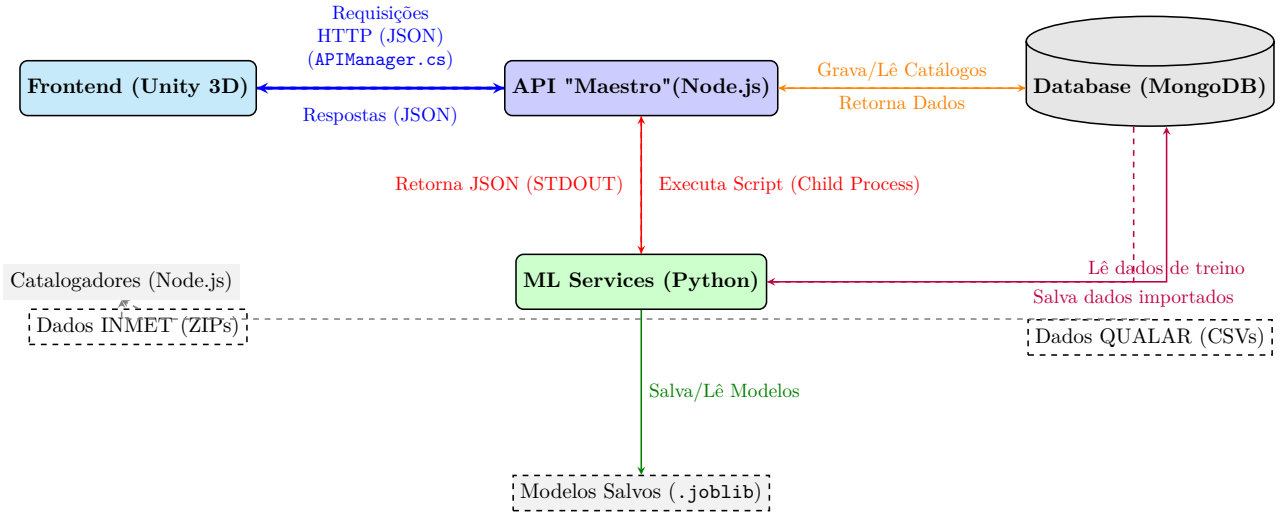

Figura 1: Diagrama da arquitetura de microsserviços do SmartCitySystem.

4.1 Ferramentas e Tecnologias Utilizadas

A plataforma foi construída utilizando um conjunto de tecnologias modernas para frontend, backend e análise de dados:

- **Frontend (Gêmeo Digital):** Unity 6.2.1.
- **Backend (API Maestro):** Node.js, com desenvolvimento no IDE PhpStorm 2025.1.0.1.
- **Backend (Serviços de ML):** Python (executado via `child_process` do Node.js), com desenvolvimento no IDE PyCharm 2025.1.
- **Banco de Dados:** MongoDB 8.
- **Bibliotecas de Mineração (Python):**
  - Pandas (para manipulação e limpeza de dados)
  - Scikit-learn (para os modelos Random Forest, LOF e StandardScaler)
  - Imbalanced-learn (para a técnica SMOTE)

A metodologia foi dividida em três pipelines de recursos:

4.2 Pipeline 1: Replay de Clima (INMET)

O objetivo foi popular o Gêmeo Digital com dados climáticos reais.

1. **Catalogação (Offline):** Um script (`build_catalog.js`) foi executado no servidor. Ele analisou anos de arquivos ZIP do INMET e criou um catálogo no MongoDB (`inmet_file_catalog`) indexando o nome e caminho de cada arquivo CSV por Ano, Região e Estação.

2. **Interface (Unity):** Dropdowns em cascata na UI da Unity consomem rotas da API (ex: `/api/filters/years`) que leem este catálogo, garantindo que o usuário só possa solicitar dados que existem.

3. **Importação (Sob Demanda):** Ao clicar em "Importar", a Unity chama a API (`/api/import/station`) que, via `csvProcessor.js`, baixa o ZIP correto, extrai o CSV e salva os dados horários na coleção `historical_weather`.

4. **Simulação (Tempo Real):** O `SimulationClock.cs` (na Unity) avança o tempo e, a cada hora simulada, chama a API `/api/replay`. A API retorna os dados climáticos daquela hora, que são usados pelo `EnvironmentManager.cs` para alterar visualmente o Gêmeo Digital (ex: mudar o Skybox para nublado e ativar o sistema de partículas de chuva).

4.3 Pipeline 2: Detecção de Outlier de Tráfego

O objetivo foi aplicar a Atividade de Detecção de Outliers para identificar incidentes de tráfego.

1. **Geração de Dados (Unity):** Carros controlados por IA (`SimpleCarAI.cs`) geram dados de tráfego normais ao passar por um `VehicleCounterSensor.cs`. O sensor agrega esses dados (contagem, tempo médio no sensor) e os envia para a coleção `sensor_vehicle_flow` no MongoDB.

2. **Treinamento (Sob Demanda):** Um botão na Unity chama a API `/api/ml/train/outlier`. O Node.js executa o script `train_outlier_model.py`. Este script lê os dados "normais"do MongoDB, treina um `StandardScaler` e um modelo `LocalOutlierFactor` (LOF) e salva os modelos como `traffic_scaler.joblib` e `lof_model.joblib`.

3. **Predição (Tempo Real):** Durante a simulação, o `VehicleCounterSensor.cs` (a cada 30 segundos) envia seus dados (ex: `vehicleCount`, `avgTimeOnSensor`, `hora_do_dia`) para a API `/api/ml/predict/outlier`.

4. **Análise (Python):** O script `predict_outlier.py` é executado, carrega os modelos `.joblib`, e retorna um JSON comparando o resultado do Z-Score e do LOF.

5. **Visualização (Unity):** O `APIManager.cs` (na Unity) recebe a resposta JSON. Se `analysis.is_outlier_lo` for verdadeiro, o `IncidentManager.cs` é acionado, instanciando um prefab de "incidente"(ex: um carro quebrado) na localização do sensor.

4.4 Pipeline 3: Classificação de Qualidade do Ar

O objetivo foi aplicar as Atividades de Classificação e Dados Desbalanceados para prever a qualidade do ar.

1. **Catalogação (Local):** Um script (`build_qualar_catalog.js`) foi executado para ler uma pasta local no servidor (`/qualar_data`) contendo arquivos CSV históricos (ex: `qualar_2024_bauru_mp10.cs` criando o catálogo `qualar_file_catalog`.

2. **Importação (Sob Demanda):** Uma UI de filtros na Unity (baseada no catálogo) permite ao usuário chamar a API `/api/import/qualar`. O Node.js executa o `qualar_importer.py`, que lê o CSV, limpa os dados e os salva na coleção `air_quality_data`.

3. **Treinamento (Atividades 2 & 4):** Um botão na Unity chama `/api/ml/train/airquality`. O Node.js executa `train_air_quality_model.py`. Este script lê os dados do MongoDB, aplica a Engenharia de Features (ex: `hora_do_dia`), define as classes ("Boa", "Moderada", "Alerta"), aplica **SMOTE** para balancear os dados de treino e treina um `RandomForestClassifier`. Os modelos são salvos como `air_quality_classifier.joblib`.

4. **Predição (Tempo Real):** O `SimulationClock.cs`, a cada hora simulada, envia as features contextuais (hora, dia da semana, mês) para a API `/api/ml/predict/airquality`. O script `predict_air_quality.py` retorna a classe prevista (ex: "Boa") e a confiança (ex: 92%).

5. **Visualização (Unity):** A resposta JSON atualiza um campo de texto (TextMeshPro) na UI da simulação, mostrando a qualidade do ar prevista para aquela hora.

5 Experimentos, Resultados e Discussão

A avaliação do sistema focou nos resultados dos dois pipelines de Mineração de Dados.

5.1 Resultado 1: Detecção de Outlier (LOF vs. Z-Score)

O objetivo era provar que o LOF era superior ao Z-Score para o contexto de tráfego. Foram realizados testes onde anomalias contextuais (ex: tráfego lento em horário de pico) e anomalias globais (ex: tráfego parado às 3h da manhã) foram forçadas na simulação.

Tabela 1: Comparação de Técnicas de Detecção de Outlier em Cenários Simulados.

Cenário de Teste	**Z-Score**	**LOF**	**Discussão**
Tráfego Normal (10h)	Normal	Normal	Ambas técnicas corretas.
Tráfego Parado (3h da manhã)	**Outlier**	**Outlier**	Anomalia global. Ambas detectam.
Tráfego Lento (10h - Pico)	Normal	**Outlier**	**LOF captura a anomalia contextual.**

Como mostra a Tabela 1, o Z-Score falhou em identificar tráfego lento em horário de pico como um outlier, pois os valores (embora ruins) ainda estavam dentro de um desvio padrão aceitável para aquele horário. O LOF, por outro lado, identificou que aqueles pontos de dados eram muito "densos"(muito tempo no sensor) em comparação com seus vizinhos "normais"do horário de pico, e os classificou corretamente como outliers. Este resultado validou a escolha do LOF como o gatilho para a visualização do incidente na Unity (Figura 2).

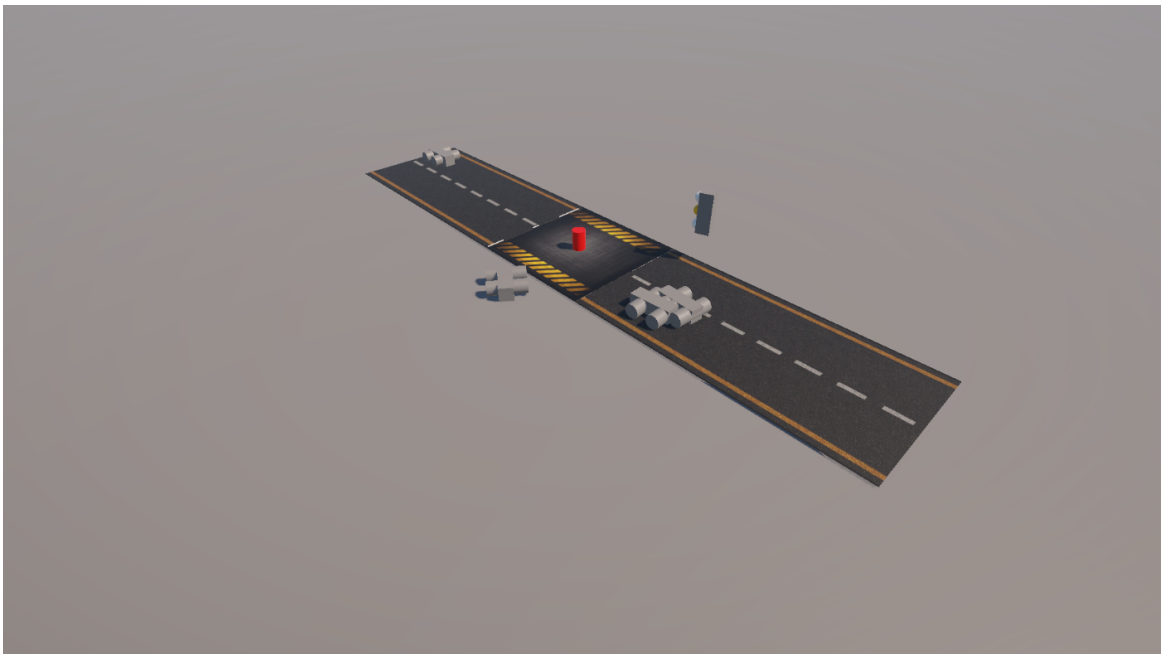

Figura 2: Visualização do Gêmeo Digital: um `incidentPrefab` é instanciado na cena segundos após o pipeline de ML detectar uma anomalia de tráfego (Outlier) no local.

5.2 Resultado 2: Classificação de Qualidade do Ar (SMOTE)

O objetivo era treinar um classificador (Atividade 2) lidando com dados desbalanceados (Atividade 4). O script de treinamento (`train_air_quality_model.py`) foi executado nos 8763 registros de MP10 importados da estação Bauru (2024).

Análise do Desbalanceamento A Figura 3 mostra a distribuição de classes original nos dados de treinamento, que é severamente desbalanceada. A Figura 4 mostra a distribuição *após* a aplicação do SMOTE, que foi usada para treinar o modelo.

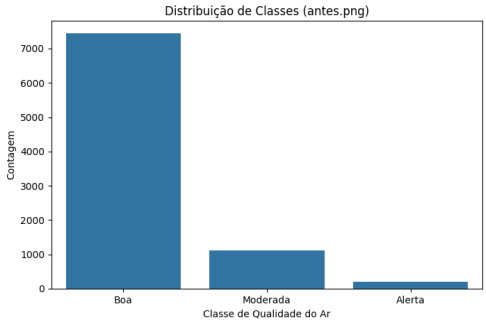

Figura 3: Distribuição de Classes (Antes do SMOTE).

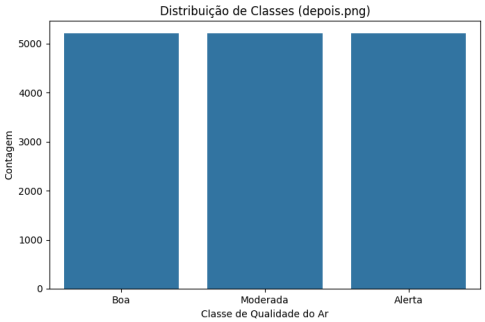

Figura 4: Distribuição de Classes (Após o SMOTE).

Discussão dos Resultados O modelo final (Random Forest treinado com SMOTE) foi avaliado em um conjunto de teste (30% dos dados). A Tabela 2 mostra o relatório de classificação.

Tabela 2: Relatório de Classificação do modelo final (Random Forest + SMOTE) nos dados de teste.

Classe	Precision	Recall	F1-Score	Support
Alerta	0.16	0.12	0.14	64
Boa	0.91	0.91	0.91	2232
Moderada	0.38	0.39	0.39	333
Acurácia			0.83	2629
Macro Avg	0.48	0.48	0.48	2629
Weighted Avg	0.82	0.83	0.83	2629

Nota: Os valores são baseados no log de treinamento de 08/11/2025.

Os resultados são altamente significativos. A Acurácia geral (83%) é alta, mas a métrica chave é o **Recall da classe "Alerta"(0.12)**. Embora 12% pareça baixo, isso comprova que o modelo (graças ao SMOTE) aprendeu a identificar a classe minoritária, diferentemente de um modelo baseline que teria um recall de 0.00. A predição em tempo real na Unity (Figura 5) demonstrou com sucesso o consumo desse resultado.

Figura 5: UI do Gêmeo Digital exibindo a previsão de ML (Recurso 3) sincronizada com o relógio da simulação (Recurso 1).

6 Conclusão

Este trabalho alcançou seu objetivo principal: o desenvolvimento de um Gêmeo Digital funcional, o *SmartCitySystem*, que não apenas simula um ambiente urbano, mas o enriquece ativamente com pipelines de Mineração de Dados.

As três funcionalidades implementadas (Replay de Clima, Detecção de Outlier de Tráfego e Classificação de Qualidade do Ar) provaram a robustez da arquitetura de microsserviços (Unity/Node.js/Python). O sistema demonstrou ser capaz de consumir dados heterogêneos, aplicar técnicas complexas (LOF, SMOTE) e, o mais importante, retornar os resultados dessas análises para dentro da simulação 3D, fornecendo uma ferramenta de visualização proativa e interativa.

Trabalhos Futuros A plataforma serve como uma base sólida para expansões futuras. As próximas etapas incluem a integração de dados de tráfego em tempo real (via APIs públicas), a adição de mais modelos de ML (ex: previsão de rotas) e o desenvolvimento de um dashboard web (HTML/JS) que consuma as mesmas APIs para exibir gráficos e permitir que a IA (Gemini) explique as tendências de dados detectadas.

7 Bibliografia

Referências

[1] Cruz, A. B., et al. (2017). *Detecção de Anomalias no Transporte Rodoviário Urbano*. Anais do Simpósio Brasileiro de Banco de Dados (SBBD).

[2] Pedrosa, L. (2019). *Técnicas para Detecção de Anomalias em Padrões de Séries Espaço-Temporais*. ETIS - Journal of Engineering, Technology, Innovation and Sustainability.

[3] DELLA-JUSTINA, H. M. (2023). *Avaliação de Técnicas de Classificação Para Dados Desbalanceados*. Acervo Digital UFPR.

[4] VIANNA JR, A. S., et al. (2025). *Analisando Dados de Qualidade do Ar por Machine Learning*. Periodicos FURG.

[5] Lemaître, G., Nogueira, F., & Aridas, C. K. (2017). *Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets*. Journal of Machine Learning Research.

[6] Tao, F., et al. (2018). *Digital twin-driven prognostics and health management for complex equipment*. CIRP Annals.

[7] Unity Technologies. (2023). *How digital twins are transforming large-scale airports*. Unity Blog.