Espinha Dorsal do Projeto de IC — IA com Dados Tabulares

Material de apoio para aluno de Iniciação Científica

28 de agosto de 2025

Sumário

Objetivo do documento

Este material serve como guia prático para um aluno de Iniciação Científica (Engenharia de Software) executar um projeto de IA aplicada a **dados tabulares**, do zero até o treinamento e avaliação de uma **rede neural (MLP)**, documentando cada decisão, acerto e erro para, ao final, compor uma apostila "Como treinar IA com dados tabulares". O documento está dividido em três grandes partes: (1) Pré-processamento; (2) Configuração da Rede Neural; (3) Treinamento, Validação e Testes. Inclui ainda checklists, modelos e apêndices.

1 Visão Geral do Projeto

1.1 Escopo e critérios de sucesso

- **Tarefa**: classificação *ou* regressão supervisionada.
- **Dados**: dataset público e aberto, com licença compatível.
- **Sucesso**: (i) pipeline reprodutível, sem vazamento; (ii) rede neural comparada a baselines clássicos; (iii) documentação completa (diário, README, model card); (iv) métricas consistentes em validação e teste.

1.2 Boas práticas

- Reprodutibilidade (seeds fixas, versões das bibliotecas, config salva).
- Prevenção de vazamento de dados em todo o pipeline.
- Comparar com baselines não-NN (p.ex. Regressão Logística/Linear, Árvores, Random Forest, XGBoost leve).
- Verificar ética/privacidade/licença do dataset.
- Explicabilidade básica e análise de erros.

1.3 Ferramental sugerido (não obrigatório)

Python 3.11+, `pandas`, `numpy`, `scikit-learn`, `PyTorch` *ou* `Keras/TensorFlow`, `optuna` (ou similar) para busca de hiperparâmetros, `matplotlib`/`plotly`. Fixe seeds.

1.4 Cronograma sugerido (exemplo)

- Semanas 1–2: escolha e entendimento do dataset; splits; limpeza; codificação; escala.
- Semanas 3–4: engenharia/seleção de atributos; baselines; definição da arquitetura MLP.
- Semanas 5–6: treinamento com early stopping; validação; comparação com baselines.
- Semana 7: testes finais; análise de erros; documentação (model card/README).

1.5 Diagrama de fluxo (alto nível)

```
Dados -> Entendimento -> Splits -> Limpeza/Imputação -> Codificação/Escala
     -> Desbalanceamento -> Engenharia de Atributos -> Baselines -> MLP
     -> Treinamento (early stopping, checkpoints) -> Validação -> Teste
     -> Comparação, Interpretação, Documentação (README, model card)
```

2 PARTE 1 — Pré-processamento de Dados

2.1 Seleção e entendimento do dataset

- Fontes: UCI, OpenML, Kaggle (verifique **licença** e implicações éticas/PII).
- Defina a tarefa (classificação ou regressão) e a variável-alvo.
- Entendimento inicial: n.º de linhas/colunas; % faltantes; tipos (num/cat); desbalanceamento; presença de datas/IDs; multicolinearidade preliminar; outliers aparentes.

2.2 Partições e prevenção de vazamento

- Splits treino/validação/teste (ex.: 60/20/20) com **estratificação** quando aplicável.
- Quando usar k-fold (ex.: 5-fold) e como evitar vazamento entre folds.
- **Critério de saída**: splits fixos (seed registrada), distribuição semelhante entre partições.

2.3 Limpeza e imputação

- Estratégias por tipo de feature: remoção vs imputação.
- Imputação simples (média/mediana/moda) vs avançada (KNN/iterativa).
- Sinal de alerta: jamais imputar usando informações de validação/teste.

2.4 Codificação de categóricas e escalonamento

- One-hot, target encoding ou *embeddings* (motivação para NN).
- Escalonamento (Standard/MinMax/Robust) com estatísticas **do treino**.
- **Critério de saída**: pipeline `fit` no treino e `transform` em val/test sem vazamento.

2.5 Tratamento de desbalanceamento (classificação)

- Pesos na função de perda vs reamostragem (SMOTE/undersampling).
- Métricas adequadas: AUC-ROC, PR-AUC, F1 macro.
- Erro comum: avaliar só por acurácia.

2.6 Engenharia de atributos

- Interações, binning, variáveis de tempo, transformações (ex.: `log`).
- Seleção de features (mutual information, importância baseada em árvore, VIF).
- **Critério de saída**: relatório do que entrou/saiu e justificativas.

2.7 Baselines obrigatórios

- Pelo menos dois: Regressão Logística/Linear, Árvore/Random Forest, XGBoost leve.
- Salvar métricas e tempos de execução.
- **Critério de saída**: baseline estabelecido para comparação justa com a NN.

Checklist de saída — PARTE 1

☐ Licença/ética verificados
☐ Splits definidos com seed
☐ Pipeline sem vazamento
☐ Categóricas e escala tratadas
☐ Desbalanceamento endereçado (se aplicável)
☐ Baselines rodados e salvos

Erros comuns — PARTE 1

- Usar estatísticas do conjunto completo antes do split.
- Comparar métricas de treino com as de teste (otimismo).
- Ignorar baseline e concluir que a NN é melhor sem evidência.

3 PARTE 2 — Configuração da Rede Neural (MLP)

3.1 Arquitetura

- **Entrada**: numéricas (após escala) + categóricas via *embeddings*
  dimensão sugerida: $\min(50, \lfloor 1.6 \cdot \sqrt{n_{\text{categorias}}} \rceil)$ por coluna categórica.
- **Corpo (MLP)**: 2–4 camadas densas; tamanho decrescente (ex.: 256→128→64).
- **Normalização**: BatchNorm entre camadas.
- **Regularização**: Dropout (0.1–0.5), L2 (weight decay).
- **Ativações**: ReLU/GELU; saída: sigmoid/softmax (classificação); linear (regressão).

3.2 Função de perda e métricas

- Binária: BCEWithLogits; Multiclasse: Cross-Entropy.
- Regressão: MSE/MAE (justifique a escolha).
- Métricas de monitoramento: além da loss, usar F1/AUC (class.) ou MAE/MAPE (regr.).

3.3 Otimizador e scheduler

- Adam/AdamW; LR inicial típico: 10^{-3} (ajuste conforme validação).
- Schedulers: CosineAnnealing, StepLR, ReduceLROnPlateau (quando usar cada um).
- **Critério de saída**: plano de LR definido e registrado.

3.4 Hiperparâmetros a explorar

- LR, batch size, nº camadas/unidades, dropout, weight decay, dimensões de embeddings, pesos de classe/reamostragem.
- Estratégia: grid pequeno *ou* otimização bayesiana (`optuna`) com orçamento (n trials) e seed fixa.

3.5 Reprodutibilidade e organização

- Fixar seeds (Python/NumPy/framework).
- Salvar: config YAML/JSON, versões das libs, hash do commit, hardware.
- Estruturar pastas: `data/`, `src/`, `models/`, `runs/`, `reports/`.

Checklist de saída — PARTE 2

☐ Arquitetura documentada
☐ Perda/métricas definidas
☐ Otimizador/scheduler definidos
☐ Espaço de hiperparâmetros descrito
☐ Seeds e versões salvas

Erros comuns — PARTE 2

- Oversizing (rede grande) com poucos dados.
- Ignorar *embeddings* para muitas categóricas.
- Não registrar a configuração exata do melhor modelo.

4 PARTE 3 — Treinamento, Validação e Testes

4.1 Protocolo de treino

- Early stopping monitorando métrica de validação (defina paciência).
- Checkpoint do melhor modelo (pela métrica principal).
- Callbacks de log (perdas, métricas por época, LR).

4.2 Avaliação correta

- Se usar cross-validation: reporte média e desvio-padrão.
- Conjunto de teste: **apenas uma vez** ao final.
- Intervalos de confiança (bootstrap opcional).

4.3 Comparação com baselines

- Tabela comparativa (métricas e tempo/recursos).
- Teste estatístico simples (ex.: McNemar para binária), se fizer sentido.
- **Critério de saída**: NN supera baseline principal *ou* justificativa clara (ex.: interpretabilidade vs performance).

4.4 Interpretação e diagnóstico

- Importância de features (ex.: permutation importance).
- Curvas ROC/PR, matriz de confusão, análise de erros por subgrupos.
- Alertas: overfitting (gap treino-val), drift entre splits, dependência de poucas features.

4.5 Documentação final e empacotamento

- Model card curto (dados, tarefa, métricas, limitações, ética/bias, uso recomendado).
- README com instruções de reprodução (instalação, download/prep dos dados, comando de treino, como avaliar).
- Exportar pesos e arquivos de ambiente (`requirements.txt`/`environment.yml`).

Checklist de saída — PARTE 3

☐ Early stopping/checkpoints
☐ Avaliação justa (val/test)
☐ Tabelas/figuras salvas
☐ Comparação com baseline
☐ Model card + README

Erros comuns — PARTE 3

- Ajustar decisões olhando o teste (contaminação).
- Reportar apenas a melhor época/métrica sem contexto.
- Não analisar *por que* o modelo erra.

Regras importantes (sempre)

- Não inventar dados nem resultados. Em dúvidas, registrar a hipótese e decidir de forma explícita.
- Destacar sempre *Sinais de alerta* e *Critérios de saída* em cada parte.
- Incluir checklists e perguntas de reflexão em todos os blocos.
- Usar exemplos concretos quando útil, mantendo o material adaptável a diferentes frameworks.
- Foco em **aprendizado documentado**: erros são oportunidades para a apostila.

A Template de Diário de Bordo / Experimentos

Use a tabela abaixo (copiável em Markdown para o repositório) ou mantenha como planilha. Cada linha = um experimento.

Data	Objetivo	Mudanças (HP/arquitetura/dados)	Seed	Tempo	Métrica (val)	Resultado

B Modelo de README do Projeto

- **Visão geral**: problema, dados, tarefa (classificação/regressão), principais métricas.
- **Requisitos**: versões de Python/bibliotecas; como criar ambiente virtual.
- **Instalação**: comandos para instalar dependências.
- **Como obter dados**: links, passos para download e preparação.
- **Como treinar**: comandos (scripts) com parâmetros padrão.
- **Como avaliar**: geração de métricas, curvas e tabelas.
- **Estrutura de pastas**: `data/`, `src/`, `models/`, `runs/`, `reports/`.
- **Resultados**: tabelas/figuras principais; comparação com baselines.
- **Limitações e Ética**: viés, uso recomendado, riscos conhecidos.
- **Licença**: compatível com a do dataset.

C Checklist de Auditoria de Qualidade

- Dados: licença verificada; PII tratada; splits corretos; sem vazamento.
- Código: seeds fixas; versões registradas; scripts rodam fim-a-fim.
- Métricas: adequadas à tarefa; comparação justa com baselines; CIs/variância.
- Resultados: reprodutíveis; análise de erros; interpretações coerentes.
- Documentação: diário de bordo; README; model card; artefatos versionados.

D Roteiro de Relato de Erros (Post-Mortem)

- O que tentamos? Qual hipótese? O que esperávamos?
- O que ocorreu? Evidências e logs.
- Causa raiz (técnica/processo/dados)? Como confirmar?
- Como prevenir? Ações corretivas (curto/médio prazo).
- O que aprendemos? Impacto no próximo ciclo.