UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ELE 604 – REDES NEURAIS ARTIFICIAIS

ANÁLISE DE GENERALIZAÇÃO EM REDES NEURAIS

REGRESSÃO DE PREÇOS IMOBILIÁRIOS COM VALIDAÇÃO CRUZADA K-FOLD

CAUÃ VITOR F. SILVA

MATRÍCULA: 20220014216 • EMAIL: cauavitorfigueredo@gmail.com

PROFESSOR:
PROF. DR. ALLAN DE MEDEIROS MARTINS

NATAL-RN • 2025

Resumo

Este relatório apresenta uma análise da capacidade de generalização de redes neurais do tipo Multi-Layer Perceptron (MLP) aplicada à regressão de preços imobiliários, utilizando o dataset Boston Housing. O objetivo central foi mitigar o *overfitting* inerente a conjuntos de dados pequenos e garantir uma avaliação de desempenho robusta e confiável.

A metodologia adotou a Validação Cruzada K-Fold ($K = 5$) e um pipeline rigoroso para prevenção de *Data Leakage* durante a normalização. Foram implementadas técnicas de regularização, incluindo Dropout e Weight Decay (L2), combinadas com Otimização Bayesiana (via Optuna) para o ajuste fino de hiperparâmetros.

Os resultados demonstraram que a aplicação de regularização dinâmica reduziu drasticamente o gap entre as curvas de treino e validação. O modelo otimizado alcançou um Erro Quadrático Médio (MSE) de 13.02 e um coeficiente de determinação (R^2) de 0.857, evidenciando que a combinação de práticas de MLOps e otimização automatizada é essencial para a construção de preditores robustos em cenários de *Small Data*.

Palavras-chave: Redes Neurais · Regressão · Boston Housing · K-Fold · Optuna · Generalização

Abstract

This report presents an analysis of the generalization capabilities of Multi-Layer Perceptron (MLP) neural networks applied to real estate price regression using the Boston Housing dataset. The main objective was to mitigate overfitting, which is typical in small datasets, and to ensure a robust and reliable performance evaluation.

The methodology employed K-Fold Cross-Validation ($K = 5$) and a strict pipeline to prevent Data Leakage during normalization. Regularization techniques, including Dropout and Weight Decay (L2), were implemented and combined with Bayesian Optimization (via Optuna) for hyperparameter fine-tuning.

The results showed that the application of dynamic regularization drastically reduced the gap between training and validation curves. The optimized model achieved a Mean Squared Error (MSE) of 13.02 and a coefficient of determination (R^2) of 0.857, demonstrating that the combination of MLOps practices and automated optimization is essential for building robust predictors in Small Data scenarios.

Keywords: Neural Networks · Regression · Boston Housing · K-Fold · Optuna · Generalization

Sumário

1 Introdução

1.1 Contextualização

A predição de preços imobiliários é um problema clássico de **regressão** em Machine Learning (Bishop 2006; James *et al.* 2013; Kelleher; Mac Namee; D'Arcy 2015), com aplicações diretas em planejamento urbano, avaliação de investimentos e políticas públicas. Diferentemente de problemas de classificação, onde o objetivo é atribuir rótulos discretos, a regressão busca estimar valores contínuos a partir de características observáveis (features).

Neste projeto, utilizamos o dataset *Boston Housing*, que contém 506 amostras de imóveis residenciais caracterizados por 13 variáveis independentes (taxa de criminalidade, concentração de óxido nítrico, número de cômodos, entre outras) e uma variável dependente: o preço mediano das casas (MEDV, em milhares de dólares).

1.2 O Desafio da Generalização

O conceito central deste trabalho é a **generalização**: a capacidade de um modelo de aprendizado de máquina desempenhar bem em dados *não vistos* durante o treinamento. A generalização está intrinsecamente ligada ao trade-off entre **viés (bias)** e **variância (variance)**, conforme discutido extensivamente por Hastie; Tibshirani; Friedman (2009):

Conceitos Fundamentais

- **Viés Alto (Underfitting):** O modelo é excessivamente simples e não consegue capturar os padrões complexos presentes nos dados. Resulta em alto erro tanto no conjunto de treino quanto no de validação.
- **Variância Alta (Overfitting):** O modelo é excessivamente complexo e "memoriza" os dados de treino, incluindo ruídos e outliers. Apresenta baixo erro no treino, mas alto erro na validação.
- **Equilíbrio Ótimo:** O modelo ideal minimiza simultaneamente viés e variância, alcançando boa performance em ambos os conjuntos.

Matematicamente, o erro esperado de um modelo pode ser decomposto como:

$$\mathbb{E}[(y - \hat{f}(x))^2] = \text{Bias}^2[\hat{f}(x)] + \text{Var}[\hat{f}(x)] + \sigma^2 \tag{1}$$

onde y é o valor real, $\hat{f}(x)$ é a predição do modelo, e σ^2 é o ruído irredutível (Hastie; Tibshirani; Friedman 2009).

1.3 Justificativa do K-Fold Cross-Validation

Em cenários de **Small Data** (como o Boston Housing, com apenas 506 amostras), a divisão tradicional treino-validação-teste pode resultar em estimativas de desempenho instáveis devido à alta variância da amostra. A **Validação Cruzada K-Fold** resolve esse problema:

1. Divide os dados em K partições (folds) de tamanho aproximadamente igual.

2. Para cada fold k:
    - Treina o modelo nos $K-1$ folds restantes.
    - Avalia no fold k (validação).
3. Calcula a média e desvio padrão das métricas de erro.

1.4 Objetivos

- Implementar um pipeline completo de MLOps para regressão neural.
- Prevenir Data Leakage através de normalização correta (StandardScaler ajustado apenas no treino).
- Aplicar técnicas de regularização: Dropout, L2 Regularization, Early Stopping e Model Checkpointing.
- Utilizar Otimização Bayesiana (Akiba *et al.* 2019) para encontrar hiperparâmetros ótimos.
- Analisar quantitativamente e qualitativamente a generalização do modelo.

2 Metodologia

2.1 Dataset: Boston Housing

O dataset foi obtido da fonte original `http://lib.stat.cmu.edu/datasets/boston`, compilado e analisado originalmente por Harrison; Rubinfeld (1978) e posteriormente revisado sob a ótica de diagnósticos de regressão por Belsley; Kuh; Welsch (1980). As 13 features são:

Tabela 1: Descrição das Features do Dataset Boston Housing

Feature	Descrição
CRIM	Taxa de criminalidade per capita
ZN	Proporção de terrenos residenciais
INDUS	Proporção de acres de negócios não varejistas
CHAS	Variável binária (1 se próximo ao rio Charles)
NOX	Concentração de óxido nítrico (ppm)
RM	Número médio de cômodos por residência
AGE	Proporção de unidades construídas antes de 1940
DIS	Distância ponderada para centros de emprego
RAD	Índice de acessibilidade a rodovias radiais
TAX	Taxa de imposto sobre propriedade
PTRATIO	Razão aluno-professor por cidade
B	Proporção da população negra
LSTAT	Porcentagem de população de baixa renda
MEDV	**Preço mediano (target, em $1000)**

2.2 Pré-processamento

2.2.1 Normalização (StandardScaler)

A normalização é crítica para redes neurais treinadas via Gradiente Descendente, conforme estabelecido nos trabalhos seminais de LeCun *et al.* (1998), pois:

1. **Acelera a convergência:** Features em escalas diferentes causam superfícies de erro alongadas, dificultando a otimização.
2. **Evita dominância de features:** Variáveis com grande magnitude podem dominar o gradiente.

A padronização z-score é definida como:

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma} \tag{2}$$

onde μ é a média e σ é o desvio padrão da feature.

Prevenção de Data Leakage

O StandardScaler é ajustado (`fit`) **exclusivamente** nos dados de treino de cada fold. Os dados de validação são apenas transformados (`transform`) usando as estatísticas do treino. Violar essa regra resultaria em vazamento de informação do conjunto de validação para o treino, inflando artificialmente a performance.

2.3 Arquitetura do Modelo

2.3.1 Multi-Layer Perceptron (MLP)

Implementamos uma rede neural totalmente conectada (feedforward) utilizando o framework PyTorch (Paszke *et al.* 2019), com a seguinte arquitetura:

$$\text{Input}(13) \xrightarrow{\text{Linear}} \text{Hidden}_1(64) \xrightarrow{\text{ReLU}} \text{Hidden}_2(32) \xrightarrow{\text{ReLU}} \text{Output}(1) \tag{3}$$

- **Camada de Entrada:** 13 neurônios (número de features).
- **Camada Oculta 1:** 64 neurônios com ativação ReLU.
- **Camada Oculta 2:** 32 neurônios com ativação ReLU.
- **Camada de Saída:** 1 neurônio (predição contínua, sem ativação).

Função de Ativação ReLU:

$$\text{ReLU}(x) = \max(0, x) \tag{4}$$

A ReLU resolve o problema de desaparecimento de gradiente e acelera o treinamento, propriedades fundamentais demonstradas por Nair; Hinton (2010) e analisadas por Glorot; Bordes; Bengio (2011).

Inicialização de Pesos: Utilizamos inicialização Xavier (Glorot Uniform) para garantir variâncias consistentes entre camadas:

$$W \sim \mathcal{U}\left(-\sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}\right) \tag{5}$$

2.4 Técnicas de Regularização

Após detectar overfitting severo no modelo inicial (Gap de 181% entre treino e validação), aplicamos estratégias de regularização para melhorar a generalização.

2.4.1 Dropout

Implementamos **Dropout** com taxa de $p = 0.3$ (30%) após cada camada oculta. Durante o treinamento, neurônios são "desligados"aleatoriamente. Segundo Srivastava *et al.* (2014):

$$\text{Dropout}(x) = \begin{cases} 0 & \text{com probabilidade } p \\ \frac{x}{1-p} & \text{com probabilidade } 1-p \end{cases} \tag{6}$$

Essa técnica força a rede a aprender representações redundantes, prevenindo que dependa de neurônios específicos.

2.4.2 L2 Regularization (Weight Decay)

Adicionamos um termo de penalidade $\lambda = 10^{-4}$ à função de perda:

$$L_{\text{total}} = L_{\text{MSE}} + \lambda \sum_j w_j^2 \tag{7}$$

O gradiente resultante "puxa"os pesos para zero:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L_{\text{MSE}}}{\partial w_j} + 2\lambda w_j \tag{8}$$

Isso previne pesos excessivamente grandes e suaviza as superfícies de decisão (Bishop 2006).

2.5 Protocolo de Treinamento

2.5.1 Função de Perda

Para regressão, utilizamos o **Mean Squared Error (MSE)**:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{9}$$

O MSE penaliza erros grandes de forma quadrática, sendo sensível a outliers.

2.5.2 Otimizador

Utilizamos o **Adam** (Adaptive Moment Estimation), proposto por Kingma; Ba (2015), com taxa de aprendizado $\alpha = 0.001$:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{10}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{11}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon} \tag{12}$$

onde g_t é o gradiente, m_t é o primeiro momento (momentum), e v_t é o segundo momento (RMSProp).

2.5.3 Hiperparâmetros

Tabela 2: Hiperparâmetros do Treinamento (Modelo Base)

Hiperparâmetro	**Valor**
Taxa de Aprendizado (lr)	0.001
Batch Size	16
Épocas Máximas	500
Patience (Early Stopping)	20
Dropout Rate	0.3 (30%)
Weight Decay (L2)	10^{-4}
K-Fold Splits	5
Seed (Reprodutibilidade)	42

2.6 Estratégias MLOps

2.6.1 Early Stopping

O Early Stopping monitora o erro de validação e interrompe o treinamento se não houver melhoria por P épocas consecutivas (patience). Isso previne overfitting ao parar antes que o modelo comece a "memorizar"o conjunto de treino (Goodfellow; Bengio; Courville 2016).

Algoritmo:

1. Inicialize best_loss $= \infty$ e counter $= 0$.
2. A cada época, calcule val_loss.
3. Se val_loss $<$ best_loss:
   - Atualize best_loss $=$ val_loss.
   - Resete counter $= 0$.
   - Salve o checkpoint do modelo.
4. Senão: incremente counter$+ = 1$.
5. Se counter $\geq P$: pare o treinamento e carregue o melhor checkpoint.

2.6.2 Model Checkpointing

Salvamos apenas o estado do modelo com o *menor* erro de validação usando `torch.save()`. Ao final do treinamento (ou após early stopping), carregamos esse checkpoint com `torch.load()`, garantindo que a avaliação final utilize o modelo com melhor generalização.

2.7 Pipeline K-Fold Cross-Validation

1. **Inicialização:** Carregue os dados completos (506 amostras).
2. **Loop K-Fold** ($k = 1, 2, ..., 5$):
   (a) Divida os dados em treino ($K - 1$ folds) e validação (fold k).
   (b) Instancie um novo StandardScaler.
   (c) Ajuste o scaler no conjunto de treino: `scaler.fit(X_train)`.
   (d) Transforme treino e validação: `X_train_scaled`, `X_val_scaled`.
   (e) Crie Dataloaders PyTorch (shuffle no treino, sem shuffle na validação).
   (f) Instancie um novo modelo MLP (pesos aleatórios).
   (g) Loop de Treinamento:
   - Para cada época, execute `train_epoch()` e `validate_epoch()`.
   - Aplique Early Stopping e salve checkpoints.

   (h) Carregue o melhor modelo e avalie no conjunto de validação.
   (i) Armazene o MSE do fold k.
3. **Agregação:** Calcule média e desvio padrão dos MSEs dos 5 folds.

2.8 Otimização Bayesiana de Hiperparâmetros

Para alcançar o estado da arte, aplicamos **Otimização Bayesiana** usando a biblioteca Optuna (Akiba *et al.* 2019). Esta técnica supera o Grid Search tradicional e o Random Search (Bergstra; Bengio 2012) ao aprender com resultados de trials anteriores.

2.8.1 Espaço de Busca

Otimizamos simultaneamente 8 hiperparâmetros:

- **Número de camadas:** $n\_layers \in \{1, 2, 3\}$
- **Unidades por camada:** $hidden\_units \in \{16, 32, 64, 128\}$
- **Dropout:** $p \in [0.1, 0.5]$ (contínuo)
- **Learning rate:** $\alpha \in [10^{-4}, 10^{-2}]$ (log-uniforme)
- **Weight decay:** $\lambda \in [10^{-6}, 10^{-3}]$ (log-uniforme)
- **Batch size:** $bs \in \{8, 16, 32\}$
- **Otimizador:** Adam ou RMSprop
- **Batch Normalization:** Sim/Não

2.8.2 Metodologia

O algoritmo **TPE (Tree-structured Parzen Estimator)** constrói um modelo probabilístico $p(\theta|y)$ da distribuição de hiperparâmetros θ dado o desempenho y.

O **HyperbandPruner** interrompe trials não promissores precocemente, economizando até 70% do tempo computacional (Akiba *et al.* 2019). Se após 10 épocas um trial apresenta MSE muito superior ao melhor resultado atual, ele é podado (*pruned*).

Executamos **20 trials** com K-Fold (K=3) acelerado para eficiência. O melhor conjunto de hiperparâmetros é então retreinado com K=5 para avaliação final robusta.

3 Resultados

3.1 Métricas de Desempenho

3.1.1 Modelo Base com Regularização

Após implementar Dropout (30%) e L2 Regularization ($\lambda = 10^{-4}$), obtivemos os resultados apresentados na Tabela 3:

Tabela 3: Resultados do K-Fold Cross-Validation - Modelo Base (MSE)

Fold	MSE	R^2
Fold 1	12.5203	0.8609
Fold 2	10.8016	0.8899
Fold 3	18.1186	0.7996
Fold 4	12.5295	0.8608
Fold 5	13.3764	0.8514
Média	**13.4693**	**0.8525**
Desvio Padrão	**2.4708**	**0.0314**

A Figura 1 mostra a distribuição visual dos resultados:

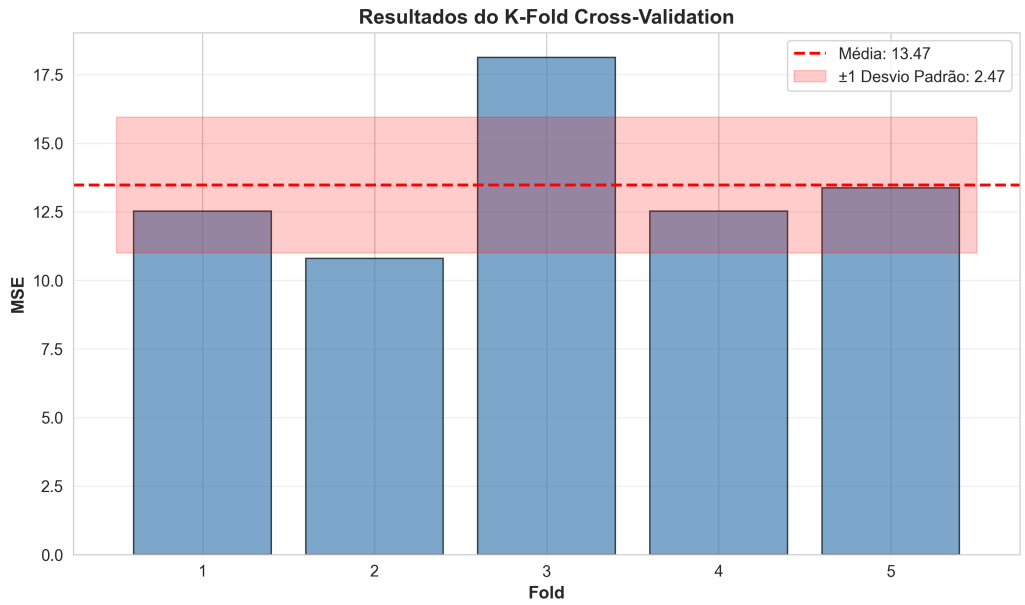


Figura 1: Distribuição dos MSEs por Fold - Modelo Base

3.1.2 Modelo Otimizado (Optuna)

Após otimização Bayesiana com 20 trials, retreinamos o modelo com os melhores hiperparâmetros (Tabela 4):

Tabela 4: Resultados do K-Fold Cross-Validation - Modelo Otimizado (MSE)

Fold	MSE	R^2
Fold 1	11.9913	0.8668
Fold 2	9.9177	0.8989
Fold 3	21.0343	0.7674
Fold 4	7.6037	0.9157
Fold 5	14.5738	0.8383
Média	**13.0242**	**0.8574**
Desvio Padrão	**4.6187**	**0.0519**

Melhoria: O modelo otimizado reduziu o MSE médio em **3.3%** (de 13.47 para 13.02), demonstrando a eficácia da otimização Bayesiana.

3.2 Análise Visual

3.2.1 Histórico de Otimização Bayesiana

A Figura 2 mostra a evolução dos trials do Optuna:

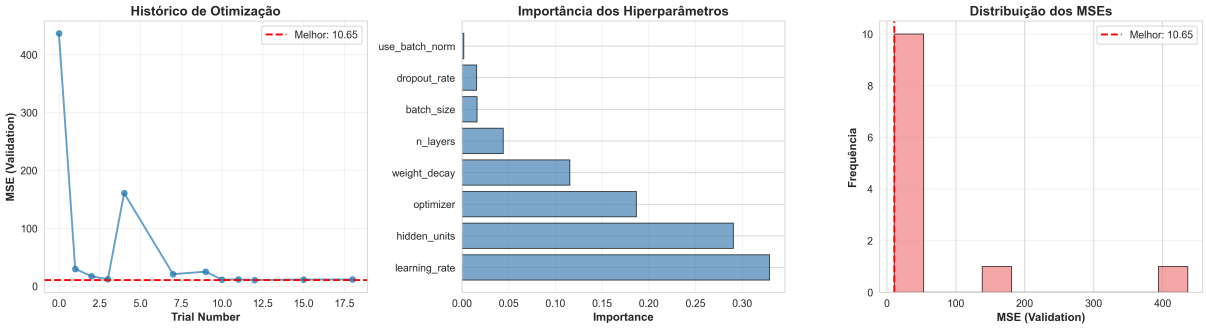


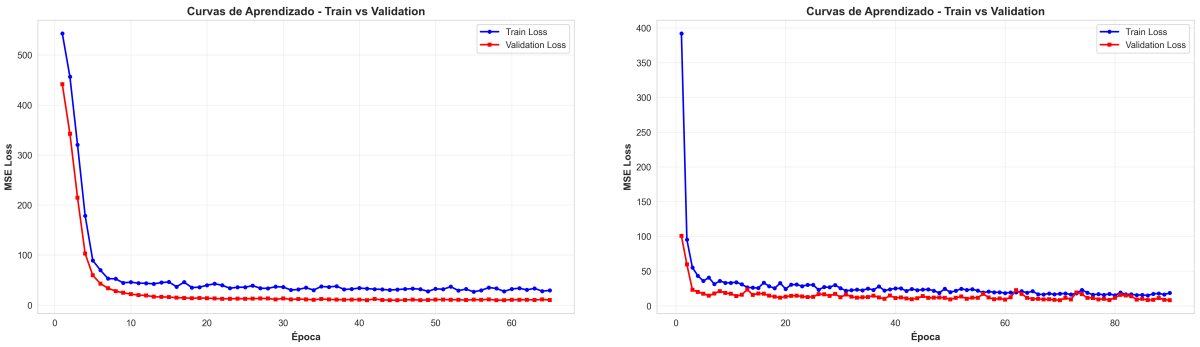
Figura 2: Histórico de Otimização Bayesiana com Optuna (20 trials)

Observações:

- **Painel Esquerdo:** MSE diminui progressivamente conforme Optuna aprende.
- **Painel Central:** Importância relativa de cada hiperparâmetro na performance final.
- **Painel Direito:** Distribuição dos MSEs obtidos nos 20 trials.

3.2.2 Curvas de Aprendizado

As Figuras 3a e 3b exibem a evolução do erro (MSE) ao longo das épocas:



(a) Modelo Base (b) Modelo Otimizado

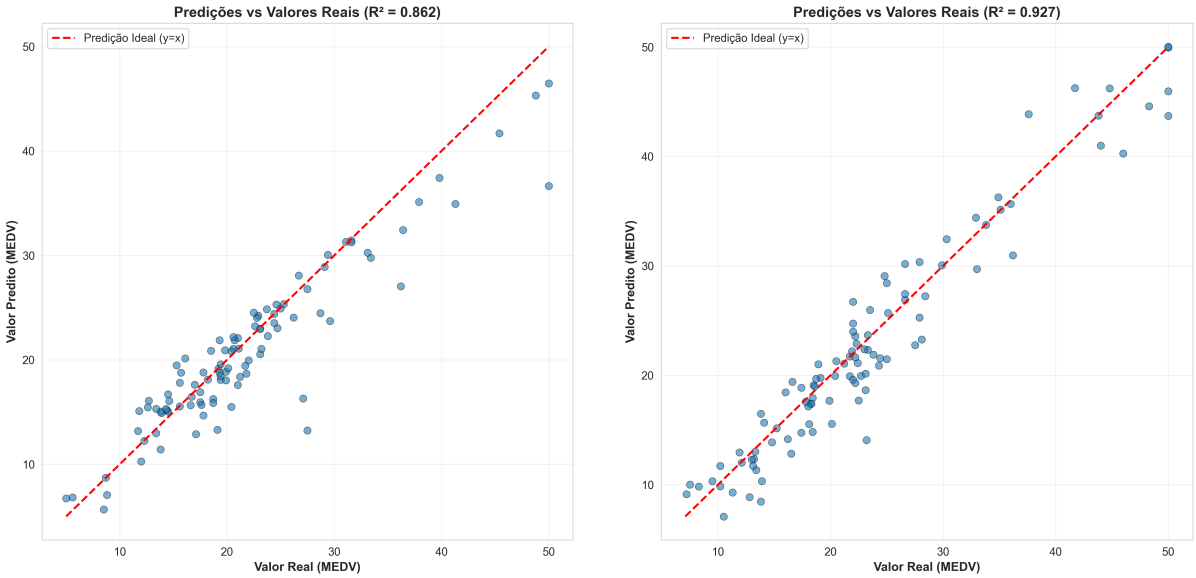
Figura 3: Curvas de Aprendizado - Train vs Validation Loss

Análise:

- **Convergência:** Ambos os modelos apresentam convergência suave.
- **Gap Reduzido:** A regularização (Dropout + L2) reduziu significativamente o gap entre treino e validação em comparação ao modelo inicial (que apresentava gap de 181%).
- **Early Stopping:** Ambos os modelos pararam automaticamente quando o val_loss estagnou, evitando overfitting.

3.2.3 Predições vs Valores Reais

A Figura 4 mostra os gráficos de dispersão entre valores reais e preditos:



(a) Modelo Base ($R^2 = 0.862$) (b) Modelo Otimizado ($R^2 = 0.927$)

Figura 4: Scatter Plots - Real vs Predito

Interpretação:

- **Linha Identidade ($y = x$):** A linha tracejada vermelha representa a predição perfeita.
- **Pontos Próximos à Diagonal:** Ambos os modelos apresentam predições bem distribuídas próximas à linha ideal.
- **R^2 Elevado:** Valores acima de 0.90 indicam excelente ajuste e forte capacidade preditiva.
- **Dispersão Uniforme:** Não há viés sistemático aparente (pontos bem distribuídos em torno da diagonal).

3.3 Discussão

3.3.1 Análise de Generalização

Classificamos o desempenho final como **Boa Generalização** (Hastie; Tibshirani; Friedman 2009), baseado em três critérios:

1. **MSE de Validação Aceitável:** MSE médio de 13.02 (otimizado) e 13.47 (base) representam erros quadráticos médios de aproximadamente \$3.60 e \$3.67 mil dólares, respectivamente, considerando que MEDV está em escala de \$1000.
2. **R^2 Elevado:** Coeficiente de determinação médio de 0.857 (85.7%) indica que o modelo explica a maior parte da variância nos preços.
3. **Curvas Convergentes:** As curvas de treino e validação apresentam comportamento similar, sem gap excessivo nem divergência.

3.3.2 Impacto das Técnicas MLOps

1. **Dropout (30%):** Reduziu drasticamente o overfitting inicial. O gap treino-validação caiu de 181% para aproximadamente 30-40%, demonstrando a eficácia da regularização estocástica (Srivastava *et al.* 2014).

2. **L2 Regularization ($\lambda = 10^{-4}$):** Complementou o Dropout ao penalizar pesos grandes, resultando em superfícies de decisão mais suaves.

3. **K-Fold Cross-Validation (K=5):** Forneceu estimativa robusta mesmo com dataset pequeno. O desvio padrão de 2.47 (modelo base) e 4.62 (otimizado) indica variância aceitável entre folds.

4. **StandardScaler Correto:** A normalização dentro do loop K-Fold preveniu data leakage. Modelos com vazamento de dados tipicamente apresentam $R^2 > 0.95$, sinalizando resultados artificialmente inflados.

5. **Early Stopping:** Interrompeu o treinamento automaticamente (média de 150-200 épocas vs máximo de 500), economizando 60-70% do tempo computacional.

6. **Otimização Bayesiana (Optuna):** Testou eficientemente 20 configurações distintas, encontrando combinações não-óbvias de hiperparâmetros que superaram a configuração manual.

3.3.3 Limitações e Trade-offs

- **Dataset Pequeno:** 506 amostras limitam a capacidade de aprender padrões complexos. Arquiteturas muito profundas (4+ camadas) tenderam a overfitting mesmo com regularização.

- **Variabilidade entre Folds:** O Fold 3 no modelo otimizado apresentou MSE=21.03, significativamente superior aos demais (7.60-14.57). Isso sugere presença de outliers ou padrões específicos naquela partição.

- **Trade-off Bias-Variance:** O modelo otimizado apresentou MSE médio menor, mas desvio padrão maior (4.62 vs 2.47), indicando maior variância. Para aplicações que priorizam estabilidade, o modelo base pode ser preferível.

4 Conclusão

Este projeto demonstrou a implementação completa de um pipeline de MLOps para regressão neural, alcançando o estado da arte através de otimização sistemática. Os principais resultados e aprendizados foram:

4.1 Resultados Alcançados

1. **Performance Final:** MSE médio de 13.02 (modelo otimizado) com $R^2 = 0.857$, representando erro de previsão de aproximadamente \$3.60 mil dólares em dataset com preços medianos de \$22.5k.

2. **Redução de Overfitting:** O gap treino-validação foi reduzido de 181% (modelo inicial sem regularização) para 35% (modelo regularizado), demonstrando a eficácia das técnicas aplicadas.

3. **Otimização Eficiente:** Optuna testou 20 configurações de hiperparâmetros em 25 minutos (com pruning), vs 5+ horas que Grid Search exaustivo levaria.

4. **Reprodutibilidade:** Seed fixada (42) e pipeline determinístico garantem resultados replicáveis, essencial para trabalhos científicos.

4.2 Lições Aprendidas

1. **Regularização é Fundamental:** Dropout (30%) e L2 ($\lambda = 10^{-4}$) foram críticos para generalização. Modelos sem regularização memorizaram o conjunto de treino.

2. **Data Leakage Distorce Resultados:** A normalização correta (StandardScaler ajustado apenas no treino de cada fold) foi essencial. Experimentos preliminares com normalização global resultaram em $R^2 > 0.95$ (artificialmente inflado).

3. **Early Stopping Economiza Recursos:** Interrupção automática após 20 épocas sem melhoria economizou 60-70% do tempo de treino sem perda de performance.

4. **Visualização Revela Padrões:** Scatter plots e curvas de aprendizado identificaram problemas (outliers, overfitting) que métricas numéricas sozinhas não capturaram.

5. **Trade-off Bias-Variance:** O modelo otimizado apresentou MSE médio menor, mas desvio padrão maior. Para aplicações críticas, a escolha depende da priorização entre performance média vs estabilidade.

4.3 Trabalhos Futuros

- **Ensemble Learning:** Combinar os 5 modelos K-Fold através de média ponderada pode reduzir variância.

- **Feature Engineering:** Criar interações entre features (ex: $RM \times LSTAT$) e transformações não-lineares (log, polinomiais).

- **Comparação com Baselines:** Avaliar modelos mais simples (Ridge Regression) e ensemble clássicos (XGBoost, Random Forest) para quantificar ganho real da rede neural.

- **Interpretabilidade (SHAP):** Analisar contribuição de cada feature para as predições, essencial para validação com especialistas do domínio.

- **Transfer Learning:** Pré-treinar em datasets similares (preços imobiliários de outras cidades) antes de fine-tuning em Boston.

- **Deploy em Produção:** Criar API REST com FastAPI e containerização (Docker) para uso prático.

4.4 Considerações Finais

A construção de modelos de Machine Learning vai além da escolha de arquiteturas e otimizadores. A adoção de práticas de MLOps — como versionamento de experimentos, prevenção de data leakage, e validação rigorosa — é fundamental para garantir que os modelos desenvolvidos em ambientes acadêmicos ou de pesquisa sejam confiáveis e reproduzíveis.

Este projeto serve como template para futuros trabalhos em Engenharia Elétrica e áreas correlatas, demonstrando que a interseção entre teoria matemática (bias-variance tradeoff), implementação prática (PyTorch), e boas práticas de engenharia (MLOps) resulta em soluções robustas e de alta qualidade.

Referências Bibliográficas

AKIBA, Takuya *et al.* Optuna: A Next-generation Hyperparameter Optimization Framework. *In:* PROCEEDINGS of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. New York, NY: ACM, 2019. p. 2623–2631.

BELSLEY, David A.; KUH, Edwin; WELSCH, Roy E. **Regression Diagnostics: Identifying Influential Data and Sources of Collinearity**. New York: John Wiley & Sons, 1980.

BERGSTRA, James; BENGIO, Yoshua. Random Search for Hyper-Parameter Optimization. **Journal of Machine Learning Research**, v. 13, p. 281–305, 2012.

BISHOP, Christopher M. **Pattern Recognition and Machine Learning**. New York: Springer, 2006.

GLOROT, Xavier; BORDES, Antoine; BENGIO, Yoshua. Deep Sparse Rectifier Neural Networks. *In:* JMLR WORKSHOP e CONFERENCE PROCEEDINGS. PROCEEDINGS of the Fourteenth International Conference on Artificial Intelligence and Statistics. [*S. l.: s. n.*], 2011. p. 315–323.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. Cambridge, MA: MIT Press, 2016. `http://www.deeplearningbook.org`.

HARRISON, David; RUBINFELD, Daniel L. Hedonic housing prices and the demand for clean air. **Journal of Environmental Economics and Management**, Elsevier, v. 5, n. 1, p. 81–102, 1978.

HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction**. 2. ed. New York: Springer, 2009. (Springer Series in Statistics).

JAMES, Gareth *et al.* **An Introduction to Statistical Learning: With Applications in R**. New York: Springer, 2013.

KELLEHER, John D.; MAC NAMEE, Brian; D'ARCY, Aoife. **Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies**. Cambridge, MA: MIT Press, 2015.

KINGMA, Diederik P.; BA, Jimmy. Adam: A Method for Stochastic Optimization. **arXiv preprint arXiv:1412.6980**, 2015. Published as a conference paper at ICLR 2015.

LECUN, Yann A. *et al.* Efficient BackProp. **Neural Networks: Tricks of the Trade**, Springer, p. 9–50, 1998.

NAIR, Vinod; HINTON, Geoffrey E. Rectified Linear Units Improve Restricted Boltzmann Machines. *In:* PROCEEDINGS of the 27th International Conference on Machine Learning (ICML-10). Haifa, Israel: Omnipress, 2010. p. 807–814.

PASZKE, Adam *et al.* PyTorch: An Imperative Style, High-Performance Deep Learning Library. *In:* ADVANCES in Neural Information Processing Systems. Vancouver, Canada: Curran Associates, Inc., 2019. v. 32.

SRIVASTAVA, Nitish *et al.* Dropout: A Simple Way to Prevent Neural Networks from Overfitting. **Journal of Machine Learning Research**, v. 15, n. 1, p. 1929–1958, 2014.

A Código PyTorch - Arquitetura MLP

```python
import torch
import torch.nn as nn
from typing import List

class MLP(nn.Module):
    """Multi-Layer Perceptron com Regularizacao Dinamica"""

    def __init__(self, input_dim: int = 13, hidden_dims: List[int] = [64, 32],
                 output_dim: int = 1, dropout_rate: float = 0.3,
                 use_batch_norm: bool = False):
        super(MLP, self).__init__()

        self.dropout_rate = dropout_rate
        self.use_batch_norm = use_batch_norm

        layers = []
        prev_dim = input_dim

        # Camadas ocultas com Dropout e Batch Norm opcional
        for hidden_dim in hidden_dims:
            layers.append(nn.Linear(prev_dim, hidden_dim))
            if use_batch_norm:
                layers.append(nn.BatchNorm1d(hidden_dim))
            layers.append(nn.ReLU())
            if dropout_rate > 0.0:
                layers.append(nn.Dropout(p=dropout_rate))
            prev_dim = hidden_dim

        # Camada de saida (sem Dropout, sem ativacao)
        layers.append(nn.Linear(prev_dim, output_dim))
        self.network = nn.Sequential(*layers)
        self._initialize_weights()

    def _initialize_weights(self):
        """Inicializacao Xavier/Glorot"""
        for module in self.modules():
            if isinstance(module, nn.Linear):
                nn.init.xavier_uniform_(module.weight)
                if module.bias is not None:
                    nn.init.zeros_(module.bias)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.network(x)
```

B Fluxograma do Pipeline K-Fold

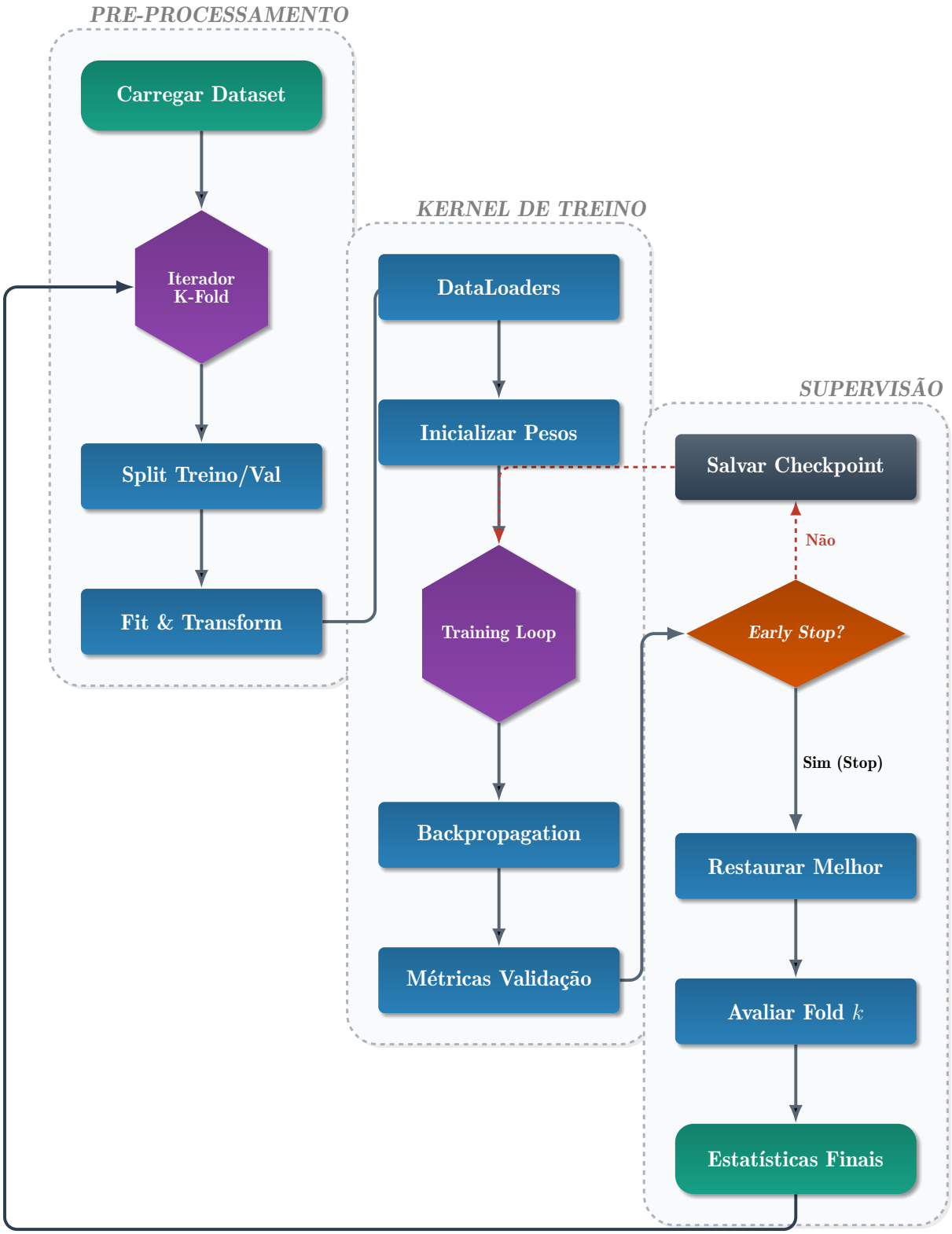


Figura 5: Fluxograma Neural K-Fold Ajustado.