Relatório Técnico: Predição de Preços de Imóveis com Redes Neurais Artificiais

Disciplina: INTELIGÊNCIA COMPUTACIONAL

Docente: Prof. Me. Bruno Brandão

Discentes: Victor Shin Iti Kanazawa Noda; Lara Portilho Marques; João Victor Araújo Costa

1. Introdução

Este trabalho aborda o desafio "House Prices - Advanced Regression Techniques" da plataforma Kaggle. O objetivo é desenvolver um modelo de regressão capaz de prever os preços de venda de imóveis, com base em um conjunto extenso de características descritivas de cada propriedade.

2. Dados

Nesta seção, descreveremos o conjunto de dados utilizado, as colunas selecionadas e rejeitadas, o tratamento de outliers, a normalização dos dados e o manejo das colunas categóricas, conforme implementado no notebook `TrabalhoRNA.ipynb`.

i. Descrição das Colunas Usadas e Não Usadas e Critérios de Remoção

O conjunto de dados de treinamento (`train.csv`) originalmente continha 1460 amostras e 81 colunas. Nem todas essas colunas foram utilizadas no modelo final. A principal variável alvo é `SalePrice`.

- **Colunas Removidas:**
    - **Critério Principal de Remoção Automática:** Colunas com mais de 50% de valores ausentes foram removidas automaticamente durante a etapa de pré-processamento simplificado na função `preprocess_data_simple`. Com base na análise de valores ausentes, as 5 colunas que se encaixam nesse critério e foram removidas são:

Colunas com mais Valores Ausentes no Dataset de Treino (Top 20):

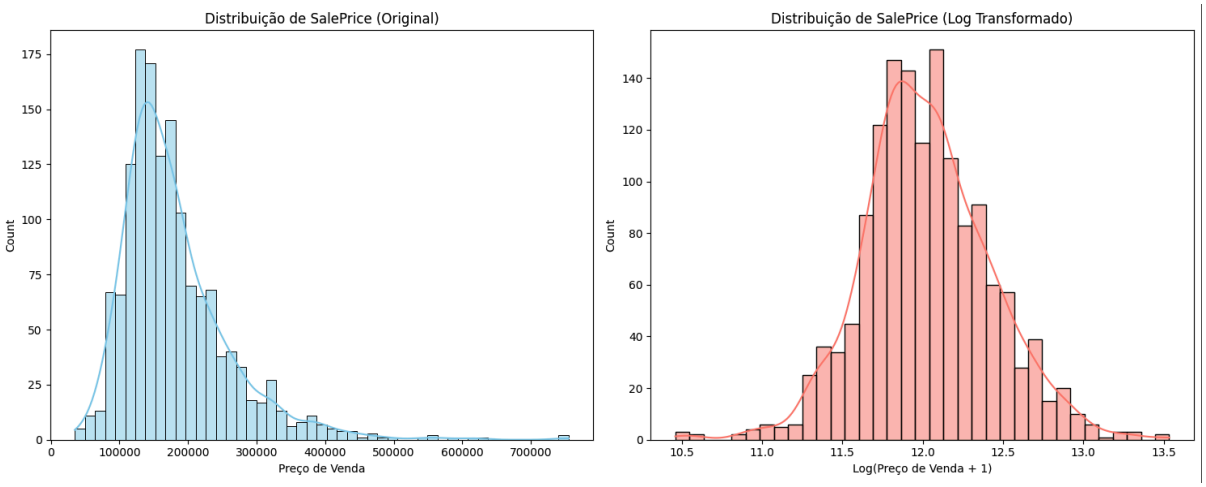
	Total Ausente	% Ausente
PoolQC	1453	99.520548
MiscFeature	1406	96.301370
Alley	1369	93.767123
Fence	1179	80.753425
MasVnrType	872	59.726027

    - A coluna `Id` também foi removida da lista de features de treinamento, pois é um identificador único para cada amostra e não possui valor preditivo intrínseco.
    - A coluna `SalePrice` original foi transformada logaritmicamente (`SalePriceLog`) para ser usada como alvo e, portanto, a coluna `SalePrice` original não foi usada diretamente como feature.
- **Colunas Usadas:**
    - Após a remoção das colunas mencionadas e da variável alvo, as colunas restantes foram consideradas para o treinamento do modelo. Estas foram divididas em features numéricas e categóricas. O pré-processamento resultou em 37 features numéricas e 38 features categóricas, totalizando 75 features utilizadas para o treinamento do modelo.
    - Valores ausentes nas features numéricas restantes foram preenchidos com a mediana da respectiva coluna, e nas features categóricas, com a moda (ou 'Unknown' caso a moda não fosse única).

ii. Definição e Remoção de Outliers

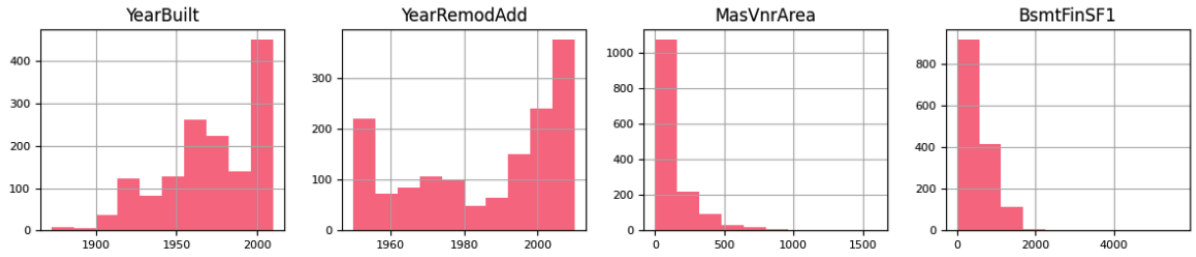
No pré-processamento simplificado implementado (`preprocess_data_simple`), não foi realizada uma etapa explícita de detecção e remoção de outliers para as *features* preditoras.

- **Tratamento da Variável Alvo:** A principal medida para lidar com a distribuição da variável alvo `SalePrice`, que apresentava assimetria positiva (valores mínimos de $34,900, máximos de $755,000, e uma média de $180,921 consideravelmente maior que a mediana de $163,000), foi a aplicação de uma transformação logarítmica (`np.log1p`). Essa transformação ajudou a normalizar sua distribuição, tornando-a mais simétrica e adequada para modelagem, como pode ser visto nos histogramas antes e depois da transformação. Esta abordagem mitiga o impacto de valores de venda extremamente altos (outliers na variável alvo) no desempenho do modelo.



iii. Normalização dos Dados

- **Variável Alvo:** Conforme mencionado acima, a variável `SalePrice` foi transformada utilizando `np.log1p` para obter `SalePriceLog`, que foi utilizada como o alvo durante o treinamento do modelo.
- **Features Preditivas:**
    1. **Imputação de Ausentes (Pré-Normalização):** Antes da normalização, os valores ausentes nas features numéricas foram preenchidos com a mediana de cada coluna.
    2. **Escalonamento (StandardScaler):** Após a divisão dos dados em conjuntos de treino e validação, as features preditivas (numéricas e categóricas já codificadas) foram normalizadas utilizando `StandardScaler` da biblioteca `scikit-learn`. Este escalonador padroniza as features removendo a média e escalonando para a variância unitária. O `scaler` foi ajustado (`fit_transform`) apenas nos dados de treino e depois aplicado (`transform`) nos dados de validação e, posteriormente, nos dados de teste para evitar vazamento de dados. Os histogramas de todas as features numéricas antes desta etapa de escalonamento (mas após a imputação). O notebook não gera histogramas individuais para todas as features *após* a aplicação do `StandardScaler`, mas o processo garante que elas tenham média próxima de 0 e desvio padrão próximo de 1.



iv. Tratamento de Colunas Categóricas

O tratamento das colunas categóricas foi realizado da seguinte forma na função `preprocess_data_simple`:

1. **Identificação:** As colunas do tipo `object` foram identificadas como categóricas.
2. **Preenchimento de Valores Ausentes:** Para cada coluna categórica, os valores ausentes (NaN) foram preenchidos com a moda (o valor mais frequente) da respectiva coluna. Caso não houvesse uma moda clara (ou seja, se a série estivesse vazia após a remoção de NaNs, o que é improvável em colunas com dados), seria preenchido com 'Unknown'.
3. **Codificação (Label Encoding):** Após o preenchimento dos valores ausentes, cada coluna categórica foi transformada em uma representação numérica utilizando `LabelEncoder` da biblioteca `scikit-learn`. Para cada coluna, um novo `LabelEncoder` foi ajustado e transformado nos valores daquela coluna (convertidos para string para garantir a compatibilidade com o encoder). Os encoders ajustados

foram armazenados para possível uso posterior (como na transformação do conjunto de teste de forma consistente). Este processo converteu todas as 38 features categóricas em representações numéricas, permitindo sua utilização pela rede neural.

3. Modelo

Esta seção descreve a definição do modelo de rede neural empregado, incluindo detalhes sobre sua arquitetura, funções de ativação, inicialização de pesos e a quantidade de parâmetros.

i. Definição do Modelo: Tamanho da Rede, Número de Camadas, Função de Ativação, Inicialização de Pesos, Quantidade de Parâmetros.

- **Estrutura Geral do Modelo:** Foi implementada uma classe `SimpleNeuralNetwork` utilizando PyTorch (`nn.Module`), que define uma rede neural do tipo Perceptron Multicamadas (MLP) para a tarefa de regressão.
- **Tamanho da Rede e Número de Camadas:**
  - A camada de entrada da rede possui um tamanho de 75 neurônios, correspondente ao número de features após o pré-processamento dos dados.
  - Foram experimentadas diferentes arquiteturas. O modelo que apresentou o melhor desempenho, denominado "SGD modelo_I", utilizou a seguinte configuração de camadas ocultas:
    - **Camada Oculta 1:** 128 neurônios.
    - **Camada Oculta 2:** 64 neurônios.
  - A **camada de saída** consiste em 1 neurônio, que fornece a predição do valor logaritmizado do preço da casa (`SalePriceLog`).
- **Função de Ativação:**
  - A função de ativação **ReLU (Rectified Linear Unit)** foi utilizada após cada camada oculta.
  - **Justificativa:** ReLU é uma escolha comum e eficaz para redes neurais profundas, pois ajuda a mitigar o problema do desaparecimento do gradiente, é computacionalmente eficiente e introduz não-linearidade no modelo, permitindo o aprendizado de relações complexas nos dados.
- **Inicialização de Pesos:**
  - A inicialização dos pesos das camadas lineares (`nn.Linear`) não foi explicitamente modificada, utilizando, portanto, a **inicialização padrão do PyTorch**.
  - **Justificativa:** Para camadas lineares, o PyTorch tipicamente usa a inicialização de Kaiming (He) uniforme quando a função de ativação subsequente é ReLU, e uma inicialização uniforme para os biases. Esta é uma prática padrão que geralmente oferece um bom ponto de partida para o treinamento de redes neurais.
- **Quantidade de Parâmetros (Pesos e Biases):**
  - Para o **melhor modelo ("SGD modelo_I")** com a arquitetura [75 -> 128 -> 64 -> 1], a quantidade total de parâmetros treináveis é:

- Camada 1 (Entrada -> Oculta1): (75 neurônios de entrada * 128 neurônios) + 1
- 28 biases = 9600 + 128 = 9728 parâmetros.
- Camada 2 (Oculta1 -> Oculta2): (128 neurônios * 64 neurônios) + 64 biases = 8192 + 64 = 8256 parâmetros.
- Camada de Saída (Oculta2 -> Saída): (64 neurônios * 1 neurônio) + 1 bias = 64 + 1 = 65 parâmetros.
- **Total de Parâmetros:** 9728 + 8256 + 65 = **18049 parâmetros**.

4. Experimentos

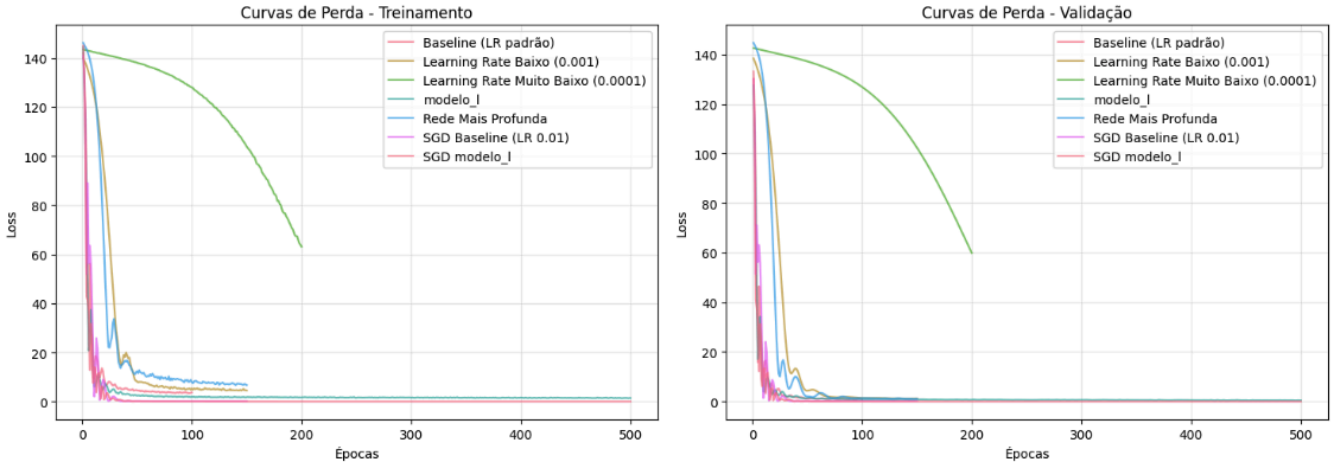
Esta seção detalha a trajetória experimental seguida para otimizar o modelo de predição de preços de casas.

i. Trajetória Científica sobre o Problema

Para encontrar a configuração de modelo mais eficaz, foram realizados múltiplos experimentos, alterando sistematicamente os hiperparâmetros e registrando os resultados. A trajetória começou com um modelo de base e progrediu através de variações nos seguintes hiperparâmetros:

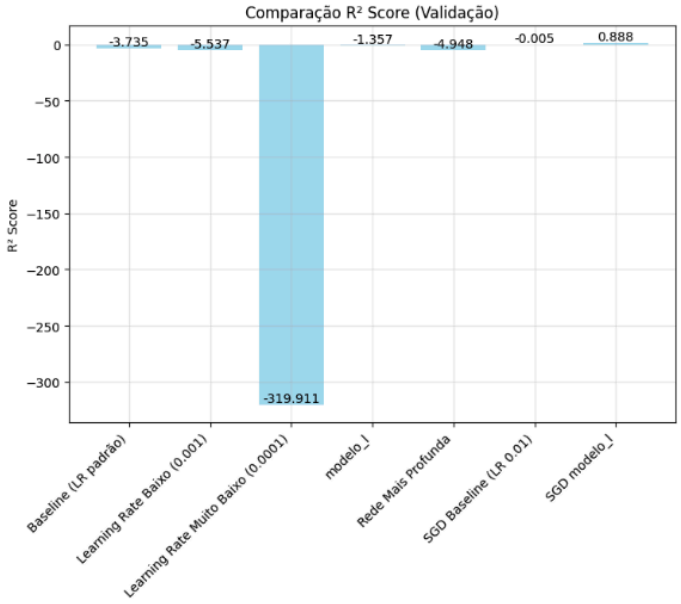
- Taxa de aprendizado (`learning_rate`)
- Arquitetura da rede (número de camadas e neurônios)
- Otimizador (Adam e SGD)
- Técnicas de regularização (Dropout e L2 `weight_decay`)

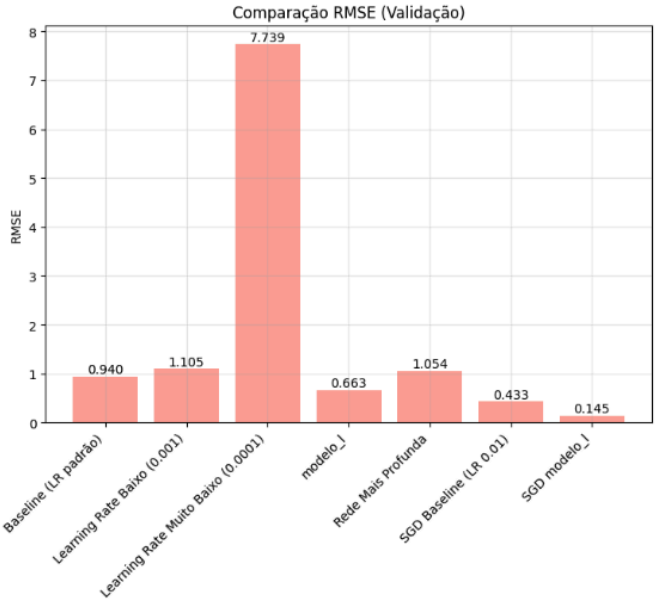
Os resultados de cada experimento foram registrados e comparados para comprovar se as mudanças resultaram em melhorias. As curvas de perda de validação, visualizadas no notebook, foram fundamentais para justificar as alterações. Por exemplo, os experimentos com taxas de aprendizado muito baixas (ex: `0.0001`) com o otimizador Adam mostraram uma convergência extremamente lenta, indicando que o modelo não aprendia de forma eficiente. Em contrapartida, configurações com o otimizador SGD demonstraram uma descida de erro mais estável e consistente.



A análise comparativa dos resultados, compilada em uma tabela e ordenada pelo R² de validação, permitiu comprovar quantitativamente qual configuração era superior. Essa análise inicial indicou que o experimento **"SGD modelo_I"** foi o mais promissor, apresentando o maior R² de validação e o menor RMSE (Erro Quadrático Médio).

	Experimento	Learning Rate	Épocas	Dropout	Weight Decay	R² Treino	R² Validação	RMSE Treino	RMSE Validação
6	SGD modelo_I	0.0100	500	0.2	0.0005	0.9240	0.8878	0.1076	0.1447
5	SGD Baseline (LR 0.01)	0.0100	150	0.2	0.0000	0.0016	-0.0045	0.3901	0.4330
3	modelo_I	0.0100	500	0.2	0.0005	0.4910	-1.3568	0.2786	0.6632
0	Baseline (LR padrão)	0.0100	100	0.2	0.0000	-0.9373	-3.7354	0.5434	0.9400
4	Rede Mais Profunda	0.0010	150	0.3	0.0001	-3.9394	-4.9478	0.8677	1.0535
1	Learning Rate Baixo (0.001)	0.0010	150	0.2	0.0000	-3.7748	-5.5375	0.8532	1.1045
2	Learning Rate Muito Baixo (0.0001)	0.0001	200	0.2	0.0000	-400.2382	-319.9110	7.8209	7.7386

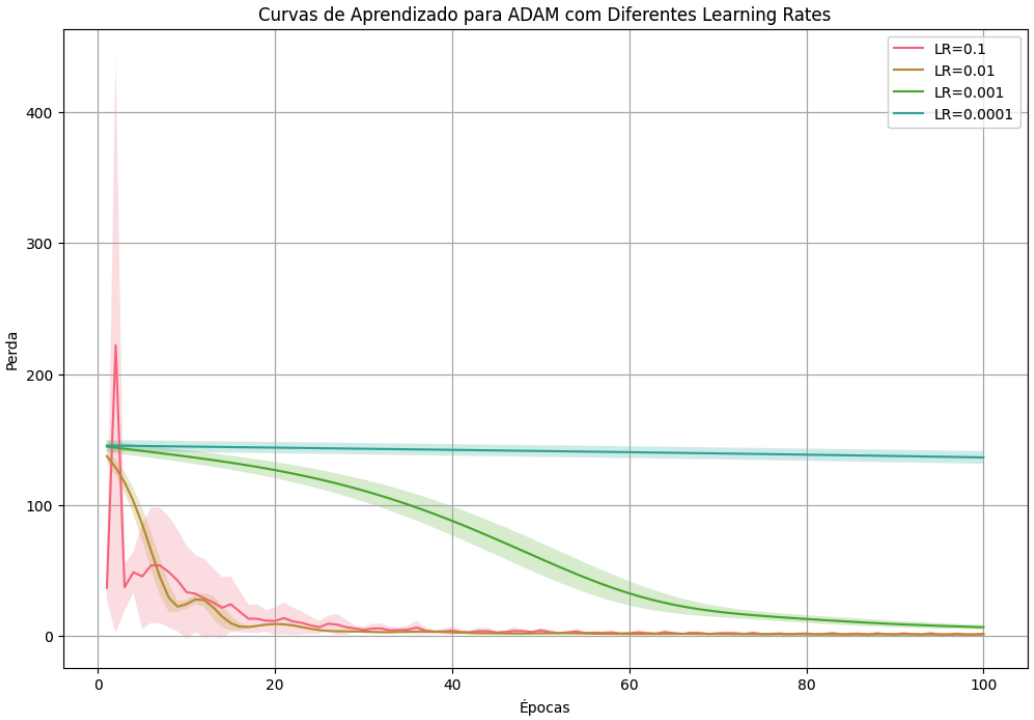




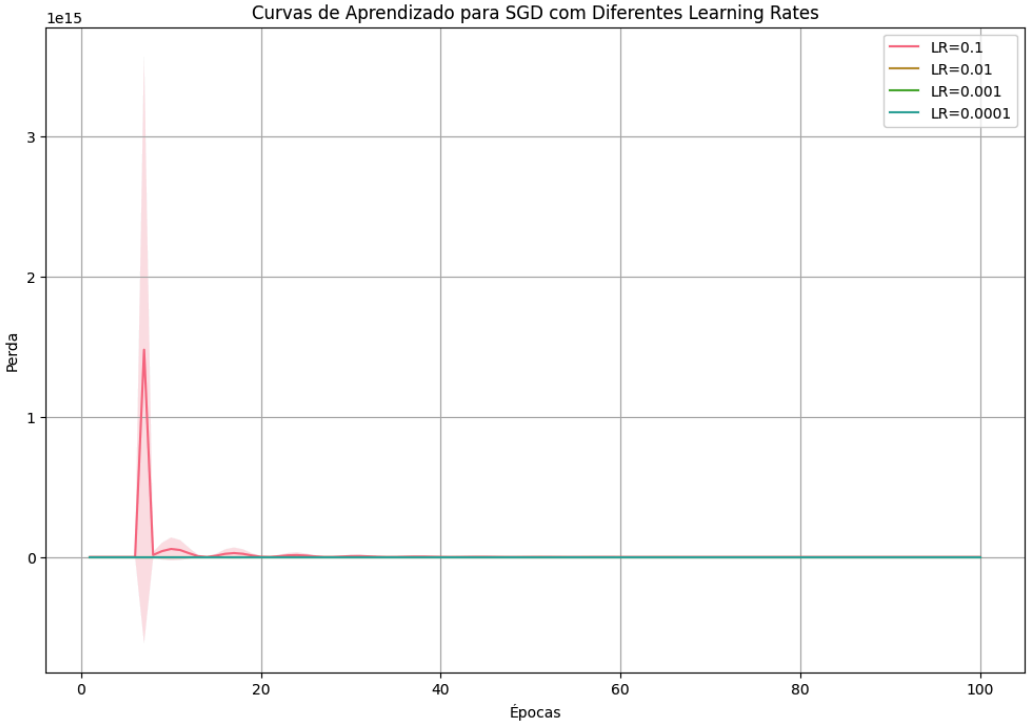
ii. Experimentos Mínimos: Otimizadores e Taxas de Aprendizado

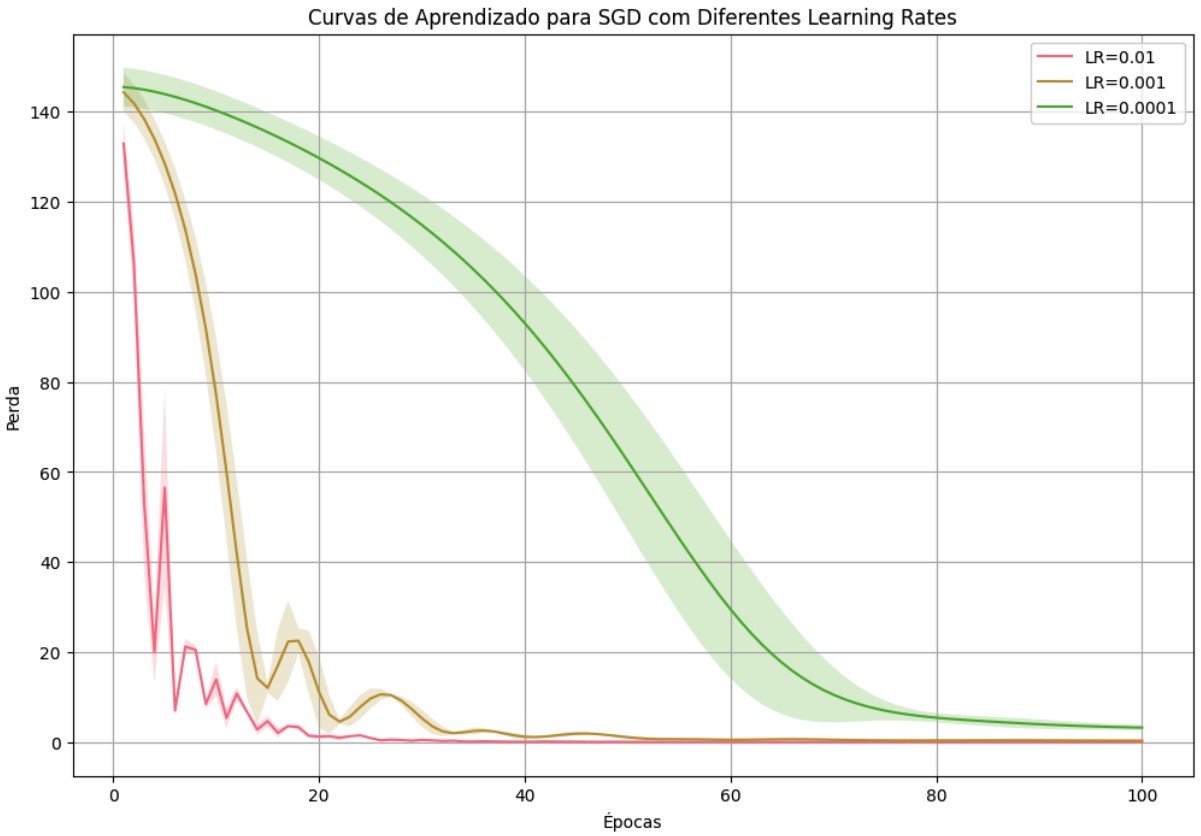
Conforme a metodologia proposta, foi conduzido um experimento detalhado para comparar o desempenho de dois otimizadores distintos, **Adam** e **SGD**, sob diferentes taxas de aprendizado. Para garantir a robustez estatística dos resultados, cada combinação de otimizador e taxa de aprendizado foi executada três vezes, permitindo o cálculo da média e do desvio padrão da performance do modelo.

- **Otimizador Adam**: A análise das curvas de aprendizado para o Adam mostrou que taxas de aprendizado mais altas (0.1 e 0.01) levaram a um treinamento instável e com perdas elevadas na validação. A taxa de 0.001 apresentou um resultado melhor, mas a convergência foi lenta, e a de 0.0001 foi ineficaz dentro do número de épocas estipulado.



- **Otimizador SGD**: O SGD apresentou um desempenho consideravelmente mais estável. A taxa de aprendizado de **0.01** demonstrou ser a mais eficaz, resultando em uma convergência rápida e consistente para o menor nível de perda de validação. As taxas de aprendizado mais baixas (0.001 e 0.0001) foram muito lentas, enquanto a taxa de 0.1 mostrou-se instável.





Os gráficos gerados, ilustram visualmente essa comparação e justificam a escolha do **SGD com learning rate de 0.01** como a combinação mais adequada para este problema. A tabela comparativa final confirmou essa observação, mostrando que essa configuração obteve, em média, o maior R² de validação (0.77) e a menor perda.

	Otimizador	Learning Rate	R² Validação (média)	R² Validação (desvio padrão)	Perda Validação Final (média)	Perda Validação Final (desvio padrão)
5	SGD	0.0100	7.718000e-01	3.260000e-02	4.260000e-02	6.100000e-03
6	SGD	0.0010	-9.586000e-01	1.466000e-01	3.655000e-01	2.740000e-02
1	ADAM	0.0100	-5.829200e+00	6.035000e-01	1.274400e+00	1.126000e-01
0	ADAM	0.1000	-8.160300e+00	5.985100e+00	1.709400e+00	1.116900e+00
7	SGD	0.0001	-1.643480e+01	2.861500e+00	3.253500e+00	5.340000e-01
2	ADAM	0.0010	-3.534710e+01	8.986400e+00	6.782800e+00	1.677000e+00
3	ADAM	0.0001	-7.298851e+02	2.581620e+01	1.363912e+02	4.817600e+00
4	SGD	0.1000	-2.091652e+10	2.958042e+10	3.903252e+09	5.520032e+09

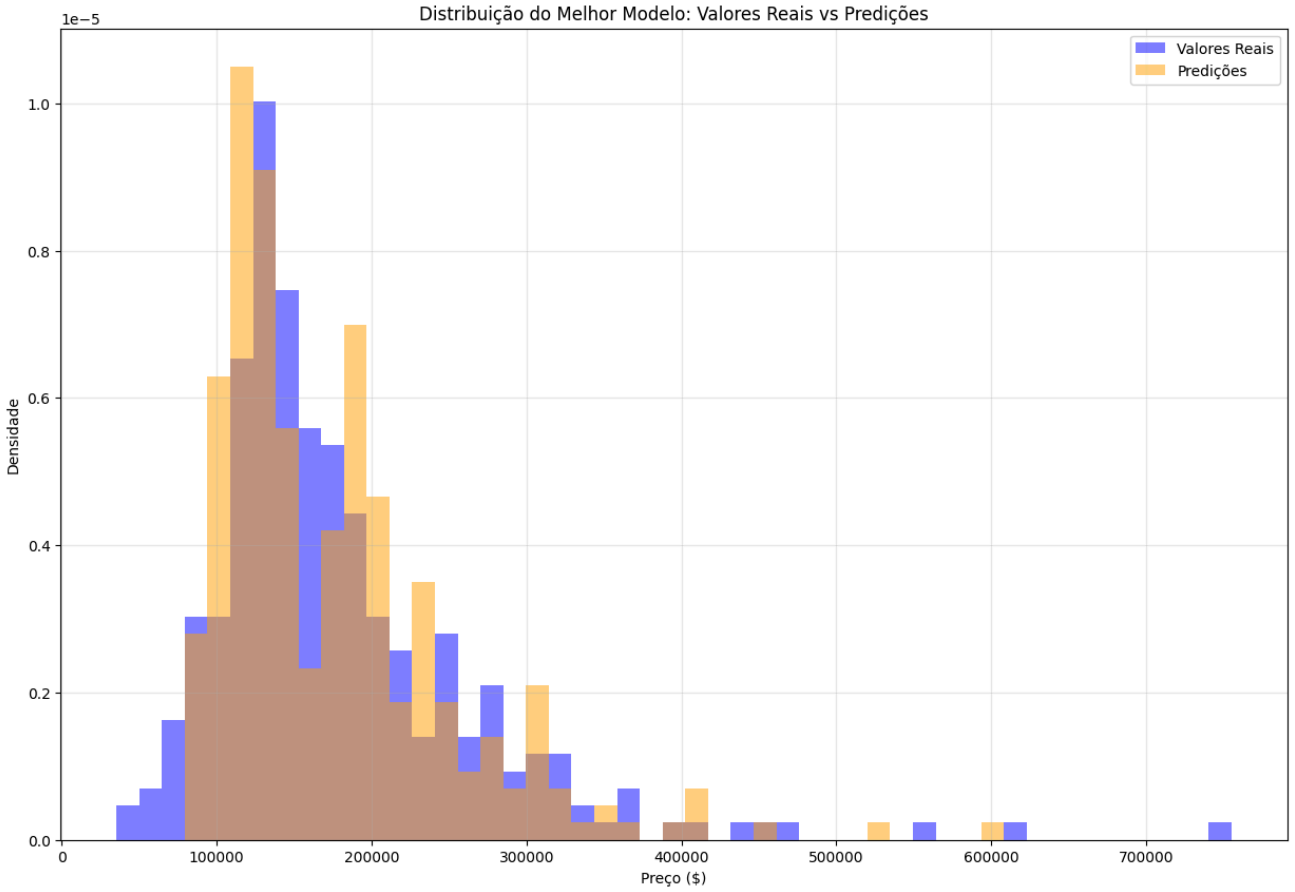
iii. Testes Comparativos e Efeito dos Hiperparâmetros

A abordagem experimental adotada seguiu o princípio de fixar a maioria dos hiperparâmetros enquanto se altera apenas um por vez para isolar e observar seu efeito.

- **Efeito do Otimizador**: Ao comparar os experimentos `low_lr_001` (Adam) com `sgd_baseline` (SGD), que possuem arquiteturas e taxas de aprendizado em ordens de grandeza similares, o impacto da escolha do otimizador ficou evidente. O SGD alcançou um R² de validação próximo de zero, enquanto o Adam resultou em

um R^2 altamente negativo, demonstrando a superioridade do SGD para este conjunto de dados e arquitetura.

- **Efeito da Arquitetura e Regularização**: A comparação entre o `SGD Baseline` e o modelo final `SGD modelo_1` ilustra o efeito da mudança na arquitetura da rede (de 3 para 2 camadas ocultas) e da introdução de `weight_decay`. Ambos usaram o otimizador SGD e a mesma taxa de aprendizado (0.01), mas o modelo final obteve um R^2 de validação drasticamente superior (**0.8878** contra **-0.0045**), provando que a rede mais simples e com regularização L2 foi uma mudança positiva e crucial para o sucesso do modelo.



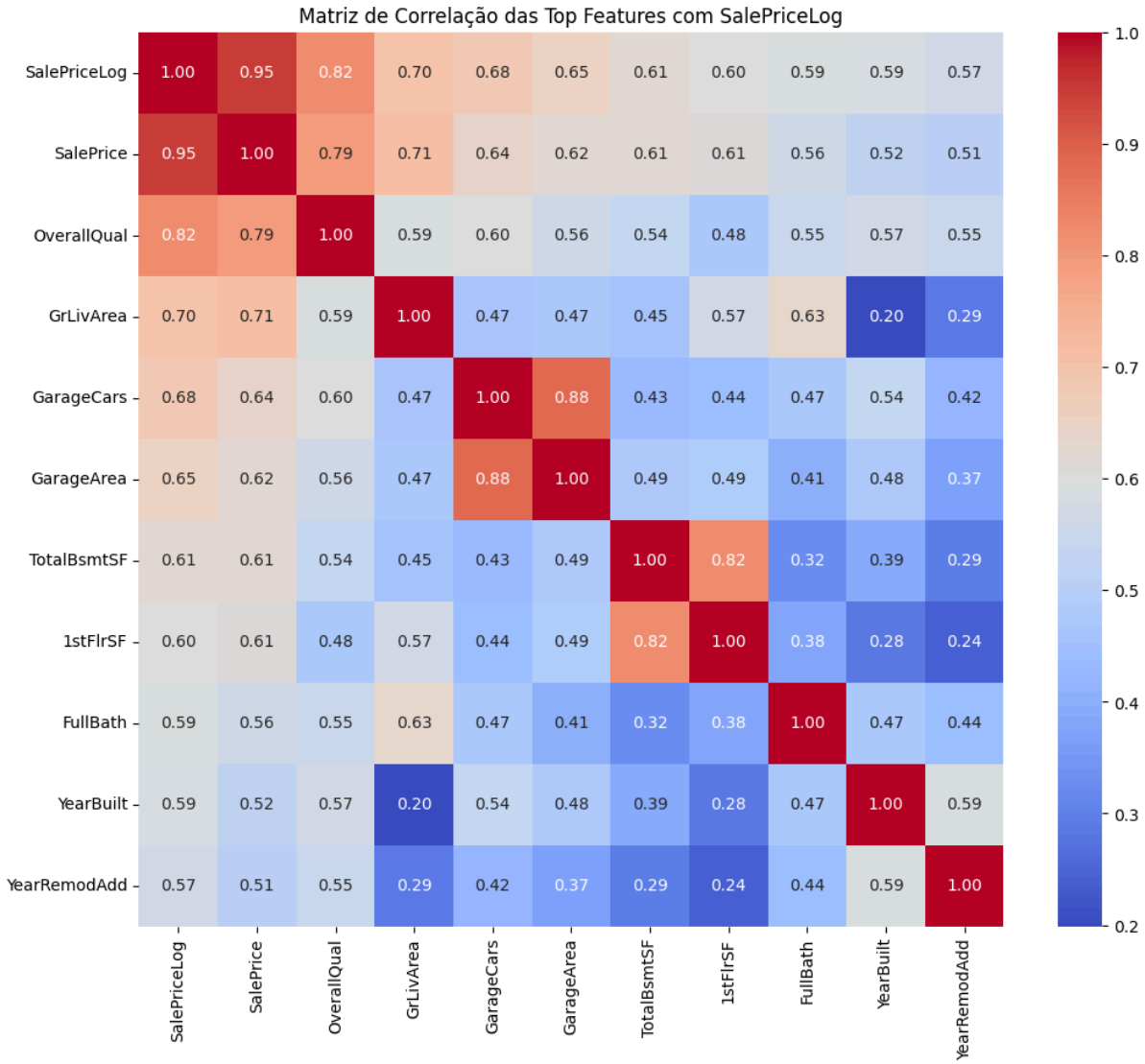
5. Conclusões

Esta seção consolida os aprendizados obtidos a partir da análise dos dados e da experimentação com diferentes configurações de modelos de rede neural, buscando responder a questões chave sobre o comportamento do modelo e a influência dos dados e hiperparâmetros.

i. Análise dos Resultados e Hiperparâmetros

A seguir, são discutidos os resultados dos experimentos em relação aos dados e hiperparâmetros, conforme as perguntas sugeridas.

1. **Quais as colunas mais importantes para a decisão do modelo?**
   A análise de correlação realizada no início do projeto oferece uma excelente indicação das features mais influentes. Ao correlacionar as variáveis numéricas com a variável alvo transformada (`SalePriceLog`), as colunas mais importantes foram aquelas ligadas à **qualidade geral e ao tamanho da propriedade**. As features com maior correlação positiva, e portanto mais importantes para o modelo, foram:



   - **OverallQual**: Qualidade geral do material e acabamento.
   - **GrLivArea**: Área de estar acima do nível do solo.
   - **GarageCars**: Capacidade da garagem em número de carros.
   - **GarageArea**: Área da garagem em pés quadrados.
   - **TotalBsmtSF**: Área total do porão em pés quadrados.
   - **1stFlrSF**: Área do primeiro andar em pés quadrados.

2. **Os exemplos em que o modelo mais erra têm algo em comum?**
   O modelo, treinado com uma maioria de dados de imóveis de valor médio, tem dificuldade em generalizar para os casos de luxo ou "fora da curva". Ele aprende os padrões da maioria, mas tende a ser conservador nos extremos, não conseguindo prever os picos de valorização de propriedades excepcionais, o que resulta em uma subestimação significativa.

3. **Qual o efeito de um hiperparâmetro específico no aprendizado?**
   O efeito da **taxa de aprendizado (learning rate)** foi claramente demonstrado nos experimentos. Este hiperparâmetro teve um impacto direto na capacidade de convergência e na estabilidade do modelo:
   - **Taxas de aprendizado altas (ex: 0.1)**: Levaram a um treinamento instável, com a perda (loss) explodindo e resultando em predições inutilizáveis. Isso indica que os passos de ajuste dos pesos eram grandes demais, fazendo com que o otimizador divergisse.
   - **Taxas de aprendizado baixas (ex: 0.001)**: Para o otimizador SGD, essa taxa se mostrou eficaz, porém com uma convergência mais lenta. Para o Adam, levou a uma melhora, mas ainda com um erro final elevado.
   - **Taxas de aprendizado muito baixas (ex: 0.0001)**: Tornaram o processo de aprendizado excessivamente lento, e o modelo não conseguiu atingir um bom desempenho no número de épocas definido.

4. **Alguma normalização é mais, ou menos, eficiente?**
   O notebook aplicou duas técnicas de normalização distintas para diferentes finalidades, ambas com sucesso.
   - A **transformação logarítmica** (`np.log1p`) foi aplicada à variável alvo `SalePrice` para corrigir sua assimetria positiva. Os histogramas antes e depois mostram que essa técnica foi altamente eficiente para aproximar a distribuição de uma normal, o que é benéfico para modelos de regressão.
   - A **padronização** (`StandardScaler`) foi aplicada às features de entrada para colocá-las em uma mesma escala (média 0 e desvio padrão 1). Isso evita que features com magnitudes maiores dominem o treinamento.

   Ambas técnicas se apresentaram bastante eficientes para a resolução do exercício.