Universidade Federal do Rio Grande do Sul
Instituto de Informática

INF01017
Aprendizado de Máquina

Trabalho Prático Final

Predição de consumo de combustível utilizando aprendizado de máquina

Luís Filipe Martini Gastmann (00276150)
Pedro Lubaszewski Lima (00341810)
Vinícius Boff Alves (00335551)

Turma U

9 de novembro de 2024

Sumário

1.1 Definição do Problema e Coleta de Dados 2
2.1 Análise Exploratória e Pré-processamento dos Dados 3
2.1.1 Análise Exploratória dos Dados 3
2.1.2 Pré-processamento dos Dados 4
3.1 Abordagem, Algoritmos e Estratégias de Avaliação 6
4.1 *Spot-checking* de Algoritmos 7
4.1.1 Revisitando os Dados após o Pré-processamento 7
4.1.2 Sumarização dos Resultados 8
5.1 Otimização de Hiperparâmetros 9
5.1.1 Análise dos Hiperparâmetros Possíveis 9
5.1.2 Resultados da Otimização 10
6.1 Comparação de Desempenhos em Dados de Teste 11
7.1 Interpretação do Modelo 12
7.1.1 Obtenção dos Coeficientes 12
7.1.2 Resultados 12
7.1.3 Análise dos Resultados 12
8.1 Conclusões Finais 14

1.1 Definição do Problema e Coleta de Dados

O objetivo deste trabalho é prever o consumo médio de combustível de um carro através de algumas das suas características e origens de fabricação. Alguns atributos das instâncias são a marca, a quantidade de cilíndros, o porte do veículo etc.

O conjunto de dados utilizado para desenvolver este trabalho foi obtido da seguinte página do Kaggle: Explore Car Performance: Fuel Efficiency Data. Essa tarefa contará com diversas técnicas de preparação dos dados para posteriormente iniciar a seleção e avaliação de modelos para essa tarefa.

2.1 Análise Exploratória e Pré-processamento dos Dados

2.1.1 Análise Exploratória dos Dados

Este *dataset* possui 550 instâncias, com 11 atributos preditores e 1 atributo alvo. Esse último torna a tarefa dos modelos em regressão, visto que o objetivo aqui é prever o consumo médio de combustível de carros. No conjunto de dados, esse atributo predito se chama *combination mpg*.

Dos atributos preditores, observa-se que há 5 atributos numéricos (*city mpg*, *cylinders*, *displacement*, *highway mpg* e *year*). Além deles, os 6 atributos restantes são categóricos: *class*, *drive*, *fuel type*, *make*, *model* e *transmission*. Com isso em mente, criou-se alguns gráficos para analisar correlações e distribuições dos dados. A seguir, o *violin plot* do atributo alvo pelo número de veículos:

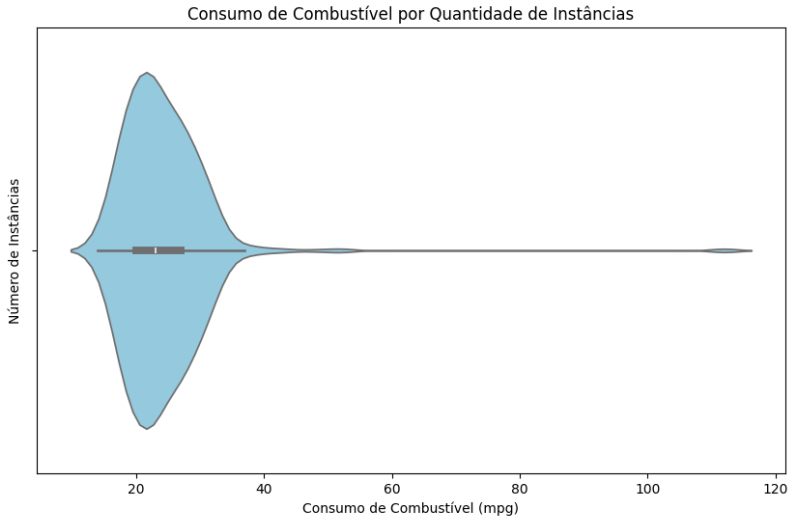

Figura 1: *Violin Plot* de Consumo Médio

Nesse ponto, já se observa que há instâncias problemáticas, claramente *outliers* que devem ser removidas do conjunto de dados. Ademais, a maior concentração dos veículos apresenta consumo médio entre 15 e 30mpg, algo que pode guiar bastante os modelos. Após isso, analisou-se o atributo de classes de veículos para ter uma ideia da sua distribuição em um gráfico de setores.

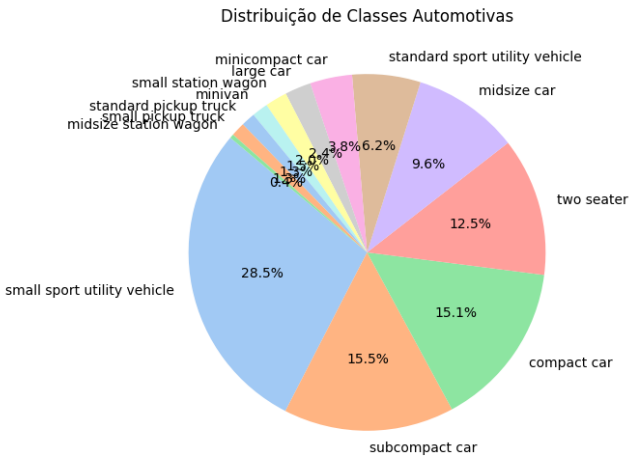

Figura 2: Distribuição de Classes de Veículos

Com ele, é perceptível que talvez seja necessário agrupar as classes de veículos e ou-

tros atributos que contenham classes com muito poucos representantes, como *midsize station wagon*, por exemplo, em uma classe geral chamada *others*. Porém, essa discussão será retomada na Subseção 4.1.1. No entanto, algo que é necessário é a codificação dos atributos categóricos em numéricos para padronizar a entrada numérica dos modelos.

Por fim, analisou-se a Correlação de Pearson entre os atributos numéricos através do *heatmap* abaixo:

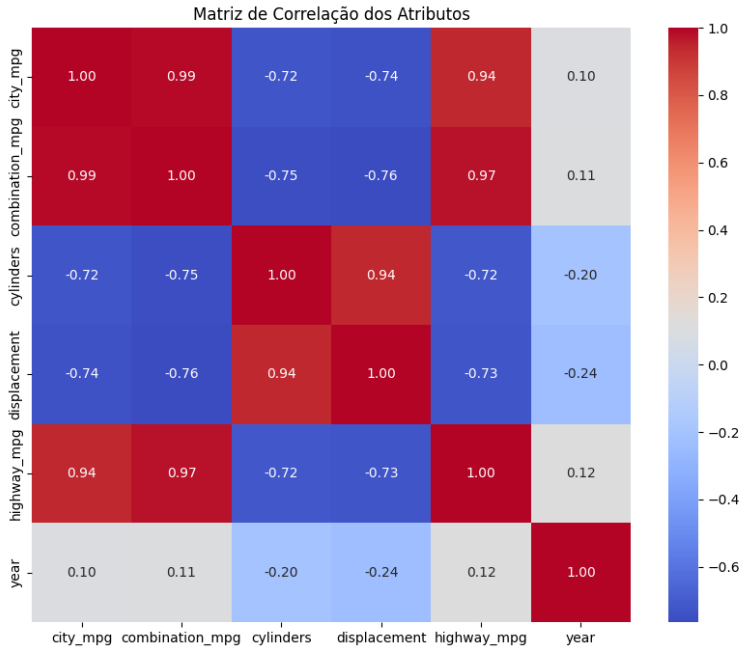

Figura 3: *Heatmap* de Correlações entre Atributos Numéricos

Olhando para o gráfico acima, percebe-se que há diversos atributos preditores que apresentam alto nível de correlação com a saída *combination mpg* e entre si. Ambos os atributos *highway mpg* e *city mpg* serão descartados do *dataset* por terem alta correlação entre si e com o valor da saída do modelo, facilitando demais a tarefa dos modelos. Além disso, observa-se que *displacement* e *cylinders* também apresentam alta correlação entre si. No entanto, a informação de *cylinders* ou cilindros de um carro é diferente da informação de *displacement* ou cilindradas desse. As cilindradas representam o volume total dos cilindros de um motor. Por conta disso, são calculados a partir dos cilindros, porém acrescentam um dado a novo ao problema. Ou seja, vale a pena mesmo assim manter ambos os atributos no *dataset*.

Isto não foi ilustrado pelos gráficos, porém os dados numéricos apresentam diversas escalas diferentes, exigindo técnicas de normalização após a separação de conjuntos de treinamento/validação e de testes.

2.1.2 Pré-processamento dos Dados

Como já discutido na seção acima e analisando alguns trabalhos realizados com esse conjunto de dados, esses sendo de Lunovian [5], da Ayşe [1] e do Priyanshu [9], implementou-se algumas estratégias de pré-processamento nos dados deste problema.

Primeiramente, realizou-se a remoção dos atributos discutidos anteriormentes. Esses sendo o *highway mpg*, *city mpg* e *displacement*, visto que apresentam alto níveis de correlação entre si e com o valor da saída do modelo. Após essa remoção, filtrou-se as instâncias com muitos atributos faltantes (em geral, carros elétricos que funcionam diferentemente dos carros que funcionam com combustíveis fósseis).

Depois disso, foi realizada a limpeza de instâncias cujo atributo alvo representava um *outlier* no gráfico de distribuição de consumo médio dos veículos. Essa estratégia é necessária para

evitar que os modelos sejam treinados com instâncias que não representam bem a grande a maioria dos veículos. Utilizou-se a medida padrão de 1.5 vezes o IQR para identificar *outliers*.

Sobre a codificação dos atributos, o grupo utilizou a codificação padrão de categórico para numérico que é o *one-hot encoding*. A justificativa para isso é que os atributos categóricos não apresentam nenhuma ordem específica que justifique utilizar uma codificação em números inteiros. Além disso, em relação à normalização, baseado no trabalho do Jason Brownlee [4] e do Matheus Vasconcelos [6], optou-se por utilizar a padronização por *min-max*, visto que é uma normalização mais eficaz na maioria dos casos, não exigindo uma distribuição normal dos dados. Tanto a codificação, quanto a normalização foram aplicadas após a separação de conjuntos de treinamento/validação e de testes separamente (evitando vazamento de dados).

Um outro problema encontrado consiste na possibilidade de não ser visto algum modelo de carro (*model*) no treinamento. Isso pode acontecer pois há uma quantidade muito grande de modelos diferentes, alguns deles com poucas instâncias. Para solucionar esse problema, "forçou-se"a haver pelo menos uma instância de cada modelo de veículo no conjunto de treinamento, antes de realizar a separação de conjunto de dados para teste. Para realizar essa última separação, utilizou-se a função `train_test_split()` com um `test_size` de 15%. Essa ação corresponde à estragégia de *holdout* vista em aula para a primeira divisão dos dados.

3.1 Abordagem, Algoritmos e Estratégias de Avaliação

Seguindo as ideias discutidas nos ítens anteriores, este problema consiste em uma tarefa de regressão. Por conta disso, o grupo buscou no conteúdo programático da disciplina algoritmos para essa tarefa. Com isso, separou-se os seguintes algoritmos de aprendizado para serem avaliados: ***k-Nearest Neighbors***, ***Random Forest***, **Regressão Linear**, **Redes Neurais** e **SVM**. A escolha desses modelos também foi inspirada nos trabalhos de Neslihan Avsar [7], de James McCaffrey [3] e de isitapol2002 [2].

Definidos esses algoritmos e as estratégias de pré-processamento, definiu-se como estratégia de validação desses algoritmos o *k-fold cross-validation*. Essa é a *magnum opus* das estratégias de validação de aprendizado supervisionado, então foi a adotada pelo grupo. O valor de `k = 13` foi definido experimentalmente. A justificativa mora na quantidade de instâncias que sobraram para a validação após a separação de modelos de carros únicos para o treinamento e do conjunto de testes. O número final de elementos por *fold* beira 18 instâncias (quantidade não muito grande, porém proporcionalmente aceita para a quantidade total de instâncias do *dataset*), gerando 13 medições de erro para cada modelo.

Em cada iteração do *k-fold cross-validation*, realizou-se a normalização e a codificação dos atributos correspondentes através dos métodos adequados do `scikit learn`. Foi fixada um `random_state = 42` para padronizar os resultados dos testes. Os hiperparâmetros utilizados nesta etapa foram todos os padrões de cada modelo da biblioteca do `scikit learn`.

A métrica de avaliação utilizada foi o *Mean Squared Error* (MSE). O grupo optou por sumarizar os resultados do *spot-checking* com apenas esta métrica (descartando o *Mean Absolute Error*) porque são métricas de natureza muito similar, já foi realizada a limpeza de *outliers* que poderiam prejudicar os resultados dessas medições e essa é uma métrica que apresenta convergência mais rápida (Nirajan Acharya [8]). Além disso, foram feitas diversas representações dessa métrica para obter maior confiança nos resultados obtidos.

4.1 *Spot-checking* de Algoritmos

4.1.1 Revisitando os Dados após o Pré-processamento

Com toda a metodologia explicada, voltar-se-á ao conjunto de dados pré-processados para realizar uma breve revisão dos dados de treinamento/validação. Primeiramente, após a remoção de *outliers*, redução de dimensionalidade e exclusão de instâncias de carros elétricos, gerou-se o seguinte gráfico de distribuição da saída a ser predita:

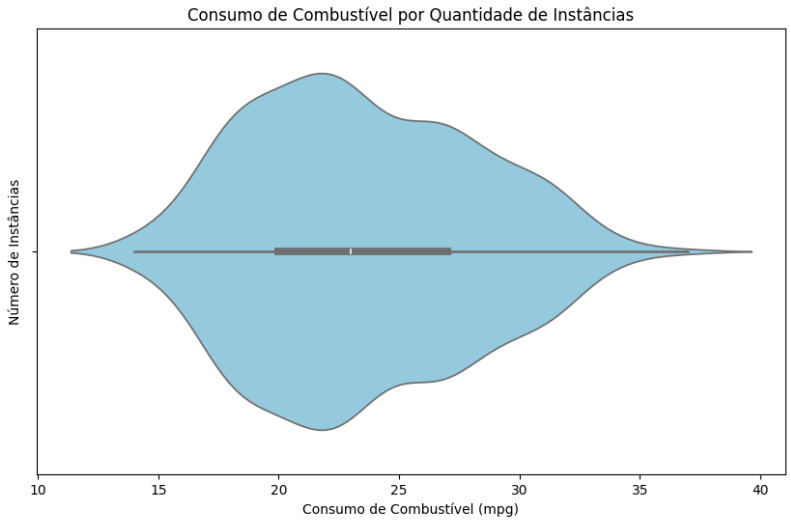

Figura 4: *Violin Plot* de Consumo Médio Pré-processado

Agora observa-se que os dados quase se distribuem em formato de uma gaussiana. Além disso, continua observando-se a concentração de dados entre 15 e 30mpg de consumo médio. O gráfico de correlação também é refeito a seguir:

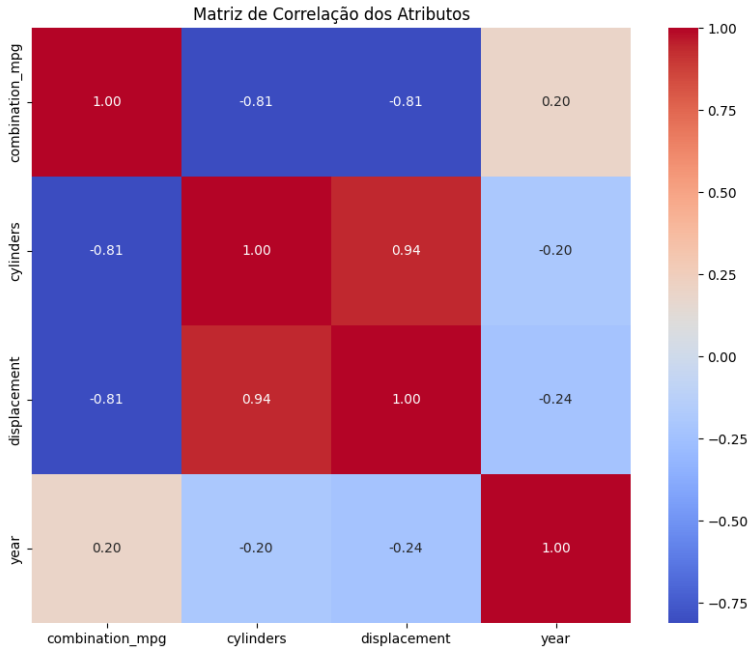

Figura 5: *Heatmap* de Correlações entre Atributos Numéricos Pré-processados

Com a remoção dos atributos citados anteriormente, observa-se uma boa redução na

correlação entre os atributos numéricos restantes. Algo que ajuda bastante a evitar a maldição da dimensionalidade.

Sobre a questão do agrupamento de atributos categóricos muito esparsos, foi percebido que o desempenho já nesta etapa se tornou inferior quando aplicado o agrupamento em comparação com o desempenho atingido sem a aplicação desse pré-processamento (resultados com média aproximada de 1,5 e desvio padrão 0,5 de MSE passaram a mostrar resultados de mais de 2,5 de média e desvio padrão de mais de 1). Por conta disso, optou-se por não adotar essa técnica para este problema.

4.1.2 Sumarização dos Resultados

Com esses dados revisitados, a seguir seguem os resultados obtidos pelos modelos:

Modelo	Média dos MSE	Desvio Padrão dos MSE
kNN	3,3227	1,3120
Random Forest	1,8084	1,0998
Regressão Linear	1,4442	0,5086
Redes Neurais	1,1889	0,4046
SVM	3,9311	1,6636

Tabela 1: Médias e Desvios Padrões dos MSE

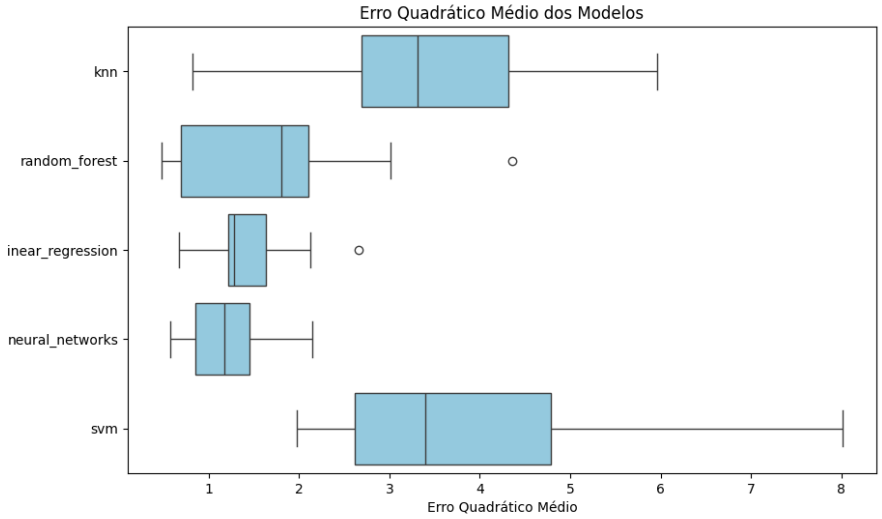

Figura 6: *Box Plot* dos MSE

Por fim, com esses resultados, mesmo que os resultados não sejam reproduzíveis exatamente com os mesmos valores a cada vez que o código é executado, fica claro que os 3 modelos mais promissores foram a ***Random Forest***, a **Regressão Linear** e as **Redes Neurais**. Como nenhum atributo foi otimizado, pode ser que os outros dois modelos se desempenhassem melhor, porém, com as configurações básicas de hiperparâmetros, esses foram os três mais promissores.

Dentre esses, fica claro que os mais eficientes nesta tarefa foram o algoritmo de Regressão Linear e o modelo de Redes Neurais, visto que os erros da *Random Forest* são mais elevados no geral quando comparado com esses últimos dois. Mesmo que as Redes Neurais apresentem média e desvio padrão melhores do que a Regressão Linear, o *box plot* acima reforça que o viés do erro da Regressão Linear é um pouco mais concentrado, próximo de um valor estável para este teste. Por conta dessas razões, serão aprofundados a *Random Forest*, o Regressor Linear e as Redes Neurais, com enfase nos últimos dois.

5.1 Otimização de Hiperparâmetros

5.1.1 Análise dos Hiperparâmetros Possíveis

O primeiro passo para a otimização dos modelos selecionados foi avaliar os hiperparâmetros relevantes para o conjunto de dados e o problema de regressão. Considerando o tamanho reduzido do conjunto de dados (550 instâncias), a dimensionalidade de 8 atributos preditores e a baixa complexidade do problema, foi realizada uma análise inicial dos parâmetros com base na documentação oficial do `scikit learn` de acordo com as páginas [11] e [3]. Através dela, utilizou-se todo o conjunto de dados de treinamento e validação unificados (não incluindo o conjunto de testes) e o método `GridSearchCV()` da biblioteca para seguir com essas avaliações.

Redes Neurais

Foram investigados os seguintes hiperparâmetros:

- `hidden_layer_sizes`: Devido à dificuldade de interpretabilidade em redes neurais, optou-se por variar `hidden_layer_sizes` para verificar se a configuração padrão (100) oferecia a complexidade adequada. Foram testadas as configurações (50, 100), (50, 50), (100, 100), (50, 100, 50) e (100).
- `early_stopping`: Esse hiperparâmetro permite encerrar o treinamento prematuramente caso a melhoria do modelo fique abaixo de um modelo predefinido (`tol`), cujo padrão é 10^{-4}, ou por 10 rodadas sem melhorias. Para agilizar o processo de treinamento, ativou-se o `early_stopping`.
- `max_iter`: Parâmetro fixado em 2500, valor mínimo identificado na etapa anterior do trabalho para garantir a convergência do modelo.
- `activation`: Foi optado por seguir com o modelo padrão linear de `activation` (`relu`) pois a função linear de ativação é adequada para problemas de regressão [12].
- Demais parâmetros: Os parâmetros restantes foram desconsiderados devido à dependência com o `solver`, não sendo compatíveis para com o `adam` ou o `lbfgs`. Isso também resultaria em um aumento considerável da quantidade de testes.

Após a primeira rodada de testes, o `hidden_layer_sizes` que melhor pontuou foi o valor padrão de (100). Por ser o valor mais baixo previsto na primeira rodada de teste, para a segunda iteração, reduziu-se a complexidade da rede e modificou-se o `solver` padrão `adam` para explorar novas alternativas.

Na segunda iteração, foram testados:

- `solver`: Além do solucionador padrão de redes neurais `adam` (rápido e eficiente tanto para modelos grandes, quanto pequenos), testou-se o `lbfgs`, recomendado na documentação oficial do *MLPRegressor* [10] por convergir e ter performance melhor para datasets menores (com menos de milhares de instâncias).
- `hidden_layer_sizes`: Foram exploradas configurações mais simples, incluindo (25), (50), (25, 50, 25), (50, 50, 25), (50,100), (50,50), (100,100), (50,100,50) e (100, 100, 100), mantendo as opções mais complexas para serem aplicadas também ao novo `solver lbfgs`.
- `early_stopping`: A ativação do `early_stopping` diminuiu o tempo de treinamento, mas a pontuação não satisfatória atribuída ao modelo resultante da primeira iteração foi atribuída à terminação prematura do modelo antes da melhor convergência, sendo desativado em seguida.

A segunda iteração demonstrou uma eficiência maior do solucionador `lbfgs` em comparação ao `adam`. O modelo com maior pontuação teve como parâmetros o `solver lbfgs` com `hidden_layer_sizes` de (25, 50, 25). Como `lbfgs` não possui nenhum parâmetro *solver-specific*, não foi realizada nenhuma outra iteração de teste.

Random Forest

Os hiperparâmetros analisados foram:

- `ccp_alpha`: Avaliou-se o uso da pós poda da árvore para mitigar *overfitting*, variando o valor de `ccp-aplha` de 0 a 0,75 em intervalos de 0,05.

- `n_estimators`: Para verificar a diversidade da floresta, o número de árvores variou de 50 a 500 em intervalos de 50.
- Outros parâmetros: O valor padrão de `criterion` (`squared error`) foi mantido por ser compatível com a métrica sendo utilizada para avaliação dos modelos, o *MSE*. Parâmetros como `max_depth` foram desconsiderados, optando-se utilizar a pós-poda ao invés da pré-poda.

Após a primeira iteração de teste, o modelo que mais pontuou foi a *Random Forest* com parâmetro `ccp_alpha` 0 (sem poda) e `n_estimators` 400, resultando em uma busca para convergir os valores escolhidos. Já na segunda iteração:

- `ccp_alpha`: Variou-se de 0 a 0,4 em intervalos de 0,025 para confirmar se a ausência de poda era realmente ideal.
- `n_estimators`: Ajustou-se de 300 a 500 árvores, em intervalos de 25, para tentar convergir em um valor ótimo.

Os testes reforçaram os achados iniciais, onde a melhor configuração teve como parâmetros `ccp_alpha` 0 (sem poda) e `n_estimators` 400. Como último caso de teste, foram testados hiperparâmetros de pré-poda na avaliação da árvore, sem obter nenhum resultado melhor.

Regressão Linear

Devido à ausência de hiperparâmetros, não foi realizada uma nova análise do modelo de Regressão Linear nesta etapa do trabalho.

5.1.2 Resultados da Otimização

Os resultados desta etapa podem ser resumidos na seguinte tabela:

Modelo	Iteração	Melhores Parâmetros	Pontuação
Redes Neurais	1	`hidden_layer_sizes` = (100), `solver` = `adam`, `early_stopping` = `True`	-7,7895
Redes Neurais	2	`hidden_layer_sizes` = (25, 50, 25), `solver` = `lbfgs`, `early_stopping` = `False`	-4,4675
Random Forest	1	`n_estimators` = 400, `ccp_alpha` = 0	-3,1145
Random Forest	2	`n_estimators` = 400, `ccp_alpha` = 0	-3,1145

Tabela 2: Resumo das Otimização de Hiperparâmetros

Devido às métricas disponibilizadas pelo `GridSearchCV()`, obteve-se pontuações negativas que continuam tendo módulo igual ao MSE utilizado anteriormente.

Além disso, percebe-se uma queda nos resultados aqui quando comparado com os resultados obtidos na seção 4.1. Isso é devido a não imputação dos dados com categorias únicas a cada iteração do processo de otimização (algo que não é possível com esse método), resultando em pontuações mais variadas. No entanto, para comparação entre os modelos de mesmo tipo, essas métricas já são válidas.

6.1 Comparação de Desempenhos em Dados de Teste

Devido à quantidade pequena de dados no conjunto de dados, foi escolhido fazer uma divisão dos dados utilizando 5-folds para determinação dos melhores hiperparâmetros, para então determinar o melhor desempenho global obtido pelos modelos de Redes Neurais, *Random Forest* e Regressão Linear com os hiperparâmetros obtidos no capítulo 5.1. As configurações finais dos modelos, portanto, foram:

- **Redes Neurais**:
  - `hidden_layer_sizes` = (25, 50, 25);
  - `solver` = `lbfgs`;
  - `early_stopping` = `False`.
- ***Random Forest***:
  - `n_estimators` = 400;
  - `ccp_alpha` = 0.
- **Regressão Linear**:
  - Parâmetros padrão.

Resultando na seguinte tabela das pontuações de cada modelo, utilizando a métrica de erro quadrático médio, para os conjuntos de treinamento e teste.

Modelo	**Pontuação (Dados de Treino)**	**Pontuação (Dados de Teste)**
Random Forest	0,3137	1,8788
Regressão Linear	0,4536	1,6111
Redes Neurais	0,03381	2,2666

Tabela 3: Resumo dos Testes dos Modelos

Percebe-se que o melhor modelo encontrado para o problema foi o **Regressor Linear**.

É possível observar, comparando os *MSE* de teste e treinamento dos três modelos, que eles sofreram com um pouco de *overfitting*, caracterizado por um desempenho muito bom no conjunto de treinamento e desempenho não muito satisfatório para o conjunto de teste, com as Redes Neurais sendo as mais afetadas.

As vantagens da *Random Forest* moram na sua capacidade de gerar diversos modelos de Árvores de Decisão (nesse caso, 400 delas), cada uma treinada nas fraquezas das outras. Isso tende a diminuir o *overfitting* dessas árvores individuais e atinge resultados estatísticamente muito bons. Por outro lado, para esse problema, é bem possível que a sua complexidade tenha sido um pouco acima do necessário;

As Redes Neurais são amplamente utilizadas por representarem modelos muito versáteis e poderosos para lidar com os mais complexos problemas. Por outro lado, elas apresentam *overfitting* elevado, tendo um desempenho muito bom no conjunto de treinamento e desempenho muito pobre para o conjunto de teste em diversos problemas. Neste caso, foi bem o fenômeno observado.

O Regressor Linear, apesar de ter apresentado o pior desempenho no conjunto de treinamento, apresentou o maior desempenho para o conjunto de testes, demonstrando um ponto positivo da simplicidade de sua implementação: a capacidade de generalizar.

7.1 Interpretação do Modelo

Como a Regressão Linear é a relação matemática entre uma variável dependente com as variáveis independentes do problema, é possível interpretar o modelo através da dos valores dos pesos que estão associados a cada variável independente. A ordem de grandeza dos pesos dita o quão significante cada variável é para a solução do problema, enquanto o sinal dita se a variável dependente irá aumentar ou diminuir à medida que a variável independente aumenta.

7.1.1 Obtenção dos Coeficientes

Como descrito na subseção 2.1.2, a solução utilizada para trabalhar com um conjunto de dados heterogêneo com dados numéricos e categóricos foi aplicar a codificação *One-Hot Encoding* para os atributos categóricos, transformando cada atributo em sua própria coluna, ou variável. No modelo de Regressão Linear cada variável tem seu próprio peso, o que implica que uma variável categórica que continha 15 diferentes valores possíveis, após o *One-Hot Encoding*, será substituído por 15 coeficientes.

Disso surge um novo desafio, como juntar estes coeficientes para possibilitar a visualização de como a categoria oficial impacta o modelo de regressão linear, levando em conta a polaridade (positivo e negativo) e a amplitude dos coeficientes. Para este fim, e para manter a simplicidade da regressão linear, foram somados todos os coeficientes de todas as variáveis, tanto categóricas quanto numéricas, resultando na seguinte tabela de coeficientes.

7.1.2 Resultados

Variável	**Coeficiente**
X_0	26,28121
cylinders	-8,79544
displacement	-1,16860
year	1.49327
class	$3.07753 \cdot 10^{-13}$
drive	$1.00919 \cdot 10^{-13}$
fuel_type	$-1.63202 \cdot 10^{-13}$
make	$1.03039 \cdot 10^{-12}$
model	$2.15827 \cdot 10^{-13}$
transmission	$-5.49560 \cdot 10^{-14}$

7.1.3 Análise dos Resultados

Percebe-se que foi dada uma importância maior aos dados numéricos que os categóricos, com cilindros sendo o mais relevante dentre eles, seguidos de deslocamento e ano de fabricação. A soma dos coeficientes dos dados categóricos foram dados um peso muito baixo, na ordem de 10^{-12} e portanto não impactam o modelo. Uma comparação com gráficos feitos por pesquisas independentes nos permite avaliar o quão realistas foram os pesos.

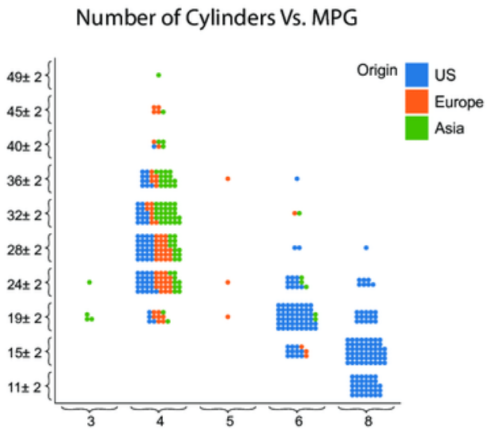

Figura 7: Milhas por galão por cilíndro.

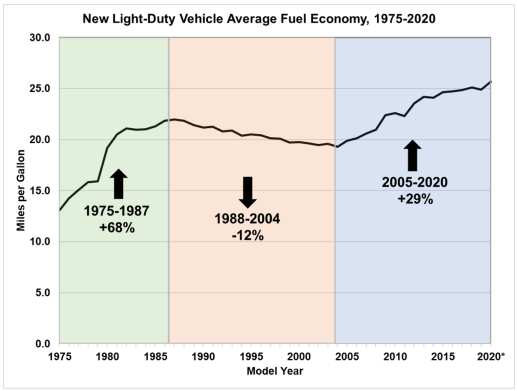

Figura 8: Milhas por galão por ano

O gráfico 7 mostra que o consumo aumenta em média 4 milhas por galão a cada cilindro, enquanto o gráfico 8 mostra que em 15 anos, ocorreu um aumento de 30% da eficiência dos combustíveis em carros, o que resulta em aproximadamente 0,5 milhas por galão ao ano, o que confirma os achados da regressão linear, onde o coeficiente atribuído ao número de cilindros é 8 vezes mais importante que o ano, portanto, enquanto a eficiência aumenta a cada ano, ela diminui a cada cilindro.

O deslocamento do carro tem uma relação direta com a quantidade de combustível utilizada, mas também com a quantidade de cilindros no carro, o que pode explicar o valor consideravelmente neutro que a variável teve.

A baixa importância dada aos atributos categóricos pode ser atribuída tanto à baixa correlação que existe entre estas variáveis independentes e a variável dependente, quanto devido ao modelo utilizado para juntar os coeficientes resultantes do *One-Hot Encoder* ser imprópria, ou pelo grau da dimensionalidade que trazem para o problema, considerando que algumas categorias foram separadas em centenas pelo *Encoder*, diminuindo sua relevância preditiva.

8.1 Conclusões Finais

Apesar do resultado ser satisfatório, a eficiência do preditor poderia ser melhorada utilizando estratificação no conjunto de treinamento, validação e teste, porém, o conjunto misto de dados numéricos e categóricos impediu o uso do parâmetro *stratify* do *train_test_split*, por tentar manter a distribuição de números raros e infrequentes das categorias numéricas nos conjuntos de treino e teste. A observabilidade poderia ser melhorada utilizando métodos para junção dos coeficientes categóricos mais adequados que a soma, como por exemplo, uma média ponderada de todos os coeficientes, ou um *box plot* que demonstre o valor médio e o desvio de todos os coeficientes dentro de cada categoria. Apesar destas dificuldades, o modelo de Regressão Linear mostrou-se muito capaz de adaptação ao problema de predição de consumo de combustível, apesar da sua simplicidade. Como visto na subseção 7.1.3, o modelo é facilmente observável, permitindo afirmar que o modelo foi capaz de identificar padrões reais do problema utilizando somente as *features* numéricas, classificando novos casos com precisão e superando modelos mais complexos e vulneráveis a *overfitting*.

FAZER REFERENCIA DOS GRAFICOS, FAZER REFERENCIA DOS SITES DO SCIKIT LEARN QUE TIRAMOS OS PARAMETROS, SÃO OS DOIS ÚLTIMOS DA BIBLIOGRAFIA

Bibliografia

[1] Ayşe Nur Durmaz. OruntuTanima-Perceptron. Website available: https://www.kaggle.com/code/ayenurdurmaz/oruntutanima-perceptron. 2024.

[2] isitapol2002. Multiple Linear Regression With scikit-learn. Website available: https://www.geeksforgeeks.org/multiple-linear-regression-with-scikit-learn/. 2022.

[3] James McCaffrey. Regression Using a scikit MLPRegressor Neural Network. Website available: https://visualstudiomagazine.com/Articles/2023/05/01/regression-scikit.aspx. 2023.

[4] Jason Brownlee. How to Use the ColumnTransformer for Data Preparation. Website available: https://machinelearningmastery.com/columntransformer-for-numerical-and-categorical-data/. 2020.

[5] Lunovian. Which Cars Contribute to a Greener Future? Website available: https://www.kaggle.com/code/anmatngu/which-cars-contribute-to-a-greener-future. 2024.

[6] Matheus Vasconcelos. Distribuição normal e a classe Standard Scaler da biblioteca Scikit-learn... Website available: https://medium.com/@mhvasconcelos/distribuio-normal-e-a-biblioteca-standard-scaler-em-python-f21c52070c6b. 2023.

[7] Neslihan Avsar. Building a Linear Regression model with Scikit-Learn — Part — 1. Website available: https://medium.com/@neslihannavsar/building-a-linear-regression-model-with-scikit-learn-part-1-eed4c04f53f9. 2022.

[8] Nirajan Acharya. Choosing Between Mean Squared Error and Mean Absolute Error. Website available: https://medium.com/@nirajan.acharya777/choosing-between-mean-squared-error-mse-and-mean-absolute-error-mae-in-regression-a-deep-dive-c16b4eeee603. 2023.

[9] Priyanshu shukla. Analysis on Car performance dataset. Website available: https://www.kaggle.com/code/priyanshu594/analysis-on-car-performance-dataset. 2024.

[10] scikit-learn. MLPRegressor. Website available: https://scikit-learn.org/1.5/modules/generated/sklearn.neural_network.MLPRegressor.html.

[11] scikit-learn. RandomForestRegressor. Website available: https://scikit-learn.org/dev/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

[12] Turing. How to Choose an Activation Function For Deep Learning. Website available: https://www.turing.com/kb/how-to-choose-an-activation-function-for-deep-learning.