João Quintella do Couto

PolieDRO.jl: Biblioteca para ciência de dados com otimização robusta a distribuições

PROJETO FINAL DE GRADUAÇÃO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DEPARTAMENTO DE INFORMÁTICA

Programa de graduação em Engenharia de Computação

Rio de Janeiro
novembro de 2024

Pontifícia Universidade Católica
do Rio de Janeiro

João Quintella do Couto

PolieDRO.jl: Biblioteca para ciência de dados com otimização robusta a distribuições

Relatório de Projeto Final

Relatório Final de Projeto de Conclusão de Curso, apresentado ao programa de Engenharia de Computação da PUC-Rio como requisito parcial para a obtenção do titulo de Engenheiro de Computação.

Orientador : Professor Alexandre Street de Aguiar
Co-orientador: Professor Davi Michel Valladão

Rio de Janeiro
novembro de 2024

Resumo

Couto, João; Street, Alexandre; Valladão, Davi. **PolieDRO.jl: Biblioteca para ciência de dados com otimização robusta a distribuições**. Rio de Janeiro, 2024. 41p. Projeto de Graduação – Departamento de Engenharia Elétrica e Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O PolieDRO é um framework introduzido recentemente que explora a conexão entre a otimização robusta a distribuições e a ciência de dados, desenvolvendo modelos para problemas de classificação e regressão puramente orientados a dados, sem a presença de hiperparâmetros. Este trabalho descreve o desenvolvimento do pacote PolieDRO.jl, uma biblioteca de código aberto na linguagem Julia que implementa deste framework. Será apresentado o processo de desenvolvimento da biblioteca, adaptações em relação aos métodos originalmente descritos e resultados da implementação.

Palavras-chave

código aberto, otimização, aprendizado de máquina, ciência de dados

Sumário

1 Introdução **4**

2 Referencial teórico **5**
2.1 Classificação e regressão 5
2.2 Otimização 8
2.3 Relação com aprendizado de máquina 9
2.4 Tecnologias existentes 11

3 Objetivos do trabalho **12**

4 Atividades realizadas **13**
4.1 Criação do ambiente de desenvolvimento 13
4.2 Cálculo de envoltórias convexas 13
4.3 Implementação das funções de perda 16
4.4 Realização de testes com datasets reais 17
4.5 Documentação 18

5 Funcionalidades da biblioteca **19**
5.1 Cálculo de envoltórias convexas 19
5.2 Criação de modelos 21
5.3 Treinamento de modelos 25
5.4 Exemplos de uso 26

6 Estrutura do projeto **29**
6.1 Organização interna 29
6.2 Testes automatizados 30
6.3 Documentação 32

7 Testes da implementação **35**

8 Considerações finais **39**

Referências bibliográficas **40**

1
Introdução

A ciência de dados é um campo que tem ganhado crescente importância no mundo moderno, onde é possível analisar quantidades cada vez maiores de informações para tomar decisões de negócio adequadas.

Métodos estatísticos de aprendizagem de máquina são comumente usados para problemas de classificação e regressão. Muitos dos algoritmos usados para estas tarefas possuem calibrações computacionalmente intensivas de hiperparâmetros, como por exemplo de coeficientes de regularização. Estes hiperparâmetros exigem múltiplas rodadas de estimação dos modelos para que seja encontrado um valor adequado.

No trabalho recente de Gutierrez, Valladão e Pagnoncelli (Gutierrez et al., 2024), é introduzido um framework que explora a conexão entre modelos de aprendizagem de máquina com otimização robusta a distribuições, traçando comparações dos modelos desenvolvidos com diversos métodos populares de machine learning, como o SVM e a regressão Lasso, porém se livrando da necessidade de hiperparâmetros.

O objetivo desse projeto foi criar de uma biblioteca open-source que implementa esse framework na linguagem Julia, e testar sua eficiência em diferentes bases de dados em comparação com métodos tradicionais de aprendizado de máquina.

Além disso, a flexibilidade do framework permite introduzir métodos originais de classificação e regressão, modelando o problema com diferentes funções de perda. Esta é uma funcionalidade também coberta pela implementação da biblioteca, sendo possível testar modelos com funções além das presentes na publicação original.

2
Referencial teórico

O framework PolieDRO, estabelecido no trabalho de Gutierrez, Valladão e Pagnoncelli (Gutierrez et al., 2024), parte de um trabalho feito por Blanchet, Kang e Murthy (Blanchet et al., 2019) que mostra a equivalência entre modelos de aprendizado de máquina tradicionais e uma formulação como problema de otimização robusta a distribuições (DRO, distributionally robust optimization) baseada na métrica da distância de Wasserstein, uma forma de medir a distância entre duas distribuições de probabilidade.

Unindo esta formulação com os trabalhos de Fernandes (Fernandes et al., 2016) e Wiesemann (Wiesemann et al., 2014), foi construído um método para formular estes problemas de DRO puramente a partir das observações de um conjunto de dados, sem os hiperparâmetros normalmente encontrados em formulações tradicionais de aprendizado de máquina.

Neste método, em resumo, são calculadas envoltórias convexas aninhadas para os dados, o que é possível a partir de algoritmos conhecidos, como o Quick Hull N-Dimensional (Barber et al., 1993) ou até problemas de programação linear (Pardalos et al., 1995). A partir da quantidade de observações em cada envoltória, é calculado empiricamente um intervalo de probabilidades para uma dada observação estar presente nesta envoltória. Estes conjuntos, juntamente com as probabilidades associadas a cada um deles, são os dados necessários para construir as formulações propostas.

Para entender o contexto acadêmico do PolieDRO, é necessário explicar alguns conceitos sobre os modelos existentes de classificação e regressão e suas dependências em hiperparâmetros. Além disso, mostra-se útil entender o que é otimização robusta a distribuições, uma generalização de dois campos da otimização sob incerteza.

2.1
Classificação e regressão

No campo de aprendizado de máquina, existem dois tipos de problema muito comuns: os problemas de classificação e os de regressão.

Em um problema de classificação, o objetivo de um modelo é ser capaz de rotular um dado a partir de observações, sendo treinado em várias instâncias onde os dados e o rótulo correto são conhecidos. Por exemplo, sabendo algumas características de uma célula, deseja-se classificá-la como saudável ou cancerígena.

Em um problema de regressão, o objetivo de um modelo é prever uma outra variável contínua sobre o dado a partir de observações. Por exemplo, sabendo algumas características de um automóvel, deseja-se estimar seu consumo de gasolina.

Existem diversos modelos capazes de realizar cada uma dessas tarefas, como o SVM e a regressão logística para classificação e a regressão LASSO para problemas de regressão. Estes e outros modelos são detalhados no livro Introduction to Statistical Learning (Gareth et al., 2023).

SVM

O SVM é um modelo de classificação que busca minimizar a perda de articulação (hinge loss) de observações classificadas erroneamente em relação a um hiperplano separador, ao mesmo tempo que procura maximizar uma margem de separação entre as observações de diferentes classificações, aplicando penalidades para observações dentro destas margens. Isto pode ser observado no gráfico desta função de perda, onde mesmo que a classe seja corretamente prevista ($x > 0$) existe uma penalidade quando a previsão está muito próxima da margem ($x < 1$).

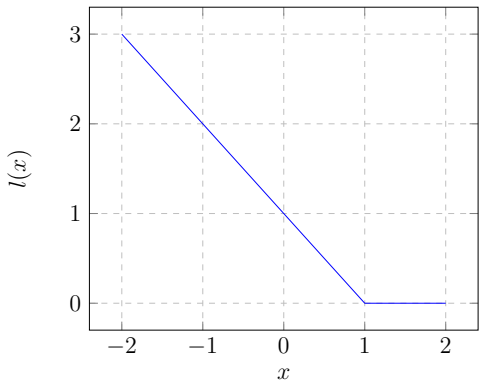


Figura 2.1: Função de perda de articulação, ou Hinge Loss

A quantidade e severidade das classificações erradas que será tolerada pelo modelo é definida por hiperparâmetros escolhidos por cross-validation. Esse tipo de treinamento é custoso, pois exige várias rodadas de treino do modelo com diferentes valores para estimar um valor adequado.

Regressão logística

Em uma regressão logística, o modelo estima a partir dos valores observados a probabilidade de um dado assumir uma certa categoria, minimizando o erro em observações em que é conhecida a categoria correta. O erro utilizado por esse modelo é a perda logística, ou "logistic loss".

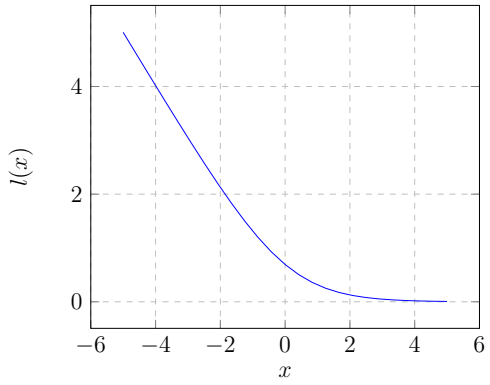


Figura 2.2: Função de perda logística, ou Logistic Loss

Para evitar um overfit do modelo, é comum utilizar uma variação regularizada do regressor, onde além do erro é punido também a quantidade e tamanho de parâmetros do modelo. O peso da punição associada a esse termo é dependente de um hiperparâmetro.

Regressão LASSO

A regressão LASSO parte da regressão linear, onde estima-se parâmetros para um modelo que minimizem uma determinada métrica de erro, em particular aqui o erro médio quadrado.

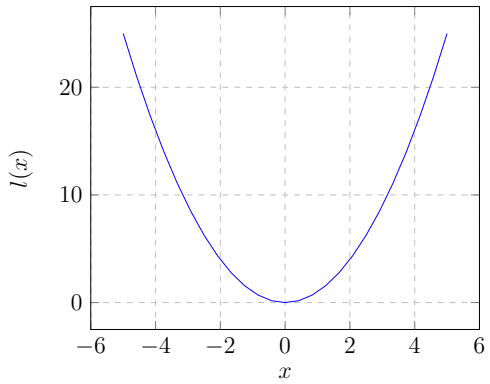


Figura 2.3: Função de perda do erro quadrático (MSE)

Da mesma maneira que a regressão logística, é introduzida uma regularização que penaliza a complexidade do modelo, evitando o overfit. Novamente como a regressão logística, essa regularização depende de um hiperparâmetro.

2.2 Otimização

O campo de otimização ou programação matemática estuda métodos para escolher um elemento ótimo obedecendo determinados critérios, como por exemplo encontrar um argumento dentro de um domínio que maximiza ou minimiza uma função-objetivo.

Para entender a otimização robusta a distribuições, campo específico que habita o framework PolieDRO, é útil entender dois outros métodos populares de otimização sob incerteza: a otimização estocástica e a otimização robusta.

Na otimização estocástica (Birge, Louveaux, 2011), estuda-se problemas em que a função-objetivo ou o domínio analisado dependem de variáveis aleatórias de distribuição conhecida. É possível, por exemplo, discretizar o problema em cenários com diferentes probabilidades de ocorrência, de maneira a criar uma aproximação da variável aleatória.

Um modelo que pode ser descrito como problema de otimização estocástica é a regressão linear. Pode-se interpretar temos N observações de uma variável aleatória $(x, y) = W$. Desejamos encontrar parâmetros β que minimizam uma função de erro $h(W, \beta)$. Utilizando como métrica o erro quadrático médio, por exemplo, tem-se $h(W, \beta) = (y - (\beta_1 * x + \beta_0))^2$:

$$\min_{\beta \in \mathcal{B}} \quad \mathbb{E}[h(\tilde{W}, \beta)] \tag{2-1}$$

Explicitando a probabilidade de ocorrência de cada cenário como $\frac{1}{N}$ pode-se escrever o problema da forma abaixo:

$$\min_{\beta \in \mathcal{B}} \quad \frac{1}{N} \sum_{i=1}^{N} h(W_i, \beta) \tag{2-2}$$

Na otimização robusta (Bertsimas et al., 2011), por outro lado, considera-se que os parâmetros incertos podem assumir um conjunto de valores, com um foco em particular no pior caso destes valores, de maneira que a solução satisfaça a todos os cenários dados como possíveis. A otimização robusta a distribuições (Goh e Sim, 2010) é uma generalização de ambos os métodos. Considera-se que os valores incertos seguem uma distribuição presente dentro de um conjunto de ambiguidade de distribuições.

2.3 Relação com aprendizado de máquina

Na publicação de Blanchet (Blanchet et al., 2019), é demonstrado que diversos métodos de aprendizado de máquina podem ser representados por um problema DRO da forma abaixo:

$$\begin{aligned} \min_{\beta \in \mathcal{B}} \quad & \max_{P \geq 0} \ \mathbb{E}_P[h(W, \beta)] \\ \text{s.t.} \quad & \int dP = 1 \\ & D(P, \hat{P}_N) \leq \lambda \end{aligned} \tag{2-3}$$

Neste problema, o objetivo é encontrar os parâmetros β que minimizam uma função de perda $h(W, \beta)$ para uma variável aleatória W de distribuição P. Pode-se pensar que estamos buscando descrever a variável W da melhor maneira possível dados os parâmetros β, minimizando o erro. Aqui, a distribuição P é desconhecida, mas ela está restrita a distribuições com uma distância da distribuição nominal dos dados $\hat{P}_N$ menor que λ, sendo utilizada a distância de Wasserstein. Este é chamado de conjunto de ambiguidade de P.

Os autores demonstram que, para diferentes funções de perda, este problema é equivalente a métodos tradicionais de ML. Por exemplo, é possível definir o SVM através deste modelo ao utilizar a perda de articulação (2.1), ou a regressão logística com a perda logística (2.2), ou a regressão LASSO com o erro médio quadrático (2.3).

O framework PolieDRO adapta este problema se utilizando de um conjunto de ambiguidade para P construído de maneira que não depende de um hiperparâmetro λ, proveniente do trabalho de Wiesemann (Wiesemann et al., 2014):

$$\min_{\beta \in \mathcal{B}} \quad \max_{P \in \mathcal{P}} \ \mathbb{E}_P[h(W, \beta)] \tag{2-4}$$

Onde $\mathcal{P} = \{P \in \mathcal{M}_+(\mathbb{R}^m) | P(W \in \mathcal{C}_i) \in [\underline{p_i}, \overline{p_i}], \forall i \in \mathcal{F}\}$. Ou seja, a distribuição P novamente não é uma distribuição conhecida, mas sabe-se que a área de P contida em cada um dos conjuntos convexos $\mathcal{C}_i$ está no intervalo de confiança $[\underline{p_i}, \overline{p_i}]$, e estes conjuntos se contém de maneira $\mathcal{C}_{\mathcal{I}} \subset \mathcal{C}_{\mathcal{I}-1} \subset \cdots \subset \mathcal{C}_1 \subset \mathcal{C}_0$. Como o conjunto $\mathcal{C}_0$ contém a totalidade da distribuição, tem-se que $\underline{p_0} = \overline{p_0} = 1$.

$\mathcal{P} = \{P \mid D(P, \hat{P}_N) \leq \lambda\}$

O PolieDRO propõe-se a modelar estes conjuntos $\mathcal{C}_i$ autocontidos através de envoltórias convexas dos dados. A envoltória convexa de todas as observações $\mathcal{C}_0$ é calculada, em seguida a envoltória $\mathcal{C}_1$ dos pontos que ficaram no interior de $\mathcal{C}_0$, recursivamente até termos a estrutura completa.

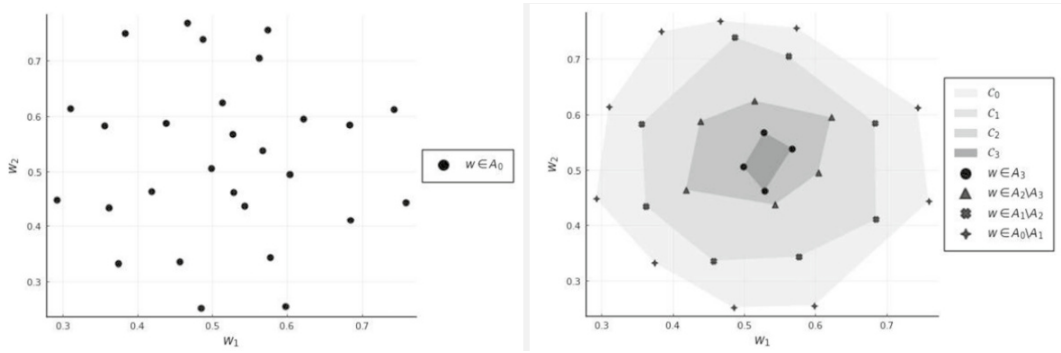


Figura 2.4: Um conjunto de observações e as envoltórias convexas necessárias para a formulação dos modelos (Gutierrez et al., 2024)

As envoltórias convexas podem ser calculadas através de diferentes algoritmos, sendo citado em particular o uso do QuickHull N-dimensional (Barber et al., 1993). Neste trabalho, diferentes métodos também serão explorados.

Cada envoltória é então associada a um intervalo de probabilidades baseado na fração de observações dentro da envoltória i ($\hat{p}_i$):

$$[\underline{p_i}, \overline{p_i}] = \Big[\hat{p}_i - q_{/\alpha 2}\sqrt{\frac{\hat{p}_i(1-\hat{p}_i)}{N}}, \hat{p}_i + q_{/\alpha 2}\sqrt{\frac{\hat{p}_i(1-\hat{p}_i)}{N}}\Big],\ \forall i \in \mathcal{F} \tag{2-5}$$

Com isso, temos um método para definir os conjuntos autocontidos $\mathcal{C}_i$ necessários para construir o problema 2-4 somente a partir dos dados iniciais. Com estas definições, é possível reescrever o problema de maneira equivalente abaixo:

$$\begin{aligned}
\min_{\beta,\lambda,\kappa}\ & \sum_{i\in\mathcal{F}}(\kappa_i\overline{p_i} - \lambda_i\underline{p_i}) \\
\text{s.t.}\ & h(w;\beta) - \sum_{l\in\mathcal{A}(i)}(\kappa_l - \lambda_l) \le 0, && \forall j \in \mathcal{V}_i, \forall i \in \mathcal{F} \\
& \lambda_i \ge 0, && \forall i \in \mathcal{F} \\
& \kappa_i \ge 0, && \forall i \in \mathcal{F} \\
& \kappa_i \ge 0, && \forall \beta \in \mathcal{B}
\end{aligned} \tag{2-6}$$

Onde $\mathcal{F}$ é o conjunto de envoltórias convexas das observações, $\mathcal{V}_i \subseteq \{1,\ldots,N\}$ contém os índices correspondentes aos vértices do conjunto $\mathcal{C}_i$, $\mathcal{A}(i)$ é definido como os índices de todas as envoltórias convexas até a envoltória i e $\overline{p_i}$ e $\underline{p_i}$ são as probabilidades associadas a cada envoltória convexa. Esta é a formulação que define o framework PolieDRO, desenvolvida na publicação original (Gutierrez et al., 2024) e implementada no pacote aqui descrito para funções de perda $h(w;\beta)$ customizáveis.

2.4 Tecnologias existentes

Para aplicações práticas de ciência de dados, existem bibliotecas em diversas linguagens de programação que implementam os métodos tradicionais de aprendizado de máquina.

Uma das linguagens com grande força neste ramo é o Python, em particular com a sua biblioteca scikit-learn (Pedregosa et al., 2011), com incontáveis modelos de colaboradores externos possuindo uma interface compatível com este ambiente. A biblioteca caret para R (Kuhn, Max, 2008) também é uma escolha popular com implementações de diversos métodos de classificação e regressão.

Foi escolhida a linguagem Julia (Bezanson et al., 2017) para o desenvolvimento da biblioteca PolieDRO devido a sua popularidade no campo da otimização, com o framework de modelagem JuMP (Lubin et al., 2023) em particular sendo de grande utilidade para implementação das formulações do trabalho original. A biblioteca MLJ (Blaom et al., 2020) é a que mais se aproxima de bibliotecas estabelecidas como o scikit-learn e o caret para a linguagem Julia, e por isso um dos objetivos do trabalho é a compatibilidade com este ambiente.

3
Objetivos do trabalho

A proposta desse trabalho é implementar o framework PolieDRO em uma biblioteca de código aberto. Isso irá permitir que cientistas de dados tenham um acesso fácil aos métodos descritos na publicação original, podendo utilizar modelos que independem de calibração de hiperparâmetros para seus problemas de classificação e regressão.

Como este trabalho trata de um framework descrito em uma publicação anterior, ele não se propõe a explicar os fundamentos teóricos de tal framework, se limitando a referenciar o trabalho original quando uma compreensão mais profunda é necessária.

Serão apresentados resultados de aplicações práticas da biblioteca, como testes dos modelos em bases de dados reais, além de experimentos com funções perda diferentes das implementadas no trabalho original.

Como anteriormente contextualizado, este trabalho foi realizado na linguagem Julia. Esta é uma linguagem que vem ganhando espaço na programação científica ao buscar balancear as facilidades de uma linguagem de alto nível como Python com a alta performance atingida por linguagens compiladas de mais baixo nível, requisito importante em um campo que envolve cálculos custosos como a programação científica.

Para base de comparação com pacotes estabelecidos de Machine Learning, serão utilizadas implementações de modelos tradicionais presentes no toolbox MLJ (Blaom et al., 2020).

Além de resultados da implementação e de testes realizados com o framework, o processo de documentação da biblioteca também será apresentado, visto que ele é essencial para que o código se torne de fácil utilização por terceiros.

4
Atividades realizadas

O projeto foi desenvolvido de acordo com o avanço do código da biblioteca, acompanhado por reuniões semanais. Em momentos que foi necessário um estudo mais aprofundado, tal pesquisa foi feita para permitir o avanço da implementação. Este processo foi seguido desde o Projeto Final I, primeira parte da disciplina, significando que o trabalho em código foi realizado desde o início.

As atividades aqui descritas não foram necessariamente cumpridas de maneira linear. Os textos indicam todo o trabalho realizado em cada tópico, no entanto estes foram realizados tanto em fases iniciais e finais do desenvolvimento, em vez de cada tópico ser realizado por completo antes do início do trabalho no seguinte.

4.1
Criação do ambiente de desenvolvimento

O repositório para o pacote foi criado assim como a estrutura para a escrita de testes automatizados, utilizando o pacote PkgTemplates.jl (equipe Julia CI, 2020). Este nos fornece um template inicial para o desenvolvimento de bibliotecas em Julia, facilitando a construção de um ambiente já integrado com testes automatizados e outros processos de CI/CD.

Com isso, um ambiente de testes já foi preparado desde o início, sendo possível implementar testes para cada novo recurso implementado no código. Estes testes são executados no momento do commit, de maneira que uma falha em algum teste gere um alerta.

4.2
Cálculo de envoltórias convexas

Como explicado na seção 2.3, para construir a formulação dos modelos no framework PolieDRO, é necessário extrair dos dados envoltórias convexas recursivamente contidas, cada uma associada a um intervalo de probabilidades. Estes conjuntos e probabilidades serão usados em passos futuros para montar o problema de otimização que irá definir os parâmetros dos modelos.

Para o cálculo das envoltórias convexas, foi utilizado inicialmente o algoritmo QuickHull N-Dimensional (Barber et al., 1993) implementado pelo pacote LazySets.jl (Forets, Schilling, 2021), que é o algoritmo citado na publicação original do PolieDRO.

Com as envoltórias em mãos, o cálculo das probabilidades associadas a cada envoltória é um processo relativamente simples, sendo apenas uma probabilidade baseada na quantidade de pontos dentro de um conjunto relativo a quantidade total. O intervalo de confiança é então retirado de uma aproximação normal da distribuição binomial que representa um ponto estar ou não no conjunto convexo.

Com este trabalho inicial, testes automatizados foram feitos para ambas funcionalidades, utilizando hipercubos de lado crescente. Desta maneira, espera-se que cada hipercubo maior contenha os menores, e é possível realizar um teste simples em que sabe as envoltórias esperadas são conhecidas. Também é verificado se o centro dos intervalos de probabilidade é a proporção de pontos dentro do conjunto testado. Estes testes serão detalhados na seção 6.2.

Quando foram realizados testes em datasets mais complexos, como exemplos reais, a implementação do Quick Hull se tornou problemática devido ao tempo de execução proibitivo, problemas de convergência e outros erros de difícil solução. Por isso, foi testada a implementação de um algoritmo de cálculo das envoltórias através de um problema de programação linear que verifica se o ponto x_k é um vértice:

$$\begin{aligned} \min_{\alpha} \quad & \alpha_k \\ \text{s.t.} \quad & \sum_{i=1}^{N} x_{i,d}\alpha_i = x_{k,d} \quad \forall d \in D \\ & \sum_{i=1}^{N} \alpha_i = 1 \\ & \alpha_i \geq 0 \qquad\qquad \forall i \in N \end{aligned} \tag{4-1}$$

Em resumo, este problema busca uma combinação convexa dos dados que gere o ponto x_k. Buscamos minimizar a contribuição do próprio ponto x_k ao colocar como objetivo o valor de α_k. Se este valor é maior que zero, então x_k é vértice. Testando todos os pontos desta maneira, podemos obter todos os vértices dos dados.

Adaptações deste problema também foram modeladas e testadas, como uma otimização proposta por Pardalos (Pardalos et al., 1995). Nesta publicação, é proposto um algoritmo em que o problema 4-1 é testado em um subconjunto crescente dos pontos originais, de maneira a produzir candidatos a vértices. Em seguida, os vértices são verificados entre si para o conjunto

verdadeiro ser obtido.

Os algoritmos foram testados em uma mesma base de dados para diferentes quantidades de dimensões e de dados disponíveis para avaliar sua performance. Abaixo, os resultados dos algoritmos QuickHull, o algoritmo de programação linear simples e o de Pardalos:

	D = 5	**D = 10**	**D = 15**	**D = 20**
N = 500	0.19	0.33	0.47	0.8
N = 1000	0.64	1.18	1.85	3.3
N = 1500	1.48	2.61	4.2	7.41
N = 2000	3.4	3.97	8.02	12.92
N = 2500	5.01	6.41	9.74	19.48
N = 3000	7.6	8.8	17.09	29.28
N = 3500	12.22	11.05	22.36	40.88
N = 4000	18.35	17.79	29.98	50.76

Tabela 4.1: Performance do QuickHull (segundos)

	D = 5	**D = 10**	**D = 15**	**D = 20**
N = 500	0.23	0.35	0.38	0.65
N = 1000	0.81	1.4	1.65	2.6
N = 1500	2.12	3.01	4.05	6.44
N = 2000	4.02	5.24	8.11	11.09
N = 2500	7.51	8.06	10.33	15.72
N = 3000	11.58	12.76	15.11	25.53
N = 3500	19.8	15.42	23.12	40.71
N = 4000	26.02	25.29	32.65	41.84

Tabela 4.2: Performance do algoritmo em PL (segundos)

	D = 5	**D = 10**	**D = 15**	**D = 20**
N = 500	0.52	0.43	0.64	1.14
N = 1000	0.7	1.52	2.36	4.68
N = 1500	1.45	3.16	5.43	10.98
N = 2000	2.62	5.49	9.56	20.31
N = 2500	4.65	8.88	16.76	27.1
N = 3000	6.59	12.01	24.43	54.04
N = 3500	10.36	17.05	33.91	61.96
N = 4000	12.71	18.29	42.7	82.64

Tabela 4.3: Performance do algoritmo em PL de Pardalos (segundos)

Em suma, verificou-se que o algoritmo QuickHull e o de PL simples possuem performances médias próximas. O QuickHull se mostra mais veloz em problemas com mais pontos e menos dimensões, enquanto que a solução por programação linear passa a ser mais eficiente em problemas de dimensão

mais alta. O algoritmo de Pardalos se mostra extremamente eficiente em baixas dimensões, sendo significativamente mais rápido até que o QuickHull, porém rapidamente se torna muito ineficiente em dimensões maiores, quando existem menos envoltórias com mais vértices em cada uma.

Foi feita a substituição do algoritmo QuickHull pela implementação em programação linear simples devido a maior conveniência e estabilidade. Isto também remove as dependências necessárias para a implementação anterior, já que da mesma forma que construímos os problemas dos modelos do PolieDRO podemos construir esta formulação sem bibliotecas adicionais.

Esta nova implementação também facilitou muito ao encontrar causas para os problemas observados com o Quick Hull, muitos destes partindo de problemas nos dados de entrada. Por isso, checagens foram colocadas nos dados de entrada do problema, discutidas mais profundamente na seção de testes (4.4).

Os resultados dos testes, no entanto, apontam para possibilidade de algoritmos mais velozes, ou ao menos de um desenvolvimento de uma heurística para definir o algoritmo utilizado para uma dada base de dados dependendo de suas dimensões.

4.3 Implementação das funções de perda

Na publicação original, três funções de perda são apresentadas no framework PolieDRO e comparadas com três algoritmos de aprendizado de máquina estabelecidos: a Hinge Loss para o SVM, a Logistic Loss para a regressão logística e o MSE para a regressão linear.

Com o cálculo das envoltórias em mãos, foram então implementadas as formulações destas primeiras funções de perda. Foi feita uma estrutura para guardar os parâmetros otimizados, assim como uma função para prever novos pontos utilizando estes parâmetros.

Com implementações iniciais realizadas, tornou-se possível fazer os primeiros testes em bases de dados. Foram feitos testes unitários no fluxo de CI comparando suas performances com modelos tradicionais implementados no pacote MLJ (Blaom et al., 2020), além de testes em mais datasets feitos em scripts externos, detalhados na seção 4.4. Com o avanço destes testes, foi possível verificar e corrigir problemas nas formulações dos modelos.

A formulação do PolieDRO, no entanto, permite o uso de uma função de perda convexa qualquer, e não somente as três exemplificadas. Para isso, foi feita uma refatoração para permitir funções genéricas, com um método em que se utiliza a formulação com uma função perda e outro método em

que é utilizado um epigrafo para modelar o máximo de diversas funções. A possibilidade de se utilizar do epigrafo é útil pois permite usar solvers lineares em funções perda que são lineares por parte (como a Hinge Loss). Os modelos originais foram reescritos utilizando a formulação geral, e também foi testado um novo modelo de regressão utilizando o erro absoluto no lugar do erro quadrático, para validar esta nova implementação generalizável.

4.4 Realização de testes com datasets reais

Os testes automatizados dos modelos são feitos com três datasets reais para cada uma das três funções de perda originais. Estes datasets são retirados do repositório de Machine Learning da UC Irvine (Dua, Graff, 2017), e os modelos são comparados com métodos tradicionais. Desta maneira, caso alguma mudança cause falhas nos testes, é porque o desempenho do modelo piorou em bases anteriormente verificadas com resultados bons.

Devido a erros e problemas de performance, rapidamente notou-se a necessidade de processar os dados antes de testes. Assim, passou-se a fazer um pré-processamento nas bases de dados. As transformações estabelecidas foram iterativamente construídas, de acordo com o que era observado.

- Inicialmente foi feita somente uma remoção de dados faltantes, primeiro removendo colunas com muitas ocorrências de valores faltantes e em seguida removendo as linhas restantes que ainda os continham.
- Como o modelo está formulado para receber matrizes de valores em ponto flutuante, características categóricas são tratadas por codificação one-hot, criando colunas para representar cada uma das possíveis classes.
- Foram observados problemas causados por dados duplicados durante a construção das envoltórias convexas, e por isso também foi feito um tratamento para remoção destes.

Para que o usuário consiga verificar mais facilmente problemas em seus dados, prevenindo a ocorrência de erros mais obscuros, foram introduzidas algumas checagens antes da construção do modelo.

- Ausência de valores infinitos: A codificação de pontos flutuantes permite valores ∞, $-\infty$ e `NaN`, porém estes causam erros quando presentes na matriz de entrada. Por isso, uma checagem foi introduzida para que o usuário fique ciente e remova estes valores para prosseguir.
- Dados duplicados: como explicado anteriormente, estes causam problemas na construção das envoltórias convexas, e por isso devem ser removidos. Uma checagem foi introduzida para que isto seja garantido.

– Dimensionalidade dos problemas: a construção das envoltórias convexas assume que haja no mínimo mais pontos do que dimensões na matriz de entrada. Por isso, é feita uma checagem para que bases com altas dimensões não causem problemas.

Para a realização de testes mais extensos, foi construído um script para automaticamente treinar os modelos em cada dataset disponível e exportar os resultados para um arquivo CSV. Este foi incrementado de acordo com o crescimento das necessidades de teste.

Foi utilizado o pacote UCIData.jl (Dunn, 2022) para acessar de maneira padronizada algumas das bases disponíveis no repositório da UC Irvine. São disponibilizadas pelo pacote em torno de 40 bases de regressão e 90 de classificação.

Rapidamente notou-se a necessidade de um ambiente de testes externo mais organizado, e por isso foi construído um repositório somente para testes com o PolieDRO. Neste, scripts testam automaticamente os modelos do PolieDRO e modelos tradicionais de aprendizado de máquina, exportando resultados de maneira mais organizada. Neste ambiente, foram gerados os resultados apresentados na seção 7.

4.5 Documentação

A documentação inicial do código ficou restrita a comentários internos, tanto explicando detalhes de implementação quanto interfaces. Seguindo o guia de Julia sobre documentação, foram escritas docstrings para as funções da interface, de forma a gerar um acesso dentro das IDEs para a documentação da função.

Em seguida, uma documentação mais extensa com exemplos foi escrita para o README da biblioteca, visível no repositório do Github. Lá, foram colocados exemplos de uso dos modelos em bases de dados e também explicações sobre como implementar uma função de perda customizada.

Com o estudo de outras bibliotecas, foram descobertos os recursos disponibilizados pelo pacote Documenter.jl (equipe Julia Docs, 2016), que permite a construção de um website para a documentação. Assim, este foi adotado e a documentação transferida para um website no Github Pages para melhor organização.

5
Funcionalidades da biblioteca

A biblioteca PolieDRO tem suas funções exportadas em dois grupos principais: as funções relativas ao uso de modelos e funções que permitem acessar o cálculo das envoltórias convexas e de suas probabilidades associadas.

É possível usar a biblioteca apenas através do primeiro grupo de funções, visto que a construção de modelos automaticamente realiza o cálculo das envoltórias e probabilidades. No entanto, caso seja a intenção do usuário treinar diversos modelos PolieDRO em um mesmo dataset, é um desperdício recalcular as envoltórias para cada um, e por isso elas podem ser pré-calculadas e acessadas pelos modelos.

5.1
Cálculo de envoltórias convexas

Para construir as envoltórias convexas necessárias para treinar o modelo, o usuário pode usar a função `calculate_convex_hulls`, com os parâmetros abaixo:

```
calculate_convex_hulls(
    X;
    silent=true
)
```

- `X::Matrix{Float64}`: Matriz $N \times D$ de pontos nos quais o modelo será treinado (N pontos, D dimensões)
- `silent::Bool`: Caso seja desejado omitir os logs de progresso durante a solução do modelo. (Valor padrão = verdadeiro)

Esta função calcula as envoltórias convexas através do algoritmo definido em 4-1 e retorna a estrutura `HullsInfo`, que guarda a informação das envoltórias convexas baseada na matriz X da forma abaixo:

```
mutable struct HullsInfo
    index_sets::Vector{Vector{Int64}}
    non_vertices::Vector{Int64}
    significance_level::Float64
    probabilities::Vector{Vector{Float64}}
end
```

– `index_sets::Vector{Vector{Int64}}`: Lista dos índices dos pontos que compõem cada envoltória, na forma abaixo:

```
[
    [
        índice do ponto 1 na envoltória 1,
        índice do ponto 2 na envoltória 1,
        # etc. (um índice por ponto na envoltória 1)
    ],
    [
        índice do ponto 1 na envoltória 2,
        índice do ponto 2 na envoltória 2,
        # etc. (um índice por ponto na envoltória 2)
    ],
    # etc. (uma entrada por envoltória)
]
```

– `non_vertices::Vector{Int64}`: Lista de íncides que não são vértices (envoltória mais interna pode ter pontos que não são vértices)

– `significance_level::Float64`: O nível de significância utilizado para calcular as probabilidades associadas a cada envoltória

– `probabilities::Vector{Vector{Int64}}`: O intervalo de confiança associado a cada envoltória convexa, na forma abaixo:

```
[
    [
        limite inferior do intervalo da envoltória 1,
        limite superior do intervalo da envoltória 1
    ],
    [
        limite inferior do intervalo da envoltória 2,
        limite superior do intervalo da envoltória 2
    ],
    # etc. (uma entrada por envoltória)
]
```

Como esta função apenas calcula as envoltórias, os valores iniciais dos campos `significance_level` e `probabilities` é `NaN` e uma lista de listas vazias. O cálculo das envoltórias convexas pode então ser feita modificando esta estrutura, utilizando a função abaixo:

```
calculate_hulls_probabilities!(
    hulls_struct,
    significance_level
)
```

– `X::Matrix{Float64}`: Matriz $N \times D$ de pontos nos quais o modelo será treinado (N pontos, D dimensões)

- `hulls_struct::HullsInfo`: Estrutura das envoltórias convexas definida anteriormente
- `significance_level::Float64`: O nível de significância utilizado para calcular as probabilidades associadas a cada envoltória

Ao chamar esta função para uma dada estrutura de envoltórias convexas, a função modifica o campo `significance_level` com o valor utilizado na função e atualiza o campo `probabilities` de acordo com os cálculos definidos em 2-5. Em Julia, a exclamação no nome da função é utilizada para denotar uma função que modifica um de seus argumentos em vez de retornar um valor diretamente, e por isso ela é usada aqui.

Esta função pode ser utilizada para recalcular probabilidades associadas a um conjunto de envoltórias já calculado, utilizando um novo nível de significância. Isso se mostra útil caso seja desejável experimentar diferentes valores para um mesmo conjunto de envoltórias, visto que o cálculo das probabilidades é computacionalmente muito mais barato do que o das envoltórias em si.
Abaixo, um exemplo de código calculando envoltórias e probabilidades associadas a uma matriz X:

```
hulls_struct = PolieDRO.calculate_convex_hulls(X)

hulls_struct.index_sets     # estrutura das envoltórias
hulls_struct.non_vertices   # pontos que não são vértices

PolieDRO.calculate_hulls_probabilities!(hulls_struct, 0.05)

hulls_struct.probabilities  # intervalo de cada envoltória

PolieDRO.calculate_hulls_probabilities!(hulls_struct, 0.01)

hulls_struct.probabilities  # intervalos atualizados
```

5.2 Criação de modelos

A geração e utilização de modelos é a parte principal do pacote. O usuário pode treinar modelos com as funções perda nativamente disponíveis ou criar as suas próprias, seja utilizando uma função ou o máximo de um conjunto de funções.

Além disso, as funções por padrão calculam internamente suas envoltórias convexas e as utilizam para construir os modelos, porém isso pode ser calculado externamente através do uso das funções da seção 5.1, sendo possível reaproveitar as envoltórias de um dataset para múltiplos modelos PolieDRO.

A construção dos modelos é feita através da função `build_model`, com diferentes parâmetros disponíveis para diferentes funcionalidades. Todas elas possuem o mesmo retorno:

- Uma estrutura do tipo `PolieDROModel`
- Uma função preditora

A estrutura do tipo `PolieDROModel` possui a estrutura definida abaixo:

```
mutable struct PolieDROModel
    model::GenericModel
    b0::Float64
    b1::Vector{Float64}
    optimized::Bool
end
```

- `model::GenericModel`: Objeto de modelo do JuMP, onde o problema está definido
- `b0::Float64`: Termo do intercepto da solução do modelo. Este é definido inicialmente como `NaN` e substituído pelo valor otimizado após a otimização
- `b1::Vector{Float64}`: Vetor de coeficientes da solução do modelo. Este é definido inicialmente como um vetor vazio e substituído pelo vetor de coeficientes após a otimização
- `optimized::Bool`: Se o modelo foi ou não otimizado

A função preditora é uma função que pode ser utilizada em conjunto com o modelo otimizado para realizar predições. Ela possui sempre a assinatura `f(model, X)` onde `model` é a estrutura do modelo e `X` é uma matriz de novos pontos X para serem previstos.

Modelos pré-definidos

Abaixo, a versão que disponibiliza as funções-perda já escritas:

```
build_model(
    X,
    y,
    loss_function;
    hulls=nothing,
    significance_level=0.1,
    silent=true
)
```

- `X::Matrix{Float64}`: Matriz $N \times D$ de pontos nos quais o modelo será treinado (N pontos, D dimensões)
- `y::Vector{Float64}`: Vetor de tamanho N com os valores da variável dependente para cada ponto da matriz X
- `loss_function::LossFunctions`: Uma das funções perda definidas no enumerador `LossFunctions`:
  - `hinge_loss`: Função perda baseada em margem para classificação
  - `logistic_loss`: Função perda baseada em probabilidade para classificação
  - `mse_loss`: Função perda baseada em distância para regressão
  - `mae_loss`: Função perda baseada em distância para regressão

Argumentos opcionais:

- `hulls::Union{HullsInfo,Nothing}`: Informação sobre as envoltórias convexas de X. Caso seja `nothing`, as envoltórias serão calculadas internamente. (Valor padrão = `nothing`)
- `significance_level::Float64`: Nível de significância utilizado para calcular as probabilidades associadas com cada envoltória, caso as envoltórias sejam calculadas internamente. (Valor padrão = 0.1)
- `silent::Bool`: Caso seja desejado omitir os logs de progresso durante a solução do modelo. (Valor padrão = verdadeiro)

Função de perda customizada

Abaixo, a implementação da função que permite o uso de uma função perda customizável:

```
build_model(
    X,
    y,
    loss_function,
    point_evaluator;
    hulls=nothing,
    significance_level=0.1,
    silent=true
)
```

Os parâmetros diferentes são:

- `loss_function::Function`: Função perda a ser minimizada
- `point_evaluator::Function`: Função que realiza a previsão de um ponto x dado os parâmetros otimizados β

A mudança principal no caso de uma função de perda customizável é a necessidade de fornecer a função de perda utilizada e uma função que utiliza os parâmetros otimizados para prever novos pontos.
A função de perda deve possuir a assinatura:
`l(x::Vector{Float64}, y::Float64, b0::VariableRef, b1::Vector{VariableRef})`

Esta realiza o cálculo da diferença do valor real para o valor previsto com os parâmetros β, e é utilizada para montar o problema do PolieDRO. Abaixo, um exemplo da funções de perda $l(x, y, \beta_0, \beta_1)$ para a implementação da função perda MSE:

$$l(x, y, \beta_0, \beta_1) = \quad (y - (\beta_0 + \beta_1^T x))^2 \tag{5-1}$$

A função avaliadora, por outro lado, deve possuir a assinatura:
`f(x::Vector{Float64}, b0::Float64, b1::Vector{Float64})`

Esta simplesmente usa os parâmetros β para prever x. Abaixo, um exemplo da função avaliadora $f(x, \beta_0, \beta_1)$ para a implementação da função perda MSE:

$$f(x, \beta_0, \beta_1) = \quad \beta_0 + \beta_1^T x \tag{5-2}$$

Função de perda através de epigrafos

Pode ser desejável construir a função de perda como epigrafo de duas ou mais funções. Isso permite preservar a linearidade do problema em funções que são lineares por parte, como por exemplo o MAE. Para isso, existe também a seguinte implementação:

```
build_model(
    X,
    y,
    loss_functions,
    point_evaluator;
    hulls=nothing,
    significance_level=0.1,
    silent=true
)
```

Os parâmetros diferentes são:

- `loss_functions::Vector{Function}`: Limites inferiores para o epigrafo que representa a função perda

Neste caso, deve-se fornecer um vetor de funções como a anteriormente definida em 5.2. A função-perda minimizada será um epigrafo limitado por baixo pelas funções definidas neste vetor.

Por exemplo, para a construção do PolieDRO para a perda MAE, a solução inicial seria simplesmente implementar esta de maneira linear por partes como abaixo:

$$l(x, y, \beta_0, \beta_1) = \quad |y - (\beta_0 + \beta_1^T x)| \tag{5-3}$$

Porém, esta função pode ser implementada através de um epigrafo sem se utilizar de uma função linear por partes. Isso permite o uso de um solver linear para otimizar o problema.

$$\begin{aligned} l_1(x, y, \beta_0, \beta_1) = \quad & y - (\beta_0 + \beta_1^T x) \\ l_2(x, y, \beta_0, \beta_1) = \quad & -y + (\beta_0 + \beta_1^T x) \end{aligned} \tag{5-4}$$

5.3 Treinamento de modelos

Com a estrutura dos modelos construída, o treinamento pode ser feita através da simples função abaixo:

```
solve_model!(
    model;
    optimizer=Ipopt.Optimizer,
    silent=true,
    attributes=nothing
)
```

Com os seguintes parâmetros:

- `model::PolieDROModel`: Estrutura do modelo retornada pela função `build_model`

Argumentos opcionais:

- `optimizer`: Um otimizador usado para resolver problemas montados pela interface JuMP. O utilizado por padrão é o Ipopt, devido a sua viabilidade para problemas não-lineares. Note que só é possível utilizar um otimizador linear caso a função-perda seja linear (ou linear por partes implementada através de epigrafo)
- `silent::Bool`: Caso seja desejado omitir os logs de progresso durante a solução do modelo. (Valor padrão = verdadeiro)
- `attributes::Union{Nothing, Dict{String}}`: Caso seja desejado passar flags para o otimizador, pode ser utilizado neste argumento um dicionário da forma "atributo"→ "valor". (Valor padrão = `nothing`)

Esta função não possui nenhum retorno, somente modifica a estrutura do modelo com os parâmetros otimizados.

5.4 Exemplos de uso

Abaixo, um código de exemplo simples para treinamento de um modelo de regressão pré-implementado. No exemplo, é utilizado o modelo de MSE.

```
# dataset de treino
X::Matrix{Float64}
y::Vector{Float64}

# dados de teste
Xtest::Matrix{Float64}

# construindo modelo
model_mse, predictor_mse = PolieDRO.build_model(X, y, PolieDRO.mse_loss)

# fitando modelo
solve_model!(model_mse)

# realizando predição
y_predict = predictor_mse(model_mse, Xtest)
```

Caso sejam treinados mais de um modelo no mesmo dataset, é interessante que seja realizado o cálculo das envoltórias convexas somente uma vez. Ele pode ser feito externamente e utilizado nos modelos da maneira abaixo. Neste exemplo, são utilizados os modelos de classificação baseados em perda de articulação e perda logística.

```
# dataset de treino
X::Matrix{Float64}
y::Vector{Float64}

# dados de teste
Xtest::Matrix{Float64}

# calculando envoltórias
hulls = PolieDRO.calculate_convex_hulls(Xtrain)

# construindo modelo hinge loss
model_hl, predictor_hl = PolieDRO.build_model(Xtrain, ytrain, PolieDRO.hinge_loss; hulls=hulls)

# construindo modelo logistic loss
model_ll, predictor_ll = PolieDRO.build_model(X, y, PolieDRO.logistic_loss; hulls=hulls)

# fitando modelos
solve_model!(model_hl)
solve_model!(model_ll)

# realizando predições
y_predict_hl = predictor_mse(model_hl, Xtest)
y_predict_ll = predictor_mae(model_ll, Xtest)
```

Abaixo, dois exemplos de implementação de um modelo com função perda customizada. É utilizado o MAE para criar um modelo de regressão, através de uma única função e de um epigrafo (máximo de duas

funções). Este modelo está disponível de maneira pré-definida no pacote (`PolieDRO.mae_loss`).
Implementação por função única:

```
# erro absoluto
function abs_error(x, y, b0::VariableRef, b1::Vector{VariableRef})
    return abs(y-(b0+sum(b1[k]*x[k] for k in eachindex(b1))))
end

# função de predição
# prevê ponto com parâmetros otimizados
function mae_evaluator(x, b0, b1)
    return b0 + b1'x
end

# dataset de treino
X::Matrix{Float64}
y::Vector{Float64}

# dados de teste
Xtest::Matrix{Float64}

# construindo modelo customizado por epigrafo
model_ll, predictor_ll = PolieDRO.build_model(X, y, abs_error, mae_evaluator)

# fitando modelos
solve_model!(model_hl)
solve_model!(model_ll)

# realizando predições
y_predict_hl = predictor_mse(model_hl, Xtest)
y_predict_ll = predictor_mae(model_ll, Xtest)
```

Implementação por epigrafo:

```
# parte positiva do erro
function pos_err(x, y, b0::VariableRef, b1::Vector{VariableRef})
    return (y-(b0+sum(b1[k]*x[k] for k in eachindex(b1))))
end

# parte negativa do erro
function neg_err(x, y, b0::VariableRef, b1::Vector{VariableRef})
    return -(y-(b0+sum(b1[k]*x[k] for k in eachindex(b1))))
end

# função de predição
# prevê ponto com parâmetros otimizados
function mae_evaluator(x, b0, b1)
    return b0 + b1'x
end

# dataset de treino
X::Matrix{Float64}
y::Vector{Float64}

# dados de teste
Xtest::Matrix{Float64}
```

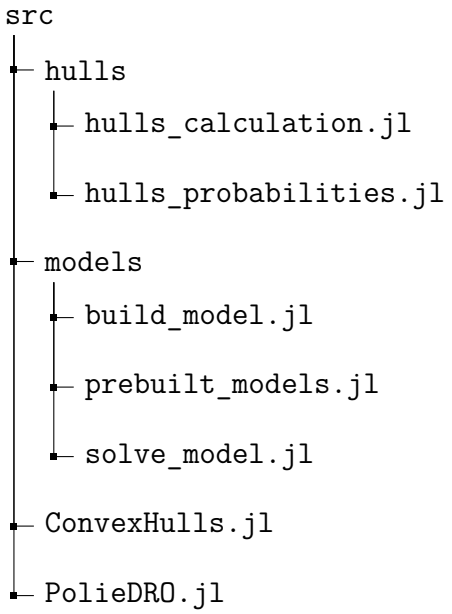
```
# construindo modelo customizado por epigrafo
model_ll, predictor_ll = PolieDRO.build_model(X, y, [pos_err, neg_err], mae_evaluator)

# fitando modelos
solve_model!(model_hl)
solve_model!(model_ll)

# realizando predições
y_predict_hl = predictor_mse(model_hl, Xtest)
y_predict_ll = predictor_mae(model_ll, Xtest)
```

6
Estrutura do projeto

6.1
Organização interna

Internamente, o pacote possui uma estrutura principal dividida em diretório fonte (src), CI/CD e testes automatizados (test) e documentação (docs). Nesta seção, o foco é o detalhamento do diretório fonte, enquanto que nas duas seções seguintes serão explorados o ambiente de CI/CD e a documentação.

Abaixo a estrutura das principais pastas e arquivos do diretório fonte:



Cálculo de envoltórias

As funções relativas ao cálculo de envoltórias convexas e suas probabilidades associadas (5.1) se encontra nos arquivos presentes no diretório `hulls/`. Estes arquivos são então incluídos com as suas dependências no arquivo `ConvexHulls.jl`, uma escolha de design recomendada pela linguagem para organização de pacotes.

O arquivo `hulls_calculation.jl` contém a definição da função `calculate_convex_hulls` e da estrutura `HullsInfo` retornada por

esta função. O arquivo `hulls_probabilities.jl` contém a função `calculate_hulls_probabilities!`, relacionada ao cálculo das probabilidades associadas às envoltórias.

Módulo principal e definição de modelos

As funções relativas à construção de modelos (5.2) e de sua estimação (5.3) se encontra nos arquivos presentes no diretório `models/`. Estes arquivos são então incluídos com as suas dependências no arquivo `PolieDRO.jl`, onde também é incluso o arquivo `ConvexHulls.jl`. É no arquivo `PolieDRO.jl` onde está definido o módulo principal do pacote PolieDRO, onde todas as funcionalidades são exportadas para uso.

O arquivo `build_model.jl` contém as duas definições da função `build_model` para modelos customizáveis (com função única e com epigrafo), além da definição da estrutura `PolieDROModel` retornada por estas funções.

As funções definidas no arquivo `build_model.jl` são então utilizadas para construir os modelos pré-definidos no pacote, presentes no arquivo `prebuilt_models.jl`. Lá também está definido o enumerador `LossFunctions.jl`, que define as funções de perda pré-implementadas.

A função `solve_model!` relativa ao treinamento de modelos está definida no arquivo `solve_model.jl`.

6.2 Testes automatizados

Os testes automatizados são realizados em um workflow definido em `.github/workflows/CI.yml` quando é feito um PR ou commit na branch principal do projeto. Os testes são executados nas versões 1.7 e nightly do Julia, em uma máquina virtual Ubuntu providenciada pelo Github Actions.

A estrutura de testes é internamente organizada da maneira abaixo:

```
test
├─ hulls
│  └─ hull_tests.jl
├─ models
│  ├─ classification_tests.jl
│  └─ regression_tests.jl
├─ utils
│  ├─ dataset_aux.jl
│  └─ hypercubes_gen.jl
└─ runtests.jl
```

Testes de envoltórias

Os testes automatizados das envoltórias convexas, presente no arquivo `hull_tests.jl`, é baseado na construção de envoltórias em pontos que formam hipercubos de aresta crescente. Em suma, a função `hypercubes_matrix` no arquivo `utils/hypercubes_gen.jl` gera uma matriz X com pontos de D dimensões organizados como vértices de N hipercubos com aresta de tamanho 1 até N, todos centrados em (0,0).

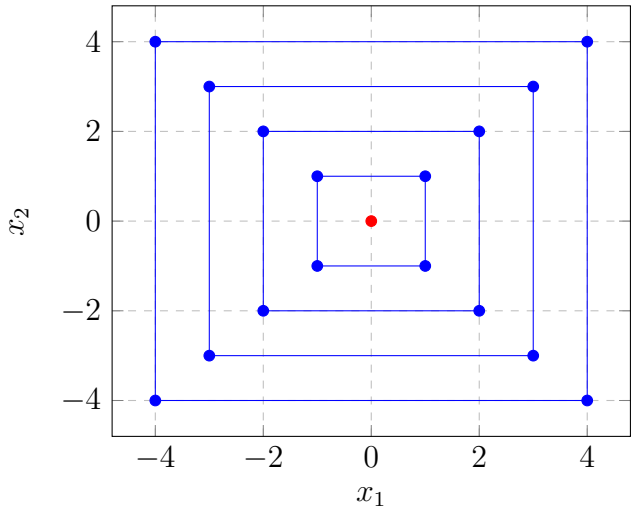


Figura 6.1: Configuração usada para testes de envoltórias (N = 4, D = 2)

Espera-se que o algoritmo que calcula envoltórias convexas encontre N envoltórias, do hipercubo de maior aresta até o menor de todos. Os testes são feitos pela geração de N = 4 hipercubos de dimensão variada D $\in [2, 10]$. Cada teste é realizado também com a inclusão da origem na matriz, com o

resultado esperado deste ponto estar contido na envoltória mais interna, porém ser marcado como um ponto que não é vértice. Este ponto pode ser visto na figura 6.1 em vermelho.

Para testar o cálculo das probabilidades associadas às envoltórias, são feitos testes da mesma maneira, com N = 4 hipercubos de dimensão variada D $\in [2, 10]$. O centro do intervalo de probabilidades é calculado a partir da quantidade de pontos dentro de cada envoltória e é comparado com a média das duas extremidades de cada intervalo.

Testes de modelos

Os testes de modelos são divididos no arquivo `classification_tests.jl`, onde estão os testes dos modelos de classificação de Hinge Loss e Logistic Loss, e no arquivo `regression_tests.jl`, onde estão os testes dos modelos de regressão de MSE e MAE.

Para cada modelo, foram escolhidos 3 datasets disponíveis no repositório da UCI. Estes datasets passam por um tratamento feito na função `utils/dataset_aux.jl` e utilizados para treinar o modelo PolieDRO e um modelo de comparação implementado no pacote MLJ. É então verificado se o modelo PolieDRO possui uma performance muito abaixo da esperada (25% abaixo do modelo de comparação). Os testes são feitos em datasets com uma performance verificada como boa, de maneira que uma falha no teste indica uma piora no modelo.

No arquivo `classification_tests.jl`, o modelo Hinge Loss é comparado com o SVM nos datasets `balloons-a`, `breast-cancer-wisconsin-diagnostic` e `iris`. O modelo Logistic Loss é comparado com um classificador logístico nos datasets `balloons-a`, `connectionist-bench` e `hayes-roth`.

No arquivo `regression_tests.jl`, os modelos MSE e MAE são ambos com uma regressão OLS nos datasets `concrete-slump-test-flow`, `lpga-2009` e `yacht-hydrodynamics`. Neste arquivo também é feito o teste do cálculo externo das envoltórias, que são calculadas apenas uma vez e usada pelos dois modelos.

6.3 Documentação

A documentação do pacote pode ser dividida em duas seções principais: a documentação interna presente no código e a página do GitHub Pages.

Documentação interna

A documentação interna do código está presente principalmente na forma de docstrings, um texto formatado colocado antes de qualquer função ou estrutura presente no pacote. Além de servir como documentação internamente, as docstrings podem ser visualizadas em ambientes de desenvolvimento como o VSCode para um acesso rápido a explicações de funcionalidades disponíveis.

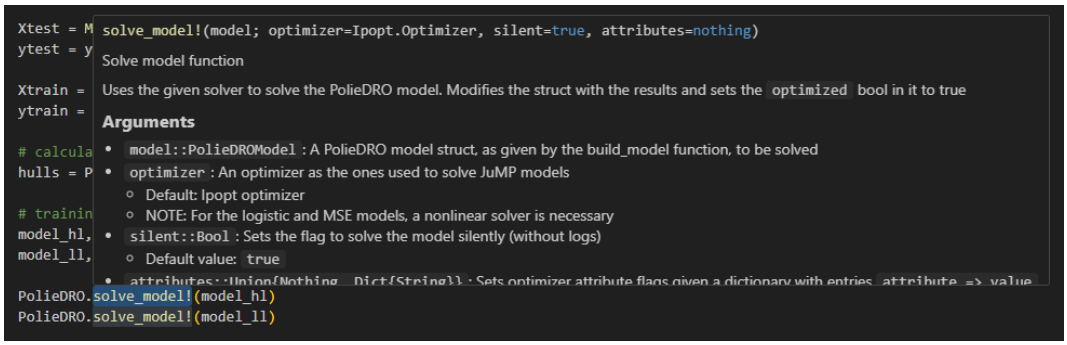


Figura 6.2: Exemplo de visualização de uma docstring por um usuário do pacote

Além das docstrings, também está presente em diversos trechos de código comentários simples, providenciando contexto para blocos de código mais complexos ou explicando soluções implementadas.

Site do GitHub Pages

Utilizando o pacote Documenter.jl, é possível criar um site em Markdown com uma documentação mais extensa do pacote, contendo códigos de exemplo, textos e definições matemáticas mais completas. A cada commit na main, ocorre uma build definida no arquivo `.github/workflows/Docs.yml` para atualizar o website da documentação.

A estrutura do site é definida no diretório docs, organizado da maneira abaixo:

```
docs
├─ src
│  ├─ index.md
│  ├─ models.md
│  └─ reference.md
└─ make.jl
```

O arquivo `make.jl` é o arquivo que define a estrutura do site utilizando o pacote Documenter.jl, definindo as páginas Markdown utilizadas, o nome do site, autor, entre outras definições gerais. Este é o arquivo executado durante o processo de build realizado pelo GitHub Actions.

O arquivo `index.md` é a página principal do site, onde está escrita uma curta descrição do pacote, uma seção 'quickstart' com um exemplo de utilização simples e as referências. É uma estrutura similar à encontrada no README visível no repositório do GitHub.

O arquivo `models.md` é uma página com uma extensa documentação do framework PolieDRO e de seus modelos. Cada um dos modelos pré-definidos possui a sua seção da página, com o seu problema de otimização detalhado e um exemplo de uso em um dataset. Além dos modelos pré-implementados, nessa página também pode ser encontrada uma explicação completa de como podem ser construídos modelos com funções de perda customizáveis.

O arquivo `reference.md` possui a referência de cada função e estrutura implementados no pacote. Esta página é criada através das docstrings presentes no código, contendo informações sobre o uso de todas as funcionalidades do pacote.

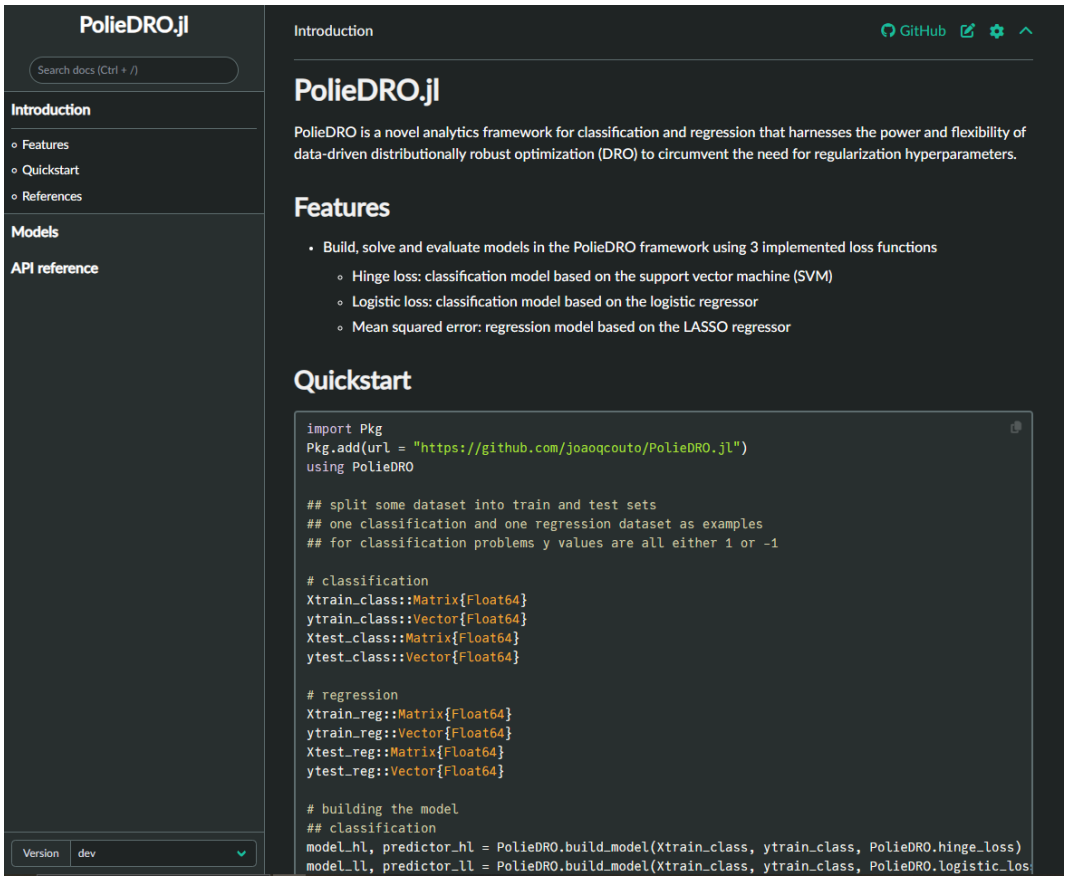


Figura 6.3: Página principal do site de documentação

7
Testes da implementação

Para testar a usabilidade do pacote, foi organizado um repositório separado somente para testar os modelos em variados datasets. Com o pacote UCIData.jl (Dunn, 2022), também utilizado para os testes automatizados dos modelos, foram feitos testes comparativos em muitas bases de dados, gerando e salvando resultados em arquivos CSV a cada execução.

Neste repositório também foram realizados testes de diferentes implementações de algoritmos de cálculo de envoltórias, cujos resultados foram apresentados em tabelas na seção 4.2.

Como métrica de comparação, novamente foram utilizados modelos estabelecidos e implementados no MLJ.jl na mesma configuração de comparação da publicação original: o modelo Hinge Loss comparado com o SVM, o modelo Logistic Loss comparado com a regressão logística e o modelo MSE comparado com a regressão linear. Foi adicionado também uma comparação do modelo MAE com a regressão quantílica.

Antes de cada teste, os dados passam pelo tratamento descrito na seção 4.4: remoção de dados faltantes e duplicados e codificação one-hot de colunas categóricas. Para os dados de classificação, foi escolhido arbitrariamente a partir do primeiro dado qual a categoria que seria considerada '1', enquanto que as restantes assumiram o valor de '-1'. Assim, todos os problemas de classificação foram treinados em um sistema um-contra-todos.

Nas páginas seguintes, é possível ver albumas tabelas comparando as performances destes modelos. Para os modelos de classificação, foi utilizado como comparação a porcentagem de previsões corretas. Para os modelos de regressão, foi utilizado como comparação o Mean Average Scaled Error (MASE) das previsões. Esta métrica é útil por ter uma escala consistente independentemente da escala dos dados, permitindo um julgamento consistente para bases diferentes. Um modelo cuja 'previsão' é a média dos valores possui MASE 1.

Dataset	PolieDRO HL	SVM
acute-inflammations-1	100.00%	100.00%
acute-inflammations-2	100.00%	100.00%
balance-scale	92.16%	92.16%
blood-transfusion-service-center	75.09%	61.95%
breast-cancer-wisconsin-original	93.55%	95.32%
climate-model-simulation-crashes	91.48%	95.37%
connectionist-bench	95.45%	97.37%
echocardiogram	68.81%	71.19%
ecoli	94.34%	96.12%
glass-identification	73.28%	75.59%
haberman-survival	72.85%	70.37%
hayes-roth	76.92%	75.38%
heart-disease-cleveland	81.51%	82.15%
heart-disease-hungarian	79.69%	82.76%
heart-disease-switzerland	63.81%	61.90%
heart-disease-va	77.24%	78.01%
hepatitis	83.03%	82.95%
horse-colic	74.86%	57.79%
image-segmentation	98.10%	98.10%
indian-liver-patient	69.97%	71.21%
iris	99.35%	100.00%
mammographic-mass	75.64%	81.54%
monks-problems-1	80.00%	80.00%
monks-problems-2	54.00%	63.00%
monks-problems-3	78.00%	75.00%
planning-relax	68.14%	71.56%
post-operative-patient	63.52%	66.29%
seeds	94.76%	89.05%
thyroid-disease-new-thyroid	86.51%	90.23%
tic-tac-toe-endgame	97.71%	98.33%
wine	97.22%	98.33%
zoo	100.00%	100.00%
MÉDIA	**83.03%**	**83.09%**

Tabela 7.1: Comparação dos modelos PolieDRO Hinge Loss e SVM

Dataset	PolieDRO LL	Logistic Reg
acute-inflammations-1	100.00%	100.00%
acute-inflammations-2	100.00%	100.00%
balance-scale	92.16%	92.16%
blood-transfusion-service-center	75.29%	75.88%
breast-cancer-wisconsin-original	93.55%	94.66%
climate-model-simulation-crashes	91.48%	95.56%
connectionist-bench	95.45%	96.87%
echocardiogram	68.81%	71.43%
ecoli	94.34%	96.42%
glass-identification	74.21%	74.66%
haberman-survival	73.21%	73.19%
hayes-roth	76.92%	72.53%
heart-disease-cleveland	81.51%	84.82%
heart-disease-hungarian	81.23%	83.14%
heart-disease-switzerland	64.76%	64.76%
heart-disease-va	77.24%	78.01%
hepatitis	83.03%	82.95%
horse-colic	69.41%	56.88%
image-segmentation	99.05%	99.05%
indian-liver-patient	69.97%	71.03%
iris	100.00%	100.00%
mammographic-mass	75.64%	78.72%
monks-problems-1	80.00%	80.00%
monks-problems-2	54.00%	54.00%
monks-problems-3	78.00%	82.00%
planning-relax	67.00%	68.71%
post-operative-patient	63.52%	57.81%
seeds	94.76%	93.81%
thyroid-disease-new-thyroid	86.51%	90.70%
tic-tac-toe-endgame	97.71%	98.33%
wine	98.27%	93.24%
zoo	98.18%	96.52%
MÉDIA	**82.98%**	**83.06%**

Tabela 7.2: Comparação dos modelos PolieDRO Logistic Loss e regressão logística

Dataset	PolieDRO MSE	LinReg
airline-costs	0.1273	0.1084
auto-mpg	0.4899	0.3975
concrete-slump-test-compressive	0.4298	0.3699
concrete-slump-test-flow	0.8847	0.9110
concrete-slump-test-slump	1.2200	1.1464
construction-maintenance	0.8192	0.5656
housing	0.9056	0.5112
hybrid-price	0.6875	0.7017
immigrant-salaries	0.8140	0.6340
lpga-2008	0.4036	0.3892
lpga-2009	0.3768	0.3363
yacht-hydrodynamics	0.7475	0.6656
MÉDIA	**0.6588**	**0.5614**

Tabela 7.3: Comparação dos modelos PolieDRO MSE e regressão linear

Dataset	PolieDRO MAE	QuantReg
airline-costs	0.126	1.258
auto-mpg	0.491	0.762
concrete-slump-test-compressive	0.430	0.402
concrete-slump-test-flow	0.885	1.016
concrete-slump-test-slump	1.220	1.131
construction-maintenance	0.813	0.694
housing	0.906	0.838
hybrid-price	0.896	0.938
immigrant-salaries	0.814	0.583
lpga-2008	0.407	0.600
lpga-2009	0.377	0.363
yacht-hydrodynamics	0.744	0.833
MÉDIA	**0.6755**	**0.7848**

Tabela 7.4: Comparação dos modelos PolieDRO MAE e regressão quantílica

Em geral, os modelos PolieDRO obtiveram uma performance muito próxima dos modelos tradicionais. O PolieDRO MSE requer mais atenção devido a sua performance, enquanto que o PolieDRO MAE obteve resultados melhores que a regressão quantílica, mas ainda assim inferior aos outros dois modelos de regressão. Nos problemas de classificação, a acurácia dos modelos tradicionais e dos modelos PolieDRO foi extremamente próxima.

Apesar das melhoras no algoritmo de cálculo de envoltórias descritas na seção 4.2, ainda nota-se que este é o maior fator determinante da performance dos modelos. Para bases de dados maiores o tempo de cálculo das envoltórias ainda é significativo, o que dificulta o uso do framework em relação aos modelos tradicionais.

8
Considerações finais

Neste trabalho foi produzida uma biblioteca de código aberto em Julia para a implementação de um novo framework de aprendizado de máquina para classificação e regressão, o PolieDRO. Além da implementação dos modelos originalmente descritos, foram implementados métodos que permitem a construção de modelos com funções de perda customizáveis, assim como um novo modelo de regressão. O estudo de métodos de envoltórias convexas também levou a uma solução mais eficiente do que o algoritmo originalmente utilizado.

Ao se testar a biblioteca em bases de dados, foi possível verificar a sua facilidade de uso, além de serem criados caminhos mais eficientes para casos específicos como o pré-cálculo das envoltórias convexas para casos em que vários modelos serão treinados. Do lado do cálculo das envoltórias convexas, por mais que avanços tenham sido feitos em relação ao método original, os testes feitos mostram que ainda existem casos em que algoritmos diferentes mostram performances promissoras. Pode ser interessante a busca por um novo método ou uma heurística para selecionar o melhor dentre diferentes métodos ao depender dos dados.

Algumas extensões já imaginadas para o projeto incluem a implementação de uma interface compatível com o MLJ, o framework mais popular para machine learning em Julia. Além disso, novas funções de perda podem se mostrar eficientes a um nível que seja justificado implementá-las nativamente na biblioteca.

Outras possibilidades de extensão incluem a utilização dos métodos para regressão do PolieDRO em modelos autoregressivos para previsão de séries temporais, além de uma extensão do framework original para problemas de alocação de portfólio. Estes novos estudos podem então levar a extensões da biblioteca para estes casos de uso.

Referências bibliográficas

[Barber et al., 1993] BARBER, C.; DOBKIN, D. ; HUHDANPAA, H.. **The quickhull algorithm for convex hulls**. ACM Transactions on Mathematical Software - TOMS, 22, 01 1993.

[Bertsimas et al., 2011] BERTSIMAS, D.; BROWN, D. B. ; CARAMANIS, C.. **Theory and applications of robust optimization.** SIAM Rev., 53(3):464–501, 2011.

[Bezanson et al., 2017] BEZANSON, J.; EDELMAN, A.; KARPINSKI, S. ; SHAH, V. B.. **Julia: A fresh approach to numerical computing**. SIAM review, 59(1):65–98, 2017.

[Birge, Louveaux, 2011] BIRGE, J. R.; LOUVEAUX, F.. **Introduction to stochastic programming**. Springer Science & Business Media, 2011.

[Blanchet et al., 2019] BLANCHET, J.; KANG, Y. ; MURTHY, K.. **Robust wasserstein profile inference and applications to machine learning**. Journal of Applied Probability, 56, 10 2019.

[Blaom et al., 2020] BLAOM, A. D.; KIRALY, F.; LIENART, T.; SIMILLIDES, Y.; ARENAS, D. ; VOLLMER, S. J.. **MLJ: A julia package for composable machine learning**. Journal of Open Source Software, 5(55):2704, 2020.

[Dua, Graff, 2017] DUA, D.; GRAFF, C.. **Uci machine learning repository**, 2017.

[Dunn, 2022] UCIDATA.JL CONTRIBUTORS. **Ucidata.jl v1.0**, 2022. [Online; acessado em junho de 2024].

[Fernandes et al., 2016] FERNANDES, B.; STREET, A.; VALLADÃO, D. ; FERNANDES, C.. **An adaptive robust portfolio optimization model with loss constraints based on data-driven polyhedral uncertainty sets**. European Journal of Operational Research, 255(3):961–970, 2016.

[Forets, Schilling, 2021] FORETS, M.; SCHILLING, C.. **LazySets.jl: Scalable Symbolic-Numeric Set Computations**. Proceedings of the JuliaCon Conferences, 1(1):11, 2021.

[Gareth et al., 2023] JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. ; TAYLOR, J.. **An Introduction to Statistical Learning**. Springer, New York, 2023.

[Goh e Sim, 2010] GOH, J.; SIM, M.. **Distributionally robust optimization and its tractable approximations.** Oper. Res., 58(4-Part-1):902–917, 2010.

[Gutierrez et al., 2024] GUTIERREZ, T.; VALLADÃO, D. ; PAGNONCELLI, B.. **Poliedro: a novel classification and regression framework with non-parametric data-driven regularization**. Machine Learning, p. 1–40, 04 2024.

[Kuhn, Max, 2008] KUHN; MAX. **Building predictive models in r using the caret package**. Journal of Statistical Software, 28(5):1–26, 2008.

[Lubin et al., 2023] LUBIN, M.; DOWSON, O.; DIAS GARCIA, J.; HUCHETTE, J.; LEGAT, B. ; VIELMA, J. P.. **JuMP 1.0: Recent improvements to a modeling language for mathematical optimization**. Mathematical Programming Computation, 2023.

[Pardalos et al., 1995] PARDALOS, P. M.; LI, Y. ; HAGER, W.. **Linear programming approaches to the convex hull problem in rm**. Computers & Mathematics with Applications, 29(7):23–29, 1995.

[Pedregosa et al., 2011] PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V. ; OTHERS. **Scikit-learn: Machine learning in python**. Journal of machine learning research, 12(Oct):2825–2830, 2011.

[Wiesemann et al., 2014] WIESEMANN, W.; KUHN, D. ; SIM, M.. **Distributionally robust convex optimization**. Operations research, 62(6):1358–1376, 2014.

[equipe Julia CI, 2020] PKGTEMPLATES.JL CONTRIBUTORS. **Pkgtemplates.jl v0.7**, 2020. [Online; acessado em maio de 2024].

[equipe Julia Docs, 2016] DOCUMENTER.JL CONTRIBUTORS. **Documenter.jl v1.6 – a documentation generator for julia**, 2024. [Online; acessado em agosto de 2024].