BarySearch: Algoritmo de *tuning* de Modelos de *Machine Learning* com o Método do Baricentro

Lucas Francisco Amaral Orosco Pellicer[1], **Felipe Miguel Pait**[1]

[1]Escola Politécnica - Universidade de São Paulo (USP)
São Paulo – SP – Brasil

{lucas.pellicer,pait}@usp.br

Abstract. *In many Machine Learning applications, it is desirable to obtain the best set of hyperparameters to optimize the performance of the application. The problem of optimizing hyperparameters is known as tuning of Machine Learning models. Despite being an optimization problem, tuning faces complex difficulties, since the models are seen as black boxes without well-defined mathematical formulation. Also, there are problems with regions of oscillations and regions of vast plateaus. In this work, we present BarySearch, an algorithm that uses the equation of the barycenter without the need to calculate derivatives of the objective function. The BarySearch technique has shown promising results in practical tests of model tuning.*

Resumo. *Em muitas aplicações de Machine Learning, é desejável obter o melhor conjunto de hiperparâmetros para otimizar o desempenho da aplicação. O problema de otimizar os hiperparâmetros é conhecido como tuning de modelos Machine Learning. Apesar de ser um problema de otimização, o tuning enfrenta dificuldades complexas, já que os modelos são vistos como caixas pretas sem formulação matemática bem definida. Além disso, há problemas com regiões de oscilações e regiões de grandes platôs. Nesse trabalho, nós apresentamos o BarySearch, um algoritmo que se utiliza da equação do baricentro sem necessidade de calcular derivadas da função objetivo. A técnica BarySearch demonstrou ter resultados promissores em testes práticos de tuning de modelos.*

1. Introdução

Muitos desafios atuais podem ser formulados como problemas de otimização complexos. O objetivo desses problemas é encontrar o melhor conjunto de parâmetros x que otimizem uma função objetivo $f(x)$. Essa função objetivo pode apresentar características que compliquem a otimização como não convexidade, não diferenciabilidade ou mesmo não continuidade [Koch et al. 2018]. Em outras situações, não é possível mesmo obter uma formulação matemática de $f(x)$ que é vista como uma caixa preta. Em situações complexas como estas descritas, é recomendado usar métodos de otimização que não necessitam de cálculos de derivadas ou gradientes de $f(x)$, conhecidos como métodos livres de derivação [Rios and Sahinidis 2013].

Em certas aplicações de *Machine Learning*, é um requisito alcançar o melhor desempenho possível. Essa exigência muitas vezes implica na necessidade de encontrar o melhor conjunto de hiperparâmetros nos modelos de *Machine Learning*. Esse problema

é conhecido como *tuning* de modelos de *Machine Learning*. O *tuning* apresenta as dificuldades já citadas de não ter funções objetivos bem definidas, mas também apresenta regiões de grande oscilação, regiões de platôs e também precisa otimizar parâmetros categóricos, contínuos, discretos e até mesmo condicionais, muitas vezes simultaneamente [Koch et al. 2018].

A primeira abordagem de resolução de *tuning* é a busca exaustiva, chamada de *Grid Search*. Essa técnica consiste em testar diferentes combinações de hiperparâmetros exaustivamente e separar a melhor. Apesar de muito simples, essa técnica apresenta problemas de dimensionalidade, pois com a inserção de um parâmetro na otimização causa um aumento exponencial de combinações a serem testadas.

Outra abordagem largamente usada é a busca aleatória, o *Random Search*. Nesse método, são testadas combinações de valores de hiperparâmetros amostradas aleatoriamente em um espaço de busca. Embora a técnica não apresentar grandes diferenças, [Bergstra and Bengio 2012] demonstrou que o *Random Search* converge em pontos melhores que a busca exaustiva na esperança.

Outras técnicas foram desenvolvidas para acelerar o processo de convergência. Algumas dessas técnicas são métodos evolucionários. Os métodos evolucionários são técnicas com inspirações em fenômenos naturais ou biológicos que podem ser descritos como métodos de seleção de melhores resultados em uma população. Exemplos de técnicas evolucionárias são Simulated Annealing [Kirkpatrick et al. 1983], Particle Swarm Optimization [Eberhart and Kennedy 1995], CMAES [Hansen and Ostermeier 2001] e algoritmos genéticos.

Outra categoria de algoritmos desenvolvidos são buscas bayesianas. As buscas bayesianas são métodos de busca apoiados na teoria de probabilidade condicional clássica de Bayes. Essas técnicas calculam curvas conhecidas como *surrogates* que calculam a probabilidade de um ponto ter um resultado promissor na função objetivo $f(x)$. Algumas dessa técnicas são processos Gaussianos [Snoek et al. 2012], Tree Parzen Estimators (TPE) [Bergstra et al. 2015] e Sequential Model-based Algorithm Configuration (SMAC) [Hutter et al. 2011].

Recentemente, foram desenvolvidos algoritmos considerados estado da arte na resolução de *tuning*. Podem ser destacados como estado da arte desses *frameworks*: TPOT [Le et al. 2020] que consiste na contrução de *pipelines* através de algoritmos genéticos; Auto-Sklearn [Feurer et al. 2015] que é uma combinação entre o SMAC de busca bayesiana com técnicas de *meta-learning* e *ensemble* (combinação) de diferentes *pipelines*; e o Hyperband [Li et al. 2017] que não é baseado em técnicas bayesianas ou evolucionárias, mas usa algoritmos de busca como Sucessive Halving [Jamieson and Talwalkar 2015] e técnicas competitivas.

Nesse trabalho, é proposto o BarySearch que é uma técnica que se aproveita das propriedades da equação do baricentro [Pait 2018] de ser livre de derivação e ter convergência garantida em algumas situações. O BarySearch guarda algumas semelhanças das técnicas evolucionárias, conforme será explicado a seguir.

2. Algoritmo

2.1. Método do Baricentro

O método do baricentro é um algoritmo recursivo que utiliza do histórico dos valores da função objetivo $f(x)$ e os valores de x [Pait 2018]. O método do baricentro não necessita de conhecimento prévio da função $f(x)$ para ser aplicado e nem necessita do cálculo de derivadas de $f(x)$, podendo ser aplicado em funções não convexas, não diferenciaveis ou mesmo não contínuas. A fórmula do baricentro é uma ponderação dos resultados de $f(x)$ e os valores de x dada pela Equação 1 .

$$\hat{x}_n = \frac{\sum_{i=1}^{n} x_i e^{-\nu f(x_i)}}{\sum_{i=1}^{n} e^{-\nu f(x_i)}}, \tag{1}$$

Contudo, além da fórmula do baricentro, o método introduz uma perturbação aleatória para realização da exploração do espaço de busca z_n, conforme Equação a seguir.

$$x_n = \hat{x}_{n-1} + z_n. \tag{2}$$

A distribuição de z_n pode ser escolhida aleatoriamente, porém o trabalho [Pait 2018] demonstra que z_n ter distribuição gaussiana, o baricentro pode ter direção que converge na esperança ao gradiente da função $f(x)$. Dessa forma, o método do baricentro pode combinar características de gradiente descendente com variáveis aleatórias que introduzem um comportamento mais exploratório.

2.2. BarySearch

O Algoritmo Barysearch utiliza da Equação 1 do baricentro para a realização da otimização dos parâmetros. Para que o baricentro consiga convergir mais rapidamente, é importante uma boa inicialização do algoritmo. A inicialização do algoritmo pode ser realizada por uma amostragem aleatória de alguns pontos no espaço de busca que costuma apresentar bons resultados.

Contudo, para o BarySearch, o trabalho propõe o uso da amostragem Latin Hyperplane Sample (LHS) [McKay 1992]. Essa amostragem é caracterizada por selecionar pontos aleatoriamente, mas os pontos precisam passar por regiões do espaço de busca. A amostragem inicia dividindo o espaço de busca em hiperplanos e os pontos são selecionados de forma a respeitar o Latin Square que em duas dimensões significa que precisa ter pelo menos uma amostra por linha e por coluna. Para mais dimensões, pode ser generalizado o conceito de linha e coluna para que sejam selecionados obrigatoriamente pontos em cada subdivisão. A principal vantagem dessa amostragem é que ela pode fazer um mapeamento melhor do espaço de busca e assim o baricentro já se inicia de forma a ponderar melhor os pontos iniciais.

Outro parâmetro de importância é o fator de velocidade de convergência ν. Esse fator pode modificar o comportamento do algoritmo do BarySearch. Para ν em módulo alto, significa ter um algoritmo mais convergente e que dê mais peso para os melhores resultados. Com módulos menores de ν, o algoritmo fica com um comportamento mais ponderado e terá uma distribuição de peso mais uniforme entre os resultados anteriores. O ν também determina o tipo de operação: caso $\nu > 0$, deseja maximizar a função objetivo

$f(x)$, caso contrário $\nu < 0$, é realizada uma minimização. Em algumas situações, o alcance da função de $f(x)$ pode ser um problema para o cálculo da exponencial, pois pode chegar a valores cujos exponenciais resultam em erros computacionais. Nesses casos, é recomendado que se faça uma padronização de $f(x)$ para a escala entre 0 a 1.

Por fim, o método do baricentro utiliza de uma perturbação aleatória z_n para exploração da busca. No BarySearch, é sugerido BarySearch com essa pertubação aleatória com distribuição normal com média igual a 0 e com desvio padrão σ que é nomeado de fator de curiosidade. O fator de curiosidade também apresenta o poder de adaptar o comportamento do algoritmo: com σ menor, o algoritmo fica mais determinístico, caso contrário o comportamento fica mais aleatório ou exploratório.

Dessa forma, o conjunto ν e σ são essenciais para a aplicação. Com ν alto e σ baixo, o algoritmo é convergente e determinístico e apresenta uma característica muito semelhante ao método do gradiente. Esse tipo de comportamento pode ser mais interessante em funções unimodais bem comportadas. Já com ν baixo e σ alto, o algoritmo fica com comportamento muito mais exploratório e mais semelhante a métodos evolucionários, como Particle Swarm [Eberhart and Kennedy 1995] ou método Gravitacional. Esse tipo de comportamento é mais indicado para funções multimodais, não bem comportadas e que são mais difíceis de otimizar. Para problemas de *tuning* com métricas que variam entre 0 a 1 (como AUC ou taxa de erro), é indicado usar valores $|\nu| = 50$ e $\sigma = 0.5$, ou seja, um algoritmo mais próximo ao evolucionário e mais comportamento aleatório.

3. Testes e Resultados

Nesta seção estão testes e resultados do BarySearch em aplicações de *tuning* de modelos de *Machine Learning*. Cada subseção mostra um teste de algum tipo de *tuning* e seus resultados.

3.1. Testes de Benchmark de Classificação

Os primeiros testes são de *tuning* de modelos *Machine Learning* em aplicações clássicas de classificação. Para esse teste, foram selecionadas bases de benchmark de UCI [Dua and Graff 2017]. Cada base foi dividida em 70% para treino e 30% para teste. Nas bases de treino, é realizado o *tuning* de modelos.

O *tuning* dos modelos é feito através de uma validação cruzada com 5 repartições. A métrica de desempenho adotada é a taxa de erro de classificação que é a mais utilizada em *tuning* de classificação [Koch et al. 2018]. O conjunto dos valores de hiperparâmetros dos modelos com menores erro na validação cruzada é testado na base de treino com o intuito de verificar o poder de generalização dos modelos treinados.

Para o *tuning*, foram selecionados dois modelos clássicos de *Machine Learning* que apresentam um conjunto grande de hiperparâmetros: Rede Neural (Multi-layer Perceptron) e LightGBM (boosting). Em ambos os modelos, foram selecionados 7 hiperparâmetros numéricos para o *tuning*. Foram apenas selecionados hiperparâmetros numéricos, pois alguns algoritmos de *tuning* apresentam restrição com parâmetros categóricos, como o próprio BarySearch na atual versão. A Tabela 1 apresenta o espaço de busca desses parâmetros.

Tabela 1. Parâmetros do *tuning*.

Modelo	Parâmetros	Espaço de Busca
LightGBM	Número de folhas	30 a 150
	Número de estimadores	100 a 1000
	Learning Rate	0.01 a 0.2
	Mínimo de amostra nos filhos	20 a 500
	Regularização α	0 a 1
	Regularização λ	0 a 1
	Colsample	0.6 a 1
MLP	Regularização α	2 a 100
	Learning Rate Inicial	0.001 a 0.01
	β_1 do otimizador ADAM	0.6 a 0.99
	β_1 do otimizador ADAM	0.85 a 0.999
	ϵ do otimizador ADAM	10^{-10} a 10^{-5}
	Número de neurônios na primeira camada	2 a 20
	Número de neurônios na segunda camada	2 a 20

Os algoritmos testados foram os clássicos: Random Search (busca aleatória), Simulated Annealing [Kirkpatrick et al. 1983], CMA-ES [Hansen and Ostermeier 2001], TPE do HyperOpt [Bergstra et al. 2015], além dos estado da arte TPOT [Le et al. 2020] e Hyperband [Li et al. 2017], todos comparados ao BarySearch. Todos os algoritmos foram testados com 25 iterações de *tuning* e o processo de *tuning* de cada algoritmo foi repetido 30 vezes com diferentes inicializações. A Tabela 2 mostra os resultados.

Os resultados demonstram que o BarySearch consegue ter melhor desempenho em boa parte dos testes, quando não, provavelmente está na segunda posição. Isso demonstra que o BarySearch tem um desempenho comparável com os algoritmos considerados estado da arte.

3.2. *Deep Learning* e Dados não Estruturados

Algumas aplicações trabalham com dados não estruturados (imagens, texto, áudios, vídeos e entre outros). Muitas dessas aplicações utilizam modelos *Deep Learning* que são Redes Neurais com arquiteturas diferenciadas das totalmente conectadas MLP. Para muitas aplicações com imagens, os melhores modelos são de redes neurais convolucionais (CNN) [LeCun et al. 2010].

As CNN apresentam um grande número de hiperparâmetros a serem definidos: quantidade de neurônios, taxa de dropout, tamanho de kernel, tamanho de camada pooling, learning rate e muito outros parâmetros. O conjunto desses parâmetros definem a arquitetura da rede e tem alto impacto na performance do modelo. Como treinar esses modelos são muitos custosos, o *tuning* desses modelos precisa ter bons resultados com menor número de iterações.

Para teste de *tuning* com esses modelos complexos, os testes utilizaram banco de dados famosos de imagens: MNIST, Fashion MNIST e o CIFAR-10, todos disponíveis em [Chollet et al. 2015]. Em todos os testes, foi definida uma CNN com 2 camadas convolucionais, uma camada de pooling e mais 2 camadas densas e os parâmetros a serem otimizados são quantidade de neurônio em cada camada, tamanho de kernel das camadas

Tabela 2. Média da taxa de erro de classificação. Os valores estão marcados com •ou $*$se a média for a melhor ou a secunda melhor respectivamente.

Base de Dados	Algoritmo	LGBM $t = 25$	MLP $t = 25$
Ionsphere	Random Search	9,0%	16,2%
	Simulated Annealing	19,3%	16,9%
	TPE	8,5% $*$	12,1%
	CMA-ES	21,1%	20,9%
	Hyperband	12,1%	15,2%
	TPOT	13,0%	12,1% $*$
	BarySearch	7,8% •	11,6% •
Credit	Random Search	9,7% $*$	29,9%
	Simulated Annealing	11,7%	29,0%
	TPE	9,9%	29,6%
	CMA-ES	12,9%	29,8%
	Hyperband	11,2%	25,3% $*$
	TPOT	12,1%	23,5% •
	BarySearch	9,4% •	27,9%
SPAM	Random Search	4,6%	17,2%
	Simulated Annealing	4,6%	17,4%
	TPE	4,6% $*$	15,3%
	CMA-ES	4,6% $*$	15,5%
	Hyperband	4,8%	11,0% •
	TPOT	4,5% •	15,1%
	BarySearch	4,5% •	13,6% $*$
Breast-Cancer	Random Search	0,6%	9,8%
	Simulated Annealing	10,0%	10,3%
	TPE	0,1% $*$	9,0%
	CMA-ES	12,4%	8,7%
	Hyperband	0,6%	3,0% •
	TPOT	0,4%	11,4%
	BarySearch	0,0% •	8,3% $*$
Dermatology	Random Search	5,9%	9,5%
	Simulated Annealing	30,9%	32,8%
	TPE	5,3% •	8,8% $*$
	CMA-ES	37,8%	28,4%
	Hyperband	8,2%	10,1%
	TPOT	6,7%	9,7%
	BarySearch	5,6% $*$	8,3% •

convolucionais, taxa de dropout em cada camada, além de parâmetros do otimizador de treino ADAM, totalizando em 17 parâmetros a serem sintonizados no *tuning*.

Nos testes, foram separadas 42.000 imagens para treino, 18.000 para a validação do modelo e mais 10.000 imagens para teste em cada uma das três bases. Cada algoritmo foi compilado com 10 iterações nesses testes e cada teste foi repetido por 10 vezes com inicializações diferentes. Novamente, foi utilizada a sugestão de parametros $\sigma = 0.5$ e $\nu = -50$ no BarySearch. Os resultados estão dispostos na Tabela 3. Não foi possível realizar testes com o CMA-ES e TPOT nesses algoritmos, pois as implementações não aceitavam modelos CNN para otimização.

Tabela 3. Taxa de Erro Médio de Classificação das Imagens.

Algoritmo	MNIST	Fashion-MNIST	CIFAR10
TPE	1.02%	9.76%	42.16%
Simulated Annealing	0.97%	9.11%	40.70%
Random Search	0.93%	9.07%	38.57%
BarySearch	0.90%	8.85%	36.33%

Os resultados demonstram que o BarySearch foi o algoritmo com melhor desempenho nos três banco de dados de imagens testados. Verifique que nesse caso algoritmos mais complexos não apresentam bom desempenho quanto o Random Search, com exceção do BarySearch. Isso demonstra o quanto o BarySearch pode ser flexível e funcionar bem tanto em problemas de *tuning* clássico, como os mais complexos.

4. Conclusões

Otimizar o desempenho de modelos de aprendizado de máquina podem ser complexos, pois os modelos são vistos como caixas- pretas por não existir uma relação matemática definida entre os hiperparâmetros e o resultado do modelo. Muitas técnicas foram propostas para resolver o problema de otimização de modelos (*tuning*). Esse texto sugere a técnica do baricentro que já demonstrou sucesso no contexto de controle de sistemas para *tuning* desses modelos.

Testes foram realizados com o BarySearch e o comparando com demais algoritmos, alguns considerados estado da arte. Os resultados foram promissores, já que o método apresentou resultado comparável aos métodos estado da arte e também apresentou flexibilidade por funcionar bem tanto em aplicações tradicionais com *Machine Learning* e como modelos complexos de *Deep Learning*.

Para trabalhos futuros, será interessante fazer uma análise mais profunda de possíveis melhores abordagens com parâmetros categóricos que hoje é uma limitação do BarySearch, bem como disponibilizar o BarySearch em um pacote nos moldes do TPOT e Auto-Sklearn.

Referências

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305.

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., and Cox, D. (2015). Hyperopt: A python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8:014008.

Chollet, F. et al. (2015). Keras. https://keras.io.

Dua, D. and Graff, C. (2017). UCI machine learning repository. http://archive.ics.uci.edu/ml.

Eberhart, R. and Kennedy, J. S. (1995). A new optimizer using particle swarm theory. *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9(2):159–195.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *LION*.

Jamieson, K. G. and Talwalkar, A. (2015). Non-stochastic best arm identification and hyperparameter optimization. In *AISTATS*.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220 4598:671–80.

Koch, P., Golovidov, O., Gardner, S., Wujek, B., Griffin, J., and Xu, Y. (2018). Autotune: A derivative-free optimization framework for hyperparameter tuning. *KDD*, pages 443–452. ISBN: 978-1-4503-4887-4.

Le, T. T., Fu, W., and Moore, J. H. (2020). Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*, 36(1):250–256.

LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010). Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256. IEEE.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18(1):6765–6816.

McKay, M. D. (1992). Latin hypercube sampling as a tool in uncertainty analysis of computer models. In *Proceedings of the 24th Conference on Winter Simulation*, WSC '92, pages 557–564, New York, NY, USA. ACM.

Pait, F. M. (2018). The Barycenter Method for Direct Optimization. *ArXiv e-prints*.

Rios, L. M. and Sahinidis, N. V. (2013). Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc.