Matrizes esparsas e aprendizado semi-supervisionado

Emílio Bergamim,
Prof. Dr. Fabricio Breve
Instituto de Geociências e Ciências Exatas
Rio Claro, Brasil
emilio.bergamim at unesp.br

***Resumo*—Algoritmos de aprendizado semi-supervisionado dependem de um produto de matrizes, o que é computacionalmente custoso. No entanto, uma das duas matrizes envolvidas corresponde a um grafo, permitindo uma implementação esparsa que traz benefícios de armazenamento e tempo de execução. Focando em desempenho, algoritmos em C++ foram desenvolvidos e, com auxílio de Cython trazidos para a linguagem Python, assegurando maior usabilidade aos mesmos.**

Área:"Inteligência Computacional".

I. Introdução

No aprendizado de máquina, grande parte dos trabalhos tem como atividade fim a classificação de dados, na qual a cada instância de um conjunto é atribuído um de q rótulos possíveis.

Em geral, o paradigma do aprendizado supervisionado é empregado nessa tarefa. Neste, a partir de um conjunto de N instâncias em um espaço $d-$dimensional representadas por $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ e de um conjunto de rótulos $\boldsymbol{y} \in \mathbb{N}^N$ é aprendido um modelo que permite atribuir novos rótulos [1].

Na prática, no entanto, é difícil obter uma grande quantidade de dados rotulados, o que pode inviabilizar o treinamento deste tipo de modelo. Frente a este desafio, o aprendizado semi-supervisionado emergiu como uma alternativa, utilizando-se de dados rotulados e não-rotulados para aprender uma rotulação apropriada. Neste paradigma, apenas uma fração dos rótulos $\boldsymbol{y}$ é previamente conhecida, e o modelo então utiliza-se da informação estrutural presente em $\boldsymbol{X}$ para aprender uma devida rotulação [1], [2].

Uma forma de descrever a dita informação estrutural é através de um grafo G (não-direcionado) no qual cada nó representa uma instância do conjunto original e as arestas, ponderadas, descrevem a similaridade entre pares de instâncias. A estrutura deste grafo pode ser previamente dada ou construída a partir $\boldsymbol{X}$ [3].

Uma fração considerável desses algoritmos se dá através de uma dinâmica de propagação que está associada à minimização de uma função custo. Identificando este nicho, partiu-se para a elaboração de uma biblioteca em Python que permitisse uma fácil implementação deste tipo de algoritmo mantendo um bom desempenho computacional.

XIII Workshop do Programa de Pós-Graduação em Ciência da Computação: Unesp, Rio Claro, 26 e 27 de novembro de 2024.

"O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001"

II. Conceitos e Técnicas

Seja $\boldsymbol{W}$ a matriz de similaridade associada a G, na qual $W_{i,j}$ descreve a similaridade entre as instâncias i e j do conjunto original. Os rótulos são então descritos por uma matriz $\boldsymbol{\Phi} \in \mathbb{R}^{N \times q}$ e estes são obtidos a partir de uma dinâmica da forma

$$\boldsymbol{\Phi}^{t+1} = \boldsymbol{\sigma}(\boldsymbol{W}\boldsymbol{\Phi}^t + \boldsymbol{\theta}), \tag{1}$$

onde

$$\theta_{i,s} = \begin{cases} 1, & \text{se } i \text{ foi rotulado como } s, \\ 0, & \text{caso contrário.} \end{cases} \tag{2}$$

descreve a informação prévia sobre os rótulos [4], [5] e $\boldsymbol{\sigma}$ é uma função de ativação. Após convergência, as instâncias são rotuladas como

$$y_i = arg\max_s \Phi_{i,s}. \tag{3}$$

Como exemplos, o modelo *Local and Global Consistency* (LGC) [4] utiliza $\boldsymbol{\sigma}$ como a transformação identidade e $f(a,b) = g(a,b) = (a-b)^2$. Outros trabalhos usam-se também de funções de ativação não-lineares, como a normalização de linhas [6] e a função *softmax* [5].

Todas essas abordagens enquadram-se naquilo que é conhecido como propagação de rótulos. No entanto, destaca-se que não são a única forma de classificação transdutiva semi-supervisionada. Outros métodos podem basear-se na competição e cooperação entre partículas [7] ou no aprendizado de uma transformação do espaço de atributos para o espaço de rótulos [8].

A questão computacional central é então garantir que o produto na expressão 1 seja eficiente, o que será discutido a seguir.

III. Metodologia de Desenvolvimento

Recordando que $\boldsymbol{W} \in \mathbb{R}^{N \times N}$, o produto $\boldsymbol{W}\boldsymbol{\Phi}_t$ terá complexidade $O(N^2 q)$. Porém, sabe-se que grafos completos não são apropriados para tarefas de classificação, já que o excesso de arestas leva à perda de contraste entre classes distintas. Assim, estudos empíricos mostram que $\boldsymbol{W}$ deve ser uma matriz esparsa, contendo em torno de $O(N \log N)$ entradas não-nulas [3], [5].

Para além disso, como o grafo em questão é não-direcionado, $\boldsymbol{W}$ é simétrica, o que também pode ser explorado para reduzir a complexidade de armazenamento desta matriz.

Frente a estas características, implementou-se uma classe para fazer uso da mesma. Nesta, um grafo é uma lista de arestas, as quais são triplas $(i, j, W_{i,j})$.

Considerando $\boldsymbol{P} = \boldsymbol{W}\boldsymbol{\Phi}$, veja que

$$P_{i,s} = \sum_{j \neq i} W_{i,j} \Phi_{j,s} \tag{4}$$

No entanto, como $\boldsymbol{W}$ é esparsa, na expressão anterior as entradas nulas de $\boldsymbol{W}$ não contribuirão para o produto. Utilizando ainda a propriedade de simetria da matriz, $(i, j, W_{i,j})$ representa duas entradas não-nulas, de forma que para cada tripla destas, atualizam-se $P_{i,s}$ e $P_{j,s}$:

$$\begin{cases} P_{i,s} \leftarrow P_{i,s} + W_{i,j} P_{j,s} \\ P_{j,s} \leftarrow P_{j,s} + W_{i,j} P_{i,s} \end{cases} \tag{5}$$

Sendo m o número de entradas não nulas de $\boldsymbol{W}$, essa operação tem então complexidade $O(ms)$ e, idealmente, $m << N^2$.

A. Três níveis de implementação

Como os algoritmos semi-supervisionados demandam operações custosas entre matrizes, faz sentido implementá-las em uma linguagem compilada e de baixo nível como C++ de forma a extrair o máximo de performance computacional.

No entanto, a programabilidade deste tipo linguagem não é das mais amigáveis, em particular em um cenário no qual a linguagem Python é dominante do aprendizado de máquina e outras aplicações.

Cython [9] provê então uma forma de escrever código em C++ e transformá-lo em código nativo de Python, o que é comumente chamado de *wrapping*. Isso garante que os laços custosos demandados pelos algoritmos serão compilados para serem executadas. O único custo adicional desse tipo de abordagem é o chamado *calling overhead*, que é o tempo que o interpretador Python leva para chamar a função compilada. Assim, somente aquelas operações nas quais a maior parte do tempo de execução se dá dentro de laços pesados serão beneficiadas.

Felizmente, uma vez que almeja-se lidar com matrizes grandes, a maior parte das operações envolvendo as mesmas cabe sob esse guarda-chuva. Assim, a biblioteca Miolo consiste essencialmente de duas classes: *Graph* e *Matrix* e diversas operações sobre as mesmas. A primeira consiste da estrutura de matriz esparsa descrita anteriormente, enquanto a segunda é uma estrutura de matriz densa armazenada em ponteiro simples.

A biblioteca conta com todas as operações de espaço vetorial para estas estruturas, além da multiplicação de matrizes. Já a estrutura *Graph* conta ainda com algumas operações particulares relevantes a algoritmos de aprendizado de máquina, como normalização simétrica e laplaciano.

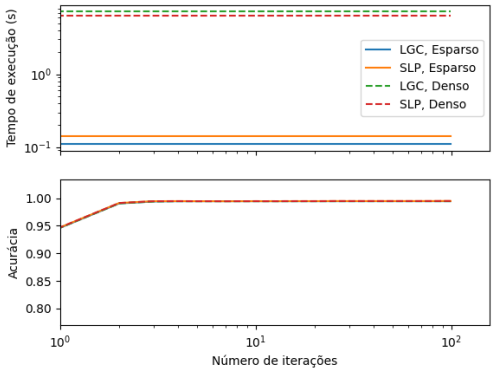

Figura 1. Resultados de tempo de execução e acurácia para dois modelos de inferência utilizando a representação densa e a representação esparsa do grafo de similaridade.

IV. Resultados Preliminares

Na Figura 1, vê-se a diferença tanto sobre o tempo de execução como sobre a acurácia do uso de matrizes densas e esparsas. Representações esparsas de fato melhoram o tempo de execução dos algoritmos, e ainda são eficientes em termos de armazenamento.

V. Considerações Finais

O trabalho até aqui focou-se na construção de uma base para proposição de novos algoritmos. O próximo passo é incluir funcionalidades de geometria diferencial, em particular da geometria riemmaniana, que são úteis enquanto mecanismos de regularização em variedades.

Referências

[1] O. Chapelle, B. Scholkopf, and A. Zien, "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 542–542, 2009.

[2] J. E. Van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Machine learning*, vol. 109, no. 2, pp. 373–440, 2020.

[3] L. Berton, A. de Andrade Lopes, and D. A. Vega-Oliveros, "A comparison of graph construction methods for semi-supervised learning," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[4] D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," *Advances in neural information processing systems*, vol. 16, 2003.

[5] E. Bergamim and F. Breve, "On tuning a mean-field model for semi-supervised classification," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2022, no. 5, p. 053402, 2022.

[6] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proceedings of the 20th International conference on Machine learning (ICML-03)*, 2003, pp. 912–919.

[7] F. Breve, L. Zhao, M. Quiles, W. Pedrycz, and J. Liu, "Particle competition and cooperation in networks for semi-supervised learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 9, pp. 1686–1698, 2011.

[8] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[9] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 31–39, 2010.