Learning to Sort

Prakhar Rathi[0000−0002−5196−3953]

Universität Paderborn, Paderborn NRW 33098, Germany
prakhrathi@gmail.com

Abstract. This paper delves into the scope of learning to sort things using Machine Learning algorithms. Sorting is implemented in a multitude of machine learning techniques. Sorting also helps in the development of models for data analysis and forecasting and is commonly used for robust statistics. The performance shall remain unaffected due to the presence of outliers or small deviations from the given data-set. However, the output of sorting, that is a vector of sorted values, is based on integer values. Therefore, sorting as a function does not exhibit differentiable properties at many points. In addition, the ranking operator outputs the rank of a value in the sorted list thus, it behaves as a piece-wise constant function with its derivatives either null or undefined. A variety of approaches have been studied to overcome this limitation such as uni-modal stochastic row matrices and optimal transport, among others. This aim of this paper is to review the literature corresponding to these approaches and define sorting and ranking operators that can be used in end-to-end training of a machine learning algorithm. The differentiable operators discussed in this paper achieve a desired time efficiency of $O(n\,log\,n)$.

Keywords: Sorting · Automated Sorting · Machine Learning · Differentiable Programming · Deep Learning.

1 INTRODUCTION

In computer science, sorting refers to arranging items in a sequence thus making it easier and faster to locate an item as compared to an unsorted sequence. The major application areas of sorting include efficient lookup or searching and merging of two or more sequences [1]. Sorting is primarily implemented for data-organization in applications such as database management systems [2], recommendation systems [3], and social media networks [4], among others. Moreover, sorting has also been adopted in big data environments. For example, in MapReduce [5], the intermediate key-value pairs created by the map shall be sorted before being rearranged to the reduce tasks [6].

As explained in the literature by Cuturi, M. et al. [21], sorting an array of n elements given as $\theta := (\theta_1, \ldots, \theta_n)$ means finding a permutation σ such that $\theta_\sigma := (\theta_{\sigma 1}, \ldots, \theta_{\sigma n})$ is in an increasing order, where $\sigma\ \in$ set of n! permutations. Sorting is also implemented in the k-NN algorithm to select neighbors and in multi-class classification by minimizing the distance between any two points in the sample space. Sorting can be further classified into comparison-based and non-comparison-based sorting algorithms. Comparison-based algorithms rearrange the input data based on a comparison of their values such as Quick Sort [8] and Merge Sort [9]. On the other hand, non-comparison-based sorting techniques sort the items based on the internal characters of the items. For example, Radix Sort [10] examines individual bits of keys and Counting Sort [11] performs sorting based on the key-value.

Sorting has been used in the least median of squares regression, replacing the traditional least sum of squares loss function by Rousseew et al (1984) [12]. This was further improved to a sum of least trimmed squares regression by Rousseew et al in 2005 [13]. However, the output vector of sorting behaves like a piece-wise linear function which is either increasing or decreasing. Since the output vector is based on integer values, it is non-differentiable at many points. Furthermore, the ranking function outputs the position of a certain element with respect to the other. This can be achieved by optimizing gradient descent over a loss function defined over the scores of each element. Thus, the ranking function is constant for a given element and is different for each element in the list. Therefore, ranking behaves as a piece-wise constant function and the derivative of the ranking loss function is either null or undefined. Due to these limitations, the backpropagation of the gradient descent over the loss function poses a big challenge.

To overcome such challenges, a multitude of approaches have been studied and implemented. These approaches formulate various differentiable proxies. These approaches can be broadly categorized under direct calculation of metrics [14–16] and the introduction of soft operators. Various researchers have proposed soft operators to be plugged into any differentiable loss function such as: the SoftSort replacing the $\arg\mathrm{sort}$ operator [17], the relaxed rank operator [18], pairwise distances between values [19], unimodal row-stochastic matrices [20], and optimal transport [21]. However, none of these approaches can achieve the $O(n\, log\, n)$ time complexity as desired from the sorting and ranking operations. The main aim of this paper is to define an efficient differentiable sorting operator that can be used as a part of an end-to-end trainable machine learning algorithm [22]. The differentiable operator is derived by casting it as a projection onto the convex hull of all permutations, known as the permutahedron.

1.1 Differentiable Programming

Differentiable programming can be defined as a programming paradigm consisting of neural networks as truly functional blocks. These networks have data-dependent branches and recursion. These programs can be differentiated throughout via automatic differentiation. Therefore, the optimization of gradient-based parameters, usually through gradient descent, occurs based on the data [23]. Yann LeCun describes differentiable programs as the programs that can update themselves dynamically by optimizing along a gradient such as a function of the input data like neural networks optimized using gradient descent [24]. However, Blondel, M. et al. [22] state that the set of such architectures for which a gradient can be formulated is limited.

This paper focuses on using permutahedron for learning to define fast differentiable sorting operator. The following sections discuss the various approaches that have been formulated to perform sorting and ranking operations. The key limitation of all these works of literature is that they do not achieve the desired $O(n\, log\, n)$ time complexity. However, the linear formulation of sorting function can be cast over a permutahedron, the convex hull of permutations, to yield a soft sorting operator. This operator can be made continuous by regularization and isotonic reduction. This approach has been studied by [22] and is also discussed in this paper in section 2.6.

The structure of this paper is as follows: Section 2 elucidates the various approaches mentioned in the paragraphs above. Different pieces of literature are reviewed which implement the corresponding algorithms. Thereafter, this paper is concluded in section 3.

2 LEARNING APPROACHES

Due to the non-differentiability of the sorting and ranking operators, as discussed above, researchers from time to time have studied differentiable proxies to sorting and ranking. In this section, an outline of these approaches is provided with references to the related work. These approaches can be categorized as follows:

2.1 Direct Approximation of Metrics

Chapelle, O. et al. (2010) [14] discuss two of the information retrieval measures, namely: Normalized Discounted Cumulitive Gain (NDCG) and Average Precision (AP). The Discounted Cumulitive Gain (DCG) can handle multiple relevance grades. NDCG can be defined by dividing the DCG score such that the best ranking has a score of 1. AP metric is used for binary judgements, $l_i \in \{0, 1\}$. Although, most machine learning algorithms employ gradient descent to optimize the model parameters, information retrieval measures are not continuous, hence non-differentiable, and their optimization using gradient descent is not possible. Chapelle, O. et al. (2010) [14] have performed a smoothing approximation of these IR measures and used gradient descent for minimization. The gradient of the smooth function is computed in $O(m^2)$ time where m denotes the number of documents per query.

However, the space of permutations grows corresponding to the number of documents in query. Adams, R. et al. (2011) [15] devised an approach to overcome this limitation, among others such as varying domain and range of functions. As noted in [15], a framework is developed for end-to-end training of a supervised gradient based learning. A key finding of this literature is that the backpropagation is possible while training the gradients through iterative projection operators. Moreover, Lapin, M. et al. (2016) [16] realize the need to address top-k error optimization, as the previous researches were focused on minimizing the top-1 error.

2.2 Soft Sort

Most of the common operators, such as arg sort, takes a vector as input and return its sorting permutation. However, such operators are non-continuous in nature and thus their gradients are almost null everywhere. This limitation makes them unsuitable for gradient-based optimization. Prillo, S. et al. (2020) [17] have identified this problem and formulated a continuous relaxation to the arg sort operator, known as the `SoftSort`. SoftSort provides meaningful gradients for driving gradient based optimization.

Mathematically, the arg sort operator is defined in [17] as $arg\,sort : \mathbb{R}^n \mapsto S_n$. The domain of this mapping is the n-dimensional real vectors $s \in \mathbb{R}^n$, which is mapped to the set of permutations over n elements $S_n \subseteq \{1, 2, \ldots, n\}^n$. The mapping `sort` : $\mathbb{R}^n \mapsto \mathbb{R}^n$ sorts s in decreasing order and is given as `sort`$(s) = P_{arg\,sort(s)}$. SoftSort derives a continuous relaxation for $P_{arg\,sort(.)}$ operator. SoftSort converges to $P_{arg\,sort(.)}$ and can be projected onto a permutation matrix. This operator can be plugged into any differentiable loss function.

2.3 Relaxed Rank Operators

Relaxed operators for sorting were initially developed in the context of learning to rank. These operators were formulated to optimize the Information Retrieval (IR) metrics, as mentioned above. The first literature proposing rank operator is given by Taylor, M. et al. (2008) [18]. The requirement is to learn a ranking function and generate a score for the input. This score is evaluated using an IR metric and is used to sort the input and produce a ranked list. As noted in [17], the operator `SoftRank` can be defined as a function mapping `SoftRank` : $\mathbb{R}^n \mapsto \mathbb{R}^n$ given by `SoftRank`$(s) = \mathbb{E}[rank(s+z)]$ where z denotes the discount function used for optimization of NDCG metric. The authors show that the gradients can be computed in $O(n^3)$ time. However, the relaxation here is provided with respect to the rank operator. This method requires additional complex calculations to obtain a relaxed arg sort operator from the relaxed rank operator.

2.4 Unimodal Row-Stochastic Matrices

Grover, A. et al. (2019) [20] have devised NeuralSort as a continuous relaxation to the sorting operator to overcome the non-differentiability of the sorting operator with respect to its input. An output of a sorting algorithm can be viewed as a square permutation matrix. NeuralSort [20] returns a unimodal row-stochastic matrix instead of a permutation matrix. An $n \times n$ matrix is defined as a Unimodal Row-Stochastic (URS) matrix if it satisfies the following conditions:

1. **Non-Negativity:** $U[i, j] \geq 0 \quad \forall i, j \in \{1, 2, \ldots, n\}$.
2. **Row Affinity:** $\Sigma_{j=1}^{n} U[i, j] = 1 \quad \forall i \in \{1, 2, \ldots, n\}$.
3. **arg max:** $u_i = arg\,max_j U[i, j] \quad \forall i \in \{1, 2, \ldots, n\}$ and u is a vector of size n. Then $u \in S_n$, i.e it is a valid permutation.

Every n-dimensional permutation $z = [z_1, z_2, \ldots, z_n]^T$ is associated to a permutation matrix $P_z \in \{0,1\}^{n \times n}$ which can be defined as:

$$P_z[i,j] = \begin{cases} 1 & if \;\; j = z_i \\ 0 & otherwise \end{cases} \tag{1}$$

The NeuralSort operator [20] can efficiently project any unimodal matrix to the desired permutation matrix through an arg max function, it is also suitable for gradient optimization. As noted in [20], a `Stochastic Computation Graph (SCG)` specifies the forward value and the backward gradient computation. An SCG is defined as a directed graph with three types of nodes namely: *input* nodes for external inputs and parameters, *deterministic* nodes representing deterministic functions, and *stochastic* nodes. SCG can be further studied for gradient estimation in [25]. By chain rule, the definition of gradients of an objective function for any node in the graph requires that gradients for intermediate nodes are well defined. The authors [20] achieve the computation in time $O(n^2)$ by stochastic optimization of `NeuralSort` over permutations.

2.5 Optimal Transport

Cuturi, M. et al. (2019) [21] consider sorting as an Optimal Transport (OT) problem. The n values to be sorted are matched to an auxillary target measure on any increasing family. The authors propose that a permutation σ for an array, given by $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, to be sorted can be recovered using Optimal Assignment (OA) problem. The OA problem is solved from an input measure on all values in x to an auxillary probability target measure supported on any increasing family $y = (y_1 < y_2 < \cdots < y_n)$. The smallest element in x is matched with y_1, the second smallest with y_2, and so forth. To recover the differentiable sorting operator, the OT problem is regularized and solved using the Sinkhorn algorithm [26]. The OT problem is defined between the two probability measures given by $\Sigma_{i=1}^{n} a_i \delta_{x_i}$ and $\Sigma_{j=1}^{n} b_j \delta_{y_j}$ where a, b are the probability weight vectors for x, y respectively, and is given by the matrix $P_{arg\,sort(s)}$.

A variant of OT employing entropic regularization is applied to yield a continuous relaxation $P_{arg\,sort}(s)^{\epsilon}$, where ϵ is the regularization strength. At $\epsilon = 0$, an integer valued vector is recovered. As ϵ increases, regularization produces a mixture of continuous values. Gradients are then computed by backpropagation using the Sinkhorn operator. Evidently, the computational complexity of this approach is $O(ln^2)$ where, l is the number of Sinkhorn iterations. However, the authors propose that the complexity can be significantly reduced by using very less auxillary target values m, where $m \neq n$, thereby reducing the complexity to $O(nml)$. Cuturi, M. et al. (2019) [21] proposed to replace the permutation matrix by a doubly stochastic matrix which is calculated in $O(Ln^2)$ time using Sinkhorn. A recent approach [22], discussed in the following sub-section, is based on direct regularization using a combination of Quadratic and Entropic regularization to achieve the target in $O(n\,log\,n)$ time.

2.6 Projections on the Permutahedron

Various approaches discussed above in Section 2 imply that replacing the inner calls to sorting operators by differentiable proxies can enhance the end-to-end trainable losses. Using this inference, Blondel, M. et al. (2020) [22] have devised the first differential sorting and ranking operators that successfully implement sorting in $O(n\,log\,n)$ time and $O(n)$ space complexity. The operators are constructed as projections on the convex hull of permutations, known as the permutahedron, followed by an isotonic optimization reduction. For $\theta = (\theta_1, \theta_2, \ldots, \theta_n) \in \mathbb{R}^n$
as the scores produced by a model and $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ as the permutation of $[n]$, the sorting operator is defined as:

$$P_{arg\,sort(\theta)} = s(\theta) := \theta_{\sigma(\theta)} \tag{2}$$

Here, $\sigma(\theta) = (\sigma(\theta_1), \sigma(\theta_2), \ldots, \sigma(\theta_n))$ defines the arg sort of θ as the indices sorting θ. Sorting can be cast as a linear program over the permutahedron and is defined as $s(\theta) = arg\,max_{y \in \mathcal{P}(\theta)} < y, \rho >$ where $\rho = (n, n-1, \ldots, 1)$ and $\mathcal{P}(\theta) := conv(\{\theta_\sigma : \sigma \in set\ of\ permutations\}) \subset \mathbb{R}^n$ gives the convex hull of permutations for a vector $\theta \in \mathbb{R}^n$ (Figure 1). The fundamental theorem of linear programming (Dantzig, G. et al. (1955), Theorem 6 [27]) states that an optimal solution over a permutahedron is always a permutaion. Since θ is used as a linear programming constraint in $s(\theta)$, the function becomes differentiable almost everywhere.

Strong convex regularizations can be introduced over the linear program to smoothen the kinks and enable differentiability throughout. A combination of regularization procedures, namely: Quadratic regularization and Entropic regularization, are introduced to cast the projections onto the permutahedron. Suppose $z, w \in R^n$ and the linear program is given as $arg\,max_{\mu\, \in \mathcal{P}(w)} < \mu, z >$. Quadratic regularization on the linear sorting program gives the Euclidean projection of z on the permutahedron $\mathcal{P}(w)$ (Equation 3). Moreover, entropic regularization ensures that the projection operators follow the convexity of the permutahedron. As represented in Equation (4), e^w ensures that μ belongs to the domain of entropic regularization (i.e. $dom(E)$) that is R^n_+.

$$P_Q(z, w) := arg\,min_{\mu \in \mathcal{P}(w)} \frac{1}{2} \| \mu - z \|^2 \tag{3}$$

$$P_E(z, w) := log\,arg\,max_{\mu \in \mathcal{P}(e^w)} < z, \mu > -E(\mu) \tag{4}$$

The projections defined in Equation (3) and Equation (4) are used to define the differentiable sorting operator. A regularization parameter $\epsilon > 0$ is multiplied with $\Psi \in \{Q, E\}$ and the differentiable sorting operator is defined as:

$$s_{\epsilon\Psi}(\theta) := P_{\epsilon\Psi}(\rho, \theta) \tag{5}$$

The parameter ϵ controls the tradeoff between the original operator and smoothness. The algorithm used for isotonic reduction splits the coordinated into a

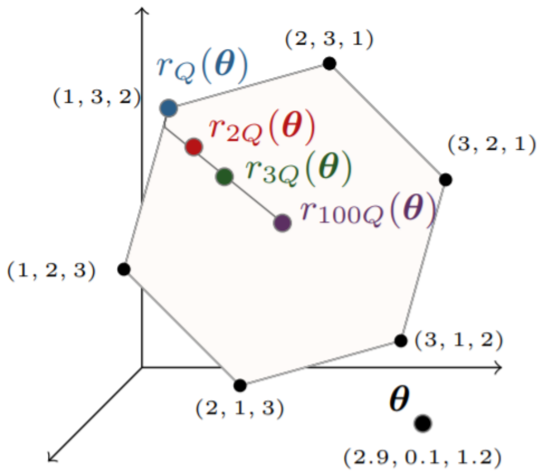
$r_Q(\boldsymbol{\theta})$
(2, 3, 1)
(1, 3, 2)
$r_{2Q}(\boldsymbol{\theta})$
$r_{3Q}(\boldsymbol{\theta})$
(3, 2, 1)
$r_{100Q}(\boldsymbol{\theta})$
(1, 2, 3)
(3, 1, 2)
(2, 1, 3)
$\boldsymbol{\theta}$
(2.9, 0.1, 1.2)

Fig. 1. As described in [22], the figure illustrates a permutahedron $\mathbb{P}(\rho)$ with the vertices as $\rho = (3, 2, 1)$. The example used in the figure deduces the ranks of $\theta = \{2.9, 0.1, 0.2\}$ to $r(\theta) = \{1, 3, 2\}$ and converges with the help of the regularization parameter ϵ. When $\epsilon \to \infty$, $r_{\epsilon Q}(\theta)$ converges towards the centroid of the permutahedron

.

contiguous set of blocks which individually represent sub-problems and are run in linear time. The differentiation of these blocks also have a simple block-wise form. The differentiation of Equation (5) is done by applying the chain rule in $O(n)$ time.

Notably, the top k classification experiment of [21] is considered and it is noted that as compared to the complexity of $O(Ln^2)$ as achieved by Optimal Transport through Sinkhorn iterations and $O(n^2)$ by computing pairwise distances as in [19], the sorting operator defined in the section 2.6 computes the soft operator in $O(n\, log\, n)$ time. In case of label ranking using Spearman's rank correlation coefficient [28], the loss function $\frac{1}{2} \| \mathbf{r} - r(\theta) \|^2$ can be replaced with a continuous function of θ, $\frac{1}{2} \| \mathbf{r} - r_{\Psi}(\theta) \|^2$.

3 CONCLUSION

The paper describes various approached that have been formulated to automate end-to-end gradient optimization for sorting. The recent work of [22] achieves a better time complexity of $O(n\, log\, n)$ as compared to the other approaches such as OT ($O(n^2)$) [21] and minimizing the IR metrics (approx. $O(n^3)$) [14–16]. Soft sorting operator can allow the backpropagation of a gradient based optimization algorithm by smoothening the sorting operator and making it differentiable almost everywhere. This operator can be used in any gradient-based loss function to enable end-to-end training of the learning model.

Considering the fastest sorting algorithms, Merge-Sort [9] performs the sorting operation with a worst case complexity of $O(n\,log\,n)$ but it requires extra space for merging the sorted elements. On the other hand, Quick-sort [8] has an average time complexity of $O(n\,log\,n)$. Therefore, the sorting operator [22] has solved the problem with respect to the time complexity.

Acknowledgements

I am grateful to Karlson Pfannschmidt for useful dicussion, guidance, and comments on the draft paper.

References

1. Kaushal, R., 2017. Why Sorting is So Important in Data Structures. In: IJSRD - Inter-national Journal for Scientific Research Development **5**(7), 2017, ISSN (online): 2321-0613.
2. Graefe, G., 2006. Implementing Sorting in Database Systems. In: ACM Computing Surveys **38**(3), Article 10, 2006. https://doi.org/10.1145/1132960.1132964.
3. Del Vasto-Terrientes, L., Valls, A., Zielniewicz, P., and Borràs, J., 2016. Erratum to: A hierarchical multi-criteria sorting approach for recommender systems. J. Intell. Inf. Syst. **46**(2), 2016, 347-348. https://doi.org/10.1007/s10844-015-0381-4.
4. Li, X., Fang, H., and Zhang, J., 2019. Supervised User Ranking in Signed Social Net-works. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, 184–191. https://aaai.org/ojs/index.php/AAAI/article/view/3784.
5. Dean, J. and Ghemawat, S., 2008. MapReduce: simplified data processing on large clus-ters. Commun. ACM **51**(1), 2008, 107–113. https://doi.org/10.1145/1327452.1327492.
6. Zhu, X. et al., 2019. NN-sort: Neural Network based Data Distribution-aware Sorting. In: arxiv: Data Structures and Algorithms, arXiv:1907.08817 [cs.DS].
7. Cuturi, M., Teboul, O., and Vert, J., 2019. Differentiable Sorting using Optimal Transport: The Sinkhorn CDF and Quantile Operator. arXiv:1905.11885 [cs. LG].
8. Cormen, T., 2009. Introduction to Algorithms. Cambridge, Mass.: MIT Press.
9. Buss, S. and Knop, A., 2019. Strategies for stable merge sorting. In: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Indus-trial and Applied Mathematics, 1272–1290.
10. Andersson, A., Hagerup, T., Nilsson, S., and Raman, R., 1998. Sorting in linear time? J. Comput. System Sci. **57**(1), 1998, 74–93.
11. Gouw, S., Boer, F., and Rot, J., 2014. Proof Pearl: The Key to Correct and Stable Sort-ing. J. Autom. Reasoning **53**(2) , 2014, 129–139. https://doi.org/10.1007/s10817-013-9300-y.
12. Rousseeuw, P. Least median of squares regression, 1984. Journal of the American statistical association **79**(388):871– 880, 1984.
13. Rousseeuw, P. and Leroy, A., 2005. Robust regression and outlier detection **589**. John wiley sons, 2005.

14. Chapelle, O. and Wu, M., 2010. Gradient descent optimization of smoothed information retrieval metrics. Information retrieval **13**(3):216–235, 2010.
15. Adams, R. and Zemel, R., 2011. Ranking via sinkhorn propagation. arXiv e-prints, 2011.
16. Lapin, M., Hein, M., and Schiele, B., 2016. Loss functions for top-k error: Analysis and insights. In Proc. of CVPR, 2016.
17. Prillo, S. and Eisenschlos, J., 2020. SoftSort: A Continuous Relaxation for the `argsort` Operator.
18. Taylor, M., Guiver, J., Robertson, S., and Minka, T., 2008. Softrank: Optimizing non-smooth rank metrics. In: Proceedings of the 2008 International Conference on Web Search and Data Mining, WSDM 08, pp. 7786, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781595939272.
19. Qin, T., Liu, T., and Li, H., 2010. A general approximation framework for direct optimization of information retrieval measures. Information retrieval **13**(4):375–397, 2010.
20. Grover, A., Wang, E., Zweig, A., and Ermon, S., 2019. Stochastic optimization of sorting networks via continuous relaxations. arXiv preprint arXiv:1903.08850, 2019.
21. Cuturi, M., Teboul, O., and Vert, J., 2019. Differentiable ranking and sorting using optimal transport. In Proc. of NeurIPS, 2019.
22. Blondel, M., Teboul, O., Berthet, Q., and Djolonga, J., 2020. Fast differentiable sorting and ranking. arXiv preprint arXiv:2002.08871, 2020.
23. Wang, F., Decker, J., Wu, X., Essertel, G., and Rompf, T., 2018. Backpropagation with Callbacks: Foundations for Efficient and Expressive Differentiable Programming. In: Advances in Neural Information Processing Systems 31, Cur-ran Associates, Inc., pp. 10201–10212.
24. LeCun, Y., 2018. Facebook post about Differential Programming. https://www.facebook.com/yann.lecun/posts/10155003011462143. Accessed 18 June 2020.
25. Schulman, J., Heess, N., Weber, T., Abbeel, P., 2015. Gradient Estimation Using Stochastic Computation Graphs. ArXiv, abs/1506.05254.
26. Cuturi, M., 2013. Sinkhorn distances: lightspeed computation of optimal transport. In: Advances in Neural Information Processing Systems 26, pages 2292–2300, 2013.
27. Dantzig, G., Orden, A., and Wolfe, P., 1955. The generalized simplex method for minimizing a linear form under linear inequality restraints. Pacific Journal of Mathematics, **5** (2):183–195, 1955.
28. Spearman, C., 1984. The proof and measurement of association between two things. American Journal of Psychology, 1904.