Journal of Machine Learning Research () Submitted –; Published –

$\mathbb{S}_n$Julia: A Julia Toolkit for Harmonic Analysis on the Symmetric Group

Gregory Plumb[†] GPLUMB@WISC.EDU
Deepti Pachauri[†] PACHAURI@CS.WISC.EDU
Risi Kondor[*∓] RISI@CS.UCHICAGO.EDU
Vikas Singh[‡†] VSINGH@BIOSTAT.WISC.EDU

[†]*Department of Computer Sciences*
[‡]*Department of Biostatistics & Med. Info.*
University of Wisconsin-Madison
Madison, WI 53706 USA

[*]*Department of Computer Sciences*
[∓]*Department of Statistics*
University of Chicago
Chicago, IL 60637 USA

Editor: - - -

Abstract

$\mathbb{S}_n$Julia is an easy to use software library written in the Julia language to facilitate harmonic analysis on the symmetric group of degree n, denoted $\mathbb{S}_n$ and make it more easily deployable within statistical machine learning algorithms. Our implementation internally creates the irreducible matrix representations of $\mathbb{S}_n$ (in parallel or in a distributed fashion, if appropriate), and efficiently computes fast Fourier transforms (FFTs) and inverse fast Fourier transforms (iFFTs). Advanced users can achieve scalability and promising practical performance by exploiting various other forms of sparsity. Further, the library also supports the partial inverse Fourier transforms which utilizes the smoothness properties of functions by maintaining only the first few Fourier coefficients. Out of the box, $\mathbb{S}_n$Julia currently offers two non-trivial operations for functions defined on $\mathbb{S}_n$, namely *convolution* and *correlation*. While the potential applicability of $\mathbb{S}_n$Julia is fairly broad, as an example, we show how it can be used for clustering ranked data, where each ranking is modeled as a distribution on $\mathbb{S}_n$.

Keywords: Symmetric group, Ranking data, Clustering, Harmonic analysis, Fourier transform, Julia

1. Introduction

Over the last few years, there has been a growing interest in the analysis of data given (or expressed) as a probability distribution over permutations. The set of all possible permutations of n elements constitutes a group called the **symmetric group**, denoted $\mathbb{S}_n$. Several recent solutions to ranking problems, hard combinatorial problems, multi-target tracking and feature point matching tasks (in computer vision) have used harmonic analysis on $\mathbb{S}_n$ to derive more efficient algorithms (Huang et al., 2009; Kondor, 2010; Pachauri et al., 2012). While the idea of generalizing the Fourier transform to non-commutative groups is well established in the Mathematics literature, an easy to use and accessible software library will facilitate the adoption of such concepts within machine learning. In this paper, we describe a Julia based open source library which implements the Fourier transform (and associated functionality) for harmonic analysis of functions defined on $\mathbb{S}_n$. The implementation can use a multi-core cluster (when available) without any need for low-level message passing interface (MPI) programming.

Harmonic analysis on $\mathbb{S}_n$ is defined via the notion of **representations**. A matrix valued function $\rho\colon \mathbb{S}_n \to \mathbb{C}^{d_\rho \times d_\rho}$ is said to be a d_ρ dimensional representation of the symmetric group if $\rho(\sigma_2)\rho(\sigma_1) = \rho(\sigma_2\sigma_1)$ for any pair of permutations $\sigma_1, \sigma_2 \in \mathbb{S}_n$. A representation ρ is said to be *reducible* if there exists a unitary basis transformation which simultaneously block diagonalizes each $\rho(\sigma)$ matrix into a direct sum of lower dimensional representations. If ρ is not reducible, then it is said to be *irreducible*. Irreducible representations or irreps are the elementary building blocks of all of $\mathbb{S}_n$'s representations. A complete set of inequivalent irreducible representations are denoted by $\mathcal{R}$.

The Fourier transform of a function $f : \mathbb{S}_n \to \mathbb{C}$ is then defined as the sequence of matrices

$$\hat{f}(\rho) = \sum_{\sigma \in \mathbb{S}_n} f(\sigma)\rho(\sigma) \qquad \rho \in \mathcal{R}. \tag{1}$$

The inverse transform is

$$f(\sigma) = \frac{1}{n!} \sum_{\rho \in \mathcal{R}} d_\rho \operatorname{tr}\big[\hat{f}(\rho)\, \rho(\sigma)^{-1}\big] \qquad \sigma \in \mathbb{S}_n. \tag{2}$$

Much of the practical interest in Fourier transform can be attributed to various interesting properties of irreps, such as conjugacy and unitarity.

1.1 The irreducible representation of $\mathbb{S}_n$

There are several ways to construct irreducible representation of $\mathbb{S}_n$ (Sagan, 2001). One such representation is called Young's orthogonal representation (YOR). The YOR matrices are real and unitary and therefore orthogonal. To benefit from the computational advantages of orthogonal matrices, $\mathbb{S}_n$Julia uses YOR internally. In the supplement, we provide a short review of the background required for constructing YORs.

2. $\mathbb{S}_n$Julia Toolkit

$\mathbb{S}_n$Julia is implemented in a high-level programming language called Julia (provided under a MIT license). The most important features of the toolkit are accessibility, extensibility, and performance. The toolkit and the required documentation is available at: `https://github.com/GDPlumb/SnJulia.jl/`.

Accessibility. We placed a great deal of emphasis on the ease of use of the toolkit. This will allow a non-specialist (in harmonic analysis) to utilize the functionality of this library within standard machine learning algorithms, when analyzing data on $\mathbb{S}_n$. In particular, the fully functionality of $\mathbb{S}_n$Julia is available simply by loading the package "SnJulia" through Julia's built in package manager. The $\mathbb{S}_n$Julia user manual provides many examples demonstrating the syntax for accessing the various features of $\mathbb{S}_n$Julia and gives a high level overview of the key properties of YOR matrices and the Fourier transform. The minimalist design and coding consistency makes $\mathbb{S}_n$Julia easy to use and modify.

Extensibility. Interoperability is a key component of Julia — it allows easy access to various pre-existing high quality and mature libraries written in many other languages with minimal additional overhead. Therefore, various machine learning libraries can be easily incorporated into $\mathbb{S}_n$Julia projects. For example, C and Fortran functions can be called directly from $\mathbb{S}_n$Julia projects without any "glue" code. $\mathbb{S}_n$Julia allows access to external libraries written in languages such as Python, Java, and R, by easily passing the data to these libraries. Later, we demonstrate this property of $\mathbb{S}_n$Julia for clustering ranked data by using R's `sparcl` library to perform hierarchical clustering. Finally, Julia code can be called directly from C/C++. As a result, $\mathbb{S}_n$Julia can be used seamlessly within existing machine learning tools as needed.

Parallelism. $\mathbb{S}_n$Julia inherits the parallelism offered by the Julia platform. It allows a multi-processing environment to run a code on multiple processes in separate memory domains concurrently. Julia uses empirically derived rules to determine the trade-off between synchronization overhead for multithread computation and single thread sequential computation and proceeds with the best option. In our implementation, $\mathbb{S}_n$Julia functions are designed to use all worker processes that a user makes available to Julia. This setup allows the user to analyze the data on a single process, on multiple processes on a local machine, or via multiple processes spread across a cluster with essentially no change to the user code beyond initially making the processes available.

Sparsity. For various practical applications, we encounter problems for n greater than 15. Even storing such data is problematic as $n!$ is ~ 1 trillion. Unless one exploits the smoothness/sparsity properties of f, computation will be intractable. But notice that often, problems exhibit interesting sparsity patterns Kueh et al. (1999); for example, the Fourier transform of functions on homogeneous spaces of $\mathbb{S}_n$ are usually band-limited in the sense that their Fourier transform is identically zero except for a small set of Fourier matrices. $\mathbb{S}_n$Julia is designed to utilize such patterns, making it very efficient. Specifically, the function `FFT_BL()` is implemented to offer significant efficiency benefits when the user a priori knows the band-limited form of f. For problems with unknown sparsity pattern, the special function `FFT_SP()` first determines the sparsity structure of f and then proceeds to the actual FFT calculation. Partial inverse Fourier transform is also supported in $\mathbb{S}_n$Julia which is important to induce smoothness in f. In particular, function `IFFT_P()` can be used to approximate f using just first few Fourier coefficients of the full Fourier transform.

3. Example: Fourier Domain Features for Clustering Ranks

Consider a ranking dataset composed of N examples where i^{th} instance $(i = 1, \cdots, N)$, is a permutation $\sigma_i \in \mathbb{S}_n$ of n items, listed in order of preference. Given such data, we want to identify groups of examples with similar preferences, which may be helpful for a downstream preference behavior study or rank prediction applications Crammer et al. (2001). Various probabilistic models for ranking are popular in the research community such as Mallows model Murphy and Martin (2003), which nicely capture the variability in the observations when the observed rankings are noisy or incomplete Busse et al. (2007). Typically, the i^{th} instance is represented as a function $f_i(\sigma) = \frac{e^{-\gamma d(\sigma_i,\sigma)}}{Z_\gamma}$ on $\mathbb{S}_n$. Here, γ is the spread parameter, $d(.,.)$ is a valid distance metric on permutations, and Z_γ is the normalization constant. The clustering problem seeks to partition the dataset into K clusters to minimize the following objective:

$$\arg\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} \sum_{1\le i,j\le N:(i,j)\in C_k} \|f_i - f_j\|^2 . \tag{3}$$

A geometric view of functions defined on $\mathbb{S}_n$ as embedded in the space $[0,1]^{n!}$ quickly becomes intractable and hard to interpret. On the other hand, the seminal work of Diaconis (1988) explains how the Fourier coefficients precisely encode the structural properties of the distributions on $\mathbb{S}_n$. Following ideas described in Diaconis (1988), recently, Clémençon et al. (2011) introduced a Fourier space formulation equivalent to (3)

$$= \frac{1}{n!} \sum_{\rho\in\mathcal{R}} d_\rho \sum_{k=1}^{K} \sum_{1\le i,j\le N:(i,j)\in C_k} \|\widehat{f}_i(\rho) - \widehat{f}_j(\rho)\|^2_{HS(d_\rho)} . \tag{4}$$

Further, they used a specialized feature selection procedure for clustering the induced spectral features as in Witten and Tibshirani (2010) and showed that frequently one only needs a few spectral features to explain the clustering choices. In $\mathbb{S}_n$Julia, only a few lines of code implement this algorithm,

```
# Construct Mallow Distribution at for each rank σ_i with spread parameter γ
f_i = MallowsDistribution(σ_i, γ)
# Create Fourier representation for S_n
RA, PT = YOR(n)
# Calculate Fourier Transforms of f_i
f̂_i = FFT(n, f_i, RA, PT)
# Pass the Fourier coefficient matrix (array of f̂_i, i = 1, ··· , N) to R's sparcl
library. Sparcl treat Fourier coefficient matrix as the feature matrix, and
perform a sparse hierarchical clustering.
# sparcl_script.R is a simple script which calls R from S_nJulia platform, and
perform clustering. The code to call this script is
R = "R"
CMD = "CMD"
BATCH = "BATCH"
loc = ``path-to-sparcl_script.R''
run(`$R $CMD $BATCH $loc`)
```

This example shows that $\mathbb{S}_n$Julia is fairly flexible and can be used with advance machine learning libraries for data analysis on $\mathbb{S}_n$. Further, the $\mathbb{S}_n$Julia distribution includes a well documented example for clustering problems on a synthetic dataset.

Acknowledgments

This work was supported in part by NSF CCF 1320344, NSF CCF 1320755, a REU supplement to NSF RI 1116584 and the University of Wisconsin Graduate School.

References

L. M. Busse, P. Orbanz, and J. M. Buhmann. Cluster analysis of heterogeneous rank data. In *ICML*, 2007.

S. Clémençon, R. Gaudel, and J. Jakubowicz. Clustering rankings in the fourier domain. 2011.

K. Crammer, Y. Singer, et al. Pranking with ranking. In *NIPS*, volume 14, 2001.

P. Diaconis. Group Representations in Probability and Statistics. *Institute of Mathematical Statistics Monograph Series*, 1988.

J. Huang, C. Guestrin, and L. Guibas. Fourier theoretic probabilistic inference over permutations. *JMLR*, 10, 2009.

R. Kondor. A Fourier space algorithm for solving quadratic assignment problems. 2010.

K.-L. Kueh, T. Olson, D. Rockmore, and K.-S. Tan. Nonlinear approximation theory on finite groups. *Department of Mathematics, Dartmouth College, Tech. Rep. PMA-TR99-191*, 1999.

T. B. Murphy and D. Martin. Mixtures of distance-based models for ranking data. *Computational statistics & data analysis*, 41, 2003.

D. Pachauri, M. Collins, V. Singh, and R. Kondor. Incorporating domain knowledge in matching problems via harmonic analysis. *ICML*, 2012.

Bruce E. Sagan. *The Symmetric Group*. Graduate Texts in Mathematics. 2001.

D. M. Witten and R. Tibshirani. A framework for feature selection in clustering. *Journal of the American Statistical Association*, 105, 2010.