SIMBSIG: Similarity search and clustering for biobank-scale data

Michael F. Adamer¹,³,†, Eljas Roellin¹,†, Lucie Bourguignon², Karsten Borgwardt¹,³

¹ Department of Biosystems Science and Engineering, ETH Zurich, Basel, 4058, Switzerland
² Department of Health Sciences and Technology, ETH Zurich, Zurich, 8008, Switzerland
³ Swiss Institute for Bioinformatics (SIB), Lausanne, Switzerland

† These authors share first authorship.

Abstract: In many modern bioinformatics applications, such as statistical genetics, or single-cell analysis, one frequently encounters datasets which are orders of magnitude too large for conventional in-memory analysis. To tackle this challenge, we introduce SIMBSIG, a highly scalable Python package which provides a scikit-learn-like interface for out-of-core, GPU-enabled similarity searches, principal component analysis, and clustering. Due to the PyTorch backend it is highly modular and particularly tailored to many data types with a particular focus on biobank data analysis.

Availability SIMBSIG is freely available from PyPI and its source code and documentation can be found on GitHub (https://github.com/BorgwardtLab/simbsig) under a BSD-3 license.

Contact: michael.adamer@bsse.ethz.ch

© The Author 2015. Published by Oxford University Press. All rights reserved. For permissions, please e-mail: journals.permissions@oup.com
The neighbors module consists of our core routine for nearest-neighbours searches in a batched, brute force manner. This guarantees exact results with a precisely controllable memory-runtime trade-off. We implemented a batched K-Nearest Neighbors (KNN) search, where the number K of neighbors is fixed a priori, and a radius neighbour search, where all neighbours within a user-defined radius are returned. Based on this core routine, we implemented classifiers and regressors.

We use a brute-force approach only due to the infeasibility of other exact methods in this scenario, while retaining most other functionality of scikit-learn such as the choice of a range of metrics including all ℓp distances. Since, especially in genetic applications, the data is often high dimensional and subject to the curse of dimensionality, we also implemented the fractional ℓp distance (Aggarwal et al., 2001). We further allow for precomputed distance matrices or user-defined functions (“callables”). The “callable” functionality is also present in scikit-learn and provides an easy way for more sophisticated (dis-)similarity functions such as kernel functions. Kernels exist, for example, for SNP data (Wessel et al., 2006) and may help obtaining more accurate results.

The second core routine is batched k-means clustering from the clustering module. The implementation is based on the fast k-means algorithm by Sculley (2010). Similarly to the NN search, we accept numpy arrays or hdf5 file handles as input and we can cluster with respect to any ℓp norm, the cosine distance or a “callable” distance function. We additionally implemented a heuristic stopping criterion based on the maximum relative change of any cluster centre. In this scenario, this stopping criterion works well for the ℓp distances, however, it may be lacking an analogue for more involved distance functions. In the current implementation, for any metrics that are not ℓp, the relative change becomes an absolute change.

The third pillar of SIMBSIG is the decompose module. This module provides an out-of-core, GPU-accelerated implementation of Halto’s approximate PCA algorithm (Halto et al., 2011)

### 3 Experiments

To test accuracy, we designed an extensive set of unit tests, where we compare all possible input choices to the output of scikit-learn which we assume to be the ground truth.

The speed of SIMBSIG was benchmarked on a large, artificial SNP dataset, where SNPs are encoded according to dominance assumption.

We sampled “participants” represented by a 10,000 dimensional vector with independent entries in {0, 1, 2}, representing 10,000 SNPs with probabilities {0.6, 0.2, 0.2}. In total, we sampled 500,000 such participants and stored the dataset in the hdf5 format. We tested in-memory CPU performance against scikit-learn and benchmarked out-of-core scenarios. For in-core computations, GPU accelerated SIMBSIG similarity searches are ~1 order of magnitude faster compared to scikit-learn. Out-of-core similarity searches run both with and without GPU acceleration, where SIMBSIG’s GPU acceleration feature offers a significant reduction in execution time for these heavy computations. A reference GPU implementation given by cuML (Raschka et al., 2020) could not use the h5py package needed to load hdf5 files.

### Funding

This work was supported by the Swiss National Science Foundation [PZ00P3_186101] and the Alfred Knupp Prize for Young University Teachers of the Alfred Knupp von Bohnen und Halbach-Stiftung [K.B.]

### References


