PLOS ONE

RESEARCH ARTICLE

DagSim: Combining DAG-based model structure with unconstrained data types and relations for flexible, transparent, and modularized data simulation

Ghadi S. Al Hajj[1]*, **Johan Pensar**[2], **Geir K. Sandve**[1]

1 Department of Informatics, University of Oslo, Oslo, Norway, **2** Department of Mathematics, University of Oslo, Oslo, Norway

* ghadia@uio.no, ghadi.alhajj8@gmail.com

Citation: Al Hajj GS, Pensar J, Sandve GK (2023) DagSim: Combining DAG-based model structure with unconstrained data types and relations for flexible, transparent, and modularized data simulation. PLoS ONE 18(4): e0284443. https://doi.org/10.1371/journal.pone.0284443

Editor: Emmanuel S Adabor, Ghana Institute of Management and Public Administration, GHANA

Received: November 30, 2022

Accepted: March 30, 2023

Published: April 14, 2023

Peer Review History: PLOS recognizes the benefits of transparency in the peer review process; therefore, we enable the publication of all of the content of peer review and author responses alongside final, published articles. The editorial history of this article is available here: https://doi.org/10.1371/journal.pone.0284443

Data Availability Statement: The data used in this paper are generated using computer simulations. The reader can simulate similar data using the described tool, which is available as a python

Abstract

Data simulation is fundamental for machine learning and causal inference, as it allows exploration of scenarios and assessment of methods in settings with full control of ground truth. Directed acyclic graphs (DAGs) are well established for encoding the dependence structure over a collection of variables in both inference and simulation settings. However, while modern machine learning is applied to data of an increasingly complex nature, DAG-based simulation frameworks are still confined to settings with relatively simple variable types and functional forms. We here present DagSim, a Python-based framework for DAG-based data simulation without any constraints on variable types or functional relations. A succinct YAML format for defining the simulation model structure promotes transparency, while separate user-provided functions for generating each variable based on its parents ensure simulation code modularization. We illustrate the capabilities of DagSim through use cases where metadata variables control shapes in an image and patterns in bio-sequences. DagSim is available as a Python package at PyPI. Source code and documentation are available at: https://github.com/uio-bmi/dagsim

Introduction

Data simulation is fundamental for machine learning (ML) and causal inference (CI), as it allows ML/CI methods to be evaluated in a controlled setting using a ground truth model [1–3]. For the purpose of designing flexible, controllable, and transparent simulator models, the class of directed acyclic graphs (DAGs) provides a highly useful framework. The DAG is used to encode the structure of a model involving multiple variables in a form that is both compact and intuitive to a human user. In addition, the resulting DAG-based model is modular and allows for building complex simulators from simpler local components or modules. In a purely probabilistic model, known as a Bayesian Network (BN) [4], the DAG is used to specify the dependence structure over the considered variables. In a causal model, known as a structural causal model (SCM) [5], the DAG is used to specify the causal structure of the underlying

package (https://pypi.org/project/dagsim/). The code (available as python scripts and as jupyter notebooks) for simulating the data used in the manuscript usecases can be found on GitHub (https://github.com/uio-bmi/dagsim/tree/main/manuscript_usecases). Also, the user can run these simulations online by clicking on the "launch binder" badge provided on that page, using the free Binder service.

Funding: The author(s) received no specific funding for this work.

Competing interests: The authors have declared that no competing interests exist.

data-generating process. In either case, a simulation model is defined by specifying the functional relations between each node and its parents in the assumed graph. In a BN, these relations are typically defined as probability distributions, while an SCM typically models relations as deterministic, where the value of a node is computed based on the value of its parents and an additional exogenous random variable (often referred to as a noise variable). In terms of simulation, there is no practical distinction between the purely probabilistic (BN) and causal perspective (SCM)–in either case, data is generated through direct forward sampling following a node ordering that is consistent with the given DAG (known as a topological ordering). However, the fundamental difference is that an SCM is equipped with some additional causal capabilities that go beyond those of a BN. For example, scenarios involving interventions and counterfactuals can be simulated by making simple local modifications to the original model ahead of a standard simulation.

While there is in principle no limitation on the types of variables or functional forms in the BN and SCM frameworks, the main emphasis has historically been on relatively small DAGs with variables of basic data types (typically ordinal/categorical scalar values) [6–12]. A visual notation known as plate notation is well-established for denoting vector-valued variables in BNs, but representing a k-dimensional tensor requires a fixed k-level nesting of plates, and there is no well-established notation for representing sets or sequences. This stands in stark contrast to the recent neural network (NN)-driven machine learning revolution, where the main aspect has been the ability to learn useful representations from data of large dimensionality and complex structure [13–15]. The canonical example of this is the learning of complex signals from large, two-dimensional structures of pixel values in image analysis, as well as from sequences of words in natural language processing.

The emphasis on simple types of variables and functional relations in the graphical model field is also apparent from the programming libraries available for structure learning, parameter inference and simulation from graphical models. For example, the seminal bnlearn R package [9] can both infer parameters and simulate data from a model but is restricted to numerical variables, whether discrete or continuous, and restricted to full conditional probability tables and linear regression models as functional forms. DAG-based simulation is also supported in a variety of other packages, either as the main purpose or as a side purpose (their properties in terms of simulation are summarised in Table 1). Many of the packages are explicitly restricted to linear relations, as in the structural equation models (SEM) framework. All the mentioned packages share with bnlearn the restriction to numerical variables and particular functional forms.

We here argue for the usefulness of combining the ideas of carefully designed models of variable relations from the graphical modelling field with the complex data types that are characteristic of the current wave of NN-driven deep learning. We present a Python library DagSim that streamlines the specification of simulation scenarios based on graphical models where variables and functional relations can be of any form. The fundamental idea of the framework is simple yet powerful: allowing the user to define a DAG-based simulation by connecting nodes to their parents through standard Python functions with unrestricted parameter and return values. DagSim provides a range of functionality for increasing the convenience and transparency of such a simulation setup—offering a choice between an external (YAML-based) or internal (Python-based) succinct and transparent domain-specific language (DSL) to specify simulations involving plates, mixture distributions and various missing data schemes. It also includes functionalities of specific use for the simulation of causal scenarios, including native support for simulating sample selection bias.

Table 1. An overview of all established frameworks that to the authors' knowledge offer DAG-based simulation functionalities, describing for each package the main purpose, the type of data it simulates, the functional forms used, and the additional simulation utilities provided. The bnlearn package [9] can both infer parameters and simulate data from a model, with numerical variables and functional forms restricted to full conditional probability tables and linear regression models. The pgmpy package [16] is similar to bnlearn in terms of its purpose and simulation functionalities. The package simCausal [11] is more aimed toward causal inference problems and thus focuses on simulating longitudinal data based on SEMs. The main goal of the simMixedDAG package [12] is to simulate "real life" datasets based on a learned generalised additive model or user-defined parametric linear models. The package MXM [7] simulates data from multivariate Gaussian distributions based on a user-defined or randomly generated adjacency matrix, while abn [6] simulates data from Poisson, multinomial, and Gaussian distributions based on a user-defined adjacency matrix. The packages dagitty [10], dagR [17], and lavaan [8] provide similar functionalities for simulating data based on SEMs.

Framework	Main Purpose	Data type	Functional form	Distinctive features	Reference
DagSim	Data simulation	Any data type (passed directly between nodes)	Any form (custom function)	Plates, selection bias, missing values, stratification	This paper
bnlearn	Structure, and parameter learning	Discrete and continuous	Categorical distribution and linear Gaussian model	Includes several off-the-shelf models	[9]
pgmpy	Model learning, and approximate, exact, and causal inference	Discrete and continuous	Categorical distribution and linear Gaussian model	Includes several off-the-shelf models	[16]
simCausal	Simulation of SEM-based complex longitudinal data structures	Discrete and continuous	Linear model	Counterfactual data, interventions, time-varying nodes	[11]
simMixedDAG	Simulation of data from parametric and non-parametric DAG models	Discrete and continuous	Generalized additive model	Learns a non-parametric model from data	[12]
MXM	Feature selection	Discrete and continuous	Linear Gaussian model	Simulates a DAG with arbitrary arc density	[7]
abn	Modelling data with additive Bayesian networks	Discrete and continuous	Generalized linear model	Simulates a DAG with arbitrary arc density	[6]
dagitty	Graphical analysis of Structural Causal Models	Binary and continuous	Linear Gaussian and logistic models	Characterisation, restructuring and random generation of DAGs	[10]
dagR	Construct and evaluate DAGs, and simulate data	Binary and continuous	Linear Gaussian and logistic models	Includes several off-the-shelf models	[17]
lavaan	Latent Variable Analysis	Continuous	Linear model	Fits a latent variable model to data	[8]

https://doi.org/10.1371/journal.pone.0284443.t001

Implementation

To specify a DagSim simulation model, a user simply defines a set of nodes (variables) along with possible input nodes (parents), which together make up the backbone of the model in the form of a directed graph. The user then assigns a general Python function for simulating the value of each node given the values of parent nodes, if any. When the model has been fully specified, the package checks that the provided directed graph is acyclic, thus ensuring that the values of any input node can be sampled prior to all downstream nodes. Following a topological ordering of the nodes, DagSim then uses standard forward sampling to simulate the values of each node by calling its assigned function and providing the values of its parents (if any) as arguments. Importantly, parent values are directly passed on as native Python values, ensuring that the framework supports general data types and any functional forms. The simulated data is saved to a CSV file.

In addition to more standard simulation scenarios, DagSim provides additional types of nodes that facilitate the simulation of different types of real-world scenarios. The Selection node, for example, allows the user to simulate selection bias [5] through a function that governs the sample selection scheme, where the arguments for that function are similar to those of a standard node. The Missing node, on the other hand, provides a convenient way to simulate missing entries in the data by specifying the variable that will consequently have missing values and another standard node that specifies which entries should be removed. Finally, the Stratify node offers a way to automatically stratify the resulting data into separate files through a single function that defines the stratum of each sample. Additionally, DagSim supports transparent a specification of simulations based on a succinct YAML format.

Step-by-step example

Suppose you would want to simulate sequences of coin tosses, each represented as a 10–20 long text of H (head) and T (tail), according to a sample-specific probability of getting heads, sampled itself from a uniform distribution. Fig 1 shows the overall workflow one would follow:

- First, define one node for the probability of getting Heads, one node for the number of coin tosses per sample, and one node for the sampled sequence itself (which has incoming edges from the two other nodes), using e.g. YAML to specify the graph
- Second, define the simulation instructions in the YAML file
- Third, define the custom function for simulating a sequence of tosses, with the other two nodes utilizing already existing functions, e.g. using numpy.
- Finally, simply run the defined simulation, e.g. from the command line.

The code and files corresponding to this example can be found in the supplementary material.

Use

The driving motivation for DagSim is the ability to combine basic (scalar, ordinal/categorical) variables with complex or high-dimensional data types in a transparent structure as defined by a DAG, without any restrictions on the complexity of the functional relations between nodes. We illustrate these capabilities through two stylized simulation examples, where basic meta-data variables are controlling 1) shapes in an image (two-dimensional numeric tensor), and 2) bio-sequence patterns in an adaptive immune receptor repertoire (set of sequences). Our main

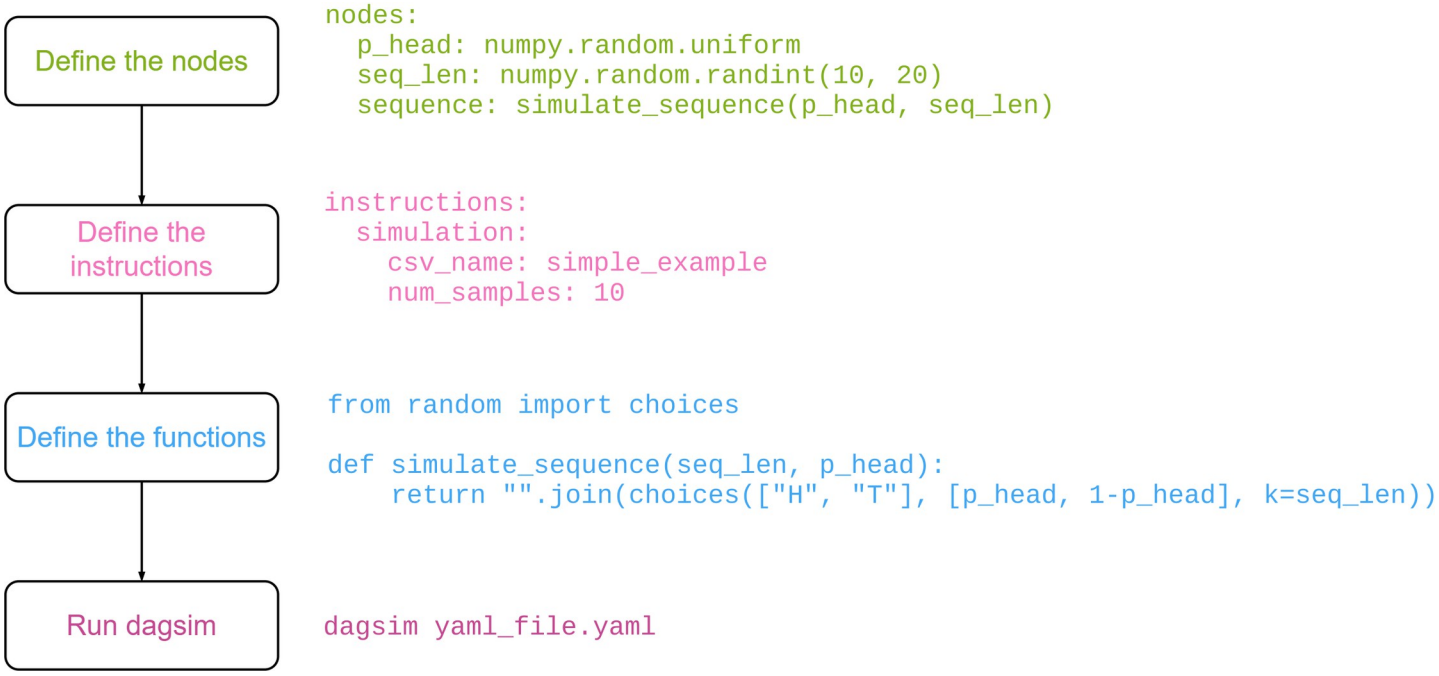


Fig 1. A typical workflow of simulating data using DagSim.

https://doi.org/10.1371/journal.pone.0284443.g001

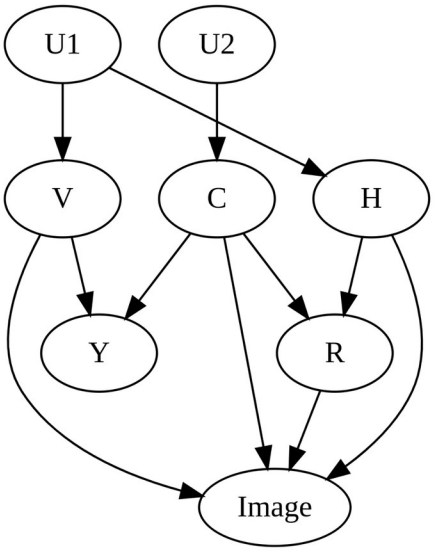
emphasis is on the ease of defining simulations and the transparency of the resulting simulation models. Detailed versions of these examples can be found in S1 and S2 Figs, respectively.

A first use case is based on a study by Sani et. al. [18] on causal learning as a tool for explaining the behaviour of black-box prediction algorithms. In order to illustrate their approach, they simulated simple images with specific shapes overlaid on a black background, where an additional set of scalar variables controlled the probability of each of the different shapes being introduced to the image. We show how such a simulation is easily reproduced using DagSim, based on a succinct, transparent, and well-modularized model specification (Fig 2A and 2B). The simulation of each node given its parents is defined by a set of Python functions provided

```yaml
graph:
  nodes:
    U1: uniform(0,1)
    U2: uniform(0,1)
    H: binomial(1, U1)
    C: binomial(1, U2)
    V: complement_binomial(U1)
    R: sigmoid_binomial(C, H, "H")
    Y: sigmoid_binomial(C, V, "V")
    Image: drawImage(H, V, R, C)
  python_file: ImagesExample.py

instructions:
  simulation:
    csv_name: Images_metadata
    num_samples: 50
```

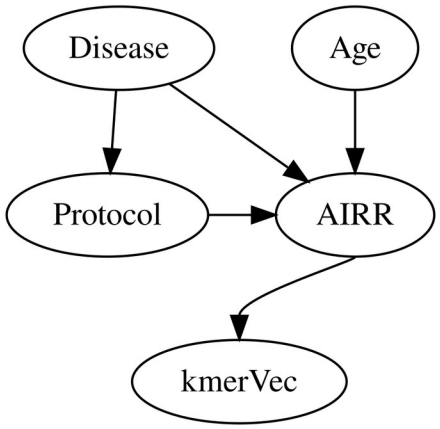
(a)



(b)

```yaml
graph:
  nodes:
    Disease: binomial(1, 0.5)
    Age: randint(10,80)
    Protocol: assign_protocol(Disease)
    AIRR:
      function: create_airr(Disease, Age, Protocol)
      observed: False
    kmerVec: encode_kmers(AIRR)
  python_file: BioseqExample.py
instructions:
  simulation:
    num_samples: 50
    csv_name: "BioseqExample_yaml"
```

(c)



(d)

Fig 2. (a-b) The YAML specification and corresponding DAG for the image simulation use case, (c-d) The YAML specification and corresponding DAG for the biosequence simulation use case.

https://doi.org/10.1371/journal.pone.0284443.g002

in the supplementary material, where the main function is the one that generates an image conditioned on the scalar metadata values. If this use case was to be performed by any of the existing DAG-based simulation frameworks, then the scalar values would have to be simulated separately using an appropriate DAG. Following that, a separate function that takes as input the variables V, C, R, and H and iterates on all the samples could be used to create the desired image. This would detach the image construction process from the rest of the simulation making the code unnecessarily complicated and less transparent.

A second use case exemplifies simulation in settings of high-dimensional biomarkers and low-dimensional patient characteristics. The considered biomarker is based on sequence patterns in a gene known as the immune receptor, which is reflecting what individual adaptive immune cells are recognizing and reacting to. The set of DNA sequences for this gene across all adaptive immune cells in the body is collectively known as the adaptive immune receptor repertoire (AIRR). Any disease state with immune system involvement, including infectious disease, auto-immunity and cancer, introduces sequence patterns in the AIRR. Additionally, it has been proposed that immune repertoires become less diverse with age [19] and that experimental protocols introduce particular sequence biases in observed AIRR datasets [20, 21]. Simulation of such biomarker signals allows benchmarking the ability of the current methodology to infer disease state from AIRR patterns [22, 23], as well as to assess the robustness of the learning process to variability in patient characteristics and experimental biases [24]. The model specification and resulting DAG are shown in Fig 2C and 2D. The Python functions are provided in the supplementary material, where the main function simulates the AIRR for each patient conditioned on disease state, age, and the experimental protocol used. If this use case was to be performed by any of the existing DAG-based simulation frameworks, one would have to use a numeric representation of the sequences, with for example the use of ad hoc end-of-sequence numeric codes to emulate a set of variable length sequences. As the frameworks also do not support the specification of custom functions, one would need to supplement a DAG-based simulation of baseline sequences (in numeric representation) with post-hoc functions for implanting desired signals.

Conclusion

We have here argued that DAG-based simulation should transcend the traditional settings of only numeric-valued variables, allowing the convenience and transparency of graphical models to see use also in settings with more complex data types. Specifically, complex data types would bring these simulations closer to applications typically considered by modern machine learning (often deep learning) models. Hence, we consider the integration of complex data types and graphical modelling for simulation purposes as highly timely, given both the increasing inclusion of complex data types in modelling scenarios and the increasing interest in causal concepts in the ML field. In terms of the latter, there has been recent research into how underlying causal mechanisms affect ML strategies [25], research into how the underlying causal structure determines whether data from different sources can be successfully fused for learning ML models [26], research into how overlaid signals arising from distinct mechanism can be disentangled [27, 28], calls for extending modern ML methods to directly predict effects of interventions [29], calls for incorporating non-linear machine learning methods for causal inference in epidemiology [30], and an increasing interest in how causal mechanisms affect the stability (generalizability) of ML models when applied to new settings [24, 31]. The combination of a DAG-based model backbone with flexible data types and functional relations provides for transparent and modularized simulation models in these emerging settings, where low-dimensional variables are connected to complex patterns in high-dimensional variables.

Through a succinct YAML format for defining the model backbone and the use of individual, native Python functions for defining the functional relations to each node, DagSim provides a straightforward, practical implementation to support such an approach. The framework also natively supports specific functionalities that are useful when simulating data, e.g. for emulating selection bias and missing data, and could in the future be further extended to natively support features like Dynamic Bayesian Network-based simulation of time series, as well as nested and intersecting plates structures for complex modelling scenarios [4]. More important than individual features is the overall ability to exploit DAG structures to improve transparency and code modularization. The examples of simulating shapes in images and patterns in biosequences are but two exemplifications of DagSim's advantages. While existing frameworks also allow to define transparent simulations in settings with standard functional relations between numeric scalars or vectors, only DagSim allows transparency and code modularization in a broad range of settings with complex data types and functional relations.

Supporting information

S1 Fig. DAG for use case I.
(TIF)

S2 Fig. DAG for use case II.
(TIF)

Acknowledgments

We wish to thank Victor Greiff and Anne H Schistad Solberg for their input on the manuscript text, and we wish to thank Milena Pavlović and Knut Rand for feedback after trying out the software.

Author Contributions

Conceptualization: Ghadi S. Al Hajj, Johan Pensar, Geir K. Sandve.

Software: Ghadi S. Al Hajj.

Supervision: Johan Pensar, Geir K. Sandve.

Writing – original draft: Ghadi S. Al Hajj, Geir K. Sandve.

Writing – review & editing: Ghadi S. Al Hajj, Johan Pensar, Geir K. Sandve.

References

1. Morris TP, White IR, Crowther MJ. Using simulation studies to evaluate statistical methods. Statistics in Medicine. 2019; 38(11):2074–102. https://doi.org/10.1002/sim.8086 PMID: 30652356
2. Schuler A, Jung K, Tibshirani R, Hastie T, Shah N. Synth-Validation: Selecting the Best Causal Inference Method for a Given Dataset. arXiv:171100083 [stat] [Internet]. 2017 Oct 31 [cited 2022 Jan 27]; Available from: http://arxiv.org/abs/1711.00083
3. Sandve GK, Greiff V. Access to ground truth at unconstrained size makes simulated data as indispensable as experimental data for bioinformatics methods development and benchmarking. Bioinformatics. 2022 Sep 8;btac612. https://doi.org/10.1093/bioinformatics/btac612 PMID: 36073940
4. Koller D, Friedman N. Probabilistic graphical models: principles and techniques. Cambridge, MA: MIT Press; 2009. 1231 p. (Adaptive computation and machine learning).
5. Pearl J. Causality [Internet]. 2nd ed. Cambridge: Cambridge University Press; 2009 [cited 2021 Nov 21]. Available from: https://www.cambridge.org/core/books/causality/B0046844FAE10CBF274D4ACBDAEB5F5B

6. Kratzer G, Lewis FI, Comin A, Pittavino M, Furrer R. Additive Bayesian Network Modelling with the R Package abn. arXiv:191109006 [cs, stat] [Internet]. 2019 Nov 20 [cited 2022 Feb 6]; Available from: http://arxiv.org/abs/1911.09006
7. Lagani V, Athineou G, Farcomeni A, Tsagris M, Tsamardinos I. Feature Selection with the R Package MXM: Discovering Statistically Equivalent Feature Subsets. Journal of Statistical Software. 2017 Sep 5; 80:1–25.
8. Rosseel Y. lavaan: An R Package for Structural Equation Modeling. Journal of Statistical Software. 2012 May 24; 48:1–36.
9. Scutari M. Learning Bayesian Networks with the bnlearn R Package. Journal of Statistical Software. 2010 Jul 16; 35:1–22.
10. Textor J, van der Zander B, Gilthorpe MS, Liśkiewicz M, Ellison GTH. Robust causal inference using directed acyclic graphs: the R package 'dagitty.' Int J Epidemiol. 2017 Jan 15;dyw341.
11. Sofrygin O, Laan MJ van der, Neugebauer R simcausal R Package: Conducting Transparent and Reproducible Simulation Studies of Causal Effect Estimation with Complex Longitudinal Data. Journal of Statistical Software. 2017 Oct 16; 81:1–47. https://doi.org/10.18637/jss.v081.i02 PMID: 29104515
12. Lin I. simMixedDAG [Internet]. GitHub. [cited 2022 Feb 6]. Available from: https://github.com/IyarLin/simMixedDAG
13. Prakash E, Shrikumar A, Kundaje A. Towards More Realistic Simulated Datasets for Benchmarking Deep Learning Models in Regulatory Genomics [Internet]. bioRxiv; 2021 [cited 2022 Jan 27]. p. 2021.12.26.474224. Available from: https://www.biorxiv.org/content/10.1101/2021.12.26.474224v1
14. Bengio Y. Deep Learning of Representations for Unsupervised and Transfer Learning. In: Proceedings of ICML Workshop on Unsupervised and Transfer Learning [Internet]. JMLR Workshop and Conference Proceedings; 2012 [cited 2022 Jan 29]. p. 17–36. Available from: https://proceedings.mlr.press/v27/bengio12a.html
15. Bengio Y, Courville A, Vincent P. Representation Learning: A Review and New Perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2013 Aug; 35(8):1798–828. https://doi.org/10.1109/TPAMI.2013.50 PMID: 23787338
16. Ankan A, Panda A. pgmpy: Probabilistic Graphical Models using Python. In Austin, Texas; 2015 [cited 2022 Apr 24]. p. 6–11. Available from: https://conference.scipy.org/proceedings/scipy2015/ankur_ankan.html
17. Breitling LP. dagR: A Suite of R Functions for Directed Acyclic Graphs. Epidemiology. 2010 Jul; 21 (4):586–7. https://doi.org/10.1097/EDE.0b013e3181e09112 PMID: 20539116
18. Sani N, Malinsky D, Shpitser I. Explaining the Behavior of Black-Box Prediction Algorithms with Causal Learning. arXiv:200602482 [cs, stat] [Internet]. 2020 Jun 3 [cited 2021 Dec 28]; Available from: http://arxiv.org/abs/2006.02482
19. Britanova OV, Putintseva EV, Shugay M, Merzlyak EM, Turchaninova MA, Staroverov DB, et al. Age-related decrease in TCR repertoire diversity measured with deep and normalized sequence profiling. J Immunol. 2014 Mar 15; 192(6):2689–98. https://doi.org/10.4049/jimmunol.1302064 PMID: 24510963
20. Trück J, Eugster A, Barennes P, Tipton CM, Luning Prak ET, Bagnara D, et al. Biological controls for standardization and interpretation of adaptive immune receptor repertoire profiling. Cowell L, Taniguchi T, editors. eLife. 2021 May 26;10:e66274.
21. Barennes P, Quiniou V, Shugay M, Egorov ES, Davydov AN, Chudakov DM, et al. Benchmarking of T cell receptor repertoire profiling methods reveals large systematic biases. Nat Biotechnol. 2021 Feb; 39 (2):236–45. https://doi.org/10.1038/s41587-020-0656-3 PMID: 32895550
22. Kanduri C, Pavlović M, Scheffer L, Motwani K, Chernigovskaya M, Greiff V, et al. Profiling the baseline performance and limits of machine learning models for adaptive immune receptor repertoire classification [Internet]. bioRxiv; 2021 [cited 2022 Apr 17]. p. 2021.05.23.445346. Available from: https://www.biorxiv.org/content/10.1101/2021.05.23.445346v2
23. Pavlović M, Scheffer L, Motwani K, Kanduri C, Kompova R, Vazov N, et al. The immuneML ecosystem for machine learning analysis of adaptive immune receptor repertoires. Nat Mach Intell. 2021 Nov; 3 (11):936–44.
24. Pavlović M, Al Hajj GS, Pensar J, Wood M, Sollid LM, Greiff V, et al. Improving generalization of machine learning-identified biomarkers with causal modeling: an investigation into immune receptor diagnostics. arXiv:220409291 [cs, q-bio] [Internet]. 2022 Apr 20 [cited 2022 Apr 24]; Available from: http://arxiv.org/abs/2204.09291
25. Schölkopf B, Janzing D, Peters J, Sgouritsa E, Zhang K, Mooij J. On causal and anticausal learning. In: Proceedings of the 29th International Coference on International Conference on Machine Learning. Madison, WI, USA: Omnipress; 2012. p. 459–66. (ICML'12).

26. Bareinboim E, Pearl J. Causal inference and the data-fusion problem. PNAS. 2016 Jul 5; 113 (27):7345–52. https://doi.org/10.1073/pnas.1510507113 PMID: 27382148

27. Träuble F, Creager E, Kilbertus N, Locatello F, Dittadi A, Goyal A, et al. On Disentangled Representations Learned from Correlated Data. In: Proceedings of the 38th International Conference on Machine Learning [Internet]. PMLR; 2021 [cited 2022 Apr 6]. p. 10401–12. Available from: https://proceedings.mlr.press/v139/trauble21a.html

28. Wang Y, Jordan MI. Desiderata for Representation Learning: A Causal Perspective. arXiv:210903795 [cs, stat] [Internet]. 2022 Feb 10 [cited 2022 Apr 6]; Available from: http://arxiv.org/abs/2109.03795

29. Prosperi M, Guo Y, Sperrin M, Koopman JS, Min JS, He X, et al. Causal inference and counterfactual prediction in machine learning for actionable healthcare. Nat Mach Intell. 2020 Jul; 2(7):369–75.

30. Balzer LB, Petersen ML. Invited Commentary: Machine Learning in Causal Inference—How Do I Love Thee? Let Me Count the Ways. American Journal of Epidemiology. 2021 Aug 1; 190(8):1483–7. https://doi.org/10.1093/aje/kwab048 PMID: 33751059

31. Subbaswamy A, Schulam P, Saria S. Preventing Failures Due to Dataset Shift: Learning Predictive Models That Transport. In: Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics [Internet]. PMLR; 2019 [cited 2022 Jan 31]. p. 3118–27. Available from: https://proceedings.mlr.press/v89/subbaswamy19a.html