arfpy: A Python Package for Density Estimation and Generative Modeling with Adversarial Random Forests

KRISTIN BLESCH (iD)

MARVIN N. WRIGHT (iD)

*Author affiliations can be found in the back matter of this article

]u[ubiquity press

ABSTRACT

This paper introduces `arfpy`, a python implementation of Adversarial Random Forests, which is a lightweight procedure for synthesizing new data that resembles some given data. The software `arfpy` equips practitioners with straightforward functionalities for both density estimation and generative modeling. The method is particularly useful for tabular data and its competitive performance is demonstrated in previous literature. As a major advantage over the mostly deep learning based alternatives, `arfpy` combines the method's reduced requirements in tuning efforts and computational resources with a user-friendly python interface. This supplies audiences across scientific fields with software to generate data effortlessly.

https://github.com/bips-hb/arfpy.

CORRESPONDING AUTHOR:

Marvin N. Wright

Leibniz Institute for Prevention Research and Epidemiology – BIPS, Bremen, Germany; Faculty of Mathematics and Computer Science, University of Bremen, Bremen, Germany; Department of Public Health, University of Copenhagen, Copenhagen, Denmark

wright@leibniz-bips.de

Kristin Blesch

Leibniz Institute for Prevention Research and Epidemiology – BIPS, Bremen, Germany; Faculty of Mathematics and Computer Science, University of Bremen, Bremen, Germany

blesch@leibniz-bips.de

INTRODUCTION

Generative modeling is a challenging task in machine learning that aims to synthesize new data which is similar to a set of given data. State of the art are computationally intense and tuning-heavy algorithms such as generative adversarial networks (GANs) [1, 2], variational autoencoders [3], normalizing flows [4], diffusion models [5] or transformer-based models [6]. A much more lightweight procedure is to use an Adversarial Random Forest (ARF) [7]. ARFs achieve competitive performance in generative modeling with much faster runtime [7], yet they do not require the practitioner to have extensive knowledge of generative modeling.

Further, ARFs are especially useful for data that comes in a table format, i.e., tabular data. That is because ARFs are based on random forests [8] which leverage the advantages that tree-based methods have over neural networks on tabular data [9] for generative modeling. Further, as part of the procedure, ARFs give access to the estimated joint density, which is useful for several other fields of research, e.g., unsupervised machine learning. For the task of density estimation, ARFs have been demonstrated to yield remarkable results as well [7]. In brief, ARFs are a promising methodological contribution to the field of generative modeling and density estimation, providing a ready-made solution to generate data for practitioners across fields.

ARFs have already gained some attention in the scientific community [10], however, the paper by Watson et al. [7] provides the audience with a R software package only. The machine learning and generative modeling community, however, is mostly using python as a programming language and to reach a broad audience more generally, a fast and user-friendly implementation of ARFs in python is highly desirable. We aim to fill this gap with the presented python implementation of ARFs.

`arfpy` is inspired by the R implementation called *arf* [11], but transfers the algorithmic structure to match the class-based structure of python code and takes advantage of computationally efficient python functions. Similar to the R implementation, separate functions for first fitting the density (FORDE algorithm [7]) and then

generating new data samples (FORGE algorithm [7]) exist. However, in `arfpy`, the functions are called for an initialized object of class `arf`, which is showcased in the usage example below.

Crucially, for practitioners working with python as programming language, the direct python implementation is more robust and convenient to users than calling fragile wrappers like `rpy2` [12] that aim to make R code running in python. The benefits of a direct python implementation of ARFs for the generative modeling community have already been recognized by now. For example, `arfpy` is integrated in the data synthesizing framework `synthcity` [13].

IMPLEMENTATION AND ARCHITECTURE

MODULE DESIGN
The general workflow of generating data with `arfpy` is (1) to initialize an object of class `arf` with real data, (2) estimate the density and (3) sample new data. This procedure is visualized in Figure 1.

The architecture of `arfpy` reflects this workflow and we have class `arf` building the backbone of the procedure. An instance of class `arf` takes the real data set as input and trains an ARF, i.e., learns the actual data's structure. To this object, functions to estimate the density (FORDE algorithm [7], function `forde()`) and generate data (FORGE algorithm [7], function `forge()`) can be applied. This architecture allows users to learn the structure of the real data once (when initializing the `arf` class object) and then flexibly adapt density estimation, e.g., using different parameters, or repeatedly sampling new data without having to refit the model.

METHODOLOGY OVERVIEW
For interested readers, we want to briefly describe the methodological foundations of ARFs, but refer to [7] for further details. From a given real data set, first, naive synthetic data is generated (initial generation step) by sampling from the marginal distributions of the features. Then, a random forest [8] is fit to distinguish this
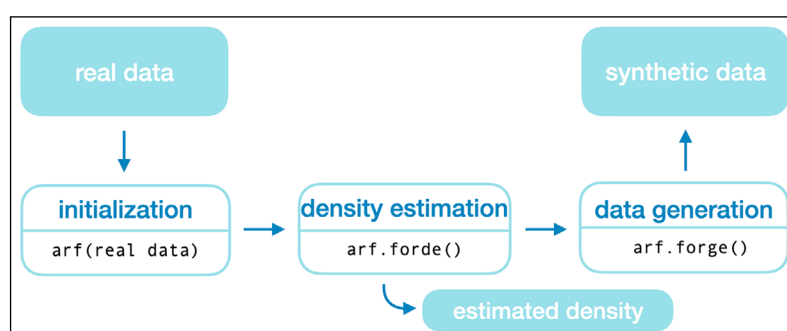
Figure 1 Workflow of `arfpy`'s core functionalities.

synthetic from the real data (initial discrimination step). This procedure, also known as fitting an unsupervised random forest [14], guides the random forest to learn the dependency structure in the data. Using this forest, we can sample observations from the leaves of the trees to generate updated synthetic data (generation step). Subsequently, a new random forest is fit to differentiate between synthetic and real data (discrimination step). Drawing on the adversarial idea of GANs, this iterative procedure of data generation and discrimination will be repeated until the discriminator cannot distinguish between generated and real data anymore. At this stage, the accuracy of the forest will be ≤0.5 and the forest is assumed to have converged, which implies mutually independent features in the terminal nodes. This drastically simplifies density estimation and generative modeling, as it allows us to formulate the univariate density for each feature separately with data in the leaves of the fitted ARF (FORDE algorithm) and then combine them to the joint density, instead of having to model a multivariate density directly. For data generation, we can use this trait to sample a new observation by drawing a leaf from the forest of the last iteration step and use the data distributions with parameters estimated from that leaf to sample each feature separately (FORGE algorithm).

EXAMPLE USAGE

Let us illustrate the usage of `arfpy` with a visually intuitive example: We create data using `make_moons` from `sklearn.datasets`, which results in data along two continuous axes that looks like two moons from different categories. Statistically speaking, this is a tabular dataset, consisting of both continuous and categorical features that exhibit a dependency structure. For a more intuitive understanding of the data, see Figure 2, Panel **A**. The task of `arfpy` is to learn the structure of this given (real) data and generate new data instances that appear similar.

   To initialize the workflow, we need to run relevant imports, including the import of class `arf` from the `arfpy` module, and create the real dataset. The `arf` class takes a `pandas DataFrame` as input, so the real

data is pre-processed to match this requirement. This incorporates setting unique column names (`'dim_1'`, `'dim_2'`,`'label'`) and ensuring that `'label'` is stored in the correct data type `'category'`.

```
 1  import pandas as pd
 2  from sklearn.datasets import make_moons
 3  from arfpy import arf
 4
 5  moons_X, moons_y = make_moons(n_samples =
       3000, noise = 0.1)
 6  df = pd.DataFrame({"dim_1" : moons_X[:,0],
       "dim_2" : moons_X[:,1], "label" : moons_y})
 7  df['label'] = df['label'].astype('category')
 8
 9  df.head()
10
11  #>  dim_1       dim_2       label
12  #>  1.782717    0.099124    1
13  #>  1.087497    0.298744    0
14  #> -0.576695    0.801675    0
15  #>  0.623931   -0.506896    1
```

With the real dataset preprocessed as needed, we can proceed with training the ARF to learn the data's structure. Creating an object of class `arf` will trigger ARF model fitting using the data provided.

```
1  my_arf = arf.arf(x = df)
2
3  #> Initial accuracy is 0.82
4  #> Iteration number 1 reached accuracy of 0.36
```

Because we have used the parameter default `verbose = True`, the training of `my_arf` prints out some interesting information: The initial accuracy, which corresponds to the accuracy of the random forest in distinguishing real data from naive synthetic data, is 0.82. This implies that the random forest is doing very well in distinguishing real from naive synthetic data and therefore, we can assume the model to have learned relevant dependencies that allow the model to make this distinction. Using this forest to sample updated synthetic data, and fitting a new random forest to distinguish this data from real data leads
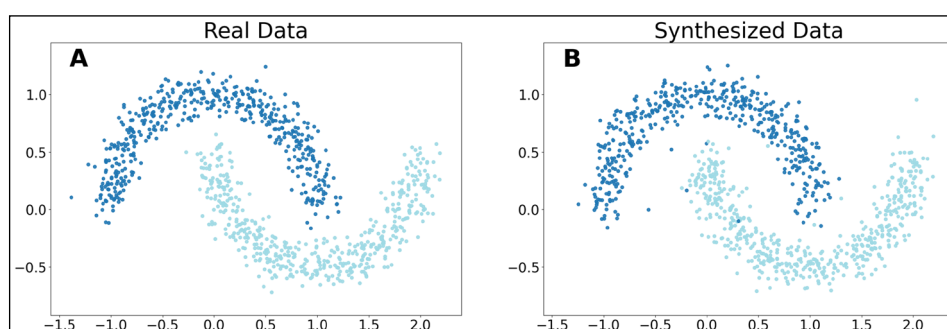

Figure 2 Comparison of real and synthesized data.

to an accuracy of only 0.36. This accuracy is below the default threshold of 0.5, so loosely speaking, the synthetic data generated with the forest cannot be accurately distinguished from real data, i.e., the generated data looks like real data, which is the goal the algorithm was aiming for. In other words, the relevant dependency structures of the real data have been learned by the forest in the first round of iteration already, so the algorithm has converged and no further iterations need to be conducted.

After the ARF has converged, we can proceed to estimating the joint density. Recap that in a converged ARF, the features are mutually independent in the leaves, which simplifies the challenging multivariate density estimation task into many simple univariate density estimation tasks. The joint density is then a factorization of the individual density estimates across leaves in the ARF. We can call function `forde()` on the `my_arf` object to estimate the density and store the returned dictionary to explore the parameters. The `FORDE` dictionary contains the estimated parameters for continuous (key `'cnt'`) and categorical features (key `'cat'`). As mentioned in the above paragraph, the parameters are estimated using the data points in the forest's leaves, so we will get estimates for each leaf individually. The parameters for the categorical features simply correspond to the empirical frequency of categories in the leaves, so for a more complex example, we can take a look at the continuous feature's parameter estimates in `FORDE['cnt']`. We have used the default distribution (truncated normal distribution) to model continuous features, so the output will reflect estimates for the mean and standard deviation for each continuous feature (`'dim_1', 'dim_2'`) in each leaf, which is uniquely identified by `'tree'` and `'nodeid'`:

```
1 FORDE = my_arf.forde()
2
3 FORDE['cnt'].iloc[:,:5].head()
4
5 #> tree  nodeid  variable  mean       sd
6 #> 0     3       dim_1      0.961437  0.214925
7 #> 0     3       dim_2     -0.671571  0.028193
8 #> 0     11      dim_1      1.040565  0.185581
9 #> 0     11      dim_2     -0.621924  0.003328
```

With the parameters estimated, we can move on to the final step of the generative modeling task and sample new data instances with the function `forge()`.

For each instance to be generated, the function randomly samples a leaf from the forest with weighted probability according to the coverage of real data in the leaves of the ARF and then uses the parameters estimated through `forde()` to sample a new data instance.

```
1 df_syn = my_arf.forge(n = 1000)
2
3 df_syn.head()
4
5 #> dim_1      dim_2      label
6 #> -0.018004   0.283963   1
7 #> 1.734200   -0.085115   1
8 #> -0.009840   1.046872   0
9 #> 0.868400   -0.352692   1
```

Calling `forge()` completes the task of generating synthetic data that mimics real data. From the generated data table itself, the similarity is hard to grasp, but we can visually inspect the quality of the synthetic data in Figure 2.

QUALITY CONTROL

The software has been tested through unit tests, which includes testing of relevant functionalities with various input data sets. The workflow of running these tests is automated on GitHub actions, but can be run locally and with customized data sets using the instructions provided in the software repository. Further, the repository allows to publicly raise questions or report bugs and gives clear guidelines on how to contribute to the open source software project are lined out.

AVAILABILITY

OPERATING SYSTEM
Platform Independent

PROGRAMMING LANGUAGE
Python ≥ 3.8

ADDITIONAL SYSTEM REQUIREMENTS
No specific requirements

DEPENDENCIES
numpy ≥ 1.20.3, pandas ≥ 1.5, scikit-learn ≥ 0.24, scipy ≥ 1.4

LIST OF CONTRIBUTORS
Blesch, Kristin[a, b];
Wright, Marvin N.[a, b, c];

(a) Leibniz Institute for Prevention Research and Epidemiology – BIPS, Bremen, Germany;
(b) Faculty of Mathematics and Computer Science, University of Bremen, Bremen, Germany;
(c) Department of Public Health, University of Copenhagen, Copenhagen, Denmark

SOFTWARE LOCATION
Archive
Name: arfpy
Persistent identifier: https://pypi.org/project/arfpy/
Licence: MIT

LANGUAGE

English

REUSE POTENTIAL

ARFs have been introduced with a solid theoretical background, yet do not have to compromise on a complex algorithmic structure and instead are a low-key algorithm that does not require extensive hyperparameter tuning [7]. In contrast to the typically deep learning based alternatives, ARF does not require background knowledge of generative modeling, intense tuning efforts or large computational resources. Given the theoretical foundation and straightforward implementation with `arfpy`, the methodology is attractive for both scholars conducting rather theoretical research in statistics, e.g., density estimation, as well as practitioners from other fields that need to generate new data samples.

Typical use cases of such synthesized data samples are, for example, the imputation of missing values, data augmentation or the conduct of analyses that respect data protection rules. With the specialty of ARFs being particularly suitable for tabular data, including a natural incorporation of both continuous and categorical features, the straightforward python implementation of ARFs provides a convenient algorithm to a broad audience from different fields.

With the python programming language being widespread, `arfpy` can smoothly integrate in the code of various applications. Further, usability is enhanced by the intuitive documentation provided at https://bips-hb.github.io/arfpy/, making `arfpy` an easily accessible tool to generate data.

In sum, `arfpy` introduces density estimation and generative modeling with ARFs to python, which enables practitioners from a wide variety of fields to generate fast and reliable synthetic data and density estimates with python as a programming language.

ACKNOWLEDGEMENTS

FUNDING INFORMATION

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR AFFILIATIONS

Kristin Blesch ⓘ orcid.org/0000-0001-6241-3079
Leibniz Institute for Prevention Research and Epidemiology – BIPS, Bremen, Germany; Faculty of Mathematics and Computer Science, University of Bremen, Bremen, Germany

Marvin N. Wright ⓘ orcid.org/0000-0002-8542-6291
Leibniz Institute for Prevention Research and Epidemiology – BIPS, Bremen, Germany; Faculty of Mathematics and Computer Science, University of Bremen, Bremen, Germany; Department of Public Health, University of Copenhagen, Copenhagen, Denmark

REFERENCES

1. **Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y.** Generative adversarial nets. *Advances in Neural Information Processing Systems.* 2014; 27.

2. **Xu L, Skoularidou M, Cuesta-Infante A, Veeramachaneni K.** Modeling tabular data using conditional GAN. *Advances in Neural Information Processing Systems.* 2019; 32.

3. **Kingma DP, Welling M.** Auto-encoding variational bayes. *International Conference on Learning Representations;* 2014.

4. **Rezende D, Mohamed S.** Variational inference with normalizing flows. *Proceedings of the 32nd International Conference on Machine Learning.* 2015; 37: 1530–1538.

5. **Ho J, Jain A, Abbeel P.** Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems.* 2020; 33: 6840–6851.

6. **Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I.** Attention is all you need. *Advances in Neural Information Processing Systems.* 2017; 30.

7. **Watson DS, Blesch K, Kapar J, Wright MN.** Adversarial random forests for density estimation and generative modeling. *International Conference on Artificial Intelligence and Statistics. PMLR.* 2023; 206: 5357–5375.

8. **Breiman L.** Random forests. *Machine learning.* 2001; 45: 5–32. DOI: https://doi.org/10.1023/A:1010933404324

9. **Grinsztajn L, Oyallon E, Varoquaux G.** Why do tree-based models still outperform deep learning on typical tabular data? *Advances in Neural Information Processing Systems.* 2022; 35: 507–520.

10. **Nock R, Guillame-Bert M.** Generative forests. *arXiv Preprint 2308.03648*; 2023. DOI: https://doi.org/10.48550/arXiv.2308.03648

11. **Wright MN, Watson DS.** *arf: Adversarial Random Forests;* 2023. URL https://CRAN.R-project.org/package=arf. R package version 0.1.3.

12. **Gautier L,** et al. rpy2: Python-r bridge. *GitHub repository;* 2023. URL https://github.com/rpy2/rpy2.

13. **van der Schaar Lab.** synthcity: A library for generating and evaluating synthetic tabular data. *GitHub repository;* 2023. URL https://github.com/vanderschaarlab/synthcity.

14. **Shi T, Horvath S.** Unsupervised learning with random forest predictors. *Journal of Computational and Graphical Statistics.* 2006; 15(1): 118–138. DOI: https://doi.org/10.1198/106186006X94072