XI RETREAT GRBIO 2026
Vilanova i la Geltrú
July 1st, 2026

Generating Synthetic Data via Archetypal and Archetypoid Analyses

AA/ADA-Dirichlet-Synthetic generator: code acceleration, PC* selection, and simulation validation

Liukuan Yu (Wellk)
Supervisors: Jordi Cortés Martínez and Daniel Fernández Martínez
Universitat Politècnica de Catalunya - BarcelonaTech (UPC)

OVERVIEW

Presentation structure

01	**Review**	Synthetic data, AA/ADA, Dirichlet sampling
02	**What was done before**	Initial AA/ADA-Dirichlet pipeline and pilot cases
03	**Post-RP work**	R implementation, C++ acceleration, and PC*
04	**Simulation study**	18 scenarios, SPECKS, synthpop comparison
05	**Next steps**	Stabilise validation, extend ADA, and move to real data

PART 01

Short review

Synthetic data and the AA/ADA-Dirichlet idea

REVIEW

What is synthetic data?

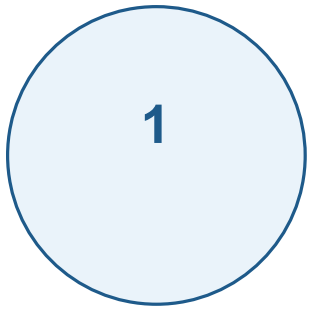
Synthetic data (SD)

Synthetic data is not real data , but it has the same statistical properties

Shareable when OD cannot be released
Useful for method testing and reproducibility
Can support small-sample or imbalanced-data settings

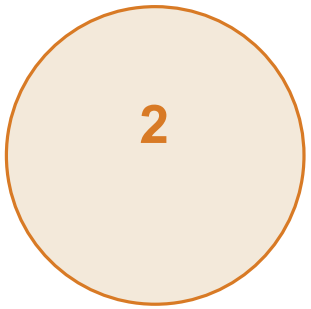
Key condition: SD must be difficult to distinguish from OD for the intended use case.

What is "good" synthetic data?



Fidelity

Preserves distributions, dependence structure, and key summaries.



Utility

Leads to similar analytical conclusions and model performance.

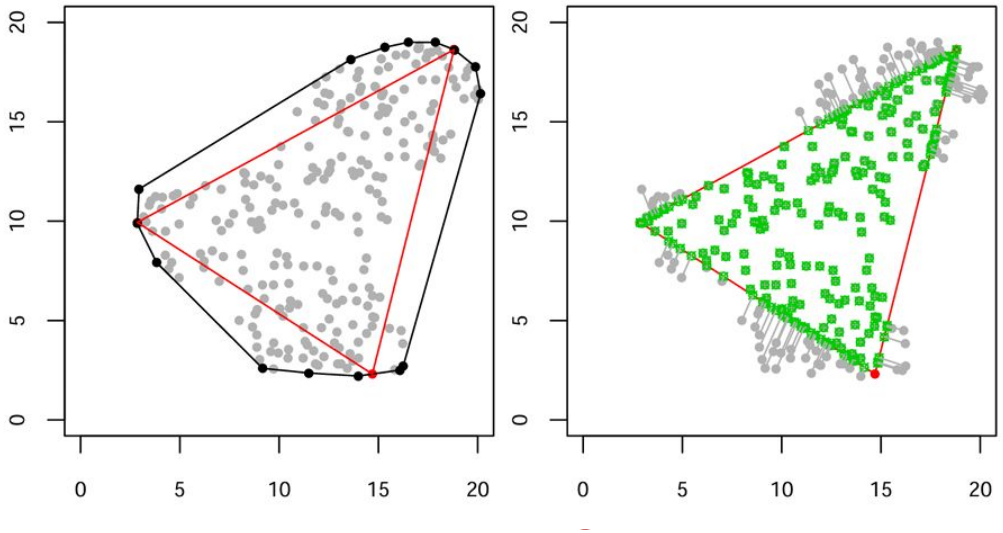
Privacy

Reduces disclosure risk and avoids releasing individual records.

My current validation focuses mainly on resemblance / distinguishability, using SPECKS rejection rate.

Archetypal Analysis (AA): extremes

AA represents each observation as a convex mixture of a few archetypes.



Formulation:

Minimize $\sum_{i=1}^{n} ||\boldsymbol{x_i} - \sum_{j=1}^{k} \alpha_{ij} \boldsymbol{z_j}||^2 = \sum_{i=1}^{n} ||\boldsymbol{x}_i - \sum_{j=1}^{k} \alpha_{ij} \sum_{l=1}^{n} \beta_{jl} \boldsymbol{x}_l||^2$

Under the constraints

1) $\sum_{j=1}^{k} \alpha_{ij} = 1$ with $\alpha_{ij} \geq 0$ for $i = 1, \ldots, n$

2) $\sum_{l=1}^{n} \beta_{jl} = 1$ with $\beta_{jl} \geq 0$ for $j = 1, \ldots, k$

Archetypes are convex combinations of observed data.
They usually lie near the boundary of the data cloud.
The alpha vector gives an interpretable mixture profile for each observation.

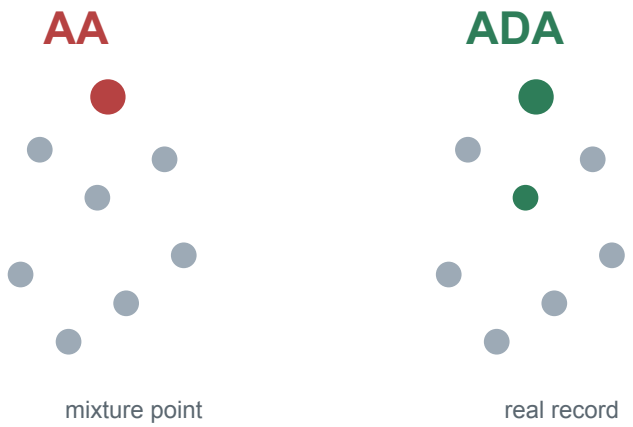
Cutler & Breiman (1994)

Archetypoid Analysis (ADA): real extreme cases

ADA keeps the same mixture idea but requires each prototype to be an actual observation.

AA: $\sum_{l=1}^{n} \beta_{jl} = 1$ with $\beta_{jl} \geq 0$ for $j = 1, \ldots, k$

ADA: $\sum_{l=1}^{n} \beta_{jl} = 1$ with $\beta_{jl} \in \{0,1\}$ for $j = 1, \ldots, k$



Stronger interpretability: selected archetypoids can be inspected directly.
Useful when “real representative extremes” are more meaningful than artificial mixtures.
Computation is harder because ADA combines continuous weights with discrete selection.

Vinué, Epifanio & Alemany (2015)

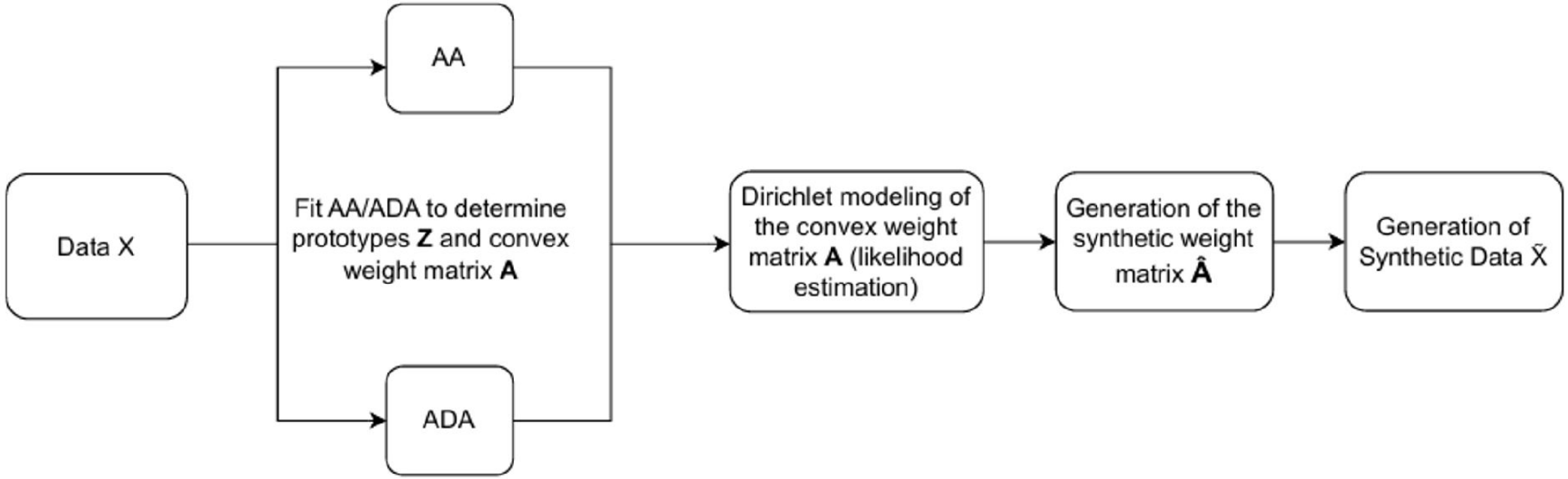
PART 02

What was done before

Initial concept, first pipeline, and pilot case studies

Baseline pipeline presented around the RP

The original RP established the core AA/ADA-Dirichlet generator and a first evaluation process



Pilot case studies: feasibility before scaling up

Case 1: AA-Dirichlet on Iris

2D geometry makes the convex-hull idea visible.
Synthetic points are generated through convex weights.
Residual augmentation was tested as a simple sensitivity variant.

Case 2: ADA-Dirichlet on NBA

Archetypoids are actual NBA players / profiles.
The method provides interpretable anchors for generation.

SPECKS was used as the first distinguishability diagnostic.

k	SPECKS	Threshold	H0 rejected	p-value
3	0.115	0.159	no	0.283
4	0.068	0.158	no	0.874
5	0.203	0.158	no	0.126
6	0.149	0.158	no	0.116
7	0.074	0.158	no	0.808
8	0.189	0.158	yes	0.010

Outcome Conceptually promising, but the original implementation was not yet fast enough for large simulation validation.

PART 03

Post-RP work

From a working method to a scalable validation workflow

Post-RP work: making the generator testable

We first solved the optimization stability problem, then accelerated the code, and finally used a PC* stopping rule to run the mini-simulation.

Optimization and implementation pipeline

Key clarification: not “R vs Python”, but NLSS/NNLS-type optimization vs gradient-based optimization.

1 Original R issue

NLSS / NNLS-type optimizer worked for small k, but could become numerically unstable when k was large, causing failed or unreliable RSS paths.

2 Python logic

Gradient-based optimization provided a more stable route for fitting AA and recovering the RSS path across k.

3 Back to R + C++

Stable logic translated back into R. C++ targeted the RSS and repeated matrix-operation bottlenecks, without changing numerical results.

4 PC* stopping rule

For each OD, compute RSS(k) and stop at the first k where:

$$PC(k) \leq 0.05$$

Mini-simulation

18 scenarios

n = 30, 100 · p = 3, 5, 10 · rho = 0, 0.3, 0.5

Compare AA-Dirichlet with synthpop CART using SPECKS rejection rate.

Code improvement: two problems solved

The original implementation had two practical barriers: some k values could fail, and the RSS path was too slow to support simulation.

1 IDENTIFY THE TWO BOTTLENECKS

A. Failure for larger k

Early R fitting used an NLSS / NNLS-type optimization routine. For larger k, the optimizer could become numerically unstable, so some fits did not return reliable RSS values.

B. Runtime bottleneck

Even when the fit worked, repeated RSS and matrix computations became increasingly slow as k increased. This made the simulation study impractical.

Solved

2 FIX THEM WITHOUT CHANGING THE RESULT

Stable optimization logic

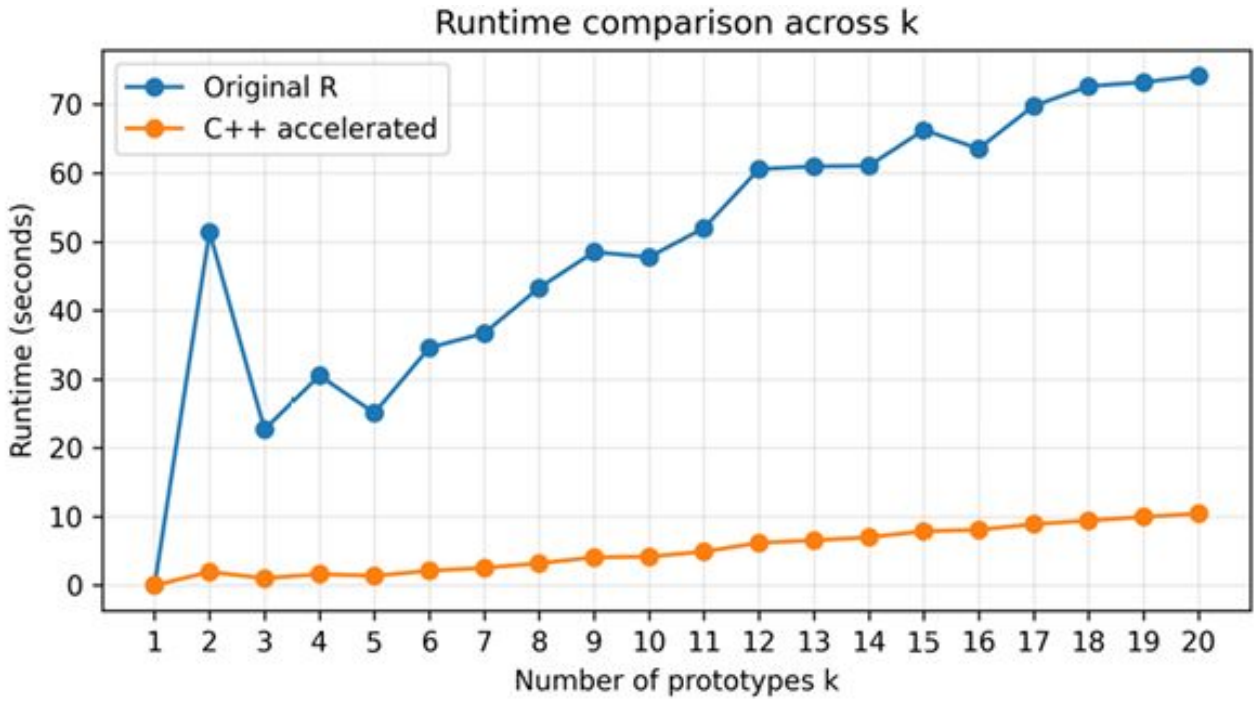
We used the more stable Python gradient-based logic, then translated it back into R line by line while preserving the same workflow and outputs.

C++ acceleration

We moved the computational bottleneck to C++, especially RSS and repeated matrix operations, and verified that the numerical result did not change.

Verification C++ version preserved the numerical output while reducing runtime, so the code became stable and fast enough for the simulation study

C++ acceleration: same result, much faster



9.83x
overall speedup

89.82%
runtime reduction

0
difference in RSS, A, Z

Quantity	Value
Original R total time	994.67 sec
C++ accelerated time	101.22 sec
Time saved	893.45 sec
Mean per-k speedup (k>1)	12.55x
Comparison for k=1,...,20	identical RSS, A, Z

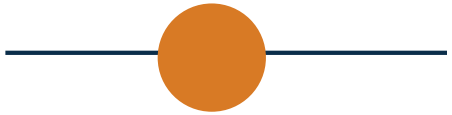
This made the later simulation study computationally feasible.

Why k selection became central

For AA/ADA-Dirichlet, k controls both reconstruction and generation.

Too small k

underfits OD structure
SD becomes too restricted



Too large k

more computation
may overfit / add instability

Goal

Choose a k that is rich enough for SD quality, but still stable, interpretable, and computationally manageable.

PC*: percentage change as a stopping rule

Instead of choosing k only from a visual elbow, we introduced a quantitative stopping criterion based on the relative improvement in RSS.

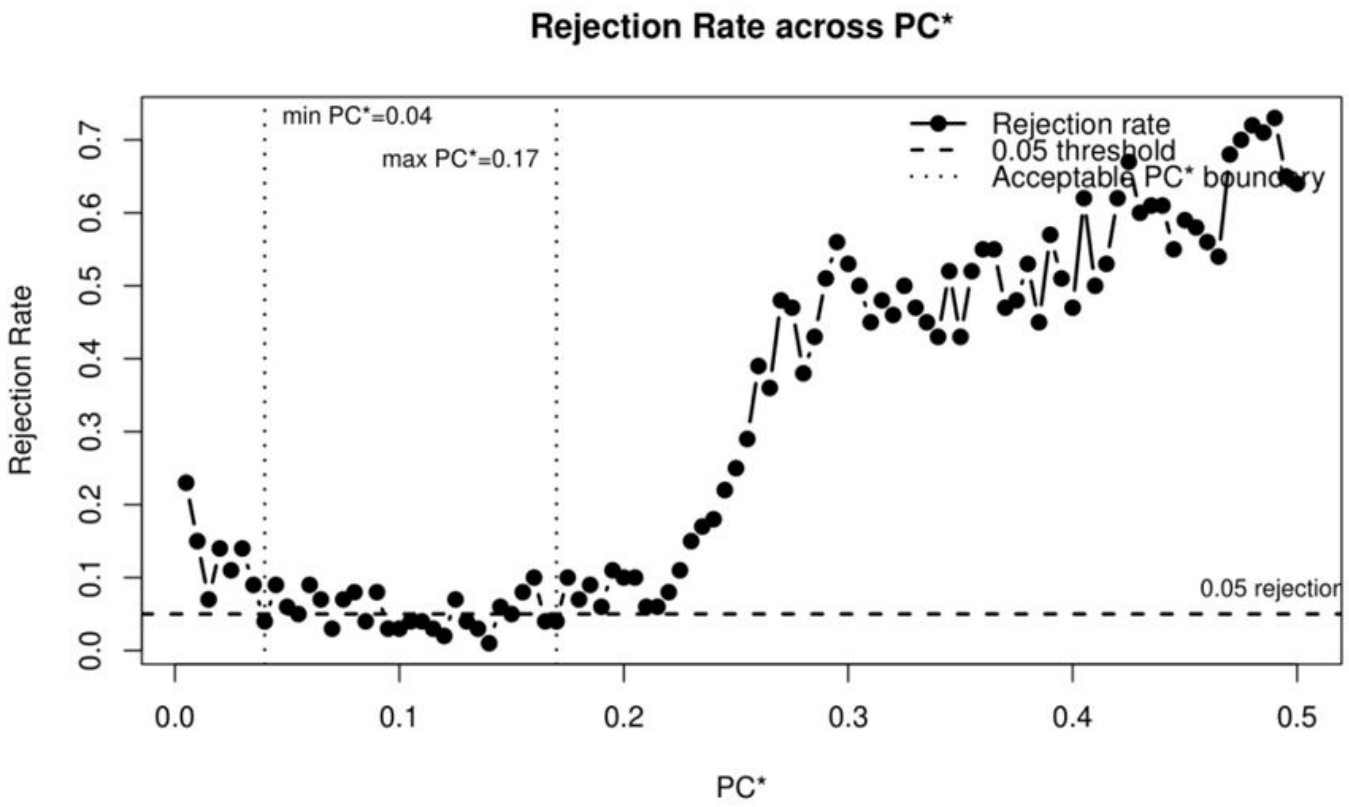
$$PC_k = \left| \frac{RSS_{k-1} - RSS_k}{RSS_{\cdot}} \right| , \; for\; k \geq 2$$

Stopping rule For a fixed threshold PC*, select the smallest k such that $PC_k \leq PC^*$.

Small PC*: strict rule, larger selected k.
Large PC*: early stopping, simpler selected model.

In our research, we are going to find the pc* which can make the rejection rate of specks in our method lower than 0.05 (type 1 error).

PC* tuning: n = 100 gives a usable region

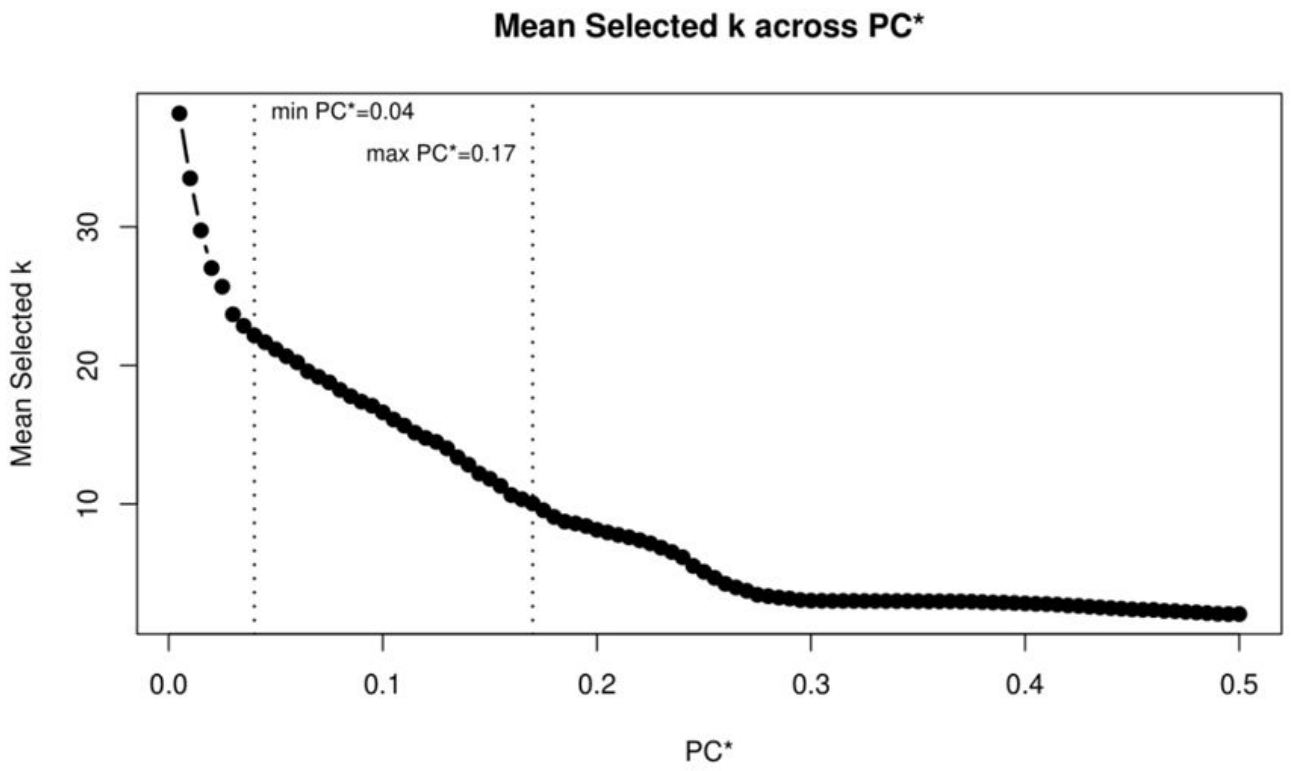


In the PC experiment with n = 100, the RSS paths for k ranging from 1 to 50 were first calculated for each OD (100 data sets which had already generated).

Subsequently, the stopping value of k was automatically selected for each PC*; synthetic data was then generated for that value of k, and the performance of the different PC* values was compared using the SPECKS rejection rate. This ultimately provided the basis for selecting PC* = 0.05.

Working choice for the next simulation: PC* = 0.05.

PC* tuning: n = 100 gives a usable region



Main interpretation

Acceptable PC* region: approximately 0.04 to 0.17.
Within this region, the SPECKS rejection rate stays close to the 0.05 target.

This supports using PC* as a model-complexity tuning mechanism.

Working choice for the next simulation: PC* = 0.05.

Large sample check: n = 1000 is much stricter

The n = 1000 runs were useful diagnostically, but they were not included in the final 18-scenario study because computation and validation became too demanding.

PC*	Rejection rate	Mean selected k	PC condition	Comment
0.0607	0.42	18.59	100/100	best observed, still high
0.0100	0.73	29.35	90/100	lower PC*, larger k
0.0010	0.96	46.71	19/100	near k upper limit

Stage 1 saved the full RSS/PC path for 100 OD datasets and k = 1,...,50: 5,000 RSS values.
Stage 2 was heavier: read large A/Z matrices, generate SD, then run SPECKS repeatedly.
SPECKS has higher power at n = 1000, so small OD-SD mismatches are detected more often.

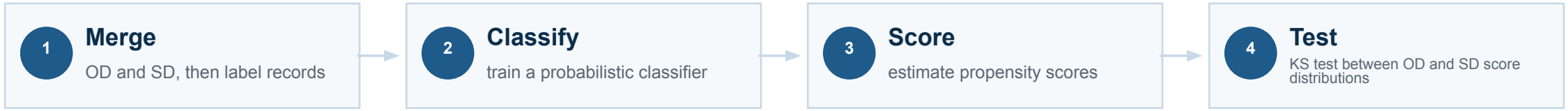
PART 04

Simulation study

Benchmarking the AA-Dirichlet generator against synthpop CART

Evaluation: SPECKS distinguishability test

SPECKS is a propensity-score Kolmogorov-Smirnov diagnostic.



H0 OD and SD are not distinguishable

H1 OD and SD are distinguishable

Lower rejection rate means SD is harder to distinguish from OD.

Simulation design: 18 scenarios

The goal is not to prove domination, but to test whether our interpretable generator is competitive with existing synthetic-data methods.

Scenario factors

18 SCENARIO DESIGN

Factors that define the simulation grid

1	Sample size	n = 30, 100
2	Dimension	p = 3, 5, 10
3	Correlation	rho = 0, 0.3, 0.5

2 × 3 × 3 = 18 scenarios

Methods / settings

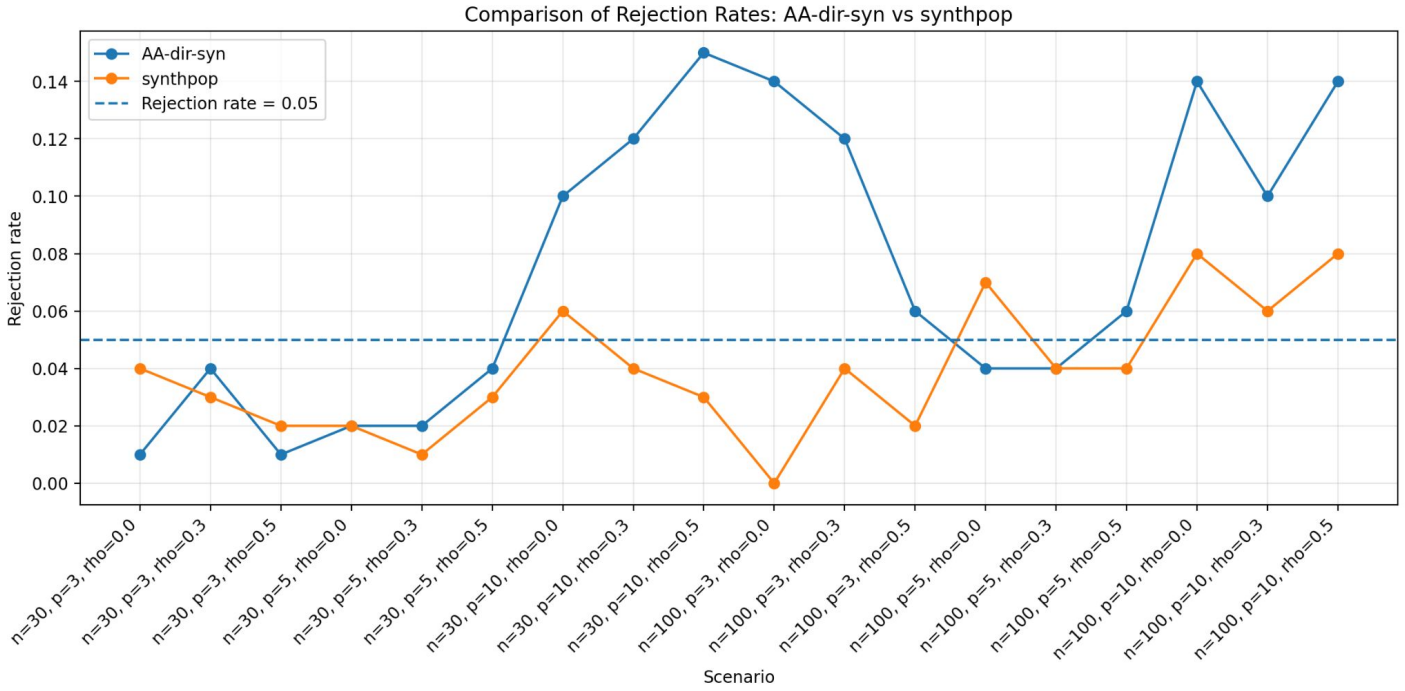
VS METHODS COMPARED

Data generation methods and fitting grid

A	Our method	AA/ADA–Dirichlet
B	Baseline	synthpop CART
k	Our k grid	k = 1, ..., 50

RESULT OF THE SIMULATION STUDY

Rejection rates show a mixed but encouraging picture: the goal was competitiveness, not domination.



Main takeaway

In some scenarios, AA-dir-syn is not worse than synthpop and can even be slightly better.

In other scenarios, its performance is more mixed and sometimes clearly worse.

So the method looks competitive, but not uniformly superior across all 18 scenarios.

These weaker cases highlight where the generator still needs further improvement.

Conclusion: our method already shows competitive behavior in part of the design space, but further optimization is still needed.

ROADMAP

Next steps after this stage

1. **Improve generation fidelity: residual augmentation + refined PC* selection**
2. **Extend the same validation to ADA**
3. **Paper writing**

Thank you

Questions and feedback

Liukuan Yu
Universitat Politècnica de Catalunya - BarcelonaTech (UPC)