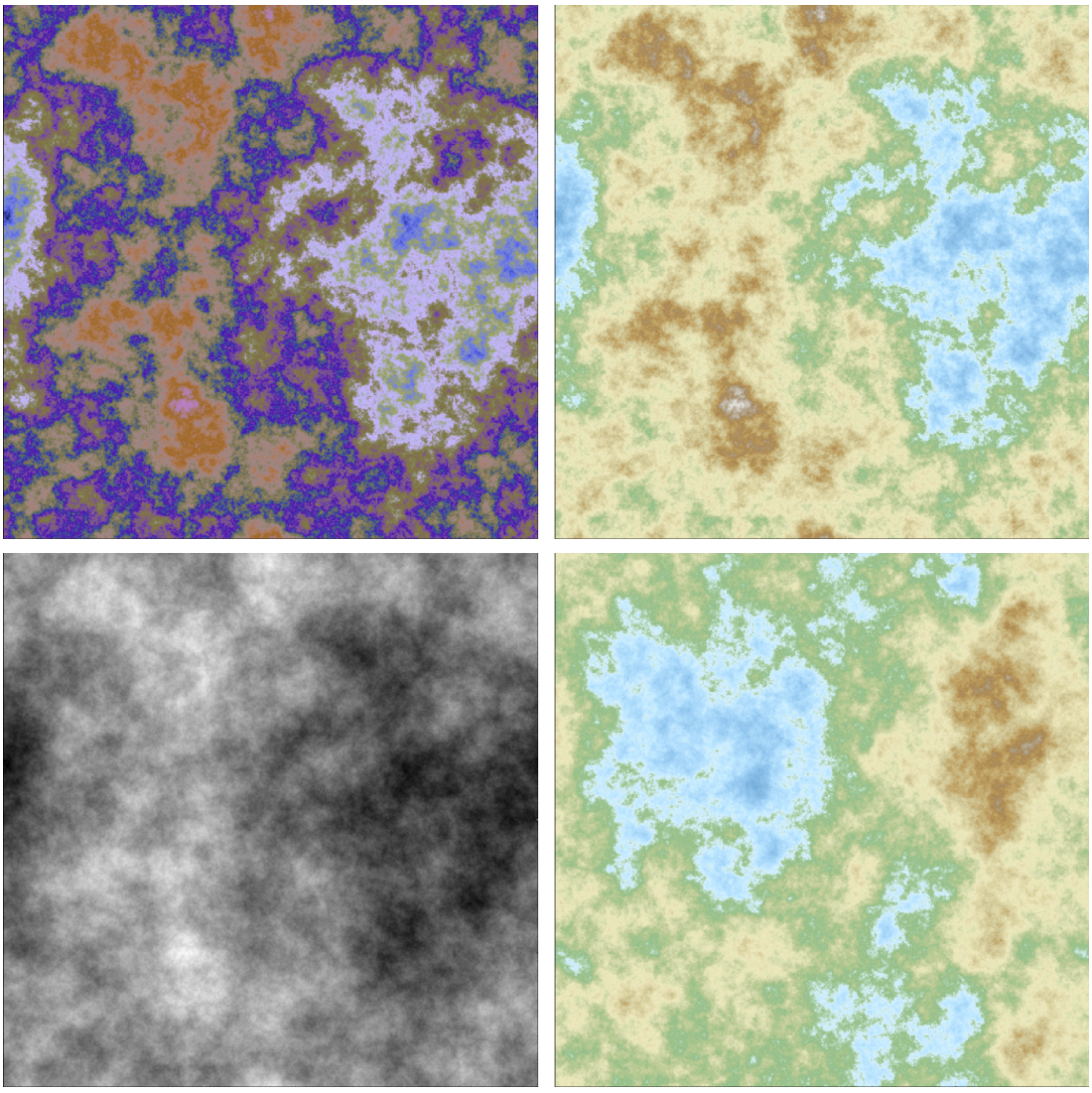
Synthetic Data and Generative AI



Vincent Granville, Ph.D. | www.MLTechniques.com | Version 4.0, March 2023

Preface

This book was first started in December 2022 and has been revised and augmented multiple times since the first writing. It now covers all the modern techniques on the subject as well as efficient proprietary methods developed by the author, with real industry use cases and Python implementations. The goal is to quickly help you pick up the right tool and run the code on your own dataset, in very little time. Yet the author provides enough background so that the reader understands all the aspects and interconnections of the methods involved, their strengths and weaknesses, potential enhancements, rule of thumbs, and best practices. Research-level material is also present throughout the book, and explained in simple English.

What is synthetic data and why use it?

Synthetic data is more than simulations, mimicking real data, fake (gibberish) data or noise injection to add variations to real data. It is defined by its usage and purpose. Four broad areas include:

- Data augmentation to produce richer training sets for predictive modeling; it leads to more robust predictions and reduced overfitting. For instance, to produce a better version of ChatGPT or better detection of cancer from medical images or tabular data.
- Generation of diversified data to test and benchmark machine learning algorithms, to identify their limits or to understand and improve black-box systems. Sensitivity testing fits in this category.
- Increasing security and compliance with data protection laws by strongly anonymizing data (especially for data sharing purposes), as well as reduction of algorithm bias impacting minorities.
- Data re-balancing in the presence of small segments (fraud / non-fraud, minority group), and smart data imputation. It is also useful in the presence of small samples with many features, when the data is difficult to obtain: for instance, clinical trials.

The data can be tabular (transactional), time series, graphs or consisting of images, videos, sound, text, spatial information or the result of agent-based systems. The goal is to identify and reproduce the structure (such as the autocorrelation function, shape, or correlation structure) rather than replicating the original data itself. In some instances (benchmarking), no real data is even needed.

Several techniques can be used for synthetization: GAN (generative adversarial networks), GMM (Gaussian mixture models) and other statistical models, interpolation, parametric noise with a target correlation structure, and more. Many metrics are available to assess the quality, be it cross-validation, ROC curves, statistical summaries, or Hellinger and related distances. All this material is reviewed in this book. In particular, chapter 10 discusses a GAN with replicable output especially designed for synthetization, illustrated on tabular data.

Book contents and target audience

This book covers the foundations of generative models and data synthetization. Emphasis is on scalability, automation, testing, optimizing, and interpretability (explainable AI). Models (including GMM, GAN and copulas) are often used to create rich synthetic data, augment real data, or to test and benchmark various methods. Many machine learning algorithms are revisited, simplified, unified, and generalized. For instance, regression techniques – including logistic and Lasso – are presented as a single method, without using advanced linear algebra. There is no need to learn 50 versions when one does it all and more. Confidence regions and prediction intervals are built using parametric bootstrap, without statistical models or probability distributions: it shows another usage of synthetization, with an application to meteorites shapes, for instance when the goal is to classify these celestial bodies.

With a focus on applications, synthetization and simulations, the book also covers clustering and classification, GPU machine learning, ensemble methods including an original boosting technique, elements of graph modeling, deep neural networks, auto-regressive and non-periodic time series, Brownian motions and related

processes, simulations, interpolation, strong random numbers, natural language processing (smart crawling, taxonomy creation and structuring unstructured data), computer vision (shapes generation and recognition), curve fitting, cross-validation, goodness-of-fit metrics, feature selection, curve fitting, gradient methods, optimization techniques and numerical stability.

Chapter 10 illustrates the use of copulas to produce synthetic data, applied to a well-known insurance dataset. It also features both GAN (generative adversarial networks) and copulas applied to an health industry data set, comparing results and showing how both methods can be blended for better synthetization and predictions, or even for data compression. Agent-based modeling and GIS applications are also covered, with interpolation techniques used for synthetization: fractal-like terrain generation with the diamond-square algorithm, disaggregation of of ocean tides time series, and geospatial interpolation of temperatures in the Chicago area.

Image and video generation include star clusters evolving over time and bound by gravity, providing potential scenarios about the past and future of our universe, or to synthesize collision graphs. It also allows you to explore alternative universes, for instance with negative masses. Chapters 16 and 17 are more advanced and may be skipped in introductory classes. The former focuses on point processes as a simple alternative to GMM. The later features synthetic multiplicative functions to discover new insights about a famous mathematical conjecture: the Riemann Hypothesis.

Methods are accompanied by enterprise-grade Python code, replicable datasets and visualizations, including data animations (gifs, videos, even sound done in Python). The code uses various data structures and library functions sometimes with advanced options. It constitutes a solid introduction to scientific programming. The code, datasets, spreadsheets and data visualizations are also on GitHub, here. Chapters are mostly independent from each other, allowing you to read in random order. A glossary, index and numerous cross-references make the navigation easy and unify all the chapters.

The style is very compact, getting down to the point quickly, and suitable to business professionals. Jargon and arcane theories are absent, replaced by simple English to facilitate the reading by non-experts, and to help you discover topics usually made inaccessible to beginners. While state-of-the-art research is presented in all chapters, the prerequisites to read this book are minimal: an analytic professional background, or a first course in calculus and linear algebra. The original presentation avoids all unnecessary math and statistics, yet without eliminating advanced topics. Finally, this book is the main reference for my course on synthetic data and generative AI.

About the author

Vincent Granville is a pioneering data scientist and machine learning expert, co-founder of Data Science Central (acquired by a publicly traded company in 2020), founder of MLTechniques.com, former VC-funded executive, author and patent owner. Vincent's past corporate experience includes Visa, Wells Fargo, eBay, NBC, Microsoft, and CNET.

Vincent is also a former post-doc at Cambridge University, and the National Institute of Statistical Sciences (NISS). He published in *Journal of Number Theory*, *Journal of the Royal Statistical Society* (Series B), and *IEEE Transactions on Pattern Analysis and Machine Intelligence*. He is also the author of multiple books, available here. He lives in Washington state, and enjoys doing research on stochastic processes, dynamical systems, experimental math and probabilistic number theory.

Contents

List of Figures 10

List of Tables 12

1 Machine Learning Cloud Regression and Optimization 13
- 1.1 Introduction: circle fitting . . . 13
  - 1.1.1 Previous versions of my method . . . 14
- 1.2 Methodology, implementation details and caveats . . . 15
  - 1.2.1 Solution, R-squared and backward compatibility . . . 15
  - 1.2.2 Upgrades to the model . . . 16
- 1.3 Case studies . . . 17
  - 1.3.1 Logistic regression, two ways . . . 17
  - 1.3.2 Ellipsoid and hyperplane fitting . . . 18
    - 1.3.2.1 Curve fitting: 250 examples in one video . . . 18
    - 1.3.2.2 Confidence region for the fitted ellipse: application to meteorite shapes . . . 19
    - 1.3.2.3 Python code . . . 20
  - 1.3.3 Non-periodic sum of periodic time series: ocean tides . . . 26
    - 1.3.3.1 Numerical instability and how to fix it . . . 27
    - 1.3.3.2 Python code . . . 28
  - 1.3.4 Fitting a line in 3D, unsupervised clustering, and other generalizations . . . 29
    - 1.3.4.1 Example: confidence region for the cluster centers . . . 30
    - 1.3.4.2 Exact solution and caveats . . . 31
    - 1.3.4.3 Comparison with K-means clustering . . . 32
    - 1.3.4.4 Python code . . . 34
- 1.4 Connection to synthetic data: meteorites, ocean tides . . . 36

2 A Simple, Robust and Efficient Ensemble Method 37
- 2.1 Introduction . . . 37
- 2.2 Methodology . . . 38
  - 2.2.1 How hidden decision trees (HDT) work . . . 38
  - 2.2.2 NLP case study: summary and findings . . . 39
  - 2.2.3 Parameters . . . 40
  - 2.2.4 Improving the methodology . . . 40
- 2.3 Implementation details . . . 40
  - 2.3.1 Correcting for bias . . . 40
    - 2.3.1.1 Time-adjusted scores . . . 41
  - 2.3.2 Excel spreadsheet . . . 41
  - 2.3.3 Python code and dataset . . . 41
- 2.4 Model-free confidence intervals and perfect nodes . . . 45
  - 2.4.1 Interesting asymptotic properties of confidence intervals . . . 45

3 Gentle Introduction to Linear Algebra – Synthetic Time Series 47
- 3.1 Power of a matrix . . . 47
- 3.2 Examples, generalization, and matrix inversion . . . 48
  - 3.2.1 Example with a non-invertible matrix . . . 49
  - 3.2.2 Fast computations . . . 49
  - 3.2.3 Square root of a matrix . . . 49
- 3.3 Application to machine learning problems . . . 50
  - 3.3.1 Markov chains . . . 50
  - 3.3.2 Time series: auto-regressive processes . . . 50
  - 3.3.3 Linear regression . . . 51

3.4 Mathematics of auto-regressive time series . . . 51
3.4.1 Simulations: curious fractal time series . . . 52
3.4.1.1 White noise: Fréchet, Weibull and exponential cases . . . 52
3.4.1.2 Illustration . . . 52
3.4.2 Solving Vandermonde systems: a numerically stable method . . . 53
3.5 Math for Machine Learning: Must-Read Books . . . 54

4 Image and Video Generation **55**
4.1 Introduction . . . 55
4.2 Applications . . . 56
4.2.1 Spatial time series . . . 56
4.2.2 Prediction intervals in any dimensions . . . 56
4.2.3 Supervised classification of an infinite dataset . . . 57
4.2.3.1 Machine learning perspective . . . 57
4.2.3.2 Six challenging problems . . . 58
4.2.3.3 Mathematical background: the Riemann Hypothesis . . . 58
4.2.3.4 Partial solutions to the six challenging problems . . . 59
4.2.4 Algorithms with chaotic convergence . . . 60
4.3 Python code . . . 60
4.3.1 Path simulation . . . 60
4.3.2 Visual convergence analysis in 2D . . . 63
4.3.3 Supervised classification . . . 64
4.4 Visualizations . . . 67

5 Synthetic Clusters and Alternative to GMM **70**
5.1 Introduction . . . 70
5.2 Generating the synthetic data . . . 71
5.2.1 Simulations with logistic distribution . . . 71
5.2.2 Mapping the raw observations onto an image bitmap . . . 72
5.3 Classification and unsupervised clustering . . . 72
5.3.1 Supervised classification based on convolution filters . . . 73
5.3.2 Clustering based on histogram equalization . . . 73
5.3.3 Fractal classification: deep neural network analogy . . . 74
5.3.4 Generalization to higher dimensions . . . 75
5.3.5 Towards a very fast implementation . . . 75
5.4 Python code . . . 76
5.4.1 Fractal classification . . . 77
5.4.2 GPU classification and clustering . . . 79
5.4.3 Home-made graphic library . . . 81

6 Shape Classification and Synthetization via Explainable AI **84**
6.1 Introduction . . . 84
6.2 Mathematical foundations . . . 84
6.3 Shape signature . . . 85
6.3.1 Weighted centroid . . . 85
6.3.2 Computing the signature . . . 86
6.3.3 Example . . . 87
6.4 Shape comparison . . . 87
6.4.1 Shape classification . . . 88
6.5 Application . . . 88
6.6 Exercises . . . 89

7 Synthetic Data, Interpretable Regression, and Submodels **90**
7.1 Introduction . . . 90
7.2 Synthetic data sets and the spreadsheet . . . 91
7.2.1 Correlation structure . . . 91
7.2.2 Standardized regression . . . 92
7.2.3 Initial conditions . . . 92
7.2.4 Simulations and Excel spreadsheet . . . 92
7.3 Damping schedule and convergence acceleration . . . 93
7.3.1 Spreadsheet implementation . . . 93
7.3.2 Interpretable regression with no overfitting . . . 94
7.3.3 Adaptive damping . . . 94

7.4 Performance assessment on synthetic data . . . 94
7.4.1 Results . . . 95
7.4.2 Distribution-free confidence intervals . . . 97
7.4.2.1 Parametric bootstrap . . . 98
7.5 Feature selection . . . 98
7.5.1 Combinatorial approach . . . 98
7.5.2 Stepwise approach . . . 99
7.6 Conclusion . . . 100

8 From Interpolation to Fuzzy Regression 102
8.1 Introduction . . . 102
8.2 Original version . . . 103
8.3 Full, non-linear model in higher dimensions . . . 103
8.3.1 Geometric proximity, weights, and numerical stability . . . 104
8.3.2 Predicted values and prediction intervals . . . 104
8.3.3 Illustration, with spreadsheet . . . 105
8.3.3.1 Output fields . . . 106
8.4 Results . . . 106
8.4.1 Performance assessment . . . 106
8.4.2 Visualization . . . 107
8.4.3 Amplitude restoration . . . 107
8.5 Exercises . . . 108
8.6 Python source code and datasets . . . 109

9 New Interpolation Methods for Synthetization and Prediction 113
9.1 First method . . . 113
9.1.1 Example with infinite summation . . . 114
9.1.2 Applications: ocean tides, planet alignment . . . 115
9.1.3 Problem in two dimensions . . . 116
9.1.4 Spatial interpolation of the temperature dataset . . . 117
9.2 Second method . . . 119
9.2.1 From unstable polynomial to robust orthogonal regression . . . 120
9.2.2 Using orthogonal functions . . . 120
9.2.3 Application to regression . . . 120
9.3 Python code . . . 121
9.3.1 Time series interpolation . . . 121
9.3.2 Geospatial temperature dataset . . . 124
9.3.3 Regression with Fourier series . . . 127

10 Synthetic Tabular Data: Copulas vs enhanced GANs 129
10.1 Sensitivity analysis, bias reduction and other uses of synthetic data . . . 130
10.2 Using copulas to generate synthetic data . . . 130
10.2.1 The insurance dataset: Python code and results . . . 131
10.2.2 Potential improvements . . . 133
10.3 Synthetization: GAN versus copulas . . . 134
10.3.1 Parameterizing the copula quantiles combined with gradient descent . . . 134
10.3.2 Feature clustering to break a big problem into smaller ones . . . 134
10.4 Deep dive into generative adversarial networks (GAN) . . . 135
10.4.1 Open source libraries and references . . . 135
10.4.2 Synthesizing medical data with GAN . . . 136
10.4.2.1 Hyperparameters . . . 137
10.4.2.2 GAN: Main steps . . . 138
10.4.3 Initial results . . . 139
10.4.4 Fine-tuning the hyperparameters . . . 140
10.4.5 Enhanced GAN: methodology and results . . . 140
10.5 Comparing GANs with the copula method . . . 141
10.5.1 Conclusion: getting the best out of copulas and GAN . . . 143
10.6 Data synthetization explained in one picture . . . 143
10.7 Python code: GAN to synthesize medical data . . . 144
10.7.1 Classification problem . . . 144
10.7.2 GAN method . . . 145
10.7.3 GAN Evaluation and post-classification . . . 147

11 High Quality Random Numbers for Data Synthetization **149**
11.1 Introduction . . . 149
11.2 Pseudo-random numbers . . . 150
11.2.1 Strong pseudo-random numbers . . . 150
11.2.1.1 New test of randomness for PRNGs . . . 151
11.2.1.2 Theoretical background: the law of the iterated logarithm . . . 151
11.2.1.3 Connection to the Generalized Riemann Hypothesis . . . 151
11.2.2 Testing well-known sequences . . . 152
11.2.2.1 Reverse-engineering a pseudo-random sequence . . . 153
11.2.2.2 Illustrations . . . 154
11.3 Python code . . . 156
11.3.1 Fixes to the faulty random function in Python . . . 156
11.3.2 Prime test implementation to detect subtle flaws in PRNG's . . . 156
11.3.3 Special formula to compute 10 million digits of $\sqrt{2}$ . . . 159
11.4 Military-grade PRNG Based on Quadratic Irrationals . . . 162
11.4.1 Fast algorithm rooted in advanced analytic number theory . . . 162
11.4.2 Fast PRNG: explanations . . . 163
11.4.3 Python code . . . 163
11.4.4 Computing a digit without generating the previous ones . . . 165
11.4.5 Security and comparison with other PRNGs . . . 165
11.4.5.1 Important comments . . . 166
11.4.6 Curious application: a new type of lottery . . . 166

12 Some Unusual Random Walks **167**
12.1 Symmetric unbiased constrained random walks . . . 167
12.1.1 Three fundamental properties of pure random walks . . . 167
12.1.2 Random walks with more entropy than pure random signal . . . 168
12.1.2.1 Applications . . . 168
12.1.2.2 Algorithm to generate quasi-random sequences . . . 169
12.1.2.3 Variance of the modified random walk . . . 169
12.1.3 Random walks with less entropy than pure random signal . . . 170
12.2 Related stochastic processes . . . 171
12.2.1 From Brownian motions to clustered Lévy flights . . . 171
12.2.2 Integrated Brownian motions and special auto-regressive processes . . . 172
12.3 Python code . . . 173
12.3.1 Computing probabilities and variances attached to S_n . . . 173
12.3.2 Path simulations . . . 174

13 Divergent Optimization Algorithm and Synthetic Functions **176**
13.1 Introduction . . . 176
13.1.1 The problem, with illustration . . . 177
13.2 Non-converging fixed-point algorithm . . . 178
13.2.1 Trick leading to intuitive solution . . . 178
13.2.2 Root detection: method and parameters . . . 178
13.2.3 Case study: factoring a product of two large primes . . . 179
13.3 Generalization with synthetic random functions . . . 179
13.3.1 Example . . . 181
13.3.2 Connection to the Poisson-binomial distribution . . . 182
13.3.2.1 Location of next root: guesstimate . . . 182
13.3.2.2 Integer sequences with high density of primes . . . 182
13.3.3 Python code: finding the optimum . . . 183
13.4 Smoothing highly chaotic curves . . . 184
13.4.1 Python code: smoothing . . . 184
13.5 Connection to synthetic data: random functions . . . 187

14 Synthetic Terrain Generation and AI-generated Art **188**
14.1 Introduction . . . 188
14.2 Terrain generation and the evolutionary process . . . 190
14.2.1 Morphing and non-linear palette operations . . . 190
14.2.2 The diamond-square algorithm . . . 190
14.2.3 The evolutionary process . . . 191
14.2.4 Finding optimum parameters . . . 191

14.2.5 Mimicking real terrain: the synthesis step . . . 191
14.3 Python code . . . 192
14.3.1 Producing data videos with four sub-videos in parallel . . . 192
14.3.2 Main program . . . 193
14.4 AI-generated art with 3D contours . . . 197
14.4.1 Python code using Matplotlib . . . 198
14.4.2 Python code using Plotly . . . 199
14.4.3 Tips to quickly solve new problems . . . 200

15 Synthetic Star Cluster Generation with Collision Graphs 201
15.1 Introduction . . . 201
15.2 Model parameters and simulation results . . . 202
15.2.1 Explanation of color codes . . . 202
15.2.2 Detailed description of top parameters . . . 202
15.2.3 Interesting parameter sets . . . 203
15.3 Analysis of star collisions and collision graph . . . 204
15.3.1 Weighted directed graphs: visualization with NetworkX . . . 205
15.3.2 Interesting findings: how the universe got started . . . 205
15.4 Animated data visualizations . . . 206
15.5 Python code and computational issues . . . 207
15.5.1 Simulating the real and synthetic universes . . . 207
15.5.2 Visualizing collision graphs . . . 211

16 Perturbed Lattice Point Process: Alternative to GMM 213
16.1 Perturbed lattices: definition and properties . . . 213
16.1.1 Point counts distribution . . . 214
16.1.2 Periodicity and amplitude of point count expectations . . . 214
16.1.3 Testing the independence of point counts . . . 215
16.1.3.1 Results and Interpretation . . . 216
16.1.3.2 About the Spreadsheet . . . 217
16.2 Cluster processes and nearest neighbor graphs . . . 217
16.2.1 Synthetic, semi-rigid cluster structures . . . 217
16.2.2 Python code to generate cluster processes . . . 219
16.2.3 References on cluster processes . . . 219
16.2.4 Superimposed perturbed lattices: an alternative to mixture models . . . 220
16.2.4.1 Hexagonal lattice, nearest neighbors . . . 221
16.2.4.2 Exercises: nearest neighbor graphs, size of connected components . . . 222
16.2.4.3 Python code to compute connected components . . . 223
16.3 Statistical inference for point processes . . . 225
16.3.1 Estimation of Core Parameters . . . 225
16.3.1.1 Intensity . . . 226
16.3.1.2 Scaling factor . . . 226
16.3.1.3 Alternative estimation method . . . 226
16.3.2 Spatial statistics, nearest neighbors, clustering . . . 227
16.3.2.1 Inference for two-dimensional processes . . . 227
16.3.2.2 Other possible tests . . . 227
16.3.2.3 Rayleigh test . . . 228
16.3.2.4 Exercises . . . 229
16.4 Special topics . . . 230
16.4.1 Minimum contrast estimation and explainable AI . . . 230
16.4.2 Model identifiability, hard-to-detect patterns . . . 231
16.4.2.1 Stochastic residues . . . 231
16.4.3 Hidden model and random permutations . . . 231
16.4.4 Retrieving the F distribution . . . 233
16.4.4.1 Theoretical values obtained by simulations . . . 233
16.4.4.2 Retrieving F from the interarrival times distribution . . . 234
16.4.5 Record distances between an observed point and its vertex . . . 234
16.4.5.1 Distribution of records . . . 235
16.4.5.2 Distribution of arrival times for records . . . 236

17 Synthetizing Multiplicative Functions in Number Theory 237
17.1 Introduction . . . 237

17.1.1 Key concepts and terminology . . . 238
17.1.2 Orbits and holes . . . 238
17.1.3 Industrial Applications . . . 238
17.2 Euler products . . . 239
17.2.1 Finite Euler Products . . . 239
17.2.1.1 Generalization using Dirichlet characters . . . 240
17.2.2 Infinite Euler products . . . 241
17.2.2.1 Special products . . . 241
17.2.2.2 Probabilistic properties and conjectures . . . 242
17.3 Finite Dirichlet series and generalizations . . . 243
17.3.1 Finite Dirichlet series . . . 243
17.3.2 Non-trivial cases with infinitely many primes and a hole . . . 245
17.3.2.1 Sums of two cubes, or cuban primes . . . 245
17.3.2.2 Primes associated to elliptic curves . . . 245
17.3.2.3 Analytic continuation, convergence, and functional equation . . . 246
17.3.2.4 Hybrid Dirichlet-Taylor series . . . 246
17.3.3 Riemann Hypothesis with cosines replaced by wavelets . . . 247
17.3.4 Riemann Hypothesis for Beurling primes . . . 248
17.3.5 Stochastic Euler products . . . 249
17.4 Exercises . . . 250
17.5 Python code . . . 253
17.5.1 Computing the orbit of various Dirichlet series . . . 253
17.5.2 Creating videos of the orbit . . . 256

18 Text, Sound Generation and Other Topics 259
18.1 Sound generation: let your data sing! . . . 259
18.1.1 From data visualizations to videos to data music . . . 259
18.1.2 References . . . 260
18.1.3 Python code . . . 260
18.2 Data videos and enhanced visualizations in R . . . 261
18.2.1 Cairo library to produce better charts . . . 261
18.2.2 AV library to produce videos . . . 262
18.3 Dual confidence regions . . . 263
18.3.1 Case study . . . 263
18.3.2 Standard confidence region . . . 263
18.3.3 Dual confidence region . . . 264
18.3.4 Simulations . . . 264
18.3.5 Original problem with minimum contrast estimators . . . 265
18.3.6 General shape of confidence regions . . . 266
18.4 Fast feature selection based on predictive power . . . 267
18.4.1 How cross-validation works . . . 268
18.4.2 Measuring the predictive power of a feature . . . 268
18.4.3 Efficient implementation . . . 269
18.5 NLP: taxonomy creation and text generation . . . 270
18.5.1 Designing a keyword taxonomy . . . 270
18.5.2 Fast clustering algorithm for keyword data . . . 271
18.5.2.1 Computational complexity . . . 271
18.5.2.2 Smart crawling of the whole Internet and a bit of graph theory . . . 272
18.6 Automated detection of outliers and number of clusters . . . 273
18.6.1 Black-box elbow rule to detect outliers . . . 273
18.7 Advice to beginners . . . 274
18.7.1 Getting started and learning how to learn . . . 274
18.7.1.1 Getting help . . . 275
18.7.1.2 Beyond Python . . . 275
18.7.2 Automated data cleaning and exploratory analysis . . . 275
18.7.3 Example of simple analysis: marketing attribution . . . 276

Glossary 277

Bibliography 280

Index 285

List of Figures

1.1 Fitted ellipse (blue), given the training set (red) distributed around a partial arc 19
1.2 Confidence region in blue, $n = 30$ training set points; 50 training sets (left) vs 150 (right) 20
1.3 Three non-periodic time series made of periodic terms (see section 17.2.2.1) 26
1.4 Training set (red), validation set (orange), fitted curve (blue) and model (gray) 27
1.6 Biased confidence region for (θ_A^*, θ_B^*); same example as in Figure 1.5; true value is $(0.5, 1.0)$ . . . 30
1.5 Finding the two centers θ_A^*, θ_B^* in sample 39; $n = 1000$ 31
1.7 Challenging mixture, requiring $p_A = 3, p_B = 1$ to identify the two cluster centers 32

2.1 Output from the Excel version of HDT 42

3.1 AR models, classified based on the types of roots of the characteristic polynomial 53

4.1 Scatterplot observations vs. predicted values, with prediction intervals (in any dimension) 67
4.2 Comets orbiting the sun: simulation 67
4.3 Comets orbiting the sun: snapshot in time 68
4.4 Three orbits of $\eta(\sigma + it)$: $\sigma = 0.5$ (red), 0.75 (blue) and 1.25 (yellow) 68
4.5 Sample orbit points of $\eta(\sigma + it)$: $\sigma = 0.5$ (red), 0.75 (blue) and 1.25 (yellow) 68
4.6 Sample orbit points of $\eta(\sigma + it)$: $\sigma = 0.5$ (red), 0.75 (blue) and 1.25 (yellow) 69
4.7 Raw orbit points of $\eta(\sigma + it)$: $\sigma = 0.5$ (red), 0.75 (blue) and 1.25 (yellow) 69
4.8 Convergence of partial sums of $\eta(z)$, for six $z = \sigma + it$ in the complex plane 69

5.1 Special interlacing of 4 lattice processes with $s = 0$. 72
5.2 Classification of left dataset; $s = 0.15$, $w = 10$. One loop (middle) vs 3 (right). 73
5.3 Clustering of left dataset; $s = 0.15$, 3 loops, $w = 10$ (middle) vs 20 (right). 74
5.4 Classification ($w = 10$) and clustering ($w = 20$); $s = 0.05$, three loops. 74
5.5 Fractal classification, $s = 0.15$. Loop 6, 250 and 400. 75
5.6 Fractal classification, $s = 0.05$.Loop: 6 and 60. 75
5.7 Fast (left) vs standard method (right), 3 loops, $s = 0.15, w = 10$. 76
5.8 Fast method, $s = 0.05, w = 20$. Three loops (middle), one loop (right). 76

6.1 Comparing two shapes 85
6.2 Weighted centroid, shape signature 86
6.3 Weight function used in Figure 6.2 87
6.4 Another interesting shape 88

7.1 Regression coefficients oscillating when using adaptive damping 95
7.2 Convergence of regression coefficients (left) and distribution of residual error (right) 96
7.3 Goodness-of-fit: training set (right) versus validation set (left) 96

8.1 Fuzzy regression with prediction intervals, original version, 1D 103
8.2 Fuzzy regression with prediction intervals, full model, 2D 105
8.3 Scatterplots: median vs weighted method, on validation (left) vs training set (right) 107
8.4 Dirichlet eta function (real part, bottom) and interpolation error (top) 109

9.1 Interpolating the real part of $\zeta(\frac{1}{2} + it)$ based on orange points 114
9.2 Tides at Dublin (5-min data), with 80 mins between interpolating nodes 117
9.3 Temperature data: interpolation with my method (observed values at dots) 118
9.4 My method: round dots represent observed values, "+" are interpolated 118
9.5 Temperature dataset: interpolation using ordinary kriging 119

10.1 Synthetic versus real data, produced by SDV GAN + copula 135

10.2 Loss function (in orange) for 10^4 successive epochs; enhanced GAN on the right plot 137
10.3 Summary statistics, medical dataset (synth 1 and 2 correspond to GAN) 138
10.4 Correlation matrix, real vs synthetic: GAN (synth 2) and copula-based 139
10.5 Copulas superior to GANs (synth 1, 2) to capture correlations in real data 139
10.6 Real data (left), copula (middle) and GAN synthetization (right) 141
10.7 Loss function (orange) and distance (grey), circle dataset 142
10.8 Data synthetization: general schema 144

11.1 Orbit of $L(z, \chi)$ at $\sigma = \frac{1}{2}$, with $0 < t < 200$ and $\chi = \chi_4$ (left) versus pseudo-random χ (right) . . 152
11.2 $L_3^*(n)$ test statistic for four sequences: Python[200] and SQRT[90,91] fail 154
11.3 $|L_3(n)|$ test statistic for four sequences: Python[200] and SQRT[90,91] fail 154
11.4 Correlations are computed on sequences consisting of 300 binary digits 165

12.1 Typical path S_n with $0 \leq n \leq 50{,}000$ for four types of random walks 168
12.2 $\delta_n = 1 - \text{Var}[S_{n+1}] + \text{Var}[S_n]$ for four types of random walks, with $0 \leq n \leq 5000$ 169
12.3 Same as Figure 12.2, using a more aesthetic but less meaningful chart type 170
12.4 Clustered Brownian process 172
12.5 AR models, classified based on the types of roots of the characteristic polynomial 173

13.1 Function $f(b)$ as a better alternative to $g(b)$ in Figure 13.2. Root at $b = 3083$. 177
13.2 Function $g(b) = 2 - \cos(2\pi b) - \cos(2\pi a/b)$, with $a = 3083 \times 7919$. 177
13.3 Transformed function f_3, amplifying the root at $b = 3083$. 178
13.4 Signal strength ρ_n, first 130 fixed-point iterations; $n = 31$ leads to a root. 181
13.5 (b_n, ρ_n) plot. Yellow and orange dots linked to roots. 181
13.6 Signal strength ρ_n, first 130 fixed-point iterations; $n = 87$ leads to a root. 181
13.7 Random function from section 13.3.1, with root at $b = 5646$. 184

14.1 Six frames from the terrain video, each one containing four images 189
14.2 Contour plot, 3D mixture model, produced with Plotly 197
14.3 Same as Figure 14.2, produced with Matplotlib 198

15.1 Collisions graph for the biggest star eater (star 47) in video 7 205
15.2 Summary statistics for the whole collision structure: the X axis represents the time 206
15.3 Snapshots of universe 4 (left) and universe 7 (right) 207

16.1 Period and amplitude of $\phi_\tau(t)$; here $\tau = 1, \lambda = 1.4, s = 0.3$ 215
16.2 A new test of independence (R-squared version) 215
16.3 Radial cluster process ($s = 0.2, \lambda = 1$) with centers in blue; zoom in on the left 218
16.4 Radial cluster process ($s = 2, \lambda = 1$) with centers in blue; zoom in on the left 218
16.5 Manufactured marble lacking true lattice randomness (left) 218
16.6 Four superimposed Poisson-binomial processes: $s = 0$ (left), $s = 5$ (right) 221
16.7 Rayleigh test to assess if a point distribution matches that of a Poisson process 229
16.8 Realization of a 5-interlacing with $s = 0.15$ and $\lambda = 1$: original (left), modulo $2/\lambda$ (right) 232
16.9 Locally random permutation σ; $\tau(k)$ is the index of X_k's closest neighbor to the right 232
16.10Each arrow links a point (blue) to its vertex (red): $s = 0.2$ (left), $s = 1$ (right) 235
16.11Distance between a point and its vertex ($\lambda = s = 1$) 236

17.1 Three orbits ($\sigma = 0.5, 0.75, 1.25$) with finite Euler product: $P = \{2, 3\}$ (left) vs $\{2, 3, 5\}$ (right) . 240
17.2 Distance between orbit and location $(c, 0)$ depending on t on the X-axis 242
17.3 Distance between orbit and location $(c, 0)$ depending on t on the X-axis 242
17.4 Distance between orbit and location $(c, 0)$ depending on t on the X-axis 242
17.5 Four orbits where the "hole" (repulsion basin) is apparent 244
17.6 Three orbits with "hole" closer to the origin, showing impact of $\beta > \frac{1}{2}$ and larger n 244
17.7 Orbit of Dirichlet eta $\eta(z)$ when cosines are replaced by other periodic functions 248

18.1 Data linked to the melody: red curve for note frequencies, blue curve for note durations 260
18.2 R plot before Cairo (left), and after (right) 261
18.3 Intermediate (left) and last frame (right) of the video 262
18.4 Example of 90% dual confidence region for (p, q) 264
18.5 Minimum contrast estimation for (λ, s) using (p, q) as proxy stats 265
18.6 Non-elliptic confidence regions with various confidence levels 266
18.7 Elbow rule (right) finds $m = 3$ clusters in Brownian motion (left) 274

List of Tables

1.1 Estimated ellipse parameters vs true values ($n = 30$), for shape in Figure 1.2 . . . 20
1.2 First and last step of `curve_fitting`, approaching the model. . . . 28
1.3 MSE for different methods and θs, same data set as in Figure 1.5 . . . 33
1.4 MSE for different methods and θs, same data set as in Figure 1.7 . . . 33

2.1 List of potential features to use in the model . . . 38
2.2 Statistics for selected HDT nodes (Excel version) . . . 41
2.3 Order of magnitude for the expectation and standard deviation of the range R_n . . . 45

3.1 Characteristic polynomials used in the simulations . . . 52

7.1 Regression coefficients and performance metrics r, s based on methodology . . . 97
7.2 Correlation matrix . . . 97
7.3 Best performance given m (number of features) . . . 98
7.4 Feature comparison table (top 32 feature combinations) . . . 100
7.5 Feature comparison table (bottom 31 feature combinations) . . . 101

8.1 R-squared ρ^2 and slope β, on training and validation sets, median vs weighted . . . 107

10.1 Comparing real data with two different synthetic copies . . . 132
10.2 Correlations on 9D circle dataset: real vs copula and GAN . . . 142

11.1 $L_3^*(n)$, for various sequences ($n = 20{,}000$); "Fail" means failing the prime test . . . 155

13.1 High ρ_n at iterations $n = 31$ and $n = 127$ points to roots 3083 and 7919 . . . 180

15.1 Description of top parameters used in the star cluster simulator . . . 203
15.2 Eight selected parameter sets covering various situations . . . 204

16.1 Variance attached to F_s, as a function of s . . . 214
16.2 Poisson process ($s = \infty$) versus $s = 39.85$ . . . 234

18.1 Extract of the mapping table used to recover (λ, s) from (p, q) . . . 266
18.2 Eight bins: 2 features (A, B) times 2 outcomes (Good/Bad) . . . 268
18.3 Amount of data collected at each level, when crawling the Internet . . . 272

implementation can be found here. Start by computing the correlation matrix attached to your real data. Rank all pairs of features $\{A, B\}$ by correlation between A and B, starting with the largest correlation in absolute value, down to the lowest one that is statistically significant and/or at least above (say) 0.30. The top pair constitutes your first group of features. Look at the second pair. If it contains a feature from the first group, merge the two groups to obtain a single group with 3 features. If not, you have two groups of features at this stage, each containing 2 features. Proceed iteratively until all pairs of features have been visited.

However, the correlation structure may not always represent all the dependencies in the data: points distributed on a circle are not correlated, yet they are highly dependent! In this case, use 1 dimension (the angular position) rather than the 2 Cartesian coordinates. More generally, appropriate data transformations and reduction can fix the issue.

10.4 Deep dive into generative adversarial networks (GAN)

In this section I discuss GAN in details, with a Python implementation to synthesize tabular data. Unlike many neural networks, my code can generate replicable outputs. Other solutions and references are provided in section 10.4.1. Later on, I discuss enhancements to the original model. Finally, I show how to blend GAN with copulas to get the best of both worlds.

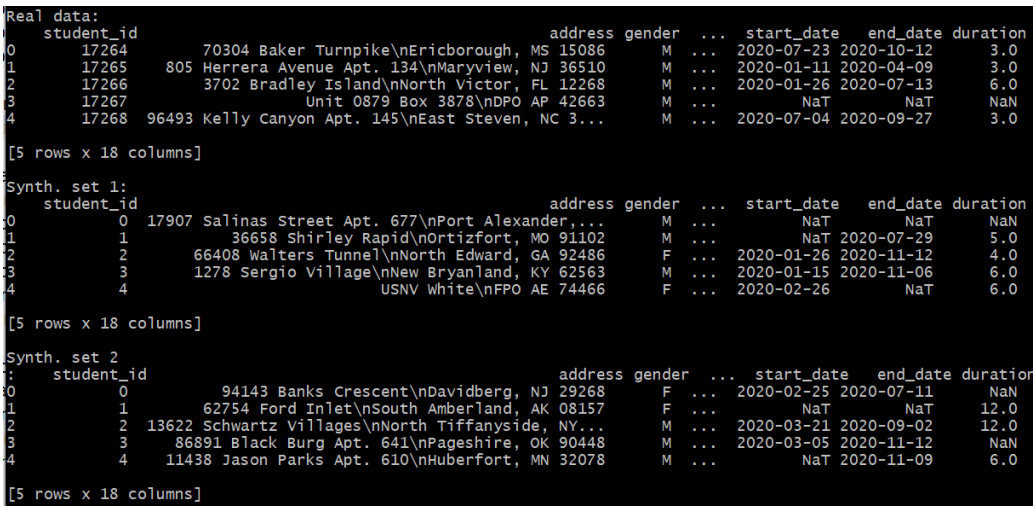


Figure 10.1: Synthetic versus real data, produced by SDV GAN + copula

10.4.1 Open source libraries and references

One of the most popular libraries for synthetization is SDV, which stands for synthetic data vault. You can check it out on GitHub, here. For sample code, see here. and here. SDV comes with 28 real-life datasets. To see the list, with the number of tables, rows and columns for each data set, run the code below.

```
from sdv.demo import get_available_demos
from sdv.demo import load_tabular_demo
from sdv.tabular import CopulaGAN

demos = get_available_demos()
print(demos) # show list of demo datasets

metadata, real_data = load_tabular_demo('student_placements_pii',metadata=True)
print("\nReal data:\n",real_data.head())
model = CopulaGAN()
model = CopulaGAN(primary_key='student_id',anonymize_fields={'address': 'address' })
model.fit(real_data)
synth_data1 = model.sample(200)
print("\nSynth. set 1:\n",synth_data1.head())

model.save('my_model.pkl')      # this shows how to save the model
loaded = CopulaGAN.load('my_model.pkl') # load the model, and
synth_data2 = loaded.sample(200) # get new set of synth. data
print("\nSynth. set 2\n:",synth_data2.head())
```

The first example is a YouTube dataset with 2 tables, 2735 rows and 10 columns. The number of rows ranges from 83 to over 6 million, and some datasets have over 300 features. The code in question also loads one dataset (`'student_placements_pii'`), and shows how to use a GAN model based on Gaussian copulas. See output in Figure 10.1. The code is on GitHub, here. Some of the fields such as the address are anonymized rather than synthesized. This is performed by implicitly calling the Python library Faker, described here. Also, you can choose the option `FAST_ML` for the optimizer (the gradient descent algorithm), for faster processing but with lower accuracy. This is done using the instruction `TabularPreset(name='FAST_ML', metadata=metadata)` as explained here.

SDV is a black-box, so to illustrate the various steps of GAN in details, I use an implementation based on the Keras library instead. Keras is easier to use than Tensorflow [Wiki], though it requires Tensorflow to be installed on your system. Another black-box alternative, allowing you to implement GAN for tabular data synthetization with just 3 lines of code, is TabGAN, available here. See [6] for a discussion. I had to install the most recent version of Numpy to get it to work. More generally, installing these libraries may also require the most recent version of pip, which you can get via the command `pip install --upgrade pip`. TabGAN uses LightGBM [Wiki], a fast version of gradient boosting, and it is thus a bit faster than my step-by-step version in section 10.7.

Another implementation very similar to mine and illustrated on time series, is discussed here. Mine includes a new version of correlation matrix distance to assess quality; see here and [64] for a standard definition. It also uses a seed for every single source of randomness in the algorithm, allowing for full replicability, as well as other specific features. If you run the code in GPU, there might be additional sources of randomness that you can't control. You can run the code (say `mycode.py`) with the following command line in that case, if replicability is important for your application:

```
> CUDA_VISIBLE_DEVICES="" PYTHONHASHSEED=0 python mycode.py
```

Finally, there are various libraries to assess the quality of the synthetized data. I use TableEvaluator in my code in section 10.7, along with home-made metrics that are more useful to me. TableEvaluator is described here. There is not much documentation about it, but you can check out the full source code on GitHub, here. That is how I found out that the output metric called "Base Statistics" is a correlation distance between the real versus synthesized data, computed on various bivariate indicators as data points (mean, median, correlation between features and so on both for real and synthesized).

Additional reading on the subject includes the book "Synthetic Data for Deep Learning" [97], "Pros and Cons of GAN Evaluation Measures" [15], "Survey on Synthetic Data Generation, Evaluation Methods and GANs" [40], and "Are GANs Created Equal? A Large-Scale Study" by Google Brain [80]. See also Lei Xu's master thesis (MIT, 2017) available here and the related article on ArXiv [120], as well as this article. For the Keras models used in my implementation, see here.

10.4.2 Synthesizing medical data with GAN

Here I summarize my implementation of GAN applied to the Kaggle diabetes data set. First, I discuss the hyperparameters, then the main steps in the methodology. The data is processed "as is", without normalization or transformation. Possible transformations (preprocessing) are discussed in section 10.5.1. However, I removed all the observations with missing values, for better comparison with the copula method. Anyway, it makes sense to treat these observations separately, as a different segment, by re-running GAN on them only. Indeed, it produces better results when they are separated from the complete observations. After removing observations with missing values, we are left with 392 rows.

The dataset has 9 features. One of them called `Outcome` is the response: it indicates whether or not the patient had cancer. Thus the problem is predicting – based on the remaining 8 features – the chance of getting cancer. It is a supervised classification problem with two groups: cancer versus non-cancer.

The first part of the code (section 10.7.1) imports the libraries, reads the data and removes observations with missing values, and then performs the classification with the random forest algorithm [Wiki]. It also defines some global variables such as the learning rate [Wiki] in the Adam gradient descent, and `seed` which allows for replicability by using the same value in each run. The quality of the classification is displayed on the screen, as the `Base Accuracy` metric.

The last part of the code (section 10.7.3) evaluates the quality of the synthetic data. It also performs the classification on the synthetic data, for comparison with the results obtained on the real data. It would be interesting to augment the real data by adding the synthetic data into it, and see if we get more robust predictions. The augmented data is not expected to increase accuracy; instead it is expected to increase robustness and reduce overfitting.

The dataset `diabetes.csv` is on my GitHub directory, here. The original can be found on Kaggle, here. It should be noted that all the features are treated as continuous, even the binary `Outcome`. Thus

the number of pregnancies (an integer) or the "outcome" generated by GAN are real numbers, that must be mapped onto integers to make sense. Other implementations such as copulas or copula-based GANs do not have this limitation. An alternative is to use one GAN for `Outcome==0`, and another one for `Outcome==1`. Also, categorical features (absent here except for Outcome) can be replaced by dummy variables [Wiki].

10.4.2.1 Hyperparameters

There is a surprisingly large number of hyperparameters that can be fine-tuned. This is one of the reasons why these systems take a lot of time to train and optimize. In addition, the gradient descent present in all GANs (with its own parameters), is the bottleneck. Self-identification of good parameters by the algorithm itself – and possibly adjusted over time – is one way to at least automate the process. But it can lead to overfitting. The best solution is to use standard hyperparameters that work well in many contexts, without trying to over-adjust.

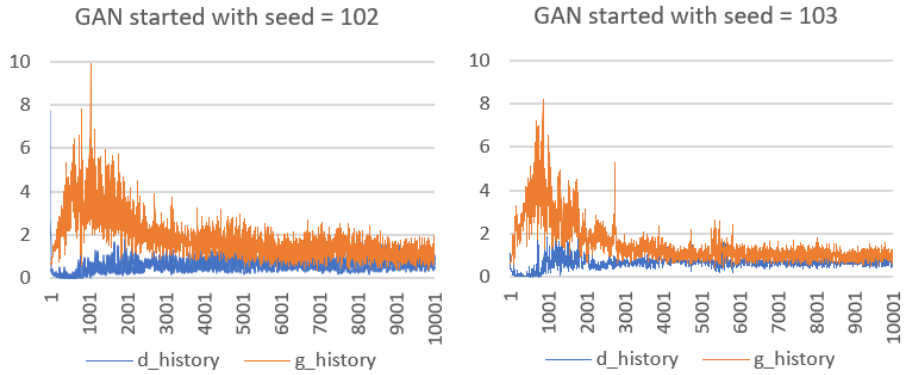


Figure 10.2: Loss function (in orange) for 10^4 successive epochs; enhanced GAN on the right plot

All these parameters are set and used in the GAN part of the code, in section 10.7.2. The only exception is the learning rate parameter.

- Epochs: One iteration of GAN consists of processing the full data set: this is done using a number of small samples (batches), one at a time. The number of iterations is the number of epochs, set to `n_epochs=10000` (a typical value) in the `train` function used to train the GAN.
- Batch size: see Epochs. Here it is set to `n_batch=128` in the `train` function. Half of the batch is used to sample from the real data, and the other half to generate latent data.
- Latent data: here latent variables [Wiki] are univariate random Gaussian deviates with zero mean and unit variance; typically their number matches the number of features. Their role is similar to the Gaussian deviates in the copula method. However, using a uniform distribution on $[-1, 1]$ is worth exploring as many GAN functions (for instance ReLU) return values between 0 and 1. Such restrictions are also used in Fourier regression in section 9.3.3.
- Activation function: in neural network, the activation function [Wiki] decides whether a neuron should be activated or not. Classic examples used in the code are ReLU [Wiki] and sigmoid [Wiki].
- Kernel initializer: defines the way to set the initial random weights of Keras layers. See documentation here.
- Number of layers: the neural networks (both the generator and discriminator) use 3 layers. A layer is added via the instruction `model.add`, with a number of options: activation function, kernel initializer, and so on. See `define_generator`, `define_discriminator` and `define_gan` (the combination of both) in the code. With 3 layers, we are dealing with a deep neural network.
- Learning rate: attached to the gradient descent. I tried 0.01 and 0.001, and settled for the latter. Small values result in quite chaotic, faster behavior (at the beginning) and somewhat reduced accuracy. Large values result in slow steady progress but you can end up stuck in a bad local optimum. Think of it as the cooling schedule in a simulated annealing algorithm.
- Gradient descent method: I use `Adam`. The alternative `SGD` (stochastic gradient descent [Wiki]) did no do well here, but does well in computer vision.
- Loss function: in gradient descent or any optimization problem, the loss function [Wiki] specifies the type of distance to the optimum vector, used for minimization. Think of it as a regression problem solved by least squares – the loss function being quadratic in this case. It is sometimes called the error function.

- Accuracy: calculates how often predictions equal labels in the context of classification. In this context, classification means assigning an observation to either real (truly real or excellent synthetization) or fake (poor synthetic data not close to the reality). See Keras documentation here. There are various options to choose from, to measure accuracy.
- Architecture: the type of neural network. In this example, set to `Sequential`.

10.4.2.2 GAN: Main steps

The steps in this section corresponds to the actual GAN procedure in section 10.7.2. It does not include the pre-processing step: checking how good the real training set is at predicting cancer in a cross-validation framework (section 10.7.1). It does not cover the post-processing step either: GAN evaluation and how good the synthesized training set is at predicting cancer in the same cross-validation framework. This part of the code is covered in section 10.7.3. It is assumed that all the GAN models have been created and compiled, with the hyperparameters discussed in section 10.4.2.1. So this section only covers the `train` function used to train GAN. That said, it is the most important part of the code.

		Pregnanci	Glucose	BloodPres	SkinThickr	Insulin	BMI	DiabetesP	Age	Outcome
AVG	synth 1	1.82	119.34	68.49	27.25	141.86	33.52	1.53	30.40	0.46
	synth 2	3.00	124.20	73.55	28.89	137.10	35.70	1.54	34.26	0.42
	copula	3.07	121.30	70.64	29.30	153.85	33.29	0.50	30.13	0.33
	real	3.30	122.63	70.66	29.15	156.06	33.09	0.52	30.86	0.33
MIN	synth 1	-5.01	26.22	13.35	4.04	14.41	7.72	0.20	2.29	0.00
	synth 2	-3.08	16.41	14.94	7.16	1.69	9.15	-0.71	8.03	0.00
	copula	0.00	62.37	28.42	7.48	17.49	18.76	0.09	21.00	0.00
	real	0.00	56.00	24.00	7.00	14.00	18.20	0.09	21.00	0.00
MAX	synth 1	13.31	355.83	194.77	86.79	662.69	95.30	4.22	75.99	1.00
	synth 2	15.67	304.61	154.16	91.65	745.49	89.84	3.63	87.25	1.00
	copula	14.83	197.93	110.00	60.03	602.11	58.01	2.30	72.57	1.00
	real	17.00	198.00	110.00	63.00	846.00	67.10	2.42	81.00	1.00
Stdev	synth 1	2.85	56.02	35.53	16.54	115.73	17.46	0.63	13.74	0.50
	synth 2	3.37	46.89	25.06	13.88	96.11	13.36	0.73	14.99	0.49
	copula	2.85	31.57	11.52	10.41	112.60	7.13	0.32	9.22	0.47
	real	3.21	30.86	12.50	10.52	118.84	7.03	0.35	10.20	0.47
p.25	synth 1	0.04	79.11	42.47	14.71	58.77	20.15	1.09	20.40	0.00
	synth 2	0.62	88.09	55.71	18.13	66.24	25.77	1.02	23.42	0.00
	copula	1.00	97.75	64.00	21.00	76.00	28.70	0.27	23.00	0.00
	real	1.00	99.00	62.00	21.00	76.75	28.40	0.27	23.00	0.00
p.75	synth 1	3.64	149.07	88.13	36.65	187.51	43.08	1.88	38.94	1.00
	synth 2	4.94	156.80	88.59	38.04	185.61	44.81	2.01	42.15	1.00
	copula	4.02	142.25	78.00	36.37	192.27	37.22	0.67	35.03	1.00
	real	5.00	143.00	78.00	37.00	190.00	37.10	0.69	36.00	1.00

Figure 10.3: Summary statistics, medical dataset (synth 1 and 2 correspond to GAN)

The following steps are repeated for each epoch in the `train` function:

- Step 1: update the discriminator: get a new sample (half batch) from the real data and assign these points to `label=1`; get a new latent data sample (half batch, random Gaussian vectors) to generate fake data and assign these points to `label=0`. The fake sampling function also maps the latent data into the space of the real data via the instruction `X=generator.predict(x_input)`. Here `x_input` represents the latent data, and `X` the mapped version. The `train_on_batch` function (one call for the real sample, one call for the fake one) also returns the losses for each data label (fake / real). Note that the discriminator is set to "non-trainable".
- Step 2: update the generator: get a full sample (full batch) of latent data, assigned this time to `label=1` to train the generator. Training aims at minimizing over time (on average) the loss function `g_loss` (an average of the `d_loss` values returned by the discriminator) until an equilibrium is reached: a local minimum in all likelihood. Note that `g_loss` oscillates over time in order to not get stuck too quickly in a local minimum. The steepest gradient descent can result in this problem. So we use the Adam gradient descent (adaptive moment estimation) instead to avoid this issue.
- Step 3: If the epoch iteration is a multiple of 200, output the summary statistics to show progress, in particular how `g_loss` is decreasing over time, on average.

Note that the output layer of the discriminator is activated by the sigmoid function to discriminate between real and fake samples.

Correl	Pregnanci	1.00	0.38	0.20	0.23	0.17	0.32	0.14	0.82	0.17
synth 2	Glucose	0.38	1.00	0.64	0.63	0.72	0.75	0.82	0.67	0.67
	BloodPres	0.20	0.64	1.00	0.48	0.35	0.75	0.52	0.70	0.51
	SkinThickr	0.23	0.63	0.48	1.00	0.45	0.92	0.18	0.44	0.44
	Insulin	0.17	0.72	0.35	0.45	1.00	0.55	0.63	0.43	0.66
	BMI	0.32	0.75	0.75	0.92	0.55	1.00	0.37	0.68	0.55
	DiabetesP	0.14	0.82	0.52	0.18	0.63	0.37	1.00	0.43	0.56
	Age	0.82	0.67	0.70	0.44	0.43	0.68	0.43	1.00	0.46
	Outcome	0.17	0.67	0.51	0.44	0.66	0.55	0.56	0.46	1.00

		Pregnanci	Glucose	BloodPres	SkinThickr	Insulin	BMI	DiabetesP	Age	Outcome
Correl	Pregnanci	1.00	0.13	0.22	0.05	0.00	-0.06	-0.02	0.56	0.25
copula	Glucose	0.13	1.00	0.27	0.25	0.62	0.25	0.20	0.36	0.57
	BloodPres	0.22	0.27	1.00	0.23	0.15	0.29	-0.02	0.26	0.21
	SkinThickr	0.05	0.25	0.23	1.00	0.22	0.67	0.13	0.15	0.26
	Insulin	0.00	0.62	0.15	0.22	1.00	0.25	0.21	0.16	0.33
	BMI	-0.06	0.25	0.29	0.67	0.25	1.00	0.13	0.12	0.25
	DiabetesP	-0.02	0.20	-0.02	0.13	0.21	0.13	1.00	0.05	0.22
	Age	0.56	0.36	0.26	0.15	0.16	0.12	0.05	1.00	0.36
	Outcome	0.25	0.57	0.21	0.26	0.33	0.25	0.22	0.36	1.00

		Pregnanci	Glucose	BloodPres	SkinThickr	Insulin	BMI	DiabetesP	Age	Outcome
Correl	Pregnanci	1.00	0.20	0.21	0.09	0.08	-0.03	0.01	0.68	0.26
real	Glucose	0.20	1.00	0.21	0.20	0.58	0.21	0.14	0.34	0.52
	BloodPres	0.21	0.21	1.00	0.23	0.10	0.30	-0.02	0.30	0.19
	SkinThickr	0.09	0.20	0.23	1.00	0.18	0.66	0.16	0.17	0.26
	Insulin	0.08	0.58	0.10	0.18	1.00	0.23	0.14	0.22	0.30
	BMI	-0.03	0.21	0.30	0.66	0.23	1.00	0.16	0.07	0.27
	DiabetesP	0.01	0.14	-0.02	0.16	0.14	0.16	1.00	0.09	0.21
	Age	0.68	0.34	0.30	0.17	0.22	0.07	0.09	1.00	0.35
	Outcome	0.26	0.52	0.19	0.26	0.30	0.27	0.21	0.35	1.00

Figure 10.4: Correlation matrix, real vs synthetic: GAN (synth 2) and copula-based

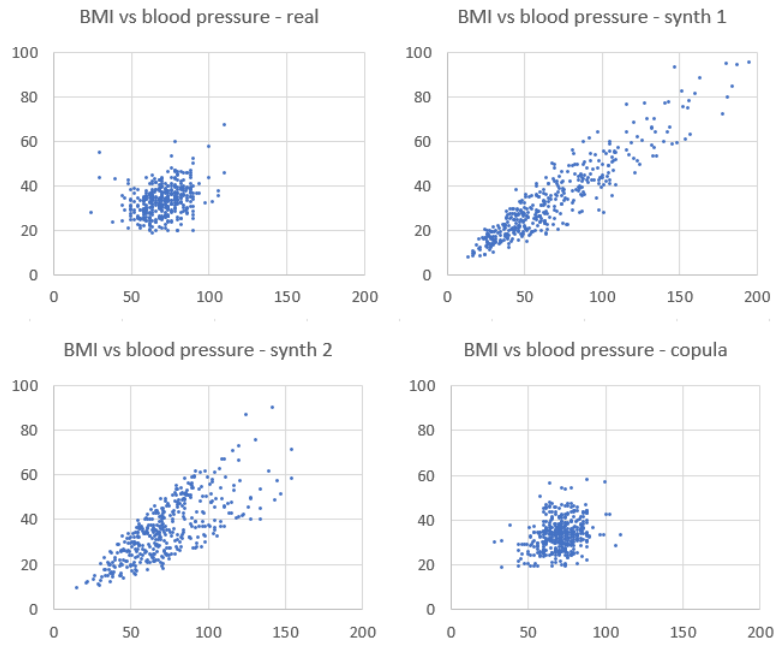


Figure 10.5: Copulas superior to GANs (synth 1, 2) to capture correlations in real data

10.4.3 Initial results

The first synthetized data using GAN on the medical dataset was disappointing. While GAN was able to replicate the mean, variance, and percentiles attached to each of the 9 features, it failed at replicating the

correlation structure. In addition, GAN was very sensitive to the seed (denoted as `seed` in the code). Also, even for a single gradient path started with a specific seed, the oscillations in the loss function over successive epochs, and thus the quality of the synthetized data, were still volatile even after 5000 epochs. I present the initial results here. It originates from a piece of code widely distributed over the Internet. I show in section 10.4.5 how to significantly improve the algorithm using front-end modifications, with little changes to the hyperparameters.

The left plot in Figure 10.2 shows the volatility in the loss function – the orange curve. In the same figure, the contrast between the left and right plot shows the huge impact of the seed on the final results. In Figures 10.3, 10.4 and 10.5, "synth 1" represents the initial version of GAN, while "synth 2" corresponds to the enhanced version. Even the enhanced version is inferior to copulas to reconstruct the correlation structure. However, GAN, even with the initial version, does a decent job at reconstructing the statistical summaries of most features (mean, variance, percentiles $p_{.25}$ and $p_{.75}$). More results are in the spreadsheet `diabetes_synthetic.xlsx`, available here on GitHub.

10.4.4 Fine-tuning the hyperparameters

You can use a back-end or front-end approach to fine-tune the hyperparameters and other GAN components, or a combination of both. The back-end strategy consists of modifying components that are buried more deeply in the architecture, such as the loss function, the number and type of layers, the type of neural network (sequential here), the gradient descent method (Adam here), the size of the batches and the ratio when splitting batches into real versus fake, the dimension and type of random deviates for the latent variables (Gaussian here, with dimension about the same as the number of features), the Keras metric used in the compile step, the learning rate, the type of activation function, and so on. I kept the original settings as found here and elsewhere, except for the learning rate and number of epochs.

Instead, I focused on front-end modifications. In particular, the enhanced version of my implementation produces a full synthetic dataset at each epoch, and computes the fit with the real data each time. It barely increases the amount of time needed to run a full GAN cycle. The fit is measured as the correlation matrix distance between the real and synthesized data. This metric is always between 0 and 1, with 0 being best. The goal was to replicate the correlation structure in the real dataset, thus the choice of this particular metric. The final synthetic data is the one obtained at the epoch where the correlation matrix distance is best (minimum). In addition, it must not come from an early epoch. For instance if the number of epochs id 10,000, I check the correlation matrix distance starting at epoch 7500.

In addition, the seed is an integral and important part of my algorithm. I made the results replicable, so if you run the program twice with the same seed, you will get the same results, unlike in most other implementations. Then, I test various seeds and pick up the one that produces the best results.

10.4.5 Enhanced GAN: methodology and results

To achieve better results, I explained how to process missing data separately, or applying a different GAN to specific segments of your population. Or a different Gan for each group of features resulting from feature clustering, as discussed in section 7.2.1.

Transforming or normalizing your data (for instance, decorrelate) may also lead to better synthetization. For instance, say your real data X is an $n \times m$ array with n rows – the observations – and m columns – the features. Assume that all the features have been normalized, thus having zero mean and unit variance. You can transform X to obtain $Y = XW$, where $\text{Cov}[Y] = W^T C_X W$ is a $m \times m$ diagonal matrix, ^{T} denotes the transposition operator, $C_X = \text{Cov}[X]$ and W has size $m \times m$. The transformed data Y consists of uncorrelated features. To achieve this goal take $W = C^{-1/2}$. There are multiple possible square roots, and this transformation is discussed in section 10.3.2. You can use an iterative algorithm to compute the matrix square root. Then synthesize Y (instead of X) and let Z be the resulting data. Your final (un-transformed) synthesized data is $X' = ZW^{-1}$, with the exact same correlation matrix as your real data X. This procedure is known as decorrelation [Wiki], followed by recorrelation.

However, the easiest solution is as follows. The GAN algorithm is very sensitive to the seed, which determines the initial configuration. In Figure 10.2, I compare two trajectories of the gradient descent based on two different seeds. Clearly, `seed=103` does a much better job than `seed=102`, attaining and staying in regions of lower loss much faster than `seed=102`. Thus the solution consists in trying different seeds. Not only that, but even with a same seed, the iterates (called epochs) oscillate wildly. In short, you could get a much better synthetization if you stop after 9800 epochs rather than (say) 10,000. The problem is further compounded by the fact that the loss function may achieve an optimization goal different from what you are looking for.

I address these issues in my enhanced version of GAN. To use it, set `mode='Enhanced'` in the Python code in section 10.7.2. Given one instance of GAN corresponding to a specific seed, it will retain the best

synthetic data (in other words, the best model produced by Keras) based on a distance function of your choice, rather than the one obtained at the last epoch and very dependent on the loss function. In my code, I was interested in synthetic data good a mimicking the correlation structure present in the real dataset, so I wrote my own function `gan_distance`, measuring the correlation distance between real and synthetic data at each epoch. It is based on the correlation matrix distance. Of course, you can modify that function to meet your own needs.

The enhancement techniques described so far are front-end: they do not modify the internal components of GAN. They are also easy to understand and implement, contributing to explainable AI. But you can also dig into the GAN black-box internals and modify some of the low-level components. This is facilitated to some extend by the Keras library, which offers some tools, for instance to customize the loss function. These back-end enhancements require more knowledge about how GANs work. You can write your own function `custom_distance` and have Keras "digest" it by choosing the option `model.compile(metrics=[custom_distance])` in your GAN model. See additional documentation here and here. Another possibility is to use an adaptive learning rate: see how to do it here. Finally, being able with reinforcement learning [Wiki] to reward configurations minimizing your own front-end distance function (rather than the default loss function) would be helpful, see [126]. By configuration, I mean the updated model and its set of updated weights obtained at the end of each epoch.

To summarize, the enhanced version of my implementation has the following upgrades:

- Replicable results
- Missing data treated separately
- Run multiple versions each with a different seed, use the best version
- Binary data: 0 and 1 processed with two different GANs (available in future version)
- Stop when your customized distance between real and synthetic data is minimum

The last feature requires `mode='Enhanced'` in the Python code. The increased performance of the enhanced version can be seen in the Figure 10.2, comparing standard (left) with enhanced mode (right). Also, the GAN synthetization in Figure 10.6 (right plot) uses the enhanced mode. It features an example on tabular data where GAN outperforms copulas. An additional upgrade is to blend copula methods with GAN. This is done in the SDV library: see code in section 10.4.1. Finally, applying GAN on decorrelated data (followed by a re-correlation step) as discussed at the beginning of this section, is guaranteed to preserve the exact correlation structure in the real data. This operation is fully reversible.

10.5 Comparing GANs with the copula method

On the medical data set, the copula method performs better as illustrated in Figures 10.3 and 10.4, and it is a lot faster. Unlike my copula method in section 10.2.1, many GAN implementations do not produce replicable results. However, this problem is solved in my implementation in section 10.7. Also GAN is very sensitive to the initial configuration (the seed), and oscillations from one epoch to the next one are rather large, even after 10,000 epochs. This latter issue may actually be an advantage: trying different seeds and/or using a stopping rule based on the quality of the synthetisation in any given epoch, leads to substantial improvements.

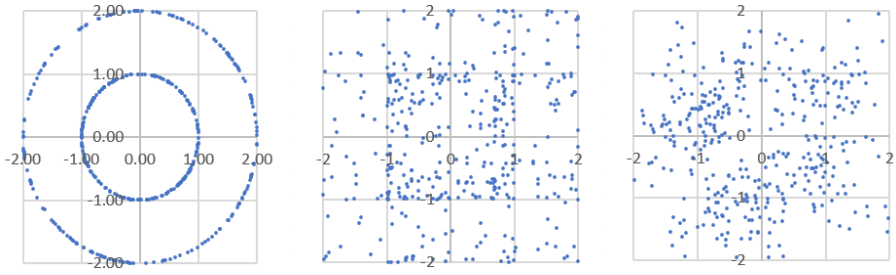


Figure 10.6: Real data (left), copula (middle) and GAN synthetization (right)

GAN has many hyperparameters that can be fine-tuned, even the loss function. This can lead to overfitting and makes the method less suitable as a black-box, compared to the parameter-free copula technique. Copulas are also less sensitive to outliers and small modifications of the real data, at least in this context (tabular data) and when using a parameter-free method based on empirical quantiles. But unlike GAN, they may not be able to generate synthetic data outside the range of observations. There are solutions to this problem: noise injection, or using parametric rather than empirical quantiles. Parametric quantiles are obtained by fitting a

feature or pair of features to a known probability distribution such as a mixture of Gaussians (GMM). It can also lead to overfitting. A nice feature of copulas is that it easily works with a mix of categorical, ordinal and continuous features.

Another issue with GAN, on this particular dataset, is the fact that the feature correlations in the synthetic data are exaggerated. This is true in the early epochs, and this phenomenon is attenuated in the last epochs, but it is still strongly noticeable, for instance on the left plot in Figure 10.2.

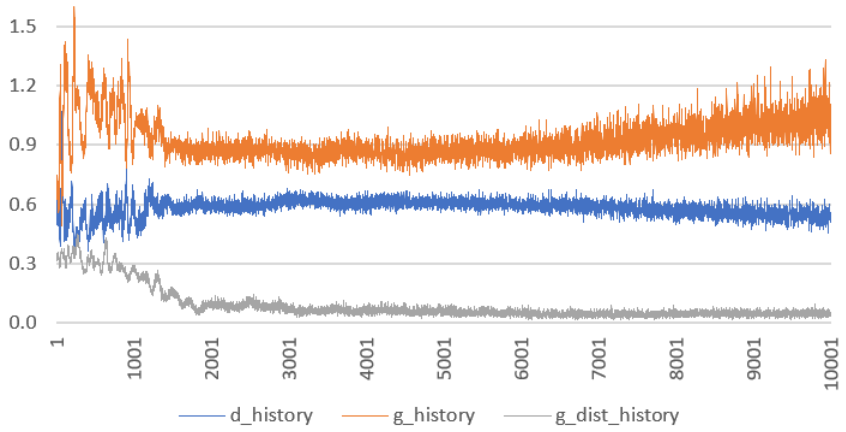


Figure 10.7: Loss function (orange) and distance (grey), circle dataset

The superiority of copulas, as seen in Figure 10.3 and 10.4, is due to the goal being achieved here: mimicking the correlation structure in the real data. Indeed, copulas are perfect at that. But what if the goal is different, or if the correlation matrix fails to capture the patterns in the real data? I tried with an artificial example, where most of the feature correlations are zero, but with strong non-linear dependencies instead. The dataset in question has 9 features; the last one is also a binary response for classification purposes, as in the medical dataset. The first two features represent points lying on two concentric circles: see Figure 10.6. It is trivial to synthesize the whole 9-D dataset with any method. However, I use it for illustration purposes as some real-life datasets have similar undetected or invisible patterns buried in very high dimensions.

In this case, it turns out that GAN does a better job. Both GAN and copula generate the correct correlation structure, though copula fails to recognize the circular patterns. Also, GAN iterations are very stable in this example, compared to the medical dataset. The copula method used in this example is based on two copulas, one for each group: the two concentric circles correspond to the two groups. This is not the case for GAN which is somewhat at a disadvantage, because of using the same model for both groups.

I computed the correlation between $X_1^2 + X_2^2$ (based on the first two features) and the binary response X_9. In the real data consisting of 400 observations, by design it is exactly 1 even though the correlation between X_1 and X_2 is zero. The copula method yields 0.014, and GAN yields a dramatic improvement with a value of 0.723, much closer to 1. Even when using a separate copula for each group, copula yields a much better correlation of 0.676 but still worse than GAN, even though GAN is based on a single model for both groups. Another interesting correlation is between X_1 and the response X_9. In the real dataset, the value is 0.067. The copula yields 0.128, and GAN yields 0.070. So GAN is slightly better. Given the way the real dataset was built, the true value would be zero if it had an infinite number of observations. More correlations are displayed in Table 10.2, and the full list is easy to obtain from the Excel spreadsheet.

Feature pair	Real data	Copula synth.	GAN synth.
$X_9, X_1^2 + X_2^2$	1.0000	0.0136	0.7235
X_1, X_9	0.0662	0.1281	0.0700
X_1, X_2	0.0641	0.1069	0.0660
X_1, X_3	1.0000	1.0000	0.9976
X_2, X_3	0.0641	0.1069	0.0495
$X_1, X_1^2 + X_2^2$	0.0662	0.1906	0.1186
X_1, X_5	−0.0278	−0.0278	−0.0047

Table 10.2: Correlations on 9D circle dataset: real vs copula and GAN

I stopped GAN at epoch 7727 which achieves the minimum overall correlation matrix distance of 0.017, a very good performance since the best possible value is zero, and the worst case is one. Epoch 10001 (the last one) yields 0.036 and epoch 9998 yields 0.057: it shows the benefit of using the enhanced version of GAN to capture the 0.017 minimum. The evolution of this GAN system is pictured in Figure 10.7. The labels `d_history`, `g_history`, `g_dist` are as in the Python code with "d" standing for the discriminator model, and "g" standing for the generator model in GAN (the one that creates the synthetization). The details are in my spreadsheet `circle8d.xlsx` on GitHub, here.

Note that even if GAN is not as good as copulas at replicating the correlation structure in the medical dataset, it is possible to first decorrelate the data, then apply GAN (or any method!) and then re-correlate, as explained in section 10.4.5. With this transformation, any synthetization algorithm that generates uncorrelated data will perfectly replicate the correlations found in the real data, after the re-correlation step.

10.5.1 Conclusion: getting the best out of copulas and GAN

I already discussed how to improve GANs in the context of tabular data, for instance by applying GAN to the decorrelated real data, or using your own distance metric or loss function, fine-tuning the learning rate, or using a faster version of the gradient descent such as LightGBM. Some improvements apply both to GANs and copulas: using a separate GAN model or copula for specific groups of observations, or for specific groups of features based on feature clustering. Or testing 10 different GANs (using different seeds) or 10 different copulas: the latter is a lot faster. Some implementations blend GAN and copulas. See for instance the CopulaGAN module in the SVD library.

To improve copulas methods specifically, you can replace the parameter-free empirical quantiles by quantiles from a parametric family of distributions, fit to the data, with parameters estimated on the real data. For instance, a Gaussian mixture model (GMM) for features with multimodal distribution. Then generate synthetic data using an iterative process based on the Hellinger distance. This distance measures the fit between the real data and the current synthesized version. And proceed iteratively as in GAN: successive iterations are obtained following the gradient path of the Hellinger distance, viewed as as multivariate function of the parameters of your model. In essence, this is very similar to the GAN approach, and can be done with or without neural networks.

In other words, you can improve GANs by integrating them with copulas and follow a gradient path that leads to an optimum of some discriminating function. And you can improve copulas by using a gradient descent algorithm (or stepwise procedure focusing on 2 parameters at a time) to navigate the parameter space until you optimize the Hellinger distance. In the end, the two techniques with the respective improvements may not be that different, especially when using multivariate parametric distributions spanning across multiple features, for the copula.

10.6 Data synthetization explained in one picture

Figure 10.8 summarizing many of the elements discussed in this book, is organized as follows. Dashed blue lines are associated to GANs (generative adversarial networks), where the goal is to produce a sequence of synthetic datasets that get better and better at mimicking the structure present in the real data, over successive iterations. The diagram features 5 such iterations, with the synthetized datasets denoted as $S_1, S_2, \ldots, S_5$. Typically, GANs follow the gradient of h to reach an optimum configuration q that can not be classified as non-real anymore. Synthetic data that gets closer to the real data gets rewarded in this reinforcement learning technique. Like any simulation-intensive method, training the neural network can be time-consuming, and this black-box approach may lack explainability.

Dashed pink lines are associated to modeling techniques (generative model, GMM) where synthetic data is obtained by simulating the underlying model using the parameter values estimated on the real data, that is, $q_k = p$ for all k. In case of GMM (Gaussian mixture models), the parameters are the cluster centers, the covariance matrix attached to each cluster, and the proportions of the mixture. For stationary time series, the parameter is typically the autocorrelation function (ACF). In some applications including when using copulas, the EDPD (empirical probability density function) is used instead.

The goal is to mimic the structure in the real data, not the real data itself. The structure is represented by a parametric configuration denoted as p in the real data. I use the notation $p_1, \ldots, p_5$ for the structures found in the 5 synthetic data sets. The quality h_k of the synthetic data set k is the distance between p_k and p, based on the Hellinger metric or some discriminating function in the case of GAN. It is assumed that the real data has been normalized (transformed) before synthesizing. "Estim. param." stands for estimated parameters in the diagram, though sometimes the parameters can be a function or matrix rather than a set of elements.

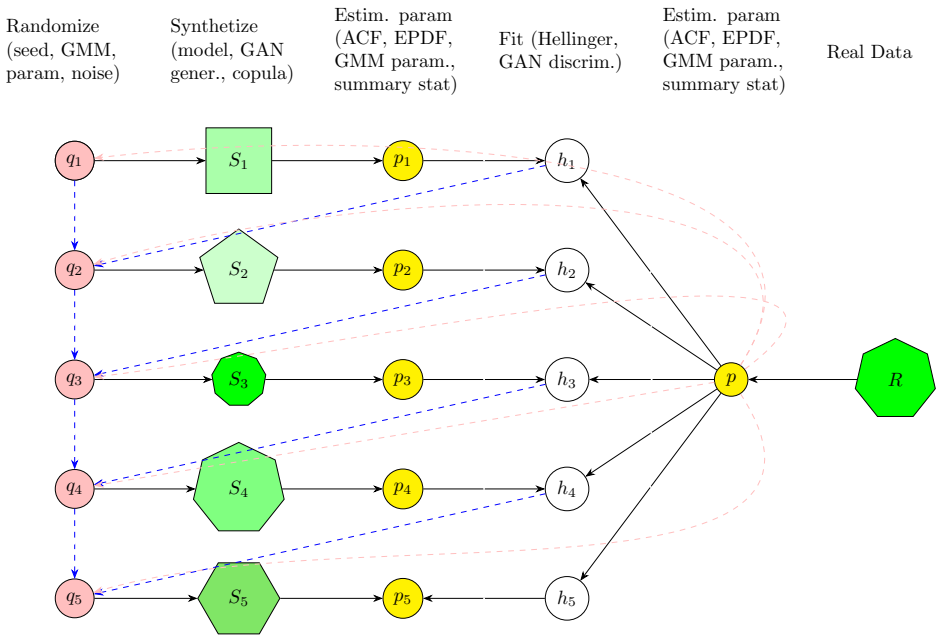


Figure 10.8: Data synthetization: general schema

10.7 Python code: GAN to synthesize medical data

I broke down the program into three pieces. First, reading the data and removing observations with missing values. During this step, I also run a classification algorithm (random forest) on the real data, as the goal is to discriminate between patients likely to get cancer, from the other ones. The rightmost column in the tabular data set, called `Outcome`, is the cancer indicator (1 = yes, 0 = no).

The second step is the core of the GAN procedure, including the production of synthetic data. Finally, the last part performs model evaluation – the fit between real and synthetic data – using the TableEvaluator library my matrix correlation distance defined in step 2. In the last part, I classify again the data with the random classifier, but this time the synthesized data, for comparison purposes with the classification on the real data obtained in the first step.

To run in enhanced mode, set `mode='Enhanced'`. The full program named `GAN_diabetes.py` is also on my GitHub repository, here. The real dataset `diabetes.csv` can be found here.

10.7.1 Classification problem

This step reads the data, removes observations with missing values, and performs a classification on the real data. It also imports all the libraries needed. and eliminates all sources of uncontrollable randomness by using a seed for all the random number generators involved (native Python, TensorFlow, Numpy). This leads to replicable results. The hyperparemeter `learning_rate` is also initialized here. You may need the most recent version of Numpy, TensorFlow and Pip.

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import random as python_random
from tensorflow import random
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam # type of gradient descent optimizer
from numpy.random import randn
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

data = pd.read_csv('diabetes.csv')
```

```
# data located at https://github.com/VincentGranville/Main/blob/main/diabetes.csv

# rows with missing data must be treated separately: I remove them here
data.drop(data.index[(data["Insulin"] == 0)], axis=0, inplace=True)
data.drop(data.index[(data["Glucose"] == 0)], axis=0, inplace=True)
data.drop(data.index[(data["BMI"] == 0)], axis=0, inplace=True)
# no further data transformation used beyond this point
data.to_csv('diabetes_clean.csv')

print (data.shape)
print (data.tail())
print (data.columns)

seed = 102 # to make results replicable
np.random.seed(seed) # for numpy
random.set_seed(seed) # for tensorflow/keras
python_random.seed(seed) # for python

adam = Adam(learning_rate=0.001) # also try 0.01
latent_dim = 10
n_inputs = 9 # number of features
n_outputs = 9 # number of features


#--- STEP 1: Base Accuracy for Real Dataset

features = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
    'BMI', 'DiabetesPedigreeFunction', 'Age']
label = ['Outcome'] # OutCome column is the label (binary 0/1)
X = data[features]
y = data[label]

# Real data split into train/test dataset for classification with random forest

X_true_train, X_true_test, y_true_train, y_true_test = train_test_split(X, y,
    test_size=0.30, random_state=42)
clf_true = RandomForestClassifier(n_estimators=100)
clf_true.fit(X_true_train,y_true_train)
y_true_pred=clf_true.predict(X_true_test)
print("Base Accuracy: %5.3f" % (metrics.accuracy_score(y_true_test, y_true_pred)))
print("Base classification report:\n",metrics.classification_report(y_true_test,
    y_true_pred))
```

10.7.2 GAN method

The main function is `train`. Adding layers to the networks, combining the discriminator and generator models of GAN, selecting the loss functions and so on, and compiling the models, is done in the satellite functions defined here. In addition, my matrix correlation distance function is defined in this step. It is heavily used in the enhanced version, when `mode=='Enhanced'`. The last instruction saves the synthesized data `data_fake` to a CSV file.

```
#--- STEP 2: Generate Synthetic Data

def generate_latent_points(latent_dim, n_samples):
   x_input = randn(latent_dim * n_samples)
   x_input = x_input.reshape(n_samples, latent_dim)
   return x_input

def generate_fake_samples(generator, latent_dim, n_samples):
   x_input = generate_latent_points(latent_dim, n_samples) # random N(0,1) data
   X = generator.predict(x_input,verbose=0)
   y = np.zeros((n_samples, 1)) # class label = 0 for fake data
   return X, y
```

```
def generate_real_samples(n):
   X = data.sample(n) # sample from real data
   y = np.ones((n, 1)) # class label = 1 for real data
   return X, y

def define_generator(latent_dim, n_outputs):
   model = Sequential()
   model.add(Dense(15, activation='relu', kernel_initializer='he_uniform',
       input_dim=latent_dim))
   model.add(Dense(30, activation='relu'))
   model.add(Dense(n_outputs, activation='linear'))
   model.compile(loss='mean_absolute_error', optimizer=adam,
       metrics=['mean_absolute_error']) #
   return model

def define_discriminator(n_inputs):
   model = Sequential()
   model.add(Dense(25, activation='relu', kernel_initializer='he_uniform',
       input_dim=n_inputs))
   model.add(Dense(50, activation='relu'))
   model.add(Dense(1, activation='sigmoid'))
   model.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])
   return model

def define_gan(generator, discriminator):
   discriminator.trainable = False # weights must be set to not trainable
   model = Sequential()
   model.add(generator)
   model.add(discriminator)
   model.compile(loss='binary_crossentropy', optimizer=adam)
   return model

def gan_distance(data, model, latent_dim, nobs_synth):

   # generate nobs_synth synthetic rows as X, and return it as data_fake
   # also return correlation distance between data_fake and real data

   latent_points = generate_latent_points(latent_dim, nobs_synth)
   X = model.predict(latent_points, verbose=0)
   data_fake = pd.DataFrame(data=X, columns=['Pregnancies', 'Glucose', 'BloodPressure',
       'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'])

   # convert Outcome field to binary 0/1
   outcome_mean = data_fake.Outcome.mean()
   data_fake['Outcome'] = data_fake['Outcome'] > outcome_mean
   data_fake["Outcome"] = data_fake["Outcome"].astype(int)

   # compute correlation distance
   R_data   = np.corrcoef(data.T) # T for transpose
   R_data_fake = np.corrcoef(data_fake.T)
   g_dist = np.average(abs(R_data-R_data_fake))
   return(g_dist, data_fake)

def train(g_model, d_model, gan_model, latent_dim, mode, n_epochs=10000, n_batch=128,
    n_eval=200):

   # determine half the size of one batch, for updating the discriminator
   half_batch = int(n_batch / 2)
   d_history = []
   g_history = []
   g_dist_history = []
   if mode == 'Enhanced':
      g_dist_min = 999999999.0

   for epoch in range(0,n_epochs+1):
```

```
        # update discriminator
        x_real, y_real = generate_real_samples(half_batch) # sample from real data
        x_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
        d_loss_real, d_real_acc = d_model.train_on_batch(x_real, y_real)
        d_loss_fake, d_fake_acc = d_model.train_on_batch(x_fake, y_fake)
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        # update generator via the discriminator error
        x_gan = generate_latent_points(latent_dim, n_batch) # random input for generator
        y_gan = np.ones((n_batch, 1))           # label = 1 for fake samples
        g_loss_fake = gan_model.train_on_batch(x_gan, y_gan)
        d_history.append(d_loss)
        g_history.append(g_loss_fake)

        if mode == 'Enhanced':
           (g_dist, data_fake) = gan_distance(data, g_model, latent_dim, nobs_synth=400)
           if g_dist < g_dist_min and epoch > int(0.75*n_epochs):
             g_dist_min = g_dist
             best_data_fake = data_fake
             best_epoch = epoch
        else:
           g_dist = -1.0
        g_dist_history.append(g_dist)

        if epoch % n_eval == 0: # evaluate the model every n_eval epochs
           print('>%d, d1=%.3f, d2=%.3f d=%.3f g=%.3f g_dist=%.3f' % (epoch, d_loss_real,
               d_loss_fake, d_loss, g_loss_fake, g_dist))
           plt.subplot(1, 1, 1)
           plt.plot(d_history, label='d')
           plt.plot(g_history, label='gen')
           # plt.show() # un-comment to see the plots
           plt.close()

    OUT=open("history.txt","w")
    for k in range(len(d_history)):
       OUT.write("%6.4f\t%6.4f\t%6.4f\n" %(d_history[k],g_history[k],g_dist_history[k]))
    OUT.close()

    if mode == 'Standard':
       # best synth data is assumed to be the one produced at last epoch
       best_epoch = epoch
       (g_dist_min, best_data_fake) = gan_distance(data, g_model, latent_dim,
           nobs_synth=400)

    return(g_model, best_data_fake, g_dist_min, best_epoch)

#--- main part for building & training model

discriminator = define_discriminator(n_inputs)
discriminator.summary()
generator = define_generator(latent_dim, n_outputs)
generator.summary()
gan_model = define_gan(generator, discriminator)

mode = 'Enhanced' # options: 'Standard' or 'Enhanced'
model, data_fake, g_dist, best_epoch = train(generator, discriminator, gan_model,
    latent_dim, mode)
data_fake.to_csv('diabetes_synthetic.csv')
```

10.7.3 GAN Evaluation and post-classification

Evaluates the quality of the synthetic data with the TableEvaluator library and `g_dist`, the matrix correlation distance obtained in the previous step. Also, performs classification, but this time on the synthetic data, to compare with the results obtained on the real data in Step 1.

```
#--- STEP 3: Classify synthetic data based on Outcome field

features = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
    'BMI', 'DiabetesPedigreeFunction', 'Age']
label = ['Outcome']
X_fake_created = data_fake[features]
y_fake_created = data_fake[label]
X_fake_train, X_fake_test, y_fake_train, y_fake_test = train_test_split(X_fake_created,
    y_fake_created, test_size=0.30, random_state=42)
clf_fake = RandomForestClassifier(n_estimators=100)
clf_fake.fit(X_fake_train,y_fake_train)
y_fake_pred=clf_fake.predict(X_fake_test)
print("Accuracy of fake data model: %5.3f" % (metrics.accuracy_score(y_fake_test,
    y_fake_pred)))
print("Classification report of fake data
    model:\n",metrics.classification_report(y_fake_test, y_fake_pred))


#--- STEP 4: Evaluate the Quality of Generated Fake Data With g_dist and Table_evaluator

from table_evaluator import load_data, TableEvaluator

table_evaluator = TableEvaluator(data, data_fake)
table_evaluator.evaluate(target_col='Outcome')
# table_evaluator.visual_evaluation()

print("Avg correlation distance: %5.3f" % (g_dist))
print("Based on epoch number: %5d" % (best_epoch))
```

Glossary

Autoregressive process	Auto-correlated time series, as described in section 3.4. Time-continuous versions include Gaussian processes and Brownian motions, while random walks are a discrete example; two-dimensional versions exist. These processes are essentially integrated white noise. See pages 50, 98, 172
Binning	Feature binning consists of aggregating the values of a feature into a small number of bins, to avoid overfitting and reduce the number of nodes in methods such as naive Bayes, neural networks, or decision trees. Binning can be applied to two or more features simultaneously. I discuss optimum binning in this book. See pages 38, 74, 268
Boosted model	Blending of several models to get the best of each one, also referred to as ensemble methods. The concept is illustrated with hidden decision trees in this book. Other popular examples are gradient boosting and AdaBoost. See pages 37, 277
Bootstrapping	A data-driven, model-free technique to estimate parameter values, to optimize goodness-of-fit metrics. Related to resampling in the context of cross-validation. In this book, I discuss parametric bootstrap on synthetic data that mimics the actual observations. See pages 16, 97, 229, 277
Confidence Region	A confidence region of level γ is a 2D set of minimum area covering a proportion γ of the mass of a bivariate probability distribution. It is a 2D generalization of confidence intervals. In this book, I also discuss dual confidence regions – the analogous of credible regions in Bayesian inference. See pages 13, 16, 19, 21, 30, 226, 227, 263, 266
Cross-validation	Standard procedure used in bootstrapping, and to test and validate a model, by splitting your data into training and validation sets. Parameters are estimated based on training set data. An alternative to cross-validation is testing your model on synthetic data with known response. See pages 16, 38, 94, 100, 138, 204, 268, 277
Decision trees	A simple, intuitive non-linear modeling techniques used in classification problems. It can handle missing and categorical data, as well as a large number of features, but requires appropriate feature binning. Typically one blends multiple binary trees each with a few nodes, to boost performance. See pages 37, 38, 40, 42, 277, 278
Dimension reduction	A technique to reduce the number of features in your dataset while minimizing the loss in predictive power. The most well known are principal component analysis and feature selection to maximize goodness-of-fit metrics. See pages 13, 17, 278, 279
Empirical distribution	Cumulative frequency histogram attached to a statistic (for instance, nearest neighbor distances), and based on observations. When the number of observations tends to infinity and the bin sizes tend to zero, this step function tends to the theoretical cumulative distribution function of the statistic in question. See pages 17, 97, 121, 130, 150, 213, 216, 222, 228, 233, 235, 247
Ensemble methods	A technique consisting of blending multiple models together, such as many decision trees with logistic regression, to get the best of each method and outperform each method taken separately. Examples include boosting, bagging, and AdaBoost. In this book, I discuss hidden decision trees. See pages 37, 84, 277

Explainable AI — Automated machine learning techniques that are easy to interpret are referred to as interpretable machine learning or explainable artificial intelligence. As much as possible, the methods discussed in this book belong to that category. The goal is to design black-box systems less likely to generate unexpected results with unintended consequences. See pages 14, 36, 70, 75, 84, 91, 129, 141, 176, 192, 230

Feature selection — Features – as opposed to the model response – are also called independent variables or predictors. Feature selection, akin to dimensionality reduction, aims at finding the minimum subset of variables with enough predictive power. It is also used to eliminate redundant features and find causality (typically using hierarchical Bayesian models), as opposed to mere correlations. Sometimes, two features have poor predictive power when taken separately, but provide improved predictions when combined together. See pages 13, 16, 38, 95, 98, 259, 267, 277, 279

Generative model — Bayesian Gaussian mixtures (GMM) combined with kernel density estimation and the EM algorithm is a classic modeling tool. In this book, I used m-interlacings instead. Generative adversarial networks (GAN) work as follows: the generator creates new observations and the discriminator tests whether the new observations are statistically indistinguishable from training set data. When this goal is achieved, the new observations is your synthetic data. New observations can also be generated via parametric bootstrap. See pages 36, 53, 100, 143, 187, 188, 190, 197, 204, 279

Goodness-of-fit — A model fitting criterion or metric to assess how a model or sub-model fits to a dataset, or to measure its predictive power on a validation set. Examples include R-squared, Chi-squared, Kolmogorov-Smirnov, error rate such as false positives and other metrics discussed in this book. See pages 16, 57, 94, 95, 268, 277, 279

Gradient methods — Iterative optimization techniques to find the minimum of maximum of a function, such as the maximum likelihood. When there are numerous local minima or maxima, use swarm optimization. Gradient methods (for instance, stochastic gradient descent or Newton's method) assume that the function is differentiable. If not, other techniques such as Monte Carlo simulations or the fixed-point algorithm can be used. Constrained optimization involves using Lagrange multipliers. See pages 16, 32, 56, 90

Graph structures — Graphs are found in decision trees, in neural networks (connections between neurons), in nearest neighbors methods (NN graphs), in hierarchical Bayesian models, and more. See pages 71, 75, 205, 271, 272

Hyperparameter — An hyperparameter is used to control the learning process: for instance, the dimension, the number of features, parameters, layers (neural networks) or clusters (clustering problem), or the width of a filtering window in image processing. By contrast, the values of other parameters (typically node weights in neural networks or regression coefficients) are derived via training. See pages 30, 57, 71, 76, 102, 136, 191, 278

Link function — A link function maps a nonlinear relationship to a linear one so that a linear model can be fit, and then mapped back to the original form using the inverse function. For instance, the logit link function is used in logistic regression. Generalizations include quantile functions and inverse sigmoids in neural network to work with additive (linear) parameters. See pages 14, 17, 278

Logistic regression — A generalized linear regression method where the binary response (fraud/non-fraud or cancer/non-cancer) is modeled as a probability via the logistic link function. Alternatives to the iterative maximum likelihood solution are discussed in this book. See pages 17, 34, 37, 41, 277, 278

Neural network — A blackbox system used for predictions, optimization, or pattern recognition especially in computer vision. It consists of layers, neurons in each layer, link functions to model non-linear interactions, parameters (weights associated to the connections between neurons) and hyperparameters. Networks with several layers are called deep neural networks. Also, neurons are sometimes called nodes. See pages 70, 74, 76, 84, 102, 277, 278

NLP
: Natural language processing is a set of techniques to deal with unstructured text data, such as emails, automated customer support, or webpages downloaded with a crawler. The example discussed in section 18.5 deals with creating a keyword taxonomy based on parsing Google search result pages. Text generation is referred to as NLG or natural language generation, using large language models (LLM). See pages 37, 270

Numerical stability
: This issue occurring in unstable optimization problems typically with multiple minima or maxima, is frequently overlooked and leads to poor predictions or high volatility. It is sometimes referred to as ill-conditioned problems. I explain how to fix it in several examples in this book, for instance in section 3.4.2. Not to be confused with numerical precision. See pages 13, 15, 60

Overfitting
: Using too many unstable parameters resulting in excellent performance on the training set, but poor performance on future data or on the validation set. It typically occurs with numerically unstable procedures such as regression (especially polynomial regression) when the training set is not large enough, or in the presence of wide data (more features than observations) when using a method not suited to this situation. The opposite is underfitting. See pages 16, 93, 102, 130, 136, 277, 279

Predictive power
: A metric to assess the goodness-of-fit or performance of a model or subset of features, for instance in the context of dimensionality reduction or feature selection. Typical metrics include R-squared, or confusion matrices in classification. See pages 39, 41, 45, 129, 267, 269, 278

R-squared
: A goodness-of-fit metric to assess the predictive power of a model, measured on a validation set. Alternatives include adjusted R-squared, mean absolute error and other metrics discussed in this book. See pages 13, 16, 36, 57, 91, 94, 96, 98, 105, 278, 279

Random number
: Pseudo-random numbers are sequences of binary digits, usually grouped into blocks, satisfying properties of independent Bernoulli trials. In this book, the concept is formally defined, and strong pseudo-number generators are built and used in computer-intensive simulations. See pages 30, 149, 156, 273

Regression methods
: I discuss a unified approach to all regression problems in chapter 1. Traditional techniques include linear, logistic, Bayesian, polynomial and Lasso regression (to deal with numerical instability and overfitting), solved using optimization techniques, maximum likelihood methods, linear algebra (eigenvalues and singular value decomposition) or stepwise procedures. See pages 13, 14, 16, 17, 20, 28, 37, 41, 47, 51, 53, 57, 90, 96, 102, 109, 278, 279

Supervised learning
: Techniques dealing with labeled data (classification) or when the response is known (regression). The opposite is unsupervised learning, for instance clustering problems. In-between, you have semi-supervised learning and reinforcement learning (favoring good decisions). The technique described in chapter 1 fits into unsupervised regression. Adversarial learning is testing your model against extreme cases intended to make it fail, to build better models. See pages 279

Synthetic data
: Artificial data simulated using a generative model, typically a mixture model, to enrich existing datasets and improve the quality of training sets. Called augmented data when blended with real data. See pages 13, 14, 16, 18, 28, 30, 34, 36, 49, 53, 56, 70, 71, 76, 89, 95, 106, 113, 119, 130, 149, 162, 168, 176, 190, 197, 204, 264, 273, 277

Tensor
: Matrix generalization with three of more dimensions. A matrix is a two-dimensional tensor. A triple summation with three indices is represented by a three-dimensional tensor, while a double summation involves a standard matrix. See pages 70, 75

Training set
: Dataset used to train your model in supervised learning. Typically, a portion of the training set is used to train the model, the other part is used as validation set. See pages 14, 16, 18, 21, 30, 37, 41, 57, 73, 89, 96, 102, 106, 204, 268, 277, 279

Validation set
: A portion of your training set, typically 20%, used to measure the actual performance of your predictive algorithm outside the training set. In cross-validation and bootstrapping, the training and validation sets are split into multiple subsets to get a better sense of variations in the predictions. See pages 16, 28, 42, 57, 94, 102, 130, 204, 268, 277, 278, 279

Bibliography

[1] Weighted percentiles using numpy. *Forum discussion*, 2020. StackOverflow [Link]. 102

[2] Jan Ackmann et al. Machine-learned preconditioners for linear solvers in geophysical fluid flows. *Preprint*, pages 1–19, 2020. arXiv:2010.02866 [Link]. 94

[3] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley, fourth edition, 2016. [Link]. 222

[4] José M. Amigó, Roberto Dale, and Piergiulio Tempesta. A generalized permutation entropy for random processes. *Preprint*, pages 1–9, 2012. arXiv:2003.13728 [Link]. 233

[5] Luc Anselin. *Point Pattern Analysis: Nearest Neighbor Statistics*. The Center for Spatial Data Science, University of Chicago, 2016. Slide presentation [Link]. 220

[6] Insaf Ashrapov. Tabular gans for uneven distribution. *Preprint*, pages 1–11, 2020. arXiv:2010.00638 [Link]. 136

[7] Adrian Baddeley. Spatial point processes and their applications. In Weil W., editor, *Stochastic Geometry. Lecture Notes in Mathematics*, pages 1–75. Springer, Berlin, 2007. [Link]. 219

[8] David Bailey and Richard Crandall. Random generators and normal numbers. *Experimental Mathematics*, 11, 2002. Project Euclid [Link]. 166

[9] N. Balakrishnan and C.R. Rao (Editors). *Order Statistics: Theory and Methods*. North-Holland, 1998. 222, 236

[10] Christopher Beckham and Christopher Pal. A step towards procedural terrain generation with GANs. *Preprint*, pages 1–5, 2017. arXiv:1707.03383 [Link]. 189

[11] Rabi Bhattacharya and Edward Waymire. *Random Walk, Brownian Motion, and Martingales*. Springer, 2021. 167

[12] Barbara Bogacka. *Lecture Notes on Time Series*. 2008. Queen Mary University of London [Link]. 50

[13] B. Bollobas and P. Erdös. Cliques in random graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 80(3):419–427, 1976. [Link]. 223

[14] Miklos Bona. *Combinatorics of Permutations*. Routledge, second edition, 2012. 233

[15] Ali Borji. Pros and cons of GAN evaluation measures: New developments. *Preprint*, pages 1–35, 2021. arXiv:2103.09396 [Link]. 136

[16] Peter Borwein, Stephen K. Choi, and Michael Coons. Completely multiplicative functions taking values in $\{-1, 1\}$. *Transactions of the American Mathematical Society*, 362(12):6279–6291, 2010. [Link]. 241

[17] Peter Borwein and Michael Coons. Transcendence of power series for some number theoretic functions. *Proceedings of the American Mathematical Society*, 137(4):1303–1305, 2009. [Link]. 243

[18] Oliver Bröker and Marcus J. Groteb. Sparse approximate inverse smoothers for geometric and algebraic multigrid. *Applied Numerical Mathematics*, 41(1):61–80, 2002. 91

[19] H. M. Bui and M. B. Milinovich. Gaps between zeros of the Riemann zeta-function. *Quarterly Journal of Mathematics*, 69(2):402–423, 2018. [Link]. 253

[20] Bartłomiej Błaszczyszyn and Dhandapani Yogeshwaran. Clustering and percolation of point processes. *Preprint*, pages 1–20, 2013. Project Euclid [Link]. 219

[21] Bartłomiej Błaszczyszyn and Dhandapani Yogeshwaran. On comparison of clustering properties of point processes. *Preprint*, pages 1–26, 2013. arXiv:1111.6017 [Link]. 219

[22] Bartłomiej Błaszczyszyn and Dhandapani Yogeshwaran. Clustering comparison of point processes with applications to random geometric models. *Preprint*, pages 1–44, 2014. arXiv:1212.5285 [Link]. 219

[23] Oliver Chikumbo and Vincent Granville. Optimal clustering and cluster identity in understanding high-dimensional data spaces with tightly distributed points. *Machine Learning and Knowledge Extraction*, 1(2):715–744, 2019. 274

[24] Keith Conrad. *L-functions and the Riemann Hypothesis*. 2018. 2018 CTNT Summer School [Link]. 152, 238, 241, 246

[25] Noel Cressie. *Statistic for Spatial Data*. Wiley, revised edition, 2015. 219

[26] D.J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes*. Springer, second edition, 2002. Volume 1 – Elementary Theory and Methods. 171

[27] D.J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes*. Springer, second edition, 2014. Volume 2 – General Theory and Structure. 171

[28] Tilman M. Davies and Martin L. Hazelton. Assessing minimum contrast parameter estimation for spatial and spatiotemporal log-Gaussian Cox processes. *Statistica Neerlandica*, 67(4):355–389, 2013. 265

[29] Marc Deisenroth, A. Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. [Link]. 54

[30] Harold G. Diamond and Wen-Bin Zhang. *Beurling Generalized Numbers*. American Mathematical Society, 2016. Mathematical Surveys and Monographs, Volume 213 [Link]. 153, 249

[31] D.J.Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes – Volume I: Elementary Theory and Methods*. Springer, second edition, 2013. 220

[32] D.J.Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes – Volume II: General Theory and Structure*. Springer, second edition, 2014. 220

[33] David Coupier (Editor). *Stochastic Geometry: Modern Research Frontiers*. Wiley, 2019. 230

[34] Ding-Geng Chen (Editor), Jianguo Sun (Editor), and Karl E. Peace (Editor). *Interval-Censored Time-to-Event Data: Methods and Applications*. Chapman and Hall/CRC, 2012. 221

[35] Khaled Emam, Lucy Mosquera, and Richard Hoptroff. *Practical Synthetic Data Generation*. O'Reilly, 2020. 100

[36] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, volume 5, pages 17–61, 1960. [Link]. 223

[37] Achim Zeileis et al. Colorspace: A toolbox for manipulating and assessing colors and palettes. *Preprint*, pages 1–45, 2019. arXiv:1903.06490 [Link] [R Library]. 189

[38] Arash Farahmand. *Math 55 Lecture Notes*. 2021. University of Berkeley [Link]. 49, 54

[39] W. Feller. On the Kolmogorov-Smirnov limit theorems for empirical distributions. *Annals of Mathematical Statistics*, 19(2):177–189, 1948. [Link]. 222, 229

[40] Alvaro Figueira and Bruno Vaz. Survey on synthetic data generation, evaluation methods and GANs. *New Insights in Machine Learning and Deep Neural Networks*, 2022. MDPI [Link]. 136

[41] Nikos Frantzikinakis. Ergodicity of the Liouville system implies the Chowla conjecture. *Preprint*, pages 1–41, 2016. arXiv [Link]. 243

[42] P. M. Gauthier. Approximating the Riemann zeta-function by polynomials with restricted zeros. *Canadian Mathematical Bulletin*, 62(3):475–478, 2018. [Link]. 253

[43] P. A. Van Der Geest. The binomial distribution with dependent Bernoulli trials. *Journal of Statistical Computation and Simulation*, pages 141–154, 2004. [Link]. 168

[44] Stamatia Giannarou and Tania Stathaki. Shape signature matching for object identification invariant to image transformations and occlusion. 2007. ResearchGate [Link]. 85

[45] Minas Gjoka, Emily Smith, and Carter Butts. Estimating clique composition and size distributions from sampled network data. *Preprint*, pages 1–9, 2013. arXiv:1308.3297 [Link]. 223

[46] B.V. Gnedenko and A. N. Kolmogorov. *Limit Distributions for Sums of Independent Random Variables*. Addison-Wesley, 1954. 172

[47] Manuel González-Navarrete and Rodrigo Lambert. Non-markovian random walks with memory lapses. *Preprint*, pages 1–14, 2018. arXiv [Link]. 167

[48] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. [Link]. 54

[49] Vincent Granville. Estimation of the intensity of a Poisson point process by means of nearest neighbor distances. *Statistica Neerlandica*, 52(2):112–124, 1998. [Link]. 220

[50] Vincent Granville. *Applied Stochastic Processes, Chaos Modeling, and Probabilistic Properties of Numeration Systems*. MLTechniques.com, 2018. [Link]. 153

[51] Vincent Granville. *Stochastic Processes and Simulations: A Machine Learning Perspective*. MLTechniques.com, 2022. [Link]. 52, 60, 172, 182, 213, 214, 216, 217, 221, 223, 249, 253, 267

[52] Vincent Granville, Mirko Krivanek, and Jean-Paul Rasson. Simulated annealing: A proof of convergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:652–656, 1996. 73

[53] Vincent Granville and Richard L Smith. Disaggregation of rainfall time series via Gibbs sampling. *NISS Technical Report*, pages 1–21, 1996. [Link]. 108

[54] Kristen Grauman. Shape matching. 2008. University of Texas, Austin [Link]. 88

[55] Hui Guo et al. Eyes tell all: Irregular pupil shapes reveal gan-generated faces. *Preprint*, pages 1–7, 2021. arXiv:2109.00162 [Link]. 129

[56] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*. O'Reilly, third edition, 2023. 36

[57] Radim Halir and Jan Flusser. Numerically stable direct least squares fitting of ellipses. *Preprint*, pages 1–8, 1998. [Link]. 18, 20

[58] Peter Hall. *Introduction to the theory of coverage processes*. Wiley, 1988. 230

[59] Adam J. Harper. Moments of random multiplicative functions, II: High moments. *Algebra and Number Theory*, 13(10):2277–2321, 2019. [Link]. 149, 249

[60] Adam J. Harper. Moments of random multiplicative functions, I: Low moments, better than squareroot cancellation, and critical multiplicative chaos. *Forum of Mathematics, Pi*, 8:1–95, 2020. [Link]. 149, 151, 249

[61] Adam J. Harper. Almost sure large fluctuations of random multiplicative functions. *Preprint*, pages 1–38, 2021. arXiv [Link]. 151, 243, 249

[62] K. Hartmann, J. Krois, and B. Waske. *Statistics and Geospatial Data Analysis*. Freie Universität Berlin, 2018. E-Learning Project SOGA [Link]. 216

[63] D. R. Heath-Brown. Primes represented by $x^3 + 2y^3$. *Acta Mathematica*, 186:1–84, 2001. [Link]. 245

[64] Markus Herdin. Correlation matrix distance, a meaningful measure for evaluation of non-stationary MIMO channels. *Proc. IEEE 61st Vehicular Technology Conference*, pages 1–5, 2005. [Link]. 136

[65] T. W. Hilberdink and M. L. Lapidus. Beurling Zeta functions, generalised primes, and fractal membranes. *Preprint*, pages 1–31, 2004. arXiv [Link]. 152, 153, 249

[66] Christian Hill. *Learning Scientific Programming with Python*. Cambridge University Press, 2016. [Link]. 20

[67] Robert V. Hogg, Joseph W. McKean, and Allen T. Craig. *Introduction to Mathematical Statistics*. Pearson, eighth edition, 2016. [Link]. 54

[68] Zhiqiu Hu and Rong-Cai Yang. A new distribution-free approach to constructing the confidence region for multiple parameters. *PLOS One*, pages 1–13, 2013. [Link]. 264

[69] Peter Humphries. The distribution of weighted sums of the Liouville function and Pólya's conjecture. *Preprint*, pages 1–33, 2011. arXiv [Link]. 250

[70] Timothy D. Johnson. Introduction to spatial point processes. *Preprint*, page 2008. NeuroImaging Statistics Oxford (NISOx) group [Link][Mirror]. 220

[71] Chigozie Kelechi. Towards efficiency in the residual and parametric bootstrap techniques. *American Journal of Theoretical and Applied Statistics*, 5(5), 2016. [Link]. 98

[72] Denis Kojevnikov, Vadim Marmer, and Kyungchul Song. Limit theorems for network dependent random variables. *Journal of Econometrics*, 222(2):419–427, 2021. [Link]. 220

[73] Samuel Kotz, Tomasz Kozubowski, and Krzystof Podgorski. *The Laplace Distribution and Generalizations: A Revisit with Applications to Communications, Economics, Engineering, and Finance*. Springer, 2001. 234

[74] Faraj Lagum. *Stochastic Geometry-Based Tools for Spatial Modeling and Planning of Future Cellular Networks*. PhD thesis, Carleton University, 2018. [Link]. 219

[75] Günther Last and Mathew Penrose. *Lectures on the Poisson Process*. Cambridge University Press, 2017. 219

[76] Yuk-Kam Lau, Gerald Tenenbaum, and Jie Wu. On mean values of random multiplicative functions. *Proceedings of the American Mathematical Society*, 142(2):409–420, 2013. [Link]. 149, 151

[77] Gary R. Lawlor. A l'Hospital's rule for multivariable functions. *Preprint*, pages 1–13, 2013. arXiv:1209.0363 [Link]. 114

[78] Jing Lei et al. Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, 113:1094–1111, 2018. [Link]. 98

[79] Hui Liu et al. A new model using multiple feature clustering and neural networks for forecasting hourly $PM_{2.5}$ concentrations. *Engineering*, 6:944–956, 2020. [Link]. 134

[80] Mario Lucic et al. Are GANs created equal? a large-scale study. *Proc. NeurIPS Conference*, pages 1–10, 2018. [Link]. 136

[81] G. Last M.A. Klatt and D. Yogeshwaran. Hyperuniform and rigid stable matchings. *Random Structures and Algorithms*, 2:439–473, 2020. [Link][PowerPoint]. 219

[82] Jorge Mateu, Frederic P Schoenberg, and David M Diez. On distances between point patterns and their applications. *Preprint*, pages 1–29, 2010. [Link]. 220

[83] Natarajan Meghanathan. Distribution of maximal clique size of the vertices for theoretical small-world networks and real-world networks. *Preprint*, pages 1–20, 2015. arXiv:1508.01668 [Link]. 223

[84] Masahiro Mine. Probability density functions attached to random Euler products for automorphic L-functions. *Preprint*, pages 1–38, 2020. arXiv [Link]. 249, 250

[85] Christoph Molnar. *Interpretable Machine Learning*. ChristophMolnar.com, 2022. [Link]. 98, 129

[86] Marc-Andreas Muendler. Linear difference equations and autoregressive processes. 2000. University of Berkeley [Link]. 50

[87] V. Kumar Murty. Seminar on Fermat's last theorem. In *Canadian Mathematical Society – Conference Proceedings*, volume 17, Toronto, Canada, 1995. [Link]. 246

[88] Peter Mörters and Yuval Peres. *Brownian Motion*. Cambridge University Press, 2010. Cambridge Series in Statistical and Probabilistic Mathematics, Volume 30 [Link]. 167, 171

[89] Jesper Møller. Introduction to spatial point processes and simulation-based inference. In *International Center for Pure and Applied Mathematics (Lecture Notes)*, Lomé, Togo, 2018. [Link][Mirror]. 220, 233, 265

[90] Jesper Møller and Rasmus P. Waagepetersen. *An Introduction to Simulation-Based Inference for Spatial Point Processes*. Springer, 2003. 220

[91] Jesper Møller and Rasmus P. Waagepetersen. *Statistical Inference and Simulation for Spatial Point Processes*. CRC Press, 2007. 220

[92] S. Ghosh N., Miyoshi, and T. Shirai. Disordered complex networks: energy optimal lattices and persistent homology. *Preprint*, pages 1–44, 2020. arXiv:2009.08811. 213

[93] Saralees Nadarajah. A modified Bessel distribution of the second kind. *Statistica*, 67(4):405–413, 2007. [Link]. 234

[94] Hasan Nasab, Mahdi Tavana, and Mohsen Yousefu. A new heuristic algorithm for the planar minimum covering circle problem. *Production and Manufacturing Research*, pages 142–155, 2014. [Link]. 230

[95] Guillermo Navas-Palencia. Optimal binning: mathematical programming formulation. *Preprint*, pages 1–21, 2020. arXiv:2001.08025 [Link]. 38

[96] Nathan Ng. Large gaps between the zeros of the Riemann zeta function. *Journal of Number Theory*, 128(3):509–556, 2007. [Link]. 253

[97] Sergey I. Nikolenko. *Synthetic Data for Deep Learning*. Springer, 2021. 136

[98] Yosihiko Ogata. Cluster analysis of spatial point patterns: posterior distribution of parents inferred from offspring. *Japanese Journal of Statistics and Data Science*, 3:367–390, 2020. 219

[99] Fred Park. Shape descriptor / feature extraction techniques. 2011. UCI iCAMP 2011 [Link]. 85

[100] Yuval Peres and Allan Sly. Rigidity and tolerance for perturbed lattices. *Preprint*, pages 1–20, 2020. arXiv:1409.4490 [Link]. 213, 219

[101] Carl Rasmussen and Christopher Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. [Link]. 53

[102] Alfred R.Osborne. Multidimensional Fourier series. *International Geophysics*, 97:115–145, 2010. [Link]. 121

[103] Kamron Saniee. A simple expression for multivariate Lagrange interpolation. *SIAM Undergraduate Research Online*, 2007. SIURO [Link]. 104

[104] Mahesh Shivanand and all. Fitting random regression models with Legendre polynomial and B-spline to model the lactation curve for Indian dairy goat of semi-arid tropic. *Journal of Animal Breeding and Genetics*, pages 414–422, 2022. [Link]. 121

[105] Karl Sigman. Notes on the Poisson process. New York NY, 2009. IEOR 6711: Columbia University course [Link]. 219

[106] Joshua Snoke et al. General and specific utility measures for synthetic data. *Journal of the Royal Statistical Society Series A*, 181:663–688, 2018. arXiv:1604.06651 [Link]. 36

[107] Luuk Spreeuwers. *Image Filtering with Neural Networks: Applications and Performance Evaluation*. PhD thesis, University of Twente, 1992. 74

[108] J. Michael Steele. Le Cam's inequality and Poisson approximations. *The American Mathematical Monthly*, 101(1):48–54, 1994. [link]. 182

[109] Dietrich Stoyan, Wilfrid S. Kendall, Sung Nok Chiu, and Joseph Mecke. *Stochastic Geometry and Its Applications*. Wiley, 2013. 230

[110] E.C. Titchmarsh and D.R. Heath-Brown. *The Theory of the Riemann Zeta-Function*. Oxford Science Publications, second edition, 1987. 59, 152, 238

[111] Chris Tofallis. Fitting equations to data with the perfect correlation relationship. *Preprint*, pages 1–11, 2015. Hertfordshire Business School Working Paper[Link]. 14

[112] D. Umbach and K.N. Jones. A few methods for fitting circles to data. *IEEE Transactions on Instrumentation and Measurement*, 52(6):1881–1885, 2003. [Link]. 15, 18

[113] D. A. Vaccari and H. K. Wang. Multivariate polynomial regression for identification of chaotic time series. *Mathematical and Computer Modelling of Dynamical Systems*, 13(4):1–19, 2007. [Link]. 18

[114] Remco van der Hofstad. *Random Graphs and Complex Networks*. Cambridge University Press, 2016. [Link]. 222

[115] Yu Vizilter and Sergey Zheltov. Geometrical correlation and matching of 2D image shapes. 2012. ResearchGate [Link]. 87

[116] Fengyun Wang and all. Bivariate Fourier-series-based prediction of surface residual stress fields using stresses of partial points. *Mathematics and Mechanics of Solids*, 2018. [Link]. 121

[117] Luyao Wang and Hai Cheng. Pseudo-random number generator based on logistic chaotic system. *Entropy*, 21, 2019. [Link]. 166

[118] Mingguang Wu, Yanjie Sun, and Yaqian Li. Adaptive transfer of color from images to maps and visualizations. *Cartography and Geographic Information Science*, pages 289–312, 2021. [Link]. 189

[119] Lan Wua, Yongcheng Qi, and Jingping Yang. Asymptotics for dependent Bernoulli random variables. *Statistics and Probability Letters*, pages 455–463, 2012. [Link]. 167

[120] Lei Xu and Kalyan Veeramachaneni. Synthesizing tabular data using generative adversarial networks. *Preprint*, pages 1–12, 2018. arXiv:1811.11264 [Link]. 136

[121] Oren Yakir. Recovering the lattice from its random perturbations. *Preprint*, pages 1–18, 2020. arXiv:2002.01508 [Link]. 219

[122] Ruqiang Yan, Yongbin Liub, and Robert Gao. Permutation entropy: A nonlinear statistical measure for status characterization of rotary machines. *Mechanical Systems and Signal Processing*, 29:474–484, 2012. 233

[123] Shaohong Yan, Aimin Yang, et al. Explicit algorithm to the inverse of Vandermonde matrix. In *2009 International Conference on Test and Measurement*, 2009. IEEE [Link]. 48

[124] D. Yogeshwaran. Geometry and topology of the boolean model on a stationary point processes : A brief survey. *Preprint*, pages 1–13, 2018. Researchgate [Link]. 220

[125] Tonglin Zhang. A Kolmogorov-Smirnov type test for independence between marks and points of marked point processes. *Electronic Journal of Statistics*, 8(2):2557–2584, 2014. 215

[126] Changgang Zheng et al. Reward-reinforced generative adversarial networks for multi-agent systems. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6:479–488, 2021. arXiv:2103.12192 [Link]. 141

Index

α-compositing, 58
m-interlacing, 72, 220, 228, 231, 278

A/B testing, 276
activation function, 137
AdaBoost, 37, 277
Adam gradient descent, 138
additive number theory, 245, 252
adversarial learning, 129, 279
agent-based modeling, 188, 201
AI art, 130, 197
algebraic number, 154
algorithmic bias, 130
analytic continuation, 239, 246
analytic function, 152
anisotropy, 206, 227, 233
anti-aliasing, 56, 60, 256, 261
association rule, 269
attraction (point process), 218
attraction basin, 59
attractor distribution, 172, 222, 228, 235
augmented data, 36, 89, 188, 190, 279
auto-correlation, 50, 153
auto-regressive process, 50, 172, 277

Bailey–Borwein–Plouffe formulas, 154
Bayesian classification, 75
Bayesian inference, 45
    hierarchical models, 278
    naive Bayes, 269
Bernoulli trials, 263
Berry-Esseen inequality, 150
Bessel function, 234
Beurling primes, 153, 249
binning, 268
    optimum binning, 38, 277
binomial distribution, 228
bisection method (root finding), 179
boosted trees, 131
bootstrapping, 16, 97, 130, 229
    percentile method, 102
boundary effect, 215, 216, 221, 222, 226–228, 232
Brownian motion, 52, 167, 171, 188, 189, 204, 273, 277
    Lévy flight, 172
Brun's theorem, 245

Cauchy distribution, 172
Cauchy-Riemann equations, 152
causality, 278
Cayley-Hamilton theorem, 48
CDF regression, 18
censored data, 221
central limit theorem, 172, 213, 263
chaotic dynamical system, 188, 204
character
    principal, 241
characteristic function, 234
characteristic polynomial, 48, 50–52, 173
ChatGPT, 270
Chebyshev's bias (prime numbers), 152, 241
checksum, 276
Chi-squared test, 215
Chowla conjecture, 243
classification, 279
clique (graph theory), 223
cluster process, 217, 221, 228
clustering, 279
Collatz conjecture, 160
collision graph, 204
color model
    RGB, 56, 262
    RGBA, 56, 57, 77, 262
color opacity, 198
color transparency, 21, 190, 262
complex random variable, 149, 249
computational complexity, 162, 270
computer vision, 13, 84
confidence band, 229
confidence interval, 45, 226, 277
confidence level, 264, 266
confidence region, 16, 30, 130, 226, 227, 263
    dual region, 45, 264, 277
conformal map, 15
confusion matrix, 268, 279
connected components, 205, 221–223, 227, 228, 271
contour level, 197, 266
contour plot, 266
convergence
    abscissa, 241, 246
    absolute, 113, 238, 239
    alternating series, 239
    conditional, 113, 151, 241
    Dirichlet test, 239
convergence acceleration, 60
convex linear combination, 109
convolution of distributions, 234
copula, 100, 130, 143
    Frank, 130, 136
    Gaussian, 130
correlation matrix distance, 136, 141, 143
cosine distance, 271

counting measure, 214
covariance matrix, 91, 263
covering (stochastic), 230
covering problem, 229
credible interval, 45
credible region (Bayesian), 264, 277
critical line (number theory), 114
cross-validation, 16, 138, 204, 268
cuban primes, 245
curse of dimensionality, 116
curve fitting, 27

data video, 188, 197
decision tree, 37
decorrelate, 143
decorrelation, 140, 143
Dedekind zeta function, 152, 249
deep neural network, 74, 137, 278
dense set (topology), 243
density estimation, 220
diamond-square algorithm, 189
Diehard tests of randomness, 151
dimensionality reduction, 17
Dirichlet character, 152, 153, 240, 245
    modulo 4, 241, 246, 247
Dirichlet eta function, 253
Dirichlet functional equation, 152, 246, 247
Dirichlet series, 149
Dirichlet's theorem, 152, 241, 243, 247
Dirichlet-L function, 152, 240, 247
disaggregation, 115
discrete Fourier series, 120
discrete orthogonal functions, 120
dissimilarity metric, 271
distributed architecture, 267
distribution
    Cauchy, 172
    Fréchet, 52, 172
    Gaussian, 263
    generalized logistic, 91, 217
    Hotelling, 264
    Laplace, 234
    logistic, 17
    Lévy, 172
    modified Bessel, 234
    Poisson-binomial, 182, 236
    Poisson-exponential, 213
    Rademacher, 149, 150
    Rayleigh, 228, 229, 235
    Weibull, 52, 172, 228
domain of attraction, 222
dot product, 15
dummy variable, 37
dummy variables, 137
dyadic map, 153
dynamical systems, 153, 222
    chaotic systems, 188, 204
    dyadic map, 153
    ergodicity, 153
    logistic map, 153
    shift map, 153
    stochastic, 189

edge effect (statistics), 221
eigenvalue, 14, 53, 91, 279
    power iteration, 93
elbow rule, 176, 221, 228, 273
elliptic curve, 245
EM algorithm, 36, 134, 278
empirical distribution, 17, 97, 121, 130, 213, 216, 222, 228, 233, 235, 247
    multivariate, 150
empirical quantiles, 102, 143
ensemble methods, 37, 84, 131
entropy, 207, 233, 269
epoch, 137, 140
equidistribution modulo 1, 155
equilibrium distribution, 189
Erdős–Rényi model, 223
ergodicity, 153, 189, 226, 228, 234
Euler product, 149, 239, 246
    random, 249
Euler's transform, 253
evolutionary process, 189
experimental design, 276
experimental math, 57, 237
explainable AI, 14, 36, 75, 84, 91, 129, 143, 176, 192, 230
exploratory analysis, 275
exponential decay, 41
exponential sums, 246
extrapolation, 109
extreme value theory, 172, 235

feature attribution, 129
feature clustering, 134, 140, 143
feature importance, 129
feature selection, 16, 98, 267
Fermat's last theorem, 246
fixed-point algorithm, 60, 90, 176, 278
flag vector, 269, 276
Fourier series, 120
Fourier transform, 234
fractal dimension, 52
fractional part function, 155
Frobenius norm, 91
Fruchterman and Rheingold algorithm, 272
Fréchet distribution, 52, 172
fuzzy classification, 57

Gamma function, 52, 172
GAN (generative adversarial networks), 36, 129, 134, 143, 192, 213, 278
Gaussian circle problem, 252
Gaussian distribution, 263
Gaussian mixture model
    see GMM, 36
Gaussian primes, 152, 248
Gaussian process, 50, 277
general linear model, 14
generalized linear model, 14, 49
generalized logistic distribution, 91, 227

generative adversarial networks
see GAN, 36
generative AI, 188, 201
generative model, 36, 53, 100, 187, 188, 190, 197, 204, 279
geostatistics, 103
GIS, 117
Glivenko-Cantelli theorem, 247
GMM (Gaussian mixture model), 70, 71, 133, 134, 143, 213, 278
Goldbach's conjecture, 245
goodness-of-fit, 57, 268
GPU-based clustering, 72
gradient (optimization), 176
gradient boosting, 277
gradient operator, 16
Gram-Schmidt orthogonalization, 120
graph, 221
collision graph, 204
connected components, 205, 227, 271
directed, 205
edge, 221
Fruchterman-Reingold, 205
nearest neighbor graph, 223, 227
node, 221, 223
random graph, 222
random nearest neighbor graph, 222
tree, 205
undirected, 221–223, 228
vertex, 221
graph database, 272
graph theory, 221
GraphViz, 205
greedy algorithm, 113, 252
grid search, 176, 191

half-tone (music), 259
Hartman–Wintner theorem, 167
hash table, 163, 207, 232, 269, 270
sparse, 270
Hausdorff distance, 88
Hellinger distance, 130, 143
Hermite polynomials, 120
hexagonal lattice, 221
hidden decision trees, 37, 38, 277
hidden layer, 74
hidden process, 214, 231, 235
hierarchical clustering, 74, 270
Hilbert primes, 248
histogram equalization, 72, 74
Hoeffding inequality, 170
homogeneity (point process), 182, 220
Hotelling distribution, 264
Hurst exponent, 52
hyperparameter, 30, 57, 104, 191
hyperparameters, 136

identifiability, 231, 233
ill-conditioned problem, 27, 53, 93, 279
image segmentation, 74
imputation (missing values), 130
index
index discrepancy, 233
intensity (stochastic process), 213, 220, 227
interarrival times, 171, 213, 222, 226, 233
standardized, 234
interlaced processes, 220
Internet of Things, 213
inverse distance weighting, 105
inverse square law, 201
iterated logarithm, 150, 151, 167
Itô integral, 53

K-means clustering, 32, 33
key-value pair, 38, 269
Kolmogorov-Smirnov test, 130, 150, 215, 222
kriging, 113
Kronecker's theorem, 243, 251

Lagrange interpolation, 53
Lagrange multiplier, 16, 278
Laplace distribution, 234
large language models, 270, 279
Lasso regression, 16, 279
latent variables, 137
lattice, 219
perturbed lattice, 213
shifted, 221
stretched, 221
law of the iterated logarithm, 150, 151, 167, 243, 249
Le Cam's theorem, 182, 214
learning rate, 136, 141, 143
least absolute residuals, 102
LightGBM, 136, 143
link function, 14, 17
Liouville function, 240, 251
LLM (large language model), 270, 279
log-polar map, 15
logistic distribution, 17, 220
logistic map, 153
logistic regression, 17
unsupervised, 34
logit function, 278
loss function, 137, 140, 143
Lévy distribution, 172
Lévy flight, 172

Map-reduce, 267
marketing attribution, 276
Markov chain, 50
MCMC, 149
Mathematica, 266
MaxCliqueDyn algorithm, 223
maximum likelihood estimation, 265, 278, 279
mean squared error, 16, 31
medoid, 32
Mersenne twister, 30, 153, 156, 169
Mertens function, 240
minimum contrast estimation, 191, 230, 233, 265
mixture model, 30, 46, 189, 197, 220, 221, 228, 266, 279
blending, 189

model fitting, 57, 278
model identifiability, 16
modulus (complex number), 173, 239
Monte Carlo simulations, 149, 278
morphing (computer vision), 188
moving average, 178
multidimensional Fourier series, 121
multiple root, 114
multiplicative function
  completely multiplicative, 149, 151, 240, 241, 252
  Rademacher, 149
Möbius function, 240

N-body problem, 201
n-gram (NLP), 270
naive Bayes, 269, 277
natural language generation, 270, 279
natural language processing, 37, 270
nearest neighbor interpolation, 102, 105
nearest neighbors, 213, 223, 229, 278
  nearest neighbor distances, 227–229, 231, 235
  nearest neighbor graph, 227
NetworkX, 205
neural network, 74
  activation function, 137
  epoch, 137
  hidden layer, 74
  hyperparameter, 76
  neuron, 74, 137, 278
  seq2seq, 187
  sparse, 70
  very deep, 74
Newton's method, 176
NLG (natural language generation), 270, 279
node (decision tree), 38, 131, 277
  perfect node, 45
  usable node, 39
node (interpolation), 114
normal number, 150, 243, 247
  strongly normal, 151
numerical stability, 48

Omega function, 240, 246
order statistics, 235
ordinary least squares, 51, 102, 120
orthogonal function, 120
Otsuka–Ochiai coefficient, 271
outliers, 235, 273
overfitting, 16, 130, 136, 233, 277

palette, 188, 262
parametric bootstrap, 21, 30, 36, 98, 130, 229, 277, 278
partial derivative, 114
partial least squares, 14
path (graph theory), 221
percentile bootstrap, 102
permutation
  entropy, 233
  random permutation, 232
perturbed lattices, 213
Plotly, 197
point count distribution, 214, 227, 230
point process
  attractive, 228
  cluster process
    Matérn, 219
    Neyman-Scott, 219
  non-homogeneous, 182, 220
  perturbed lattice process, 219
  radial, 220
  renewal process, 219
  repulsive, 218
Poisson point process, 171, 182, 213, 227
Poisson-binomial distribution, 182, 213, 236
Poisson-exponential distribution, 213
positive semidefinite (matrix), 49, 92
power iteration, 93
preconditioning, 93
prediction interval, 16, 97, 102
predictive power, 38, 45, 129, 268, 269
prime test (of randomness), 151, 162, 168
principal component analysis, 49, 129, 277
probability generating function, 168
proxy space, 266
pseudo-inverse matrix, 49
pseudo-random numbers, 169, 273
  combined generators, 166
  congruential generator, 156
  Diehard tests, 151, 163
  Mersenne twister, 156, 169, 190
  prime test, 151, 168
  strongly random, 151, 154
  TestU01, 151
Pólya conjecture, 242

quadratic irrational, 153, 156, 162
quantile, 264, 278
  empirical, 102, 130
  weighted, 102
quantile function, 100, 121, 213, 217, 228
quantile regression, 16

R-squared, 16, 36, 191
Rademacher distribution, 150
Rademacher function, 149, 243, 249
  random, 151
random function, 181
random graph, 222, 223
random multiplicative function, 149
  Rademacher, 151
random permutation, 232
random variable
  complex, 149
random walk, 167, 191, 277
  first hitting time, 168, 171
  zero crossing, 167
Rayleigh distribution, 228, 229, 235
Rayleigh test, 228
records, 235
regression splines, 14
regular expression, 270, 276
reinforcement learning, 141, 143, 279

rejection sampling, 131
ReLU function, 137
renewal process, 219
repulsion (point process), 218, 229
repulsion basin, 239
resampling, 97, 229
Riemann Hypothesis, 108, 114
    Generalized, 151, 241, 245, 247
Riemann zeta function, 114, 149, 152, 247, 249
root mean squared error, 57

scaling factor, 227, 235
SDV (Python library), 135
seed (random number generator), 131, 136, 140, 163, 190
semi-supervised learning, 279
shape signature, 85
Shapley value, 129
Shepard's method, 105
shift map, 153
sigmoid function, 137, 278
simplex, 250
singular value decomposition, 14, 279
singularity, 207
six degrees of separation, 272
Sklar's theorem, 130
smoothing parameter, 104
spatial statistics, 103, 219
spectral domain, 189
spline regression, 121
square root (matrix), 49, 92, 140
square-free integer, 150, 163, 243
stable distribution, 172, 190, 234
state space, 189
stationary distribution, 53
stationary process, 50, 172, 189, 204, 215, 220, 227
stepwise regression, 99
stochastic convergence, 189
stochastic function, 52
stochastic geometry, 230
stochastic gradient descent, 137
stochastic process, 213
stochastic residues, 231
stop word (NLP), 270
stretching (point process), 221
Sturm-Liouville theory, 120
superimposition (point processes), 220
supervised classification, 72
surface plot, 197
SVD (Python library), 143
swarm optimization, 28, 278
synthetic data, 14, 28, 30, 53, 89, 91, 113, 119, 130, 149, 162, 168, 176, 190, 197, 204, 237, 264
synthetic metric, 269

TabGAN (Python library), 136
Tarjan's algorithm, 271
tensor, 75
TensorFlow, 136
text normalization, 270
Theil-Sen estimator, 102
time series, 51
    auto-regressive, 52, 172
    disaggregation, 108
    Hurst exponent, 52
    non-periodic, 26
total least squares, 14
training set, 102, 204, 268
transcendental number, 154
transformer, 74, 187
tree (graph theory), 205
twin primes, 245

universality property, 240, 243, 245
unsupervised clustering, 72
unsupervised learning, 34, 279

validation set, 16, 57, 102, 130, 204, 268
Vandermonde matrix, 48, 53
vertex, 213, 221, 222, 235
video compression
    FFmpeg, 56, 60

Waring's problem, 245
Watts and Strogatz model, 272
Weibull distribution, 52, 172, 228, 235
weighted least squares, 14
weighted quantiles, 102
weighted regression, 17
white noise, 28, 50, 172, 277
wide data, 121, 279

XOR operator, 156