

Histogram methods and the curse of dimensionality

Prof. Dr. Emre Onur Kahya · Department of Physics Engineering, Istanbul Technical University

The simplest learning algorithm cuts the input space into cells and counts. It solves all three tasks of Note 1 — classification, regression and density estimation — works well in one or two dimensions, and fails completely in ten. Understanding why it fails is the best introduction to the central difficulty of machine learning in high dimension, and to the ideas — adaptive partitions, low intrinsic dimension, learned representations — that modern methods use to escape it. The note also walks through the steps of designing and analysing an algorithm.

1 Cut the space into cells and count

Every learning algorithm must produce a prediction at *every* point of the input space, including where no training example lies. The most direct way to do this is to partition the space into cells $C_1, \dots, C_M$ and make one prediction per cell. With m equal bins along each of d axes the cells are boxes of side h and there are $M = m^d$ of them. Write $C(\mathbf{x})$ for the cell containing $\mathbf{x}$ and n_C for the number of training points in cell C . One rule then serves each task of Note 1, Table 2:

Task	Prediction at $\mathbf{x}$	Interpretation
Classification	majority label among the points in $C(\mathbf{x})$	estimate of $P(Y = k \mid \mathbf{X} \in C(\mathbf{x}))$ by class frequencies
Regression	average of y_i over the points in $C(\mathbf{x})$	estimate of $\mathbb{E}[Y \mid \mathbf{X} \in C(\mathbf{x})]$
Density	$\hat{p}(\mathbf{x}) = \frac{n_{C(\mathbf{x})}}{n \text{vol}(C(\mathbf{x}))}$	fraction of the data per unit volume

The density estimate is a valid density: it is non-negative and $\int \hat{p} = \sum_C n_C / n = 1$. All three rules replace the unknown conditional quantity at $\mathbf{x}$ by its average over a neighbourhood — here, the cell. The neighbourhood is fixed in advance, which is both the simplicity and the weakness of the method; Note 5 centres the neighbourhood on the query point instead.

1.1 Density estimation in one dimension

Figure 1 shows the effect of the number of bins on a density estimate from 200 points. Too few bins and the estimate is a coarse staircase that misses the structure (*bias*); too many and most bins contain zero or one point, so the estimate is dominated by the luck of the sample (*variance*). The counts are random: n_C is binomial, approximately Poisson with mean $n P(\mathbf{X} \in C)$, so its relative fluctuation is about $1/\sqrt{n_C}$ — the $\sqrt{N}$ error bar physicists draw on every bin.

Balancing the two errors gives the classical result that the best bin width shrinks like $h \propto n^{-1/3}$; the Freedman–Diaconis rule $h = 2 \text{IQR} \cdot n^{-1/3}$, where IQR is the interquartile range, is a robust default. The mean squared error of the optimised histogram then decreases like $n^{-2/3}$. In d dimensions the

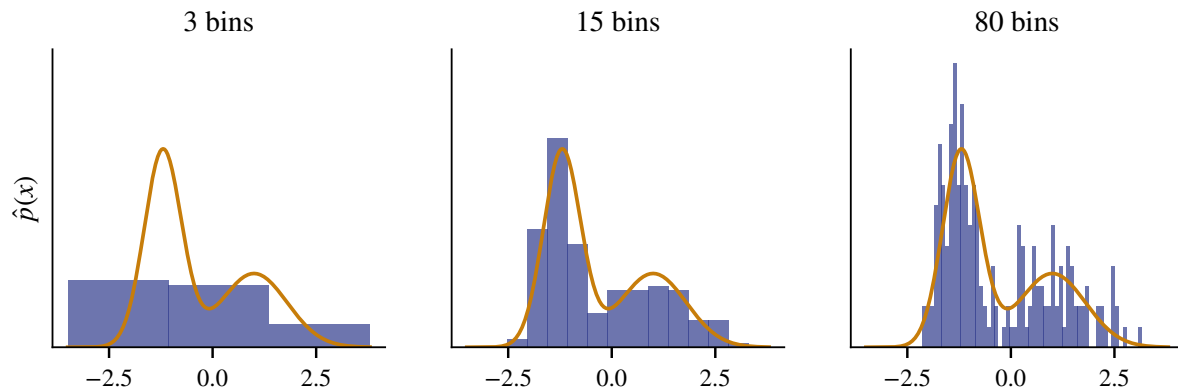


Figure 1. Histogram density estimates from the same 200 points with 3, 15 and 80 bins; the curve is the true density.

same analysis gives $n^{-2/(d+2)}$ [7]: to halve the error one needs about $2^{3/2} \approx 2.8$ times more data in $d = 1$ but $2^6 = 64$ times more in $d = 10$. This is the curse of dimensionality in its most quantitative form.

In physics — histograms are the language of experiments

Most results in particle and nuclear physics are histograms: event counts in bins of an invariant mass, a momentum or an angle. The 2012 Higgs-boson discovery showed a small excess in the histogram of diphoton invariant masses near 125 GeV [1]. A *binned likelihood fit* treats each count as Poisson with a mean predicted by a signal-plus-background model — the density-estimation rule of the table above, turned into a statistical test. The physicist’s choice of binning is exactly the bias–variance compromise of Figure 1.

2 More features help — up to a point

Take the two-class problem of Note 1 (signal and background events described by two observables) with $n = 400$ training events and classify with histograms (Figure 2). Using only x_1 with 8 bins, the classifier reaches 20.2% test error. Adding x_2 with an 8×8 grid lowers it to 14.4%: the second feature carries information the first lacks. Refining to 40×40 cells lowers the *training* error to 5.2% but raises the test error to 34.6%. With 1600 cells and 400 points, most cells hold zero or one example; the predictions in those cells are noise or a default. The same table of counts has gone from summarising the data to memorising it.

Two lessons follow. First, the number of cells is a knob that controls how flexible the classifier is; it must be chosen on data not used for counting, which is the subject of Note 4. Second, the knob runs out of room quickly: every additional feature multiplies the number of cells by m , while the amount of data stays the same.

3 The curse of dimensionality

Cells outnumber data. With $m = 10$ bins per axis there are 10^d cells. A dataset of $n = 10^5$ points spread uniformly fills every cell with 10^4 points when $d = 1$, about ten per cell when $d = 4$, and when $d = 6$ only about 10% of the cells contain any point at all. For $d = 10$, a fraction 10^{-5} of the cells is occupied: a new test point almost surely lands in a cell the training data never visited (Figure 3, left). A 28×28 grey-scale image has $d = 784$, and 10^{784} dwarfs the roughly 10^{80} atoms in the observable

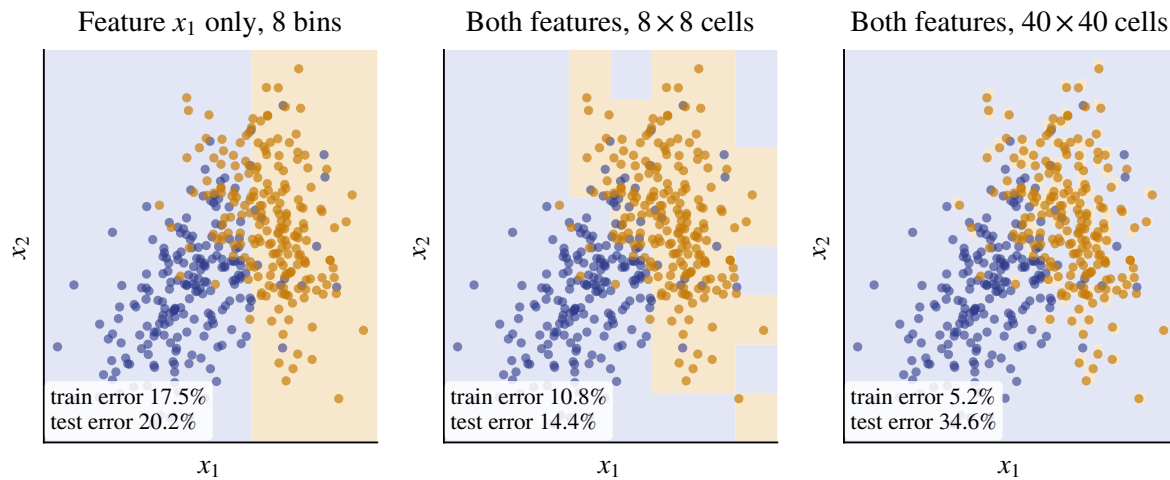


Figure 2. Histogram classifiers on 400 training events (indigo: background, amber: signal). Shaded regions show the predicted class; empty cells predict the overall majority class. Errors are measured on the training set and on 4000 fresh test events.

universe.

Neighbourhoods stop being local. To capture a fraction r of uniformly distributed data in the unit cube with a sub-cube, its edge must be $r^{1/d}$. Capturing 1% of the data needs an edge of 0.01 in $d = 1$, 0.1 in $d = 2$, 0.63 in $d = 10$ and 0.95 in $d = 100$. A “neighbourhood” that contains enough points to average over spans almost the whole range of every coordinate.

Volume lives near the boundary. The fraction of a d -dimensional ball’s volume that lies within 10% of its surface is $1 - 0.9^d$: 19% for $d = 2$, 65% for $d = 10$ and more than 99% for $d = 50$. The ball inscribed in a cube occupies 79% of the cube in $d = 2$ but only 0.25% in $d = 10$; almost all of the cube is in its corners. A standard Gaussian in d dimensions concentrates near the sphere of radius $\sqrt{d}$ (Note 2, Exercise E5), so its typical samples are nowhere near its most probable point.

Distances concentrate. For points drawn at random, the ratio between the largest and the smallest distance from a query point tends to one as d grows (Figure 3, right): the nearest neighbour is hardly nearer than the farthest. Any method that relies on “nearby” points — histograms, and also the neighbourhood methods of Note 5 — loses its footing.

The curse A method that needs examples *near* every query point needs a number of examples that grows exponentially with the dimension. The number of distinct functions such a method can represent grows just as fast, so without further assumptions the data cannot tell them apart. Histograms are useful in $d \leq 3$ and useless beyond; all learning algorithms feel the curse, some far less than others.

4 Why learning in high dimension is possible at all

Image classifiers work on inputs with $d \approx 10^5$; language models on sequences of thousands of tokens. They do not contradict the curse; they exploit structure that uniform random data do not have.

1. **Low intrinsic dimension.** Real data occupy a tiny, structured part of $\mathbb{R}^d$. Natural images of

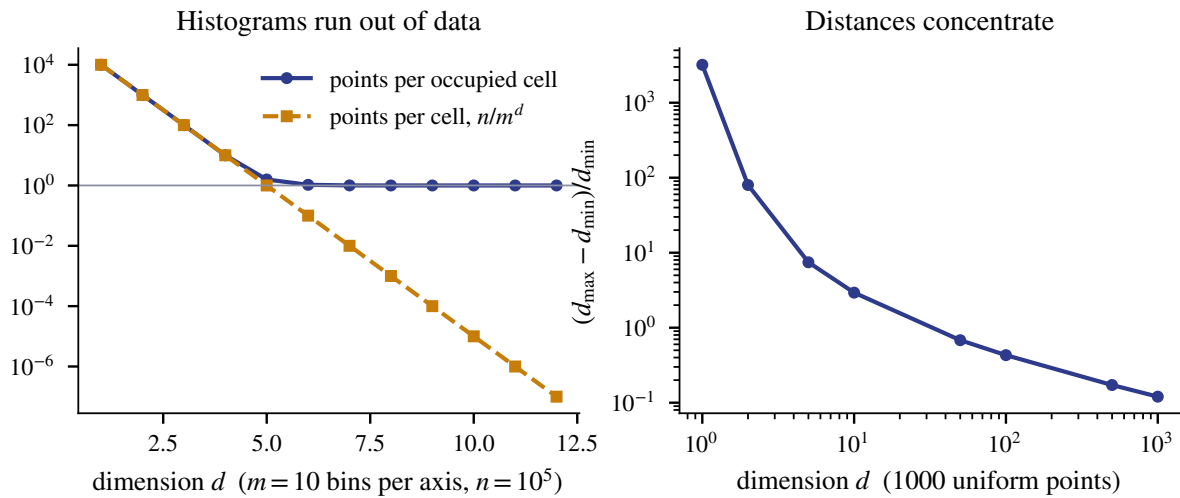


Figure 3. Left: average number of points per cell and per *occupied* cell for $n = 10^5$ uniform points and 10 bins per axis. Beyond $d \approx 5$ occupied cells hold a single point and almost all cells are empty. Right: relative contrast between the farthest and the nearest of 1000 uniform points from a random query (median of 20 trials).

a given size are a vanishing subset of all pixel arrays, plausibly close to a manifold of much lower dimension. Estimates of the intrinsic dimension of standard image datasets give values of order 10 to 40, compared with input dimensions in the thousands, and learning is easier when the intrinsic dimension is lower [5].

2. **Smoothness and invariance.** The target changes slowly or not at all along many directions: a translated cat is still a cat. Architectures that build these invariances in (convolutions for translations, graph networks for permutations, equivariant networks for rotations) reduce the effective number of functions to be distinguished.
3. **Adaptive partitions.** Instead of a fixed grid, split only where the data require it and only along informative directions. A decision tree does exactly this; its number of cells grows with n , not with m^d .
4. **Learned representations.** Map the input to a lower-dimensional space in which simple methods work, and learn the map from data. The embeddings of Note 1, Section 3, are such maps.

In physics — many degrees of freedom, few that matter

A configuration of the two-dimensional Ising model on a 40×40 lattice is a point in a space of 2^{1600} states, yet near the phase transition its physics is governed by one order parameter. Applying principal component analysis to Monte Carlo configurations recovers the magnetisation as the leading direction without being told about it, and locates the critical temperature [8]. Statistical mechanics is, in this sense, a theory of low intrinsic dimension; machine learning tries to discover such coordinates when no theory supplies them.

Modern connection — the histogram's descendants

Decision trees are adaptive histograms, and ensembles of them — random forests and gradient-boosted trees — remain the strongest general-purpose methods on tabular data of moderate size, often ahead of deep networks [2]. Boosted decision trees were the standard multivariate classifier in particle physics for over a decade [6]. Modern implementations even keep the literal histogram: LightGBM bins each feature into at most a few hundred values and finds

splits by scanning those per-feature histograms, which is what makes training on millions of rows fast [4]. Binning one feature at a time is cheap; binning all features jointly is what the curse forbids.

5 From idea to algorithm

Designing a learning algorithm follows a sequence that is worth making explicit once:

1. explain the idea in words (“count labels per cell, predict the majority”);
2. formalise it mathematically (the table in Section 1);
3. write pseudocode (Algorithm 1), identifying its *hyperparameters* — quantities fixed before training, here the bins — and its *parameters* — quantities computed from data, here the counts;
4. implement and test on low-dimensional synthetic data, where decision regions can be plotted and errors are visible (as in Figure 2);
5. analyse cost in time and memory;
6. evaluate on real data against other algorithms, with the methodology of Note 4.

Algorithm 1 Histogram classifier with m bins per axis on the box $[a_1, b_1] \times \dots \times [a_d, b_d]$

```

1: function CELLINDEX( $\mathbf{x}$ )
2:   for  $j = 1, \dots, d$  do
3:      $c_j \leftarrow \min(m - 1, \max(0, \lfloor m(x_j - a_j)/(b_j - a_j) \rfloor))$  ▷ clip to the box
4:   end for
5:   return  $(c_1, \dots, c_d)$ 
6: end function
7: function TRAIN( $D_{\text{train}}$ )
8:    $N[c, k] \leftarrow 0$  for every cell  $c$  and class  $k$ 
9:   for each  $(\mathbf{x}_i, y_i) \in D_{\text{train}}$  do
10:     $N[\text{CELLINDEX}(\mathbf{x}_i), y_i] \leftarrow N[\text{CELLINDEX}(\mathbf{x}_i), y_i] + 1$ 
11:   end for
12: end function
13: function PREDICT( $\mathbf{x}$ )
14:    $c \leftarrow \text{CELLINDEX}(\mathbf{x})$ 
15:   if  $\sum_k N[c, k] = 0$  then return the majority class of  $D_{\text{train}}$ 
16:   end if
17:   return  $\arg \max_k N[c, k]$  ▷  $N[c, \cdot] / \sum_k N[c, k]$  estimates  $P(Y | \mathbf{X} \in c)$ 
18: end function

```

Cost. With n training points, d features, m bins per axis and K classes:

	Dense array of counts	Hash table of occupied cells
Memory	$O(Km^d)$	$O(K \min(n, m^d))$
Training time	$O(nd)$	$O(nd)$
Prediction time per query	$O(d + K)$	$O(d + K)$ on average

Training and prediction are as fast as any algorithm can be: one pass over the data, constant work per query. Storing only occupied cells in a hash table (a Python dictionary keyed by the cell index) removes the exponential memory. It does not remove the curse: a query in high dimension still lands, almost surely, in a cell with no data. The bottleneck is statistical, not computational.

Summary Partitioning the input space and counting gives a classifier, a regressor and a density estimator in a few lines, with the bin width trading bias against variance. The number of cells grows as m^d , so in high dimension cells are empty, neighbourhoods are not local, volume sits near the boundary and distances concentrate. Learning in high dimension succeeds by exploiting structure — low intrinsic dimension, smoothness, invariances — through adaptive partitions and learned representations.

Exercises

- E1. Show that the histogram density estimate integrates to one. For a smooth density p in one dimension and a bin C of width h , show that $\mathbb{E}[\hat{p}(x)] = \frac{1}{h} \int_C p$ and that $\text{Var}[\hat{p}(x)] \approx p(x)/(nh)$ for small h . Explain how these two facts produce the bias–variance trade-off of Figure 1.
- E2. A test point is uniform in $[0, 1]^d$ and the n training points are uniform and independent. With m bins per axis, what is the probability that the test point’s cell contains no training point? Evaluate it for $n = 10^6$, $m = 4$ and $d = 5, 10, 20$.
- E3. Using $V_d(r) = \pi^{d/2} r^d / \Gamma(d/2 + 1)$, compute the ratio of the volume of the unit ball to that of the cube $[-1, 1]^d$ for $d = 1, \dots, 20$ and plot it. Where is the volume of a high-dimensional cube?
- E4. Implement Algorithm 1 with a dictionary of occupied cells and reproduce the three panels of Figure 2 with data of your own. Add the regression variant and test it on $y = \sin(2\pi x_1) + \text{noise}$.
- E5. Draw 1000 points uniformly in $[0, 1]^d$ and one query point. Plot the histogram of the 1000 distances for $d = 2, 20$ and 200 . Rescale each histogram by its mean and comment.
- E6. The Freedman–Diaconis rule gives the bin width in one dimension. Propose a way to choose the number of bins using only the data, without a formula, and say why the training log-likelihood cannot be used for this.

Further reading. Hastie, Tibshirani and Friedman [3], section 2.5, for the curse of dimensionality; Scott [7] for histograms and their optimal bin widths.

References

- [1] ATLAS Collaboration, Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC, *Phys. Lett. B* 716, 1–29 (2012).
- [2] L. Grinsztajn, E. Oyallon and G. Varoquaux, Why do tree-based models still outperform deep learning on typical tabular data?, *NeurIPS Datasets and Benchmarks* (2022).
- [3] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, 2nd ed., Springer (2009).
- [4] G. Ke *et al.*, LightGBM: a highly efficient gradient boosting decision tree, *NeurIPS* (2017).
- [5] P. Pope *et al.*, The intrinsic dimension of images and its impact on learning, *ICLR* (2021).

- [6] B. P. Roe *et al.*, Boosted decision trees as an alternative to artificial neural networks for particle identification, *Nucl. Instrum. Meth. A* 543, 577–584 (2005).
- [7] D. W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*, 2nd ed., Wiley (2015).
- [8] L. Wang, Discovering phase transitions with unsupervised learning, *Phys. Rev. B* 94, 195105 (2016).

The selection and order of topics in weeks 1–3 follow the course IFT 6390 *Fondements de l'apprentissage machine* at Université de Montréal (taught by Ioannis Mitliagkas, slides by Pascal Vincent). The text, examples, figures and exercises of this note are original. Figures are computed from synthetic data by `figures.py`, published with the sources of these notes.