

Neighbourhood methods: nearest neighbours, Parzen windows and distances

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

Neighbourhood methods predict at a point by letting nearby training examples vote. They need no training, make no assumption about the shape of the target function, and handle classification, regression and density estimation with one formula. This note develops k -nearest neighbours and Parzen windows (kernel smoothing and kernel density estimation), discusses the distances they rely on, and shows that the same ideas power retrieval in today's AI systems and the attention mechanism of transformers. Model selection follows Note 4; the geometry of high dimension follows Note 3.

1 Let the neighbours vote

Fix a distance $d(\cdot, \cdot)$ on the input space, Euclidean unless stated otherwise. For a query $\mathbf{x}$, let $N_k(\mathbf{x})$ be the set of indices of the k training points closest to $\mathbf{x}$. The k -nearest-neighbour (k -NN) prediction averages their targets. All three tasks fit one formula,

$$\bar{y}(\mathbf{x}) = \frac{\sum_{i=1}^n w_i(\mathbf{x}) t_i}{\sum_{i=1}^n w_i(\mathbf{x})}, \quad w_i(\mathbf{x}) = \mathbf{1}\{i \in N_k(\mathbf{x})\}, \quad (1)$$

with a task-dependent target encoding t_i and read-out:

- **regression:** $t_i = y_i$ and $f(\mathbf{x}) = \bar{y}(\mathbf{x})$, the mean target of the neighbours;
- **classification with m classes:** $t_i = \text{onehot}(y_i) \in \{0, 1\}^m$, so $\bar{y}(\mathbf{x})$ is the vector of class frequencies among the neighbours — an estimate of $P(Y = \cdot \mid \mathbf{X} = \mathbf{x})$ — and $f(\mathbf{x}) = \arg \max_c \bar{y}_c(\mathbf{x})$, the majority vote;
- **binary classification** with $y_i \in \{-1, +1\}$: $f(\mathbf{x}) = \text{sign } \bar{y}(\mathbf{x})$.

Like the histograms of Note 3, k -NN replaces $\mathbb{E}[Y \mid \mathbf{X} = \mathbf{x}]$ by an average over a neighbourhood, but the neighbourhood is now centred on the query and adapts its size to the local density of data: small where examples are plentiful, large where they are sparse. No cell is ever empty.

Neighbourhood methods are called *non-parametric*: they assume nothing about the form of f , and their “parameters” are the training set itself, which must be stored. Their hyperparameters — k and the distance — are chosen on validation data as in Note 4.

Decision regions. For $k = 1$, each training point owns the region of space closer to it than to any other point, its *Voronoi cell*. The classifier predicts the label of the cell's owner, so the decision boundary is made of pieces of perpendicular bisectors between differently labelled neighbours (Figure 1). It can be arbitrarily complex: one mislabelled point carves out its own island.

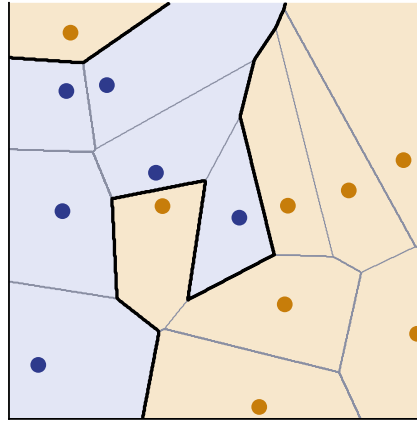


Figure 1. The 1-NN classifier for 14 points. Thin lines: Voronoi cells; thick line: decision boundary; shading: predicted class.

The choice of k . Figure 2 shows k -NN on 300 points from two interleaved half-moons. With $k = 1$ the training error is zero and the boundary follows every noisy point; the test error is 9.2%. With $k = 15$ the boundary is smooth and training and test errors agree at about 6%. With $k = 151$, half the training set votes everywhere, the boundary is too simple to follow the moons and both errors rise to about 19%. Small k means low bias and high variance; large k the reverse (Note 4, Section 6). In the limit $k = n$ every query receives the overall majority class.

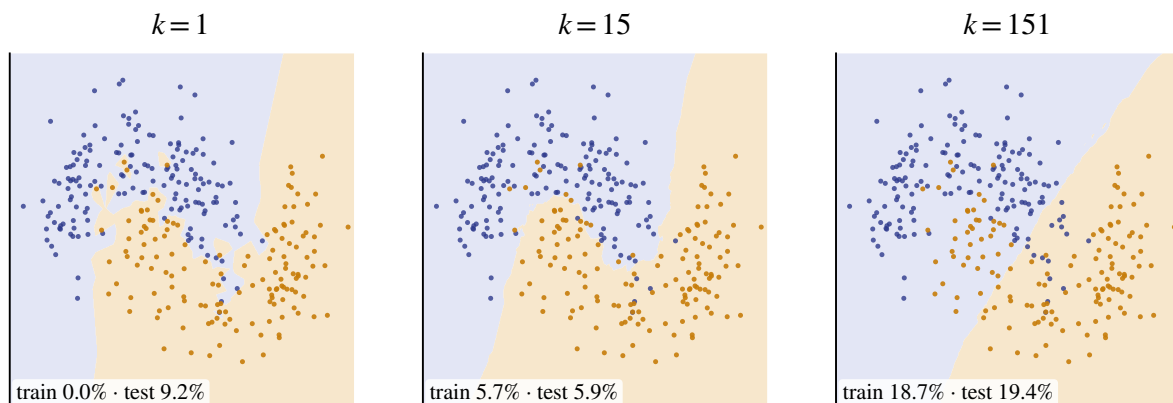


Figure 2. k -NN classification of 300 training points; errors on the training set and on 3000 test points.

What theory guarantees. With enough data, nearest neighbours are close to the query and the method approaches the ideal. As $n \rightarrow \infty$, the error of 1-NN is at most twice the Bayes error (for two classes, $R^* \leq R_{1\text{-NN}} \leq 2R^*(1 - R^*)$) [2], and k -NN is *consistent* — its risk converges to the Bayes risk for every distribution — if $k \rightarrow \infty$ while $k/n \rightarrow 0$ [13]. The catch is the rate: in d dimensions the distance to the nearest neighbour shrinks only like $n^{-1/d}$, and Note 3 showed what that means for $d = 100$.

2 Computing neighbours

The algorithm is short. Training stores $X \in \mathbb{R}^{n \times d}$ and the targets. Prediction for a batch of queries computes all squared distances with the matrix identity of Note 1, Section 6, selects the k smallest per row and averages:

```
def knn_classify(Xtr, ytr, Xq, k, m):          # ytr: integer labels 0..m-1
    D = sq_distances(Xq, Xtr)                # (q, n), Note 1
    idx = np.argpartition(D, k - 1, axis=1)[: , :k]  # k nearest, unordered
    votes = np.eye(m)[ytr[idx]].sum(axis=1)      # (q, m) class counts
    return votes.argmax(axis=1)
```

Partial selection (`argpartition`) finds the k smallest of n numbers in $O(n)$ time; a full sort would cost $O(n \log n)$.

	Brute force	Approximate index (e.g. graph-based)
Memory	$O(nd)$	$O(nd)$ plus the index
Training	none (store the data)	build the index, roughly $O(n \log n)$ distance evaluations
Prediction per query	$O(nd)$	roughly $O(d \log n)$, returns <i>most</i> true neighbours

Training is free and prediction is expensive — the opposite of most models. Exact tree structures such as k -d trees prune the search in low dimension but degrade to brute force as d grows, another face of the curse. Large systems therefore accept *approximate* neighbours: graph-based indices such as HNSW [9] and compressed vector codes on GPUs [5] make collections of billions of vectors searchable, at the price of occasionally missing a true neighbour.

Modern connection — nearest neighbours inside AI systems

Nearest-neighbour search in embedding space (Note 1, Section 3) is everywhere. *Retrieval-augmented generation* embeds a question, retrieves the nearest passages from a document collection and gives them to a language model as context [8]; the “vector databases” behind such systems are approximate k -NN indices. A language model can also be improved without re-training by interpolating its next-token distribution with a k -NN estimate computed from a datastore of contexts [6] — exactly the classification rule (1) with tokens as classes. And self-supervised vision models are routinely evaluated by the accuracy of a k -NN classifier on their features: if a simple vote among neighbours works, the representation has organised the data well.

3 Parzen windows: soft neighbourhoods

Hard windows. Instead of the k nearest points, let every point within a fixed radius h vote: $w_i(\mathbf{x}) = 1\{d(\mathbf{x}_i, \mathbf{x}) < h\}$ in (1). The neighbourhood now has fixed size and variable population; in sparse regions it may be empty, and a default prediction is needed, as for histograms.

Soft windows. Better still, let *all* training points vote with weights that decrease smoothly with distance, $w_i(\mathbf{x}) = K(\mathbf{x}_i, \mathbf{x})$ for a *kernel* K . The most common choice is the Gaussian (radial basis function) kernel

$$K_\sigma(\mathbf{x}_i, \mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{2\sigma^2}\right),$$

whose normalising constant cancels in the ratio (1). Regression with these weights,

$$f(\mathbf{x}) = \frac{\sum_i K_\sigma(\mathbf{x}_i, \mathbf{x}) y_i}{\sum_i K_\sigma(\mathbf{x}_i, \mathbf{x})}, \quad (2)$$

is the Nadaraya–Watson estimator; classification uses one-hot targets and an arg max, exactly as for k -NN. The width σ plays the role of k (Figure 3): small widths give fragmented regions that follow individual points, large widths give smooth boundaries that may miss structure.

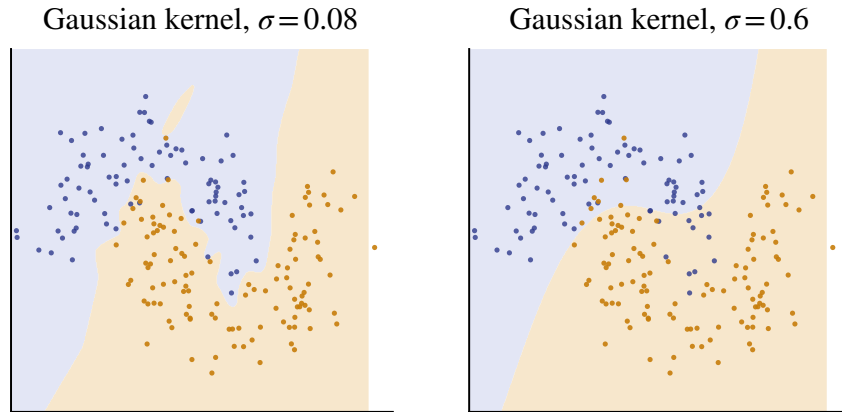


Figure 3. Parzen-window classification of 200 points with a narrow and a wide Gaussian kernel.

A numerical detail. For a query far from all data, or a small σ , every weight underflows to zero and (2) becomes $0/0$. Since the ratio is unchanged when all weights are multiplied by a constant, subtract the largest log-weight first:

```
def parzen_regress(Xtr, ytr, Xq, sigma):
    logW = -sq_distances(Xq, Xtr) / (2 * sigma**2)
    W = np.exp(logW - logW.max(axis=1, keepdims=True)) # largest weight is 1
    return (W @ ytr) / W.sum(axis=1)
```

Modern connection — attention is kernel smoothing

The attention operation at the heart of every transformer [15] computes, for a query vector $\mathbf{q}$, keys $\mathbf{k}_i$ and values $\mathbf{v}_i$,

$$\text{Attention}(\mathbf{q}) = \sum_i \frac{\exp(\mathbf{q}^\top \mathbf{k}_i / \sqrt{d})}{\sum_j \exp(\mathbf{q}^\top \mathbf{k}_j / \sqrt{d})} \mathbf{v}_i,$$

which is (2) with an exponential kernel: each value votes with a weight set by the similarity of its key to the query [14]. When queries and keys have unit norm, $\exp(\mathbf{q}^\top \mathbf{k} / \tau)$ is a Gaussian kernel in $\|\mathbf{q} - \mathbf{k}\|$ (Exercise E5). The “subtract the maximum” trick above is the same one used in every implementation of softmax. What a transformer adds is that keys, queries and values are *learned* functions of the input, so the network learns which neighbours should vote.

4 Kernel density estimation

For density estimation there are no targets: each training point contributes a small bump of probability, and the estimate is their average,

$$\hat{p}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n K_h(\mathbf{x} - \mathbf{x}_i),$$

where K_h is itself a probability density of width h , typically the isotropic Gaussian $\mathcal{N}(\mathbf{x}; \mathbf{x}_i, h^2 I)$ of Note 2. Because each bump is non-negative and integrates to one, so does $\hat{p}$. Unlike a histogram,

the estimate is smooth and does not depend on where bin edges happen to fall. The density panel of Note 1, Figure 1, was drawn this way.

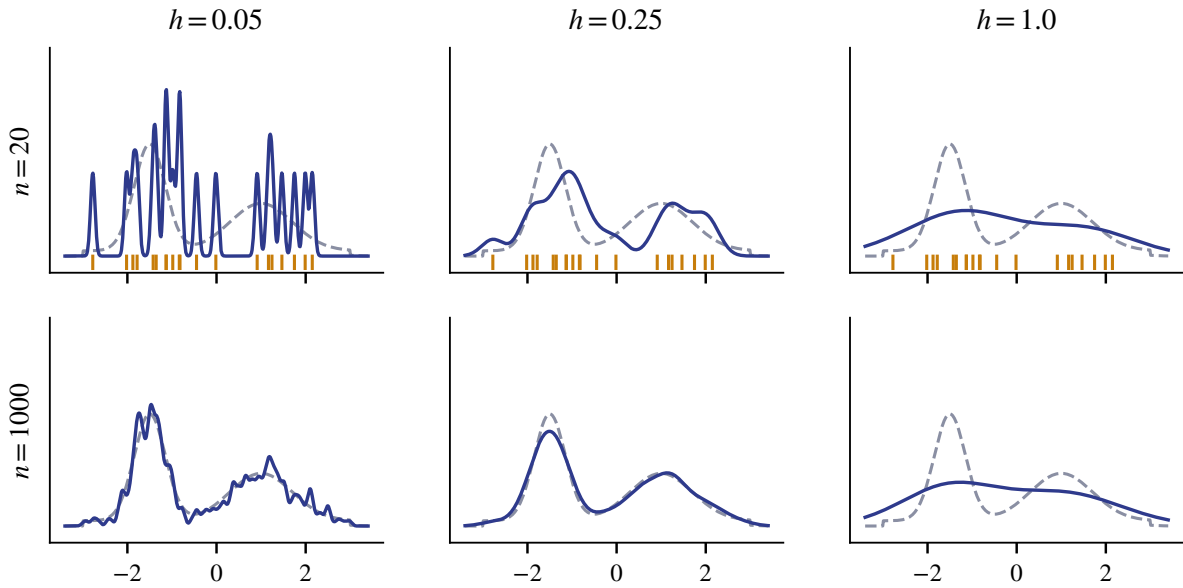


Figure 4. Kernel density estimates (solid) of a two-peaked density with a uniform background (dashed), from $n = 20$ points (top; ticks mark the data) and $n = 1000$ points (bottom), for three bandwidths.

The bandwidth h trades bias for variance (Figure 4). A narrow kernel puts a spike on each data point; a wide kernel blurs the two peaks into one. More data allow a narrower kernel. For a Gaussian kernel in one dimension the normal-reference rule $h = 1.06 \hat{\sigma} n^{-1/5}$ is a common starting point [11]; a more reliable choice maximises the log-likelihood of held-out data, or the leave-one-out log-likelihood (Note 4). With the best h the mean squared error falls as $n^{-4/5}$ in one dimension — faster than the $n^{-2/3}$ of histograms — but only as $n^{-4/(d+4)}$ in d dimensions: kernels soften the curse of dimensionality without lifting it.

In physics — unbinned shapes

When a particle-physics analysis needs the distribution of an observable for signal or background, but has only a finite set of simulated events, kernel estimation provides a smooth, unbinned template; adaptive kernels whose width grows in the tails, where events are scarce, are standard in the field [3].

5 Distances

Every method in this note is only as good as its notion of “near”. A *metric* on a set satisfies, for all $\mathbf{a}, \mathbf{b}, \mathbf{c}$:

1. non-negativity: $d(\mathbf{a}, \mathbf{b}) \geq 0$, with $d(\mathbf{a}, \mathbf{b}) = 0$ if and only if $\mathbf{a} = \mathbf{b}$;
2. symmetry: $d(\mathbf{a}, \mathbf{b}) = d(\mathbf{b}, \mathbf{a})$;
3. triangle inequality: $d(\mathbf{a}, \mathbf{c}) \leq d(\mathbf{a}, \mathbf{b}) + d(\mathbf{b}, \mathbf{c})$.

Minkowski distances. $d_p(\mathbf{a}, \mathbf{b}) = (\sum_j |a_j - b_j|^p)^{1/p}$ for $p \geq 1$. The cases $p = 1$ (Manhattan), $p = 2$ (Euclidean) and $p \rightarrow \infty$ ($\max_j |a_j - b_j|$) are the common ones. Their unit balls (Figure 5) show what each considers “equally far”. For $p < 1$ the formula still measures dissimilarity, but the triangle inequality fails (Exercise E4).

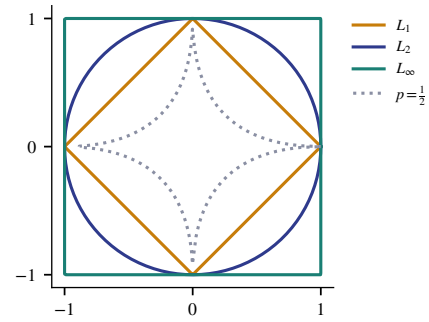


Figure 5. Unit balls $\{\mathbf{a} : d_p(\mathbf{a}, \mathbf{0}) = 1\}$ in $\mathbb{R}^2$.

Similarities for other data. For embeddings, the *cosine similarity* $\mathbf{a}^T \mathbf{b} / (\|\mathbf{a}\| \|\mathbf{b}\|)$ ignores length and compares direction; for binary fingerprints, such as the substructure bit vectors used to search chemical databases, the Tanimoto (Jaccard) similarity $|\mathbf{a} \wedge \mathbf{b}| / |\mathbf{a} \vee \mathbf{b}|$ counts shared bits relative to bits set in either vector.

When Euclidean distance misleads. Three problems are common, each with a standard remedy.

- *Units and scales.* A feature measured in GeV and another in radians contribute to the distance in proportions set by the units. Standardise each feature with the mean and standard deviation of the training set (Note 1, Section 5).
- *Correlations and irrelevant features.* Correlated features count the same information twice, and irrelevant ones add noise to every distance. The Mahalanobis distance of Note 2 corrects for correlations; feature selection or a learned projection removes irrelevant directions.
- *Invariances.* Two images of the same digit, shifted by two pixels, can be farther apart in pixel space than images of different digits. *Tangent distance* handles this by minimising the distance over small transformations of one of the images [12].

Modern connection — learning the metric

Today the distance is usually learned. Contrastive methods train a network so that two transformed views of the same image map to nearby embeddings and views of different images map far apart [1]; image–text models do the same for an image and its caption [10]. The invariances that tangent distance builds in by hand are learned from the transformations used in training, and the plain cosine or Euclidean distance in the learned space then works well for k -NN, retrieval and clustering.

In physics — a metric for collider events

Komisike, Metodiev and Thaler defined the distance between two collider events as the minimal “work” needed to rearrange one event’s energy flow into the other’s — an optimal-transport, or earth mover’s, distance — which respects the physical requirement that soft or collinear emissions should change it only slightly [7]. With this metric, a k -NN classifier on jets performs competitively, and geometric quantities of the space of events become measurable. Choosing the distance is where the physics enters.

Summary k -NN and Parzen windows predict by a weighted vote of training examples, with weights given by neighbourhood membership or by a kernel; the same formula serves regression, classification and, without targets, density estimation. They need no training, are consistent as $n \rightarrow \infty$, and have one main hyperparameter — k or the width — that trades bias for variance. Prediction costs $O(nd)$ per query unless approximate search is used. Their weak point is the distance: it must reflect the problem's units, correlations and invariances, which is why modern systems learn the representation and then use simple neighbourhood methods on top of it.

Exercises

- E1. Show that the 1-NN decision boundary between two training points of different classes lies on their perpendicular bisector. Sketch the Voronoi cells and the boundary for the four points $(0, 0)$, $(2, 0)$, $(0, 2)$, $(2, 2)$ labelled $+$, $-$, $-$, $+$.
- E2. Implement `knn_classify` and generate data of your own with two interleaved classes. Choose k by 5-fold cross-validation and plot the validation error against k . How does the selected k change when you double n ?
- E3. Show that the kernel density estimate integrates to one, and that $\mathbb{E}[\hat{p}(\mathbf{x})] = \int K_h(\mathbf{x} - \mathbf{u}) p(\mathbf{u}) d\mathbf{u}$. Conclude that for fixed h the estimate stays biased however large n is, and explain why h must decrease with n , but not too fast.
- E4. Find three points in $\mathbb{R}^2$ for which $d_{1/2}$ violates the triangle inequality.
- E5. Let $\|\mathbf{q}\| = \|\mathbf{k}_i\| = 1$. Show that $\exp(\mathbf{q}^\top \mathbf{k}_i / \tau)$ is proportional to a Gaussian kernel in $\|\mathbf{q} - \mathbf{k}_i\|$ and find its width. What happens to attention as $\tau \rightarrow 0$?
- E6. A retrieval system stores 10^7 embeddings in $\mathbb{R}^{768}$ as 32-bit floats. How much memory do they take, and how many multiply-add operations does one brute-force query need? At 10^{12} operations per second, how many queries per second is that?

Further reading. Hastie, Tibshirani and Friedman [4], sections 2.3, 6.1–6.6 and 13.3; Silverman [11] for kernel density estimation.

References

- [1] T. Chen, S. Kornblith, M. Norouzi and G. Hinton, A simple framework for contrastive learning of visual representations, *ICML* (2020).
- [2] T. M. Cover and P. E. Hart, Nearest neighbor pattern classification, *IEEE Trans. Inf. Theory* 13, 21–27 (1967).
- [3] K. Cranmer, Kernel estimation in high-energy physics, *Comput. Phys. Commun.* 136, 198–207 (2001).
- [4] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, 2nd ed., Springer (2009).
- [5] J. Johnson, M. Douze and H. Jégou, Billion-scale similarity search with GPUs, *IEEE Trans. Big Data* 7, 535–547 (2021).
- [6] U. Khandelwal *et al.*, Generalization through memorization: nearest neighbor language models, *ICLR* (2020).

- [7] P. T. Komiske, E. M. Metodiev and J. Thaler, Metric space of collider events, *Phys. Rev. Lett.* 123, 041801 (2019).
- [8] P. Lewis *et al.*, Retrieval-augmented generation for knowledge-intensive NLP tasks, *NeurIPS* (2020).
- [9] Y. A. Malkov and D. A. Yashunin, Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 824–836 (2020).
- [10] A. Radford *et al.*, Learning transferable visual models from natural language supervision, *ICML* (2021).
- [11] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*, Chapman & Hall (1986).
- [12] P. Simard, Y. LeCun and J. S. Denker, Efficient pattern recognition using a new transformation distance, *NeurIPS* (1993).
- [13] C. J. Stone, Consistent nonparametric regression, *Ann. Statist.* 5, 595–620 (1977).
- [14] Y.-H. H. Tsai *et al.*, Transformer dissection: a unified understanding of transformer’s attention via the lens of kernel, *EMNLP* (2019).
- [15] A. Vaswani *et al.*, Attention is all you need, *NeurIPS* (2017).

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.