Local Classification: Distances, k-NN, and Kernel Density Estimation

Consolidated Lecture Notes — CS6140 Machine Learning

---

1 Introduction: Similarity-Based, Local Classification

- Every method so far in this course (linear/ridge regression, decision trees) fits *one global structure* to the whole training set — a single weight vector w, or one tree. **Local** methods instead predict a query point $\mathbf{x}_t$ from whichever *training* points happen to be *similar* to it: k-NN, kernel density estimation, clustering, and collaborative filtering are all instances of this idea.
- Measuring similarity well is often the crux of the whole problem: once you have a good similarity/distance function, the prediction itself is almost trivial (a vote or an average). This is why this note spends its first half purely on similarity and distance functions, before ever touching k-NN itself.
- **No training phase.** Unlike regression or trees, local methods do no work up front; all computation happens at query time, comparing the query point against (some or all of) the training set. This is sometimes called *instance-based* or *lazy* learning — cheap to "fit," more expensive to predict.

Remark. **A preview, not a destination.** In the SVM dual formulation (Weeks 5–6), every training point also appears only through a similarity/dot-product with other points, $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ — exactly the object this note studies. We will develop that connection (kernels, Mercer's condition, feature maps) properly later; here we only need the similarity *functions* themselves, treated as fixed, hand-chosen formulas.

2 Distance and Similarity Measures

2.1 Distance Metrics: The Minkowski Family

A **distance metric** on a set X is a function $d : X \times X \to [0, \infty)$ satisfying, for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in X$:

$$\text{(i) } d(\mathbf{x},\mathbf{y}) = 0 \iff \mathbf{x} = \mathbf{y}, \qquad \text{(ii) } d(\mathbf{x},\mathbf{y}) = d(\mathbf{y},\mathbf{x}), \qquad \text{(iii) } d(\mathbf{x},\mathbf{y}) \le d(\mathbf{x},\mathbf{z}) + d(\mathbf{z},\mathbf{y}). \tag{1}$$

(X together with such a d is a *metric space*.) For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$, the **Minkowski** (or ℓ_p) distance is

$$\|\mathbf{x} - \mathbf{y}\|_p = \Big(\sum_{j=1}^{m} |x^j - y^j|^p \Big)^{1/p}, \qquad p > 0, \tag{2}$$

with three important special cases:

- $p = 2$: **Euclidean** distance, $\|\mathbf{x}-\mathbf{y}\|_2 = \sqrt{\sum_j (x^j - y^j)^2}$ — HW1 Problem 4 uses this for Spambase.

- $p = 1$: **Manhattan** (taxicab/cityblock) distance, $\|\mathbf{x} - \mathbf{y}\|_1 = \sum_j |x^j - y^j|$ — sums absolute per-feature differences instead of the squared ones; more robust to a single large outlier feature.
- $p = \infty$: **Chebyshev/supremum** distance, $\|\mathbf{x} - \mathbf{y}\|_\infty = \max_j |x^j - y^j|$ — only the single most-different feature matters.

Remark. The triangle inequality (axiom (iii) above) only holds for $p \geq 1$; for $0 < p < 1$, $\|\cdot\|_p$ is still a common "distance" in practice (e.g. for sparse-signal applications) but is not, technically, a metric.

Remark. Euclidean distance implicitly assumes "one unit of difference" means the same thing on every feature. If feature 3 ranges over $[0, 1]$ and feature 7 ranges over $[0, 1000]$, feature 7 will dominate $\|\mathbf{x} - \mathbf{y}\|_2$ regardless of how informative feature 3 actually is.

2.2 Normalization: Why Scale Matters

Before computing any Minkowski distance on real data, features are typically rescaled to comparable ranges:

- **Min-max**: $x^j \mapsto \dfrac{x^j - \min_i x_i^j}{\max_i x_i^j - \min_i x_i^j} \in [0, 1]$.
- **Z-score standardization**: $x^j \mapsto \dfrac{x^j - \bar{x}^j}{\sigma^j}$ (subtract the training mean, divide by the training standard deviation) — puts every feature at mean 0, unit variance.
- **Log attenuation**: $x^j \mapsto \operatorname{sign}(x^j) \log(|x^j| + 1)$ — compresses heavy-tailed features (e.g. word counts, dollar amounts) so a handful of huge values don't dominate the distance.

Key point: This is exactly the same issue flagged for ridge regression's penalty $\lambda \sum_j (w^j)^2$ (Regression notes, Implementation Notes): any method that treats all features symmetrically in one formula — a penalty, or here a distance — silently reweights them by whatever scale they happen to be measured in, unless you standardize first. Always normalize/standardize before computing distances on raw features with different units.

2.3 Mahalanobis Distance

Z-score standardization treats every feature independently; the **Mahalanobis distance** additionally accounts for *correlations* between features, using the data's covariance matrix Σ:

$$d_{Mahal}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})\, \Sigma^{-1}\, (\mathbf{x} - \mathbf{y})^T}. \tag{3}$$

If features are independent with variances σ_j^2 (so $\Sigma = \operatorname{diag}(\sigma_1^2, \ldots, \sigma_m^2)$), this reduces to Euclidean distance on z-scored data, $\sqrt{\sum_j \left(\frac{x^j - y^j}{\sigma_j}\right)^2}$; with $\Sigma = I$ it is exactly Euclidean distance. The general Σ^{-1} additionally "un-skews" correlated features, so that two points differing along a direction of naturally high joint variance are not treated as more different than two points differing by the same raw amount along a direction where the data barely varies.

Worked example: $\Sigma = \begin{bmatrix} 0.3 & 0.2 \\ 0.2 & 0.3 \end{bmatrix}$, so $\Sigma^{-1} = \begin{bmatrix} 6 & -4 \\ -4 & 6 \end{bmatrix}$ (since $\det \Sigma = 0.05$). For $\mathbf{x} = (0, 1)$, $\mathbf{y} = (0.5, 0.5)$, $\mathbf{z} = (1.5, 1.5)$:

$$(\mathbf{x}-\mathbf{y})\,\Sigma^{-1}(\mathbf{x}-\mathbf{y})^T = 5 \;\Rightarrow\; d_{Mahal}(\mathbf{x},\mathbf{y}) = \sqrt{5} \approx 2.24, \qquad (\mathbf{y}-\mathbf{z})\,\Sigma^{-1}(\mathbf{y}-\mathbf{z})^T = 4 \;\Rightarrow\; d_{Mahal}(\mathbf{y},\mathbf{z}) = 2.$$

(Compare: the plain Euclidean distances here are $\|\mathbf{x}-\mathbf{y}\|_2 = \sqrt{0.5} \approx 0.71$ and $\|\mathbf{y}-\mathbf{z}\|_2 = \sqrt{2} \approx 1.41$ — Mahalanobis distance inflates both, and by different relative amounts, because Σ's positive off-diagonal correlation makes the diagonal direction "cheaper" than the axes.)

2.4 Hamming Distance (Categorical Data)

For $\mathbf{x}, \mathbf{y}$ with m **nominal** (categorical, non-numeric) attributes:

$$d_{Hamm}(\mathbf{x},\mathbf{y}) = \sum_{j=1}^{m} \mathbb{I}(x^j \neq y^j) \tag{4}$$

— simply the count of attributes that disagree; equivalent to ℓ_1 distance on a one-hot (binary indicator) encoding. E.g. $\mathbf{x}$ =(big, black, cat), $\mathbf{y}$ =(small, black, rat), $\mathbf{z}$ =(big, blue, bulldog): $d_{Hamm}(\mathbf{x},\mathbf{y}) = 2$ (differ on size, species), $d_{Hamm}(\mathbf{x},\mathbf{z}) = 2$ (differ on color, species), $d_{Hamm}(\mathbf{y},\mathbf{z}) = 3$ (differ on all three).

2.5 Similarity and Affinity Functions

A **similarity** (or *affinity*) function is $a : X \times X \to [0, 1]$ with $a(\mathbf{x},\mathbf{x}) = 1$ (maximal self-similarity) and $a(\mathbf{x},\mathbf{y}) = a(\mathbf{y},\mathbf{x})$ (symmetric) — the complementary notion to a distance, higher meaning "more alike" instead of "more apart."

Dot product and cosine similarity. The dot product $\langle \mathbf{x},\mathbf{y}\rangle = \sum_j x^j y^j$ is a natural raw similarity: it carries over all the usual geometric notions of angle and length,

$$\cos(\mathbf{x},\mathbf{y}) = \frac{\langle \mathbf{x},\mathbf{y}\rangle}{\|\mathbf{x}\|\,\|\mathbf{y}\|} \in [-1, 1],$$

the **cosine similarity** — the dot product normalized to remove the effect of vector *length*, leaving only *direction*. HW1 Problem 4 uses cosine distance ($1 - \cos(\mathbf{x},\mathbf{y})$) for MNIST: two images of the same digit at different brightness/ink-thickness have very different Euclidean norms but nearly identical direction, so cosine similarity is far more appropriate there than raw Euclidean distance.

Pearson correlation. Centering both vectors before taking cosine similarity gives the **Pearson correlation coefficient**,

$$\rho(\mathbf{x},\mathbf{y}) = \frac{\sum_j (x^j - \bar{x})(y^j - \bar{y})}{\sqrt{\sum_j (x^j - \bar{x})^2}\sqrt{\sum_j (y^j - \bar{y})^2}} = \cos(\mathbf{x} - \bar{x},\ \mathbf{y} - \bar{y}) \in [-1, 1], \tag{5}$$

which additionally ignores any constant additive offset between $\mathbf{x}$ and $\mathbf{y}$ (not just scale, like cosine). This is the standard similarity measure in **collaborative filtering** (e.g. "users who rated similarly to you also liked ..."): $\mathbf{x}, \mathbf{y}$ are two users' rating vectors, and ρ is robust to one user habitually rating a full point higher than another on the same shared scale.

Gaussian/RBF affinity. Given any distance d, one standard way to convert it into a bounded similarity is

$$k(\mathbf{x}, \mathbf{y}) = \exp\Big(- \frac{d(\mathbf{x}, \mathbf{y})^2}{2\epsilon^2} \Big) \in (0, 1], \tag{6}$$

often called the **Gaussian** or **RBF** (radial basis function) affinity when d is Euclidean; `sklearn`'s `rbf_kernel` uses the equivalent form $k(\mathbf{x}, \mathbf{y}) = \exp(-\gamma \|\mathbf{x} - \mathbf{y}\|^2)$, i.e. $\gamma = 1/(2\epsilon^2)$. Two points are similar if they lie within the same "spherical neighborhood" of radius $\sim \epsilon$.

Polynomial similarity. Another fixed similarity formula used directly in HW1 Problem 4 (MNIST, degree 2) is

$$k(\mathbf{x}, \mathbf{y}) = \big(\gamma \langle \mathbf{x}, \mathbf{y} \rangle + c\big)^d, \tag{7}$$

`sklearn`'s `polynomial_kernel` parameterization (γ, `coef0`$= c$, `degree`$= d$).

Key point: We are using $k(\mathbf{x}, \mathbf{y})$ here purely as a *formula* for "how similar are these two points," the same role Euclidean or cosine distance plays. **We are not yet asking** which such formulas correspond to a valid dot product in some feature space, or how to exploit that fact algorithmically (the "kernel trick") — that theory (Mercer's condition, feature maps, the SVM dual) is developed properly in Weeks 5–6. Everything in this note only needs the formula itself, computed directly on the original features.

Remark. A concrete gotcha with the polynomial formula, already seen in HW1's own solution code: with $c = 0$ (`coef0=0`), $k(\mathbf{x}, \mathbf{y}) = (\gamma \langle \mathbf{x}, \mathbf{y} \rangle)^d$ ranks neighbors *identically* to the raw dot product $\langle \mathbf{x}, \mathbf{y} \rangle$ — *provided* $\langle \mathbf{x}, \mathbf{y} \rangle \geq 0$ for every comparison being ranked, since raising a non-negative base to a fixed power d is monotonic. This holds automatically for both Spambase (word counts ≥ 0) and MNIST (pixel intensities ≥ 0); it can fail for arbitrary real-valued (e.g. centered/standardized) features, where an even d breaks monotonicity outright — $(-3)^2 = 9 > (-1)^2 = 1$ even though $-3 < -1$. As soon as $c > 0$ (`sklearn`'s default is `coef0=1`), the relationship to the raw dot product breaks regardless of sign, and the induced neighbor ranking can genuinely differ. Comparing a from-scratch implementation against `sklearn`'s `polynomial_kernel` without matching `coef0` silently compares two different similarity functions, not a bug in either one.

2.6 Similarity for Binary/Categorical Attributes: Jaccard and Friends

For n binary attributes, two common similarity measures:

$$\text{Simple Matching Coefficient: } SMC(\mathbf{x}, \mathbf{y}) = \frac{\#\{j : x^j{=}y^j{=}1\} + \#\{j : x^j{=}y^j{=}0\}}{n} \tag{8}$$

$$\text{Jaccard coefficient: } J(\mathbf{x}, \mathbf{y}) = \frac{\#\{j : x^j{=}1 \wedge y^j{=}1\}}{\#\{j : x^j{=}1 \vee y^j{=}1\}} \tag{9}$$

SMC counts *both* agreements (both 1 *and* both 0); J only counts positive ("1") agreements over the union of positive attributes — the right choice when most attributes are 0 for most points (e.g. a large sparse vocabulary, where agreeing on the *absence* of most words is uninformative). The **Tanimoto** coefficient extends Jaccard to continuous, non-binary attributes: $T(\mathbf{x}, \mathbf{y}) = \dfrac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - \langle \mathbf{x}, \mathbf{y} \rangle}$.

2.7 Combining Multiple Similarities

When different attributes naturally call for different similarity functions (e.g. some numeric, some categorical), a common fix is a **weighted combination**:

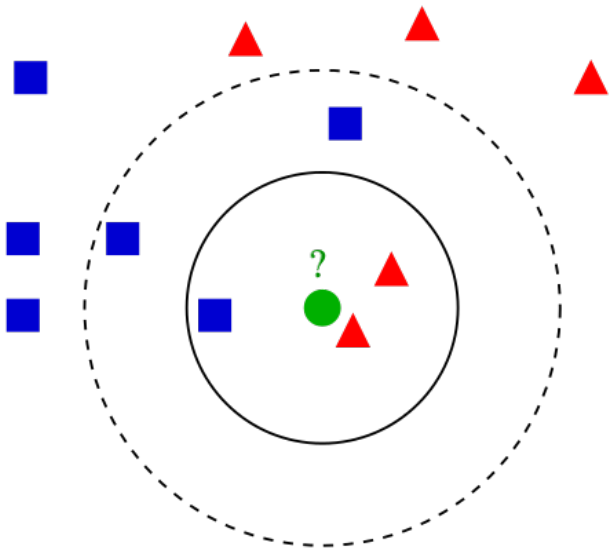
$$a(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{m} w_j \, a_j(\mathbf{x}, \mathbf{y}), \qquad \sum_j w_j = 1, \tag{10}$$

where a_j is whatever similarity is appropriate for attribute j (Hamming-style for categorical, Gaussian for numeric, etc.), weighted by how much each attribute should count toward the total. A refinement handles *missing/incomparable* attributes by a binary indicator $\delta_j(\mathbf{x}, \mathbf{y}) \in \{0, 1\}$ ("do both points have usable data for attribute j?") and renormalizes only over comparable attributes: $a(\mathbf{x}, \mathbf{y}) = \frac{\sum_j w_j \delta_j(\mathbf{x},\mathbf{y}) a_j(\mathbf{x},\mathbf{y})}{\sum_j w_j \delta_j(\mathbf{x},\mathbf{y})}$.

3 k-Nearest-Neighbors (k-NN)

Given a distance/similarity function, the simplest possible local predictor: to predict at query point $\mathbf{x}_t$, find the k closest *training* points, and

- **classification**: predict the majority class among the k neighbors;
- **regression**: predict the average label among the k neighbors.



For $k = 1$: nearest point is red ⇒ predict red. For $k = 3$: majority of 3 nearest is still red. For $k = 5$: majority of 5 nearest flips to blue. Changing k can change the prediction at the very same query point.

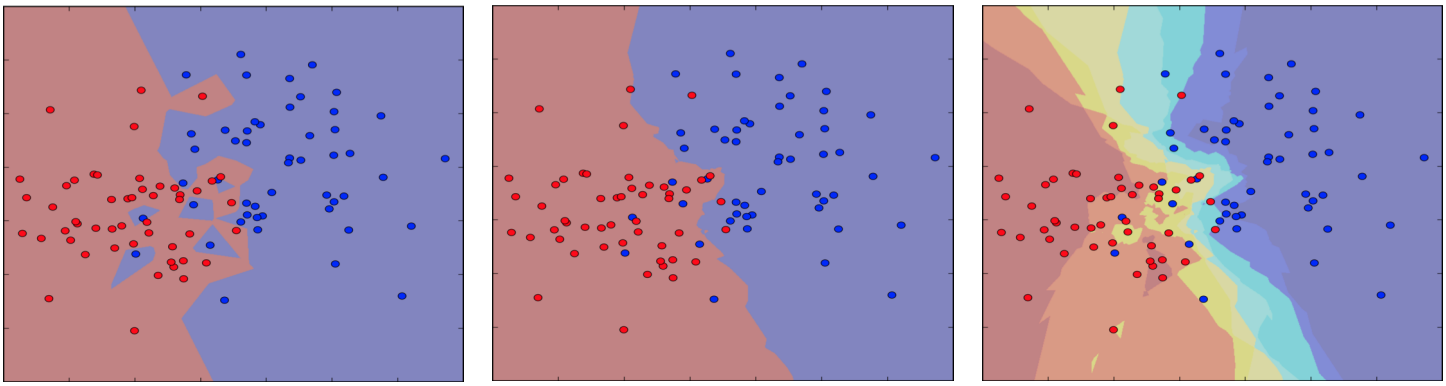
Worked example (2 real Spambase features, `word_freq_free` and `word_freq_money`; we reuse these same 5 points and query throughout this note):

	A	B	C	D	E	query $\mathbf{x}_t$
`free`	0	0	0.34	5	6.25	0.3
`money`	0	0.42	0	0	0	0.1
label	non-spam	non-spam	**spam**	**spam**	**spam**	?

Euclidean distances to the query, sorted: $d(\mathbf{x}_t, C)$=0.108, $d(\mathbf{x}_t, A)$=0.316, $d(\mathbf{x}_t, B)$=0.439, $d(\mathbf{x}_t, D)$=4.70, $d(\mathbf{x}_t, E)$=5.95.

- k=1: nearest is C (spam) $\Rightarrow$ predict **spam**.
- k=3: three nearest are $\{C$=spam, A=non-spam, B=non-spam$\}$ $\Rightarrow$ majority **non-spam** — the prediction *flips* versus $k = 1$.

Key point: This is exactly why HW1 Problem 4 asks you to compare $k = 1, 3, 7$ on the same data: small k is low-bias/high-variance (one noisy or mislabeled neighbor can flip the prediction, as C alone does above); large k is smoother but can wash out genuinely local structure by pulling in irrelevant, far-away points (here, including D, E for large enough k would drag the vote back toward spam) — the exact same bias–variance knob as tree depth (Decision Trees notes) and ridge's λ (Regression notes), just turned by a different mechanism.



1-NN decision boundary (left) is jagged, exactly fitting every training point (like an unpruned decision tree); 7-NN classification (middle) is smoother; 7-NN regression (right) similarly averages over more neighbors.

Remark. A similarity function can replace the distance directly, but only if it is genuinely a function of that distance. Gaussian/RBF affinity, $k(\mathbf{x}, \mathbf{y}) = \exp(-d(\mathbf{x}, \mathbf{y})^2/2\epsilon^2)$, *is* — a monotonically decreasing function of d alone — so ranking neighbors by largest k gives the identical neighbor set as ranking by smallest d, no new algorithm needed. The `coef0=0` polynomial similarity is different in kind: it is a function of the dot product $\langle \mathbf{x}, \mathbf{y} \rangle$, not of Euclidean distance directly, so it reproduces *dot-product* ranking (Remark 2.5), which agrees with Euclidean-distance ranking only when every candidate $\mathbf{y}$ has the same norm — from $\|\mathbf{x}-\mathbf{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\langle \mathbf{x}, \mathbf{y} \rangle$, a larger dot product implies a smaller distance only if $\|\mathbf{y}\|$ does not vary across the points being compared. On raw (unnormalized) MNIST pixel vectors this need not hold, so polynomial-similarity neighbors and Euclidean-distance neighbors can legitimately differ — not a bug, just a different (angular/inner-product) notion of closeness.

4 Fixed-Window (Radius) k-NN

Instead of fixing the *count* k, fix a **radius** R around the query point, and use *all* training points within it:

Algorithm: Fixed-Window (R-Radius) k-NN

1. Choose a radius R.
2. For query $\mathbf{x}_t$, construct the neighborhood $NN(\mathbf{x}_t) = \{\mathbf{x}_i \text{ training} : d(\mathbf{x}_i, \mathbf{x}_t) \leq R\}$.
3. Predict the majority class (classification) or average label (regression) over $NN(\mathbf{x}_t)$.

Unlike fixed-k, the neighborhood *size* now varies from query to query — exactly the point, and exactly the source of its failure mode (§5).

Worked example, continued (same 5 points and query as §3):

- $R = 1$: $NN(\mathbf{x}_t) = \{A, B, C\}$ (all three within distance 1) $\Rightarrow$ majority non-spam (same as k=3 above, since here the 3 nearest happen to all lie within radius 1).
- $R = 0.2$: only C (d=0.108) qualifies $\Rightarrow$ $NN(\mathbf{x}_t) = \{C\}$ $\Rightarrow$ predict spam.
- $R = 0.05$: *nobody* qualifies (even C is at distance $0.108 > 0.05$) $\Rightarrow$ $NN(\mathbf{x}_t) = \emptyset$ — **fixed-window k-NN cannot predict at all here.**

5 Fixed-k vs. Fixed-Window: The Density Tradeoff (HW1 Problem 6)

The $R = 0.05$ case above is not a corner case invented for this note — it is exactly HW1 Problem 6's failure mode, and it is entirely a consequence of **local data density** varying across feature space:

- In a **dense** region (many training points packed close together, as around our query point above), a small fixed radius R already captures several neighbors, and fixed-window k-NN behaves sensibly — often *better* than fixed-k, since it naturally uses more evidence where more relevant evidence exists.
- In a **sparse** region (query point far from most training data), the same fixed R may capture zero neighbors ($NN(\mathbf{x}_t) = \emptyset$, as above) — **fixed-window fails to predict at all.** Fixed-k never has this problem: it always returns exactly k neighbors, by construction.
- But fixed-k's robustness in sparse regions is bought at a cost: those k "nearest" neighbors may be *very* far away (imagine k=3 demanded in a region where the closest real neighbor is at distance 50) — arguably not meaningfully similar to the query point at all, yet they still get an equal vote.

Key point: Fixed-k and fixed-window fail in *opposite* regimes of the same underlying problem (non-uniform density): fixed-window fails by predicting nothing where data is sparse; fixed-k fails by predicting confidently from irrelevant, distant points where data is sparse. Neither failure is visible in a dense, uniform toy dataset — both only show up once density varies, which real data (Spambase, MNIST) always does.

Remark. Two natural hybrid fixes, corresponding to each failure mode: (a) *fixed-window with a fallback*: if $NN(\mathbf{x}_t) = \emptyset$, fall back to the single nearest neighbor (or expand R until non-empty) rather than refusing to predict; (b) *fixed-k with a distance cap*: take the k nearest neighbors, but only count/weight those within some sanity-check maximum distance, rather than blindly trusting all k regardless of how far away they are. Weighting every neighbor continuously by distance — rather than hard-including or hard-excluding it — is exactly the idea developed next.

6 Weighted/Soft k-NN: Kernel Density Estimation (HW1 Problem 4C)

Instead of a hard cutoff (top-k, or within-R), **weight every training point** by a similarity/kernel function of its distance to the query, so nearer points count more and farther points count less, continuously:

$$\hat{p}(\mathbf{x} \mid C) = \frac{1}{N_C} \sum_{i:\, label(\mathbf{x}_i)=C} k(\mathbf{x}, \mathbf{x}_i) \tag{11}$$

(an average of a kernel "bump" centered at each same-class training point — this is exactly **Parzen-window** / kernel density estimation). If k integrates to 1 (a proper density kernel), this is a valid estimate of the class-conditional density; treating $\hat{P}(C) = N_C/N$ (empirical class frequency) as the prior, Bayes' rule gives

$$\hat{P}(C \mid \mathbf{x}) = \frac{\hat{P}(\mathbf{x} \mid C)\, \hat{P}(C)}{\hat{P}(\mathbf{x})} = \frac{\sum_{i:label(\mathbf{x}_i)=C} k(\mathbf{x}, \mathbf{x}_i)}{\sum_i k(\mathbf{x}, \mathbf{x}_i)} \tag{12}$$

(the N_C, N normalizations cancel between numerator and denominator). In words: **count each training point not as a vote of 1, but as a vote of** $k(\mathbf{x}, \mathbf{x}_i)$ — a strictly smoother, continuous relaxation of k-NN's hard 0/1 vote, using *every* training point instead of a hard-selected top-k or within-R subset.

Worked example, continued (same 5 points and query; Gaussian kernel $k(\mathbf{x}, \mathbf{x}_i) = \exp(-d(\mathbf{x}, \mathbf{x}_i)^2/2)$, i.e. $\epsilon = 1$):

	A	B	C	D	E
d^2	0.10	0.192	0.012	22.1	35.4
$k(\mathbf{x}_t, \cdot)$	0.951	0.908	0.994	$\approx$0	$\approx$0

$\sum_{\text{non-spam}} k = 0.951 + 0.908 = 1.860$, $\sum_{\text{spam}} k = 0.994$ (D, E contribute essentially nothing at this bandwidth — they are simply too far):

$$\hat{P}(\text{non-spam} \mid \mathbf{x}_t) = \frac{1.860}{1.860 + 0.994} = 0.65, \qquad \hat{P}(\text{spam} \mid \mathbf{x}_t) = 0.35.$$

This **agrees with the** k=3 **majority vote** from §3 (non-spam) — with this bandwidth, the far points D, E are automatically down-weighted to near-zero, so KDE and 3-NN happen to agree here. A much wider bandwidth would let D, E contribute non-trivially and could pull the prediction back toward spam — bandwidth in KDE plays exactly the role k or R play above.

7 k-NN for Feature Selection: The RELIEF Algorithm

The same "nearby points" idea can score *features* instead of predicting labels. Intuition: feature j is useful near training point $\mathbf{x}_i$ if same-class neighbors tend to agree on it, while opposite-class neighbors tend to disagree. Concretely, RELIEF iterates over training points $\mathbf{x}_i$; for each, find its nearest same-class neighbor $\mathbf{z}_{same}$ and nearest opposite-class neighbor $\mathbf{z}_{opp}$ (from $NN(\mathbf{x}_i)$, by any distance above), and update each feature's running quality score:

$$Q^j \leftarrow Q^j - \left(x_i^j - z_{same}^j\right)^2 + \left(x_i^j - z_{opp}^j\right)^2. \tag{13}$$

A feature that stays close within-class but differs across classes accumulates a *large positive* Q^j (useful, discriminative); a feature that varies just as much within a class as across classes accumulates $Q^j \approx 0$ (useless).

Worked example (same 5 Spambase points, feature $j =$`word_freq_free`, evaluated at $\mathbf{x}_i = C = 0.34$, whose nearest same-class (spam) neighbor is $D = 5$ and nearest opposite-class (non-spam) neighbor is $A = 0$):

$$\Delta Q^{\texttt{free}} = -(0.34 - 5)^2 + (0.34 - 0)^2 = -21.72 + 0.12 = -21.6.$$

A large *negative* update — at this particular point, `word_freq_free` looks *unhelpful*: the same-class neighbor D differs enormously from C on this feature, while the opposite-class neighbor A is actually closer to C on it. This is not a contradiction of `word_freq_free` being a broadly useful spam indicator; it is a reminder that a single point's update is noisy (here, driven by D happening to be a distant same-class neighbor) — RELIEF's score is only meaningful after *averaging* this update over many training points $\mathbf{x}_i$, not from one point in isolation.

8 Aside: Comparing Misaligned Sequences (Dynamic Time Warping)

Euclidean distance compares vectors position-by-position, which fails when two sequences represent the "same shape" at different speeds or offsets (e.g. two spoken recordings of the same word, one said faster than the other). **Dynamic Time Warping (DTW)** instead finds the lowest-cost *alignment* between sequences $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_m)$ (possibly of different lengths $n \neq m$), via the dynamic program

$$DTW[i,j] = (x_i - y_j)^2 + \min\big(DTW[i{-}1,j],\ DTW[i,j{-}1],\ DTW[i{-}1,j{-}1]\big), \qquad DTW[0,0] = 0, \quad (14)$$

with the DTW dissimilarity being $DTW[n,m]$, and the optimal alignment recovered by backtracking the path of minimizing choices.

Worked example: $\mathbf{x} = (0, 2, 4)$, $\mathbf{y} = (0, 1, 3, 4)$:

	y_1=0	y_2=1	y_3=3	y_4=4
x_1=0	0	1	10	26
x_2=2	4	1	2	6
x_3=4	20	10	2	**2**

$DTW(\mathbf{x}, \mathbf{y}) = 2$, achieved by aligning $0 \leftrightarrow 0$, $2 \leftrightarrow$1-or-3, $4 \leftrightarrow 4$ — far cheaper than forcing a rigid 1-to-1 comparison of two different-length sequences.

Remark. A related idea, the **Earth Mover's Distance (EMD)**, asks a different question: what is the cheapest way to transform one whole *distribution* into another, moving probability mass Ω_{ij} from bin i to bin j at cost $|i - j|^p$? Both DTW and EMD replace a rigid, position-by-position comparison with an optimized correspondence — solvable by dynamic programming (DTW) or as an optimal-transport/assignment problem (EMD, via the Hungarian algorithm or faster specialized methods).

9 Implementation Notes (HW1)

HW1 Problem 4 implements k-NN with no training phase — all of the work below happens inside `predict`, called once per query point.

- **(A) Fixed k** ($k = 1, 3, 7$): Spambase with Euclidean distance; MNIST with cosine distance, a Gaussian kernel, and a degree-2 polynomial kernel (§2.5–3 above, in that order).
- **(B) Fixed window** (§4): radius R instead of count k; Spambase (Euclidean), MNIST (cosine).
- **(C) Kernel density estimation** (§6, optional): per-class $\hat{P}(\mathbf{x} \mid C)$ with a Gaussian kernel, Spambase.
- **Library calls**:
  - `KNeighborsClassifier` (part A)
  - `RadiusNeighborsClassifier` (part B)
  - `rbf_kernel`, `polynomial_kernel` (both parts, MNIST).

Common bugs (all silent — wrong numbers, not a crash):

- **Comparing distances computed two different ways**: fitting with one metric (say Euclidean) but scoring/plotting with another silently produces nonsensical neighbor rankings; always fix the distance/similarity function *before* touching k or R.
- `coef0` **mismatch** (Remark 2.5): comparing a from-scratch polynomial-kernel implementation against `sklearn`'s without matching `coef0` compares two different similarity functions — this is a real bug this course's own HW1 solution code had, caught only by checking that library and from-scratch numbers matched *exactly*, not just approximately.
- **Distance vs. similarity sign confusion**: "cosine distance" is $1 - \cos(\mathbf{x}, \mathbf{y})$, not $\cos(\mathbf{x}, \mathbf{y})$ itself — using similarity directly where a distance (something to *minimize*) is expected inverts every neighbor ranking.
- **Self-inclusion leakage**: if computing distances from the *training* set to itself (e.g. for a sanity check or a leave-one-out estimate), a point's distance to itself is always 0 and will dominate any k-NN vote unless explicitly excluded.
- **Cost**: naively, one query costs $O(N)$ distance computations (compare against every training point) — fine at HW1's scale, but the reason production systems use spatial index structures (KD-trees, ball trees, approximate nearest-neighbor methods) instead, not required here.

10 Optional/Advanced: The Curse of Dimensionality

Every method in this note rests on one assumption: that "nearest" is meaningful — that some training points are *genuinely* closer to a query than others. This assumption quietly breaks down as the number of features m grows, in a way that is precise enough to derive.

Let $\mathbf{x}, \mathbf{y} \in [0,1]^m$ have independent, uniformly random coordinates (a stand-in for "m unrelated, comparably-scaled features"). Their squared Euclidean distance is a sum of m i.i.d. terms,

$$\|\mathbf{x}-\mathbf{y}\|_2^2 = \sum_{j=1}^{m}(x^j - y^j)^2, \qquad \text{each term i.i.d. with mean } \mu = \tfrac{1}{6},\ \text{variance } \sigma^2 = \tfrac{7}{180}$$

(the mean and variance of $(x^j - y^j)^2$ for $x^j, y^j \sim \mathrm{Unif}(0,1)$, independent — a short calculus exercise, not derived here). Because the m terms are independent, the sum's mean and variance simply add:

$$\mathbb{E}\big[\|\mathbf{x}-\mathbf{y}\|_2^2\big] = m\mu = \frac{m}{6}, \qquad \mathrm{Var}\big[\|\mathbf{x}-\mathbf{y}\|_2^2\big] = m\sigma^2 = \frac{7m}{180}. \tag{15}$$

The *relative* spread of the distance around its mean — the coefficient of variation — is

$$CV_m = \frac{\sqrt{\mathrm{Var}}}{\mathbb{E}} = \frac{\sqrt{7m/180}}{m/6} = \sqrt{\frac{7}{5}}\cdot\frac{1}{\sqrt{m}} \approx \frac{1.18}{\sqrt{m}} \xrightarrow{m\to\infty} 0. \tag{16}$$

m	1	10	100	1,000	10,000
CV_m (relative spread of $\|\mathbf{x}-\mathbf{y}\|_2^2$)	1.18	0.37	0.12	0.037	0.012

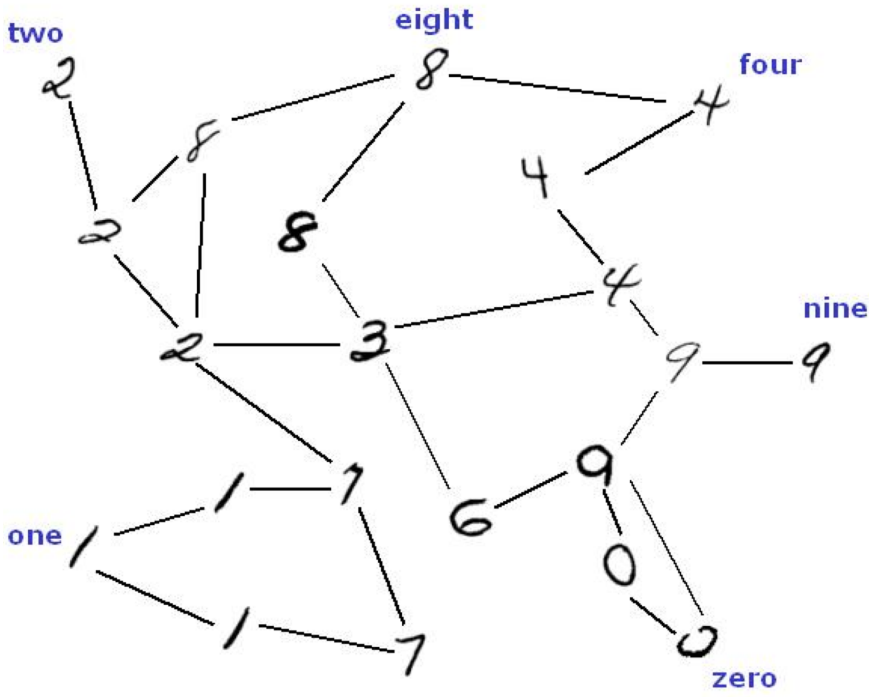
(Verified by direct simulation, not just the formula: sampling 200,000 random pairs at each m reproduces these numbers to 3 decimal places.)

Key point: As m grows, every pairwise distance concentrates tightly around the *same* value $m/6$ — the "nearest" neighbor and the "farthest" neighbor of a random query point become, in relative terms, almost equally far away. This is the **curse of dimensionality** for distance-based methods: k-NN, KDE, and RBF/Gaussian similarity all silently degrade in high dimensions, not because of a bug, but because the entire notion of "close vs. far" they rely on becomes statistically meaningless once m is large enough, *even if every feature individually is perfectly reasonable.*

Remark. This is exactly why §7's RELIEF algorithm (or any feature selection/dimensionality reduction, e.g. PCA in Weeks 5–6) is not just an accuracy nicety but often a *prerequisite* for distance-based methods on high-dimensional data (raw MNIST is 784 pixels): reducing m to the handful of features that actually carry signal is what keeps "nearest neighbor" meaningful in the first place. This also explains why cosine similarity (§2.5) is often preferred over Euclidean distance on very high-dimensional, sparse data (e.g. text) — normalizing away vector length removes one source of the concentration effect, though not the underlying dimensionality problem itself.

11 Optional: Local Classification via Graph Harmonic Functions

A more sophisticated way to use only pairwise similarities $k_{it} = k(\mathbf{x}_i, \mathbf{x}_t)$ (following Zhu, Ghahramani & Lafferty, *Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions*): build a weighted graph with one node per datapoint (labeled *and* unlabeled) and edge weights k_{it}, then solve for label values that are *smooth* over this graph.



A similarity graph: nodes are datapoints, edges are similarity values k_{ij} (missing edge = k_{ij}=0). Labeled points are marked in blue; the task is to infer values for the rest.

Physical analogy: think of f (the label/value function) as *temperature*, labeled points as fixed hot/cold sources, and k_{it} as how well heat conducts between i and t. The equilibrium (steady-state) temperature distribution minimizes the energy

$$E(f) = \frac{1}{2}\sum_{i,t} k_{it}\big(f(\mathbf{x}_i) - f(\mathbf{x}_t)\big)^2, \tag{17}$$

which penalizes disagreement between f-values in proportion to how similar the two points are — highly similar points (k_{it} large) are forced to nearly agree, while dissimilar points (k_{it} small) are barely constrained relative to each other.

Theorem (Zhu–Ghahramani–Lafferty). The minimizer of $E(f)$ is **harmonic** on unlabeled points: each unlabeled point's value is exactly the similarity-weighted average of its neighbors',

$$f(\mathbf{x}_i) = \frac{1}{d_i}\sum_{t:k_{it}>0} k_{it}\, f(\mathbf{x}_t), \qquad d_i = \sum_{t:k_{it}>0} k_{it}, \tag{18}$$

equivalently $f = Pf$ with $P = D^{-1}K$ ($D = \mathrm{diag}(d_i)$), or $(D-K)f = 0$; this solution is unique. Partitioning into labeled (l) and unlabeled (u) points and their similarity blocks,

$$K = \begin{bmatrix} K_{ll} & K_{lu} \\ K_{ul} & K_{uu} \end{bmatrix}, \qquad f = \begin{bmatrix} f_l \\ f_u \end{bmatrix},$$

the unlabeled values are recovered in closed form: $f_u = (D_{uu} - K_{uu})^{-1}K_{ul}f_l = (I - P_{uu})^{-1}P_{ul}f_l$.

Remark. Like the closed-form solutions in the Regression notes, this exact matrix inverse is often impractical at scale; a numerical optimizer minimizing $E(f)$ directly is used in practice instead. Unlike k-NN or KDE, this method exploits similarities *between unlabeled points too* (via the graph structure), not just similarities to labeled points — information k-NN and KDE both ignore entirely.

12 Where This Goes Next

- The similarity/kernel *formulas* used informally throughout this note (Gaussian, polynomial, dot product) return properly in Weeks 5–6, where we ask a sharper question: for which such k does there *provably* exist a feature map Φ with $k(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$ (Mercer's condition) — and why that guarantee is exactly what lets SVMs work entirely in terms of k, never computing Φ explicitly (the "kernel trick").

- Clustering (grouping *unlabeled* points by mutual similarity, rather than classifying labeled ones) reuses every distance/similarity function in §2 unchanged — only the algorithm built on top differs.

- Decision trees and boosting (separate notes) are a completely different kind of "local" method — their neighborhoods (tree leaves) are *learned* from labels via greedy splitting, rather than fixed in advance by a distance formula. That comparison is developed in the Boosting notes' own connection-to-k-NN discussion, not repeated here.