
Binary Prediction: Goals and Historical Data

David I. Inouye

Last updated: September 22, 2026

What would justify trusting a prediction for someone we have never seen?

| Different settings ask **the same kind of prediction question**

Setting	Prediction question with two possible answers
Bank phone campaign	Will this customer open a term-deposit account?
Wine screening	Will this wine receive a high sensory rating?
Activity recognition	Is this person walking?

Each uses information available now to predict **an outcome we do not yet know**.

Our running case: the bank phone campaign. A **term-deposit account** holds money for a fixed period to earn interest.

| Each practical question needs **a mathematical counterpart**

Practical question	Bank-case answer	Mathematical counterpart
What information is available?	New customer information known before the call	?
What answer do we want?	Whether the customer opens the offered account	?
What makes a predictor good?	Few mistakes across unseen customers	?

We will fill in one mathematical counterpart at a time.

| Available information becomes **the input** x

Question: What information is available?

Bank-case answer: New customer information known before the call.

Formalization:

$$x \in \mathcal{X}$$

x is one customer's available information; $\mathcal{X}$ is the set of valid input representations.

| The desired answer becomes **the output** y

Question: What answer do we want?

Bank-case answer: Whether the customer opens the offered account.

Formalization:

$$x \xrightarrow{f} \hat{y} \in \{0, 1\}$$

$y \in \{0, 1\}$ is the actual outcome. The prediction is correct when $\hat{y} = y$.

| Population error formalizes **few mistakes on unseen customers**

A **distribution** p describes how common different unseen customer–outcome pairs are. A random pair $(\mathbf{x}, y)$ drawn from p contains the available information and what actually happens.

The **population error** is the chance that our prediction is wrong:

$$R_p(f) = \Pr_{(\mathbf{x}, y) \sim p} (f(\mathbf{x}) \neq y).$$

Goal: choose $f^* \in \arg \min_{f: \mathcal{X} \rightarrow \{0,1\}} R_p(f)$.

This completes the goal—but it does **not** tell us how to find f .

| The prediction contract now has a mathematical form

Practical question	Bank-case answer	Mathematical counterpart
What information is available?	New customer information known before the call	Input $x \in \mathcal{X}$
What answer do we want?	Whether the customer opens the offered account	Outcome $y \in \{0, 1\}$; prediction $\hat{y} = f(x)$
What makes a predictor good?	Few mistakes across unseen customers	Small population error $R_p(f)$

The unresolved question: How do we find the right function f ?

| Machine learning **infers the rule f from examples**

In ordinary computation, the rule is given. For example, a circuit simulator is given equations and parameters, then computes an output from an input.

Ordinary computation

$$f \text{ given, } \quad x \longmapsto f(x)$$

Machine learning

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n \longmapsto \text{infer } f$$

The historical input–outcome pairs $\mathcal{D}$ are the **training data**. They give examples of the mapping, but not the rule itself.

| A dataset column is not automatically an allowed input

Prediction moment: before calling this customer.

Historical columns: recorded age and job · duration of last call · whether the customer opens this account

Can call duration be used to predict whether this customer will open the account before the call begins?

A. Yes, because it appears in the historical training data.

B. Yes, if the historical dataset is sufficiently large.

C. No, because it is unavailable at the prediction moment.

Larger principle: available columns describe a dataset; allowed inputs are determined by the real decision we are trying to support.

| Assumptions mark where a learned predictor can break

Training pairs are the evidence. Performance on unseen pairs is the claim. An explicit assumption must connect them:



The most common simple starting point is **i.i.d. sampling**: model the training pairs and unseen pairs as independent draws from the same distribution p .

I.i.d. is deliberately naive. When it reasonably approximates the problem, it provides a useful foundation for analysis. **We will use it for now.**

| Independence rules out **linked training examples**

The first “I” in **i.i.d.** is **independent**.

Knowing one sampled pair $(\mathbf{x}_i, y_i)$ should not change the probability of another draw once the population distribution is fixed. The unseen case should also be independent of the sample used to construct f .

Possible violation: if the same customer appears after several campaign contacts, those rows and outcomes may be linked rather than independent examples.

| Identically distributed connects training pairs to **unseen cases**

The second “I” in **i.i.d.** is **identically distributed**.

Training pairs and the unseen pairs we care about are modeled as draws from the same joint distribution p —both the inputs and how they relate to the outcome.

Possible violation: if the bank changes who it contacts or changes the offered account, deployment cases may no longer follow the training distribution.

Same distribution does not mean the same customers or repeated feature values.

| The contract and evidence now motivate **a first method**

The contract asks for low error on unseen customers using only information available before the call. Training pairs become evidence under our working assumption that they and the unseen cases are connected.

First idea: find past customers similar to the new customer and consult their outcomes.

Next question: What should “similar” mean, and why should their outcomes help?

2. How can examples produce a predictor?

KNN makes learning from local evidence visible without making the algorithm the point.

| KNN predicts from **nearby observed outcomes**

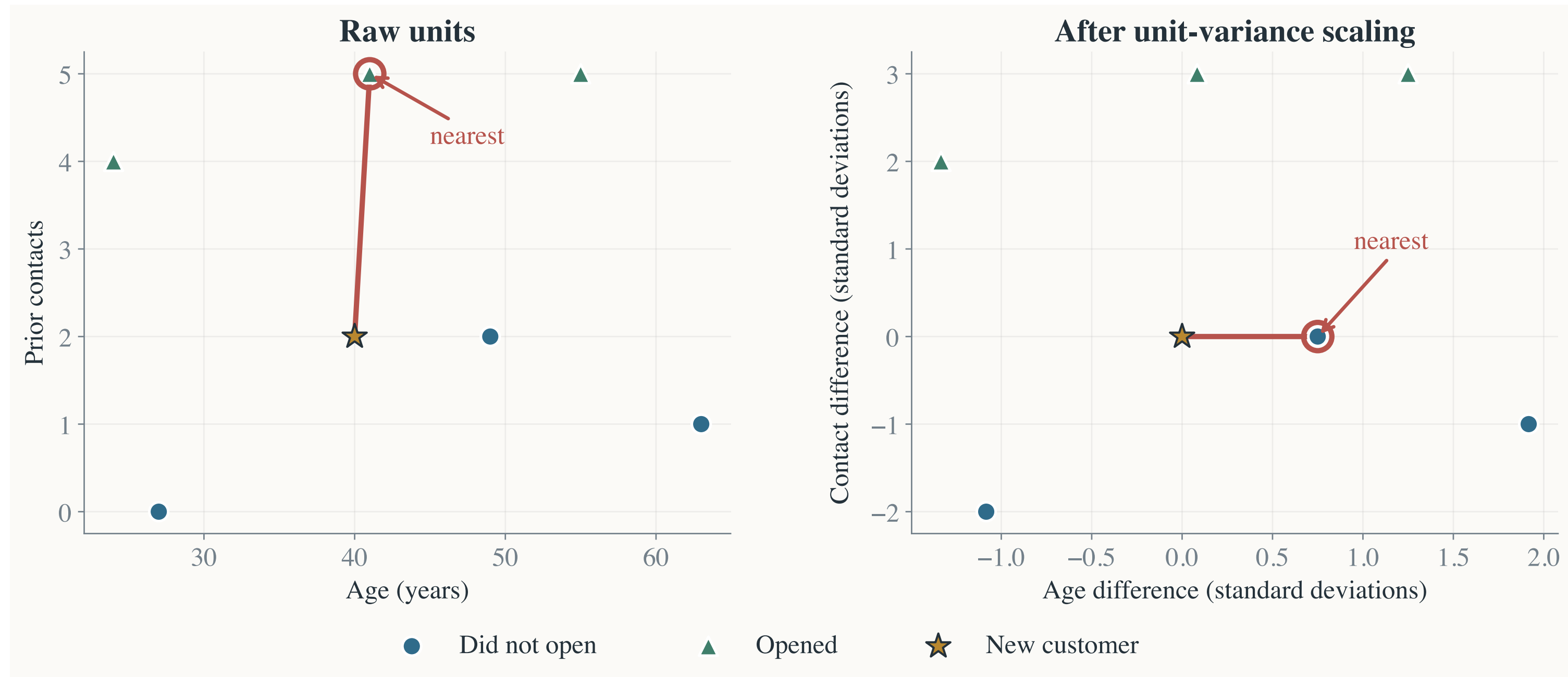
For a new customer $\boldsymbol{x}$:

1. compute the distance to every training input $\boldsymbol{x}_i$;
2. keep the indices $\mathcal{N}_k(\boldsymbol{x})$ of the k nearest;
3. average their observed outcomes and take a majority vote.

$$\hat{\eta}_k(\boldsymbol{x}) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} y_i, \quad \hat{y} = \mathbb{1} \left[\hat{\eta}_k(\boldsymbol{x}) \geq \frac{1}{2} \right].$$

The method constructs a prediction directly from the stored examples.

| Changing the axes can change **who is nearest**



Distance measures differences in the coordinates we supply—not an intrinsic notion of customer similarity.

| Unit-variance scaling is a **simple default metric**

For each feature j , estimate its training mean and standard deviation, then use

$$x'_{ij} = \frac{x_{ij} - \hat{\mu}_j}{\hat{s}_j}.$$

- Apply the same rule to numeric columns and binary indicator columns.
- Encode categorical attributes first—for example, with one indicator per category.
- Estimate $\hat{\mu}_j$ and $\hat{s}_j$ from training data only; remove constant columns.

Interpretation: every retained coordinate has empirical variance one before distance is computed.

| The neighbor fraction estimates a **local outcome probability**

The population quantity is

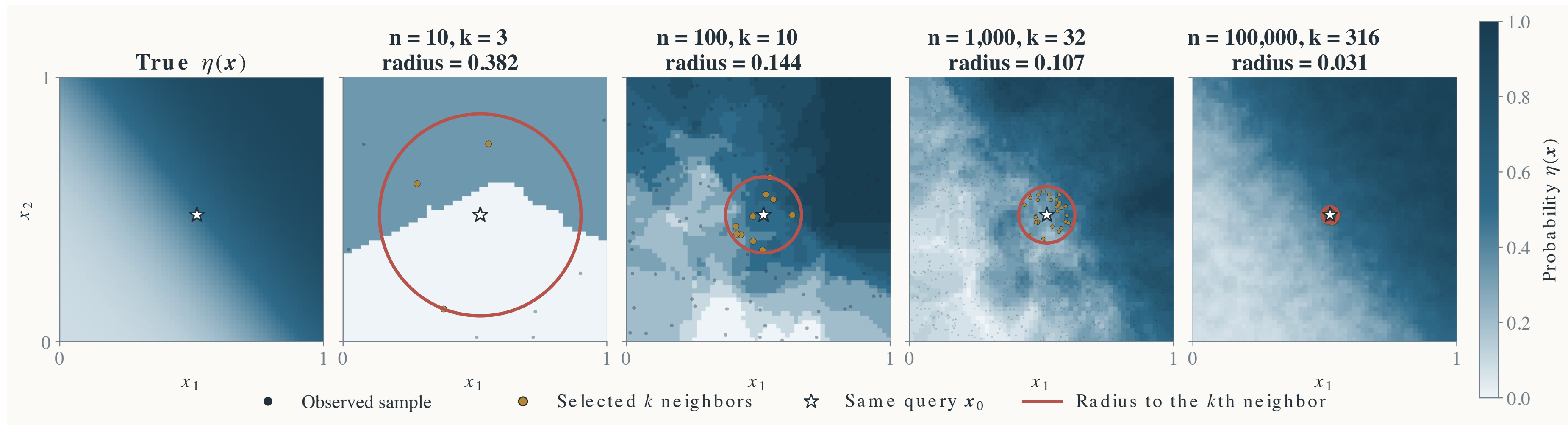
$$\eta(\boldsymbol{x}) = \Pr(y = 1 \mid \mathbf{x} = \boldsymbol{x}).$$

KNN substitutes nearby observed outcomes for repeated outcomes at exactly the same input:

$$\hat{\eta}_k(\boldsymbol{x}) = \frac{\text{nearby customers who opened the account}}{k}.$$

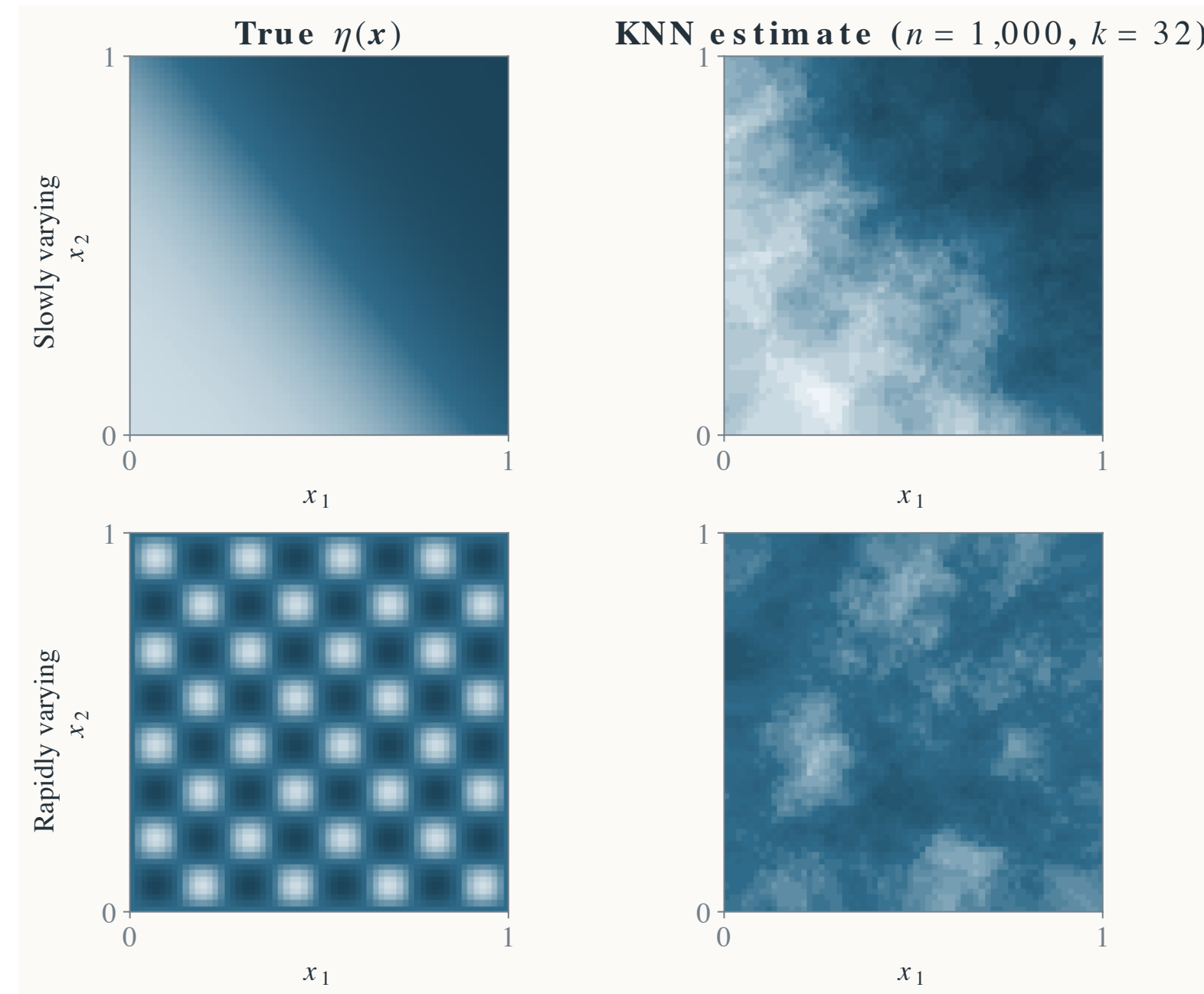
Similar customers need not have identical outcomes; their **probabilities** must be locally related.

| With more data, k can grow while neighborhoods stay **local**



Here $k = \lfloor \sqrt{n} \rfloor$, so $k \rightarrow \infty$ while $k/n \rightarrow 0$: the same query averages more labels inside a smaller neighborhood.

| Smooth local structure is easier to estimate **with finite data**



Local averaging recovers slowly varying probabilities sooner. Fine-scale or choppy structure can remain blurred until the sample is extraordinarily dense.

| KNN defers nearly all computation **until prediction**

```
1 def fit(X, y):
2     return np.asarray(X, float), np.asarray(y)
3
4 def predict_proba(model, X_new, k):
5     X, y = model
6     d2 = ((X_new[:, None, :] - X[None, :, :])**2).sum(2)
7     nbr = np.argpartition(d2, k - 1, axis=1)[:k]
8     return y[nbr].mean(axis=1)
```

Fit

- no parameter optimization
- store $n \times d$ inputs and n labels
- memory and copy: $O(nd)$

Predict one input

- compute all n distances
- partial selection avoids a full sort
- time: $O(nd)$

Small reference sets can be cheap; very large ones motivate models that compress training evidence into learned parameters.

3. Why can fitting the examples mislead us?

Observed success may reflect memorization, sample luck, or the wrong flexibility.

| Training accuracy cannot tell us which k predicts best

Training accuracy on 3,000 distinct records from the bank use case:

k	1	25	151
Training accuracy	100.0%	90.1%	89.7%

Which value of k will give us the best predictions?

- A. $k = 1$, because its training accuracy is highest.
- B. $k = 25$, because it balances local detail and averaging.
- C. $k = 151$, because it averages the most neighbors.

None of them. Training accuracy does not tell us which value predicts **unseen** customers best. We need different evidence.

| Training error answers **an easier question**

Quantity	Examples being scored	Question answered
Training error	Used to construct f	How well does f reproduce evidence it already saw?
Population error $R_p(f)$	Unseen draws from p	How often will f fail on the cases we care about?

The same observations cannot simultaneously serve as unrestricted construction material and untouched evidence of generalization.

| A holdout lets observed data **play the role of unseen data**

We do not have outcomes for the genuinely unseen customers we ultimately care about. How can we estimate their error?



Reserve some observed pairs **before fitting**. Because their outcomes did not help construct f , predictions on them mimic the information boundary of an unseen case—under the same sampling assumptions.

| Cross-validation rotates **which fold is held out**

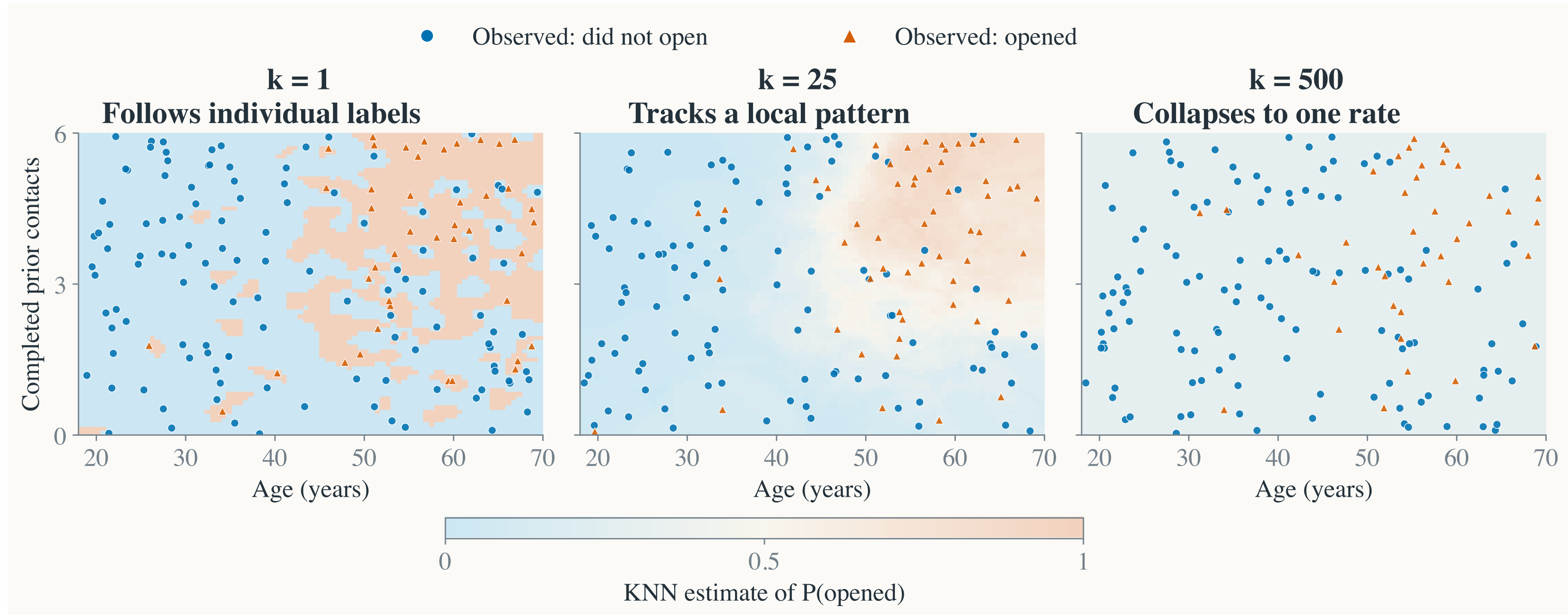
Split one observed dataset into folds. Each fold takes one turn as untouched evaluation data.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Fit 1	score e_1	train	train	train	train
Fit 2	train	score e_2	train	train	train
Fit 3	train	train	score e_3	train	train
Fit 4	train	train	train	score e_4	train
Fit 5	train	train	train	train	score e_5

K fits produce K errors, then we average: $\widehat{R}_{CV} = \frac{1}{K} \sum_{j=1}^K e_j$

Leave-one-out cross-validation is the small-data extreme: $K = n$.

| The neighbor count controls **how much local detail survives**



$k = 1$ can overfit individual outcomes; using every example erases useful local structure. Held-out error can reveal both failures.

| Bias compares the average fitted estimate with **the target relationship**

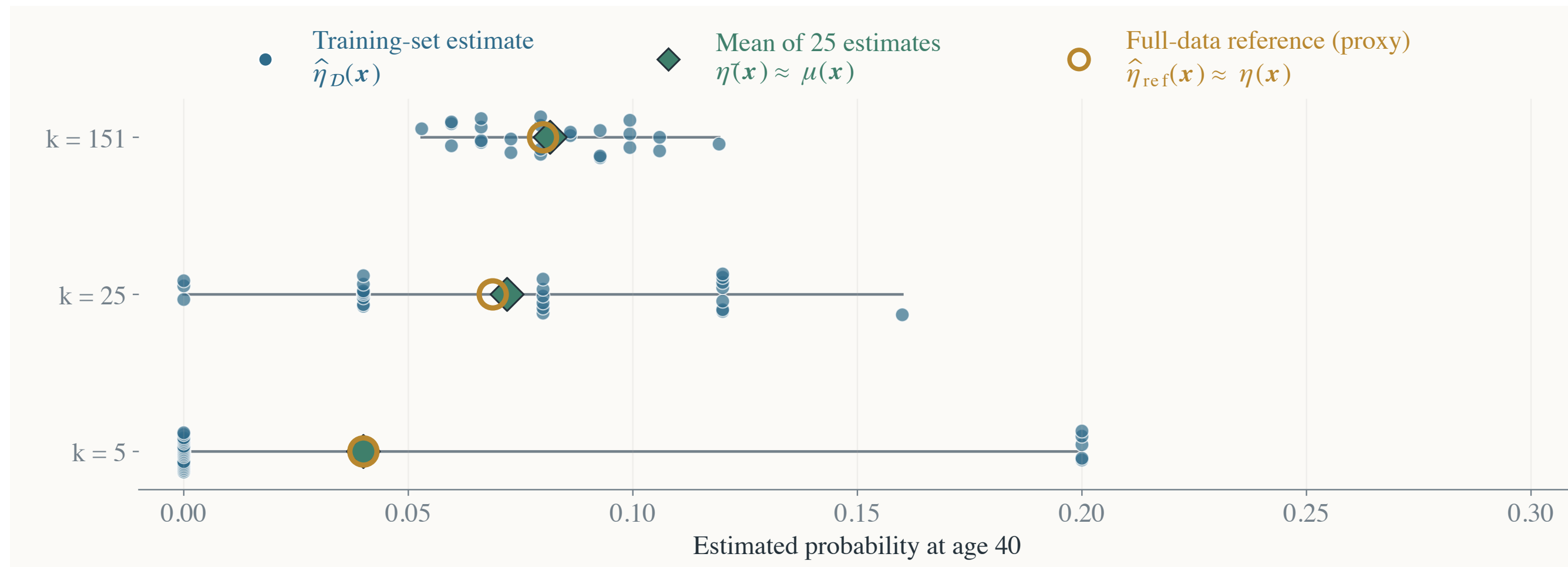
At one fixed test point $\boldsymbol{x}$, imagine refitting the method on many possible training samples. Their average prediction is where the method tends to land:

$$\mu(\boldsymbol{x}) = \mathbb{E}_{\mathcal{D}} [\hat{\eta}_{\mathcal{D}}(\boldsymbol{x})] , \quad \text{Bias}(\boldsymbol{x}) = \mu(\boldsymbol{x}) - \eta(\boldsymbol{x}).$$

Real data lets us observe how estimates vary across samples. It does not reveal the true $\eta(\boldsymbol{x})$, so formal bias remains unknown.

| Bias measures **offset**; variance measures **spread**

At **age 40** (the only feature), each blue dot uses a different disjoint training set $\mathcal{D}$ of $n = 1,647$ or $1,648$ records. Each row reuses the same 25 sets.



$$\text{Bias}(x) = \mu(x) - \eta(x)$$

$$\text{Var}_{\mathcal{D}}(\hat{\eta}_{\mathcal{D}}(x)) = \mathbb{E}_{\mathcal{D}}[(\hat{\eta}_{\mathcal{D}}(x) - \mu(x))^2]$$

True η is unknown: the diamond-to-ring gap is a bias proxy. The reference may itself be biased.

| Expected squared error combines **bias, variance, and label noise**

Draw a new input $\mathbf{x}$ and binary outcome y from the target population, independently of the training set $\mathcal{D}$. How far is the predicted probability from the **actual outcome**, averaged over new cases and refits?

$$\underbrace{\mathbb{E}_{\mathcal{D}, \mathbf{x}, y} [(\hat{\eta}_{\mathcal{D}}(\mathbf{x}) - y)^2]}_{\text{expected squared prediction error}}$$
$$= \mathbb{E}_{\mathbf{x}} \left[\underbrace{\text{Bias}(\mathbf{x})^2}_{\text{systematic mismatch}} + \underbrace{\text{Var}_{\mathcal{D}}(\hat{\eta}_{\mathcal{D}}(\mathbf{x}))}_{\text{sample sensitivity}} + \underbrace{\eta(\mathbf{x})(1 - \eta(\mathbf{x}))}_{\text{unresolved outcome variation}} \right].$$

Squared error loss gives this exact sum. Modeling choices can trade bias against variance; the represented inputs determine the remaining outcome noise.

| Centering the outcome at η separates estimation error and noise

Fix $\mathbf{x}$. Write $\hat{\eta}_{\mathcal{D}} = \hat{\eta}_{\mathcal{D}}(\mathbf{x})$ and $\eta = \mathbb{E}[y \mid \mathbf{x}]$. The new outcome is independent of the training set given this input.

$$\begin{aligned} & \mathbb{E}_{\mathcal{D}, y}[(\hat{\eta}_{\mathcal{D}} - y)^2 \mid \mathbf{x}] \\ &= \mathbb{E}_{\mathcal{D}, y}[(\hat{\eta}_{\mathcal{D}} - \eta) + (\eta - y)]^2 \mid \mathbf{x}] && \text{(Add and subtract } \eta) \\ &= \mathbb{E}_{\mathcal{D}}[(\hat{\eta}_{\mathcal{D}} - \eta)^2] + \mathbb{E}_y[(\eta - y)^2 \mid \mathbf{x}] \\ &\quad + 2\mathbb{E}_{\mathcal{D}}[\hat{\eta}_{\mathcal{D}} - \eta] \underbrace{\mathbb{E}_y[\eta - y \mid \mathbf{x}]}_{\eta - \mathbb{E}[y \mid \mathbf{x}] = \eta - \eta = 0} && \text{(Expand; independence factors the cross term, and centering makes it zero)} \\ &= \mathbb{E}_{\mathcal{D}}[(\hat{\eta}_{\mathcal{D}} - \eta)^2] + \underbrace{\mathbb{E}[y^2 \mid \mathbf{x}] - \eta^2}_{\text{Var}(y \mid \mathbf{x})} && \text{(Expand the remaining outcome square)} \\ &= \underbrace{\mathbb{E}_{\mathcal{D}}[(\hat{\eta}_{\mathcal{D}} - \eta)^2]}_{\text{probability-estimation error}} + \underbrace{\eta(1 - \eta)}_{\text{outcome noise}} && \text{(Binary } y^2 = y, \text{ so } \eta - \eta^2 = \eta(1 - \eta)) \end{aligned}$$

| Centering the fit at μ separates **squared bias and variance**

At the same fixed x , let $\mu = \mathbb{E}_{\mathcal{D}}[\hat{\eta}_{\mathcal{D}}]$. Continue from the preceding slide, keeping its outcome-noise term.

$$\begin{aligned} & \mathbb{E}_{\mathcal{D}}[(\hat{\eta}_{\mathcal{D}} - \eta)^2] + \eta(1 - \eta) \\ &= \mathbb{E}_{\mathcal{D}}[((\hat{\eta}_{\mathcal{D}} - \mu) + (\mu - \eta))^2] + \eta(1 - \eta) \\ &= \mathbb{E}_{\mathcal{D}}[(\hat{\eta}_{\mathcal{D}} - \mu)^2] + (\mu - \eta)^2 + \eta(1 - \eta) \\ &\quad + 2(\mu - \eta) \underbrace{\mathbb{E}_{\mathcal{D}}[\hat{\eta}_{\mathcal{D}} - \mu]}_{\mathbb{E}_{\mathcal{D}}[\hat{\eta}_{\mathcal{D}}] - \mu = \mu - \mu = 0} \\ &= \underbrace{(\mu - \eta)^2}_{\text{squared bias}} + \underbrace{\mathbb{E}_{\mathcal{D}}[(\hat{\eta}_{\mathcal{D}} - \mu)^2]}_{\text{variance}} + \underbrace{\eta(1 - \eta)}_{\text{outcome noise}} \end{aligned}$$

(Add and subtract μ)

(Expand; $\mu - \eta$ is constant across training sets, and the centered fit has mean zero)

(The cross term is zero; reorder the remaining terms)

This exact sum uses **squared error loss**. Other losses need different analyses.

| Averaging over inputs gives **population squared prediction error**

Now draw a new input $\mathbf{x}$ from the target population and its outcome y . Average the pointwise identity over those inputs:

$$\underbrace{\mathbb{E}_{\mathcal{D}, \mathbf{x}, y}[(\hat{\eta}_{\mathcal{D}}(\mathbf{x}) - y)^2]}_{\text{average error on new cases, across refits}} \\ = \underbrace{\mathbb{E}_{\mathbf{x}}[\text{Bias}(\mathbf{x})^2]}_{\text{average squared bias}} + \underbrace{\mathbb{E}_{\mathbf{x}}[\text{Var}_{\mathcal{D}}(\hat{\eta}_{\mathcal{D}}(\mathbf{x}))]}_{\text{average variance}} + \underbrace{\mathbb{E}_{\mathbf{x}}[\eta(\mathbf{x})(1 - \eta(\mathbf{x}))]}_{\text{average outcome noise}}.$$

- Inputs count according to how often they occur in the **target population**. The decomposition stays exact, but its terms now summarize all inputs.
- Average **squared** bias: positive and negative biases at different inputs cannot cancel each other.

Why it matters: smoothing can reduce sample sensitivity while increasing systematic mismatch. Held-out error measures their combined effect, plus outcome noise.

| Increasing k exchanges local detail for stability

Smaller k	Larger k
uses closer evidence	reaches farther from the query
follows more sample-specific detail	averages more observed outcomes
often lower local bias, higher variance	often higher local bias, lower variance

Neither extreme is automatically best. Held-out error measures the combined effect for the population and representation we care about.

| Data and representation improve **the same local-evidence mechanism**

Change	What happens to a KNN neighborhood?
More representative training data	genuinely similar examples can be closer
Better scaling or features	distance favors differences relevant to the outcome
Larger k	more outcomes are averaged, but evidence comes from farther away

All three change the evidence inside $\hat{\eta}_k(\mathbf{x})$ and therefore its bias and variance.

4. How can we choose a model without fooling ourselves?

Choosing a predictor and measuring its performance are two different uses of data.

| Which neighbor count should we choose?

We fit KNN with $k = 1, 5, 25$ and choose the lowest **training error**.
Inputs are distinct, and each training point counts as its own neighbor.
Is this a reliable way to choose the best predictor for new cases?
A. Yes: evaluating every candidate on the same examples makes the comparison fair.
B. No: training error rewards reproducing outcomes the predictor already used.
C. No: we should always choose the largest k to get the most stable predictions.

B. Here, $k = 1$ gets zero training error by retrieving each point's own label. That does not establish how well it predicts a new outcome.

| A held-out **validation set** lets us compare candidate values of k

Fit each candidate using $\mathcal{D}_{\text{train}}$. Evaluate its predictions on the same separate $\mathcal{D}_{\text{validation}}$, then choose the lowest validation error.

$$\mathcal{D}_{\text{train}} \xrightarrow{\text{fit}} f_1, f_5, f_{25} \xrightarrow{\text{score on validation}} \hat{k} = \arg \min_{k \in \{1, 5, 25\}} \widehat{R}_{\text{validation}}(f_k).$$

The validation outcomes are unavailable when fitting each candidate. They give us evidence about prediction on cases outside that candidate's training set.

| What does the selected model's 4.5% error tell us?

We choose k with the lowest validation error and report its 4.5% as the selected model's estimated population error.

Across repetitions of this procedure, is the reported error lower on average, unbiased, or higher on average?

- A. Lower than the selected model's true population error on average.
- B. Unbiased for the selected model's true population error.
- C. Higher than the selected model's true population error on average.

A: optimistic selection bias. Choosing the lowest observed error also favors favorable validation noise. Validation appropriately helps choose k , but the winning score is no longer an unbiased final evaluation.

| Validation is held out of each fit, but participates in selection

What are we evaluating?

One candidate fitted at a fixed k

The selected pipeline, including the choice of k

Did validation outcomes help construct it?

No: only the training data determined that fit.

Yes: validation errors determined which candidate won.

Selection is part of learning. Information from validation enters through our choice of k , even though the KNN fitting step never reads those labels.

The same reuse occurs when validation guides features, preprocessing, or model families. It leaks into a claimed final evaluation through those choices.

| Separate validation for choosing from test data for reporting

Reserve all three roles **before development**:

Data	Role	What may it influence?
Training	Fit each candidate.	Its learned parameters or stored examples.
Validation	Compare candidates and choose k .	The selected pipeline.
Test	Score the settled pipeline once.	The reported performance estimate.

Freeze the pipeline before using the test outcomes. An independent, representative test set gives an **unbiased error estimate for that fixed pipeline**, with finite-sample uncertainty.

| Refit the chosen pipeline **before opening the test set**

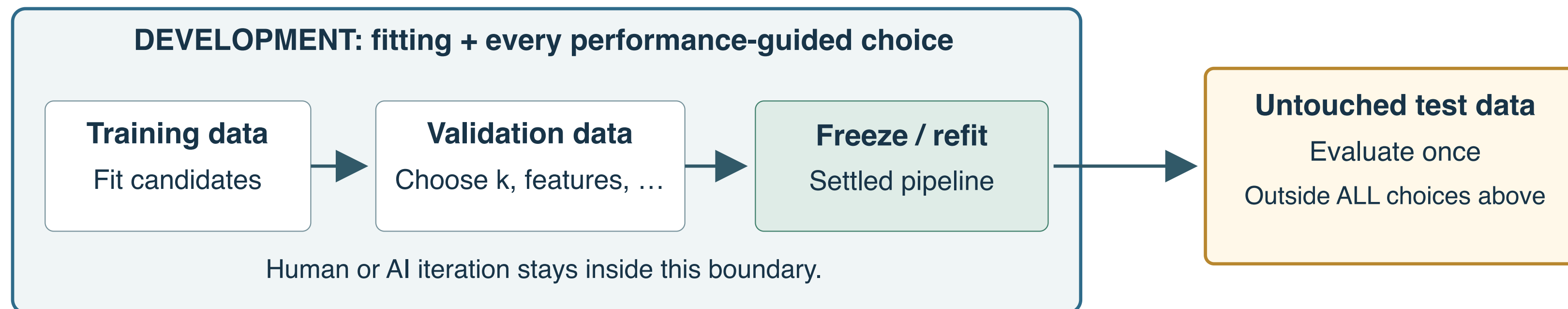
After choosing k and all other pipeline settings, we may fit once more on **training + validation data** to use all development examples.

$$\underbrace{\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{validation}}}_{\text{refit with settled choices}} \longrightarrow f_{\text{final}} \xrightarrow{\mathcal{D}_{\text{test}}} \widehat{R}_{\text{test}}(f_{\text{final}}).$$

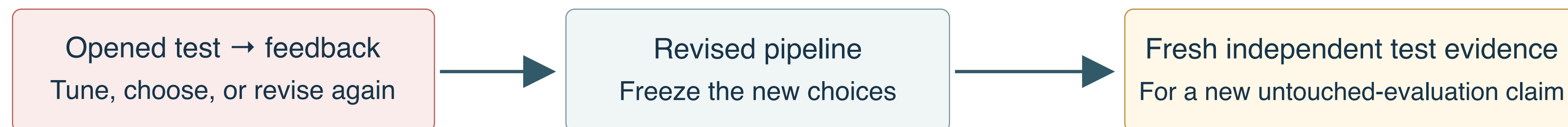
For KNN, this means storing all development examples with the chosen k and refitting any learned preprocessing on those development examples.

The test result describes this final predictor. Changing it after seeing the test result turns that test into more development evidence.

| Keep final testing outside **every development feedback loop**



If test results guide another change, the old test joins development.



One final test can cover all development choices. A new split is needed when we reuse final-test feedback to make further choices—not for every fitting step.

5. When and why should a learned solution hold?

A test score becomes useful when we understand its uncertainty and the scope of its evidence.

| Interpret the score through **three connected questions**

Our reasoning process	Question to resolve
1. What did the test measure? ← now	How much evidence is behind this score?
2. Does it represent deployment?	Could dependence, population, or available information differ?
3. What can we claim?	State the supported result and the conditions for extending it.

We protected the test from selection. Now interpret what it tells us.

| A rough 95% interval is **the score $\pm$ two standard errors**

Fix the model before testing on m independent examples from the target population. Let $\hat{a}$ be the fraction of test predictions that are correct.

$$\text{Approximate 95\% interval: } \hat{a} \pm 2\sqrt{\frac{\hat{a}(1 - \hat{a})}{m}}.$$

900 correct out of 1,000: 90% accuracy, with a margin of about **1.9 percentage points**.

“With approximately 95% confidence, this fixed model’s accuracy in this population is between **88% and 92%**.”

| More test examples support more precise claims

Suppose measured accuracy is **90%** in each case:

Independent test examples	Correct predictions	Rough 95% margin (pp)
10	9/10 = 90%	±19†
30	27/30 = 90%	±11†
100	90/100 = 90%	±6
1,000	900/1,000 = 90%	±1.9
2,000	1,800/2,000 = 90%	±1.3
5,000	4,500/5,000 = 90%	±0.8
10,000	9,000/10,000 = 90%	±0.6

10–30: wide uncertainty. **1,000–10,000:** roughly 2-point to sub-1-point precision here.

pp = percentage points. Margins use the previous slide’s rough formula. †Small-sample scale only: use Wilson for an actual interval. Fixed model;

| Next ask **whether the evidence matches deployment**

Our reasoning process	What we have established
1. What did the test measure?	A count of errors from a finite sample.
2. Does it represent deployment? ← now	Check independence, population, and available information.
3. What can we claim?	Combine the measured result with the scope of its evidence.

A very large test set can measure the wrong deployment population very precisely.

| Repeated rows need not provide **independent evidence**

Imagine 100 customers, each represented by 10 related campaign records.

- A random row split may put the **same customer** in both training and test data.
- A model may exploit person-specific information that will not help on a new customer.
- The 1,000 rows also need not provide the precision of 1,000 independent customers.

If the claim concerns new customers, keep each customer in one split.

| Independent examples can still come from **a different population**

$$R_p(f) = \Pr_{(\mathbf{x}, y) \sim p} \{f(\mathbf{x}) \neq y\}, \quad R_q(f) = \Pr_{(\mathbf{x}, y) \sim q} \{f(\mathbf{x}) \neq y\}.$$

- The test set estimates performance under p .
- Deployment may draw customers and outcomes from $q \neq p$: distribution shift.
- **Out-of-distribution (OOD) performance** depends on q ; even perfect knowledge of $R_p(f)$ does not determine $R_q(f)$.

What changes operationally—and could it change who responds to our offer?

| Would this new contact list preserve accuracy?

Synthetic bank campaign: a frozen age-only KNN model is 90% accurate under the original list. A new supplier provides the **same age distribution** and **same overall account-opening rate**.

Do those checks establish similar accuracy on the new list?

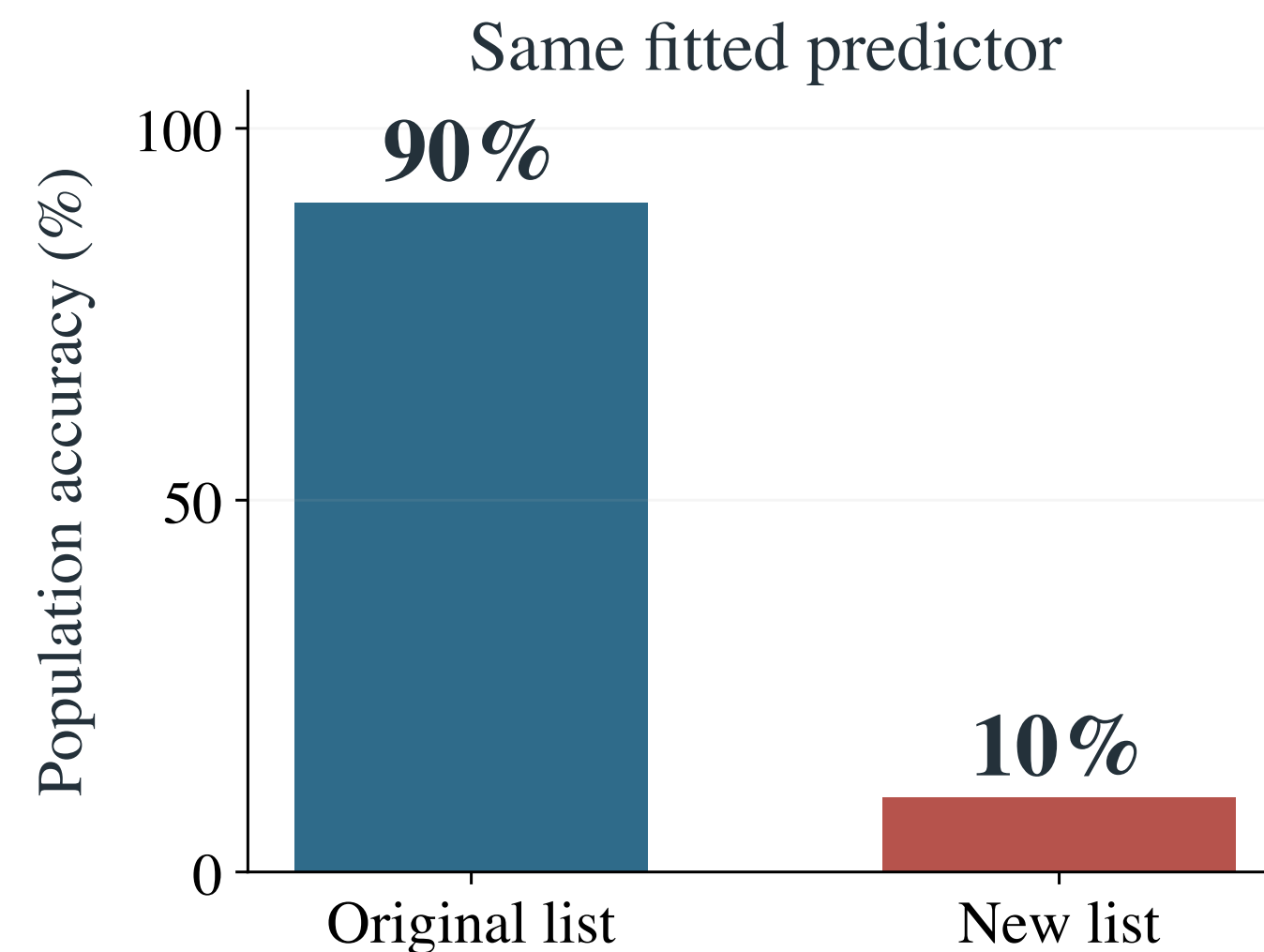
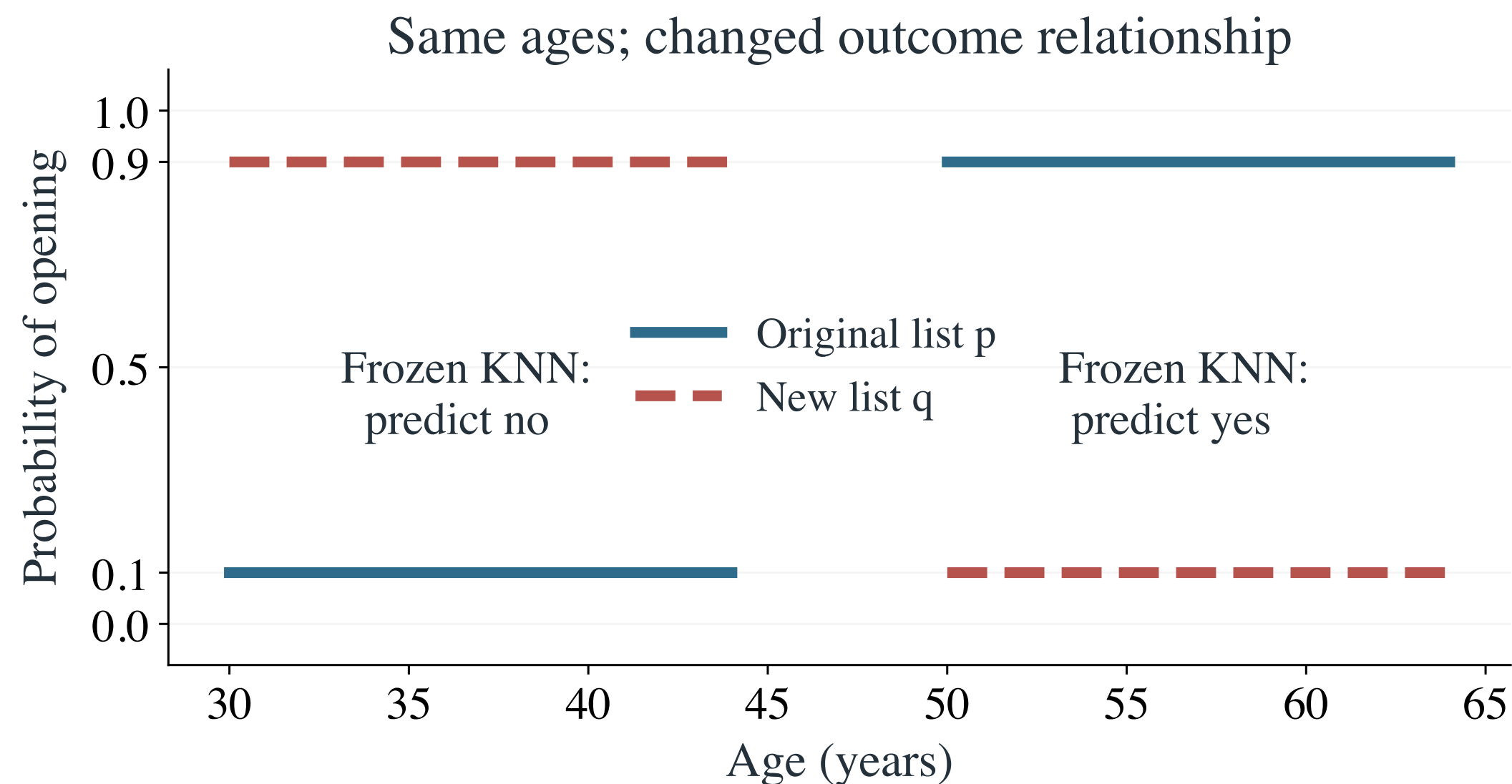
A. Yes: the input distribution and overall outcome rate are unchanged.

B. No: the relationship between age and opening may have changed.

C. Yes, if we make the original-list test set much larger.

B. Matching the separate summaries does not establish a matching joint relationship.

| The unchanged predictor falls from 90% to 10% accuracy



Synthetic bank example. Both lists: half ages 30–44, half ages 50–64; uniform ages within each group; **50% open overall.** Only the relationship between age and outcome changes.

The old 90% remains valid for the old population. More old-list data cannot establish new-list accuracy.

| Which test matches the claim about new people?

A walking classifier uses many sensor windows from each volunteer. Deployment will involve **people absent from development data**.

Which split most directly tests that claim?

- A. Randomly assign windows to training and test sets.
- B. Put each volunteer entirely in development or entirely in test.
- C. Test later windows from volunteers already used for training.

B. It tests unseen people. It still does not establish transfer to different sensors, activities, or collection settings.

| Choose the evaluation design **from the deployment claim**

Intended claim	More relevant evidence	What still needs reasoning?
More cases from the same population	Random held-out examples	Does the sample represent deployment?
Future campaigns	Train earlier, test on a later campaign	Will later changes resemble this one?
New customers	Keep each customer entirely in one split	Are these customers representative?

All designs require **prediction-time inputs**: current-call duration remains unavailable before the call. A split cannot repair that information mismatch.

| Finally state **what the evidence supports**

Our reasoning process	What we have established
1. What did the test measure?	Report the errors and number of test examples.
2. Does it represent deployment?	Examine dependence, changed populations, and available inputs.
3. What can we claim? ← now	State the supported result and the conditions for extending it.

A useful number comes with a reasoned account of where it applies.

| Report a result, its scope, and its uncertainty

Illustrative report: “The frozen before-call predictor made 100 errors on 1,000 independent held-out customers from supplier A’s campaign: **10% error.**”

- **Scope:** one prediction per customer; account opening as the outcome; test data excluded from all development choices.
- **Uncertainty:** roughly **10% $\pm$ 1.9 percentage points** at 95% confidence.
- **Extension:** a different supplier may change response patterns within age groups. This result does not establish its error without evidence connecting the populations.

Say what may change, why it matters, and what evidence would resolve it.

| From interpreting evidence to **rethinking the predictor**

So far: What does a settled model's test score tell us about deployment?

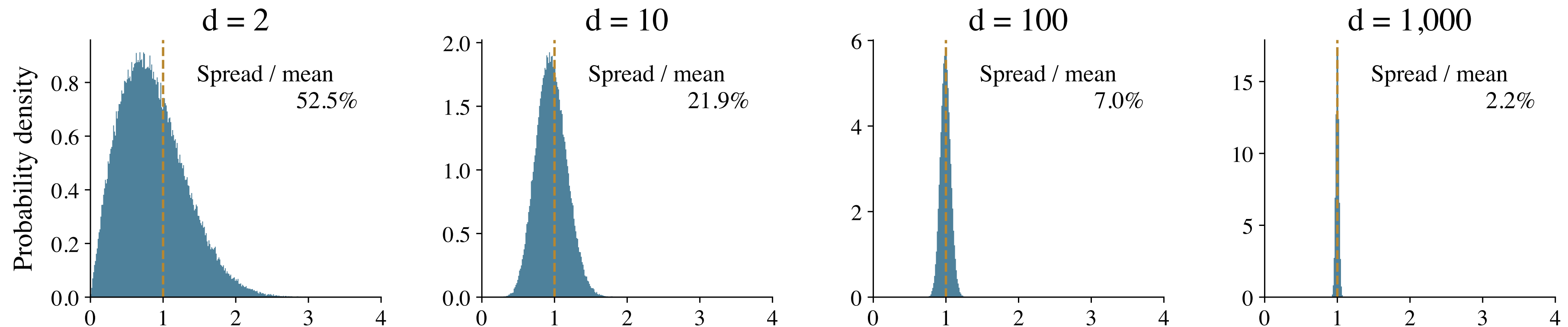
Now: When does KNN's notion of "nearby" stop being useful—and what should we learn instead?

Our next steps:

1. See how near and far distances become less distinct in high dimensions.
2. Test what happens when irrelevant features distort distance.
3. Motivate a learned score that emphasizes predictive information.

| In high dimensions, **near and far become less distinct**

Draw **500 points** with independent standard-normal coordinates: $\mathbf{x}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$. Plot every pairwise distance.



Euclidean distance / $\sqrt{2d}$ (same horizontal scale in every panel)

Typical raw distance grows like $\sqrt{2d}$; its relative spread shrinks. Dividing by $\sqrt{2d}$ lets us compare dimensions on the same scale.

A “near” point is only slightly closer than a “far” point: **distance loses contrast.**

| Add measurements that contain **no new information**

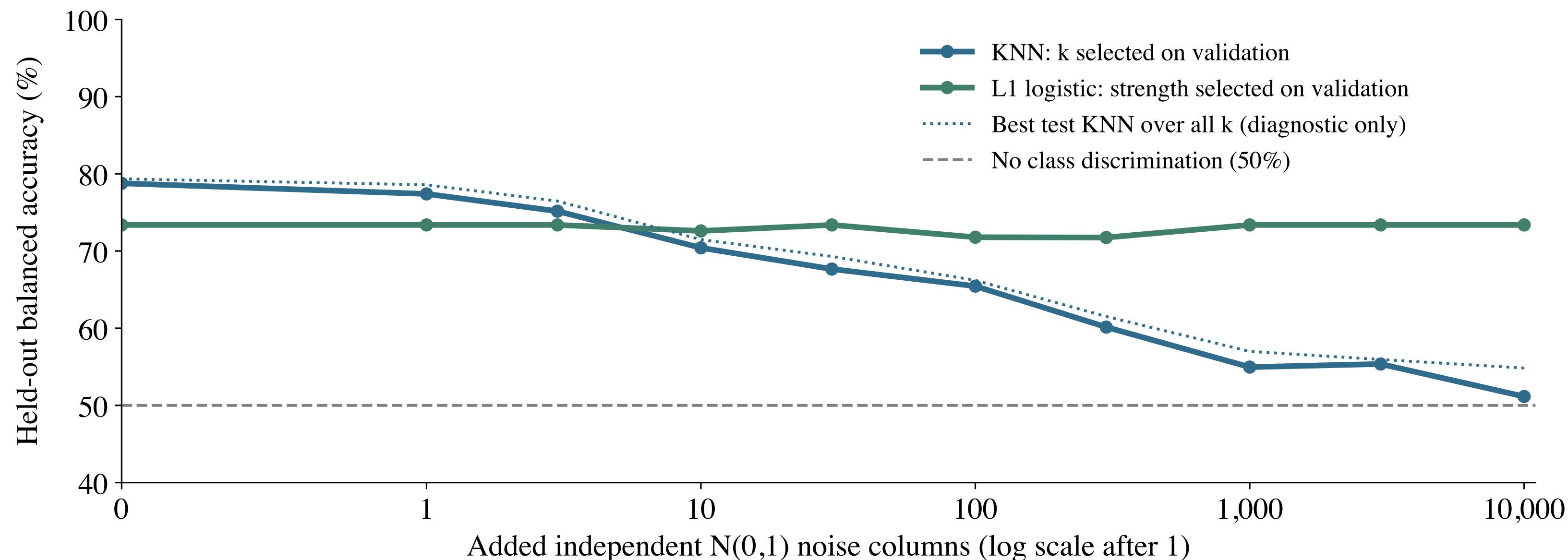
Return to our **simulated bank campaign**: age and prior contacts predict opening. Keep every customer and outcome fixed; append independent Gaussian columns.

$$\mathbf{x}_{\text{aug}} = [\text{age}, \text{prior contacts}, z_1, \dots, z_d], \quad z_j \stackrel{\text{iid}}{\sim} \mathcal{N}(0, 1).$$

- Noise is independent of both the original features and the outcome.
- Standardize using training data; select k or regularization on validation.
- **Same split**: 500 training, 500 validation, 2,000 test customers.

Available predictive information is unchanged. Is distance still useful?

| Independent noise can make neighbors **nearly uninformative**



Balanced accuracy: average of the two class recall rates; **50% means no discrimination.** At 10,000 noise columns: **KNN 51.1%; learned-score model 73.4%** (preview of Arc 4).

| Noise dominates distance; **which coordinates should matter?**

$$\|\mathbf{x}_{\text{aug}} - \mathbf{x}'_{\text{aug}}\|^2 = \underbrace{\|\mathbf{x}_{\text{signal}} - \mathbf{x}'_{\text{signal}}\|^2}_{\text{two relevant coordinates}} + \underbrace{\sum_{j=1}^d (z_j - z'_j)^2}_{\text{many irrelevant coordinates}} .$$

- Independent noise differences accumulate and scramble which customers are nearest.
- Increasing k averages more labels; it does not identify the relevant coordinates.
- Could we **learn which coordinates matter**, giving irrelevant ones less influence?

High dimension alone is not the verdict: correlated features can carry redundant, useful structure. Real data need not fill the whole ambient space.

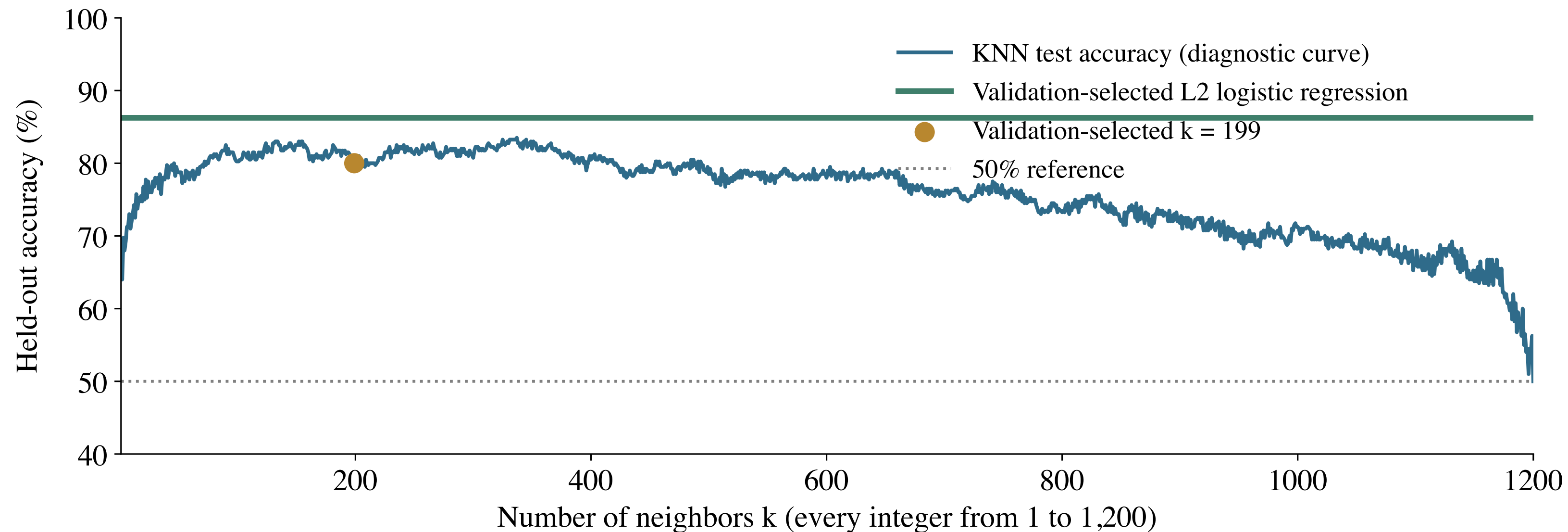
| Real-world example: **movie-review sentiment**

- **2,000 real reviews:** predict positive or negative sentiment.
- **18,746 vocabulary words:** represent each review with one numerical feature per word.
- Shared words need not mean shared sentiment: **“good” versus “not good.”**

A new idea: learn a positive or negative score for each word and combine the contributions. This leads to a **linear model**, which we will study next.

| On real reviews, a learned score **outperforms every tested k**

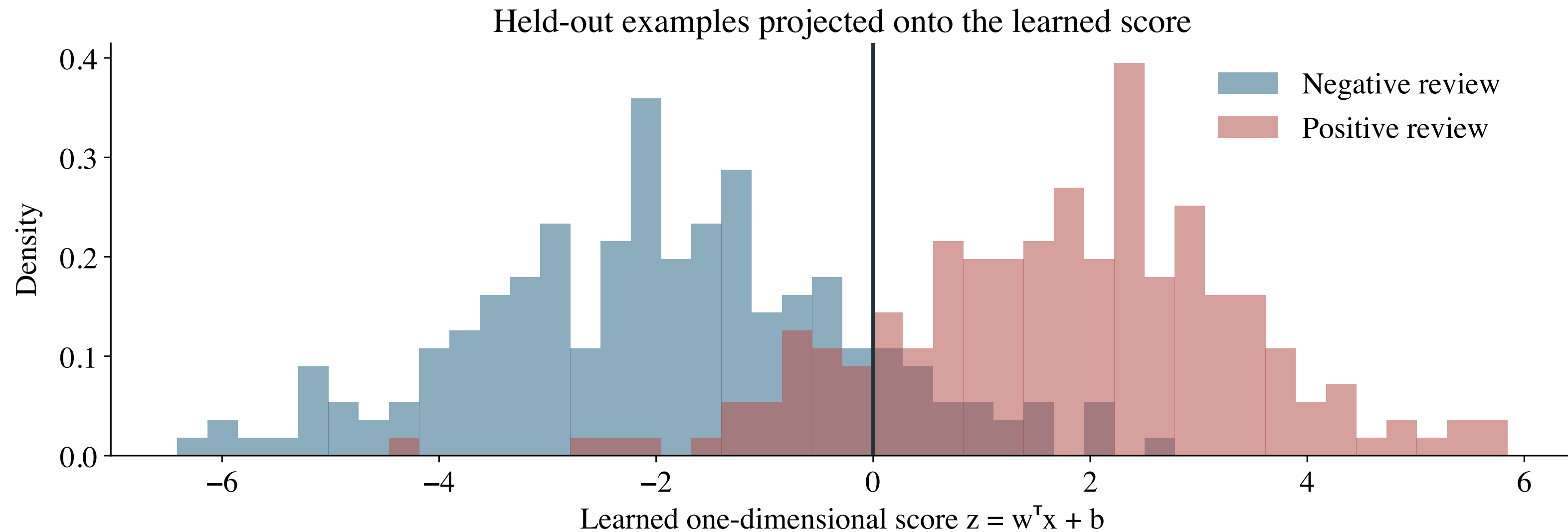
Cornell movie reviews: positive versus negative sentiment; **18,746 word features**. Train: 1,200 reviews; validate: 400; test: 400. No synthetic examples or added noise.



Validation-selected test accuracy: **KNN 80.0%; learned word-score model 86.25%**. Meaningful word features are not independent noise: **KNN still has useful predictive signal.**

| A useful one-dimensional projection **separates the outcomes**

$$x \in \mathbb{R}^{18746} \longrightarrow z = w^T x + b \in \mathbb{R} \longrightarrow \hat{y} = \mathbf{1}\{z \geq 0\}.$$



The weights are **learned from training reviews**; these distributions show held-out reviews.

| Next: learn the predictive direction from labels

KNN asks which stored examples are nearby. A **linear model** learns how much each input contributes to one predictive score.

PCA preserves **input variation**; our goal is to separate **outcomes**.

Arc 4: How do labels tell us which weights to learn—and what can a linear score miss?