k - Nearest Neighbors

Sujay Sanghavi

k-NN Classification

One of the simplest and essentially model-free ideas for classification.

Suppose we are given n training samples $(x^{(i)}, y^{(i)})$, and a new *query* feature vector x that we want to label.

- Find the set $\mathcal{S}_k(x)$ of **nearest neighbors** of x from your training samples.
  - Need to choose a **distance metric** $d(\cdot, \cdot)$ between any two feature vectors
  - Need to choose k, the **number of nearest neighbors** considered.
- Predict $\widehat{y}(x)$ to be the label with the biggest representation in the training samples of $\mathcal{S}_k(x)$

k-NN Classification

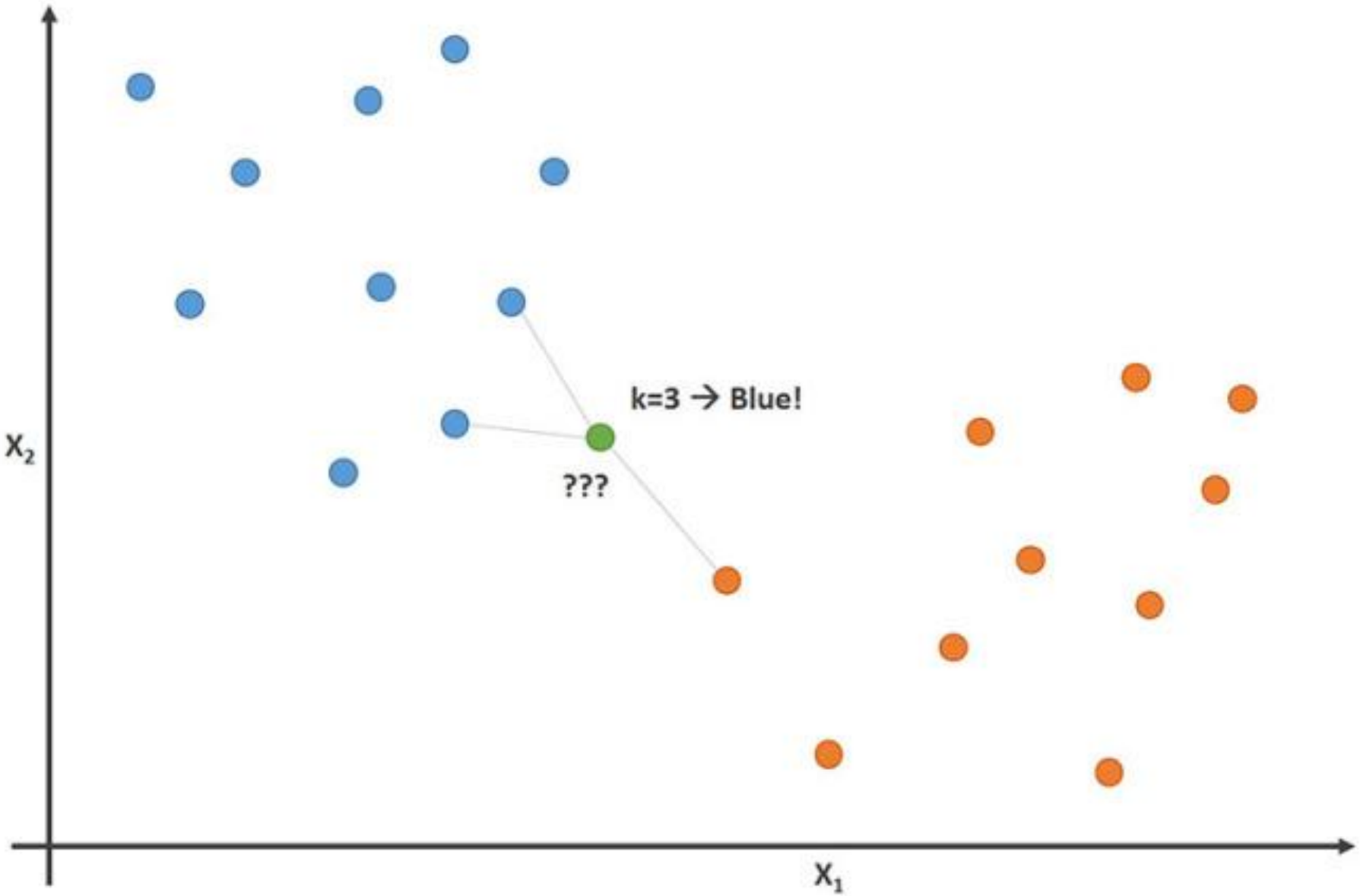
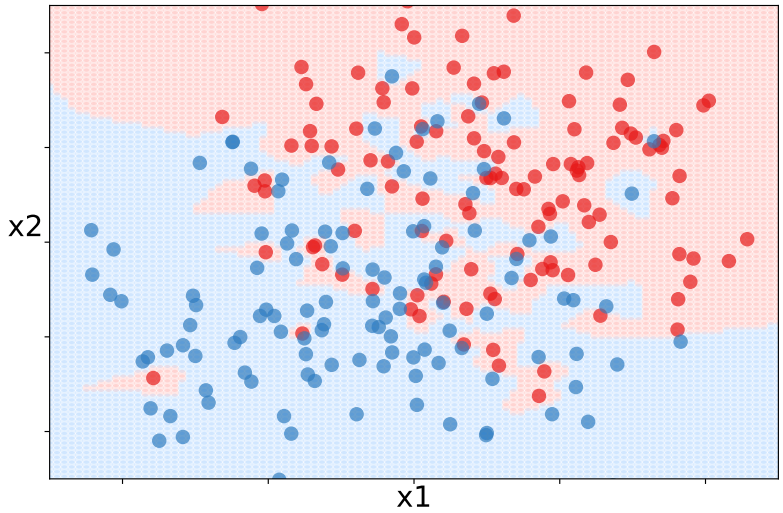


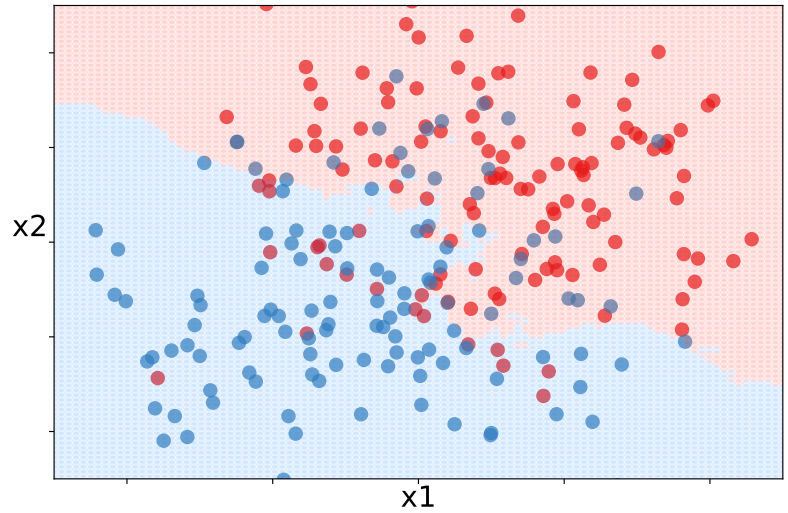
Image credit: kdnuggets

k-NN Classification

$k = 1$



$k = 15$



$k \uparrow$: Smoother boundaries, less overfitting, more underfitting

$k \downarrow$: Sharper / finer boundaries, more overfitting

k-NN vs Logistic

k-NN

Logistic Regression

- Needs to learn $\widehat{\beta}$ from data
  - computation before there is any query
- Assumes data comes from linear model
  - more non-linear the boundary, worse performance
  - model complexity does not change with more training samples
- Very fast inference: $x^\top \widehat{\beta} \gtrless \tau$
- Interpretability: reveals which features are important

k-NN vs Logistic

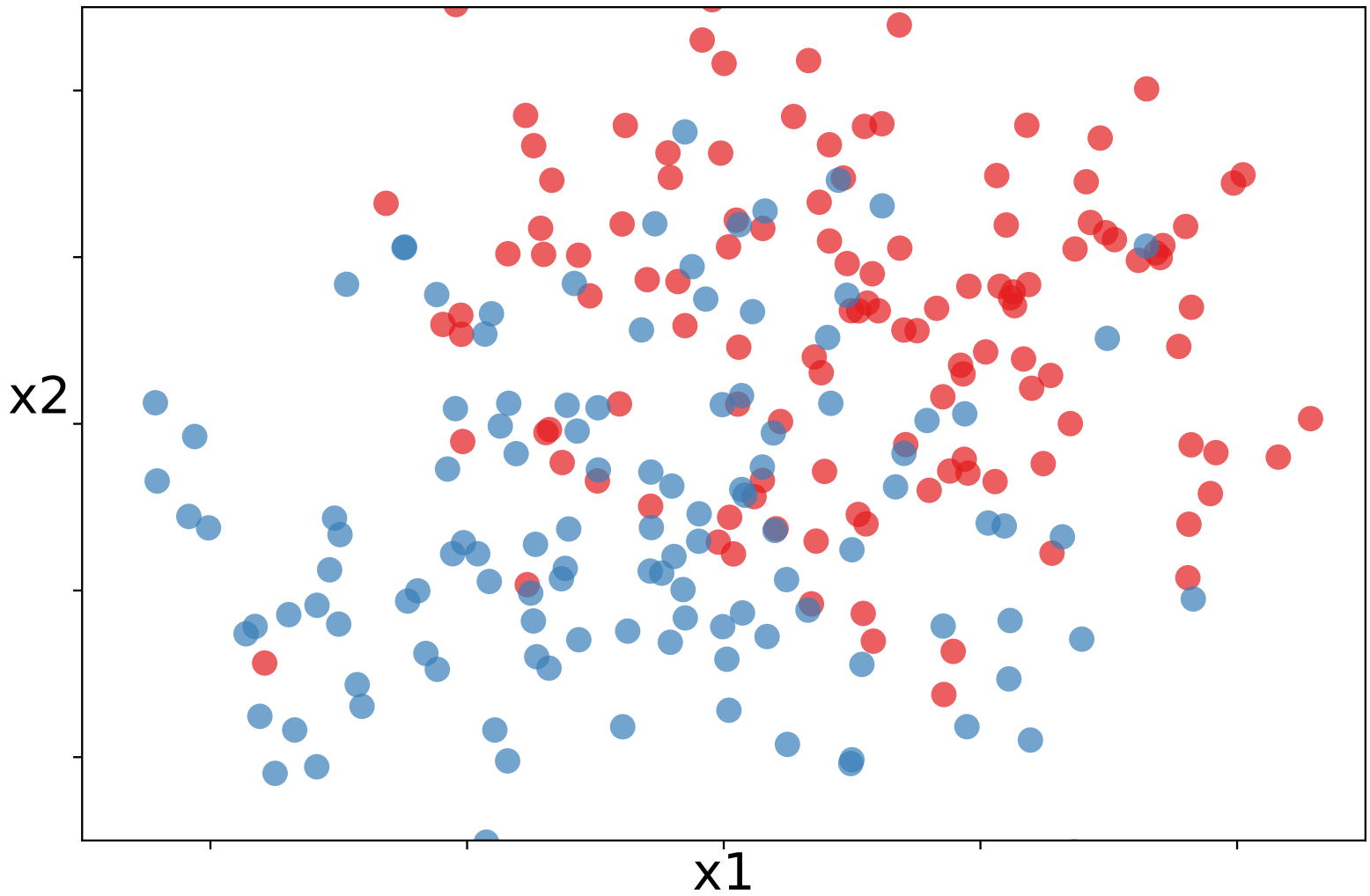
Logistic Regression

- Needs to learn $\widehat{\beta}$ from data
  - computation before there is any query
- Assumes data comes from linear model
  - more non-linear the boundary, worse performance
  - model complexity does not change with more training samples
- Very fast inference: $x^\top \widehat{\beta} \gtrless \tau$
- Interpretability: reveals which features are important

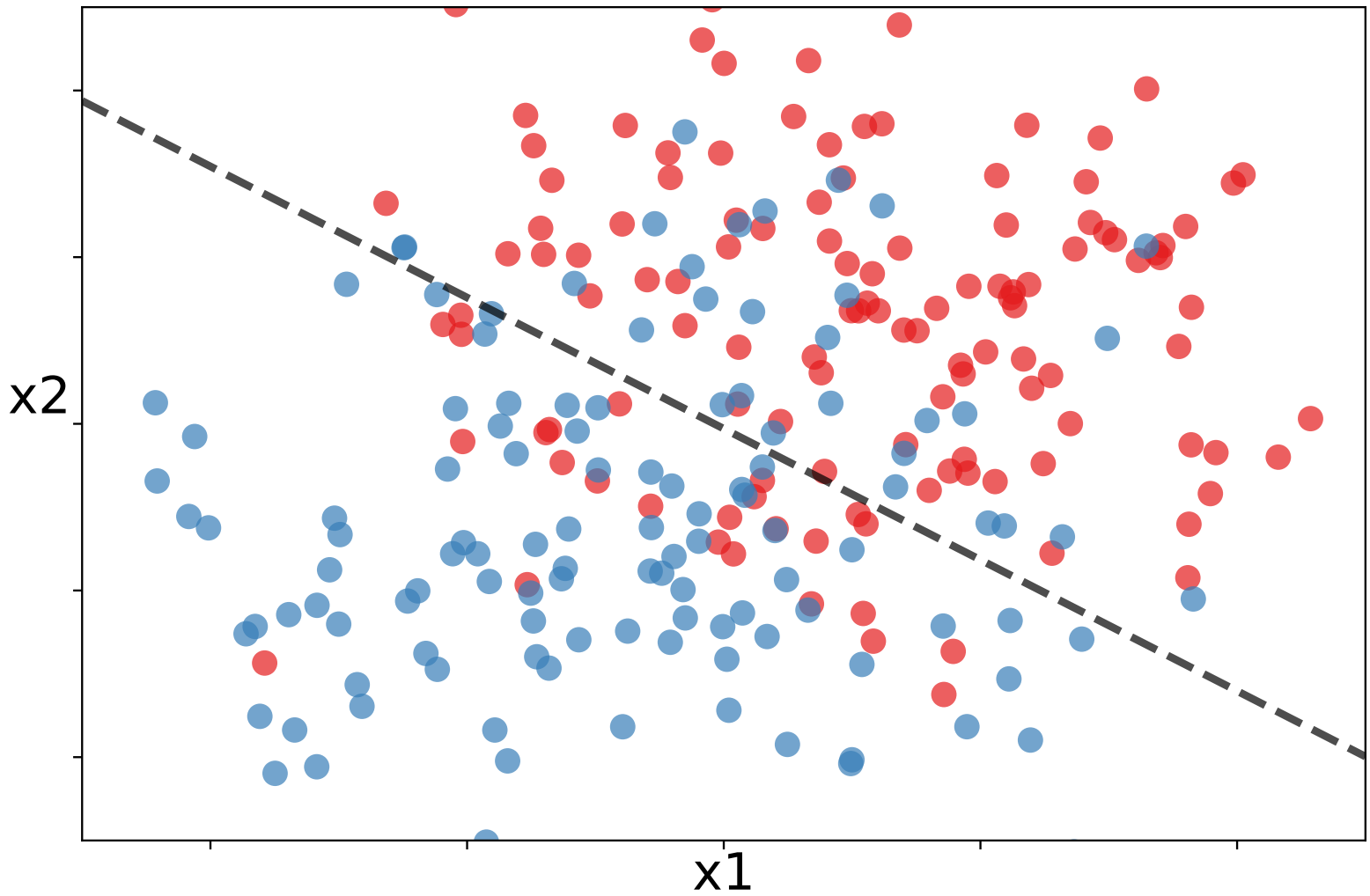
k-NN

- No learning a-priori
  - Computation only when there is a query
- No model assumption
  - only that nearby points more likely to have similar labels
  - decision boundary complexity can increase with more training samples
- Slow inference: for every query, need to traverse entire training data
- Interpretability: reveals which training samples "caused" a particular label decision

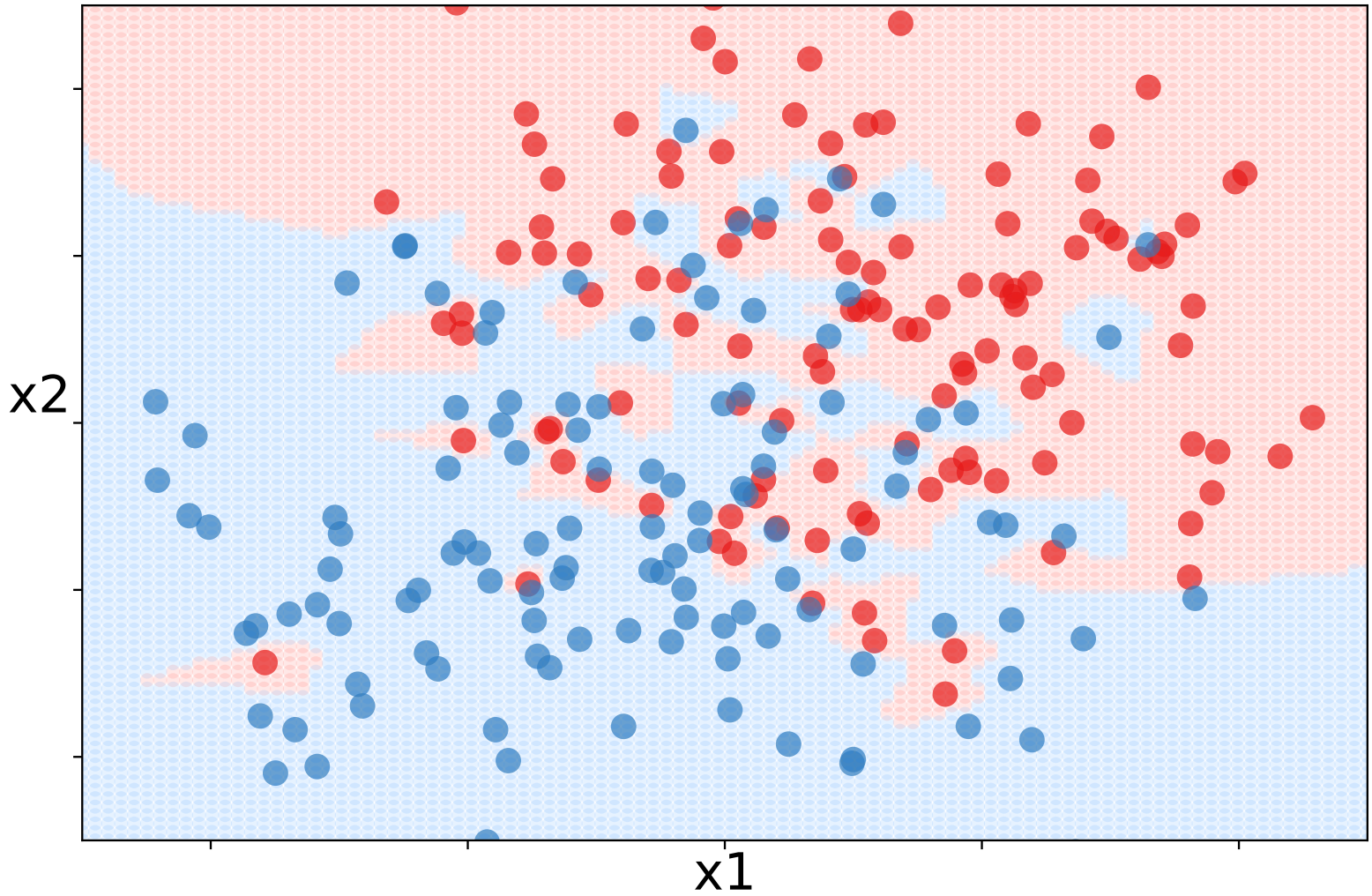
k-NN vs Logistic



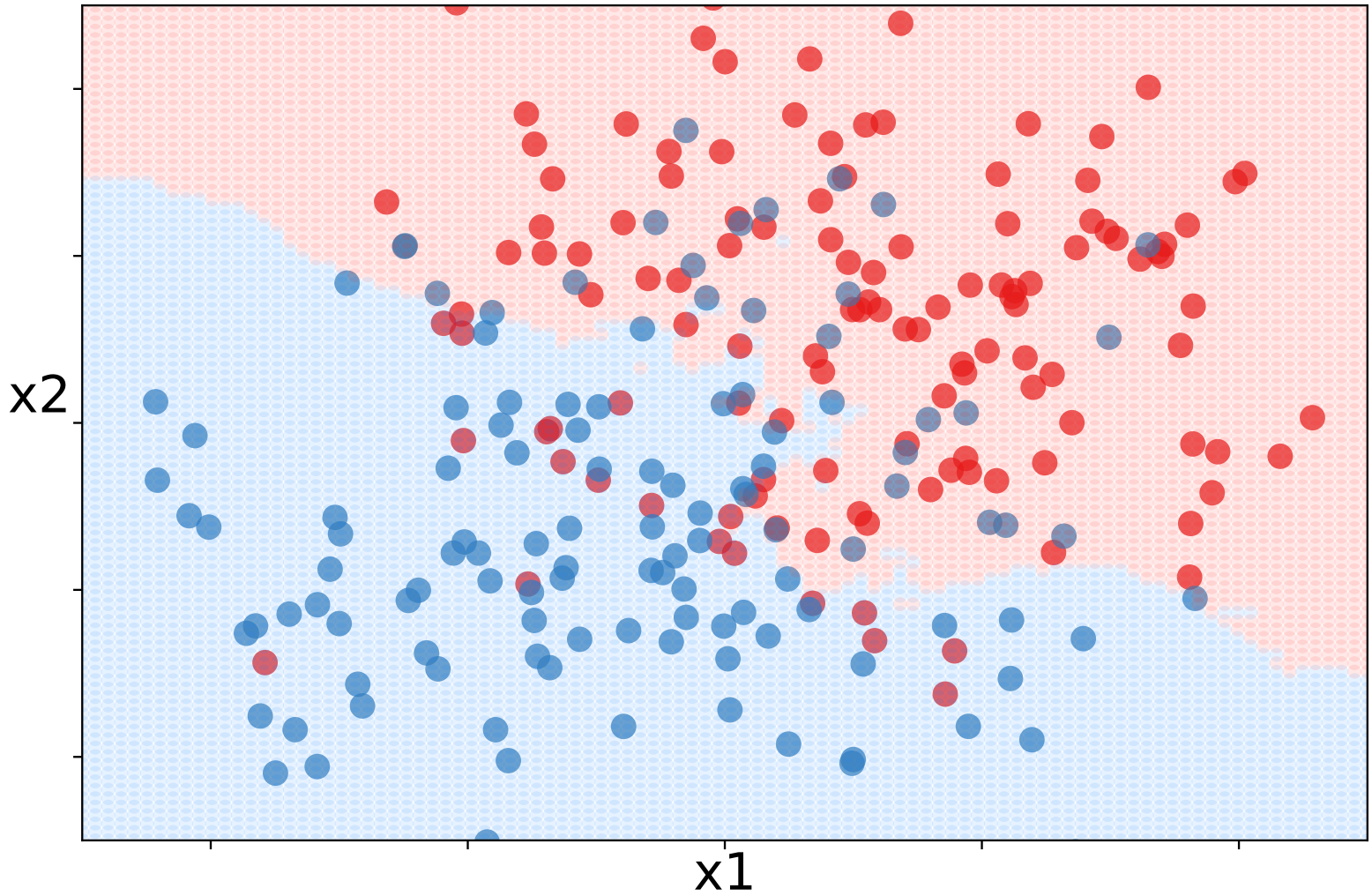
k-NN vs Logistic



k-NN vs Logistic



k-NN vs Logistic



k-NN Classification

+ **Do not presume any model for the data**
  - As opposed to, e.g, linear/logisitc regression, SVMs etc.

k-NN Classification

+ **Do not presume any model for the data**
  - As opposed to, e.g, linear/logisitc regression, SVMs etc.
+ (can) become **more powerful with more training samples** n
  - (can) increase k as n increases ...

k-NN Classification

+ **Do not presume any model for the data**
  - As opposed to, e.g, linear/logisitc regression, SVMs etc.
+ (can) become **more powerful with more training samples** n
  - (can) increase k as n increases ...
+ **No model selection** required – just store points in memory ...

k-NN Classification

+ **Do not presume any model for the data**
  - As opposed to, e.g, linear/logisitc regression, SVMs etc.
+ (can) become **more powerful with more training samples** n
  - (can) increase k as n increases ...
+ **No model selection** required – just store points in memory ...
– **High query-time complexity**
  - Naively, every query needs to look at all training data
  - Many optimizations/approximations. E.g. locality-sensitive hashing

k-NN Classification

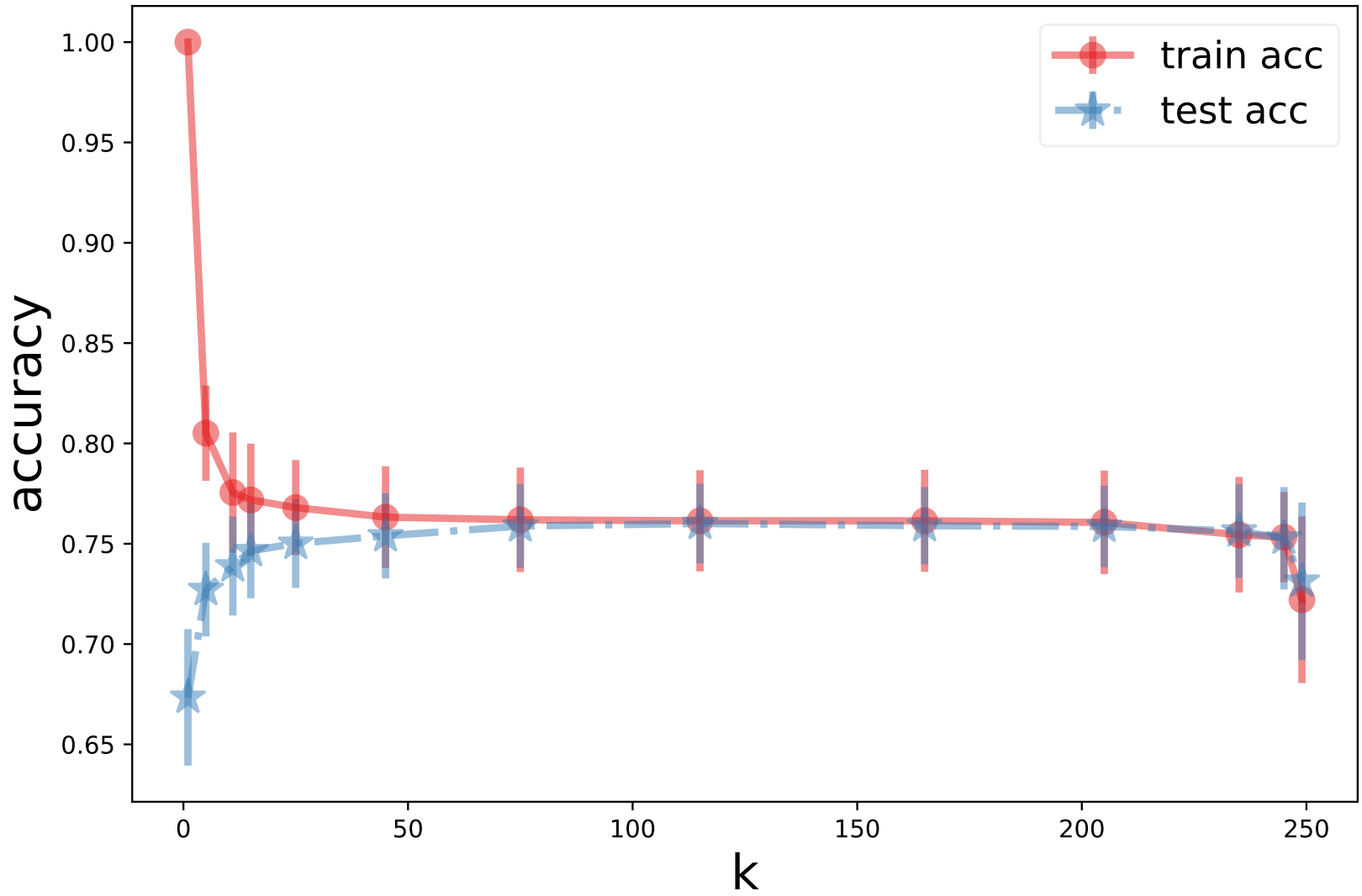
+ **Do not presume any model for the data**
  - As opposed to, e.g, linear/logisitc regression, SVMs etc.
+ (can) become **more powerful with more training samples** n
  - (can) increase k as n increases ...
+ **No model selection** required – just store points in memory ...
– **High query-time complexity**
  - Naively, every query needs to look at all training data
  - Many optimizations/approximations. E.g. locality-sensitive hashing
– Highly susceptible to **curse of dimensionality**
  - Especially if there are lots of irrelevant features

k-NN Classification

+ **Do not presume any model for the data**
  - As opposed to, e.g, linear/logisitc regression, SVMs etc.
+ (can) become **more powerful with more training samples** n
  - (can) increase k as n increases ...
+ **No model selection** required – just store points in memory ...
– **High query-time complexity**
  - Naively, every query needs to look at all training data
  - Many optimizations/approximations. E.g. locality-sensitive hashing
– Highly susceptible to **curse of dimensionality**
  - Especially if there are lots of irrelevant features
- Performance often depends on choice of distance measure

k-NN Classification

Train and test accuracy as a function of k



Curse of Dimensionality

k-NN performance degrades if there are a lot of irrelevant features.

In the following examples:

$+1$ samples: d-dimensional gaussians with center $(1, 1, 1, 1, 1, 0, \ldots, 0)$

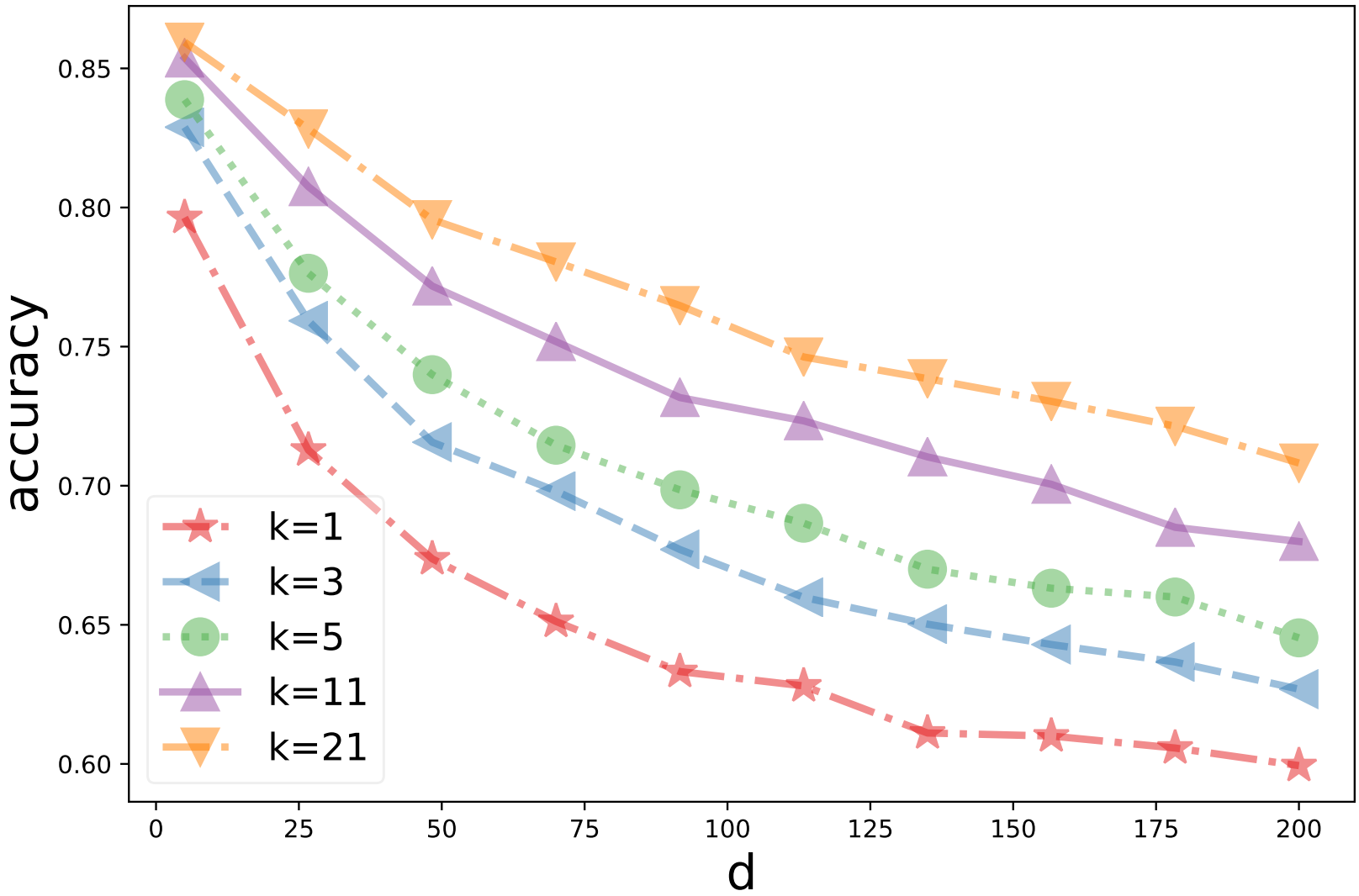
-1 samples: d-dimensional gaussians with center $(-1-, 1, -1, -1, -1, 0, \ldots, 0)$

i.e. **only first 5 features actually matter** – remaining $d - 5$ just add noise.

We compare k-NN and LDA.

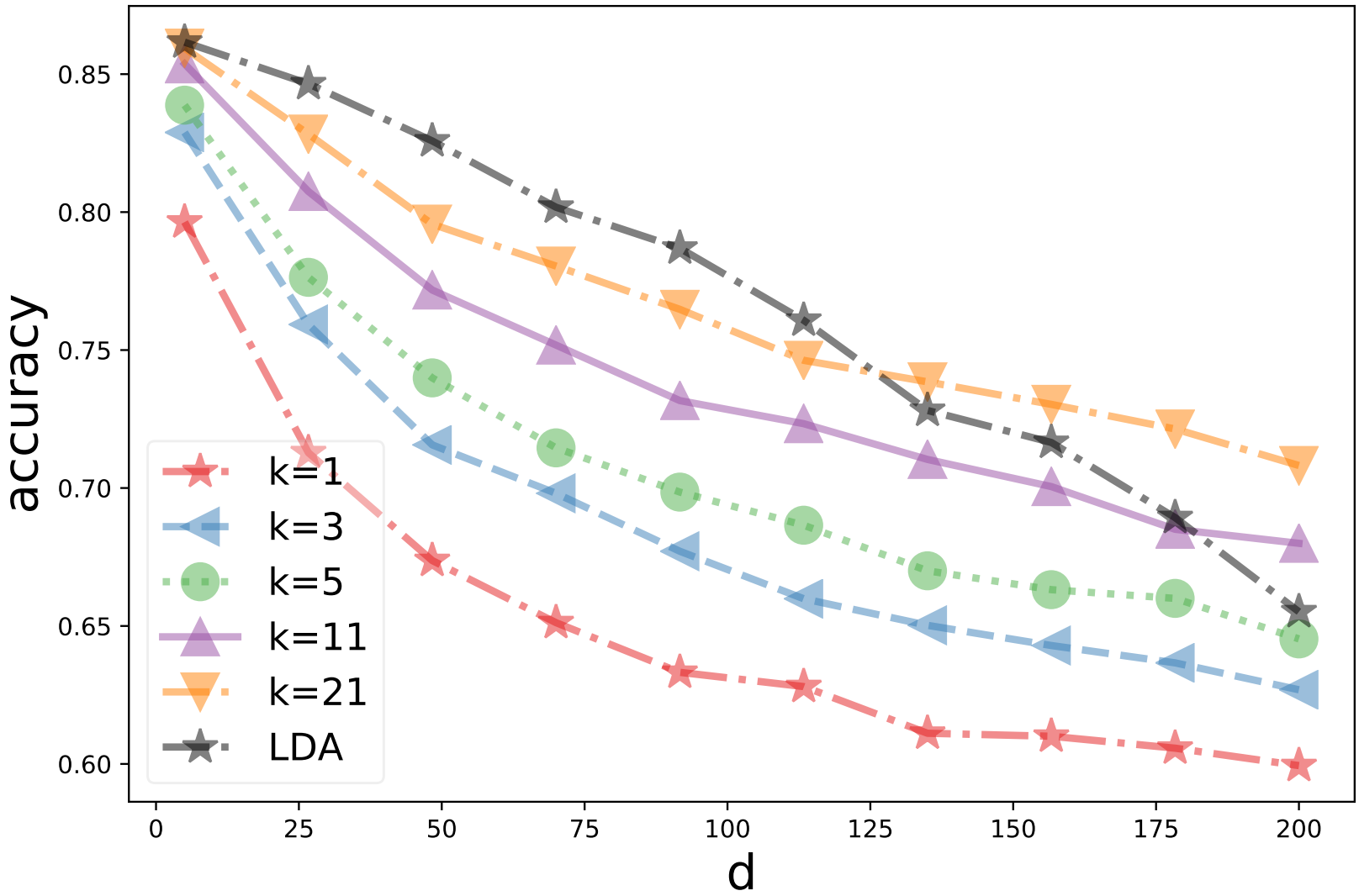
Curse of Dimensionality

250 training samples



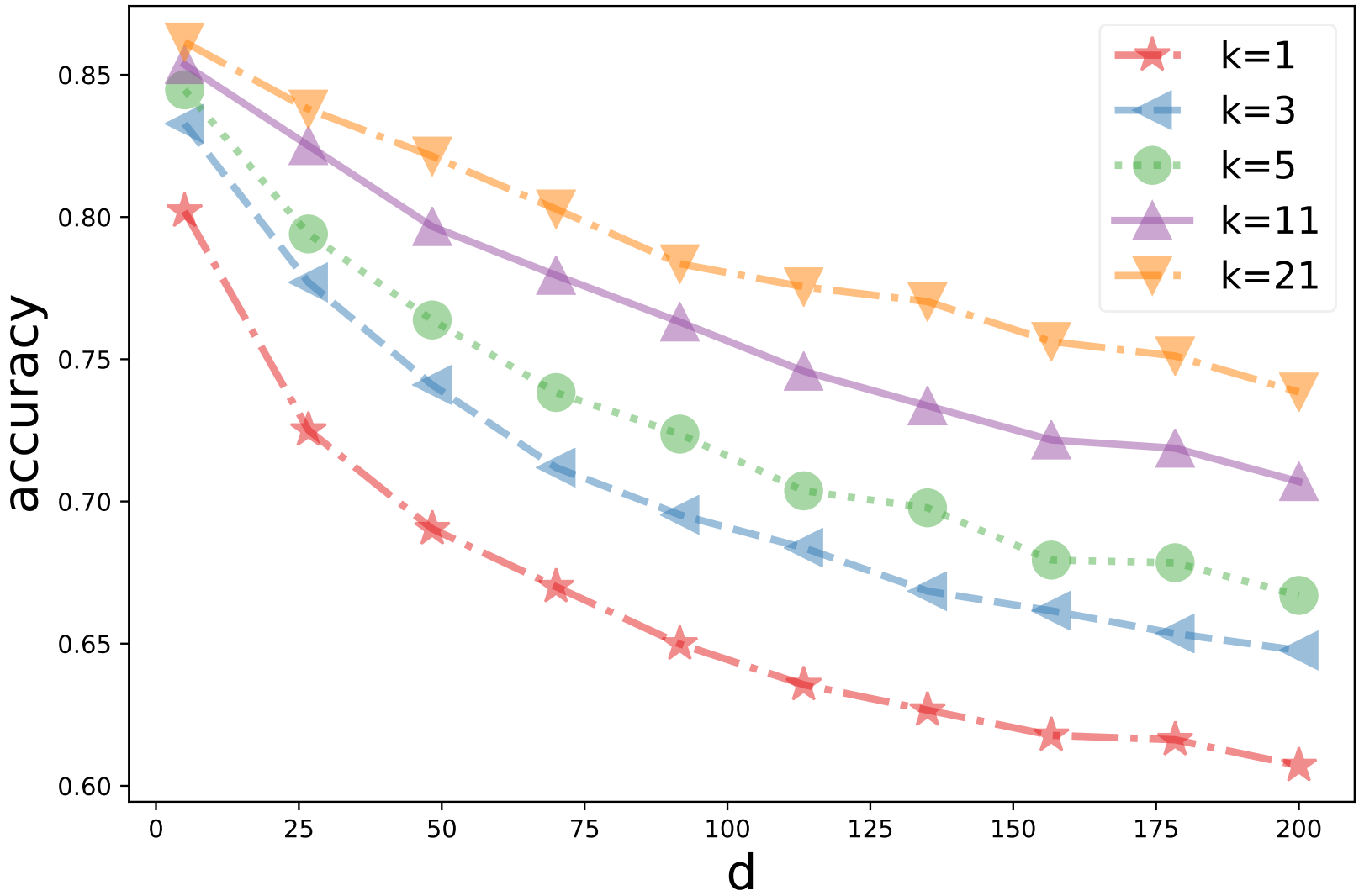
Curse of Dimensionality

250 training samples, with LDA



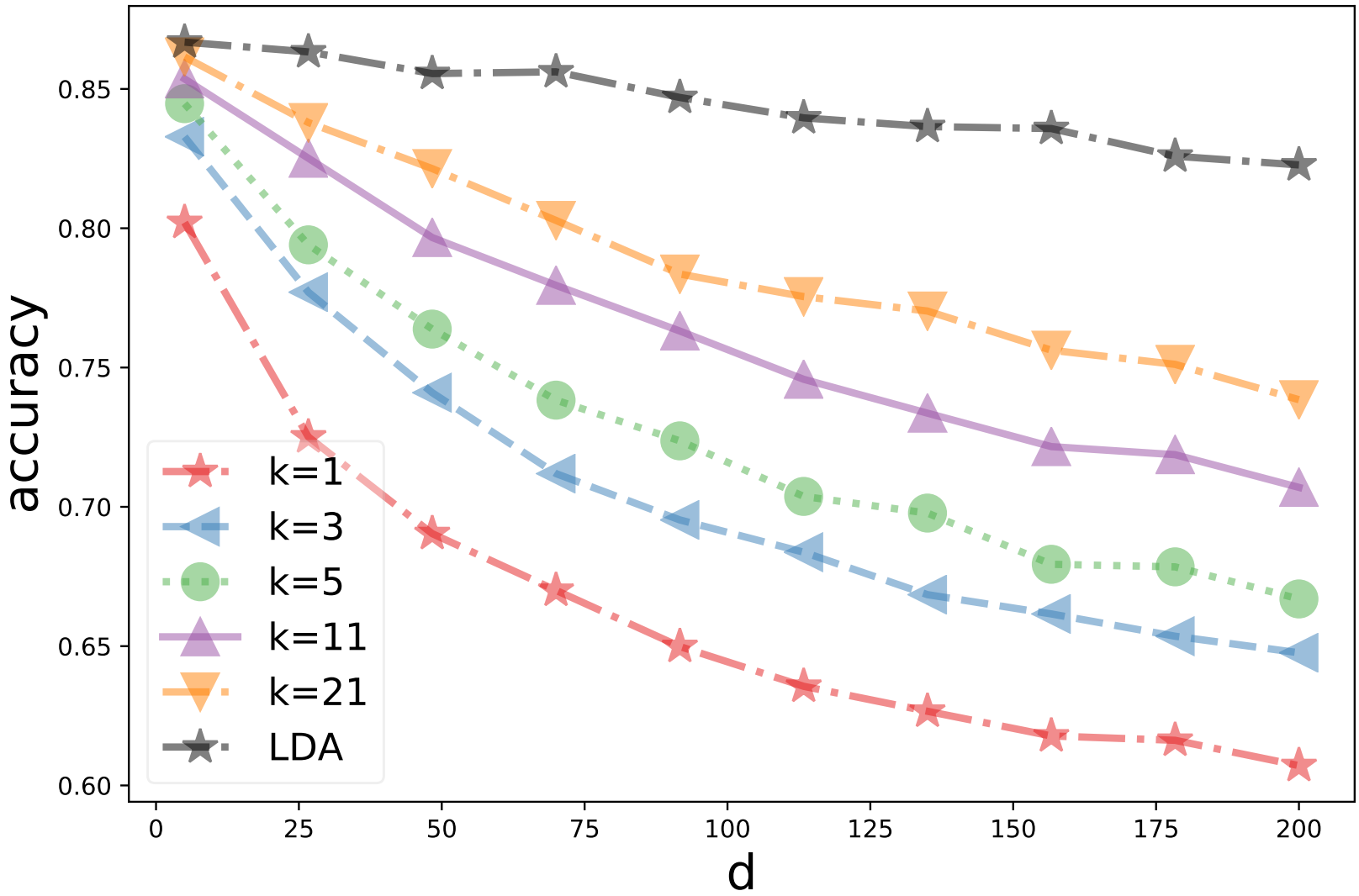
Curse of Dimensionality

1000 training samples



Curse of Dimensionality

1000 training samples, with LDA

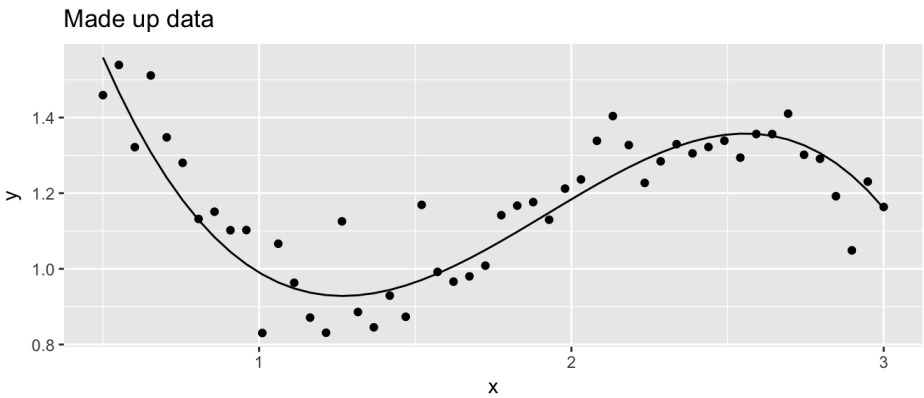


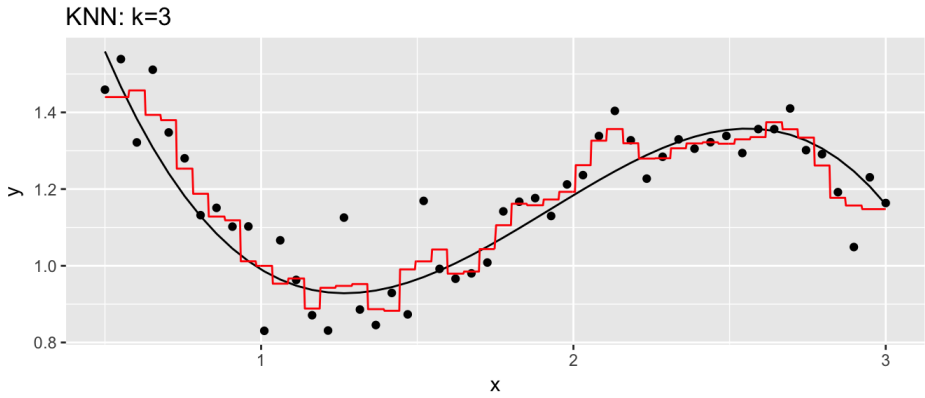
k-NN methods

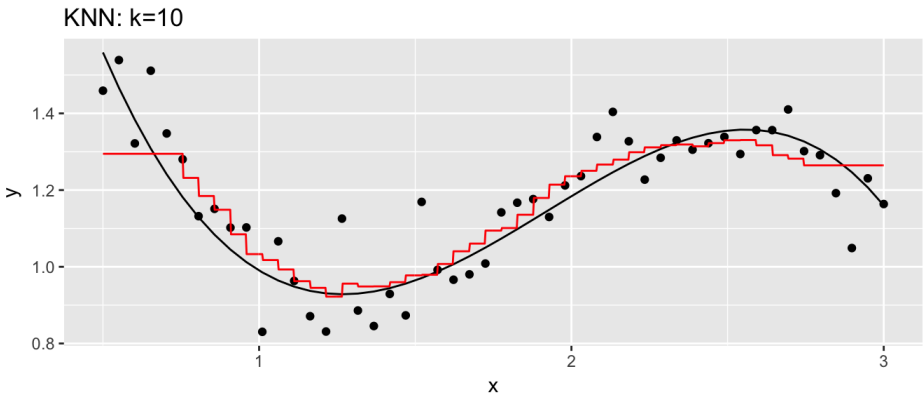
Exact k-NN: find the k nearest neighbors of a query exactly.
Methods: Exhaustive search, Space partitioning
Only attempted for small datasets, due to high time and space complexity

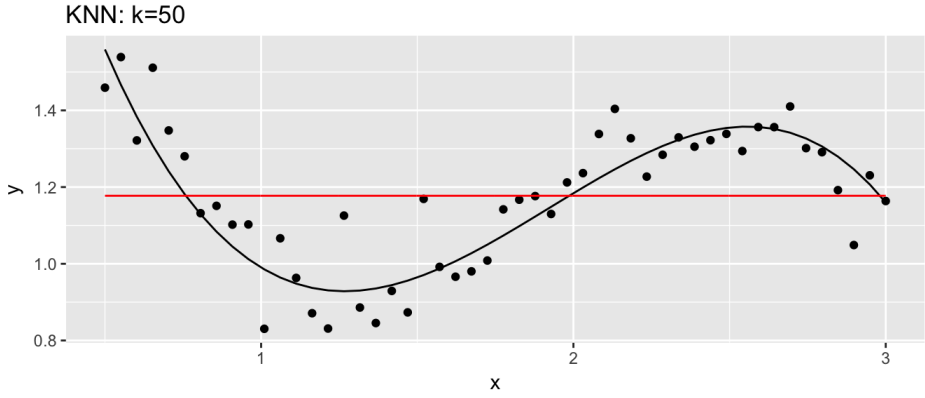
Approximate k-NN: preprocess the data into efficient data structures, and then use this data structure to find the nearest neighbors approximately.
Very widely used, many high-quality open-source solutions available

k-NN Regression









(Credit: Derek Sonderegger, ASU)

Local Linear Regression

An extension of the k-NN regression idea: now fit a model locally

- Learn a custom $\widehat{\beta}$ just for the query x:

$$\widehat{\beta}(x) \;=\; \arg\min_{\beta} \sum_{i \in \mathcal{S}_k(x)} \left(y^{(i)} - \beta^\top x^{(i)} \right)^2$$

- Predict $\widehat{y}(x) \;=\; x^\top \widehat{\beta}(x)$

More generally, weight each sample i based on distance of $x^{(i)}$ from x, then do weighted regression, etc etc...

Approximate NN

Biggest drawback of k-NN: finding nearest neighbors at query time is slow.

Approximate NN:

1. (offline) Build data structure using the training points
2. (online) query data structure to obtain approximate nearest neighbors

We will study two techniques:

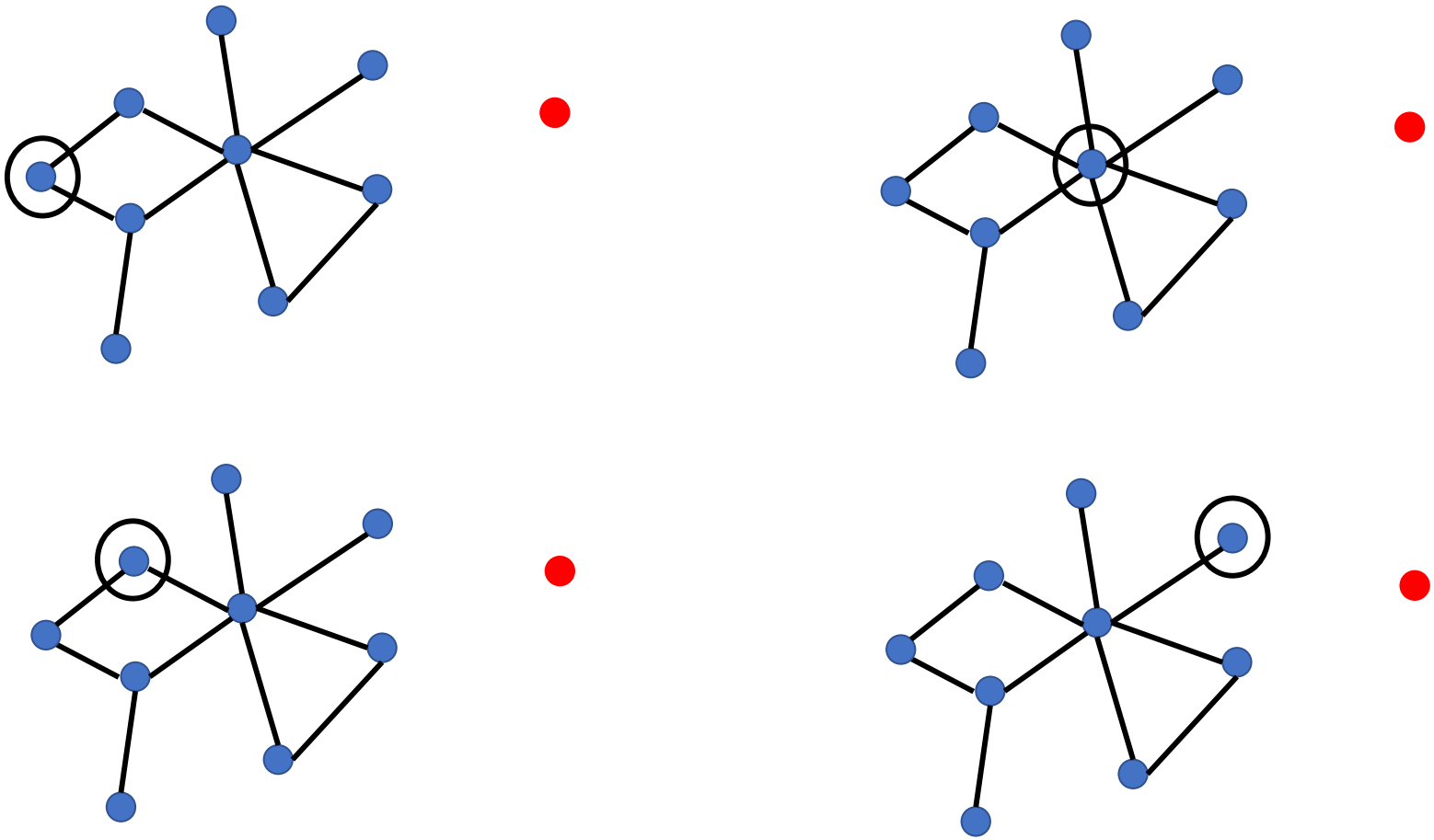
- Locality Sensitive Hashing
- Proximity graph based methods

Proximity graphs and greedy search for NN

Basic idea:

1. Offline: Build a graph where each node is a training datapoint
2. Online: given a query q,
   1. start at a node in the graph
   2. greedily update: look at all neighbors of current node, and see if one is closer to q
   3. if yes, move to that node. if not, declare current node as closest.

Proximity graphs and greedy search for NN



Considerations in Proximity Graph Methods

Building the graph

1. Needs to guarantee that greedy search will find (near) exact near-neighbors
2. Will get close to query in a small number of hops
3. Graph can be built efficiently

What is a "good" graph depends on the distance function, distribution of points, etc.

In recent years, proximity graph based methods are generally believed to be the state of the art.

Locality Sensitive Hashing

Hash function: maps each feature vector into one of a finite number of buckets (very fast)

Idea: us a **special hash function** $h(x)$ such that if u and v are close, then the event $h(u) = h(v)$ is more likely ...

Then: replace the bottleneck "find nearest neighbors" step in k-NN with "other points in the same bucket" (very fast)

Locality Sensitive Hashing

Locality sensitive hash (LSH) function: $h(x) = +1$ or -1, such that $\mathbf{P}[h(u) = h(v)]$ is higher if u and v closer.

e.g. suppose $x \in \mathbf{R}^d$. For random vector r, consider $h_r(x) = sign(r^\top x)$

For this,

$$\mathbf{P}[h_r(u) = h_r(v)] \;=\; 1 - \frac{\theta(u,v)}{\pi}$$

i.e. the smaller the angle $\theta(u,v)$ between u and v, the more likely they will be hashed to the same value.

Note: Here the probability is over the randomness in r ...

Locality Sensitive Hashing

An **LSH family** is called (d_1, d_2, p_1, p_2)-sensitive if for any two vectors u and v, when we pick a random h from this family, we get

$$\begin{aligned} \mathbf{P}[h(u) = h(v)] &\geq p_1 \quad \text{if } d(u, v) \leq d_1 \\ \mathbf{P}[h(u) = h(v)] &\leq p_2 \quad \text{if } d(u, v) \geq d_2 \end{aligned}$$

Here $d_1 < d_2$ and $p_1 > p_2$

Locality Sensitive Hashing

An **LSH family** is called (d_1, d_2, p_1, p_2)-sensitive if for any two vectors u and v, when we pick a random h from this family, we get

$$\begin{aligned} \mathbf{P}[h(u) = h(v)] &\geq p_1 \quad \text{if } d(u,v) \leq d_1 \\ \mathbf{P}[h(u) = h(v)] &\leq p_2 \quad \text{if } d(u,v) \geq d_2 \end{aligned}$$

Here $d_1 < d_2$ and $p_1 > p_2$

Amplification: Pick k random functions from family, and then combine in AND or OR fashion:

$$\begin{aligned} g_{AND}(u) = g_{AND}(v) &\leftrightarrow h_i(u) = h_i(v) \text{ for all } i \in [k] \\ g_{OR}(u) = g_{OR}(v) &\leftrightarrow h_i(u) = h_i(v) \text{ for some } i \in [k] \end{aligned}$$

Locality Sensitive Hashing

An **LSH family** is called (d_1, d_2, p_1, p_2)-sensitive if for any two vectors u and v, when we pick a random h from this family, we get

$$\begin{aligned} \mathbf{P}[h(u) = h(v)] &\geq p_1 \quad \text{if } d(u,v) \leq d_1 \\ \mathbf{P}[h(u) = h(v)] &\leq p_2 \quad \text{if } d(u,v) \geq d_2 \end{aligned}$$

Here $d_1 < d_2$ and $p_1 > p_2$

Amplification: Pick k random functions from family, and then combine in AND or OR fashion:

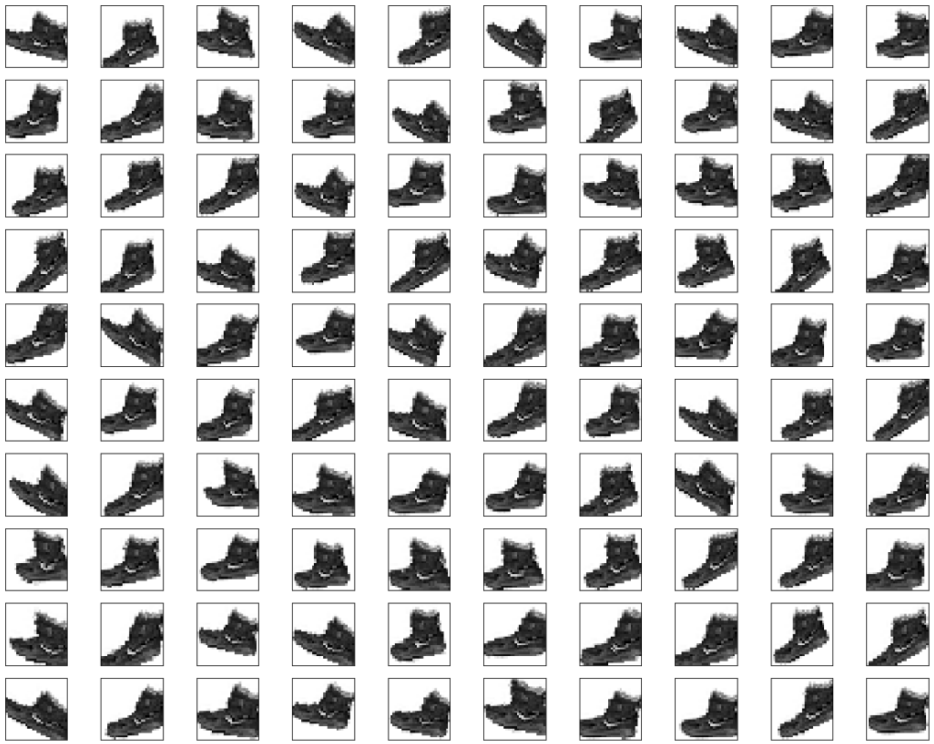
$$\begin{aligned} g_{AND}(u) = g_{AND}(v) &\leftrightarrow h_i(u) = h_i(v) \text{ for all } i \in [k] \\ g_{OR}(u) = g_{OR}(v) &\leftrightarrow h_i(u) = h_i(v) \text{ for some } i \in [k] \end{aligned}$$

g_{AND} is (d_1, d_2, p_1^k, p_2^k)-sensitive
g_{OR} is $(d_1, d_2, 1-(1-p_1)^k, 1-(1-p_2)^k)$-sensitive.
Concatenating ANDs and ORs can make arbitrarily powerful families.

Data Augmentation

In high-dimensional spaces, k-NN can be very noisy because most points "seem equally far away"

A common empirical trick: add *fake* training samples by perturbing existing ones



How to augment: case by case basis.

Especially popular for image datasets, where one can rotate, shear, shift, mirror etc.

Data augmentation now also very popular in neural networks.