Nearest Neighbor
&
Instance-Based Learning

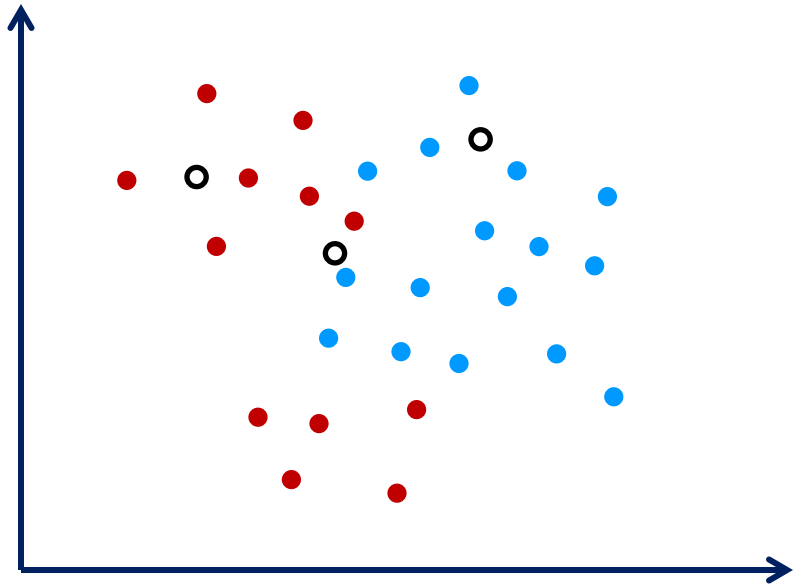
Zhiyao Duan
Professor of ECE and CS
University of Rochester

Some figures are copied from the following books

- **LWLS** - Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, Thomas B. Schön, *Machine Learning: A First Course for Engineers and Scientists*, Cambridge University Press, 2022.
- **Mitchell** - Tom M. Mitchell, *Machine Learning*, McGraw-Hill Education, 1997.

Key Idea of NN

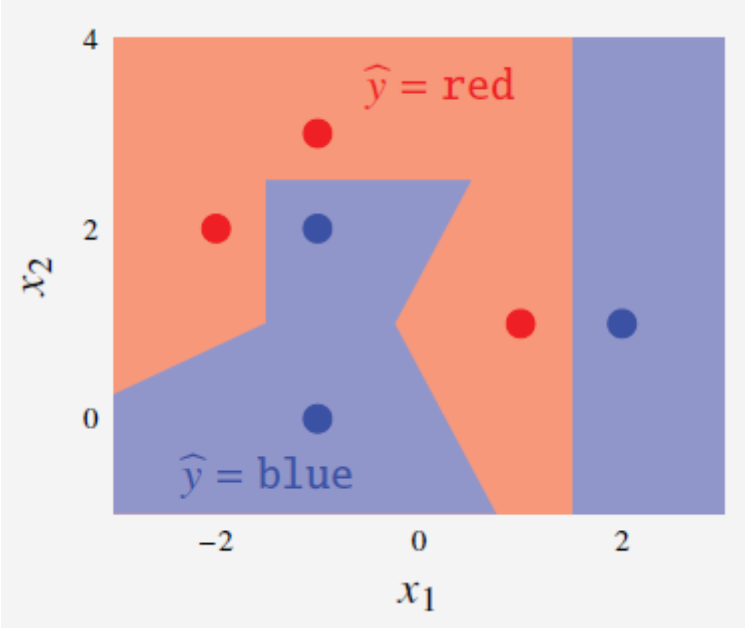
- Supervised learning: given examples $(\boldsymbol{X}, \boldsymbol{y})$, learn f: $\boldsymbol{x} \mapsto y$
- For a new sample $\boldsymbol{x}$, predict its label y based on that of its closest training sample $\boldsymbol{x}_{nn}$
  - A local method



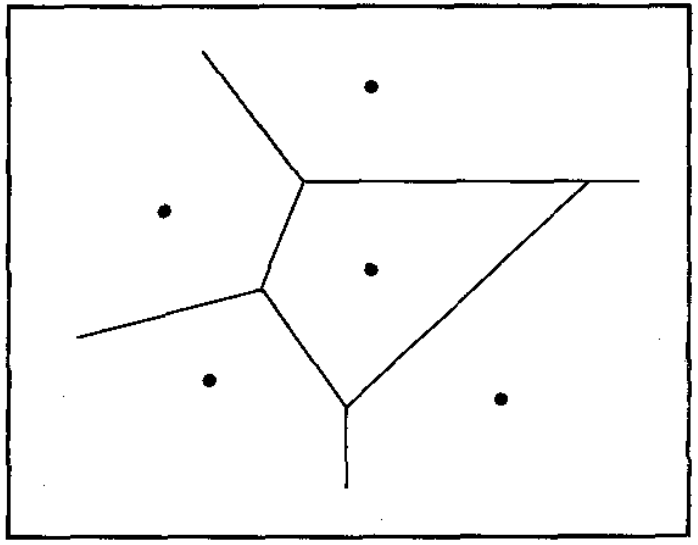
- We do not learn f from training data explicitly; we only process training data to predict $f(\boldsymbol{x})$ when $\boldsymbol{x}$ is presented – Lazy learning

Classification Boundary

- Given a training set, although f is not learned explicitly, it exists.
- Classification boundary: the boundary across which $f(\boldsymbol{x})$ changes
  - Linear vs. non-linear
  - For NN classification, it is piecewise linear



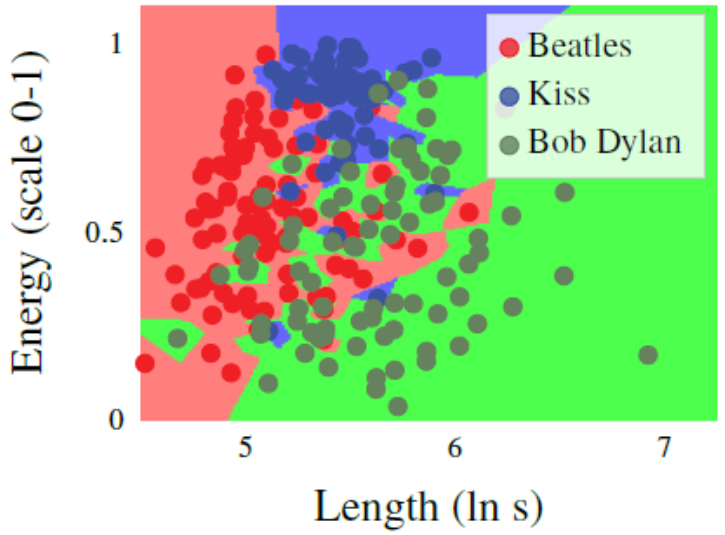
(Fig. 2.4 in LWLS)



Voronoi diagram
(Fig. 8.1 in Mitchell)

Classification Error

- Training error: error made by the learned f on the training set
- Test error: error made by the learned f on the test set
- Training error rate is generally lower than test error rate, but a big mismatch between training and test error rates indicates overfitting



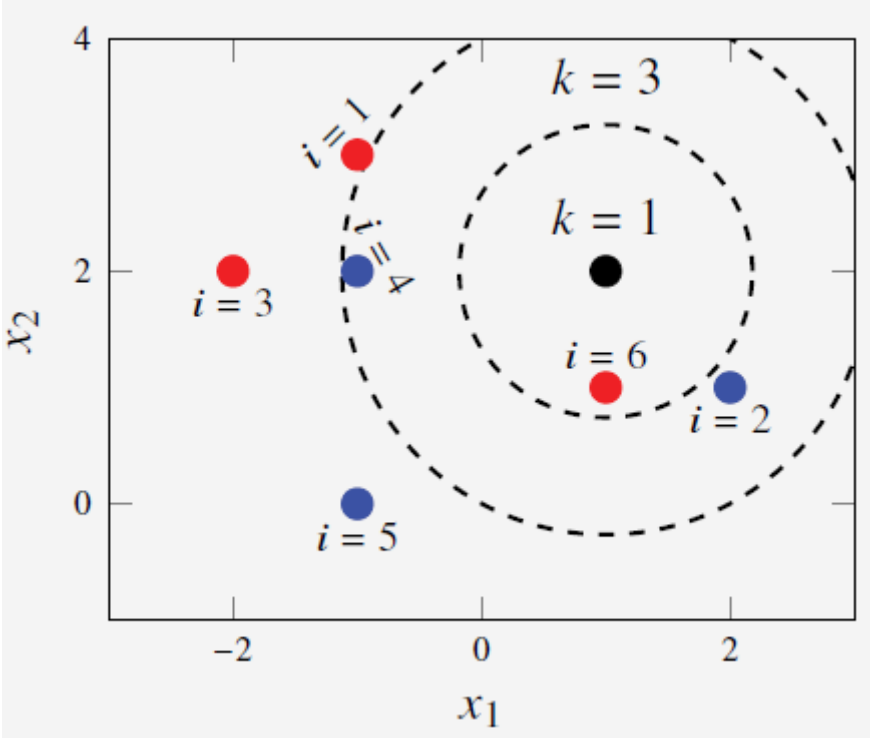
(Fig. 2.5(a) in LWLS)

- NN has zero training error, very likely to overfit

k-Nearest Neighbor (kNN)

- Key idea: let's consult several closest points and take the majority vote

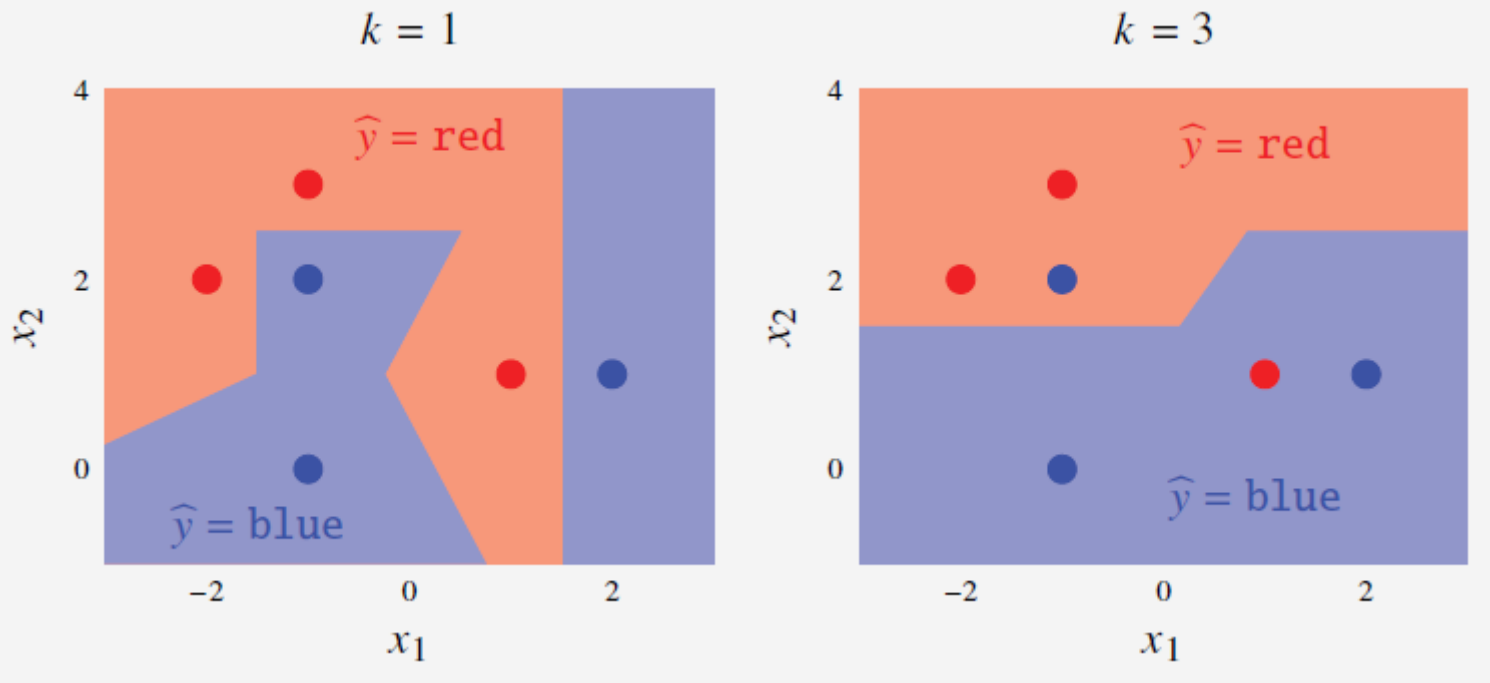
$$f(\boldsymbol{x}) = MajorityVote\{y^{(i)}: i \in \mathcal{N}_k(\boldsymbol{x})\} = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} \delta(y, y^{(i)})$$



(Fig. 2.3 in LWLS)

Classification Boundary

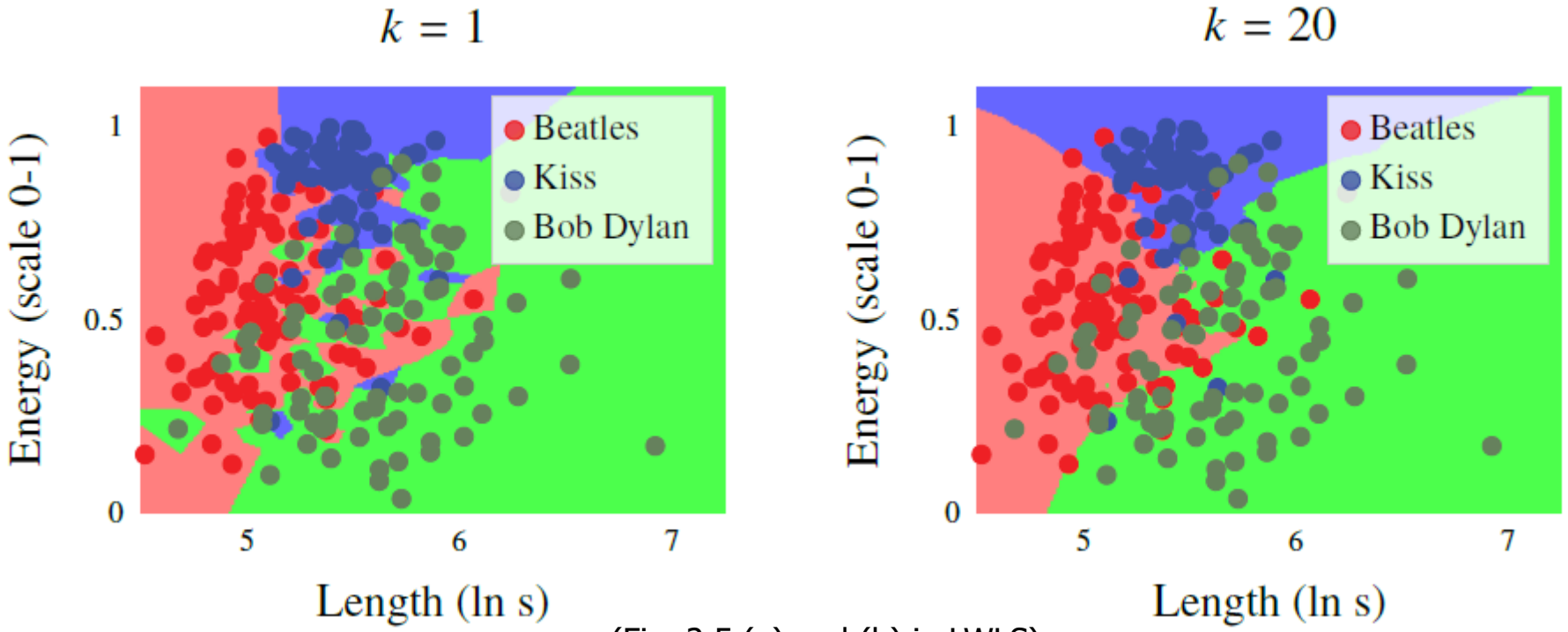
- For kNN, the classification boundary is also piecewise linear. Why?
  - Hint: think about when the k-neighborhood changes



(Fig. 2.4 in LWLS)

Classification Error

- Training set error of kNN is no longer 0



(Fig. 2.5 (a) and (b) in LWLS)

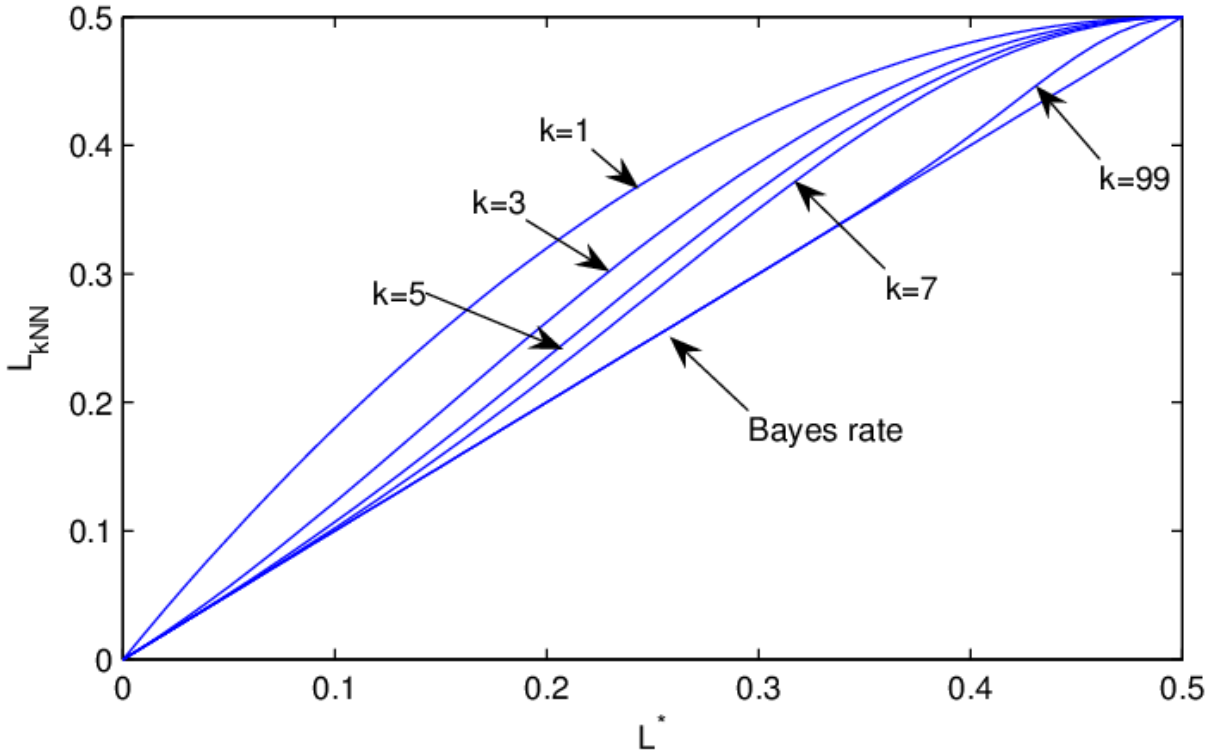
- Less overfitting: classification boundary is smoother; test error is generally lower

Choosing k

- k is the hyper-parameter of kNN
- Higher k generally results in a smoother classification boundary, making it more robust to label noise in the training data
- Too high of k would "wash out" interesting patterns in the training data, leading to meaningless results
- Selecting k is a trade-off between flexibility and rigidity
- Cross validation is an effective way to setting hyper-parameters, including k

Error Bounds

- Asymptotic error rate of kNN vs. Bayes error rate (the best possible)
  - Error rate is bounded between the arc and diagonal line

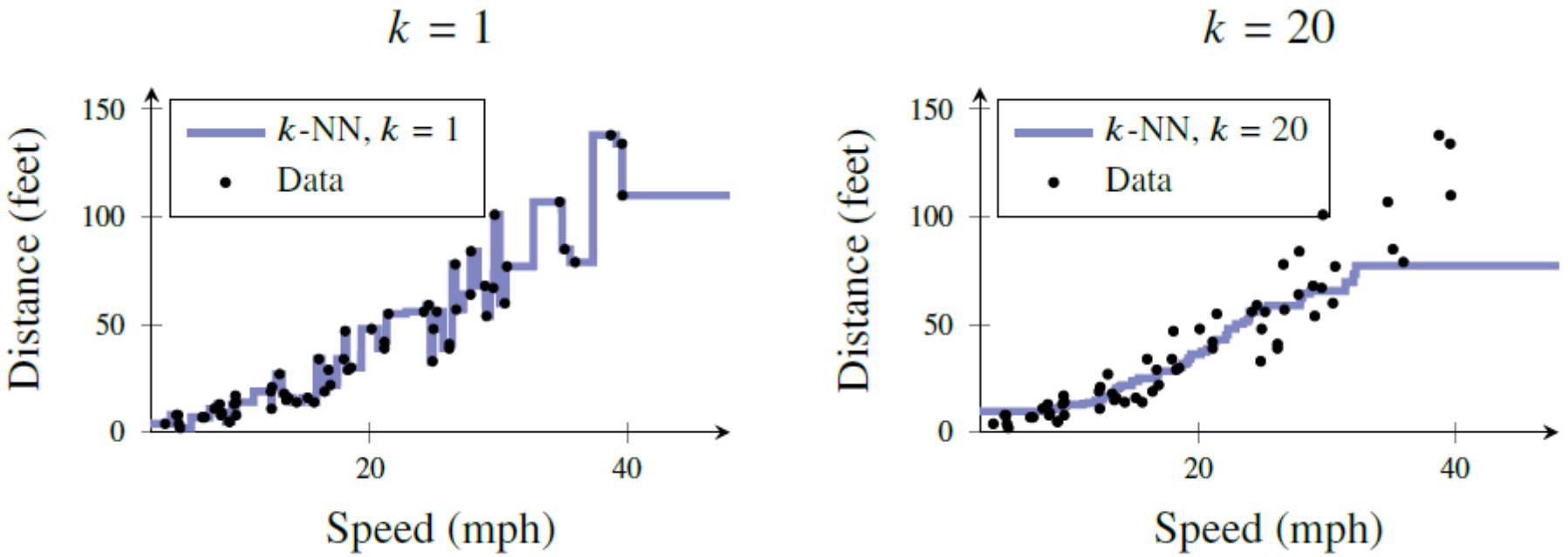


(Figure from [Castellana & Biagi, 2008])

kNN Regression

- Supervised learning: given examples $(\boldsymbol{X}, \boldsymbol{y})$, learn $f\colon \boldsymbol{x} \mapsto y$, where y is continuous
- Key idea: let's consult several closest points and take the average

$$f(\boldsymbol{x}) = Average\{y^{(i)} : i \in \mathcal{N}_k(\boldsymbol{x})\} = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} y^{(i)}$$



(Fig. 2.5 (c) and (c) in LWLS)

Distance Weighted NN

- Key idea: weigh training samples according to distances from query
- Classification

$$f(\boldsymbol{x}) = \operatorname*{argmax}_{y \in \mathcal{Y}} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} w_i \delta(y, y^{(i)})$$

- Regression

$$f(\boldsymbol{x}) = \frac{1}{\sum_{i=1}^{k} w_i} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} w_i y^{(i)}$$

- Weight decreases with distance, e.g., $w_i = \frac{1}{d(x, x^{(i)})^2}$

- With this formulation, we can use all training samples instead of just the k nearest neighbors
  - Becomes a global method

Distances

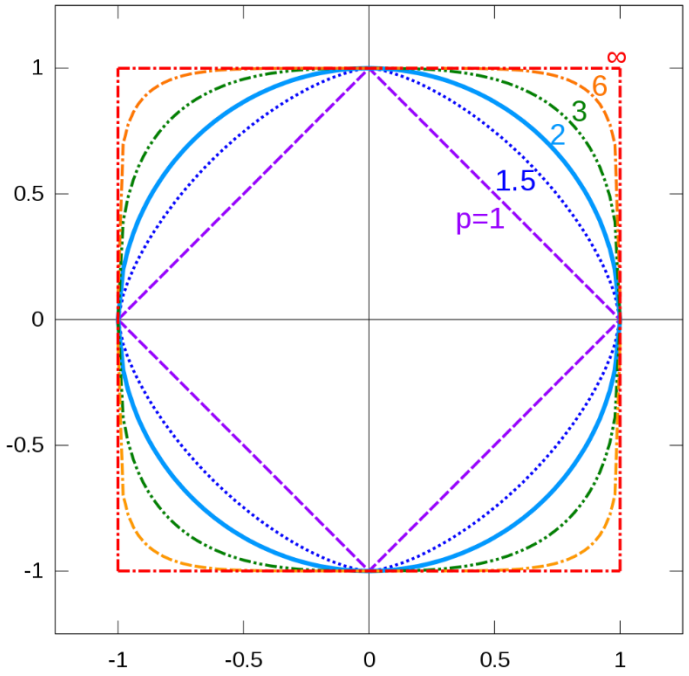
- $d: X \times X \to \mathbb{R}$ is a distance (or metric) if $\forall a, b, c \in X$
  - $d(a, a) = 0$
  - Positivity: $d(a, b) > 0$ if $\mathrm{a} \neq \mathrm{b}$
  - Symmetry: $d(a, b) = d(b, a)$
  - Triangle inequality: $d(a, b) + d(b, c) \geq d(a, c)$
- For Euclidian space $\mathbb{R}^n$
  - Euclidian distance: $d(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$, i.e., L^2 distance
  - Manhattan distance: $d(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{n}|a_i - b_i|$, i.e., L^1 distance
  - Chebyshev distance: $d(\boldsymbol{a}, \boldsymbol{b}) = \max_i |a_i - b_i|$, i.e., L^∞ distance
  - Hamming distance for binaries: $d(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{n}(1 - \delta(a_i - b_i))$, i.e., L^0 "distance"
  - In general, L^p distance is defined as

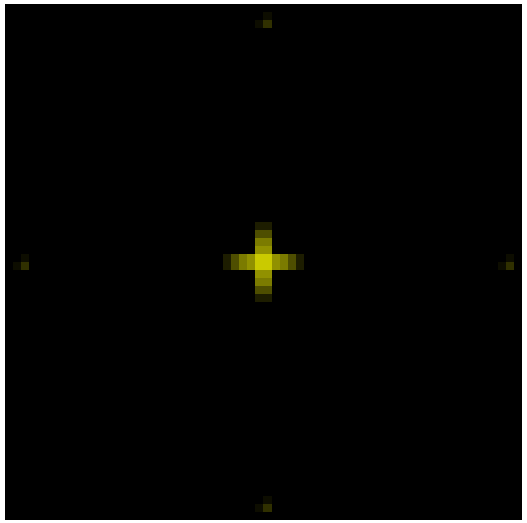
$$d(\boldsymbol{a}, \boldsymbol{b}) = \left(\sum_{i=1}^{n}|a_i - b_i|^p\right)^{\frac{1}{p}}, \text{ if } \mathrm{p} \geq 1$$

$$d(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=1}^{n}|a_i - b_i|^p, \text{ if } 0 \leq \mathrm{p} < 1$$

Distances

- Norm of a vector: distance from the origin





Figures from: https://en.wikipedia.org/wiki/Lp_space

Other Similarity Measures

- Cosine similarity

$$\cos\theta = \frac{< \boldsymbol{a}, \boldsymbol{b} >}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|}$$

- Cosine "distance": does not satisfy triangle inequality

$$D_{cos}(\boldsymbol{a}, \boldsymbol{b}) = 1 - \cos\theta$$

- Kullback-Leibler divergence: not symmetric

$$D_{KL}(P, Q) = \sum_x P(x) \ln \frac{P(x)}{Q(x)}$$

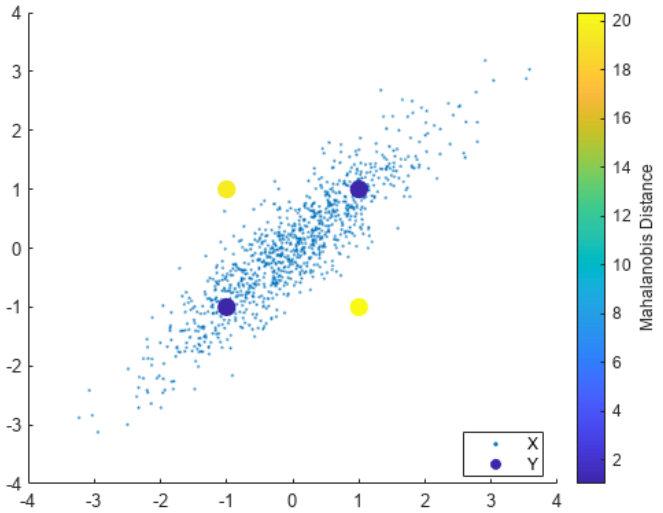
- Jensen-Shannon divergence: does not satisfy triangle inequality

$J(P, Q) = \frac{1}{2}(D_{KL}(P, R) + D_{KL}(Q, R))$, where $R = \frac{P+Q}{2}$

Data-Driven Distances

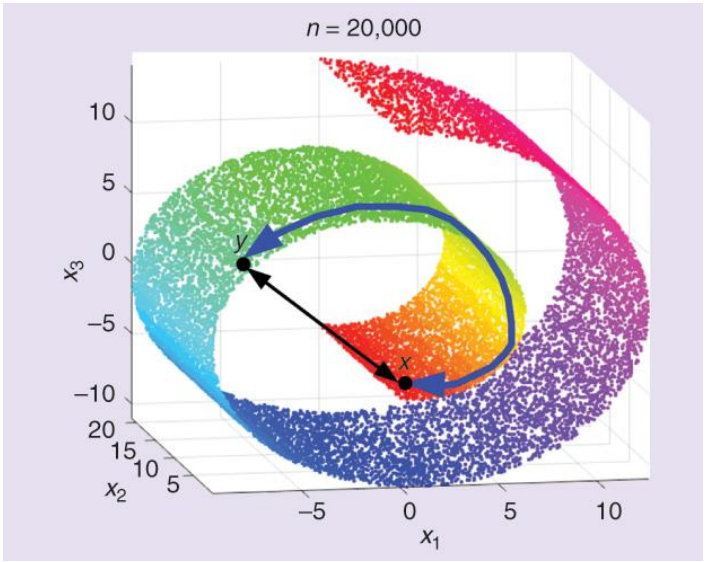
- Mahalanobis distance
  - Assume training data $\boldsymbol{x} \sim p(\boldsymbol{x})$, and its covariance matrix $\boldsymbol{C}$ is positive-definite

$$d_M(\boldsymbol{a}, \boldsymbol{b}) = \sqrt{(\boldsymbol{a} - \boldsymbol{b})^T \boldsymbol{C}^{-1} (\boldsymbol{a} - \boldsymbol{b})}$$



(Figure from https://www.mathworks.com/help/stats/mahal.html)

- Geodesic distance: length of shortest path through the data manifold



(Figure from https://www.embs.org/pulse/articles/what-is-the-distance-between-objects-in-a-data-set/)

Data Normalization

- Different features may be at different scales as they have different meanings (e.g., length vs. voltage) and use arbitrary units (e.g., cm vs. meter)
- Min-max normalization

$$x_j^{new} = \frac{x_j - \min_i x_j^{(i)}}{\max_i x_j^{(i)} - \min_i x_j^{(i)}}$$

where i is index of training examples

- Variance normalization

$$x_j^{new} = \frac{x_j - \mu_j}{\sigma_j}$$

where μ_j and σ_j are mean and standard deviation of the training examples

- Note: normalization parameters should be calculated from training data, and then be used on validation and test data.

Which features are relevant?

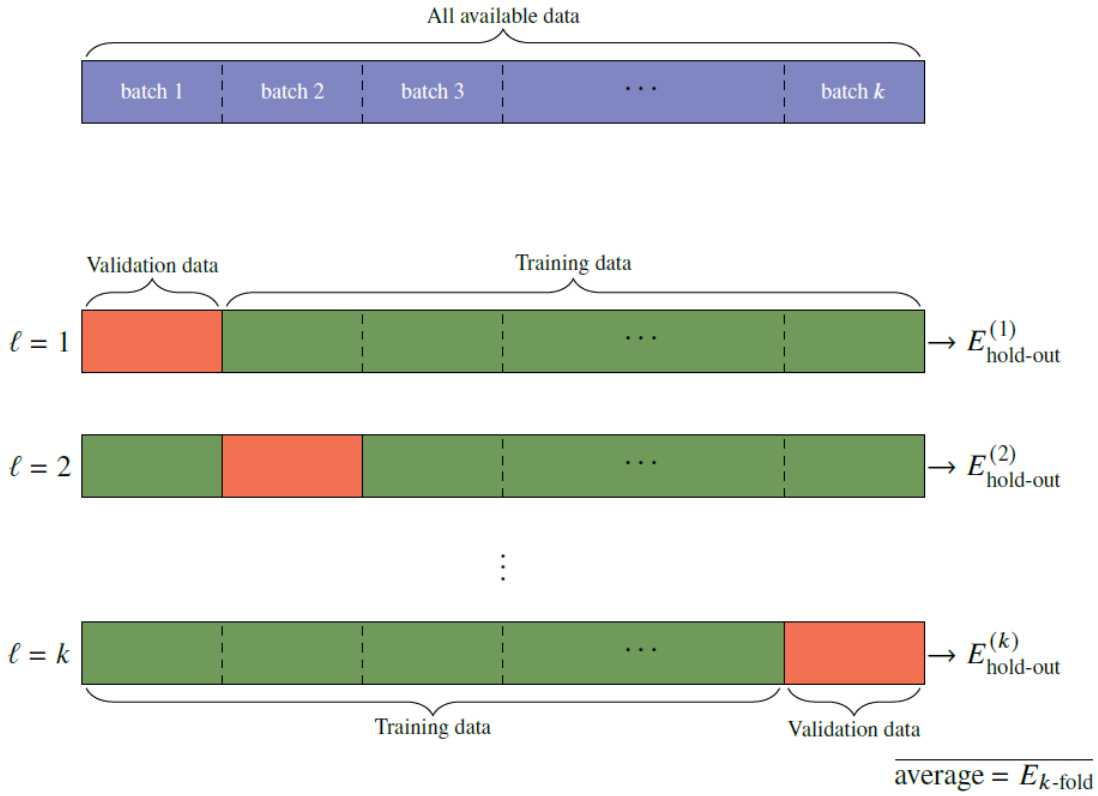
- The distance calculation uses all features (dimensions), but what if some dimensions are irrelevant?

- (One type of) curse of dimensionality: The more dimensions we have, the more irrelevant dimensions there are, and the distance calculation is more easily dominated by those irrelevant dimensions

- Find out the usefulness of dimensions on the validation set
  - Add/remove certain dimensions
  - Scale certain dimensions
  - These are hyper-parameters

Cross Validation

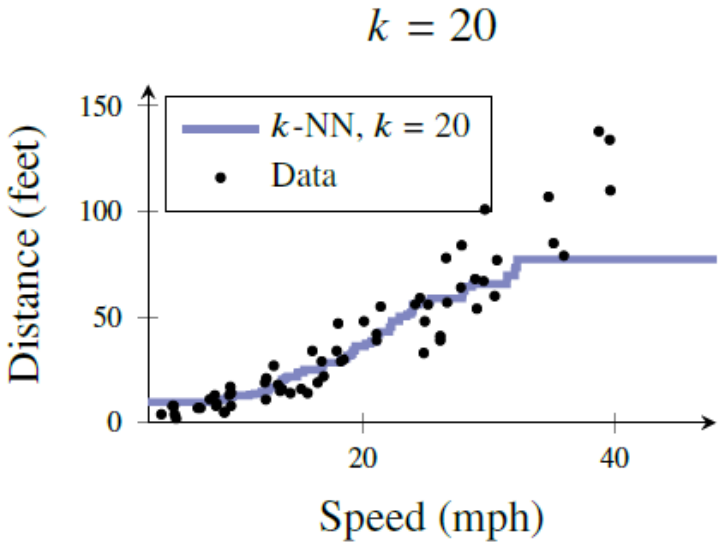
- K-fold cross validation: partition dataset into k-folds, and rotate



- Leave-one-out: treating each sample as a fold
- Note: cross-validation error E_{k-fold} is usually not a good estimate of the error on unseen data. Why?

Locally Weighted Regression

- kNN regression: $f(\boldsymbol{x}) = \frac{1}{k}\sum_{i\in\mathcal{N}_k(\boldsymbol{x})} y^{(i)}$
  - Local regression
  - Piecewise constant fitting
  - Equal weights from all k neighbors
  - Is one kind of locally weighted regression



(Fig. 2.5 (d) in LWLS)

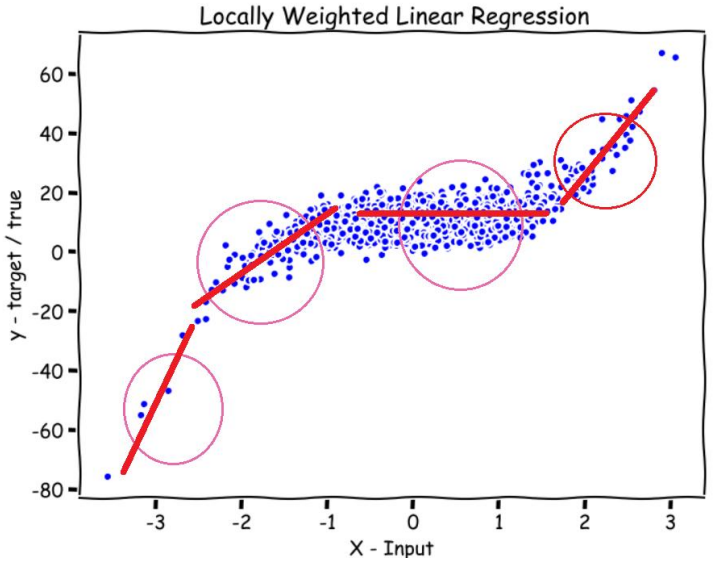
- Generally, locally weighted regression
  - Local regression
  - Piecewise constant, linear, quadratic, etc.
  - Typically weighted by distance from query

Locally Weighted Linear Regression

- Linear regression: $f(\boldsymbol{x}) = w_0 + \sum_{i=1}^{p} w_i x_i$
- Global fitting, minimizing squared error: $\min_{\boldsymbol{w}} \sum_{\boldsymbol{x} \in \boldsymbol{X}} \left(y - f(\boldsymbol{x})\right)^2$
- Locally weighted fitting, minimizing distance-weighted squared error around $\boldsymbol{x}$

$$\min_{\boldsymbol{w}} \sum_{i \in \mathcal{N}_k(\boldsymbol{x})} \left(y^{(i)} - f\left(\boldsymbol{x}^{(i)}\right)\right)^2 K\left(d\left(\boldsymbol{x}, \boldsymbol{x}^{(i)}\right)\right),$$

where $K(\cdot)$ is a kernel function, usually decreasing



(Figure from https://towardsdatascience.com/locally-weighted-linear-regression-in-python-3d324108efbf)

Lazy vs. Eager

- Lazy methods, e.g., kNN and locally weighted regression, defer the processing of training data till test data is presented
  - They can consider the test example when making prediction
- Eager methods, e.g., linear regression, processes training data and generalizes beyond training data before observing any test data
  - Predictions for all test examples are already made during training

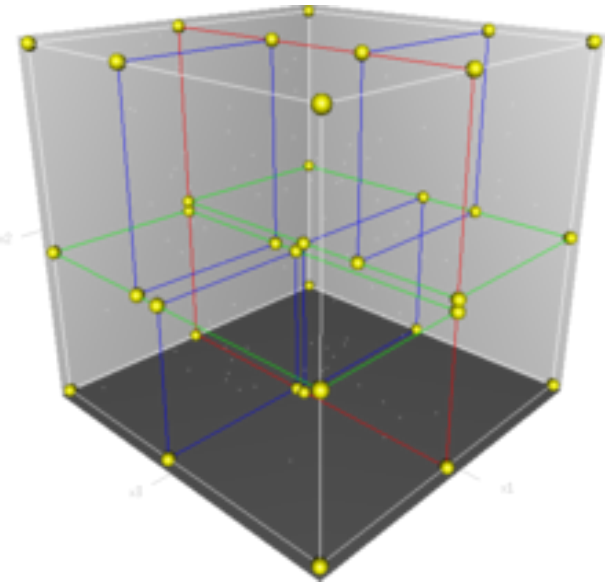
- Lazy methods represent the target function by a combination of many local approximations
  - E.g., locally weighted linear regression (lazy) learns a piecewise linear function, while linear regression (eager) learns a globally linear function

Computational Efficiency

- The inference process of lazy methods is slow, as the processing of training data is delayed till then
- A naïve kNN algorithm would need to traverse the training set to find nearest neighbors: O(N)

- How to make the search more efficient?

- kd-tree (k-dimensional tree) algorithm
  - Binary tree
  - Each layer chooses one attribute to split
    - E.g., from root down: x -> y -> z -> x -> y -> z -> …
  - Leaves store training examples
  - Search: O(logN)



(Figure from https://en.wikipedia.org/wiki/K-d_tree)

Radial Basis Functions (RBF)

- Linear combination of multiple RBF kernels

$$f(x) = w_0 + \sum_{i=1}^{k} w_i K_i\big(d(\boldsymbol{x}^{(i)}, \boldsymbol{x})\big),$$

where $\boldsymbol{x}^{(i)}$ are k representative examples (called centers), and $K_i(\cdot)$ is a kernel,

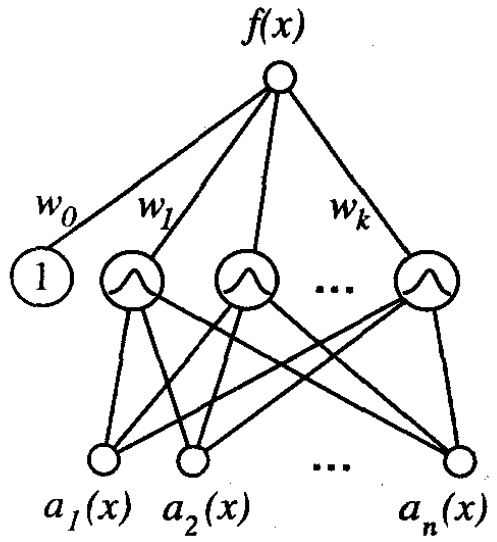
e.g., Gaussian kernel $K_i\big(d(\boldsymbol{x}^{(i)}, \boldsymbol{x})\big) = e^{-\frac{d^2(\boldsymbol{x}^{(i)}, \boldsymbol{x})}{2\sigma_i^2}}$.

- This can be viewed as a two-layer network

Output layer
(linear combination)

Gaussian kernel layer
(Gaussian function)

Input layer



Training Procedure

Step 2: weights are learned through linear regression

Step 1: centers and variances are optimized to have a good coverage of training set

Radial Basis Functions (RBF)

- We can set the same variance for all kernels
  - Define one kernel at each training example
    - Training set can be fit exactly
    - Slow
  - Define one kernel at a subset of training examples
    - Training set cannot be fit exactly
    - More efficient
    - How to choose this subset?
      - Uniformly distributing centers
      - Random sample of training examples
      - Finding prototypes through clustering

Summary

- Instance-based learning delays processing of training examples until test examples are presented

- They can model complex target functions through a combination of many local approximations
  - kNN learns piecewise constant functions
  - Locally weighted regression generalizes over kNN through an explicit local approximation (e.g., linear regression)

- RBF networks is an eager method that incorporates the idea of representing the globally complex target function through a combination of many local RBF kernel functions