Lecture 24 EE 559 Apr 17, 2024

Machine Learning I: Supervised Methods

B. Keith Jenkins

Announcements

- Slido event code: 4252576
- Homework 8 is due Mon. 4/22

Reading

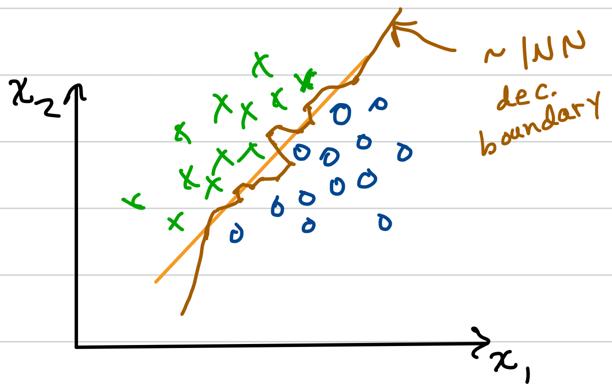
- Bishop 4.2.0-4.2.3 (parameter estimation)
  - parts that are covered in lecture

Today's lecture

- Comments on kNN and KDE
- Theoretically optimal MSE regression
- Density estimation techniques for regression
  - k nearest neighbors (kNN)
- Comments on kNN implementation
- d.o.f. and constraints in density estimation
  - "curse of dimensionality"
  - how bad is it, really?
- Parameter estimation (part 1)
  - Problem set-up and notation $\rightarrow$ deferred

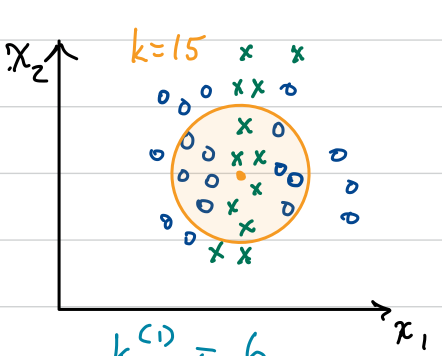
Comments on kNN and KDE for classification

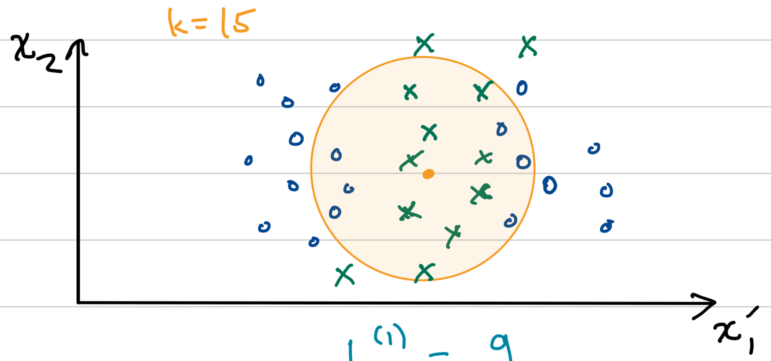
1. Normalization matters!
   - Sensitive to differing scale sizes of different features



x S_1

o S_2





$k_{15}^{(1)} = 6$

$k_{15}^{(2)} = 9$

$\Rightarrow \underline{x} \in \Gamma_2$

$x_1' \approx 2x_1$

$k_{15}^{(1)} = 9$

$k_{15}^{(2)} = 6$

$\Rightarrow \underline{x} \in \Gamma_1$

2. Choice of k (kNN) or kernel width h (KDE) affects amount of smoothing and resolution (i.e. underfit / overfit).

3. Can be computationally slow for large N or large D.
   - There are algorithms to speed up computation (e.g., tree search methods), usually giving an approximate result.

Regression: Theoretically minimum MSE predictor

→ Analogous to Bayes min. error for classification

$$J = \overset{\text{Total MSE}}{\text{TMSE}} = E_{\underline{x},y}\{[\hat{y}(\underline{x}) - y(\underline{x})]^2\} \triangleq \iint (\hat{y}-y)^2 \, p(\underline{x},y)\, d\underline{x}\, dy$$

$$= \iint (\hat{y}-y)^2 \, p(y|\underline{x})\, p(\underline{x})\, d\underline{x}\, dy$$

$$= \int \underbrace{\left[\int (\hat{y}(\underline{x})-y(\underline{x}))^2 \, p(y|\underline{x})\, dy\right]}_{\text{Conditional MSE (CMSE)}} p(\underline{x})\, d\underline{x}$$

Outer integral: integrates over all $\underline{x}$. For each point $\underline{x}$: minimize the inner integral (it's non-negative), to minimize the contribution to TMSE.

Let $\text{CMSE} \triangleq E_{y|\underline{x}}\{[\hat{y}(\underline{x}) - y(\underline{x})]^2\} \triangleq \int (\hat{y}-y)^2 \, p(y|\underline{x})\, dy$

$\Rightarrow$ For each $\underline{x}$, choose $\hat{y}$ that minimizes CMSE

$\therefore$, $\quad J' = \text{CMSE} = \int (\hat{y}-y)^2 \, p(y|\underline{x})\, dy$, $\quad \hat{y}(\underline{x}) = \underset{\hat{y}}{\operatorname{argmin}}\, J'$

$$\frac{\partial}{\partial \hat{y}} J' = \int \frac{\partial}{\partial \hat{y}} (\hat{y}-y)^2 p(y|\underline{x})\, dy$$

$$= \int 2(\hat{y}-y)\, p(y|\underline{x})\, dy = 0$$

$$\Rightarrow \quad \hat{y} \underbrace{\int p(y|\underline{x})\, dy}_{=1} = \int y\, p(y|\underline{x})\, dy$$

$$\Rightarrow \quad \hat{y} = E\{y \mid \underline{x}\} \qquad \text{(MMSE solution)}$$

(1) $$\boxed{\hat{y}(\underline{x}) = \int y(\underline{x})\, p(y \mid \underline{x})\, dy = E\{y \mid \underline{x}\}}$$

Theoretically optimal prediction for minimizing TMSE on unknowns.

So, we want to:

(i) Estimate $p(y \mid \underline{x})$ from the data (directly or indirectly)

(ii) For any given $\underline{x}$, obtain $\hat{y}(\underline{x}) = E\{y \mid \underline{x}\}$.

kNN Regression

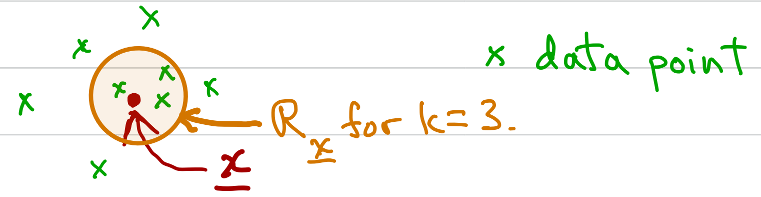
$\longrightarrow$ Est. $E\{y|\underline{x}\}$.

Use kNN to choose $R_{\underline{x}}$: $R_{\underline{x}}$ just large enough to enclose the k data points nearest to $\underline{x}$. Call them $\underline{x}_n \in \mathcal{X}$, and their indeces n are $n \in \mathcal{N}$.

$\mathcal{X}$ is the set of k data points $\underline{x}_n$ closest to $\underline{x}$.

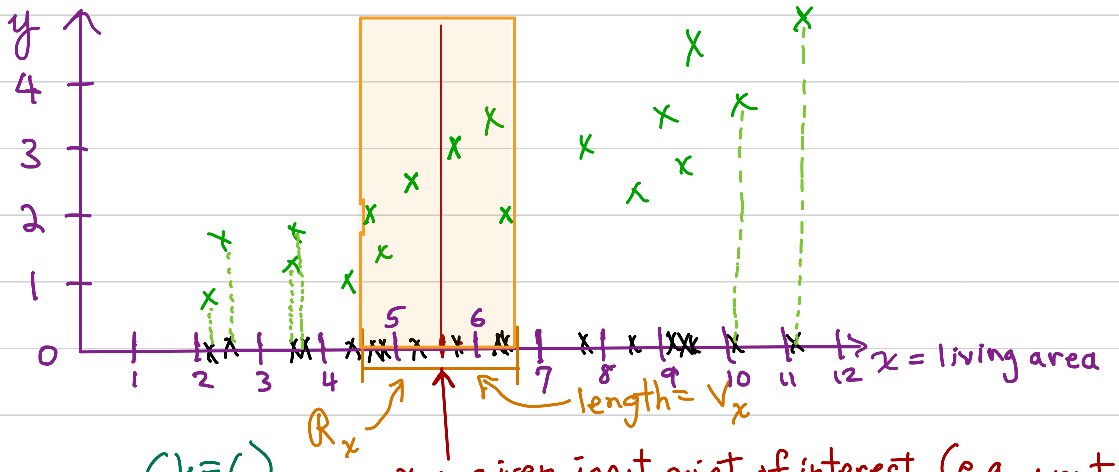
$\mathcal{N}$ is the set of indeces n of the data pts in $\mathcal{X}$.

2D $\underline{x}'$ space:



[

1D feature space:



⑤

(k=6)

x = given input point of interest (e.g., want to predict $\hat{y}(\underline{x})$).

We can estimate $E\{y|\underline{x}\}$ directly, using data in $\mathcal{X}$ (shaded region).

Simplest: take sample mean of y_n for all $n \in \mathcal{N}$:

$$\boxed{\hat{y}(\underline{x}) = \frac{1}{N_{\mathcal{N}}} \sum_{n \in \mathcal{N}} y_n.} \quad (2)$$

, $N_{\mathcal{N}}$ = # data pts. in $\mathcal{X}$ (shaded column)

Or, because the $\underline{x}_n$ may have varying distances to $\underline{x}$, take a weighted sample mean:

$$(3) \qquad \boxed{\hat{y}(\underline{x}) = \frac{1}{\sum_{n\in\eta} w_n'} \sum_{n\in\eta} w_n' y_n} \; , \qquad w_n' \geq 0 \;\; \forall n.$$

Choices of w_n' : kernel fcn.. $K(\cdot)$

$$w_n' = K(\underline{x} - \underline{x}_n)$$ (kernel function that represents similarity between $\underline{x}, \underline{x}_n$). (or window fcn. $\Delta(\underline{x}-\underline{x}_n)$)

$\Rightarrow$ <u>kernel regression</u>, <u>kernel smoothing</u>, or <u>Nadaraya-Watson model</u>.

Examples:

$$K(\underline{x}-\underline{x}_n) = 1 - \frac{d(\underline{x},\underline{x}_n)}{d_{max}}, \quad d_{max} = \max_{n\in\eta} d(\underline{x},\underline{x}_n)$$

$= \text{dist. to } k^{\underline{th}}$ nearest neighbor

Note that $w'_{k^{th} N.N.} = 0$

So $1 - \dfrac{d(\underline{x},\underline{x}_n)}{(1+\epsilon)\, d_{max}}$, $\epsilon > 0$, may be better.

or $d_{max} = \text{dist. to } (k+1)^{\underline{th}}$ nearest neighbor.

$$K(\underline{x}-\underline{x}_n) = d^{-1}(\underline{x},\underline{x}_n)$$ (blows up for $\underline{x}_n \approx \underline{x}$)

So $\dfrac{1}{a + d(\underline{x},\underline{x}_n)}$, $a > 0$ may be better.

- Note: $d(\underline{x}, \underline{x}_n)$ in the kernel function could be:

  1. Euclidean: $d_E(\underline{x}, \underline{x}_n) = \|\underline{x} - \underline{x}_n\|_2$

  2. Mahalanobis: $d_M(\underline{x}, \underline{x}_n) = \left[(\underline{x} - \underline{x}_n)^T \underline{\underline{\Sigma}}_{\underline{x}}^{-1} (\underline{x} - \underline{x}_n)\right]^{1/2}$

     Requires estimating $\underline{\underline{\Sigma}}_{\underline{x}}^{-1}$ from data:

     Option 1. Assume $\underline{\underline{\Sigma}}_x = \text{diag}\{\sigma_1^2, \sigma_2^2, \cdots, \sigma_D^2\}$
     and use sample variances of training data:

$$\left.\begin{aligned} \hat{\sigma}_i^2 &= \frac{1}{N_{Tr}-1} \sum_{n=1}^{N_{Tr}} (x_{ni} - \bar{x}_{ni})^2 \\ \bar{x}_i &= \frac{1}{N} \sum_{n=1}^{N_{Tr}} x_{ni} \end{aligned}\right\} \quad i = 1, 2, \cdots, D$$

     Option 2. Estimate full cov. matrix:

$$\hat{\underline{\underline{\Sigma}}} = \frac{1}{N_{Tr}-1} \sum_{n=1}^{N_{Tr}} (\underline{x}_n - \underline{\bar{x}})(\underline{x}_n - \underline{\bar{x}})^T$$

$$\underline{\bar{x}} = \frac{1}{N_{Tr}} \sum_{n=1}^{N_{Tr}} \underline{x}_n$$

- Comment

  For defining the k nearest neighbors (for R_n or $R_{\underline{x}}$), have similar choice of distance functions:

  d_E and d_M are common choices.

Comments on kNN implementation (regression & classification)

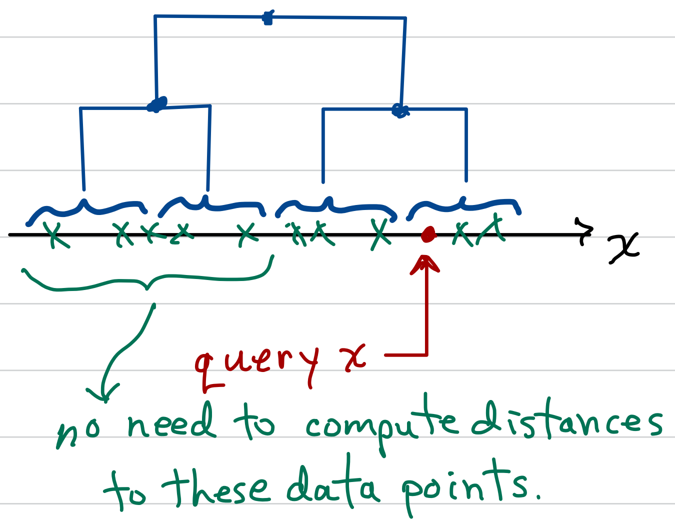
1. For large N and D, kNN can be slow.

   For one query (one point $\underline{x}$),
   using straight-forward ("brute force") algorithm,
   - distances to all data points need to be computed: $O(ND)$
   - find NN: $O(N)$

2. There exist faster sort algorithms

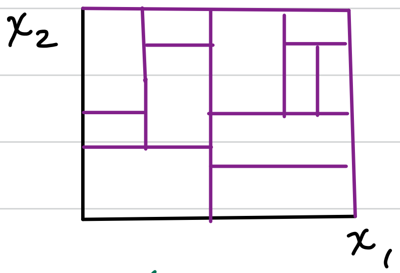
   → involve some pre-computing on the data.
   e.g., grow a tree that partitions the data into local groups.

   → the NN algorithm uses location (or distances) of groups to avoid unnecessary distance calculations.
   e.g., if a group is too far from $\underline{x}$, all of its data points are too far from $\underline{x}$.



(i) K-D Tree algorithms

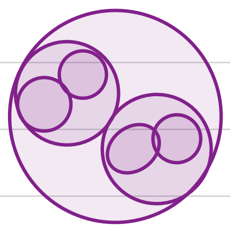
Grows a K-dimensional tree in feature space;
each leaf node is a hyper-rectangle



find NN, including necessary distance calculations: $O(D \log N)$
→ Much faster vs. N, same dependence on D

(ii) Ball trees

Each leaf node is a hypersphere.
Requires more pre-computation.



For large D, is faster than K-D Tree.
Actual savings in computation time is
very data-dependent.

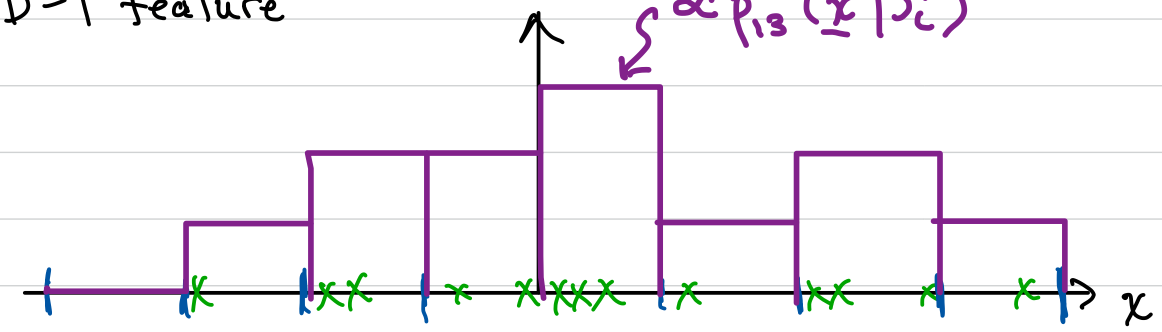
For python functions and more explanation:
scikit-learn user guide, 1.6 Nearest Neighbors.

d.o.f. and constraints / assumptions in density est. techniques

Estimate: $p(\underline{x} \mid S_i)$

Consider histogram method of estimation:

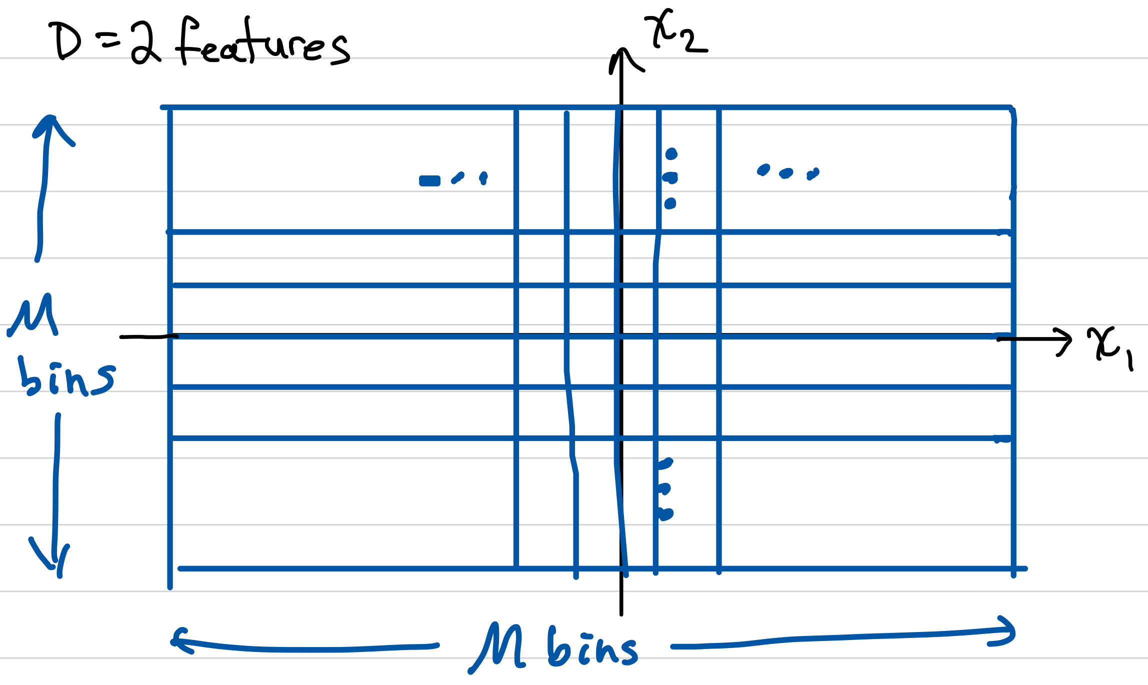
D=1 feature



d.o.f. = ? = #bins = 8.

generally: M bins, d.o.f. = M.

D=2 features



⑤ d.o.f. = M^2

D features: d.o.f. = ? = M^D

# constraints = N = \# data pts.

Ex:

$D = 100$

$M = 10$ bins / feature

d.o.f. = $M^D = 10^{100}$.

$\Rightarrow$ "Curse of dimensionality" !

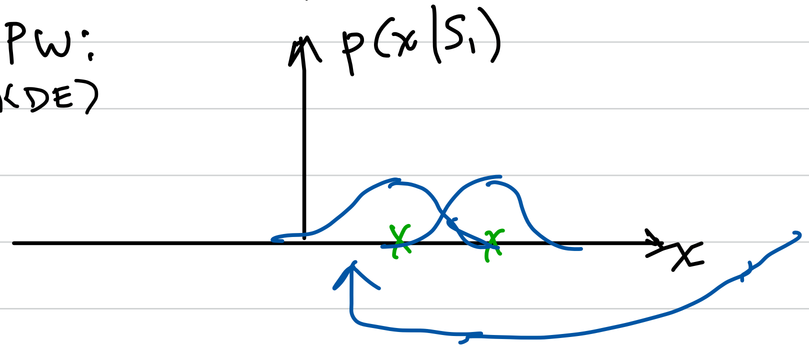
Fortunately:

PW and kNN $\rightarrow$ don't scale as bad with D (empirical evidence).

Why?

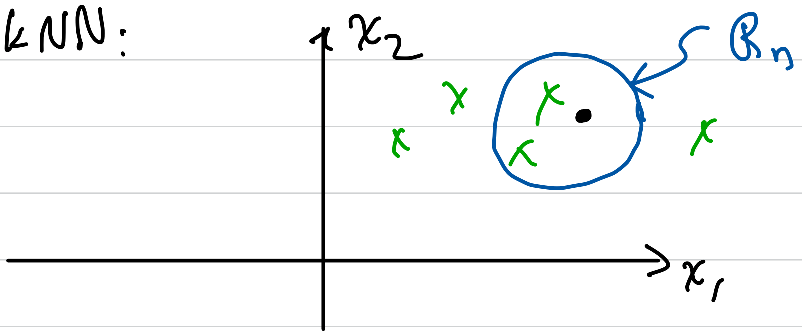
1. Assumptions, constraints, averaging / regularization

PW:
(KDE)



Assumes $p(\underline{x}|S_i)$ varies not too quickly in the vicinity of each $\underline{x}_i$.

kNN:



Assumes $p(\underline{x}, S_i) \approx$ const. in R_n.

Both: prediction $\hat{y}(\underline{x})$ is typically based on a sum or averaging over a region around $\underline{x}$, which acts as a sort of regularizer.

2. In regions of very few data pts., typically $p(\underline{x})$ is small, so inaccurate estimates don't hurt classification accuracy very much.

__Density estimation approaches to ML – advantages, disadvantages__

+ Makes very few assumptions on $p(\underline{x}|S_i)$, etc.

can get good est's knowing very little about $p(\underline{x}|S_i)$ a priori.

– Makes very few assumptions on $p(\underline{x}|S_i)$, etc.

$\rightarrow$ Has high d.o.f., especially for large D.

$\rightarrow$ __Parameter estimation techniques:__

Assume we know the functional form of $p(\underline{x}|S_i)$, etc., in terms of some unknown parameters that are learned from the data.

e.g.: $p(\underline{x}|S_i) = N(\underline{x}, \underline{\mu}_i, \underline{\underline{\Sigma}}_i)$ is assumed.

with $\underline{\mu}_i$, $i=1,2,\cdots,C$ unknown
$\rightarrow$ learned from the data.

$\underline{\underline{\Sigma}}_i$, $i=1,2,\cdots,C$ known or assumed.

5 d.o.f. in this case ?

$\underline{\mu}_i$: DC

$\rightarrow$ __much__ better!