Advanced Introduction to Machine Learning

— Spring Quarter, Week 7 —

https://canvas.uw.edu/courses/1372141

Prof. Jeff Bilmes

University of Washington, Seattle
Departments of: Electrical & Computer Engineering, Computer Science & Engineering
http://melodi.ee.washington.edu/~bilmes

May 11th/13th, 2020

Announcements

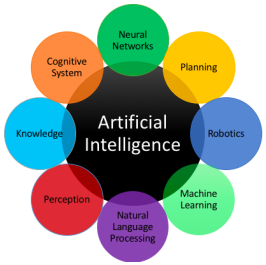
- HW3 is due May 15th, 6:00pm via our assignment dropbox (`https://canvas.uw.edu/courses/1372141/assignments`).
- Virtual office hours this week, Thursday night at 10:00pm via zoom (same link as class).

Class Road Map

- W1(3/30,4/1): What is ML, Probability, Coins, Gaussians and linear regression, Associative Memories, Supervised Learning
- W2(4/6,4/8): More supervised, logistic regression, complexity and bias/variance tradeoff
- W3(4/13,4/15): Bias/Variance, Regularization, Ridge, CrossVal, Multiclass
- W4(4/20,4/22): Multiclass classification, ERM, Gen/Disc, Naïve Bayes
- W5(4/27,4/29): Lasso, Regularizers, Curse of Dimensionality
- W6(5/4,5/6): Curse of Dimensionality, Dimensionality Reduction, k-NN
- W7(5/11,5/13): k-NN, LSH, DTs, Bootstrap/Bagging, Boosting & Random Forests, GBDTs
- W8(5/18,5/20): Graphs; Graphical Models (Factorization, Inference, MRFs, BNs);
- W9(5/27,6/1): Learning Paradigms; Clustering; EM Algorithm;
- W10(6/3,6/8): Spectral Clustering, Graph SSL, Deep models, (SVMs, RL); The Future.

Last lecture is 6/8 since 5/25 is holiday (or we could just have lecture on 5/25).

Class (and Machine Learning) overview

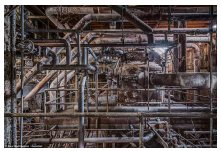


1. Introduction
- What is ML
- What is AI
- Why are we so interested in these topics right now?

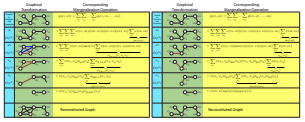
2. ML Paradigms/Concepts
- Overfitting/Underfitting, model complexity, bias/variance
- size of data, big data, sample complexity
- ERM, loss + regularization, loss functions, regularizers
- Supervised, unsupervised, and semi-supervised learning;
- reinforcement learning, RL, multi-agent, planning/control
- transfer and multi-task learning
- federated and distributed learning
- active learning, machine teaching
- self-supervised, zero/one-shot, open-set learning

3. Dealing with Features
- dimensionality reduction, PCA, LDA, MDS, T-SNE, UMAP
- Locality sensitive hashing (LSH)
- feature selection
- feature engineering
- matrix factorization & feature engineering
- representation learning

4. Evaluation
- accuracy/error, precision/recall, ROC, likelihood/posterior, cost/utility, margin
- train/eval/test data splits
- n-fold cross validation
- method of the bootstrap



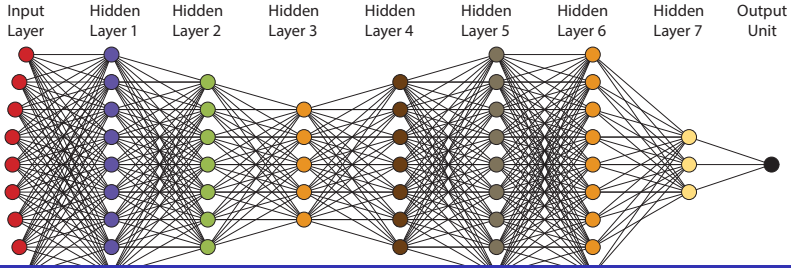
5. Optimization Methods
- Unconstrained Continuous Optimization: (stochastic) gradient descent (SGD), adaptive learning rates, conjugate gradient, 2nd order Newton
- Constrained Continuous Optimization : Frank-Wolf (conditional gradient descent), projected gradient, linear, quadratic, and convex programming
- Discrete optimization - greedy, beam search, branch-and-bound, submodular optimization.



6. Inference Methods
- probabilistic inference
- MLE, MAP
- belief propagation
- forward/backpropagation
- Monte Carlo methods

7. Models & Representation
- linear least squares, linear regression, logistic regression, sparsity, ridge, lasso
- generative vs. discriminative models
- Naive Bayes
- k-nearest neighbors
- clustering, k-means, k-mediods, EM & GMMs, single linkage
- decision trees and random forests
- support vector machines, kernel methods, max margin
- perceptron, neural networks, DNNs
- Gaussian processes
- Bayesian nonparametric methods
- ensemble methods
- the bootstrap, bagging, and boosting
- graphical models
- time-series, HMMs, DBNs, RNNs, LSTMs, Attention, Transformers
- structured prediction
- grammars (as in NLP)

12. Other Techniques
- compressed sensing
- submodularity, diversity/homogeneity modeling

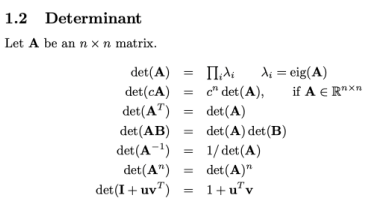


8. Philosophy, Humanity, Spirituality
- artificial intelligence (AI)
- artifical general intelligence (AGI)
- artificial intelligence vs. science fiction

9. Applications
- computational biology
- social networks
- computer vision
- speech recognition
- natural language processing
- information retrieval
- collaborative filtering/matrix factorization

10. Programming
- python
- libraries (e.g., NumPy, SciPy, matplotlib, scikit-learn (sklearn), pytorch, CNTK, Theano, tensorflow, keras, H2O, etc.
- HPC: C/C++, CUDA, vector processing

11. Background
- linear algebra
- multivariate calculus
- probability theory and statistics
- information theory
- mathematical (e.g., convex) optimization



1.2 Determinant

Let $\mathbf{A}$ be an $n \times n$ matrix.

$$\begin{aligned}
\det(\mathbf{A}) &= \textstyle\prod_i \lambda_i \qquad \lambda_i = \mathrm{eig}(\mathbf{A}) \\
\det(c\mathbf{A}) &= c^n \det(\mathbf{A}), \qquad \text{if } \mathbf{A} \in \mathbb{R}^{n\times n} \\
\det(\mathbf{A}^T) &= \det(\mathbf{A}) \\
\det(\mathbf{AB}) &= \det(\mathbf{A})\det(\mathbf{B}) \\
\det(\mathbf{A}^{-1}) &= 1/\det(\mathbf{A}) \\
\det(\mathbf{A}^n) &= \det(\mathbf{A})^n \\
\det(\mathbf{I} + \mathbf{uv}^T) &= 1 + \mathbf{u}^T\mathbf{v}
\end{aligned}$$

Using 2D to represent High-D as if 2D was High-D

- Relationship between unit-radius ($r = 1$) sphere and unit volume (side length $= 1$) cube as the dimensions grow if they were to be also true in 2D.
- Note that for cube, as m gets higher, distance from center to face stays at $1/2$ but distance from center to vertex grows as $\sqrt{m}/2$.
- For $m = 4$ vertex distance is $\sqrt{m}/2 = 1$.

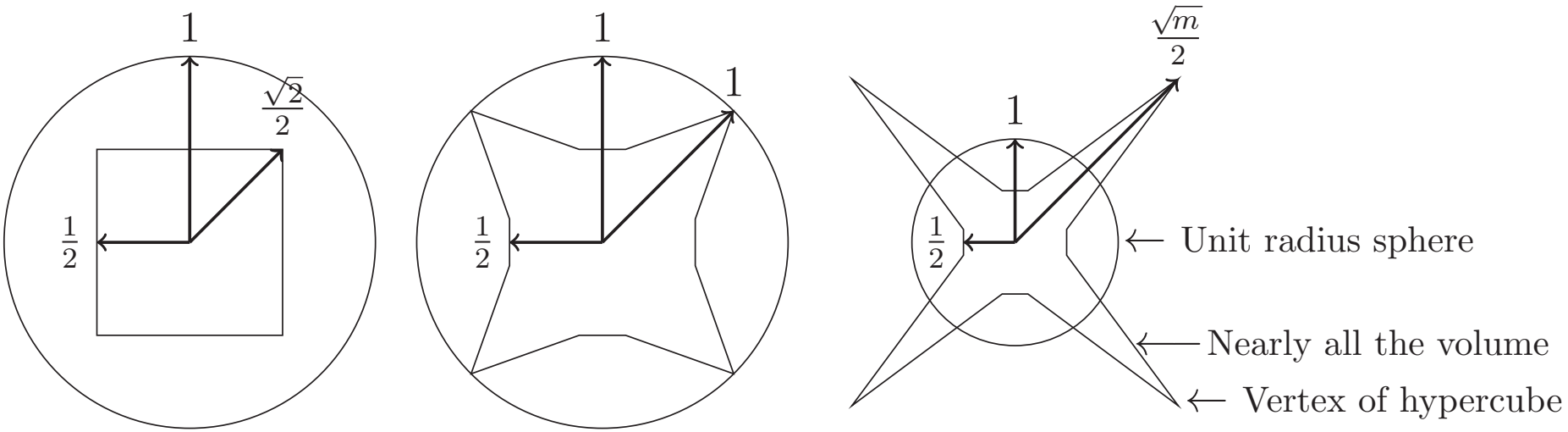
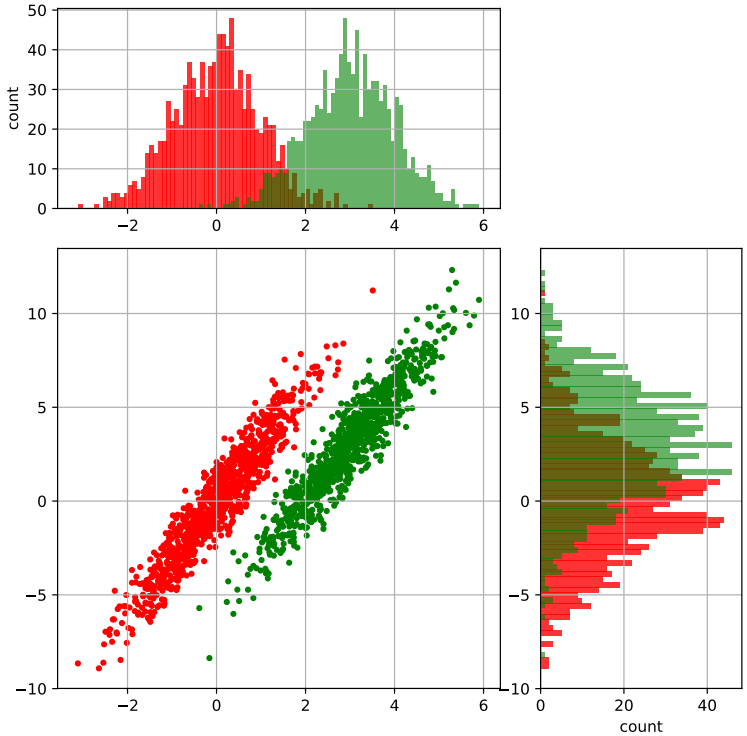


Illustration of the relationship between the sphere and the cube in 2, 4, and m-dimensions (from Blum, Hopcroft, & Kannan, 2016).

The Blessing of Dimensionality

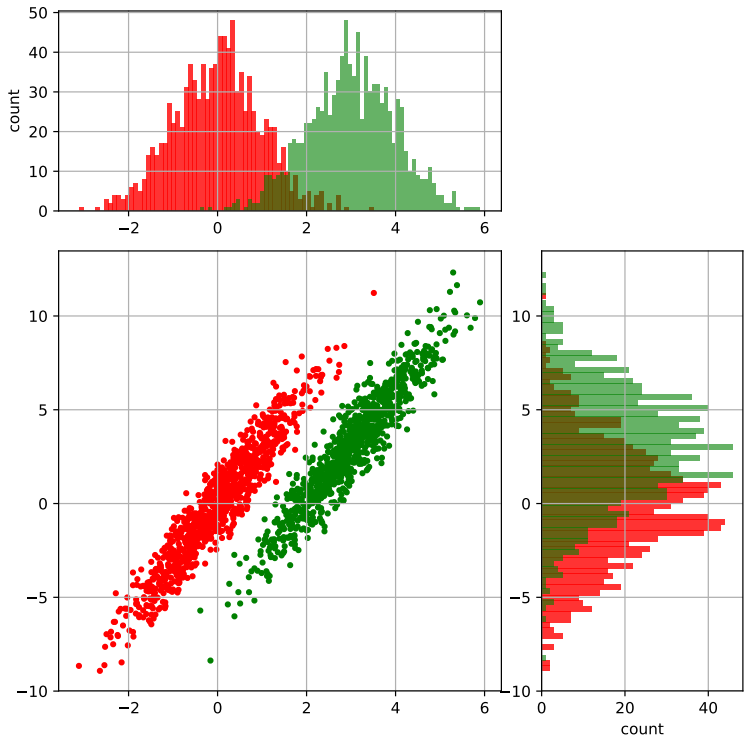
- More dimensions can help to distinguish categories, make pattern recognition easier (and even possible). Recall Voronoi tessellation



- Support vector machines (SVMs) can find and exploit data patterns extant only in extremely high (or even infinite) dimensional space!!!
- For still more blessings, see "High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality", David L. Donoho

Feature Selection vs. Dimensionality Reduction

- We've already seen that feature selection is one strategy to reduce feature dimensionality.
- Feature selection, each feature is either selected or not, all or nothing.
- Other dimensionality reduction strategies take the input $x \in \mathbb{R}^m$ and encode each x into $\mathbf{e}(x) \in \mathbb{R}^{m'}$, a lower dimensional space, $m' < m$.
- Core advantage: If $U = \{1, 2, \ldots, m\}$, there may be no subset $A \subseteq U$ such that x_A will work well. There might be a combination $\mathbf{e}(x)$ that works well. On the right, simple linear combination $a_1 x_1 + a_2 x_2$ works quite well.



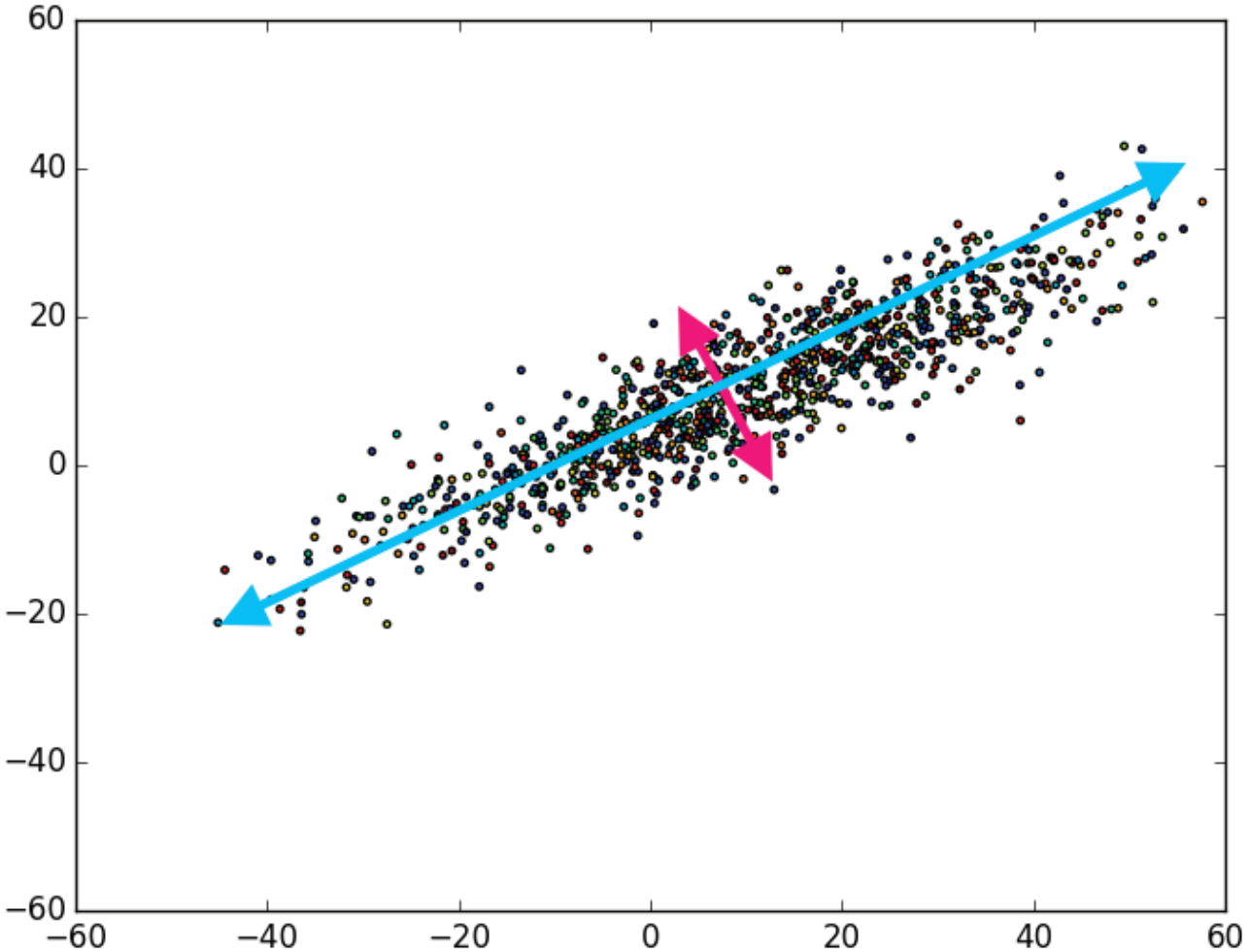
PCA: Minimizing reconstruction error $m' < m$

- Let u_i, $i \in [m]$ be a set of orthonormal vectors in $\mathbb{R}^m$.
- Basis decomposition: any sample $x^{(i)}$ can be written as $x^{(i)} = \sum_{j=1}^{m} \alpha_{i,j} u_j = \sum_{j=1}^{m} \left\langle x^{(i)}, u_j \right\rangle u_j$.
- Suppose we decide to use only $m' < m$ dimensions, so $\tilde{x}^{(i)} = \sum_{j=1}^{m'} \left\langle x^{(i)}, u_j \right\rangle u_j$.
- Reconstruction error

$$J_{m'} = \frac{1}{n} \sum_{i=1}^{n} \|x^{(i)} - \tilde{x}^{(i)}\|_2^2 \tag{7.1}$$

- To minimize, can be shown that we should choose u_i to be the eigenvector of S corresponding to the ith largest eigenvalue.
- Let $W_{m'}$ be the matrix of column eigenvectors of S sorted decreasing by eigenvalue, and $W(:, 1 : m')$ the $m \times m'$ matrix of the first m' columns.
- Then $XW_{m'}$ projects down to the first m' principle components, and is also widely known as the Karhunen-Loève transform (KLT)

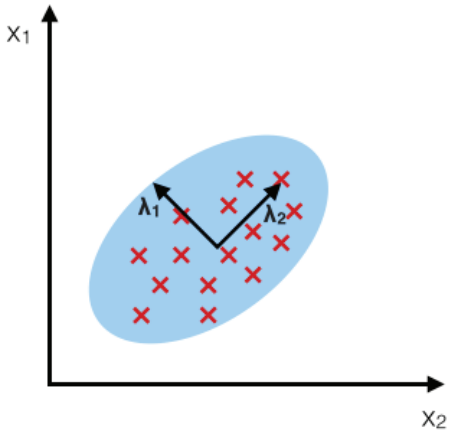
Example: 2D and two principle directions



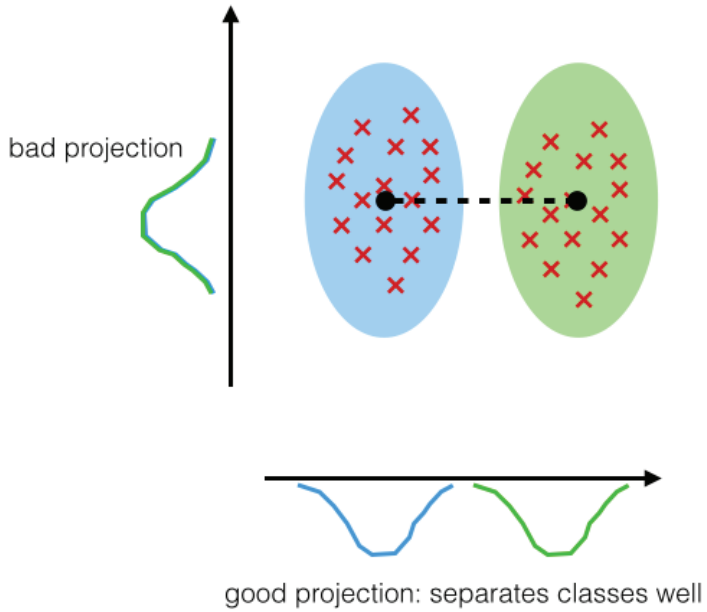
PCA vs. LDA

From Bishop 2006

PCA:
component axes that maximize the variance



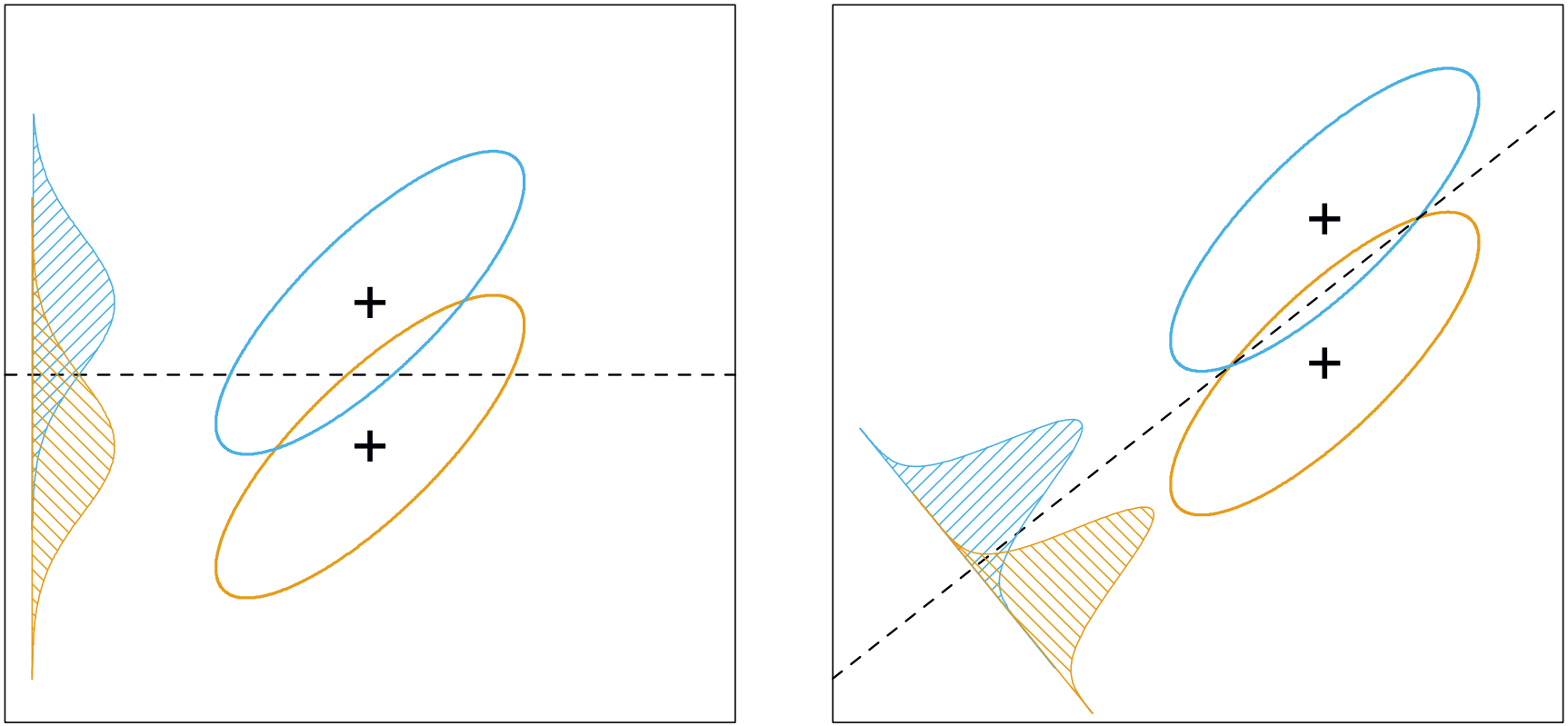
LDA:
maximizing the component axes for class-separation



PCA vs. Linear Discriminant Analysis (LDA).

Axis parallel projection vs. LDA

From Hastie et. al., 2009



Axis parallel (left) vs. LDA (right).

Linear Discriminant Analysis (LDA), 2 classes

- Consider class-conditional Gaussian data, so $p(x|y) = \mathcal{N}(x|\mu_y, C_y)$ for mean vectors $\{\mu_y\}_y$ and covariance matrices $\{C_y\}_y$, $x \in \mathbb{R}^m$.

$$p(x|y) = \frac{1}{|2\pi C_y|^{m/2}} \exp\left(-\frac{1}{2}(x-\mu_y)^\intercal C_y^{-1}(x-\mu_y)\right) \tag{7.2}$$

- Two class case $y \in \{0,1\}$ with equal covariances $C_0 = C_1 = C$ (homoscedasticity property) and priors $p(y=0) = p(y=1)$, consider log posterior odds ratio:

$$\log \frac{p(y=1|x)}{p(y=0|x)} = -\frac{1}{2}(x-\mu_1)^\intercal C_1^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_0)^\intercal C_0^{-1}(x-\mu_0) \tag{7.3}$$

$$= (C^{-1}\mu_1 - C^{-1}\mu_0)^\intercal x + \mu_0{}^\intercal C^{-1}\mu_0 - \mu_1{}^\intercal C^{-1}\mu_1 \tag{7.4}$$

$$= \theta^\intercal x + c \tag{7.5}$$

- θ is a projection ($m \times 1$ matrix, linear transformation) down to the 1 dimension that is sufficient for prediction w/o loss.

Linear Discriminant Analysis (LDA), ℓ classes

- Within class scatter matrix

$$S_W = \sum_{c=1}^{\ell} S_c \text{ where } S_c = \sum_{i:i \text{ is class } c} (x^{(i)} - \mu_c)(x^{(i)} - \mu_c)^\intercal \quad (7.4)$$

  and μ_c is the class c mean, $\mu_c = \frac{1}{n_c} \sum_{i \text{ is of class } c} x^{(i)}$
- Between class scatter matrix, n_c = number of samples of class c.

$$S_B = \sum_{c=1}^{\ell} n_c(\mu_c - \mu)(\mu_c - \mu)^\intercal \quad (7.5)$$

- First projection, maximize ratio of variances (recall PCA): $a_1 \in \operatorname{argmax}_{a \in \mathbb{R}^m} \frac{a^\intercal S_B a}{a^\intercal S_W a}$. Second orthogonal to first, $a_2 \in \operatorname{argmax}_{a \in \mathbb{R}^m : \langle a, a_1 \rangle = 0} \frac{a^\intercal S_B a}{a^\intercal S_W a}$, and so on.
- In general, directions of greatest variance of the matrix $S_W^{-1} S_B$, maximizes the between class spread relative to within-class spread. I.e., we can do PCA on $S_W^{-1} S_B$ to get LDA reduction.
- LDA dimensionality reduction: find $\ell' \times m$ matrix linear transform on x, project to subspace corresponding to ℓ' greatest eigenvalues of $S_W^{-1} S_B$.

Uniformly at random unit vectors on unit sphere

- Let $S_{m-1}(r)$ indicate surface "area" of m-dimensional sphere radius r.
- Define uniform distribution $U_{S_{m-1}(1)}$ on $S_{m-1}(1)$ and independently draw two random vectors $x, y \sim U_{S_{m-1}(1)}$, hence $\|x\|_2 = \|y\|_2 = 1$
- Two vectors are orthogonal if $\langle x, y \rangle = 0$ and are nearly so if $|\langle x, y \rangle| < \epsilon$ for small ϵ.
- It can be shown that if x, y are independent random vectors uniformly distributed on the m-dimensional sphere, then:

$$\Pr(|\langle x, y \rangle| < \epsilon) > 1 - e^{-m\epsilon^2/2} \tag{7.11}$$

  which means that uniformly at random vectors in high dimensional space are almost always nearly orthogonal!
- One of the reasons why high dimensional random projections preserve information — if two random high dimensional vectors are almost orthogonal, then projections onto them wil also be almost orthogonal.

k-NN classifier, multiclass data

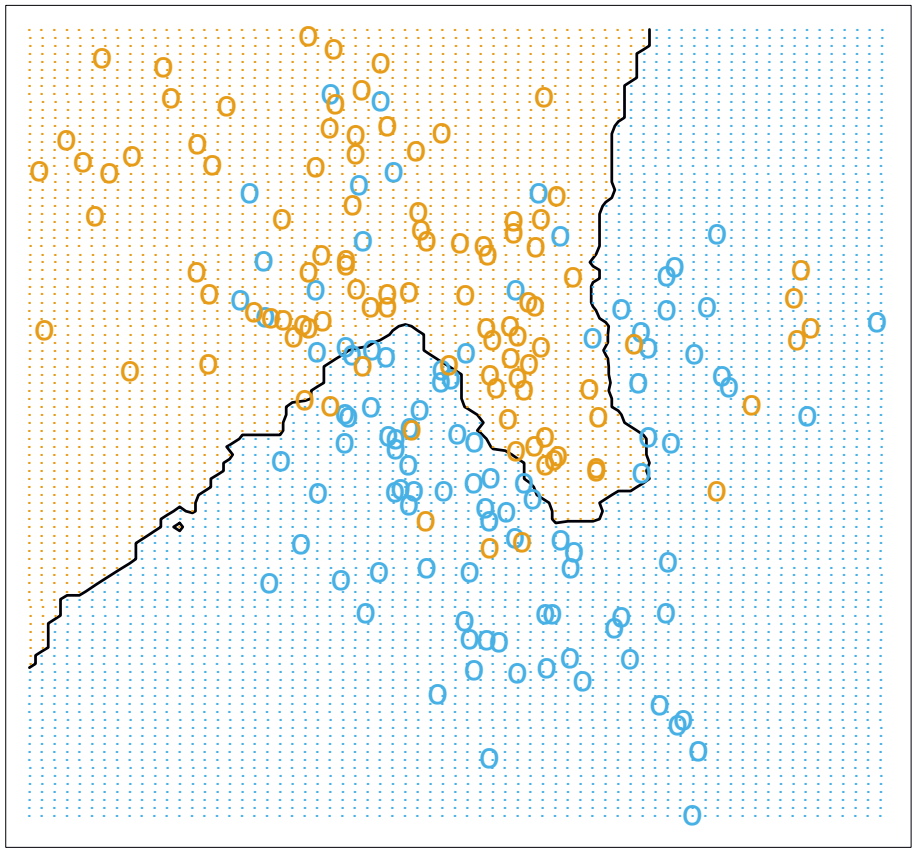
- Let $\mathcal{D} = \left\{x^{(i)}, y^{(i)}\right\}_{i=1}^{n}$ be a training set of n samples (as always).
- Let $\mathcal{N}_k(x, \mathcal{D}) \subseteq \mathcal{D}$ be the subset of $\mathcal{D}$ of size k which are the k nearest neighbors of x. That is $|\mathcal{N}_k(x, \mathcal{D})| = k$ and for any $(x^{(i)}, y^{(i)}) \in \mathcal{N}_k(x, \mathcal{D})$ we have that $d(x^{(i)}, x)$ is no more than the kth farthest data point in $\mathcal{D}$ from x, under distance $d(\cdot, \cdot)$.
- We can then estimate a posterior probability in binary case:

$$p_{k\text{NN}}(y = c|x) = \frac{1}{k} \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{N}_k(x, \mathcal{D})} \mathbf{1}_{\{y^{(i)} = c\}} \tag{7.15}$$

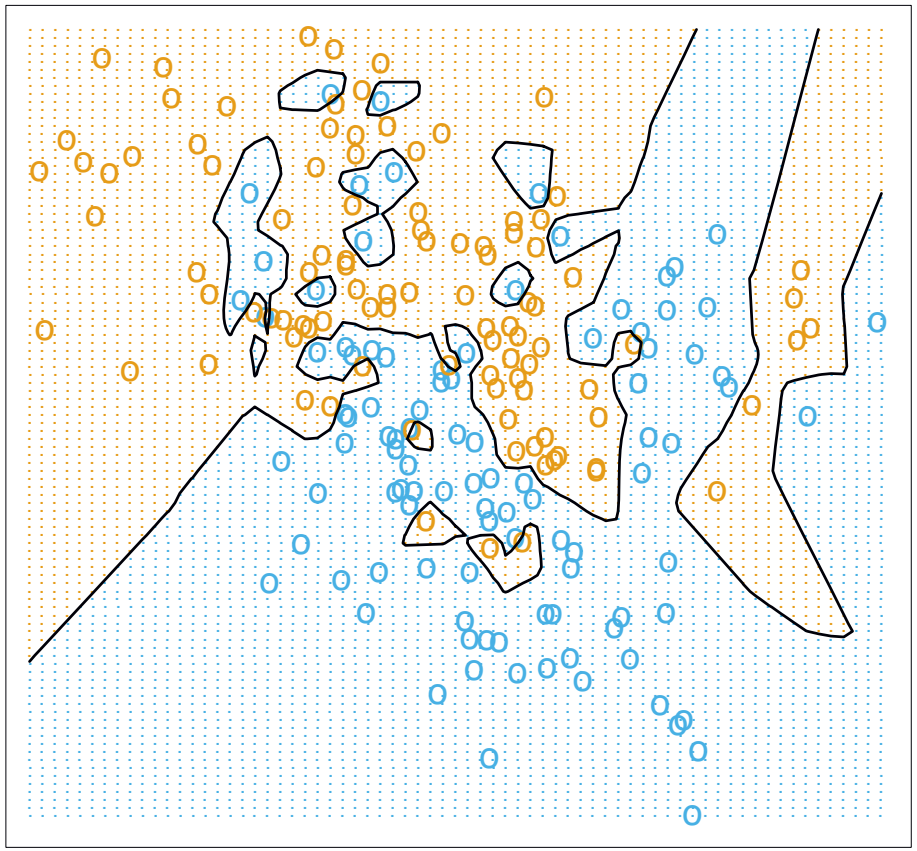
- Valid probability $0 \leq p_{k\text{NN}}(y = c|x) \leq 1$ and $\sum_{c=1}^{\ell} p_{k\text{NN}}(y = c|x) = 1$.

k-NN classifier, various k

From Hastie et. al., 2009



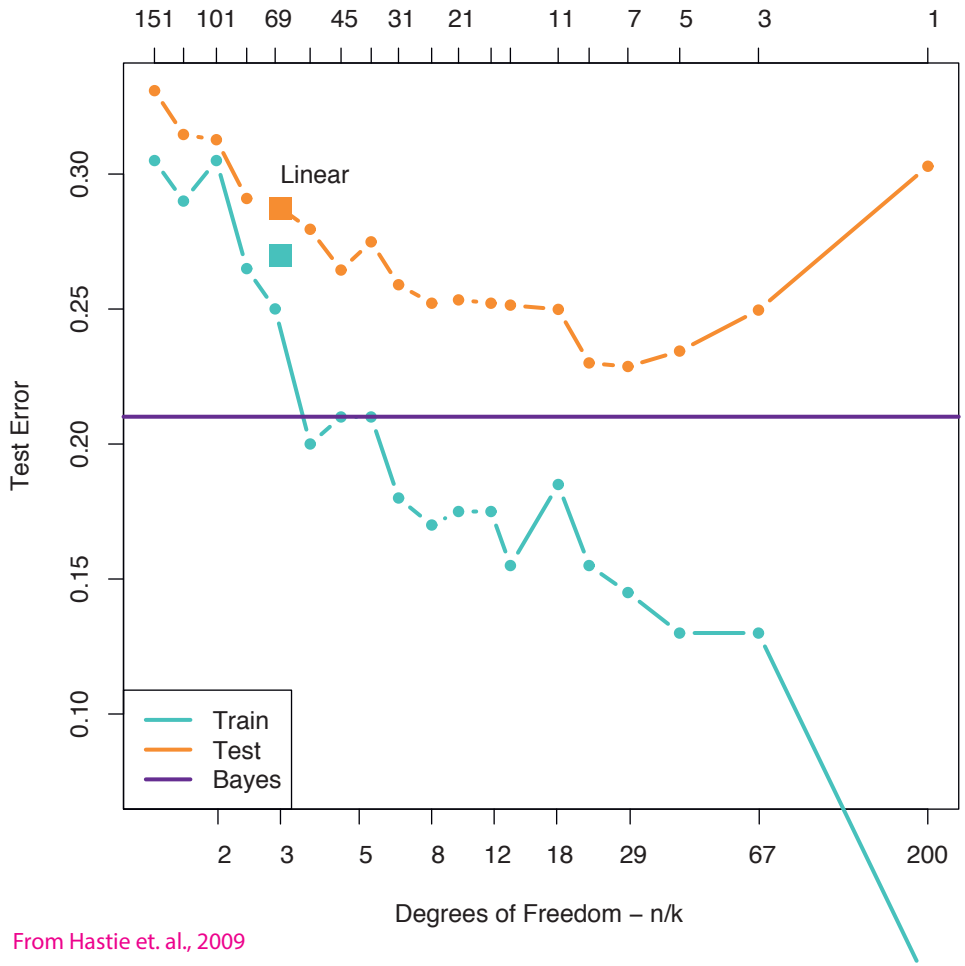
k-NN classifier, with $k = 15$



k-NN classifier, with $k = 1$

k-NN classifiers, effective degrees of freedom

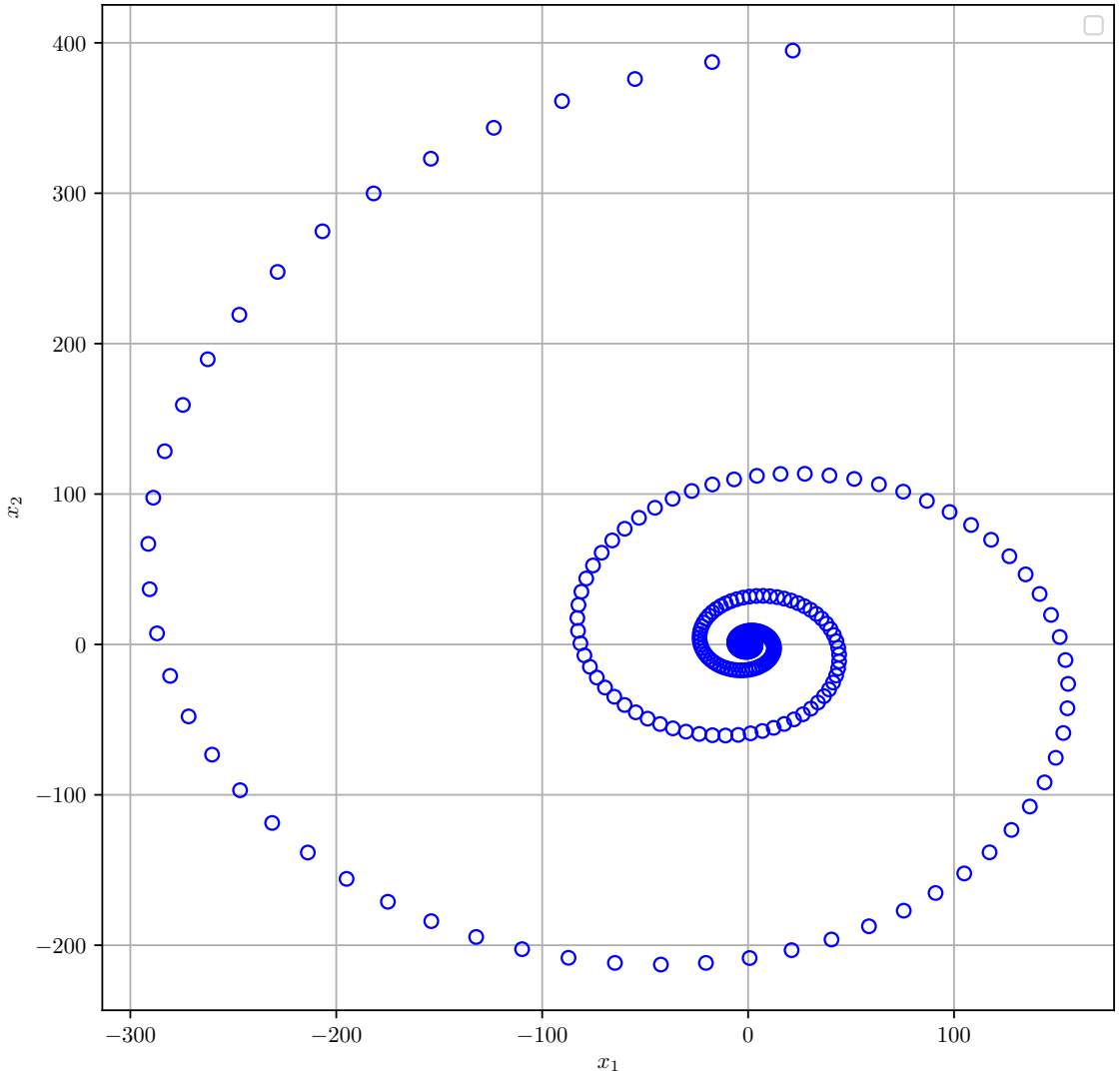
- Hyperparameter is k (similar to λ in ridge or lasso).
- A useful measure is n/k which is effective degrees of freedom in a k-NN classifier. n/k is effective number of neighborhoods within which we might fit a mean.
- We can look at train/test error as a function of n/k and consider relative to logistic regression.



From Hastie et. al., 2009

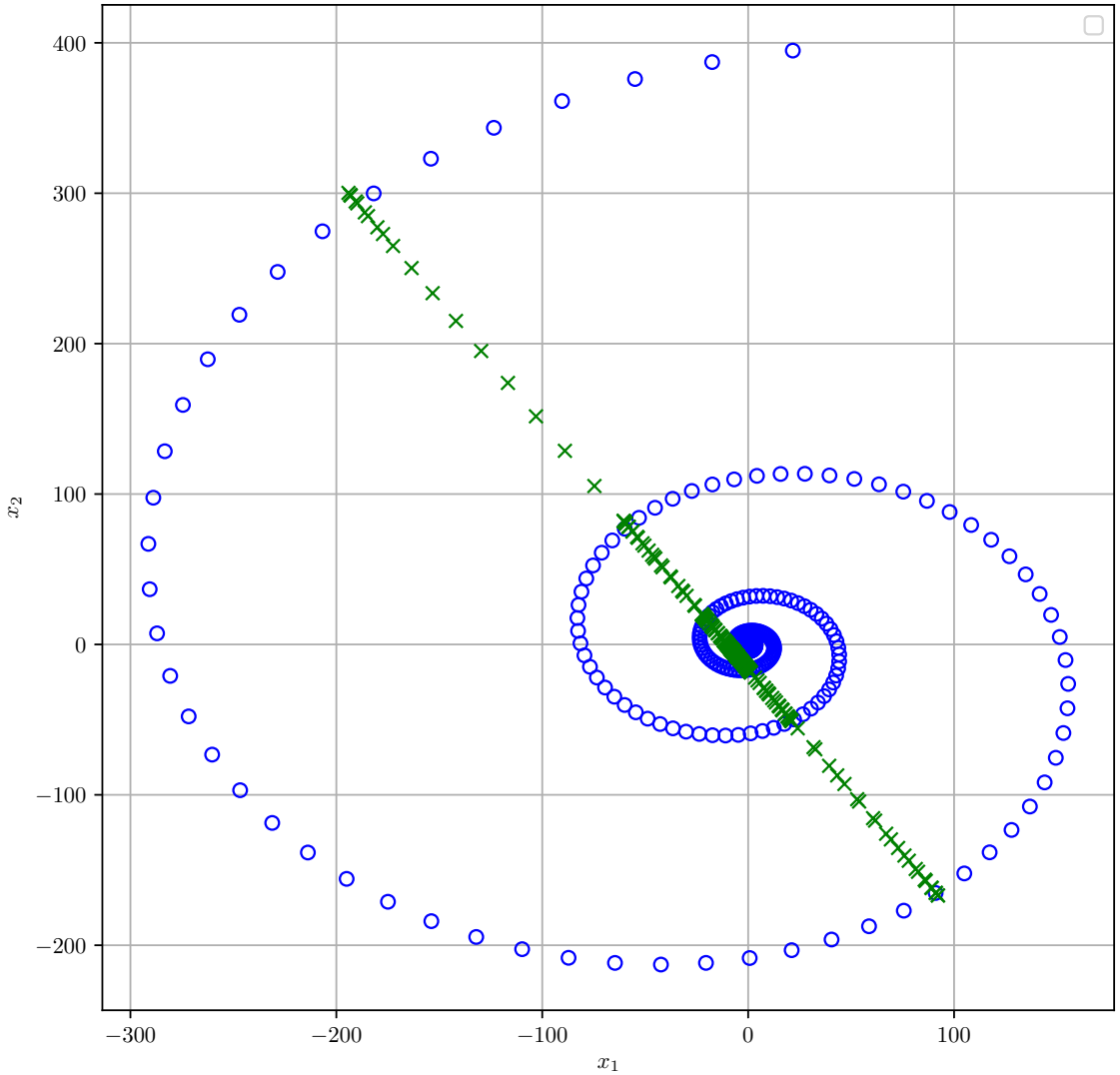
1D manifolds in 2D ambient space

One-dimensional spiral manifold in 2D space



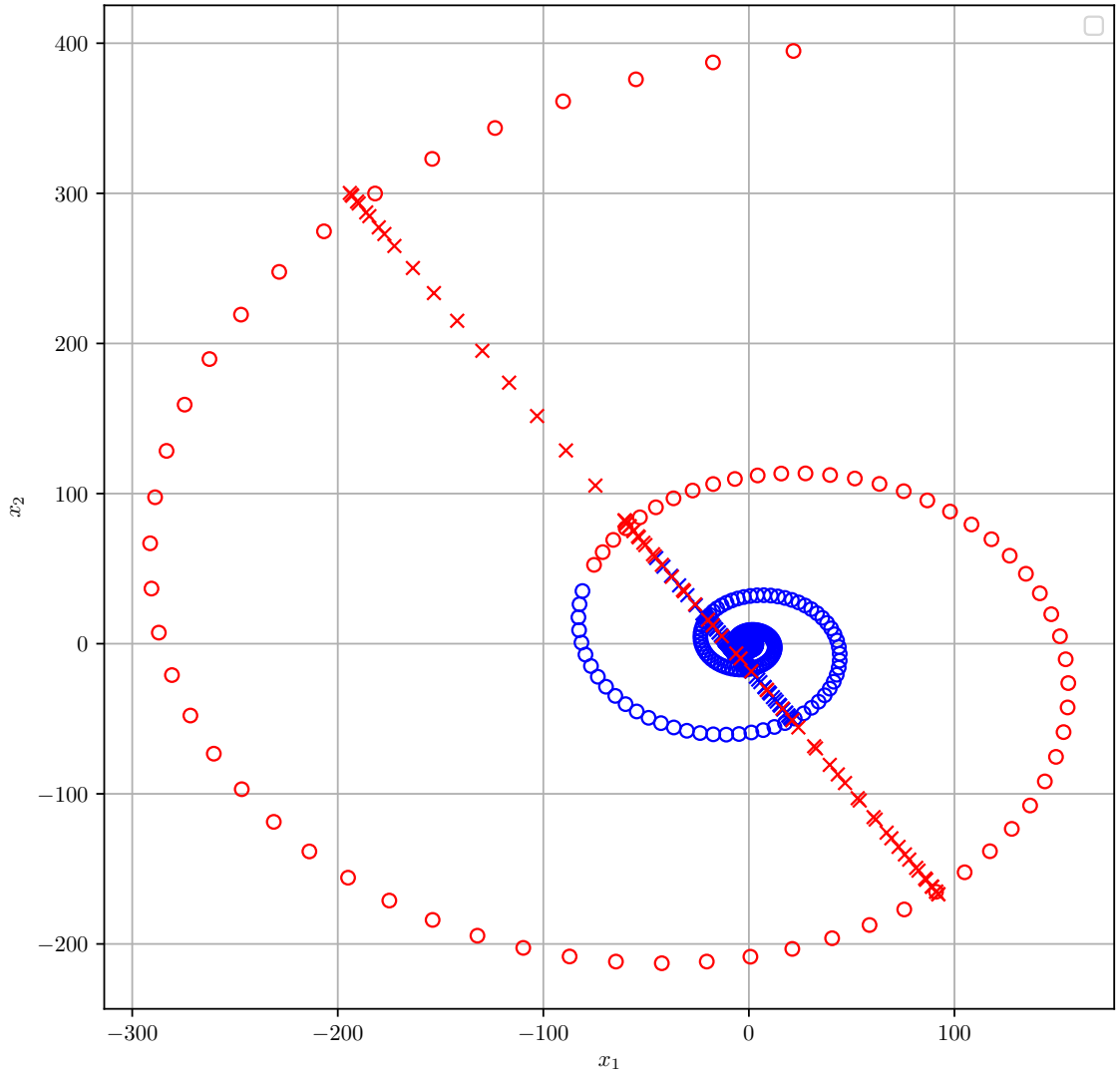
1D manifolds in 2D ambient space, two classes

One-dimensional spiral manifold in 2D space, along with PCA projection.



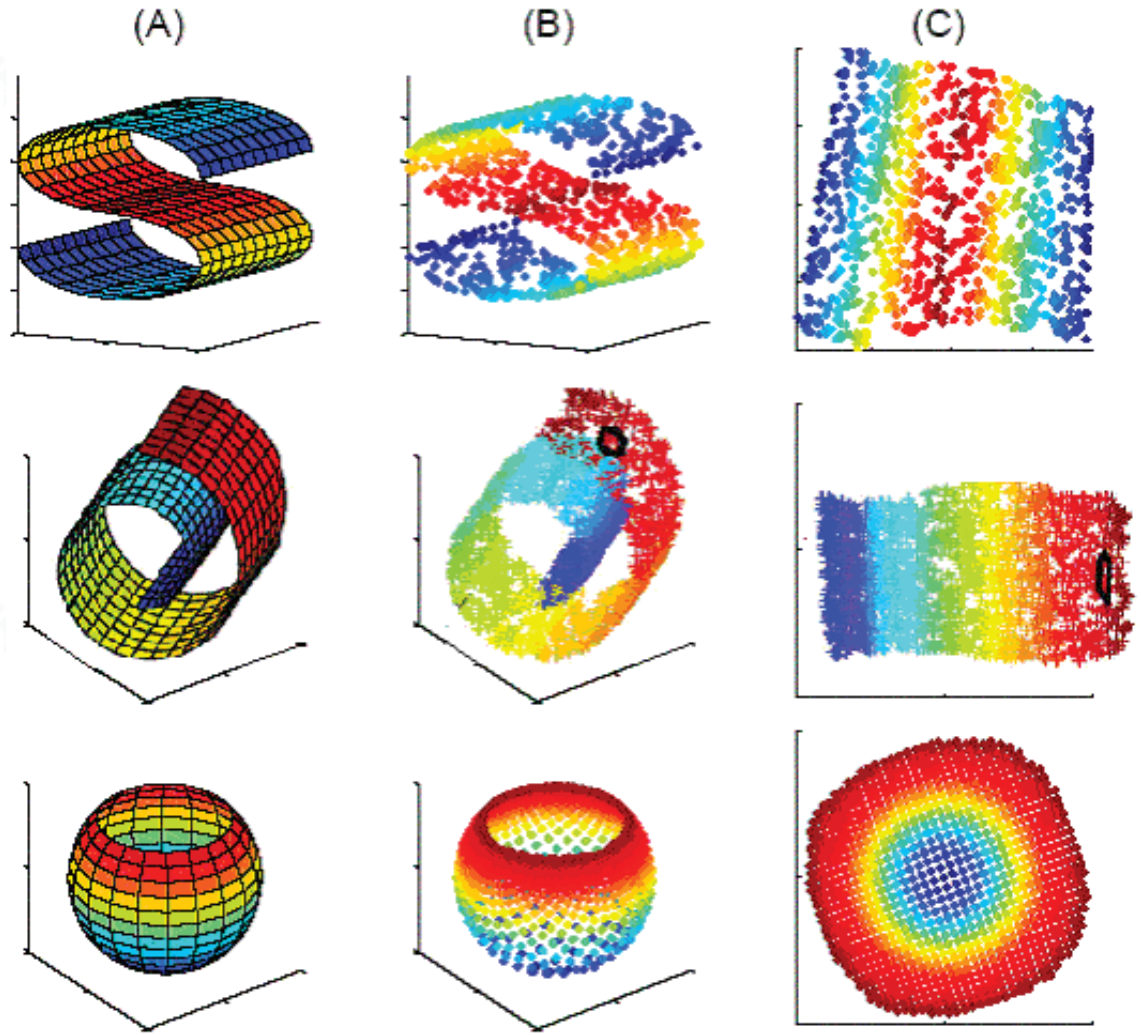
1D manifolds in 2D ambient space

One-dimensional spiral manifold in 2D space, along with PCA projection, 2 class case.



2D Manifold Examples in 3d Ambient Space

(A): manifolds. (B) samples from manifold. (C) Inherent flattened 2D manifolds.



From: Roweiss & Saul, 1999

Manifold Learning in ML

- Goal: find the lower-dimensional manifold on which the data lies, within the ambient high-dimensional space where the is coordinized.

Manifold Learning in ML

- Goal: find the lower-dimensional manifold on which the data lies, within the ambient high-dimensional space where the is coordinized.
- I.e., the high-dimensional data rests near or on an inherently low-dimensional manifold (if flattened) within the high dimensional ambient (input) space.

Manifold Learning in ML

- Goal: find the lower-dimensional manifold on which the data lies, within the ambient high-dimensional space where the is coordinized.
- I.e., the high-dimensional data rests near or on an inherently low-dimensional manifold (if flattened) within the high dimensional ambient (input) space.
- Useful in machine learning for non-linear methods for dimensionality reduction.

Manifold Learning in ML

- Goal: find the lower-dimensional manifold on which the data lies, within the ambient high-dimensional space where the is coordinized.
- I.e., the high-dimensional data rests near or on an inherently low-dimensional manifold (if flattened) within the high dimensional ambient (input) space.
- Useful in machine learning for non-linear methods for dimensionality reduction.
- Example algorithms: Isomap, Laplacian Eigenmaps, Kernel PCA, etc.

Manifold Learning in ML

- Goal: find the lower-dimensional manifold on which the data lies, within the ambient high-dimensional space where the is coordinized.
- I.e., the high-dimensional data rests near or on an inherently low-dimensional manifold (if flattened) within the high dimensional ambient (input) space.
- Useful in machine learning for non-linear methods for dimensionality reduction.
- Example algorithms: Isomap, Laplacian Eigenmaps, Kernel PCA, etc.
- Rather than find the coordinates along the manifold of the data items, the k-NN approaches approximates distances on the manifold. I.e., **geodesic distance** as distance between points along the k-NN graph-represented manifold.

k-NN, memory, parametric vs. non-parametric

- k-NN classifiers do not require an explicit model (e.g., linear, non-linear) and are examples of non-parametric models, unlike the parametric models we've so far seen (but more like t-SNE).

k-NN, memory, parametric vs. non-parametric

- k-NN classifiers do not require an explicit model (e.g., linear, non-linear) and are examples of non-parametric models, unlike the parametric models we've so far seen (but more like t-SNE).
- k-NN classifiers need to memorize the entire training data set

k-NN, memory, parametric vs. non-parametric

- k-NN classifiers do not require an explicit model (e.g., linear, non-linear) and are examples of non-parametric models, unlike the parametric models we've so far seen (but more like t-SNE).
- k-NN classifiers need to memorize the entire training data set
- Other parametric models include: linear models, logistic regression, most neural networks (including the deep variety), or anything involving a fixed and finite number of parameters regardless of the training data size.

k-NN, memory, parametric vs. non-parametric

- k-NN classifiers do not require an explicit model (e.g., linear, non-linear) and are examples of non-parametric models, unlike the parametric models we've so far seen (but more like t-SNE).
- k-NN classifiers need to memorize the entire training data set
- Other parametric models include: linear models, logistic regression, most neural networks (including the deep variety), or anything involving a fixed and finite number of parameters regardless of the training data size.
- Other non-parametric models: k-NN, histogram density estimates, kernel density estimates, kernel/spline/wavelet based regression, support vector machines, kernel machines.

k-NN, memory, parametric vs. non-parametric

- k-NN classifiers do not require an explicit model (e.g., linear, non-linear) and are examples of non-parametric models, unlike the parametric models we've so far seen (but more like t-SNE).
- k-NN classifiers need to memorize the entire training data set
- Other parametric models include: linear models, logistic regression, most neural networks (including the deep variety), or anything involving a fixed and finite number of parameters regardless of the training data size.
- Other non-parametric models: k-NN, histogram density estimates, kernel density estimates, kernel/spline/wavelet based regression, support vector machines, kernel machines.
- Non-parametric methods are useful in that methods to learn them will automatically adjust the complexity to the natural complexity in the data.

Computational Considerations

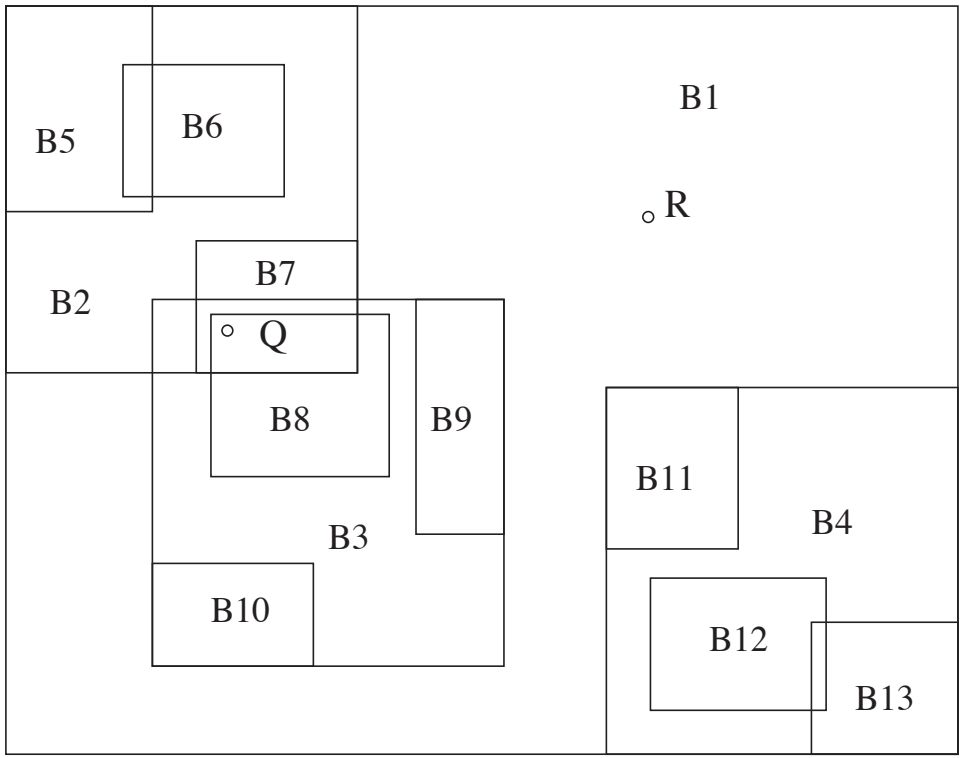
- k-NN classifiers require memory at least to store the n data items.

Computational Considerations

- k-NN classifiers require memory at least to store the n data items.
- With no pre-computation, a naive implementation of a query takes $O((n \log n) + nm)$ or $O((n \log k) + nm)$ time ($O(n)$ distances sorted or top k, and each distance $O(m)$). Also, we need n queries!

Computational Considerations

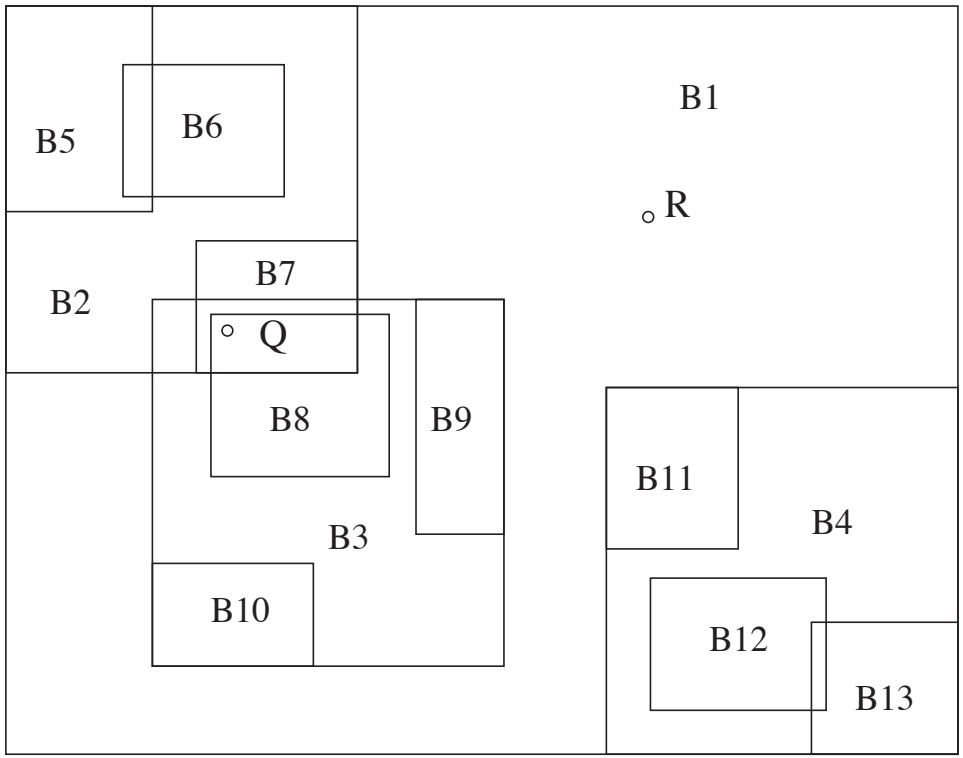
- k-NN classifiers require memory at least to store the n data items.
- With no pre-computation, a naive implementation of a query takes $O((n \log n) + nm)$ or $O((n \log k) + nm)$ time ($O(n)$ distances sorted or top k, and each distance $O(m)$). Also, we need n queries!
- There are data structures for spatial access methods for doing fast nearest neighbor search, using methods analogous to things such as the R-tree.



From Cheung & Fu, "Enhanced Nearest Neighbour Search on the R-Tree"

Computational Considerations

- k-NN classifiers require memory at least to store the n data items.
- With no pre-computation, a naive implementation of a query takes $O((n \log n) + nm)$ or $O((n \log k) + nm)$ time ($O(n)$ distances sorted or top k, and each distance $O(m)$). Also, we need n queries!
- There are data structures for spatial access methods for doing fast nearest neighbor search, using methods analogous to things such as the R-tree.



From Cheung & Fu, "Enhanced Nearest Neighbour Search on the R-Tree"

- Good reference: Chen, Fang, Saad, "Fast Approximate kNN Graph Construction for High Dimensional Data via Recursive Lanczos Bisection", 2009.

Recall: Bayes error

- Recall next two slides from lecture 4.

0/1-loss and Bayes error

- Let L be 0/1, i.e., $L(y, y') = \mathbf{1}_{\{y \neq y'\}}$.
- Probability of error for a given x

$$\Pr(h_\theta(x) \neq Y) = \int p(y|x)\mathbf{1}_{\{h_\theta(x) \neq y\}} dy = 1 - p(h_\theta(x)|x) \quad (7.7)$$

- To minimize the probability of error, $h_\theta(x)$ should ideally choose a class having maximum posterior probability for the current x.
- Smallest probability of error is known is Bayes error for x

$$\text{Bayes error}(x) = \min_y (1 - p(y|x)) = 1 - \max_y p(y|x) \quad (7.8)$$

- Bayes classifier (or predictor) predicts using highest probability, assuming have access to $p(y|x)$. I.e.,

$$\text{BayesPredictor}(x) \triangleq \operatorname{argmax}_y p(y|x) \quad (7.9)$$

- Bayes predictor has 'Bayes error' as its error. Irreducible error rate.

Bayes Error, overall difficulty of a problem

- For Bayes error, we often take the expected value w.r.t. x. This gives an indication overall of how difficult a classification problem is, since we can never do better than Bayes error.
- Bayes error, various equivalent forms (Exercise: show last equality):

$$\text{Bayes Error} = \min_h \int p(x,y)\mathbf{1}_{\{h(x)\neq y\}} dydx \tag{7.13}$$

$$= \min_h \Pr(h(X) \neq Y) \tag{7.14}$$

$$= E_{p(x)}[\min(p(y|x), 1 - p(y|x))] \tag{7.15}$$

$$= \frac{1}{2} - \frac{1}{2} E_{p(x)}[|2p(y|x) - 1|] \tag{7.16}$$

- For binary classification, if Bayes error is $1/2$, prediction can never be better than flipping a coin (i.e., x tells us nothing about y).
- Bayes error, property of the distribution $p(x, y)$; can be useful to decide between say $p(x, y)$ vs. $p(x', y)$ for two different feature sets x and x'.

Aside: modes of convergence

- $X_n \to X$ almost surely $(X_n \xrightarrow{\text{a.s.}} X)$ if the set

$$\{\omega \in \Omega : X_n(\omega) \to X(\omega) \text{ as } n \to \infty\} \tag{7.1}$$

is an event with probability 1.

Aside: modes of convergence

- $X_n \to X$ almost surely $(X_n \xrightarrow{\text{a.s.}} X)$ if the set

$$\{\omega \in \Omega : X_n(\omega) \to X(\omega) \text{ as } n \to \infty\} \tag{7.1}$$

is an event with probability 1. So all such events combined together have probability 1, and any event left out has probability zero, according to the underlying probability measure.

Aside: modes of convergence

- $X_n \to X$ almost surely $(X_n \xrightarrow{\text{a.s.}} X)$ if the set

$$\{\omega \in \Omega : X_n(\omega) \to X(\omega) \text{ as } n \to \infty\} \tag{7.1}$$

  is an event with probability 1. So all such events combined together have probability 1, and any event left out has probability zero, according to the underlying probability measure.
- $X_n \to X$ in the rth mean $(r \geq 1)$, (written $X_n \xrightarrow{r} X$) if

$$E|X_n^r| < \infty \ \ \forall n, \text{ and } E(|X_n - X|^r) \to 0 \text{ as } n \to \infty \tag{7.2}$$

Aside: modes of convergence

- $X_n \to X$ almost surely $(X_n \xrightarrow{\text{a.s.}} X)$ if the set

$$\{\omega \in \Omega : X_n(\omega) \to X(\omega) \text{ as } n \to \infty\} \tag{7.1}$$

  is an event with probability 1. So all such events combined together have probability 1, and any event left out has probability zero, according to the underlying probability measure.
- $X_n \to X$ in the rth mean $(r \geq 1)$, (written $X_n \xrightarrow{r} X$) if

$$E|X_n^r| < \infty \ \ \forall n, \text{ and } E(|X_n - X|^r) \to 0 \text{ as } n \to \infty \tag{7.2}$$

- $X_n \to X$ in probability (written $X_n \xrightarrow{p} X$) if

$$p(|X_n - X| > \epsilon) \to 0 \text{ as } n \to \infty, \ \ \epsilon > 0 \tag{7.3}$$

Aside: modes of convergence

- $X_n \to X$ almost surely $(X_n \xrightarrow{\text{a.s.}} X)$ if the set

$$\{\omega \in \Omega : X_n(\omega) \to X(\omega) \text{ as } n \to \infty\} \tag{7.1}$$

is an event with probability 1. So all such events combined together have probability 1, and any event left out has probability zero, according to the underlying probability measure.

- $X_n \to X$ in the rth mean $(r \geq 1)$, (written $X_n \xrightarrow{r} X$) if

$$E|X_n^r| < \infty \ \ \forall n, \text{ and } E(|X_n - X|^r) \to 0 \text{ as } n \to \infty \tag{7.2}$$

- $X_n \to X$ in probability (written $X_n \xrightarrow{p} X$) if

$$p(|X_n - X| > \epsilon) \to 0 \text{ as } n \to \infty, \ \ \epsilon > 0 \tag{7.3}$$

- $X_n \to X$ in distribution (written $X_n \xrightarrow{D} X$) if

$$p(X_n \leq x) \to P(X < x) \text{ as } n \to \infty \tag{7.4}$$

for all points x at which $F_X(x) = p(X \leq x)$ is continuous.

Aside: modes of convergence

1) • $X_n \to X$ almost surely ($X_n \xrightarrow{\text{a.s.}} X$) if the set

$$\{\omega \in \Omega : X_n(\omega) \to X(\omega) \text{ as } n \to \infty\} \tag{7.1}$$

∃ a sequence of r.v.s $X, X_1, X_2, X_3, \ldots$. How might we measure how X_n approaches X? There are four standard ways:

is an event with probability 1. So all such events combined together have probability 1, and any event left out has probability zero, according to the underlying probability measure.

2) • $X_n \to X$ in the rth mean ($r \geq 1$), (written $X_n \xrightarrow{r} X$) if

$$E|X_n^r| < \infty \;\; \forall n, \text{ and } E(|X_n - X|^r) \to 0 \text{ as } n \to \infty \tag{7.2}$$

3) • $X_n \to X$ in probability (written $X_n \xrightarrow{p} X$) if

$$p(|X_n - X| > \epsilon) \to 0 \text{ as } n \to \infty, \;\; \epsilon > 0 \tag{7.3}$$

a.s. ⇒ prob ⇒ dist; r.t.s ⇒ prob

4) • $X_n \to X$ in distribution (written $X_n \xrightarrow{D} X$) if

$$p(X_n \leq x) \to P(X < x) \text{ as } n \to \infty \tag{7.4}$$

for all points x at which $F_X(x) = p(X \leq x)$ is continuous.

- a.s. $\Rightarrow$ p, r $\Rightarrow$ p, p $\Rightarrow$ D, and if $r > s \geq 1$, then $r \Rightarrow s$.

On Bayes error and Consistency

- Let h be a classifier, its error (or risk) is $R(h) = \Pr(h(X) \neq Y)$.

On Bayes error and Consistency

- Let h be a classifier, its error (or risk) is $R(h) = \Pr(h(X) \neq Y)$.
- Define $R^* = \min_h \Pr(h(X) \neq Y)$ to be Bayes error.

On Bayes error and Consistency

- Let h be a classifier, its error (or risk) is $R(h) = \Pr(h(X) \neq Y)$.
- Define $R^* = \min_h \Pr(h(X) \neq Y)$ to be Bayes error.
- $\mathcal{D}_n$ is a training set with n samples, randomly drawn from $p(x, y)$, hence $\mathcal{D}_n$ is a random variable.

On Bayes error and Consistency

- Let h be a classifier, its error (or risk) is $R(h) = \Pr(h(X) \neq Y)$.
- Define $R^* = \min_h \Pr(h(X) \neq Y)$ to be Bayes error.
- $\mathcal{D}_n$ is a training set with n samples, randomly drawn from $p(x, y)$, hence $\mathcal{D}_n$ is a random variable.
- $\theta(\mathcal{D}_n)$ is learning process, mapping from data to learnt model.

On Bayes error and Consistency

- Let h be a classifier, its error (or risk) is $R(h) = \Pr(h(X) \neq Y)$.
- Define $R^* = \min_h \Pr(h(X) \neq Y)$ to be Bayes error.
- $\mathcal{D}_n$ is a training set with n samples, randomly drawn from $p(x, y)$, hence $\mathcal{D}_n$ is a random variable.
- $\theta(\mathcal{D}_n)$ is learning process, mapping from data to learnt model.
- $h_{\theta(\mathcal{D}_n)}$ is the learnt model from training data $\mathcal{D}_n$ also a random variable with a conditional probability of error $\Pr(h_{\theta(\mathcal{D}_n)}(X) \neq Y | \mathcal{D}_n)$.

On Bayes error and Consistency

- Let h be a classifier, its error (or risk) is $R(h) = \Pr(h(X) \neq Y)$.
- Define $R^* = \min_h \Pr(h(X) \neq Y)$ to be Bayes error.
- $\mathcal{D}_n$ is a training set with n samples, randomly drawn from $p(x, y)$, hence $\mathcal{D}_n$ is a random variable.
- $\theta(\mathcal{D}_n)$ is learning process, mapping from data to learnt model.
- $h_{\theta(\mathcal{D}_n)}$ is the learnt model from training data $\mathcal{D}_n$ also a random variable with a conditional probability of error $\Pr(h_{\theta(\mathcal{D}_n)}(X) \neq Y | \mathcal{D}_n)$.
- Define $R(h_{\theta(\mathcal{D}_n)} | \mathcal{D}_n) = \Pr(h_{\theta(\mathcal{D}_n)}(X) \neq Y | \mathcal{D}_n)$.

On Bayes error and Consistency

- Let h be a classifier, its error (or risk) is $R(h) = \Pr(h(X) \neq Y)$.
- Define $R^* = \min_h \Pr(h(X) \neq Y)$ to be Bayes error.
- $\mathcal{D}_n$ is a training set with n samples, randomly drawn from $p(x, y)$, hence $\mathcal{D}_n$ is a random variable.
- $\theta(\mathcal{D}_n)$ is learning process, mapping from data to learnt model.
- $h_{\theta(\mathcal{D}_n)}$ is the learnt model from training data $\mathcal{D}_n$ also a random variable with a conditional probability of error $\Pr(h_{\theta(\mathcal{D}_n)}(X) \neq Y | \mathcal{D}_n)$.
- Define $R(h_{\theta(\mathcal{D}_n)}|\mathcal{D}_n) = \Pr(h_{\theta(\mathcal{D}_n)}(X) \neq Y|\mathcal{D}_n)$.
- We consider how $E_{\mathcal{D}_n}\left[R(h_{\theta(\mathcal{D}_n)}|\mathcal{D}_n)\right]$ improves with n.

On Bayes error and Consistency

- Let h be a classifier, its error (or risk) is $R(h) = \Pr(h(X) \neq Y)$.
- Define $R^* = \min_h \Pr(h(X) \neq Y)$ to be Bayes error.
- $\mathcal{D}_n$ is a training set with n samples, randomly drawn from $p(x, y)$, hence $\mathcal{D}_n$ is a random variable.
- $\theta(\mathcal{D}_n)$ is learning process, mapping from data to learnt model.
- $h_{\theta(\mathcal{D}_n)}$ is the learnt model from training data $\mathcal{D}_n$ also a random variable with a conditional probability of error $\Pr(h_{\theta(\mathcal{D}_n)}(X) \neq Y | \mathcal{D}_n)$.
- Define $R(h_{\theta(\mathcal{D}_n)}|\mathcal{D}_n) = \Pr(h_{\theta(\mathcal{D}_n)}(X) \neq Y|\mathcal{D}_n)$.
- We consider how $E_{\mathcal{D}_n}\left[R(h_{\theta(\mathcal{D}_n)}|\mathcal{D}_n)\right]$ improves with n.

Definition 7.3.1 (Consistency)

A classification process is consistent for distribution $p(x, y)$ if $E_{\mathcal{D}_n}[R(h_{\theta(\mathcal{D}_n)}|\mathcal{D}_n)] \xrightarrow{p} R^*$ as $n \to \infty$.

On Bayes error and Consistency

- Let h be a classifier, its error (or risk) is $R(h) = \Pr(h(X) \neq Y)$.
- Define $R^* = \min_h \Pr(h(X) \neq Y)$ to be Bayes error.
- $\mathcal{D}_n$ is a training set with n samples, randomly drawn from $p(x, y)$, hence $\mathcal{D}_n$ is a random variable.
- $\theta(\mathcal{D}_n)$ is learning process, mapping from data to learnt model.
- $h_{\theta(\mathcal{D}_n)}$ is the learnt model from training data $\mathcal{D}_n$ also a random variable with a conditional probability of error $\Pr(h_{\theta(\mathcal{D}_n)}(X) \neq Y | \mathcal{D}_n)$.
- Define $R(h_{\theta(\mathcal{D}_n)} | \mathcal{D}_n) = \Pr(h_{\theta(\mathcal{D}_n)}(X) \neq Y | \mathcal{D}_n)$.
- We consider how $E_{\mathcal{D}_n}\left[R(h_{\theta(\mathcal{D}_n)} | \mathcal{D}_n)\right]$ improves with n.

Definition 7.3.1 (Consistency)

A classification process is consistent for distribution $p(x, y)$ if $E_{\mathcal{D}_n}[R(h_{\theta(\mathcal{D}_n)} | \mathcal{D}_n)] \to R^*$ as $n \to \infty$.

Definition 7.3.2 (Universal Consistency)

A classification process is universally consistent if it is consistent for all distributions $p(x, y)$.

How far away are we from Bayes error

- Why is consistency and universal consistency a desirable property?

How far away are we from Bayes error

- Why is consistency and universal consistency a desirable property?
  - As we get more data we know we can't get worse and will approach eventually the best we can ever do, for a given distribution $p(x, y)$.

How far away are we from Bayes error

- Why is consistency and universal consistency a desirable property?
  - As we get more data we know we can't get worse and will approach eventually the best we can ever do, for a given distribution $p(x, y)$.
  - Universality means that we can expect this to be true regardless of the distribution.

How far away are we from Bayes error

- Why is consistency and universal consistency a desirable property?
  - As we get more data we know we can't get worse and will approach eventually the best we can ever do, for a given distribution $p(x, y)$.
  - Universality means that we can expect this to be true regardless of the distribution.
- Ideally, we'd like to show that there is a family of classification methods that are universally consistent.

How far away are we from Bayes error

- Why is consistency and universal consistency a desirable property?
  - As we get more data we know we can't get worse and will approach eventually the best we can ever do, for a given distribution $p(x, y)$.
  - Universality means that we can expect this to be true regardless of the distribution.
- Ideally, we'd like to show that there is a family of classification methods that are universally consistent.
- Wasn't known until relatively recently (in the "stone age"), in 1977 Chuck Stone proved a famous theorem, that the k-NN classifier is universally consistent!

How far away are we from Bayes error

- Why is consistency and universal consistency a desirable property?
  - As we get more data we know we can't get worse and will approach eventually the best we can ever do, for a given distribution $p(x, y)$.
  - Universality means that we can expect this to be true regardless of the distribution.
- Ideally, we'd like to show that there is a family of classification methods that are universally consistent.
- Wasn't known until relatively recently (in the "stone age"), in 1977 Chuck Stone proved a famous theorem, that the k-NN classifier is universally consistent!
- Let $h_{k\text{-NN},\mathcal{D}_n}(x)$ be the decision decided by the k-NN binary classifier (majority vote over neighbors) on data set $\mathcal{D}_n$ drawn randomly from $p(x, y)$.

How far away are we from Bayes error

- Why is consistency and universal consistency a desirable property?
  - As we get more data we know we can't get worse and will approach eventually the best we can ever do, for a given distribution $p(x, y)$.
  - Universality means that we can expect this to be true regardless of the distribution.
- Ideally, we'd like to show that there is a family of classification methods that are universally consistent.
- Wasn't known until relatively recently (in the "stone age"), in 1977 Chuck Stone proved a famous theorem, that the k-NN classifier is universally consistent!
- Let $h_{k\text{-NN},\mathcal{D}_n}(x)$ be the decision decided by the k-NN binary classifier (majority vote over neighbors) on data set $\mathcal{D}_n$ drawn randomly from $p(x, y)$. Stone showed that:

How far away are we from Bayes error

- Why is consistency and universal consistency a desirable property?
  - As we get more data we know we can't get worse and will approach eventually the best we can ever do, for a given distribution $p(x, y)$.
  - Universality means that we can expect this to be true regardless of the distribution.
- Ideally, we'd like to show that there is a family of classification methods that are universally consistent.
- Wasn't known until relatively recently (in the "stone age"), in 1977 Chuck Stone proved a famous theorem, that the k-NN classifier is universally consistent!
- Let $h_{k\text{-NN},\mathcal{D}_n}(x)$ be the decision decided by the k-NN binary classifier (majority vote over neighbors) on data set $\mathcal{D}_n$ drawn randomly from $p(x, y)$. Stone showed that:

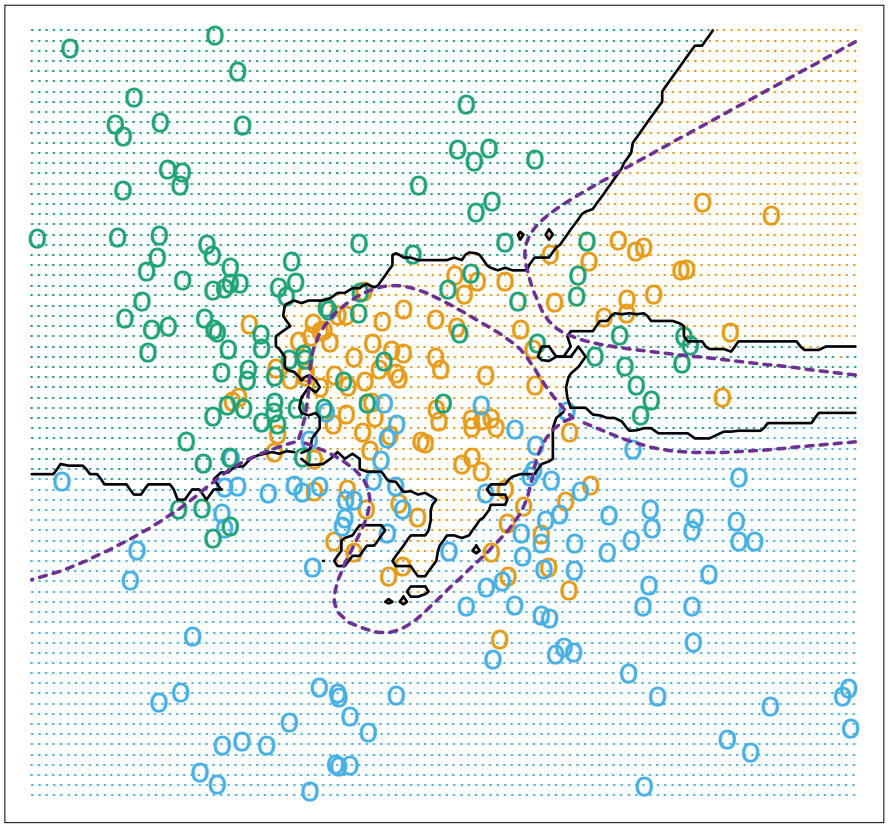
Theorem 7.3.3 (Stone, 1977)

Let $k \to \infty$ with increasing n such that $k/n \to 0$ as $n \to \infty$. Then for all probability distributions $p(x, y)$, $E_{\mathcal{D}_n}[R(h_{k\text{-NN},\mathcal{D}_n})] \to R^*$ as $n \to \infty$.

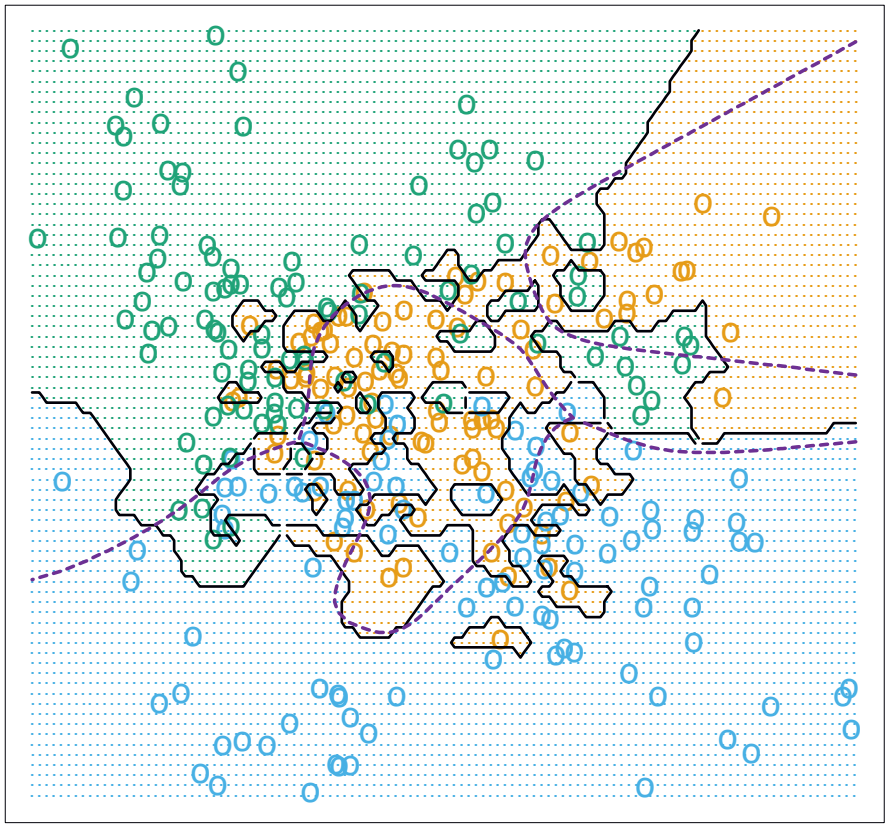
k-NN and Bayes error

3-class example for various k and comparing to Bayes error (purple dashed lines). From Hastie et. al., 2009

15-Nearest Neighbors



1-Nearest Neighbor



What about 1-NN?

- Even the 1-NN classifier is good in a certain way.

What about 1-NN?

- Even the 1-NN classifier is good in a certain way. Let $h_{\text{1-NN},\mathcal{D}_n}(x)$ be the nearest neighbor, classifier, based on data set $\mathcal{D}_n$.

What about 1-NN?

- Even the 1-NN classifier is good in a certain way. Let $h_{\text{1-NN},\mathcal{D}_n}(x)$ be the nearest neighbor, classifier, based on data set $\mathcal{D}_n$. We have:

What about 1-NN?

- Even the 1-NN classifier is good in a certain way. Let $h_{\text{1-NN},\mathcal{D}_n}(x)$ be the nearest neighbor, classifier, based on data set $\mathcal{D}_n$. We have:

Theorem 7.3.4 (Cover & Hart, 1967)

For the 1-NN rule and for any distribution $p(x, y)$, we have

$$\lim_{n\to\infty} E_{\mathcal{D}_n}[R(h_{\text{1-NN},\mathcal{D}_n})] = E[2\eta(X)(1-\eta(X))] \tag{7.5}$$

where $\eta(x) = E[Y|X = x] = p(y|x)$.

What about 1-NN?

- Even the 1-NN classifier is good in a certain way. Let $h_{\text{1-NN},\mathcal{D}_n}(x)$ be the nearest neighbor, classifier, based on data set $\mathcal{D}_n$. We have:

Theorem 7.3.4 (Cover & Hart, 1967)

For the 1-NN rule and for any distribution $p(x, y)$, we have

$$\lim_{n\to\infty} E_{\mathcal{D}_n}[R(h_{\text{1-NN},\mathcal{D}_n})] = E[2\eta(X)(1-\eta(X))] \tag{7.5}$$

where $\eta(x) = E[Y|X = x] = p(y|x)$.

- As a consequence, we have

What about 1-NN?

- Even the 1-NN classifier is good in a certain way. Let $h_{\text{1-NN},\mathcal{D}_n}(x)$ be the nearest neighbor, classifier, based on data set $\mathcal{D}_n$. We have:

Theorem 7.3.4 (Cover & Hart, 1967)

For the 1-NN rule and for any distribution $p(x, y)$, we have

$$\lim_{n\to\infty} E_{\mathcal{D}_n}[R(h_{\text{1-NN},\mathcal{D}_n})] = E[2\eta(X)(1-\eta(X)] \tag{7.5}$$

where $\eta(x) = E[Y|X = x] = p(y|x)$.

- As a consequence, we have

Theorem 7.3.5 (Cover & Hart, 1967)

For the 1-NN rule and for any distribution $p(x, y)$, we have

$$\lim_{n\to\infty} E_{\mathcal{D}_n}[R(h_{\text{1-NN},\mathcal{D}_n})] \leq 2R^* \tag{7.6}$$

What about 1-NN?

- Even the 1-NN classifier is good in a certain way. Let $h_{\text{1-NN},\mathcal{D}_n}(x)$ be the nearest neighbor, classifier, based on data set $\mathcal{D}_n$. We have:

Theorem 7.3.4 (Cover & Hart, 1967)

For the 1-NN rule and for any distribution $p(x, y)$, we have

$$\lim_{n\to\infty} E_{\mathcal{D}_n}[R(h_{\text{1-NN},\mathcal{D}_n})] = E[2\eta(X)(1-\eta(X)] \tag{7.5}$$

where $\eta(x) = E[Y|X = x] = p(y|x)$.

- As a consequence, we have

Theorem 7.3.5 (Cover & Hart, 1967)

For the 1-NN rule and for any distribution $p(x, y)$, we have

$$\lim_{n\to\infty} E_{\mathcal{D}_n}[R(h_{\text{1-NN},\mathcal{D}_n})] \leq 2R^* \tag{7.6}$$

- Thus, simple 1-NN rule is, asymptotically, no worse than twice Bayes error. If Bayes is small, so is 1-NN (eventually, and on average!).

Aspects of k-NN

- Universal consistency, converges on average to Bayes rule if $k \to \infty$, $k/n \to 0$ as $n \to \infty$.

Aspects of k-NN

- Universal consistency, converges on average to Bayes rule if $k \to \infty$, $k/n \to 0$ as $n \to \infty$.
- How does it behave for finite k? k is smoothing parameter (analogous to λ tradeoff parameter) as we have seen. How to choose k?

Aspects of k-NN

- Universal consistency, converges on average to Bayes rule if $k \to \infty$, $k/n \to 0$ as $n \to \infty$.
- How does it behave for finite k? k is smoothing parameter (analogous to λ tradeoff parameter) as we have seen. How to choose k? (via cross-validation)

Aspects of k-NN

- Universal consistency, converges on average to Bayes rule if $k \to \infty$, $k/n \to 0$ as $n \to \infty$.
- How does it behave for finite k? k is smoothing parameter (analogous to λ tradeoff parameter) as we have seen. How to choose k? (via cross-validation)
- Can we weight the nearest neighbors? Why should they always count equally? Perhaps by distance (suggesting kernel estimators)?

Aspects of k-NN

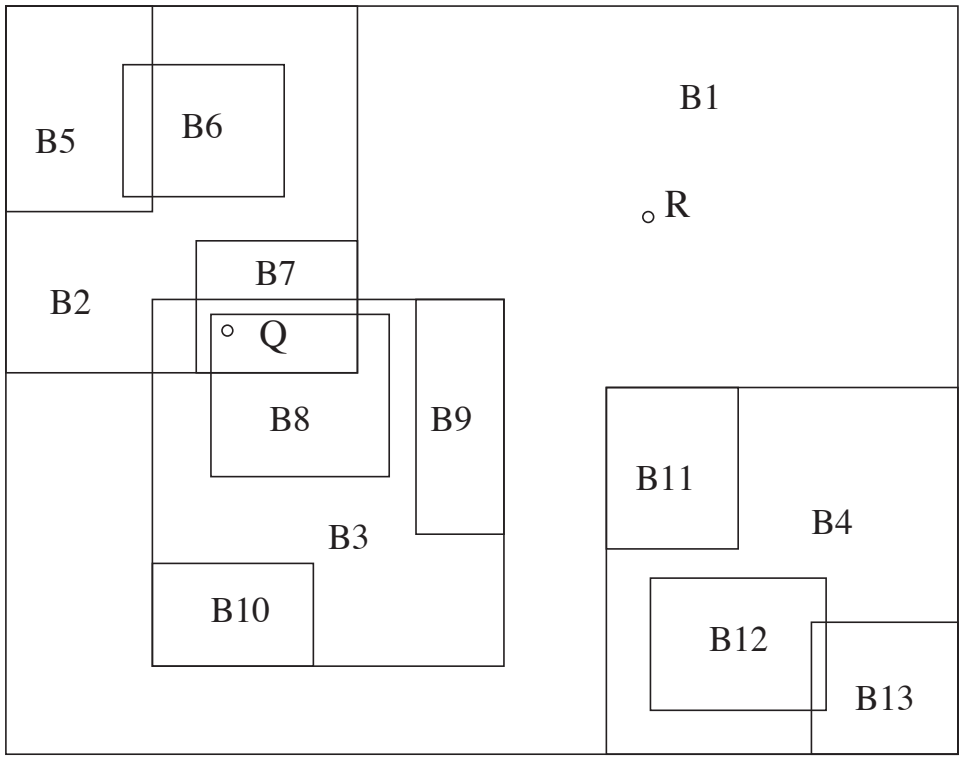
- Universal consistency, converges on average to Bayes rule if $k \to \infty$, $k/n \to 0$ as $n \to \infty$.
- How does it behave for finite k? k is smoothing parameter (analogous to λ tradeoff parameter) as we have seen. How to choose k? (via cross-validation)
- Can we weight the nearest neighbors? Why should they always count equally? Perhaps by distance (suggesting kernel estimators)?
- What distance metric? Euclidean, or L_p distance $d(x, x') = \|x - x'\|_p$.

Aspects of k-NN

- Universal consistency, converges on average to Bayes rule if $k \to \infty$, $k/n \to 0$ as $n \to \infty$.
- How does it behave for finite k? k is smoothing parameter (analogous to λ tradeoff parameter) as we have seen. How to choose k? (via cross-validation)
- Can we weight the nearest neighbors? Why should they always count equally? Perhaps by distance (suggesting kernel estimators)?
- What distance metric? Euclidean, or L_p distance $d(x, x') = \|x - x'\|_p$.
- Memory: What if we reduce the training data size, come up with exemplars. Clustering, EM, unsupervised learning.

Computational Considerations

- k-NN classifiers require memory at least to store the n data items.
- With no pre-computation, a naive implementation of a query takes $O((n \log n) + nm)$ or $O((n \log k) + nm)$ time ($O(n)$ distances sorted or top k, and each distance $O(m)$). Also, we need n queries!
- There are data structures for spatial access methods for doing fast nearest neighbor search, using methods analogous to things such as the R-tree.



From Cheung & Fu, "Enhanced Nearest Neighbour Search on the R-Tree"

- Good reference: Chen, Fang, Saad, "Fast Approximate kNN Graph Construction for High Dimensional Data via Recursive Lanczos Bisection", 2009.

Nearest Neighbor Search and Computation

- Nearest Neighbor Search (NNS) Problem: Given a set of n points $\left\{x^{(i)}\right\}_{i=1}^{n}$, $x^{(i)} \in \mathbb{R}^m$, finding nearest to query $q \in \mathbb{R}^m$ is $\operatorname{argmin}_i d(x^{(i)}, q)$.

Nearest Neighbor Search and Computation

- Nearest Neighbor Search (NNS) Problem: Given a set of n points $\left\{x^{(i)}\right\}_{i=1}^{n}$, $x^{(i)} \in \mathbb{R}^m$, finding nearest to query $q \in \mathbb{R}^m$ is $\operatorname{argmin}_i d(x^{(i)}, q)$.
- Naïve costs $O(nm)$ both time and memory. To find all points' nearest neighbors (1-NN) is $O(n^2 m)$ naively.

Nearest Neighbor Search and Computation

- Nearest Neighbor Search (NNS) Problem: Given a set of n points $\{x^{(i)}\}_{i=1}^{n}$, $x^{(i)} \in \mathbb{R}^m$, finding nearest to query $q \in \mathbb{R}^m$ is $\text{argmin}_i \, d(x^{(i)}, q)$.
- Naïve costs $O(nm)$ both time and memory. To find all points' nearest neighbors (1-NN) is $O(n^2 m)$ naively.
- Recall m-dimension spherical Gaussian, zero mean, variance σ^2

$$p(x) = \frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{\|x\|_2^2}{2\sigma^2}\right) \tag{7.7}$$

Nearest Neighbor Search and Computation

- Nearest Neighbor Search (NNS) Problem: Given a set of n points $\{x^{(i)}\}_{i=1}^{n}$, $x^{(i)} \in \mathbb{R}^m$, finding nearest to query $q \in \mathbb{R}^m$ is $\operatorname{argmin}_i d(x^{(i)}, q)$.
- Naïve costs $O(nm)$ both time and memory. To find all points' nearest neighbors (1-NN) is $O(n^2 m)$ naively.
- Recall m-dimension spherical Gaussian, zero mean, variance σ^2

$$p(x) = \frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{\|x\|_2^2}{2\sigma^2}\right) \tag{7.7}$$

- As m gets large, this Gaussian (as we saw in HW3) starts to behave very differently.

Gaussian Annulus Theorem: HW3 Redux

- HW3: when $\sigma^2 = 1$, very little probability mass over unit $r = 1$ ball since so little volume there; only when $r = \sqrt{m}$ is mass over ball significant. Beyond $r = \sqrt{m}$ mass does not increase much (as fast as ball volume continues to increase, density drops at higher rate). I.e., HW3 proves:

Gaussian Annulus Theorem: HW3 Redux

- HW3: when $\sigma^2 = 1$, very little probability mass over unit $r = 1$ ball since so little volume there; only when $r = \sqrt{m}$ is mass over ball significant. Beyond $r = \sqrt{m}$ mass does not increase much (as fast as ball volume continues to increase, density drops at higher rate). I.e., HW3 proves:

Theorem 7.4.1 (Gaussian Annulus Theorem)

For any m-dimensional spherical Gaussian with common unit variance, for any $\beta \leq \sqrt{m}$, all but at most $3e^{-c\beta^2}$ of the probability mass lies within annulus $\sqrt{m} - \beta \leq \|x\|_2 \leq \sqrt{m} + \beta$, where $c > 0$ is a constant.

Gaussian Annulus Theorem: HW3 Redux

- HW3: when $\sigma^2 = 1$, very little probability mass over unit $r = 1$ ball since so little volume there; only when $r = \sqrt{m}$ is mass over ball significant. Beyond $r = \sqrt{m}$ mass does not increase much (as fast as ball volume continues to increase, density drops at higher rate). I.e., HW3 proves:

Theorem 7.4.1 (Gaussian Annulus Theorem)

For any m-dimensional spherical Gaussian with common unit variance, for any $\beta \leq \sqrt{m}$, all but at most $3e^{-c\beta^2}$ of the probability mass lies within annulus $\sqrt{m} - \beta \leq \|x\|_2 \leq \sqrt{m} + \beta$, where $c > 0$ is a constant.

- Theorem says that $\Pr(\left|\|X\|_2 - \sqrt{m}\right| \leq \beta) > 1 - 3e^{-c\beta^2}$.

Gaussian Annulus Theorem: HW3 Redux

- HW3: when $\sigma^2 = 1$, very little probability mass over unit $r = 1$ ball since so little volume there; only when $r = \sqrt{m}$ is mass over ball significant. Beyond $r = \sqrt{m}$ mass does not increase much (as fast as ball volume continues to increase, density drops at higher rate). I.e., HW3 proves:

Theorem 7.4.1 (Gaussian Annulus Theorem)

For any m-dimensional spherical Gaussian with common unit variance, for any $\beta \leq \sqrt{m}$, all but at most $3e^{-c\beta^2}$ of the probability mass lies within annulus $\sqrt{m} - \beta \leq \|x\|_2 \leq \sqrt{m} + \beta$, where $c > 0$ is a constant.

- Theorem says that $\Pr(\left|\|X\|_2 - \sqrt{m}\right| \leq \beta) > 1 - 3e^{-c\beta^2}$.
- Choose any $0 < \epsilon = 3e^{-c\beta^2}$ to be the fraction to ignore and set β achieve that fraction (easy thanks to $\exp(\cdot)$). As m gets big, β range becomes negligible since $|\sqrt{m} + \beta - (\sqrt{m} - \beta)|/\sqrt{m} \to 0$ as $m \to \infty$.

Gaussian Annulus Theorem: HW3 Redux

- HW3: when $\sigma^2 = 1$, very little probability mass over unit $r = 1$ ball since so little volume there; only when $r = \sqrt{m}$ is mass over ball significant. Beyond $r = \sqrt{m}$ mass does not increase much (as fast as ball volume continues to increase, density drops at higher rate). I.e., HW3 proves:

Theorem 7.4.1 (Gaussian Annulus Theorem)

For any m-dimensional spherical Gaussian with common unit variance, for any $\beta \leq \sqrt{m}$, all but at most $3e^{-c\beta^2}$ of the probability mass lies within annulus $\sqrt{m} - \beta \leq \|x\|_2 \leq \sqrt{m} + \beta$, where $c > 0$ is a constant.

- Theorem says that $\Pr(\left|\|X\|_2 - \sqrt{m}\right| \leq \beta) > 1 - 3e^{-c\beta^2}$.
- Choose any $0 < \epsilon = 3e^{-c\beta^2}$ to be the fraction to ignore and set β achieve that fraction (easy thanks to $\exp(\cdot)$). As m gets big, β range becomes negligible since $|\sqrt{m} + \beta - (\sqrt{m} - \beta)|/\sqrt{m} \to 0$ as $m \to \infty$.
- $E[\|X\|_2^2] = \sum_{i=1}^m E[X_i^2] = mE[X_i^2] = m$, mean squared distance of a point to center is m. As m increases, things concentrate at this distance.

Gaussian Annulus Theorem: HW3 Redux

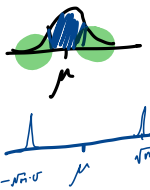
- HW3: when $\sigma^2 = 1$, very little probability mass over unit $r = 1$ ball since so little volume there; only when $r = \sqrt{m}$ is mass over ball significant. Beyond $r = \sqrt{m}$ mass does not increase much (as fast as ball volume continues to increase, density drops at higher rate). I.e., HW3 proves:

Theorem 7.4.1 (Gaussian Annulus Theorem)

For any m-dimensional spherical Gaussian with common unit variance, for any $\beta \leq \sqrt{m}$, all but at most $3e^{-c\beta^2}$ of the probability mass lies within annulus $\sqrt{m} - \beta \leq \|x\|_2 \leq \sqrt{m} + \beta$, where $c > 0$ is a constant.

- Theorem says that $\Pr(\left|\|X\|_2 - \sqrt{m}\right| \leq \beta) > 1 - 3e^{-c\beta^2}$.
- Choose any $0 < \epsilon = 3e^{-c\beta^2}$ to be the fraction to ignore and set β achieve that fraction (easy thanks to $\exp(\cdot)$). As m gets big, β range becomes negligible since $|\sqrt{m} + \beta - (\sqrt{m} - \beta)|/\sqrt{m} \to 0$ as $m \to \infty$.
- $E[\|X\|_2^2] = \sum_{i=1}^m E[X_i^2] = mE[X_i^2] = m$, mean squared distance of a point to center is m. As m increases, things concentrate at this distance.
- $\sqrt{m}$ sometimes called radius of the Gaussian.

Gaussian Annulus Theorem: HW3 Redux



- HW3: when $\sigma^2 = 1$, very little probability mass over unit $r = 1$ ball since so little volume there; only when $r = \sqrt{m}$ is mass over ball significant. Beyond $r = \sqrt{m}$ mass does not increase much (as fast as ball volume continues to increase, density drops at higher rate). I.e., HW3 proves:

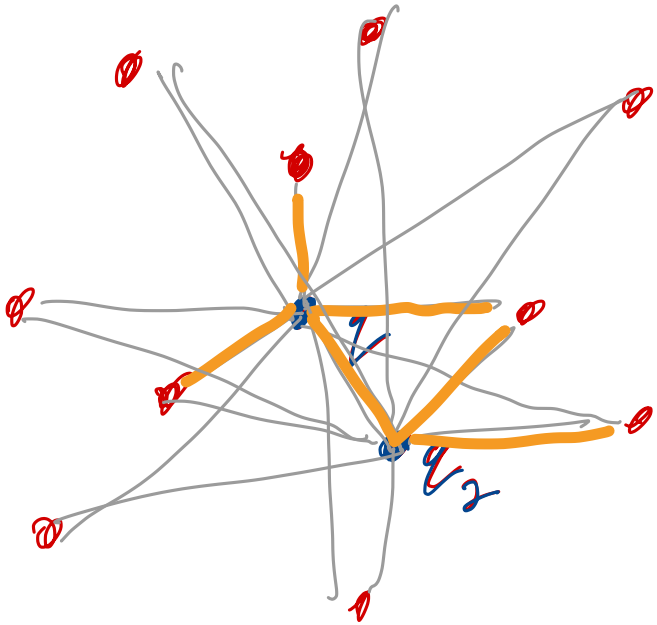
Theorem 7.4.1 (Gaussian Annulus Theorem)

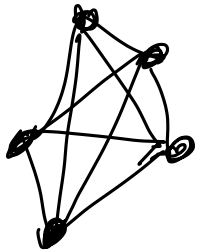
For any m-dimensional spherical Gaussian with common unit variance, for any $\beta \leq \sqrt{m}$, all but at most $3e^{-c\beta^2}$ of the probability mass lies within annulus $\sqrt{m} - \beta \leq \|x\|_2 \leq \sqrt{m} + \beta$, where $c > 0$ is a constant.

- Theorem says that $\Pr(\left|\|X\|_2 - \sqrt{m}\right| \leq \beta) > 1 - 3e^{-c\beta^2}$.
- Choose any $0 < \epsilon = 3e^{-c\beta^2}$ to be the fraction to ignore and set β achieve that fraction (easy thanks to $\exp(\cdot)$). As m gets big, β range becomes negligible since $|\sqrt{m} + \beta - (\sqrt{m} - \beta)|/\sqrt{m} \to 0$ as $m \to \infty$.
- $E[\|X\|_2^2] = \sum_{i=1}^m E[X_i^2] = mE[X_i^2] = m$, mean squared distance of a point to center is m. As m increases, things concentrate at this distance.
- $\sqrt{m}$ sometimes called radius of the Gaussian. $(\sqrt{m} \cdot \sigma)$
- For details, see the text by Blum, Hopcroft, and Kannan, 2018.

Random Projections to speed NN Search (NNS)

- In lower dimensionality space $m' << m$, NNS much faster.





Random Projections to speed NN Search (NNS)

- In lower dimensionality space $m' << m$, NNS much faster.
- Possible solution: Let $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ be a projection from m to $m' \ll m$ dimensions.

Random Projections to speed NN Search (NNS)

- In lower dimensionality space $m' << m$, NNS much faster.
- Possible solution: Let $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ be a projection from m to $m' \ll m$ dimensions. Is x and y near if $f(x)$ near $f(y)$?

Random Projections to speed NN Search (NNS)

- In lower dimensionality space $m' << m$, NNS much faster.
- Possible solution: Let $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ be a projection from m to $m' \ll m$ dimensions. Is x and y near if $f(x)$ near $f(y)$?
  $x = (100, 0, 0, \ldots, 0) \in \mathbb{R}^m$, $y = (0, 0, \ldots, 0) \in \mathbb{R}^m$.

Random Projections to speed NN Search (NNS)

- In lower dimensionality space $m' << m$, NNS much faster.
- Possible solution: Let $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ be a projection from m to $m' \ll m$ dimensions. Is x and y near if $f(x)$ near $f(y)$? $x = (100, 0, 0, \ldots, 0) \in \mathbb{R}^m$, $y = (0, 0, \ldots, 0) \in \mathbb{R}^m$.Ex-a: $f_a(x) = x_1 \in \mathbb{R}$?

Random Projections to speed NN Search (NNS)

- In lower dimensionality space $m' << m$, NNS much faster.
- Possible solution: Let $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ be a projection from m to $m' \ll m$ dimensions. Is x and y near if $f(x)$ near $f(y)$?
  $x = (100, 0, 0, \ldots, 0) \in \mathbb{R}^m$, $y = (0, 0, \ldots, 0) \in \mathbb{R}^m$.Ex-a: $f_a(x) = x_1 \in \mathbb{R}$? Ex-b: $f_b(x) = x_2 \in \mathbb{R}$?

$$f_b(x) = f_b(y)$$

Random Projections to speed NN Search (NNS)

- In lower dimensionality space $m' << m$, NNS much faster.
- Possible solution: Let $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ be a projection from m to $m' \ll m$ dimensions. Is x and y near if $f(x)$ near $f(y)$? $x = (100, 0, 0, \ldots, 0) \in \mathbb{R}^m$, $y = (0, 0, \ldots, 0) \in \mathbb{R}^m$.Ex-a: $f_a(x) = x_1 \in \mathbb{R}$? Ex-b: $f_b(x) = x_2 \in \mathbb{R}$?
- Approach: pick m' vectors $u^{(1)}, u^{(2)}, \ldots, u^{(m')} \in \mathbb{R}^m$ drawn independently at random from unit variance spherical Gaussian $p(x)$. Note, the $u^{(i)}$s themselves are neither unit length (mean radius $\sqrt{m}$) nor necessarily orthogonal.

Random Projections to speed NN Search (NNS)

- In lower dimensionality space $m' << m$, NNS much faster.
- Possible solution: Let $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ be a projection from m to $m' \ll m$ dimensions. Is x and y near if $f(x)$ near $f(y)$? $x = (100, 0, 0, \ldots, 0) \in \mathbb{R}^m$, $y = (0, 0, \ldots, 0) \in \mathbb{R}^m$.Ex-a: $f_a(x) = x_1 \in \mathbb{R}$? Ex-b: $f_b(x) = x_2 \in \mathbb{R}$?
- Approach: pick m' vectors $u^{(1)}, u^{(2)}, \ldots, u^{(m')} \in \mathbb{R}^m$ drawn independently at random from unit variance spherical Gaussian $p(x)$. Note, the $u^{(i)}$s themselves are neither unit length (mean radius $\sqrt{m}$) nor necessarily orthogonal. Define $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ as

$$f(x) = (\langle u^{(1)}, x \rangle, \langle u^{(2)}, x \rangle, \ldots, \langle u^{(m')}, x \rangle) \in \mathbb{R}^{m'} \tag{7.8}$$

as the vector of inner (dot) products.

Random Projections to speed NN Search (NNS)

- In lower dimensionality space $m' << m$, NNS much faster.
- Possible solution: Let $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ be a projection from m to $m' \ll m$ dimensions. Is x and y near if $f(x)$ near $f(y)$? $x = (100, 0, 0, \ldots, 0) \in \mathbb{R}^m$, $y = (0, 0, \ldots, 0) \in \mathbb{R}^m$.Ex-a: $f_a(x) = x_1 \in \mathbb{R}$? Ex-b: $f_b(x) = x_2 \in \mathbb{R}$?
- Approach: pick m' vectors $u^{(1)}, u^{(2)}, \ldots, u^{(m')} \in \mathbb{R}^m$ drawn independently at random from unit variance spherical Gaussian $p(x)$. Note, the $u^{(i)}$s themselves are neither unit length (mean radius $\sqrt{m}$) nor necessarily orthogonal. Define $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ as

$$f(x) = (\langle u^{(1)}, x \rangle, \langle u^{(2)}, x \rangle, \ldots, \langle u^{(m')}, x \rangle) \in \mathbb{R}^{m'} \tag{7.8}$$

  as the vector of inner (dot) products.
- We claim that for any x, $\|f(x)\|_2 \approx \sqrt{m'}\|x\|_2$ with high probability.

Random Projections to speed NN Search (NNS)

- In lower dimensionality space $m' << m$, NNS much faster.
- Possible solution: Let $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ be a projection from m to $m' \ll m$ dimensions. Is x and y near if $f(x)$ near $f(y)$? $x = (100, 0, 0, \ldots, 0) \in \mathbb{R}^m$, $y = (0, 0, \ldots, 0) \in \mathbb{R}^m$.Ex-a: $f_a(x) = x_1 \in \mathbb{R}$? Ex-b: $f_b(x) = x_2 \in \mathbb{R}$?
- Approach: pick m' vectors $u^{(1)}, u^{(2)}, \ldots, u^{(m')} \in \mathbb{R}^m$ drawn independently at random from unit variance spherical Gaussian $p(x)$. Note, the $u^{(i)}$s themselves are neither unit length (mean radius $\sqrt{m}$) nor necessarily orthogonal. Define $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ as

$$f(x) = (\langle u^{(1)}, x \rangle, \langle u^{(2)}, x \rangle, \ldots, \langle u^{(m')}, x \rangle) \in \mathbb{R}^{m'} \tag{7.8}$$

  as the vector of inner (dot) products.
- We claim that for <u>any</u> x, $\|f(x)\|_2 \approx \sqrt{m'}\|x\|_2$ with high probability. Note, $u^{(i)}$s are not unit, so norms in lower dimensional space are larger.

Random Projections to speed NN Search (NNS)

- In lower dimensionality space $m' << m$, NNS much faster.
- Possible solution: Let $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ be a projection from m to $m' \ll m$ dimensions. Is x and y near if $f(x)$ near $f(y)$? $x = (100, 0, 0, \ldots, 0) \in \mathbb{R}^m$, $y = (0, 0, \ldots, 0) \in \mathbb{R}^m$.Ex-a: $f_a(x) = x_1 \in \mathbb{R}$? Ex-b: $f_b(x) = x_2 \in \mathbb{R}$?
- Approach: pick m' vectors $u^{(1)}, u^{(2)}, \ldots, u^{(m')} \in \mathbb{R}^m$ drawn independently at random from unit variance spherical Gaussian $p(x)$. Note, the $u^{(i)}$s themselves are neither unit length (mean radius $\sqrt{m}$) nor necessarily orthogonal. Define $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ as

$$f(x) = (\langle u^{(1)}, x\rangle, \langle u^{(2)}, x\rangle, \ldots, \langle u^{(m')}, x\rangle) \in \mathbb{R}^{m'} \tag{7.8}$$

  as the vector of inner (dot) products.
- We claim that for any x, $\|f(x)\|_2 \approx \sqrt{m'}\|x\|_2$ with high probability. Note, $u^{(i)}$s are not unit, so norms in lower dimensional space are larger.
- To the extent this is true, distances can be approximated in low-D:

$$\|x - y\|_2 = \frac{\sqrt{m'}}{\sqrt{m'}}\|x - y\|_2 \approx \frac{1}{\sqrt{m'}}\|f(x - y)\|_2 = \frac{1}{\sqrt{m'}}\|f(x) - f(y)\|_2 \tag{7.9}$$

Random Projection Theorem

- Specifically, it can be shown that:

Random Projection Theorem

- Specifically, it can be shown that:

Theorem 7.4.2 (Random Projection Theorem)

Let $x \in \mathbb{R}^m$ and f as given. Then for $\epsilon \in (0, 1)$,

$$\Pr\left(\left|\|f(x)\|_2 - \sqrt{m'}\|x\|_2\right| \geq \epsilon\sqrt{m'}\|x\|_2\right) \leq 3e^{-cm'\epsilon^2} \tag{7.10}$$

where probability is taken over random draws of u vectors.

Random Projection Theorem

- Specifically, it can be shown that:

Theorem 7.4.2 (Random Projection Theorem)

Let $x \in \mathbb{R}^m$ and f as given. Then for $\epsilon \in (0, 1)$,

$$\Pr\left(\left|\|f(x)\|_2 - \sqrt{m'}\|x\|_2\right| \geq \epsilon\sqrt{m'}\|x\|_2\right) \leq 3e^{-cm'\epsilon^2} \tag{7.10}$$

where probability is taken over random draws of u vectors.

- Follows from the Gaussian Annulus Theorem.

Random Projection Theorem

- Specifically, it can be shown that:

Theorem 7.4.2 (Random Projection Theorem)

Let $x \in \mathbb{R}^m$ and f as given. Then for $\epsilon \in (0, 1)$,

$$\Pr\left(\left|\|f(x)\|_2 - \sqrt{m'}\|x\|_2\right| \geq \epsilon\sqrt{m'}\|x\|_2\right) \leq 3e^{-cm'\epsilon^2} \tag{7.10}$$

where probability is taken over random draws of u vectors.

- Follows from the Gaussian Annulus Theorem.
- So probability of norm of projection deviating from $\sqrt{m'}\|x\|_2$ is exponentially small in m'.

Random Projection Theorem

- Specifically, it can be shown that:

Theorem 7.4.2 (Random Projection Theorem)

Let $x \in \mathbb{R}^m$ and f as given. Then for $\epsilon \in (0, 1)$,

$$\Pr\left(\left|\|f(x)\|_2 - \sqrt{m'}\|x\|_2\right| \geq \epsilon\sqrt{m'}\|x\|_2\right) \leq 3e^{-cm'\epsilon^2} \tag{7.10}$$

where probability is taken over random draws of u vectors.

- Follows from the Gaussian Annulus Theorem.
- So probability of norm of projection deviating from $\sqrt{m'}\|x\|_2$ is exponentially small in m'.
- Given n samples $x^{(i)}$, the probability of any of the $O(n^2)$ distances deviating by such an amount is also small (say $\leq \delta$) by the union bound ($\Pr(\cup_i A_i) \leq \sum_i \Pr(A_i)$) as long as if m' is not too small, i.e., $m' \geq 3\log(n)/(c\epsilon^2)$, giving $\delta = 3/n$.

Random Projection Theorem

- Specifically, it can be shown that:

Theorem 7.4.2 (Random Projection Theorem)

Let $x \in \mathbb{R}^m$ and f as given. Then for $\epsilon \in (0, 1)$,

$$\Pr\left(\left|\|f(x)\|_2 - \sqrt{m'}\|x\|_2\right| \geq \epsilon\sqrt{m'}\|x\|_2\right) \leq 3e^{-cm'\epsilon^2} \tag{7.10}$$

where probability is taken over random draws of u vectors.

- Follows from the Gaussian Annulus Theorem.
- So probability of norm of projection deviating from $\sqrt{m'}\|x\|_2$ is exponentially small in m'.
- Given n samples $x^{(i)}$, the probability of any of the $O(n^2)$ distances deviating by such an amount is also small (say $\leq \delta$) by the union bound ($\Pr(\cup_i A_i) \leq \sum_i \Pr(A_i)$) as long as if m' is not too small, i.e., $m' \geq 3\log(n)/(c\epsilon^2)$, giving $\delta = 3/n$.
- This means we can accurately calculate approximate distances in m'-dimensional space after this random projection. w.h.p.

Johnson-Lindenstrauss Lemma

Theorem 7.4.3 (Johnson-Lindenstrauss Lemma)

For any $\epsilon \in (0,1)$, any n, let $m' \geq \frac{3}{c\epsilon^2}\log n$. Let X be set of n points in $\mathbb{R}^m$. The random projection $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ has the property that for all $x^{(i)}, x^{(j)}$, then with probability at least $1 - 3/(2n)$, we have:

$$(1-\epsilon)\sqrt{m'}\|x^{(i)} - x^{(j)}\| \leq \|f(x^{(i)}) - f(x^{(j)})\| \leq (1+\epsilon)\sqrt{m'}\|x^{(i)} - x^{(j)}\| \tag{7.11}$$

- I.e., the probability of all $\binom{n}{2} < n^2/2$ pairs of points so deviating is at most $3/(2n)$, so so can be used to do fast NNS.

Johnson-Lindenstrauss Lemma

Theorem 7.4.3 (Johnson-Lindenstrauss Lemma)

For any $\epsilon \in (0,1)$, any n, let $m' \geq \frac{3}{c\epsilon^2}\log n$. Let X be set of n points in $\mathbb{R}^m$. The random projection $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ has the property that for all $x^{(i)}, x^{(j)}$, then with probability at least $1 - 3/(2n)$, we have:

$$(1-\epsilon)\sqrt{m'}\|x^{(i)} - x^{(j)}\| \leq \|f(x^{(i)}) - f(x^{(j)})\| \leq (1+\epsilon)\sqrt{m'}\|x^{(i)} - x^{(j)}\| \tag{7.11}$$

- I.e., the probability of <u>all</u> $\binom{n}{2} < n^2/2$ pairs of points so deviating is at most $3/(2n)$, so so can be used to do fast NNS.
- m' grows only logarithmically with n. In high dimensions $m' \ll m$ even for small ϵ.

Johnson-Lindenstrauss Lemma

Theorem 7.4.3 (Johnson-Lindenstrauss Lemma)

For any $\epsilon \in (0,1)$, any n, let $m' \geq \frac{3}{c\epsilon^2}\log n$. Let X be set of n points in $\mathbb{R}^m$. The random projection $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ has the property that for all $x^{(i)}, x^{(j)}$, then with probability at least $1 - 3/(2n)$, we have:

$$(1-\epsilon)\sqrt{m'}\|x^{(i)} - x^{(j)}\| \leq \|f(x^{(i)}) - f(x^{(j)})\| \leq (1+\epsilon)\sqrt{m'}\|x^{(i)} - x^{(j)}\| \tag{7.11}$$

- I.e., the probability of <u>all</u> $\binom{n}{2} < n^2/2$ pairs of points so deviating is at most $3/(2n)$, so so can be used to do fast NNS.
- m' grows only logarithmically with n. In high dimensions $m' \ll m$ even for small ϵ.
- Example: $n = 10^{10}$, in proof of theorem $c = 96$ works, $\epsilon = 0.05$ then $m' \geq 3/(96 \cdot 0.05^2)\log(10^{10}) = 288$.

$O(n^2 m) / O(n^2 \cdot m')$

Johnson-Lindenstrauss Lemma

Theorem 7.4.3 (Johnson-Lindenstrauss Lemma)

For any $\epsilon \in (0,1)$, any n, let $m' \geq \frac{3}{c\epsilon^2}\log n$. Let X be set of n points in $\mathbb{R}^m$. The random projection $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ has the property that for all $x^{(i)}, x^{(j)}$, then with probability at least $1 - 3/(2n)$, we have:

$$(1-\epsilon)\sqrt{m'}\|x^{(i)} - x^{(j)}\| \leq \|f(x^{(i)}) - f(x^{(j)})\| \leq (1+\epsilon)\sqrt{m'}\|x^{(i)} - x^{(j)}\| \tag{7.11}$$

- I.e., the probability of <u>all</u> $\binom{n}{2} < n^2/2$ pairs of points so deviating is at most $3/(2n)$, so so can be used to do fast NNS.
- m' grows only logarithmically with n. In high dimensions $m' \ll m$ even for small ϵ.
- Example: $n = 10^{10}$, in proof of theorem $c = 96$ works, $\epsilon = 0.05$ then $m' \geq 3/(96 \cdot 0.05^2)\log(10^{10}) = 288$. Speedup: $10^{10}/288 = 34$ million!

Johnson-Lindenstrauss Lemma

Theorem 7.4.3 (Johnson-Lindenstrauss Lemma)

For any $\epsilon \in (0,1)$, any n, let $m' \geq \frac{3}{c\epsilon^2}\log n$. Let X be set of n points in $\mathbb{R}^m$. The random projection $f : \mathbb{R}^m \to \mathbb{R}^{m'}$ has the property that for all $x^{(i)}, x^{(j)}$, then with probability at least $1 - 3/(2n)$, we have:

$$(1-\epsilon)\sqrt{m'}\|x^{(i)} - x^{(j)}\| \leq \|f(x^{(i)}) - f(x^{(j)})\| \leq (1+\epsilon)\sqrt{m'}\|x^{(i)} - x^{(j)}\| \tag{7.11}$$

- I.e., the probability of all $\binom{n}{2} < n^2/2$ pairs of points so deviating is at most $3/(2n)$, so so can be used to do fast NNS.
- m' grows only logarithmically with n. In high dimensions $m' \ll m$ even for small ϵ.
- Example: $n = 10^{10}$, in proof of theorem $c = 96$ works, $\epsilon = 0.05$ then $m' \geq 3/(96 \cdot 0.05^2)\log(10^{10}) = 288$. Speedup: $10^{10}/288 = 34$ million!
- Other examples: $(n = 10000, m' = 115)$, $(n = 1000, m' = 86)$, $(n = 100, m' = 57)$.

Approximate Nearest Neighbor Search: LSH

- Recall: nearest neighbor search (NNS) problem: Given a set of n points $\left\{x^{(i)}\right\}_{i=1}^{n}$, $x^{(i)} \in \mathbb{R}^m$, finding nearest to query $q \in \mathbb{R}^m$ is $\operatorname{argmin}_i d(x^{(i)}, q)$.

Approximate Nearest Neighbor Search: LSH

- Recall: nearest neighbor search (NNS) problem: Given a set of n points $\left\{x^{(i)}\right\}_{i=1}^{n}$, $x^{(i)} \in \mathbb{R}^m$, finding nearest to query $q \in \mathbb{R}^m$ is $\operatorname{argmin}_i d(x^{(i)}, q)$.
- Rather than trying to find $\operatorname{argmin}_i d(x^{(i)}, q)$ we find some $x^{(i)}$ so that $d(q, x^{(i)}) \leq c \min_j d(q, x^{(j)})$ for some $c \geq 1$ and with some (high) probability.

Approximate Nearest Neighbor Search: LSH

- Recall: nearest neighbor search (NNS) problem: Given a set of n points $\left\{x^{(i)}\right\}_{i=1}^{n}$, $x^{(i)} \in \mathbb{R}^m$, finding nearest to query $q \in \mathbb{R}^m$ is $\operatorname{argmin}_i d(x^{(i)}, q)$.
- Rather than trying to find $\operatorname{argmin}_i d(x^{(i)}, q)$ we find some $x^{(i)}$ so that $d(q, x^{(i)}) \leq c \min_j d(p, x^{(j)})$ for some $c \geq 1$ and with some (high) probability.
- Locality Sensitive Hashing (LSH) helps us achieve this.

Approximate Nearest Neighbor Search: LSH

- Recall: nearest neighbor search (NNS) problem: Given a set of n points $\left\{x^{(i)}\right\}_{i=1}^{n}$, $x^{(i)} \in \mathbb{R}^m$, finding nearest to query $q \in \mathbb{R}^m$ is $\operatorname{argmin}_i d(x^{(i)}, q)$.
- Rather than trying to find $\operatorname{argmin}_i d(x^{(i)}, q)$ we find some $x^{(i)}$ so that $d(q, x^{(i)}) \leq c \min_j d(p, x^{(j)})$ for some $c \geq 1$ and with some (high) probability.
- Locality Sensitive Hashing (LSH) helps us achieve this.
- Normally, cryptographic hash functions (e.g., SHA-1, SHA-256, etc., computed with linux's `shasum`) mean small differences map to different buckets, unlikely collision.

Approximate Nearest Neighbor Search: LSH

- Recall: nearest neighbor search (NNS) problem: Given a set of n points $\left\{x^{(i)}\right\}_{i=1}^{n}$, $x^{(i)} \in \mathbb{R}^m$, finding nearest to query $q \in \mathbb{R}^m$ is $\operatorname{argmin}_i d(x^{(i)}, q)$.
- Rather than trying to find $\operatorname{argmin}_i d(x^{(i)}, q)$ we find some $x^{(i)}$ so that $d(q, x^{(i)}) \leq c \min_j d(p, x^{(j)})$ for some $c \geq 1$ and with some (high) probability.
- Locality Sensitive Hashing (LSH) helps us achieve this.
- Normally, cryptographic hash functions (e.g., SHA-1, SHA-256, etc., computed with linux's `shasum`) mean small differences map to different buckets, unlikely collision.
- In LSH, by contrast, small differences likely to map to same bucket, collision likely with similar inputs.

Approximate Nearest Neighbor Search: LSH

- Recall: nearest neighbor search (NNS) problem: Given a set of n points $\left\{x^{(i)}\right\}_{i=1}^{n}$, $x^{(i)} \in \mathbb{R}^m$, finding nearest to query $q \in \mathbb{R}^m$ is $\operatorname{argmin}_i d(x^{(i)}, q)$.
- Rather than trying to find $\operatorname{argmin}_i d(x^{(i)}, q)$ we find some $x^{(i)}$ so that $d(q, x^{(i)}) \leq c \min_j d(p, x^{(j)})$ for some $c \geq 1$ and with some (high) probability.
- Locality Sensitive Hashing (LSH) helps us achieve this.
- Normally, cryptographic hash functions (e.g., SHA-1, SHA-256, etc., computed with linux's `shasum`) mean small differences map to different buckets, unlikely collision.
- In LSH, by contrast, small differences likely to map to same bucket, collision likely with similar inputs.
- LSH is similar to clustering (which we will discuss in a future lecture).

LSH definition

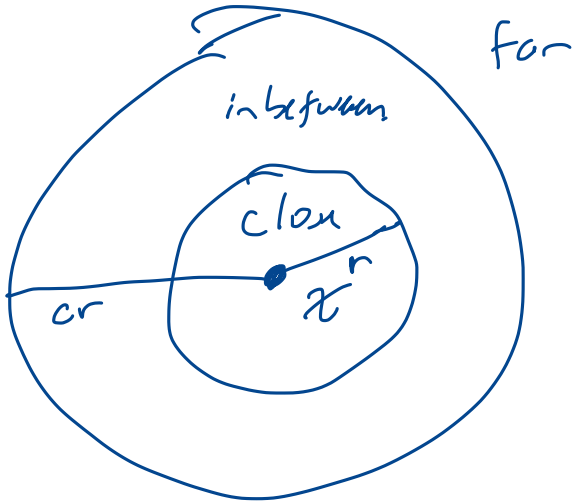
- Given two probabilities p_1, p_2 with $p_1 \geq p_2$, a constant $c > 1$, and a distance metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, $\mathcal{X}$ is the domain of the input (e.g., $\mathbb{R}^m$).

LSH definition

- Given two probabilities p_1, p_2 with $p_1 \geq p_2$, a constant $c > 1$, and a distance metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, $\mathcal{X}$ is the domain of the input (e.g., $\mathbb{R}^m$).
- A family of hash functions is defined relative to probabilities p_1, p_2, the constant c, a notion of closeness (radius r), and the distance metric d.

$d(x,y) < r \Rightarrow x, y$ close

$d(x,y) > c \cdot r \Rightarrow x, y$ far



LSH definition

- Given two probabilities p_1, p_2 with $p_1 \geq p_2$, a constant $c > 1$, and a distance metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, $\mathcal{X}$ is the domain of the input (e.g., $\mathbb{R}^m$).
- A family of hash functions is defined relative to probabilities p_1, p_2, the constant c, a notion of closeness (radius r), and the distance metric d.

Definition 7.4.4 (LSH)

A family of hash functions h is said to be (r, c, p_1, p_2)-LSH with $p_1 \geq p_2$, $r > 0$, and $c > 1$ if (when h is drawn uniformly at random from H):

family

LSH definition

- Given two probabilities p_1, p_2 with $p_1 \geq p_2$, a constant $c > 1$, and a distance metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, $\mathcal{X}$ is the domain of the input (e.g., $\mathbb{R}^m$).
- A family of hash functions is defined relative to probabilities p_1, p_2, the constant c, a notion of closeness (radius r), and the distance metric d.

Definition 7.4.4 (LSH)

A family of hash functions h is said to be (r, c, p_1, p_2)-LSH with $p_1 \geq p_2$, $r > 0$, and $c > 1$ if (when h is drawn uniformly at random from H):

(a) $\Pr[h(x) = h(y)] \geq p_1$ when $d(x, y) \leq r$ ("close" points have high probability of mapping to same bucket).

LSH definition

- Given two probabilities p_1, p_2 with $p_1 \geq p_2$, a constant $c > 1$, and a distance metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, $\mathcal{X}$ is the domain of the input (e.g., $\mathbb{R}^m$).
- A family of hash functions is defined relative to probabilities p_1, p_2, the constant c, a notion of closeness (radius r), and the distance metric d.

Definition 7.4.4 (LSH)

A family of hash functions h is said to be (r, c, p_1, p_2)-LSH with $p_1 \geq p_2$, $r > 0$, and $c > 1$ if (when h is drawn uniformly at random from H):

(a) $\Pr[h(x) = h(y)] \geq p_1$ when $d(x, y) \leq r$ (close points have high probability of mapping to same bucket).

(b) $\Pr[h(x) = h(y)] \leq p_2$ when $d(x, y) \geq cr$ ("distant" points have low probability mapping to same bucket).

LSH definition

- Given two probabilities p_1, p_2 with $p_1 \geq p_2$, a constant $c > 1$, and a distance metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, $\mathcal{X}$ is the domain of the input (e.g., $\mathbb{R}^m$).
- A family of hash functions is defined relative to probabilities p_1, p_2, the constant c, a notion of closeness (radius r), and the distance metric d.

Definition 7.4.4 (LSH)

A family of hash functions h is said to be (r, c, p_1, p_2)-LSH with $p_1 \geq p_2$, $r > 0$, and $c > 1$ if (when h is drawn uniformly at random from H):

(a) $\Pr[h(x) = h(y)] \geq p_1$ when $d(x, y) \leq r$ (close points have high probability of mapping to same bucket).

(b) $\Pr[h(x) = h(y)] \leq p_2$ when $d(x, y) \geq cr$ (distant points have low probability mapping to same bucket).

- When things are close ($d(x, y) \leq r$) then we have high probability ($\geq p_1$) that the two vectors x, y will hash to the same place ($h(x) = h(y)$).

LSH definition

- Given two probabilities p_1, p_2 with $p_1 \geq p_2$, a constant $c > 1$, and a distance metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, $\mathcal{X}$ is the domain of the input (e.g., $\mathbb{R}^m$).
- A family of hash functions is defined relative to probabilities p_1, p_2, the constant c, a notion of closeness (radius r), and the distance metric d.

LSH Allows a tradeoff between computation and statistical accuracy.

Definition 7.4.4 (LSH)

A family H of hash functions $h \in H$ is said to be (r, c, p_1, p_2)-LSH with $p_1 \geq p_2$, $r > 0$, and $c > 1$ if (when h is drawn uniformly at random from H):

(a) $\Pr[h(x) = h(y)] \geq p_1$ when $d(x, y) \leq r$ (close points have high probability of mapping to same bucket).

(b) $\Pr[h(x) = h(y)] \leq p_2$ when $d(x, y) \geq cr$ (distant points have low probability mapping to same bucket).

- When things are close ($d(x, y) \leq r$) then we have high probability ($\geq p_1$) that the two vectors x, y will hash to the same place ($h(x) = h(y)$). When things are far ($d(x, y) \geq cr$), then we have low probability ($\leq p_2$) that two vectors x, y will hash to the same place.

LSH instances

Def: Chia-Lin Distance:
given $x, y \in \mathcal{X}$, $ch(x,y) =$,- - - -

- We can define LSH hash functions for a variety of problems, including for the

LSH instances

- We can define LSH hash functions for a variety of problems, including for the
  - Hamming metric, $\mathcal{X} = [0,1]^m$, $d(x,y) = \sum_{i=1}^m \mathbf{1}_{\{x_i \neq y_i\}}$.

LSH instances

- We can define LSH hash functions for a variety of problems, including for the
  - Hamming metric, $\mathcal{X} = [0,1]^m$, $d(x,y) = \sum_{i=1}^m \mathbf{1}_{\{x_i \neq y_i\}}$. A very simple example of a LSH function for Hamming is, $h(x) = x[a]$ for some $a \in \{0, 1, \ldots, m-1\}$ with a drawn uniformly at random. We study this in HW4.

To sample $h \sim H$,
we sample $a \in U[0, m-1]$
output $h(x) = x[a]$

LSH instances

- We can define LSH hash functions for a variety of problems, including for the
  - Hamming metric, $\mathcal{X} = [0,1]^m$, $d(x,y) = \sum_{i=1}^m \mathbf{1}_{\{x_i \neq y_i\}}$. A very simple example of a HSH function for Hamming is, $h(x) = x[a]$ for some $a \in \{0, 1, \ldots, m-1\}$ with a drawn uniformly at random. We study this in HW4.
  - Also there exist LSH functions for: (1) Jaccard similarity, $\mathcal{X} = 2^V$, $d(A,B) = \frac{|A \cap B|}{|A \cup B|}$ and for (2) Euclidean distance, $\mathcal{X} = \mathbb{R}^m$, $d(x,y) = \|x - y\|_2$.

LSH

- In general, h is drawn uniformly random from H in order to get above probabilities (i.e., $\Pr[h(x) = h(y)] \geq p_1$ when $d(x, y) \leq r$ and $\Pr[h(x) = h(y)] \leq p_2$ when $d(x, y) \geq cr$).

LSH

- In general, h is drawn uniformly random from H in order to get above probabilities (i.e., $\Pr[h(x) = h(y)] \geq p_1$ when $d(x, y) \leq r$ and $\Pr[h(x) = h(y)] \leq p_2$ when $d(x, y) \geq cr$).
- While $p_1 \geq p_2$, sometimes p_1 is too small (and thus $\Pr[h(x) = h(y)]$ is too small, meaning we never get a hit)

LSH

- In general, h is drawn uniformly random from H in order to get above probabilities (i.e., $\Pr[h(x) = h(y)] \geq p_1$ when $d(x, y) \leq r$ and $\Pr[h(x) = h(y)] \leq p_2$ when $d(x, y) \geq cr$).
- While $p_1 \geq p_2$, sometimes p_1 is too small (and thus $\Pr[h(x) = h(y)]$ is too small, meaning we never get a hit)
- To be able to have more flexible and wider range of values for p_1, p_2, we construct compound LSHs by using a set of LSHs multiple times, drawn randomly. I.e., $g(x) = (h_1(x), h_2(x), \ldots, h_k(x))$ where h_i is randomly drawn from H.

$k \ll m$

LSH

- In general, h is drawn uniformly random from H in order to get above probabilities (i.e., $\Pr[h(x) = h(y)] \geq p_1$ when $d(x, y) \leq r$ and $\Pr[h(x) = h(y)] \leq p_2$ when $d(x, y) \geq cr$).
- While $p_1 \geq p_2$, sometimes p_1 is too small (and thus $\Pr[h(x) = h(y)]$ is too small, meaning we never get a hit)
- To be able to have more flexible and wider range of values for p_1, p_2, we construct compound LSHs by using a set of LSHs multiple times, drawn randomly. I.e., $g(x) = (h_1(x), h_2(x), \ldots, h_k(x))$ where h_i is randomly drawn from H.
- We sometimes use multiple g's drawn randomly too.

LSH

- In general, h is drawn uniformly random from H in order to get above probabilities (i.e., $\Pr[h(x) = h(y)] \geq p_1$ when $d(x, y) \leq r$ and $\Pr[h(x) = h(y)] \leq p_2$ when $d(x, y) \geq cr$).
- While $p_1 \geq p_2$, sometimes p_1 is too small (and thus $\Pr[h(x) = h(y)]$ is too small, meaning we never get a hit)
- To be able to have more flexible and wider range of values for p_1, p_2, we construct compound LSHs by using a set of LSHs multiple times, drawn randomly. I.e., $g(x) = (h_1(x), h_2(x), \ldots, h_k(x))$ where h_i is randomly drawn from H.
- We sometimes use multiple g's drawn randomly too.
- Homework 4 explores this further, and we see we can get more flexible values for p_1 and p_2.

$x \in \mathcal{X}$ $|\mathcal{X}|$ is big. in bits

to represent a given $x \in \mathcal{X}$ takes many many bits.

$\mathcal{D} = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$ n is large but $n \ll 2^{|\mathcal{X}|}$

is "$q \in \text{Set}(\mathcal{D})$" ?

linear search $O(n)$

binary search $O(\log n)$

can we do $O(1)$

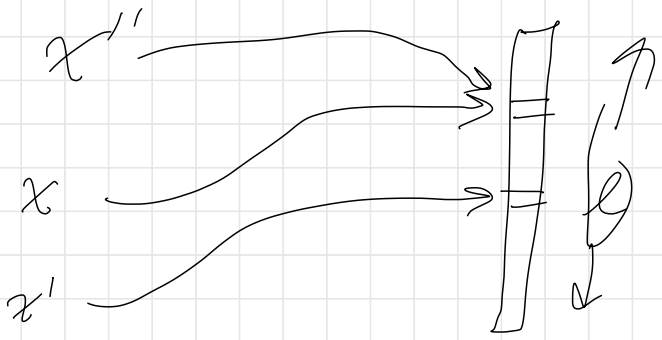
$h : \mathcal{X} \to \mathcal{Y}$

$|\mathcal{Y}| \ll |\mathcal{X}|$

$\therefore 2^{|\mathcal{Y}|} \ll 2^{|\mathcal{X}|}$

to check if x is in the set, we allocate an array of size ℓ and check slot

$(h(x) \bmod \ell)$ to see if it is occupied.



Recall, from lecture 3

- Recall from lecture 3 the following slide.

Linear Voronoi tessellation

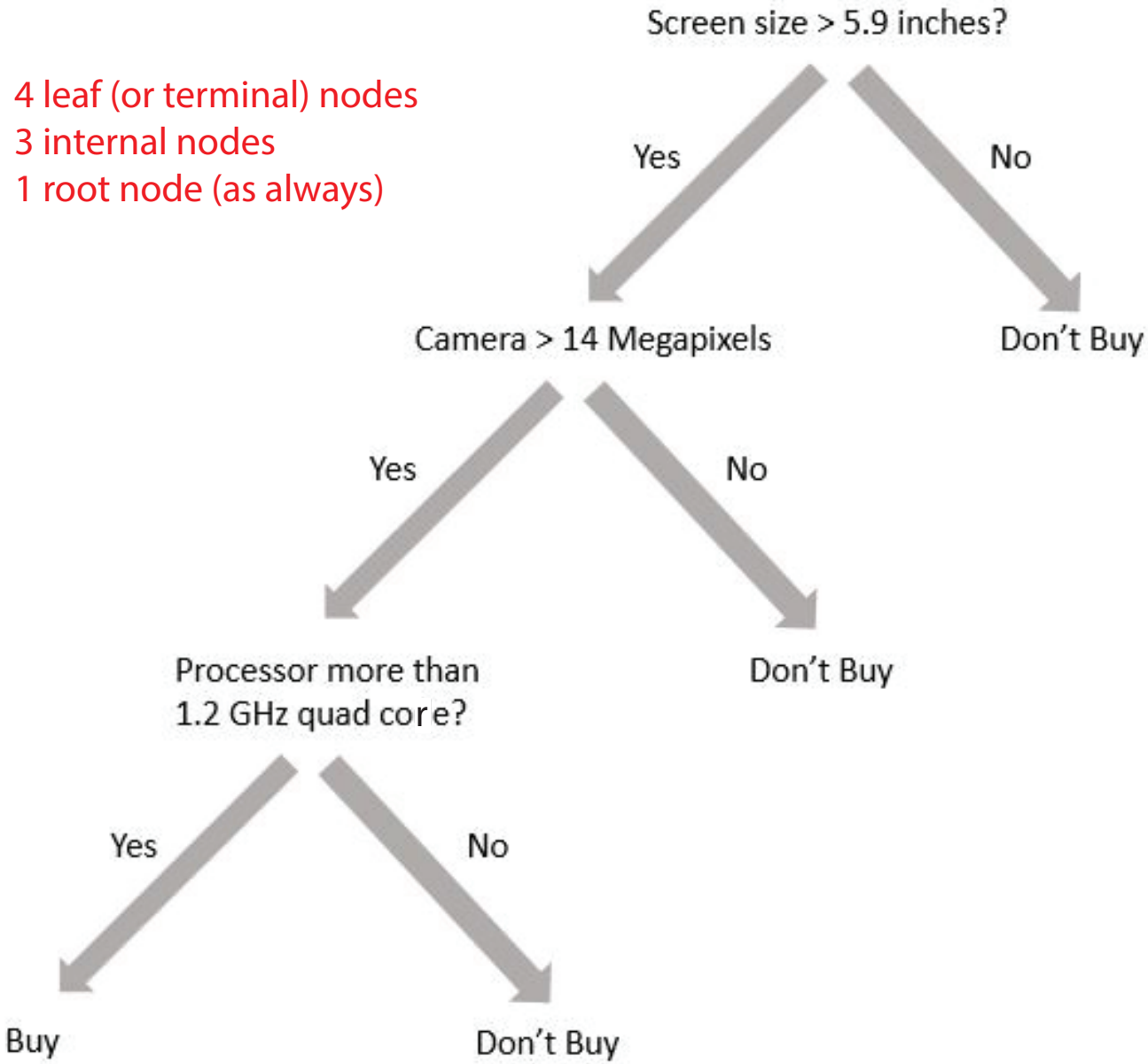
When discriminant functions are linear (i.e., $\langle \theta^{(j)}, x \rangle$) boundaries are lines. Example in two dimensions :

From `https://en.wikipedia.org/wiki/Voronoi_diagram`

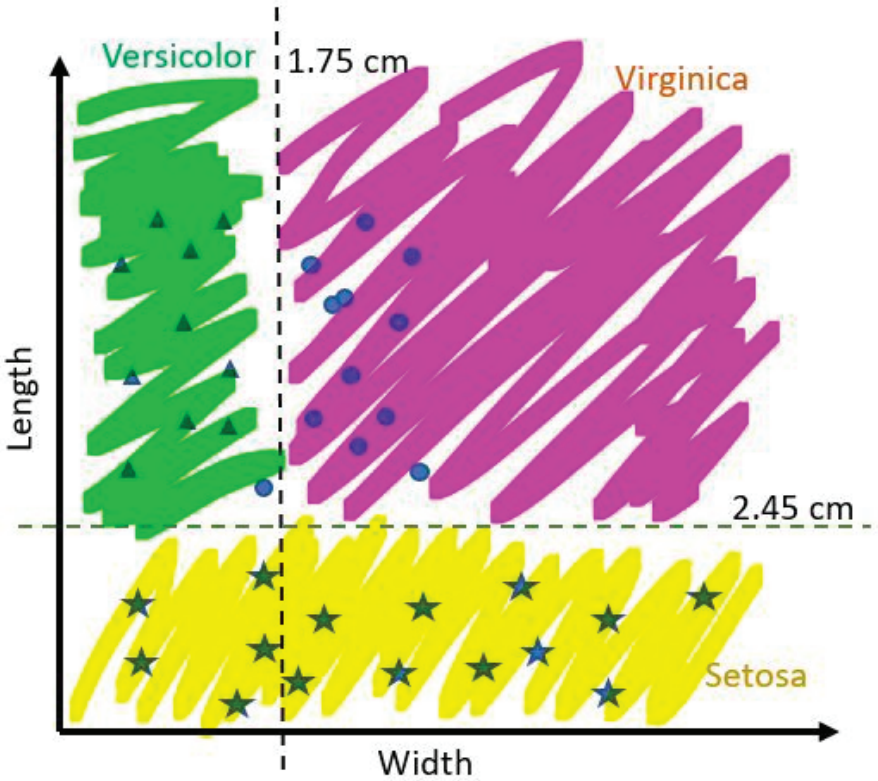
Using a Tree to Make Decisions

https://medium.com/x8-the-ai-community/decision-trees-an-intuitive-introduction-86c2b39c1a6c

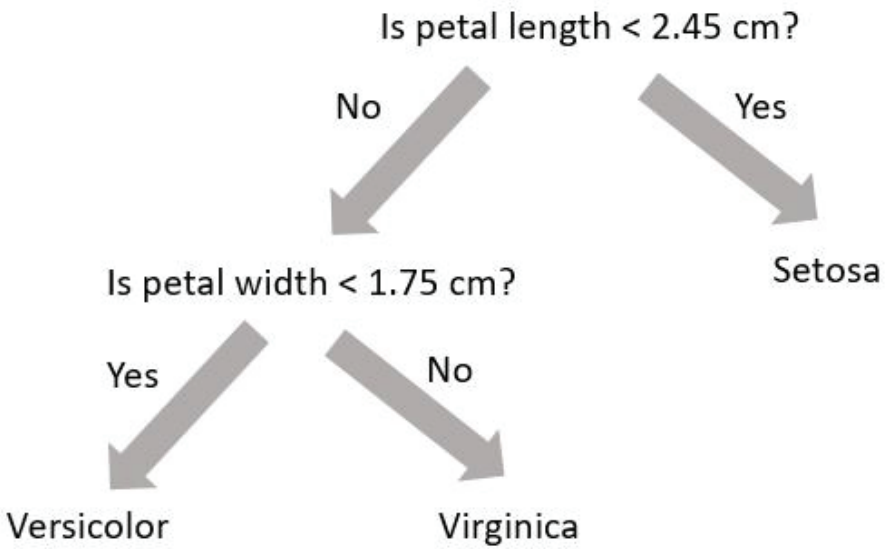
4 leaf (or terminal) nodes
3 internal nodes
1 root node (as always)



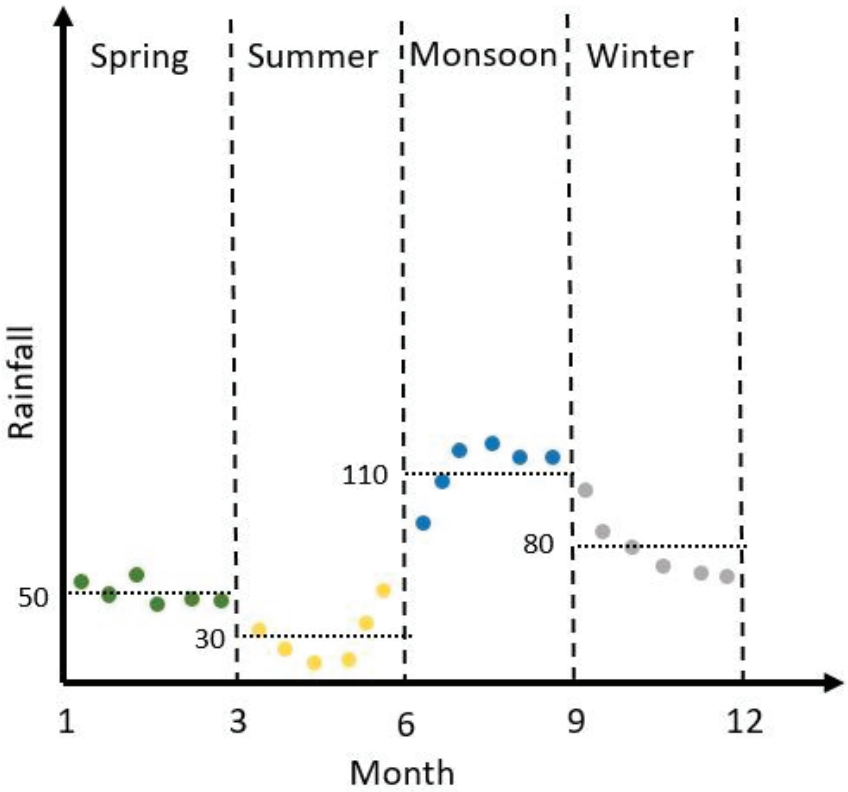
Iris flower classifier via petal length and width



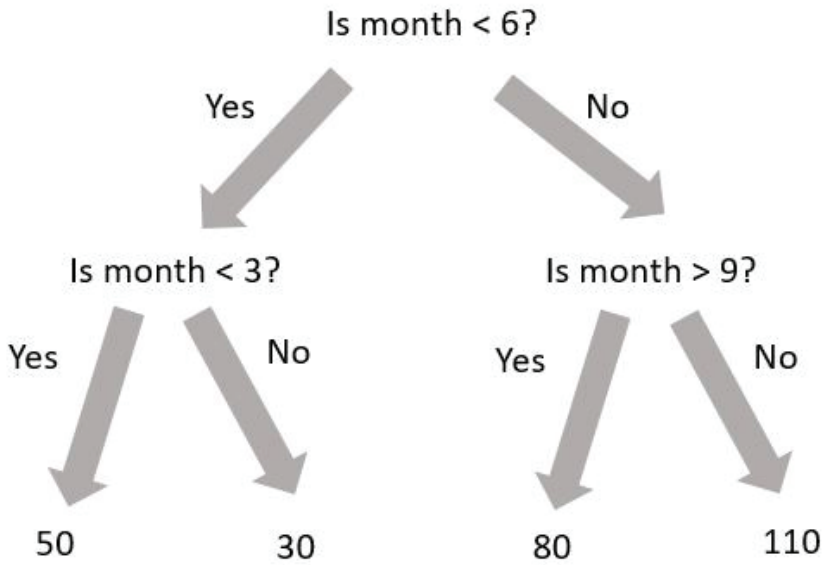
https://medium.com/x8-the-ai-community/decision-trees-an-intuitive-introduction-86c2b39c1a6c



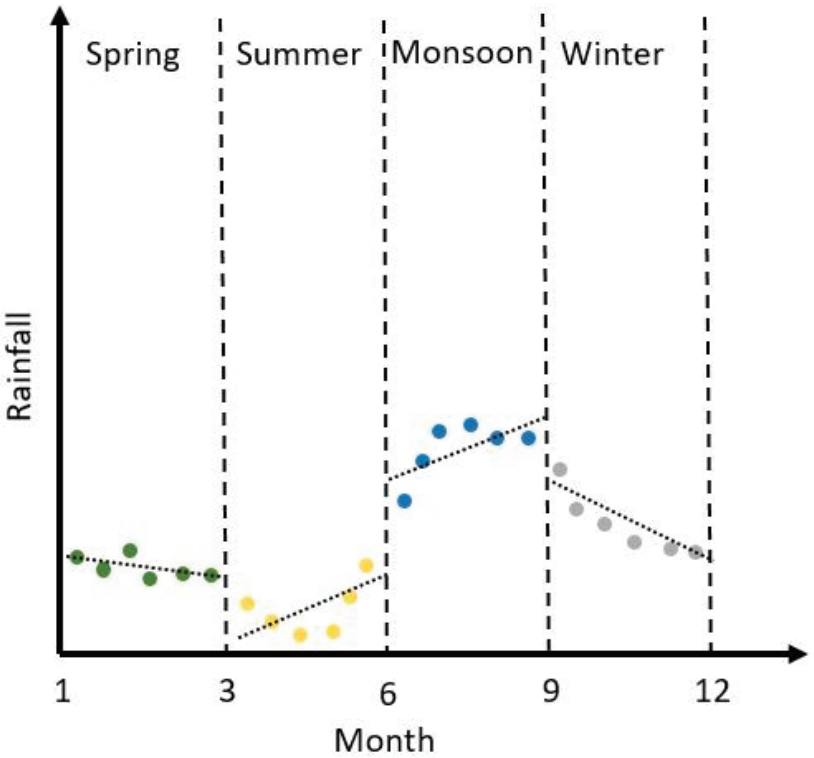
Regression with a tree, constant decisions



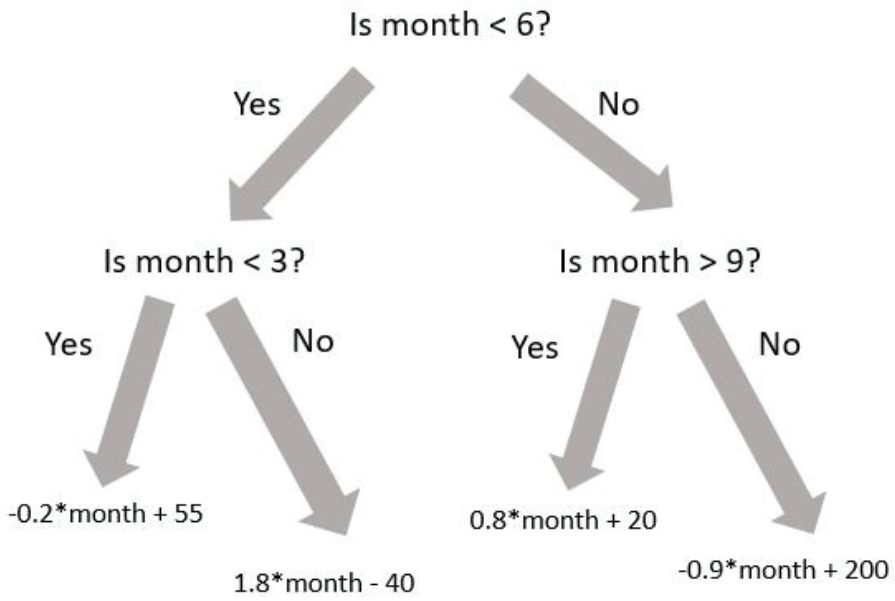
https://medium.com/x8-the-ai-community/decision-trees-an-intuitive-introduction-86c2b39c1a6c



Regression with a tree, linear decisions



https://medium.com/x8-the-ai-community/decision-trees-an-intuitive-introduction-86c2b39c1a6c



Decision Trees

- Partition space into rectangular regions, each region we make a final decision (i.e., regression value, or classification).

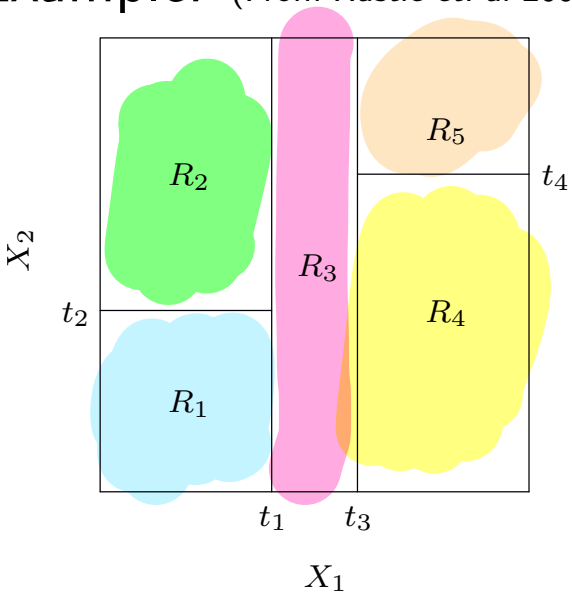
Decision Trees

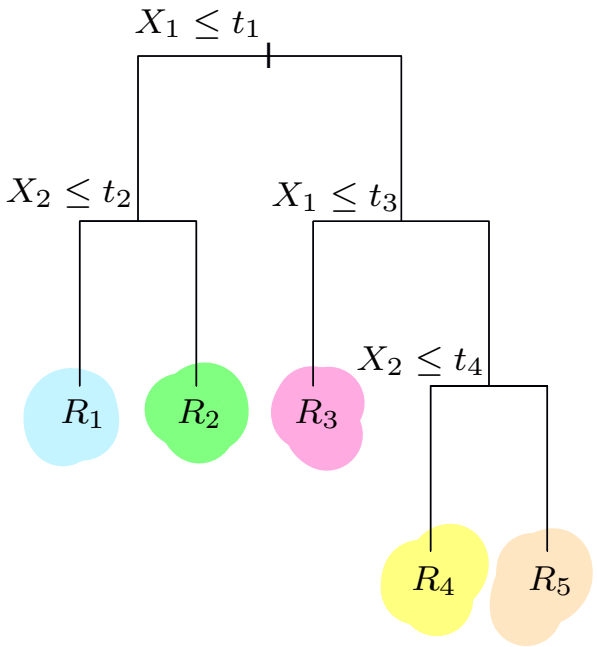
- Partition space into rectangular regions, each region we make a final decision (i.e., regression value, or classification).
- Regions formed by recursively splits space based on binary questions about one coordinate of input at a time (i.e., "Is $x_7 < 92$?")

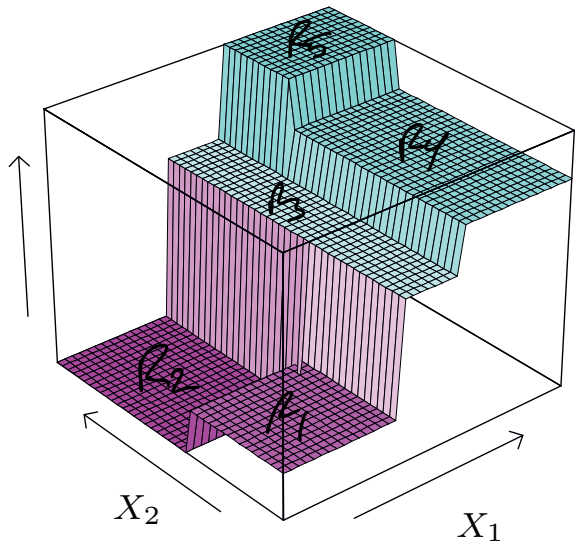
Decision Trees

- Partition space into rectangular regions, each region we make a final decision (i.e., regression value, or classification).
- Regions formed by recursively splits space based on binary questions about one coordinate of input at a time (i.e., "Is $x_7 < 92$?")

Example: (From Hastie et. al 2009 text).



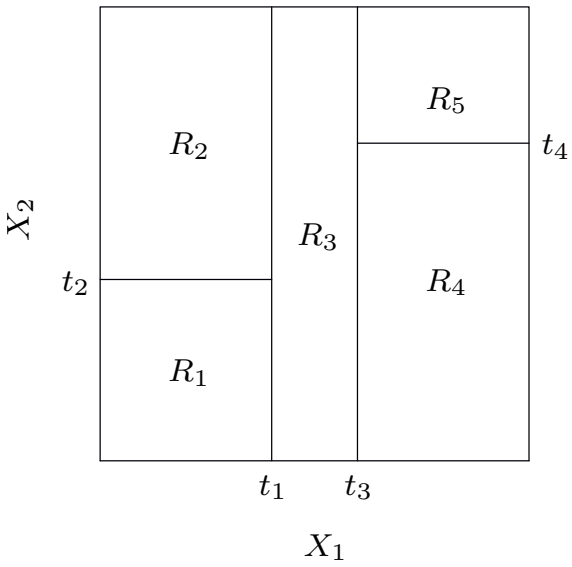


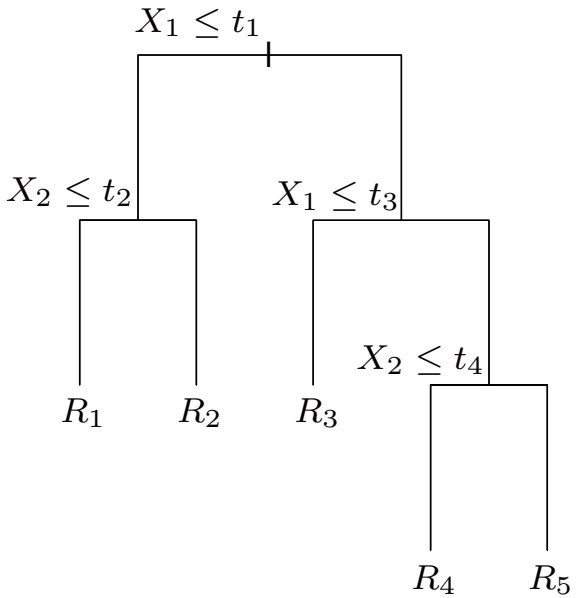


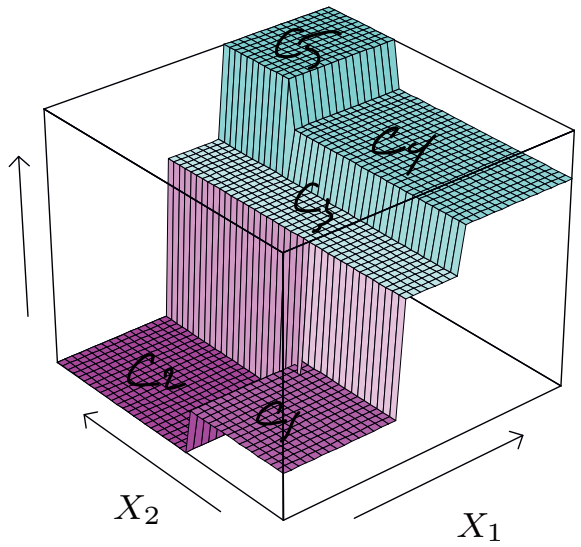
Decision Trees

- Partition space into rectangular regions, each region we make a final decision (i.e., regression value, or classification).
- Regions formed by recursively splits space based on binary questions about one coordinate of input at a time (i.e., "Is $x_7 < 92$?")

Example: (From Hastie et. al 2009 text).







Corresponds to

$$h(x) = \sum_{i=1}^{5} c_i \mathbf{1}_{\{x \in R_i\}} \tag{7.12}$$

Regression Trees

- Each of the r regions could correspond to its own function, i.e., $h_{\theta_k}(x)$. Here we keep it simple, each region is constant, $h_{\theta_k}(x) = c_k$.

$$h_\theta(x) = \sum_{k=1}^{r} c_k \mathbf{1}_{\{x \in R_k\}} \tag{7.13}$$

Regression Trees

- Each of the r regions could correspond to its own function, i.e., $h_{\theta_k}(x)$. Here we keep it simple, each region is constant, $h_{\theta_k}(x) = c_k$.

$$h_\theta(x) = \sum_{k=1}^{r} c_k \mathbf{1}_{\{x \in R_k\}} \tag{7.13}$$

- Given data $\mathcal{D} = \left\{x^{(i)}, y^{(i)}\right\}$, how to grow? Minimize square error $\sum_i (y^{(i)} - h_\theta(x^{(i)}))^2$.

θ determines a decision tree.

Regression Trees

- Each of the r regions could correspond to its own function, i.e., $h_{\theta_k}(x)$. Here we keep it simple, each region is constant, $h_{\theta_k}(x) = c_k$.

$$h_\theta(x) = \sum_{k=1}^{r} c_k \mathbf{1}_{\{x \in R_k\}} \tag{7.13}$$

- Given data $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}$, how to grow? Minimize square error $\sum_i (y^{(i)} - h_\theta(x^{(i)}))^2$.
- Once regions are decided, best within-region constant predictor is the simple average $\hat{c}_k = \text{average}(y^{(i)} | x^{(i)} \in R_k)$, i.e,:

(7.14)

$$\hat{c}_k = \frac{1}{|\{i : x^{(i)} \in R_k\}|} \sum_{i: x^{(i)} \in R_k} y^{(i)} \tag{7.15}$$

Regression Trees

- Each of the r regions could correspond to its own function, i.e., $h_{\theta_k}(x)$. Here we keep it simple, each region is constant, $h_{\theta_k}(x) = c_k$.

$$h_\theta(x) = \sum_{k=1}^{r} c_k \mathbf{1}_{\{x \in R_k\}} \tag{7.13}$$

- Given data $\mathcal{D} = \left\{x^{(i)}, y^{(i)}\right\}$, how to grow? Minimize square error $\sum_i (y^{(i)} - h_\theta(x^{(i)}))^2$.
- Once regions are decided, best within-region constant predictor is the simple average $\hat{c}_k = \text{average}(y^{(i)} | x^{(i)} \in R_k)$, i.e,:

(7.14)

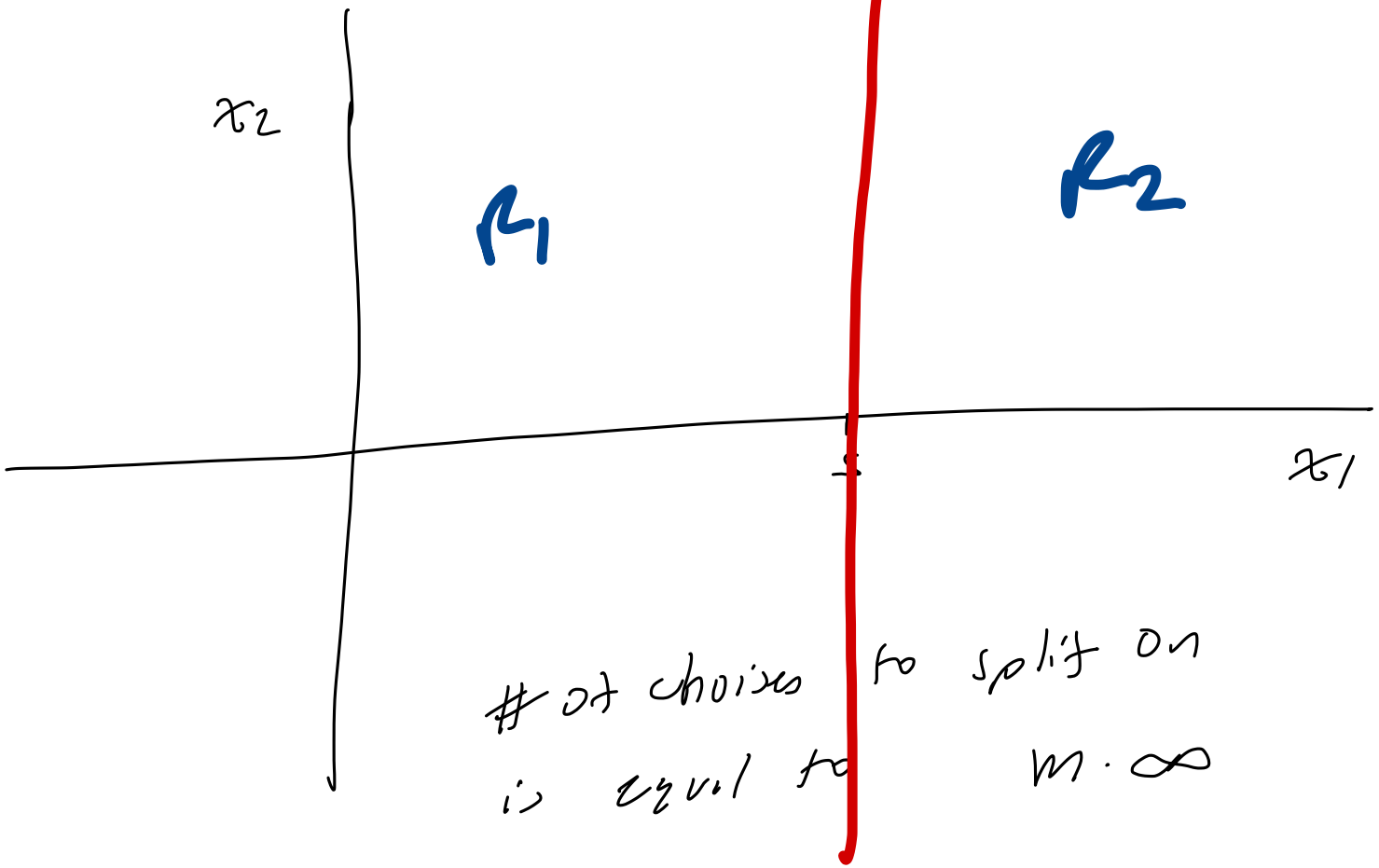
$$\hat{c}_k = \frac{1}{\left|\left\{i : x^{(i)} \in R_k\right\}\right|} \sum_{i: x^{(i)} \in R_k} y^{(i)} \tag{7.15}$$

- How to find the r regions?

Greedy strategy, binary splitting

- Splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes

$$R_1(j,s) = \{x : x_j \leq s\} \qquad R_2(j,s) = \{x : x_j > s\} \tag{7.16}$$



Greedy strategy, binary splitting

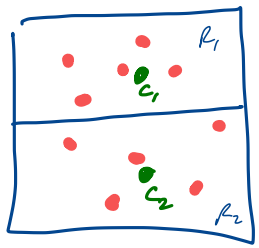
- Splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes

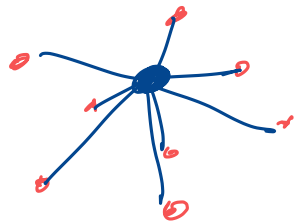
$$R_1(j,s) = \{x : x_j \leq s\} \qquad R_2(j,s) = \{x : x_j > s\} \tag{7.16}$$

- Use optimization, find the j, s that achieves the best (minimum) in:

$$\min_{j,s} \left[\sum_{x^{(i)} \in R_1(j,s)} (y^{(i)} - \hat{c}_1)^2 + \sum_{x^{(i)} \in R_2(j,s)} (y^{(i)} - \hat{c}_2)^2 \right] \tag{7.17}$$

where $\hat{c}_k = \text{average}(y^{(i)} | x^{(i)} \in R_k)$.





Given $\mathcal{D} = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$,

consider $J(c) = \frac{1}{2} \sum_{i=1}^{n} (x^{(i)} - c)^2$

& finding $\operatorname{argmin}_{c \in \mathbb{R}} J(c)$

$$\frac{\partial}{\partial c} J(c) = \sum_{i=1}^{n} (x^{(i)} - c) = 0$$

$$\sum_{i=1}^{n} x^{(i)} = \sum_{i=1}^{n} c = n \cdot c$$

$$\therefore c = \frac{1}{n} \sum_{i=1}^{n} x^{(i)}$$

Greedy strategy, binary splitting

- Splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes

$$R_1(j,s) = \{x : x_j \leq s\} \qquad R_2(j,s) = \{x : x_j > s\} \tag{7.16}$$

- Use optimization, find the j, s that achieves the best (minimum) in:

$$\min_{j,s} \left[\sum_{x^{(i)} \in R_1(j,s)} (y^{(i)} - \hat{c}_1)^2 + \sum_{x^{(i)} \in R_2(j,s)} (y^{(i)} - \hat{c}_2)^2 \right] \tag{7.17}$$

$O(m \cdot n)$

where $\hat{c}_k = \text{average}(y^{(i)} | x^{(i)} \in R_k)$.

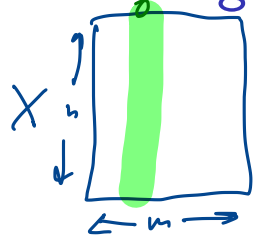
- For each j, can find s by sorting the data via x_j and scanning in order to achieve min. Select that j, s as the split point.

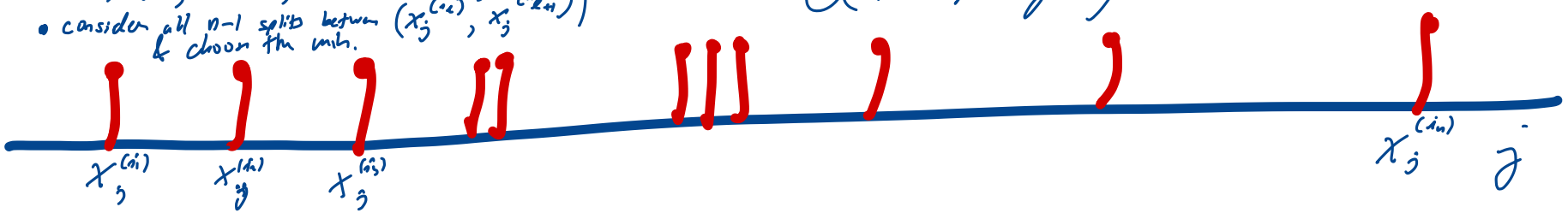
$D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ $x^{(i)} \in \mathbb{R}^m$ $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_j^{(i)}, \ldots, x_m^{(i)})^T$

For $j = 1, \ldots, m$

- Sort data based on $x_j^{(1)}, x_j^{(2)}, \ldots, x_j^{(n)}$
  $\rightarrow x_j^{(i_1)} \leq x_j^{(i_2)} \leq \cdots \leq x_j^{(i_n)}$
- consider all $n-1$ splits between $(x_j^{(i_k)}, x_j^{(i_{k+1})})$ & choose the min.

$O(m \cdot n)$
$O(m \cdot n \log n)$





Greedy strategy, binary splitting

- Splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes

$$R_1(j,s) = \{x : x_j \leq s\} \qquad R_2(j,s) = \{x : x_j > s\} \tag{7.16}$$

- Use optimization, find the j, s that achieves the best (minimum) in:

$$\min_{j,s} \left[\sum_{x^{(i)} \in R_1(j,s)} (y^{(i)} - \hat{c}_1)^2 + \sum_{x^{(i)} \in R_2(j,s)} (y^{(i)} - \hat{c}_2)^2 \right] \tag{7.17}$$

  where $\hat{c}_k = \text{average}(y^{(i)} | x^{(i)} \in R_k)$.
- For each j, can find s by sorting the data via x_j and scanning in order to achieve min. Select that j, s as the split point.
- Can also be seen as maximizing the MSE gain

$$\max_{j,s} \left(\sum_{x^{(i)}} (y^{(i)} - \hat{c}_0)^2 - \left[\sum_{x^{(i)} \in R_1(j,s)} (y^{(i)} - \hat{c}_1)^2 + \sum_{x^{(i)} \in R_2(j,s)} (y^{(i)} - \hat{c}_2)^2 \right] \right) \tag{7.18}$$

$\sum_{y^{(i)}} (y^{(i)} - \bar{c}_0)^2$

Greedy strategy, recursive binary splitting

- Given R_1 and R_2, apply same strategy to each of R_1 and R_2.

Greedy strategy, recursive binary splitting

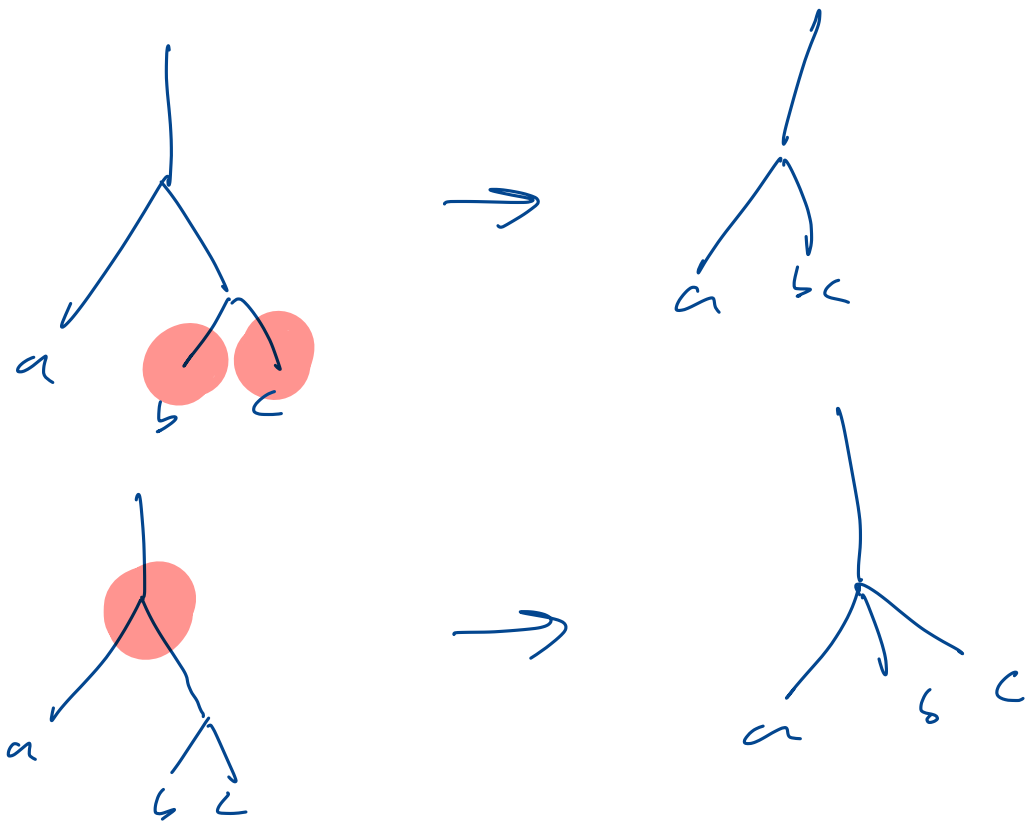
- Given R_1 and R_2, apply same strategy to each of R_1 and R_2.
- When to stop?

Greedy strategy, recursive binary splitting

- Given R_1 and R_2, apply same strategy to each of R_1 and R_2.
- When to stop? One option: stop when gain falls below threshold.

Greedy strategy, recursive binary splitting

- Given R_1 and R_2, apply same strategy to each of R_1 and R_2.
- When to stop? One option: stop when gain falls below threshold.
- Better: minimize cost/complexity. Grow to large tree T_0. Let $T \subset T_0$ be any subtree (via pruning internal nodes, merging regions).



Greedy strategy, recursive binary splitting

- Given R_1 and R_2, apply same strategy to each of R_1 and R_2.
- When to stop? One option: stop when gain falls below threshold.
- Better: minimize cost/complexity. Grow to large tree T_0. Let $T \subset T_0$ be any subtree (via pruning internal nodes, merging regions).
- Let: k be a terminal (leaf) node, with region R_k.

Greedy strategy, recursive binary splitting

- Given R_1 and R_2, apply same strategy to each of R_1 and R_2.
- When to stop? One option: stop when gain falls below threshold.
- Better: minimize cost/complexity. Grow to large tree T_0. Let $T \subset T_0$ be any subtree (via pruning internal nodes, merging regions).
- Let: k be a terminal (leaf) node, with region R_k. $|T|$ be the number of terminal nodes,

, ,

Greedy strategy, recursive binary splitting

- Given R_1 and R_2, apply same strategy to each of R_1 and R_2.
- When to stop? One option: stop when gain falls below threshold.
- Better: minimize cost/complexity. Grow to large tree T_0. Let $T \subset T_0$ be any subtree (via pruning internal nodes, merging regions).
- Let: k be a terminal (leaf) node, with region R_k. $|T|$ be the number of terminal nodes, $n_k = \left|\left\{x^{(i)} \in \mathcal{D} : x^{(i)} \in R_k\right\}\right|$ be number of data points in region k, ,

DT specific

Greedy strategy, recursive binary splitting

- Given R_1 and R_2, apply same strategy to each of R_1 and R_2.
- When to stop? One option: stop when gain falls below threshold.
- Better: minimize cost/complexity. Grow to large tree T_0. Let $T \subset T_0$ be any subtree (via pruning internal nodes, merging regions).
- Let: k be a terminal (leaf) node, with region R_k. $|T|$ be the number of terminal nodes, $n_k = |\{x^{(i)} \in \mathcal{D} : x^{(i)} \in R_k\}|$ be number of data points in region k, $\hat{c}_k = (1/n_k) \sum_{x^{(i)} \in R_k} y^{(i)}$ be prediction in region k,

Greedy strategy, recursive binary splitting

- Given R_1 and R_2, apply same strategy to each of R_1 and R_2.
- When to stop? One option: stop when gain falls below threshold.
- Better: minimize cost/complexity. Grow to large tree T_0. Let $T \subset T_0$ be any subtree (via pruning internal nodes, merging regions).
- Let: k be a terminal (leaf) node, with region R_k. $|T|$ be the number of terminal nodes, $n_k = |\{x^{(i)} \in \mathcal{D} : x^{(i)} \in R_k\}|$ be number of data points in region k, $\hat{c}_k = (1/n_k) \sum_{x^{(i)} \in R_k} y^{(i)}$ be prediction in region k, $Q_k(T) = (1/n_k) \sum_{x^{(i)} \in R_k} (y^{(i)} - \hat{c}_k)^2$ be error in region k.

Greedy strategy, recursive binary splitting

- Given R_1 and R_2, apply same strategy to each of R_1 and R_2.
- When to stop? One option: stop when gain falls below threshold.
- Better: minimize cost/complexity. Grow to large tree T_0. Let $T \subset T_0$ be any subtree (via pruning internal nodes, merging regions).
- Let: k be a terminal (leaf) node, with region R_k. $|T|$ be the number of terminal nodes, $n_k = |\{x^{(i)} \in \mathcal{D} : x^{(i)} \in R_k\}|$ be number of data points in region k, $\hat{c}_k = (1/n_k) \sum_{x^{(i)} \in R_k} y^{(i)}$ be prediction in region k, $Q_k(T) = (1/n_k) \sum_{x^{(i)} \in R_k} (y^{(i)} - \hat{c}_k)^2$ be error in region k.
- Overall cost complexity defined as

$$C_\alpha(T) = \sum_{k=1}^{|T|} n_k Q_k(T) + \alpha |T| \tag{7.19}$$

tradeoff coefficient.

regularizer

loss

Greedy strategy, recursive binary splitting

- Given R_1 and R_2, apply same strategy to each of R_1 and R_2.
- When to stop? One option: stop when gain falls below threshold.
- Better: minimize cost/complexity. Grow to large tree T_0. Let $T \subset T_0$ be any subtree (via pruning internal nodes, merging regions).
- Let: k be a terminal (leaf) node, with region R_k. $|T|$ be the number of terminal nodes, $n_k = |\{x^{(i)} \in \mathcal{D} : x^{(i)} \in R_k\}|$ be number of data points in region k, $\hat{c}_k = (1/n_k) \sum_{x^{(i)} \in R_k} y^{(i)}$ be prediction in region k, $Q_k(T) = (1/n_k) \sum_{x^{(i)} \in R_k} (y^{(i)} - \hat{c}_k)^2$ be error in region k.
- Overall cost complexity defined as

$$C_\alpha(T) = \sum_{k=1}^{|T|} n_k Q_k(T) + \alpha |T| \tag{7.19}$$

- Goal, for each α find the $T_\alpha \subset T_0$ that minimizes $C_\alpha(T_\alpha)$.

Greedy strategy, recursive binary splitting

- Given R_1 and R_2, apply same strategy to each of R_1 and R_2.
- When to stop? One option: stop when gain falls below threshold.
- Better: minimize cost/complexity. Grow to large tree T_0. Let $T \subset T_0$ be any subtree (via pruning internal nodes, merging regions).
- Let: k be a terminal (leaf) node, with region R_k. $|T|$ be the number of terminal nodes, $n_k = |\{x^{(i)} \in \mathcal{D} : x^{(i)} \in R_k\}|$ be number of data points in region k, $\hat{c}_k = (1/n_k) \sum_{x^{(i)} \in R_k} y^{(i)}$ be prediction in region k, $Q_k(T) = (1/n_k) \sum_{x^{(i)} \in R_k} (y^{(i)} - \hat{c}_k)^2$ be error in region k.
- Overall cost complexity defined as

$$C_\alpha(T) = \sum_{k=1}^{|T|} n_k Q_k(T) + \alpha |T| \tag{7.19}$$

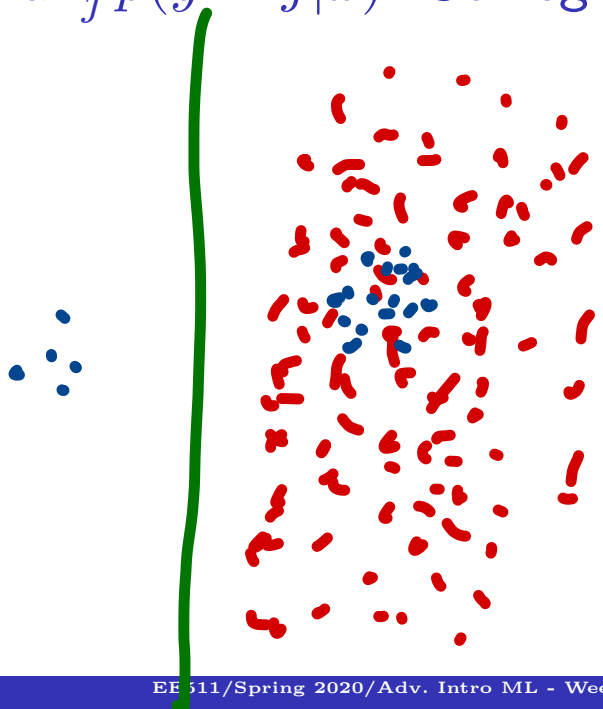
- Goal, for each α find the $T_\alpha \subset T_0$ that minimizes $C_\alpha(T_\alpha)$.
- weakest link pruning (Brieman et al. 1984) solves this efficiently by successively collapsing internal node w. smallest per-node increase in $\sum_{k=1}^{|T|} n_k Q_k(T)$.

Classification Trees

- At each leaf, predict a class rather than a real value. Produce region-specific posterior distribution to make classification decisions for a $x \in \mathbb{R}^m$. With n_k = count of training samples in region k, we have:

$$\hat{p}(y = j|x) = \frac{1}{n_{k(x)}} \sum_{x^{(i)} \in R_{k(x)}} \mathbf{1}_{\{y^{(i)}=j\}} = \hat{p}_{k(x)}(y = j) \tag{7.20}$$

where $k(x)$ is the region containing x. Make decisions using $\hat{y}(x) = \text{argmax}_j \, \hat{p}(y = j|x)$. So region k always decides class $\hat{y}_k$.



Classification Trees

- At each leaf, predict a class rather than a real value. Produce region-specific posterior distribution to make classification decisions for a $x \in \mathbb{R}^m$. With n_k = count of training samples in region k, we have:

$$\hat{p}(y = j|x) = \frac{1}{n_{k(x)}} \sum_{x^{(i)} \in R_{k(x)}} \mathbf{1}_{\{y^{(i)}=j\}} = \hat{p}_{k(x)}(y = j) \tag{7.20}$$

  where $k(x)$ is the region containing x. Make decisions using $\hat{y}(x) = \text{argmax}_j \, \hat{p}(y = j|x)$. So region k always decides class $\hat{y}_k$.
- With this we can use the same greedy strategy to grow the tree from top down.

Classification Trees

- At each leaf, predict a class rather than a real value. Produce region-specific posterior distribution to make classification decisions for a $x \in \mathbb{R}^m$. With n_k = count of training samples in region k, we have:

$$\hat{p}(y = j|x) = \frac{1}{n_{k(x)}} \sum_{x^{(i)} \in R_{k(x)}} \mathbf{1}_{\{y^{(i)}=j\}} = \hat{p}_{k(x)}(y = j) \tag{7.20}$$

  where $k(x)$ is the region containing x. Make decisions using $\hat{y}(x) = \text{argmax}_j \, \hat{p}(y = j|x)$. So region k always decides class $\hat{y}_k$.
- With this we can use the same greedy strategy to grow the tree from top down.
- How to judge each region? Several ways:

Classification Error

$$\frac{1}{n_k} \sum_{i \in R_k} \mathbf{1}_{\{y^{(i)} \neq \hat{y}_k\}} \tag{7.21}$$

Entropy in region k

$$-\sum_{j=1}^{\ell} \hat{p}_k(y = j) \log \hat{p}_k(y = j) \tag{7.22}$$

Entropy

Definition 7.5.1 (Entropy)

Given a discrete random variable X over a finite sized alphabet, the entropy of the random variable is:

$$H(X) \triangleq E \log \frac{1}{p(X)} = \sum_x p(x) \log \frac{1}{p(x)} = -\sum_x p(x) \log p(x) \quad (7.23)$$

- $1/p(x)$ is surprise of x, $-\log p(x)$ is log surprise.

Entropy

Definition 7.5.1 (Entropy)

Given a discrete random variable X over a finite sized alphabet, the entropy of the random variable is:

$$H(X) \triangleq E \log \frac{1}{p(X)} = \sum_x p(x) \log \frac{1}{p(x)} = -\sum_x p(x) \log p(x) \quad (7.23)$$

- $1/p(x)$ is surprise of x, $-\log p(x)$ is log surprise.
- Entropy is typically units of "bits" (logs base 2) but also can be units of "nats" (logs base e). For optimization it doesn't matter.

Entropy

Definition 7.5.1 (Entropy)

Given a discrete random variable X over a finite sized alphabet, the entropy of the random variable is:

$$H(X) \triangleq E \log \frac{1}{p(X)} = \sum_x p(x) \log \frac{1}{p(x)} = -\sum_x p(x) \log p(x) \quad (7.23)$$

- $1/p(x)$ is surprise of x, $-\log p(x)$ is log surprise.
- Entropy is typically units of "bits" (logs base 2) but also can be units of "nats" (logs base e). For optimization it doesn't matter.
- Measures the degree of uncertainty in a distribution.

Entropy

Definition 7.5.1 (Entropy)

Given a discrete random variable X over a finite sized alphabet, the entropy of the random variable is:

$$H(X) \triangleq E \log \frac{1}{p(X)} = \sum_x p(x) \log \frac{1}{p(x)} = -\sum_x p(x) \log p(x) \quad (7.23)$$

- $1/p(x)$ is surprise of x, $-\log p(x)$ is log surprise.
- Entropy is typically units of "bits" (logs base 2) but also can be units of "nats" (logs base e). For optimization it doesn't matter.
- Measures the degree of uncertainty in a distribution.
- Measures the disorder or spread of a distribution.

Entropy

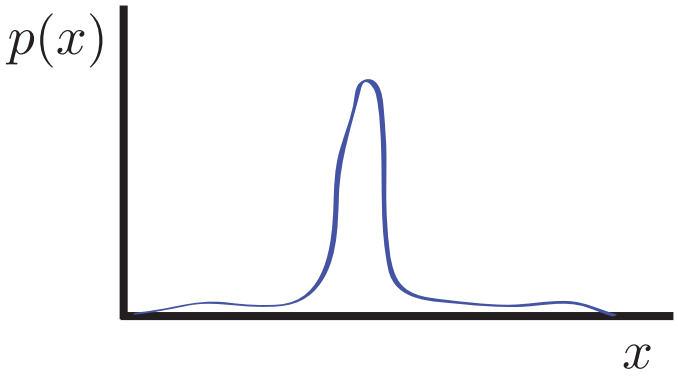
Definition 7.5.1 (Entropy)

Given a discrete random variable X over a finite sized alphabet, the entropy of the random variable is:

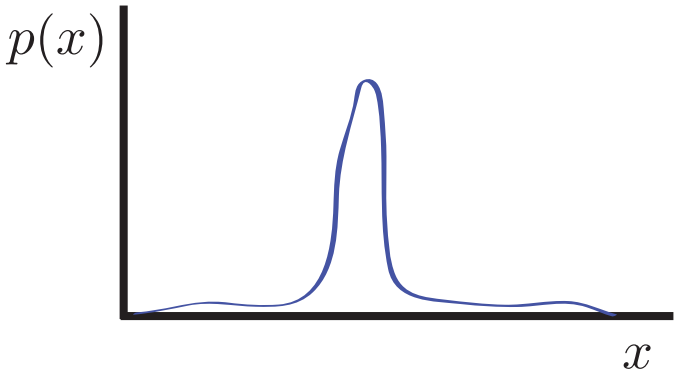
$$H(X) \triangleq E \log \frac{1}{p(X)} = \sum_x p(x) \log \frac{1}{p(x)} = -\sum_x p(x) \log p(x) \quad (7.23)$$

- $1/p(x)$ is surprise of x, $-\log p(x)$ is log surprise.
- Entropy is typically units of "bits" (logs base 2) but also can be units of "nats" (logs base e). For optimization it doesn't matter.
- Measures the degree of uncertainty in a distribution.
- Measures the disorder or spread of a distribution.
- Measures the "choice" that a source has in choosing symbols according to the density (higher entropy means more choice)

Entropy Of Distributions

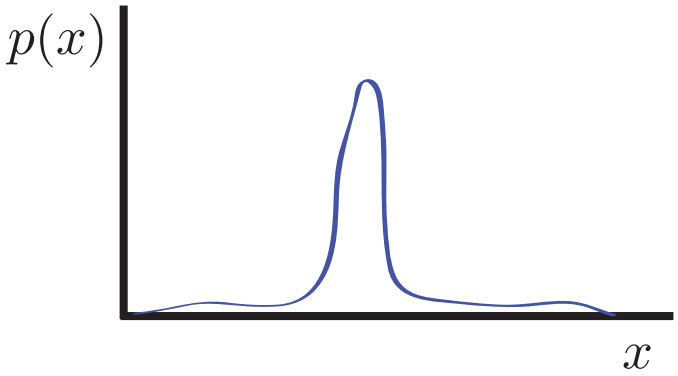


Entropy Of Distributions

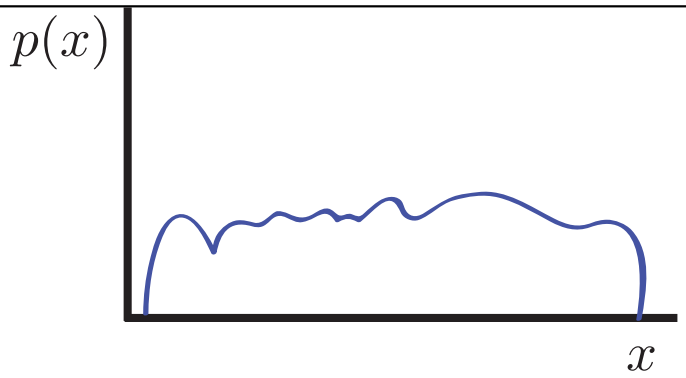


Low Entropy

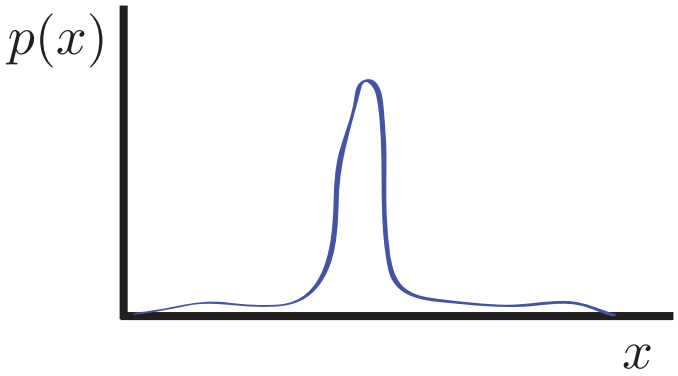
Entropy Of Distributions



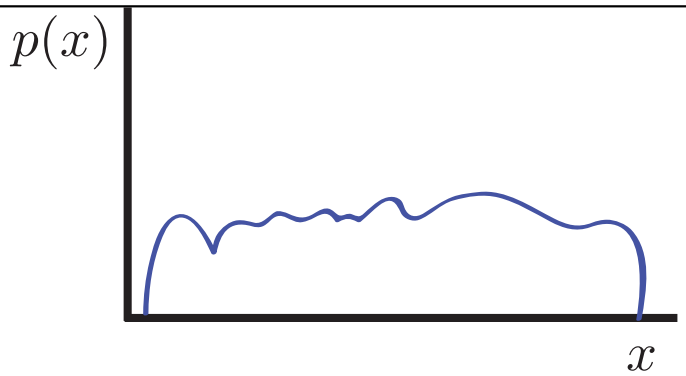
Low Entropy



Entropy Of Distributions

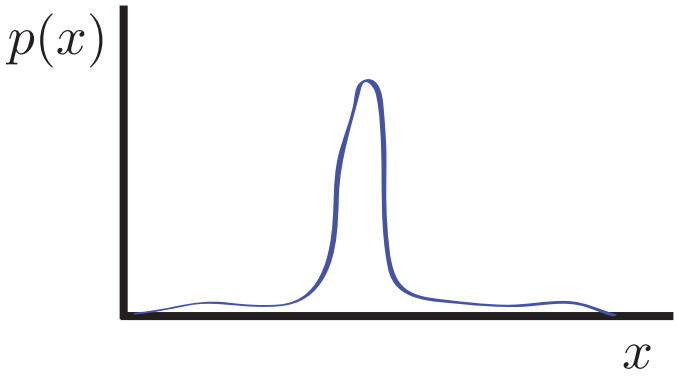


Low Entropy

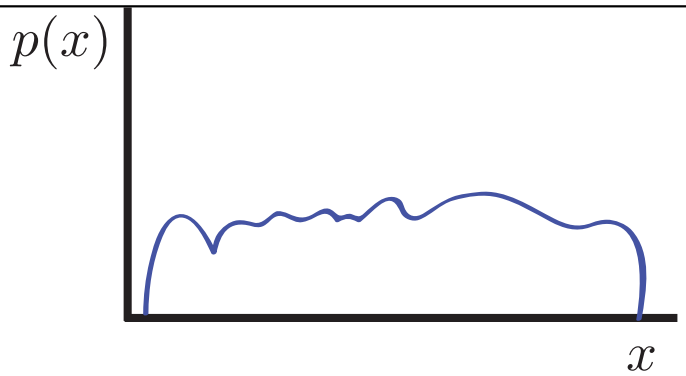


High Entropy

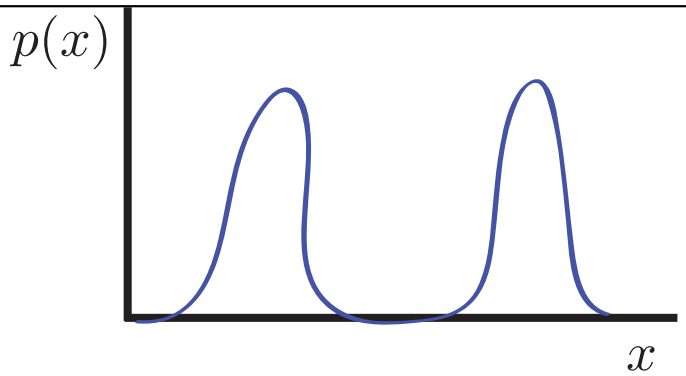
Entropy Of Distributions



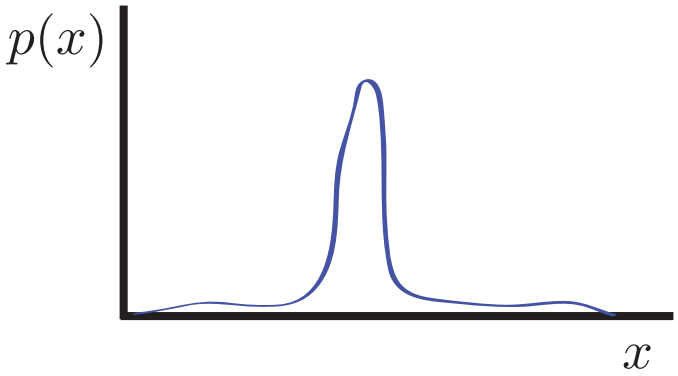
Low Entropy



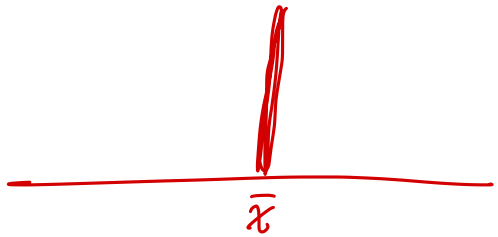
High Entropy

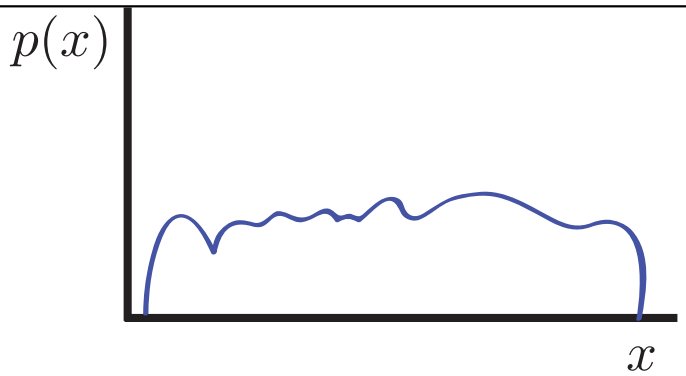


Entropy Of Distributions

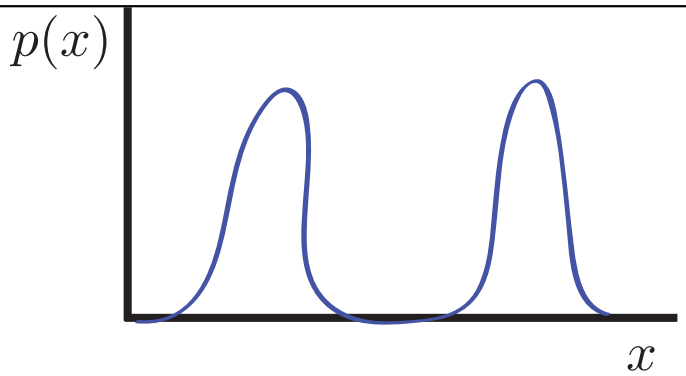


Low Entropy





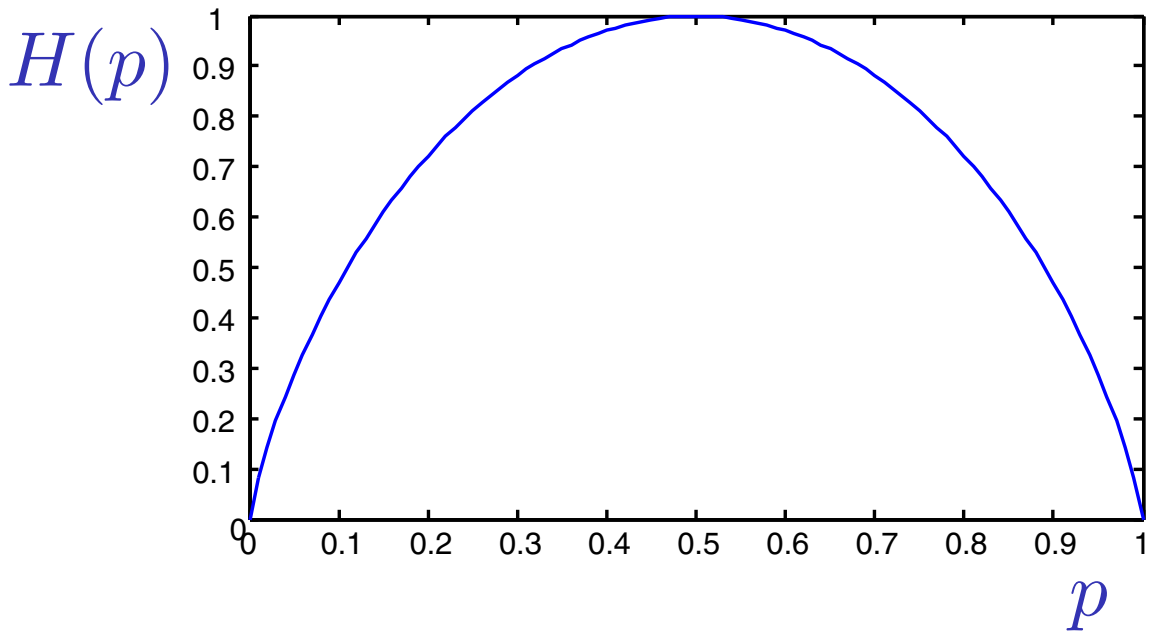
High Entropy



In Between

Binary Entropy

- Binary alphabet, $X \in \{0, 1\}$ say.
- $p(X = 1) = p = 1 - p(X = 0)$.
- $H(X) = -p \log p - (1 - p) \log(1 - p) = H(p)$.
- As a function of p, we get:



- Note, greatest uncertainty (value 1) when $p = 0.5$ and least uncertainty (value 0) when $p = 0$ or $p = 1$.
- Note also: concave in p.

Classification Trees and Entropy

- We can measure entropy in each region.

Classification Trees and Entropy

- We can measure entropy in each region.
- Again, splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes

$$R_1(j, s) = \{x : x_j \leq s\} \qquad R_2(j, s) = \{x : x_j > s\} \tag{7.24}$$

Classification Trees and Entropy

- We can measure entropy in each region.
- Again, splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes

$$R_1(j, s) = \{x : x_j \leq s\} \qquad R_2(j, s) = \{x : x_j > s\} \tag{7.24}$$

- A region's posterior is essentially a histogram of class labels in region.

Classification Trees and Entropy

- We can measure entropy in each region.
- Again, splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes

$$R_1(j, s) = \{x : x_j \leq s\} \qquad R_2(j, s) = \{x : x_j > s\} \tag{7.24}$$

- A region's posterior is essentially a histogram of class labels in region.
- Starting from one large region, we find the variable j and split s that, when made, reduces the entropy the most.

Classification Trees and Entropy

- We can measure entropy in each region.
- Again, splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes

$$R_1(j,s) = \{x : x_j \leq s\} \qquad R_2(j,s) = \{x : x_j > s\} \tag{7.24}$$

- A region's posterior is essentially a histogram of class labels in region.
- Starting from one large region, we find the variable j and split s that, when made, reduces the entropy the most.
- Maximizing information gain, split region R_0 into $R_1(j,s)$ and $R_2(j,s)$

$$\max_{j,s} \left[H(p_0(y)) - \left(H(p_{R_1(j,s)}(y)) + H(p_{R_2(j,s)}(y)) \right) \right] \tag{7.25}$$

Classification Trees and Entropy

- We can measure entropy in each region.
- Again, splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes

$$R_1(j,s) = \{x : x_j \leq s\} \qquad R_2(j,s) = \{x : x_j > s\} \tag{7.24}$$

- A region's posterior is essentially a histogram of class labels in region.
- Starting from one large region, we find the variable j and split s that, when made, reduces the entropy the most.
- Maximizing information gain, split region R_0 into $R_1(j,s)$ and $R_2(j,s)$

$$\max_{j,s} \left[H(p_0(y)) - \left(H(p_{R_1(j,s)}(y)) + H(p_{R_2(j,s)}(y)) \right) \right] \tag{7.25}$$

- Leads to regions with lower entropy, higher certainty, least diversity in each region, creating homogeneous regions.

Classification Trees and Entropy

- We can measure entropy in each region.
- Again, splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes

$$R_1(j,s) = \{x : x_j \leq s\} \qquad R_2(j,s) = \{x : x_j > s\} \tag{7.24}$$

- A region's posterior is essentially a histogram of class labels in region.
- Starting from one large region, we find the variable j and split s that, when made, reduces the entropy the most.
- Maximizing information gain, split region R_0 into $R_1(j,s)$ and $R_2(j,s)$

$$\max_{j,s} \left[H(p_0(y)) - \left(H(p_{R_1(j,s)}(y)) + H(p_{R_2(j,s)}(y)) \right) \right] \tag{7.25}$$

- Leads to regions with lower entropy, higher certainty, least diversity in each region, creating homogeneous regions.
- Term: CART, for classification and regression trees.

Trees, Expressivity, Bias, and Variance

- Decision trees are extremely flexible.

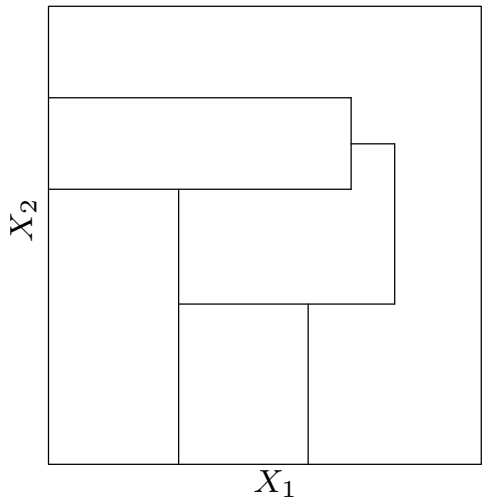
Trees, Expressivity, Bias, and Variance

- Decision trees are extremely flexible.
- Like k-NN methods, DT methods in general have high variance and low bias (they can flexibly fit any data sets). With a tall tree, a DT can perfectly fit any training data with non-conflicting labels (i.e., different y for the same x). Like nearest neighbor, but neighborhoods are rectangular (for top-down greedy tree procedure).

$$\min_T \quad \text{Loss} + \alpha \cdot |T|$$

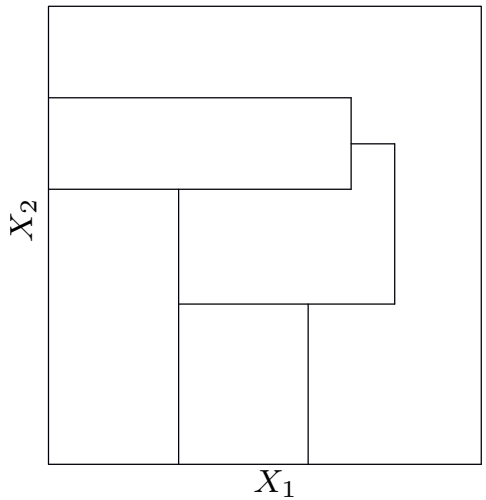
Trees, Expressivity, Bias, and Variance

- Decision trees are extremely flexible.
- Like k-NN methods, DT methods in general have high variance and low bias (they can flexibly fit any data sets). With a tall tree, a DT can perfectly fit any training data with non-conflicting labels (i.e., different y for the same x). Like nearest neighbor, but neighborhoods are rectangular (for top-down greedy tree procedure).
- Top-down Binary split tree can't achieve all regions, ex:



Trees, Expressivity, Bias, and Variance

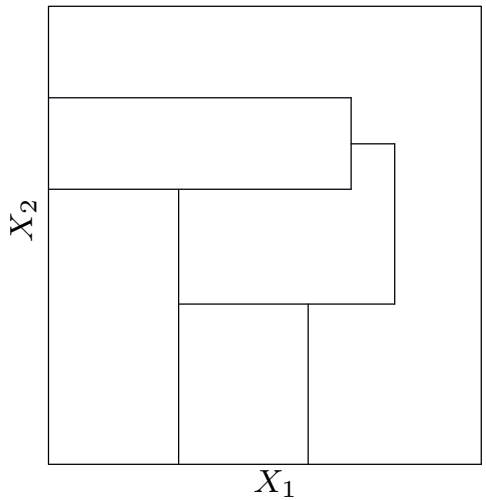
- Decision trees are extremely flexible.
- Like k-NN methods, DT methods in general have high variance and low bias (they can flexibly fit any data sets). With a tall tree, a DT can perfectly fit any training data with non-conflicting labels (i.e., different y for the same x). Like nearest neighbor, but neighborhoods are rectangular (for top-down greedy tree procedure).
- Top-down Binary split tree can't achieve all regions, ex:



- DTs can be constructed using other procedures that can do this.

Trees, Expressivity, Bias, and Variance

- Decision trees are extremely flexible.
- Like k-NN methods, DT methods in general have high variance and low bias (they can flexibly fit any data sets). With a tall tree, a DT can perfectly fit any training data with non-conflicting labels (i.e., different y for the same x). Like nearest neighbor, but neighborhoods are rectangular (for top-down greedy tree procedure).
- Top-down Binary split tree can't achieve all regions, ex:



- DTs can be constructed using other procedures that can do this.
- Another advantage of trees, they lead to interpretable machine learning models since all decisions are based on original inputs, rather than based on mysterious learnt non-convex combinations thereof.