DSC 140A Lecture

lecture 1 :

Feature : numerical

Feature vector

Dimensionality : # of features in a feature vector

Training set

$X = \{(\vec{x}^{(1)}, y_1) \cdots (\vec{x}^{(n)}, y_n)\}$

Train error

Test error ⇒ generalization

Overfitting

Underfitting

Nearest Neighbors Predictor

useful for both classification & regression

Predict

Find the closest point to $\vec{z}$ in X :

$$i^* = \underset{i \in \{1, \cdots, n\}}{\arg\min} \| \vec{x}^{(i)} - \vec{z} \|$$

Predicted label : y_i

- Euclidean Distance - 2-norm

$$\|\vec{p} - \vec{q}\| = \sqrt{\sum_{k=1}^{d} (p_k - q_k)^2} = \sqrt{(\vec{p} - \vec{q}) \cdot (\vec{p} - \vec{q})}$$

- Using Euclidean Distance treats all features equally

⇓ fix

Standardlizing features (z-score).

$$(x_1, x_2)^T \Rightarrow (z_1, z_2)^T = \left(\frac{x_1 - \mu_1}{\sigma_1}, \frac{x_2 - \mu_2}{\sigma_2} \right)$$

- Decision Boundary

- KNN
  - choose k nearest neighbors and vote

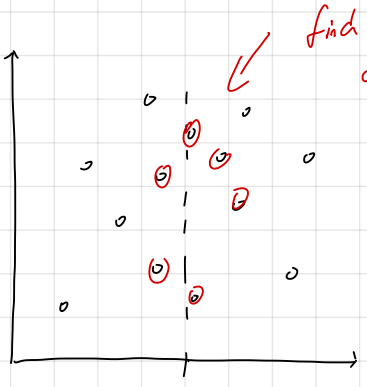
Predict
- find the most common labels in the k nearest neighbors points.

- k controls complexity

k ↑ -> underfitting -> simply boundary

k ↓ -> overfitting -> complex boundary

NN Regression



KNN => average the neighbor

Empirical Risk Minimization

Prediction Function

$H(\vec{x}) \rightarrow y$

$\vec{x}$: takes in a feature vector

y: predicted label.

ex) $H(\text{experience}, \text{GPA}) = 50000 + 10000 \times \text{exp.} + 5000 \times \text{GPA}$

Assumption: the future will be like the past

↑ unseen

idea: use past data to "measure" how good a prediction function is.

Fit: formally quantify how well a prediction function fits the data?

⇓

by measuring error.

- loss function → measure difference between $H(\vec{x}^{(i)})$ & y_i
  - Absolute loss: $l_{abs}(H(\vec{x}^{(i)}), y_i) = |H(\vec{x}^{(i)}) - y_i|$
  - Square loss: $l_{sq}(H(\vec{x}^{(i)}), y_i) = (H(\vec{x}^{(i)}) - y_i)^2$

- Average loss => Empirical Risk

$$R(H) = \frac{1}{n} \sum_{i=1}^{n} l(H(\vec{x}^{(i)}), y_i)$$

- depends on training set & choice of loss function.

⇓

Minimizing Empirical Risk.

Restricting our search for prediction functions to a smaller set of functions.

↓

hypothesis class.

So, ERM

ex) linear function

1. hypothesis class
2. loss function
3. find H minimizing risk

Linear Function

$$H(x) = w_0 + w_1 x$$

Generally, $H(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$.

$w_0 \cdots w_d$ are the parameters or weights.

- w_0 → bias, determine the prediction when all other are 0.
- w_d → how much prediction change

- parameter vector : $\vec{x} \in R^d \Rightarrow \vec{w} \in R^{d+1}$

$$\vec{w} = (w_0, w_1, \cdots w_d)^T$$

So, $H(\vec{x}, \vec{w}) = (w_0, \cdots w_d)^T (1, x_1, \cdots, x_d)^T$

$$Aug(\vec{x}) = (1, x_1, \cdots, x_d)^T = (1, \vec{x})^T$$

So, $H(\vec{x}, \vec{w}) = w_0 + w_1 x_1 + \cdots + w_d x_d$

$= \vec{w} \cdot \text{Aug}(\vec{x})$.

Absolute loss: $R_{abs}(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} |H(\vec{x}^{(i)}) - y_i|$

$= \frac{1}{n} \sum_{i=1}^{n} |\vec{w} \cdot \text{Aug}(\vec{x}^{(i)}) - y_i|$

Risk Surface: imagine plotting $R_{abs}(\vec{w})$ for all values of $\vec{w}$

$\vec{w}$ that makes the surface lowest minimizes the risk

↓ How?

- Calculus -> gradient $\frac{d}{d\vec{w}} R_{abs}(\vec{w})$

  set it equal to 0

  solve for $\vec{w}$

- However, R_{abs} is not differentiable

↓ try square loss

$R_{sq}(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} (H(\vec{x}^{(i)}) - y_i)^2$

$= \frac{1}{n} \sum_{i=1}^{n} (\vec{w}^{(i)} \cdot \text{Aug}(\vec{x}^{(i)}) - y_i)^2$

(Mean Squared Error)

(MSE).

Least Square:

$R_{sq} = \frac{1}{n} \sum_{i=1}^{n} (\vec{w}^{(i)} \cdot \text{Aug}(\vec{x}^{(i)} - y_i)^2$

$= \frac{1}{n} \| X\vec{w} - \vec{y} \|^2$ ← let $p_i = \text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}$

$\vec{p} = (p_1, \ldots p_n)^T$

$\vec{p} \in R^n$

design matrix

Design Matrix

$$X = \begin{pmatrix} \text{Aug}(\vec{x}^{(1)}) \\ \vdots \\ \text{Aug}(\vec{x}^{(n)}) \end{pmatrix}$$

$$= \begin{pmatrix} 1 & x_1^{(1)} & \cdots & x_d^{(1)} \\ 1 & & & \\ \vdots & & \ddots & \\ 1 & x_1^{(n)} & & x_d^{(n)} \end{pmatrix} \in \mathbb{R}^{n \times d+1}$$

$$K_{sq} = \frac{1}{n} \sum_{i=1}^{n} (p_i - y_i)^2$$

$$= \frac{1}{n} \| \vec{p} - \vec{y} \|^2$$

So $\vec{p} = X \vec{w}$

$$\Downarrow$$

$$K_{sq} = \frac{1}{n} \| X \vec{w} - \vec{y} \|^2$$

Find gradient, $\frac{d}{d\vec{w}} \left[\frac{1}{n} \| X \vec{w} - \vec{y} \|^2 \right]$

$$\Downarrow$$

$$\frac{dK_{sq}}{d\vec{w}} = \left(\frac{\partial K_{sq}}{\partial w_0}, \dots \right)$$ partial derivative

$$K_{sq} = (X \vec{w} - \vec{y}) \cdot (X \vec{w} - \vec{y})$$

$$= (X \vec{w} - \vec{y})^T (X \vec{w} - \vec{y})$$

$$= (\vec{w}^T X^T - \vec{y}^T)(X \vec{w} - \vec{y})$$

$$= \vec{w}^T X^T X \vec{w} - \vec{w}^T X^T \vec{y} - \vec{y}^T X \vec{w} + \vec{y}^T \vec{y}$$

$$\vec{w}^T X^T \vec{y} = (\vec{y}^T X \vec{w})^T \leftarrow \text{scalar}$$

$$= \vec{y}^T X \vec{w}$$

$$= \vec{w}^T X^T X \vec{w} - 2 \vec{y}^T X \vec{w} + \vec{y}^T \vec{y}$$

Find gradient: $\frac{d}{d\vec{w}} \left[\vec{w}^T X^T X \vec{w} - 2 \vec{y}^T X \vec{w} + \vec{y}^T \vec{y} \right]$

$$\frac{d}{d\vec{w}}\left((X\vec{w})^T(X\vec{w})\right) \sim \frac{d}{d\vec{w}}(x^2w^2) = 2x^2w$$

$$= 2X^TX\vec{w}$$

$$\frac{d}{d\vec{w}}\left[y^TX\vec{w}\right] = X^T\vec{y}$$

$$\frac{d}{d\vec{w}}\left[y^Ty\right] = 0$$

So,

$$= \frac{1}{n}\left(2X^TX\vec{w} - 2X^T\vec{y} + 0\right)$$

set $\frac{d\,RSS}{d\vec{w}} = 0$

$$\frac{1}{n}\left(2X^TX\vec{w} - 2X^T\vec{y}\right) = 0$$

$$X^TX\vec{w} = X^T\vec{y} \quad \text{(normal equation)}$$

$$\Downarrow$$

$$\vec{w}^* = (X^TX)^{-1}X^T\vec{y} \quad \text{(least square solution)}$$

direct solution.

Gradient Descent

derivative → as slope of curve

↳ Positive : increasing

Negative : decreasing

Zero : flat

- Partial derivative.

  for f as function of $\vec{z} = (z_1, z_2)^T$

  - $\frac{\partial f}{\partial z_1}$: slope in z_1 direction
  - $\frac{\partial f}{\partial z_2}$: slope in z_2 direction.

- Gradient

$$\frac{df}{d\vec{z}}(\vec{z}) = \begin{pmatrix} \frac{\partial f}{\partial z_1}(\vec{z}) \\ \vdots \\ \frac{\partial f}{\partial z_2}(\vec{z}) \end{pmatrix}$$

for $f: \mathbb{R}^d \to \mathbb{R}$

(the ith component is slope of f at $\vec{z}$ in ith direction).

- input vector $\in \mathbb{R}^d$
- output vector $\in \mathbb{R}^d$

- gradient approximation:

slope for $\vec{z}$

$$f(\vec{z} + \vec{\delta}) \simeq f(\vec{z}) + \vec{\delta} \cdot \frac{df}{d\vec{z}}(\vec{z}).$$

change in $\vec{z}$

- gradient in the direction of steepest ascent.

⇓

negative

(steepest descent)

⇓

why?

change $\vec{\delta} \cdot \frac{df}{d\vec{z}}(\vec{z}) = \|\vec{\delta}\| \left\| \frac{df}{d\vec{z}} \right\| \cos\theta$

⇓

maximized when $\theta = 0$

⇓

direction of gradient

- Contour map

⇓

gradient orthogonal to the contours

- Gradient descent (optimization).

⇓

Find the minimum

- idea: iterate towards a minimum step by step

start an arbitary location

move in direction of steepest descent. (negative gradient).

pick arbitary point $\vec{z}^{(0)}$, learning rate $\eta > 0$

- compute gradient: $\frac{df}{d\vec{z}}(\vec{z}^{(t)})$ at $\vec{z}^{(t)}$
- update rule

$$\vec{z}^{(t+1)} = \vec{z}^{(t)} - \eta \times \frac{df}{d\vec{z}}(\vec{z}^{(t)})$$ for η as learning rate

- when converged, return $\vec{z}^{(t)}$

⇓

local minimum.

stop when $\left\| \frac{df}{d\vec{z}}(\vec{z}) \right\|$ is small

or $\| \vec{z}^{(t+1)} - \vec{z}^{(t)} \|$ is small.

- Gradient descent with ERM:

$d+1 \times 1$

$n \times d+1$ $\quad d+1 \times 1$

$$\frac{dR}{d\vec{w}}(\vec{w}) = \frac{d}{d\vec{w}}\left(\frac{1}{n}\sum_{i=1}^{n} \ell(Aug(\vec{x}^{(i)}) \cdot \vec{w}, y_i)\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n} \frac{d\ell}{d\vec{w}}(Aug(\vec{x}^{(i)}) \cdot \vec{w}, y_i)$$

for squared loss:

$$\frac{dR}{d\vec{w}}(\vec{w}) = \frac{2}{n}\sum_{i=1}^{n} (Aug(\vec{x}^{(i)}) \cdot \vec{w} - y_i)\, Aug(\vec{x}^{(i)})$$

So update rule:

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta \times \frac{2}{n}\sum_{i=1}^{n} (Aug(\vec{x}^{(i)}) \cdot \vec{w}^{(t)} - y_i)\, Aug(\vec{x}^{(i)})$$

Benefit of gradient descent vs. nominal equation

computation of gradient can be distributed, don't have to computed at once;

memory efficient.

SGD

idea Use a random smaller subset of data B, to compute gradient.

$$\frac{d}{d\vec{w}} K(\vec{w}) \approx \frac{1}{|B|} \sum_{i \in B} \frac{d}{d\vec{w}} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i).$$

- the smaller set B is called mini-batch
- it is called stochastic gradient.

↓

an approximation of the full gradient

bootstrapping ↓

- process same as gradient descent

except gradient is computed randomly (mini-batch)

Lease Square gradient

$$\vec{g} = \frac{2}{m} \sum_{i \in B} (Aug(\vec{x}^{(i)}) \cdot \vec{w} - y_i) Aug(\vec{x}^{(i)})$$

$$= \frac{2}{m} X_B^T (X_B \vec{w} - y_B)$$

Trade off

- each step of GD, move in the "best" direction

but slowly

- each step of SGD, move in a "good" direction,

but faster.

usaful for massive dataset

Absolute loss

not differentiable at some point ✓ => subgradient descent

(piecewise function)

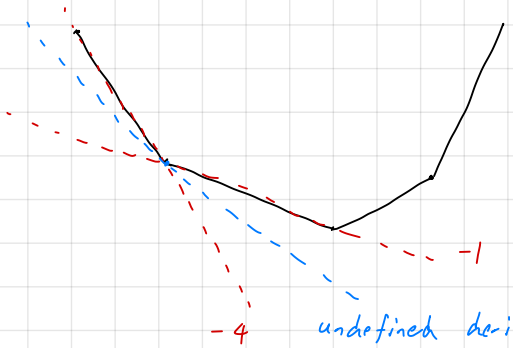
↓↓

problem: reach a point where the gradient is not defined

⇓

a replacement

idea:



undefined derivative

⇓

replace with subderivative

(any number between -4 and -1)

Any valid subderivative func. defines a line lies below the function.

⇓

$f_s(z) = f(z_0) + s(z - z_0)$

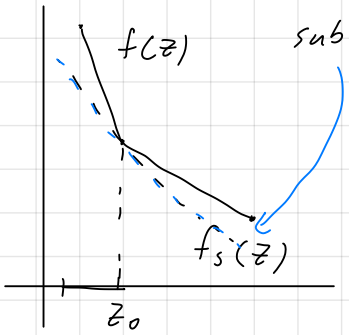
s is a subderivative if

$f(z) \geq f_s(z)$ for all z,

$f(z) \geq f(z_0) + s(z - z_0)$.

⇓

sub derivative line



⇓

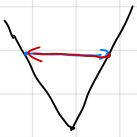
Subgradient:

in all dimension

valid subgradient if lies below or at the function

$$f(\vec{z}) \geq f(\vec{z}^{(0)}) + \vec{s} \cdot (\vec{z} - \vec{z}^{(0)})$$

subgradient descent



b/c same derivative

problem: might not converge

⇓

decrease learning rate in each iteration

⇓

choose a decreasing learning rate schedule $\eta(t) > 0$

choose $\eta(t) = c/\sqrt{t}$, t is iteration, c is constant

Absolute loss subgradient descent

Same as GD except use subgradient inseead of gradient.

⇓

sub gradient empirical risk

$$\text{subgrad } R(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n} \text{subgrad } \ell(\vec{w}; \vec{x}^{(i)}, y_i)$$

So, $\ell_{abs}(\vec{w}\cdot Aug(\vec{x}^{(i)}), y_i) = \begin{cases} \vec{w}\cdot Aug(\vec{x}^{(i)}) - y_i & \text{if } \vec{w}\cdot Aug(\vec{x}^{(i)}) > y_i \\ y_i - Aug(\vec{x}^{(i)}) & \text{if } \vec{w}\cdot Aug(\vec{x}^{(i)}) < y_i \\ 0 & \text{if } \vec{w}\cdot Aug(\vec{x}^{(i)}) = y_i \end{cases}$

⇓ subgradient

$$s(\vec{w}, \vec{x}^{(i)}, y_i) = \begin{cases} Aug(\vec{x}^{(i)}) & \text{if } \vec{w}\cdot Aug(\vec{x}^{(i)}) > y_i \\ -Aug(\vec{x}^{(i)}) & \text{if } \vec{w}\cdot Aug(\vec{x}^{(i)}) < y_i \\ \vec{0} & \text{if } \vec{w}\cdot Aug(\vec{x}^{(i)}) = y_i \end{cases}$$

zero vector as subgradient.

⇓ empirical risk

$$\frac{1}{n}\sum_{i=1}^{n} s(\vec{w}, \vec{x}^{(i)}, y_i)$$

⇓

use this for GD.

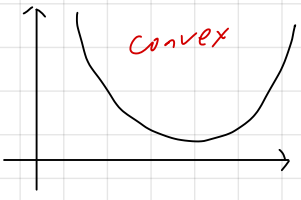
$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta(t) \times \frac{1}{n}\sum_{i=1}^{n} s(\cdot,\cdot)$$

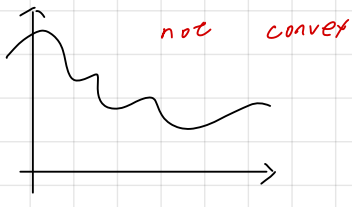
$\eta(t)$: decreasing learning rate. not fixed.

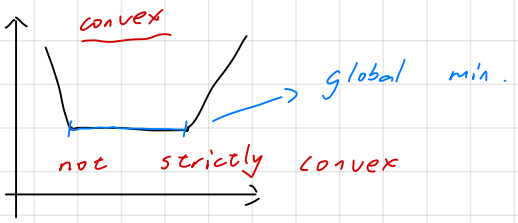
⇓ result

median regression, robuse to outliers.

Convexity





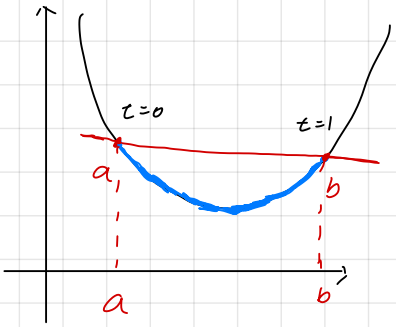


① Def: Function $f: \mathbb{R} \to \mathbb{R}$ is convex if for every choice of $a, b \in \mathbb{R}$ and $t \in [0,1]$:

$$(1-t) f(a) + t f(b) \geq f((1-t)a + tb).$$

↓ the line connected from a to b

↓ the function curve



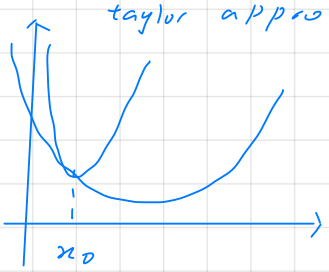
② if $\frac{d^2 f}{dx^2}(x) \geq 0$ for all x, then f is convex.

↓ concave up

only if f is twice differentiable.

why?

taylor approximation.



"best" parabola at x_0

$$f_2(x) = f(x_0) + f'(x_0) \cdot (x - x_0) + \frac{1}{2} f''(x_0) \cdot (x - x_0)^2$$ ← Taylor expansion

possibilities: upward, downward, flat.

Convex if for every x_0, parabola is upward-facing

⇓

$f''(x_0) \geq 0$.

Convex Property

Suppose $f(x)$ & $g(x)$ are convex.

① $w_1 f(x) + w_2 g(x)$ is convex, provided $w_1, w_2 \geq 0$.

② $g(f(x))$ is convex, provided g is non-decreasing.

③ $\max\{f(x), g(x)\}$ is convex.

Theorem $f(x)$ convex, GD converges to global optimum of f.

given that step size is small enough.

Convexity for more dimension

Def: $f\ R^d \to R$ is convex if for every choice of $\vec{a}, \vec{b} \in R^d$, $t \in [0,1]$.

$$(1-t) f(\vec{a}) + t f(\vec{b}) \geq f((1-t)\vec{a} + t\vec{b}).$$

Second Derivative in d-dimension:

there are d^2 # of second derivatives.

⇓ Hessian Matrix.

For $f\ R^d \to R$,

$$H(\vec{x}) = \begin{pmatrix} \frac{\partial f^2}{\partial x_1^2}(\vec{x}) & \cdots & \frac{\partial f^2}{\partial x_1 x_d}(\vec{x}) \\ \frac{\partial f}{\partial x_2 x_1} & \cdots & \frac{\partial f^2}{\partial x_2 x_d}(\vec{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_d x_1} & \cdots & \frac{\partial f}{\partial x_L^2}(\vec{x}) \end{pmatrix} \in R^{d \times d}$$

⇓

Def: convex if all eignvalues of the Hessian matrix $H(\vec{x})$ are ≥ 0.

simple case: matrix is diagonal ⇒ diagonal entries are the eignvalues.

Property:

① $w_1 f(\vec{x}) + w_2 g(\vec{x})$ is convex, for $w_1, w_2 \geq 0$.

② Let A be matrix, $\vec{x}, \vec{b}$ vector,

$g(\vec{x}) = f(A\vec{x} + \vec{b})$ is convex as a function of $\vec{x}$,

for $f(x)$ convex

ex) $f(\vec{w}) = (\vec{x} \cdot \vec{w} - \vec{y})^2$ convex

since $f(x) = x^2$ convex.

Linear Classifier

Problem: output of $H(\vec{x})$ is number

$$H(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2.$$

⇓

Encoding

$y \in \{0, 1\}$: $y = 0$ / $y = 1$

$y \in \{-1, 1\}$: $y = 1$ / $y = -1$

$$\text{sign}(z) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \end{cases}.$$

⇓

$\text{sign}(H(\vec{x}, \vec{w}))$

linear classifier / linear decision function.

idea: a weighted vote

each term $w_i x_i$ votes on the label.

- $H(\vec{x})$ is a (hyper) plane.
- $H(\vec{x}) = 0$ is the decision boundary

if $\vec{x} \in R^2$, a stright line

if $\vec{x} \in R^d$, a $d-1$ dimensional (hyper) plane.

- Magnitude of $H(\vec{x})$ is proportional to the distance from the decision boundary.
  - let $\vec{w}' = (w_1, \ldots w_d)$.

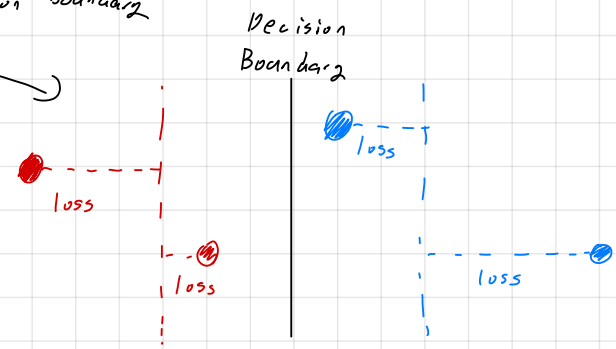
The decision boundary of $H(\vec{x}) = \text{Aug}(\vec{x}) \cdot \vec{w}$ is orthogonal to $\vec{w}'$.

Lease Square Classifier

train: normal equation $\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$.

predict: $\text{sign}(\text{Aug}(\vec{x}) \cdot \vec{w}^*)$.

problem: penalizes points that are far from decision boundary even if they are correctly classified.



Outlier!

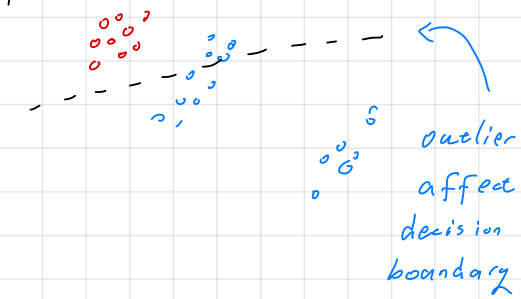
0-1 Loss

$$\ell_{0\text{-}1}(H(\vec{x}^{(i)}), y_i) = \begin{cases} 0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i \end{cases}$$

$$R_{0\text{-}1}(H) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 0 & \text{if } \ldots = y_i \\ 1 & \text{if } \ldots \neq y_i \end{cases}$$

empirical risk

problem:

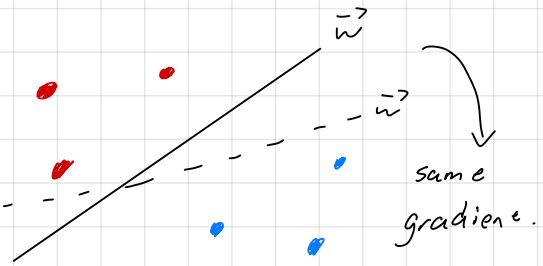


⇓

misclassification rate = $\frac{\#\text{ of correct prediction}}{n}$

minimize risk ⇒ maximize accuracy

problem:



⇓

No GD!

problem: gradient is $\vec{0}$ almost everywhere.

NP-Hard to optimize expected 0-1 loss.

Surrogate loss

$$\ell_{\text{tron}}(H(\vec{x}), y) = \begin{cases} 0 & \text{sign}(H(\vec{x})) = y \\ |H(\vec{x})| & \text{sign}(H(\vec{x})) \neq y \end{cases}$$

proportional to d from decision boundary.

perceperon loss

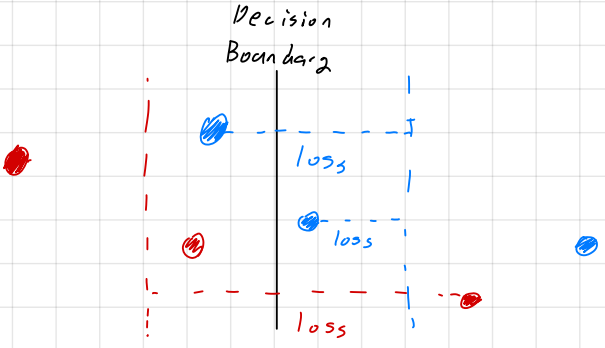
⇓

penalty grows with distance from decision boundary as point misclassified.

$\ell_{tron}(Aug(\vec{x}) \cdot \vec{w}, y) = \max(0, -y\,Aug(\vec{x}) \cdot \vec{w})$

⇓

Convex.



minimize empirical risk.

⇓

subgradient descent.

⇓

if $-y\,Aug(\vec{x}) \cdot \vec{w} > 0$, $grad = -y\,Aug(\vec{x})$

$-y\,Aug(\vec{x}) \cdot \vec{w} < 0$, $grad = \vec{0}$

$-y\,Aug(\vec{x}) \cdot \vec{w} = 0$, $subgrad = \vec{0}$.

subgrad $\ell_{tron}(Aug(\vec{x}) \cdot \vec{w}, y)$.

$$= \begin{cases} \vec{0} & \text{if } sign(Aug(\vec{x}) \cdot \vec{w}) = y \\ -y\,Aug(\vec{x}) & \text{if } sign(Aug(\vec{x}) \cdot \vec{w}) \neq y. \end{cases}$$

train:

$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta^{(t)} \times \frac{1}{n} \sum_{i=1}^{n} \begin{cases} \vec{0} & sign(Aug(\vec{x}^{(i)}) \cdot \vec{w}) = y_i \\ -y_i\,Aug(\vec{x}^{(i)}) & sign(Aug(\vec{x}^{(i)}) \cdot \vec{w}) \neq y_i \end{cases}$$

predict: $sign(Aug(\vec{x}) \cdot \vec{w}^*)$.

Maximum Margin Classifiers.

Problem with perceptron: no penalty to points correctly classified

⇓

overfitting

Def. linearly separable — if there exisc a linear classifier which perfectly classifies the data.

Def: Margin — smallest distance between decision boundary and a training point.

//

Find classifier that maximize margin

⇓

better generalization.

Constraint to ensure points sufficiently far from the boundary.

$|H(\vec{x}^{(i)})| \geq 1$.

⇓

ensure that no point is within the "exclusion zone".

Problem: this only makes the prediction plane very steep.

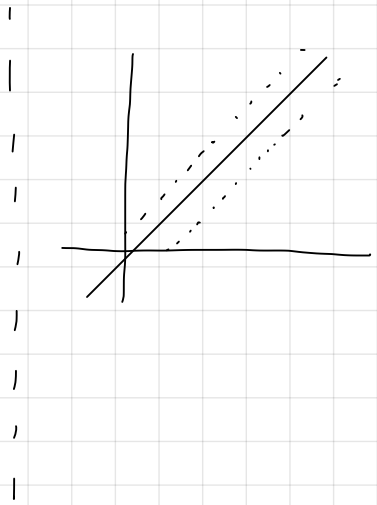
⇓

by making $w_0 \dots w_n$ very large.

⇓

smaller margin

Not what we want.



⇓

idea: $\|\vec{w}\|$ as small as possible.

Out of all $\vec{w}$ satisfying $y_i \vec{w} \cdot Aug(\vec{x}^{(i)}) \geq 1$,

for all data points.

find that minimize $\|\vec{w}\|$.

$\vec{w}^* = \underset{\vec{w}}{\operatorname{argmin}} \|\vec{w}\|$. ⇐ How?

convex quadratic optimization.

$\forall i,\ y_i \vec{w} \cdot Aug(\vec{x}) \geq 1$.

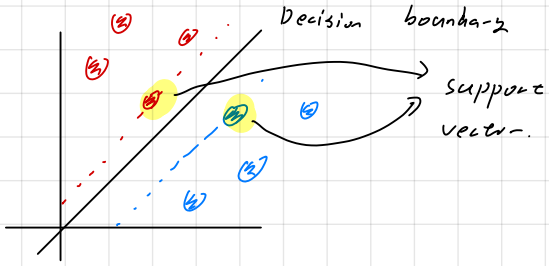
(Hard Support Vector Machine).

⇓

Maximize margin.

- Support Vector — a training point $\vec{x}^{(i)}$, such that

$y_i \vec{w} \cdot Aug(\vec{x}^{(i)}) = 1$.



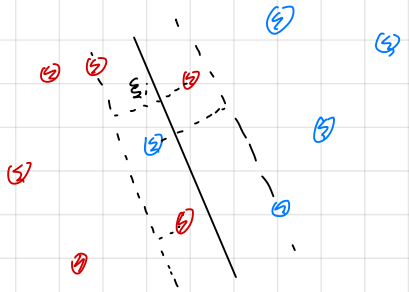
(The solution to Hard-SVM is always a linear combination of the support vector).

Let S be the set of support vectors.

$$\vec{w}^* = \sum_{i \in S} y_i a_i \, Aug(\vec{x}^{(i)}).$$

Problem: what if non-separability?

idea: allow some classifications to be ξ_i wrong but not too wrong.



misclassification

For $C \geq 0$.

$$\min_{\vec{w} \in R^{d+1}, \vec{\xi} \in R} \|\vec{w}\|^2 + C \sum_{i=1}^{n} \xi_i$$

Large C: dont give much slack, avoid

Small C: allow more slack at the cost of misclassification.

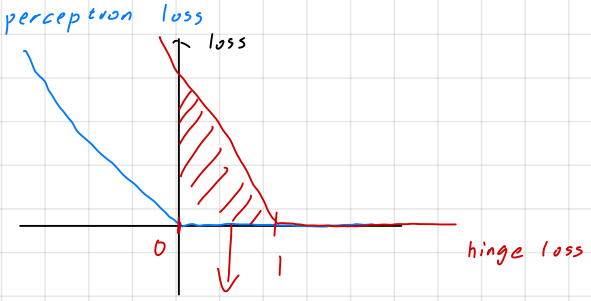
subject to $y_i \vec{w} \cdot Aug(\vec{x}^{(i)}) \geq 1 - \xi_i$ for all i, $\vec{\xi} \geq 0$.

(Soft margin SVM), As $C \to \infty$, margin hardens.

Hinge loss

$$\ell_{hinge} = \begin{cases} 0 & y H(\vec{x}) \geq 1 \\ 1 - y H(\vec{x}) & y H(\vec{x}) < 1. \end{cases}$$

$$= \max\{0, 1 - y H(\vec{x})\}.$$



penalize points too close to the boundary.

Thus, Soft-SVM $\Leftrightarrow$ finding $\vec{w}$ that minimizes

$$R_{svm}(\vec{w}) = \|\vec{w}\|^2 + C \sum_{i=1}^{n} \max\{0, 1 - y_i \vec{w} \cdot \vec{x}^{(i)}\}.$$

SGD to train

a regularized risk

Feature Maps

Non-linear feature maps

⇓

allow us to

- fit complex, non-linear patterns
- while still using linear models.

careful about overfitting.

Feature Map: $\vec{\phi} : \mathbb{R}^d \to \mathbb{R}^k$ function that takes in a d-dimensional vector and outputs a k-dimensional feature vector.

i.e. creates new features from the old ones.

ex) $\vec{\phi} : \mathbb{R}^2 \to \mathbb{R}^5$:

$$\vec{\phi}(x_1, x_2) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2)^T.$$

A basis function : $\phi_i : \mathbb{R}^d \to \mathbb{R}$.

↑ vector ↑ single new feature.

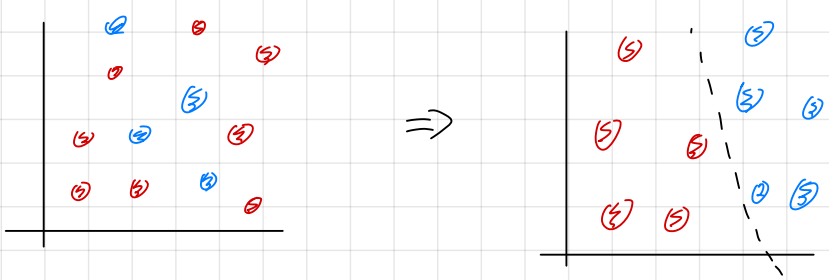
think of $\vec{\phi} : \mathbb{R}^d \to \mathbb{R}^k \Rightarrow \vec{\phi}(\vec{x}) = (\phi_1(\vec{x}), \cdots, \phi_k(\vec{x}))$.

Feature map made of many basis function.

Why feature map?

⇓

turn non-linear patterns. ⇒ linear patterns.



Input Space Feature Space.

Procedure:

Train: 1. Training set

2. Feature map, new data set

3. linear model

Predict: 1. given new input $\vec{x}$.

2. map to feature space $\vec{x} \to \vec{\phi}(\vec{x})$.

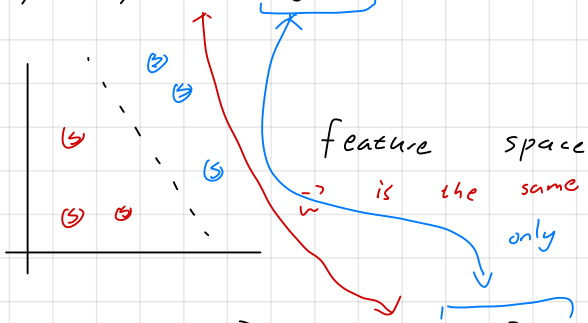
3. Predict: $H(\vec{x}, \vec{w}) = \vec{w}^* \cdot Aug(\vec{\phi}(\vec{x}))$.

Design matrix

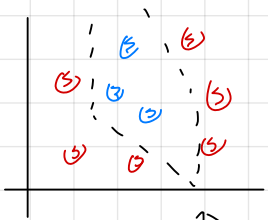
$(n \times h) \Rightarrow (n \times k)$

Prediction function:

$H_\phi(\vec{z}) = \vec{w} \cdot Aug(\vec{z})$ takes in vector $\vec{z}$ in feature space



$H(\vec{x}) = H_\phi(\vec{\phi}(\vec{x})) = \vec{w} \cdot Aug(\vec{\phi}(\vec{x}))$ take in vector $\vec{x}$ in input space



input space.

looks non-linear

ERM with Feature Maps

H_ϕ that minimizes risk in feature space

⇓

H that minimizes risk in input space.

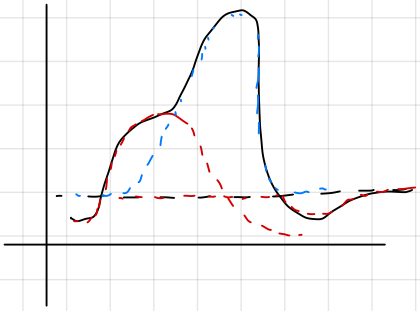
as long as H is a linear function of the parameter.

Thus, we can fit any function of

$$H(x) = w_1 \phi_1(x) + \cdots + w_k \phi_k(x)$$

Gaussian Radial Basis Function (RBF)

idea: sum of bump



$$H(x) = w_1 \text{bump}_1(x) + w_2 \text{bump}(x_2) + w_3 \text{bump}(x_3).$$

Bump:

$$\phi_i(x) = \exp\left(-\frac{(x-\mu_i)^2}{\sigma_i^2}\right)$$

measure how close x_i to μ_i.

with center μ_i and width σ_i for each Gaussian basis function.

⇓

makes very general basis function.

by adjusting $\vec{w}$, we can fit complex pattern.

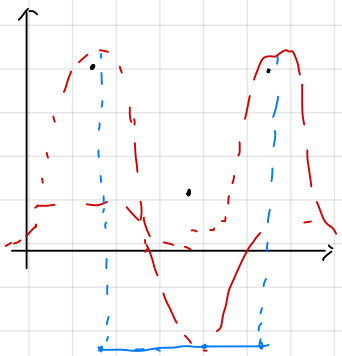
Height = w_i

Too many Gaussian function:

⇒ Overfitting.

⇓

Regularization:



as $|w| \uparrow$, $\|\vec{w}\| \uparrow$

$\|\vec{w}\|$ large as Guassian function more complex / overfitting

↓

measure the complexity of the model.

⇑

minimize $\|\vec{w}\|$ to prevent overfitting

Regularized least square :

idea : penalize large $\|\vec{w}\|$

$$\tilde{R}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}(\vec{w}\cdot\phi(\vec{x}^{(i)}) - y_i)^2 + \lambda\|\vec{w}\|^2$$

hyperparameter that controls the trade-off

↓

$\lambda\|\vec{w}\|^2$: regularizer.

regularization term

↑

Ridge Regression

$$\tilde{R}(\vec{w}) = \frac{1}{n}\|\phi\vec{w} - \vec{y}\|^2 + \lambda\|\vec{w}\|^2.$$

Calculate $\frac{d\tilde{R}}{d\vec{w}}$ and set to 0:

solution:

$$\vec{w}^* = (\phi^T\phi + n\lambda I)^{-1}\phi^T\vec{y}$$

↑

add small number λ to diagonal of $\phi^T\phi$.

⇓

improve condition number

helpful when multicolinearity exist.

$\lambda \uparrow \Rightarrow$ more regularization.

$\Downarrow$

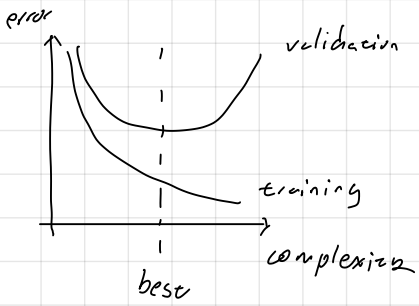
$\|\vec{w}\|^2 \downarrow$

$\Downarrow$

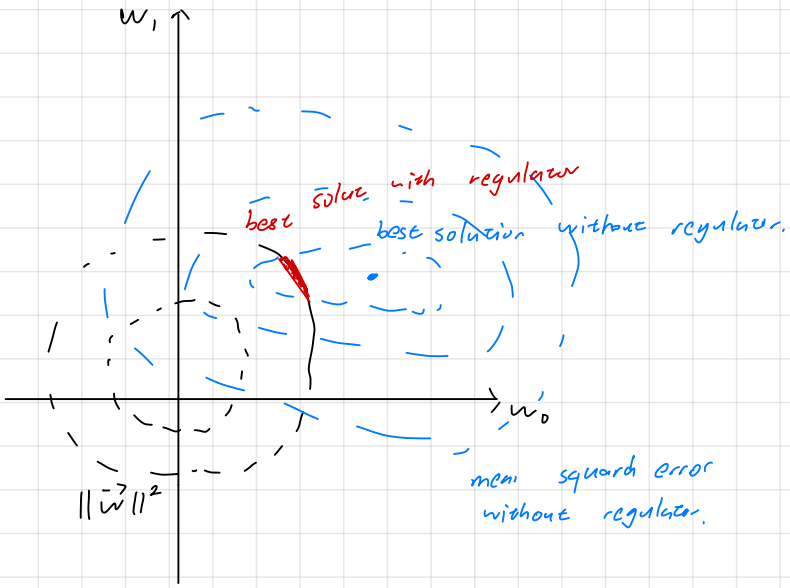
simpler model / penalize complexion model

How pick λ ?

Cross-validation.



Understanding: $\Rightarrow$ an increase in risk for a decrease in complexity.



Solution: least risk &

least $\|\vec{w}\|^2$.

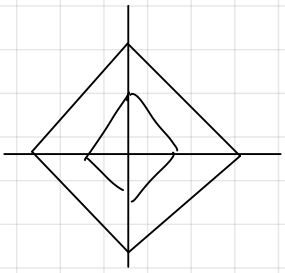
Lasso

p-norm: for $\forall p \in [0, \infty]$.

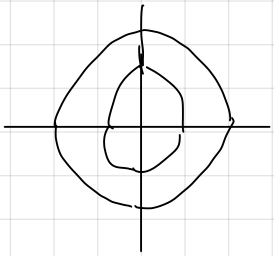
$$\|\vec{u}\|_p = \left(\sum_{i=1}^{d} |u_i|^p \right)^{1/p}.$$

$p = 2 \Rightarrow$ encliden distance.

$p = 1 \Rightarrow$ manhatten distance



1-norm.

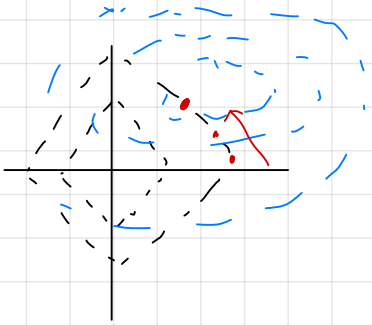


2-norm.

Use 1-norm => Lasso.

$$\widehat{R}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}(\vec{w}\cdot\phi(\vec{x}^{(i)}) - y_i)^2 + \lambda\|\vec{w}\|_1$$

- No closed form solution.
  - subgradient Descent?
- Sparse solution —> feature importance.
- ⇓ why?



General regularization:

$$\widehat{R}(\vec{w}) = R(\vec{w}) + P(\vec{w}).$$

Regularized linear model:

Loss	Regularization	Name
square	2-norm	ridge regression.
square	1-norm	Lasso
square	1-norm + 2-norm	elastic net
hinge	2-norm	soft-SVM

Other regularization method: "Early Stopping". DNN.

High-Dimensional Feature Maps

linear prediction rule → straight linear line

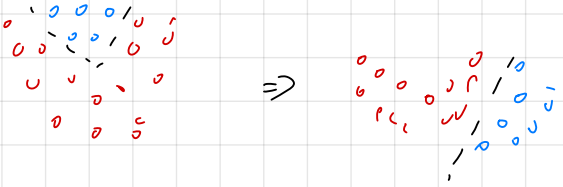
(limitation!)

⇓

Basis functions $\vec{\phi}(\vec{x})$

↓

maps data to a new space that is linearly separable.



Data space ⇒ feature space

⇒ train $H(\vec{x}) = w_0 + w_1 \phi_1(\vec{x}) + \cdots$

$\mathbb{R}^d \longrightarrow \mathbb{R}^k$

$\vec{\phi}(\vec{x})$

Decision boundary no longer a straight line.

Problem: How to choose $\vec{\phi}$?

Design a general feature map function that makes any dataset linearly-separable.

↓

idea: very high-dimensional generic feature map.

(each additional feature makes the data easier to classify).

ex) Monomial: $\vec{\phi}(\vec{x}) = (1, x_1, x_2, x_1 x_2, x_1^2, x_2^2)^T$

$(1,\ x_i,\ x_i x_j,\ x_i^2,\ \dots)^T$

So, $\vec{x} \in R^d$, then $\vec{\phi}(\vec{x}) \in R^{1+2d+\binom{d}{2}}$

$(1,\ x_i,\ x_i x_j,\ x_i^2,\ x_i x_j x_k,\ x_i^3)^T$

or $\vec{\phi}(\vec{x}) \in R^{1+3d+\binom{d}{2}+\binom{d}{3}}$

⋮

problem: computationally costly for fitting a linear prediction rule.

Kernal Trick

idea: we can train many linear predictors as if we have mapped data to feature space, without actually doing so.

$k(\vec{x}, \vec{x}') = \vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$ ← computational costly

↑

there is such function that does not require mapping to feature

↓

Kernal Function

ex) $\vec{\phi}(\vec{x}) = (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2)^T$

$k(\vec{x}, \vec{x}') = (1 + \vec{x} \cdot \vec{x}')^2$ is a kernal for this $\vec{\phi}$.

polynomial kernal

↓

$k(\vec{x}, \vec{x}') = (1 + \vec{x} \cdot \vec{x}')^k$ is kernal for k-order monomial mappings.

Idea: replacing all instances of $\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$ with $k(\vec{x}, \vec{x}')$

kernalize the algorithm

avoid explicitly mapping to feature space. (kernal trick)

- Only certain feature map have efficiently-computed kernals.

only certain learning algorithms can be kernalized.

Kernel Ridge Regression & Kernal SVM

ridge regression:

$$\underset{\vec{w}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (\vec{\phi}(\vec{x}^{(i)}) \cdot \vec{w} - y_i)^2 + \lambda \|\vec{w}\|^2.$$

$$\Updownarrow$$

$$\underset{\vec{w}}{\operatorname{argmin}} \frac{1}{n} \|\phi \vec{w} - \vec{y}\|^2 + \lambda \vec{w}^T \vec{w}$$

↗

kernalize, replace all instances of $\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$ with $k(\vec{x}, \vec{x}')$

↓

rewrite in dual form.

Fact: The solution $\vec{w}^*$ is a linear combination of $\vec{\phi}(\vec{x}^{(i)})$:

$$\vec{w}^* = \sum_{i=1}^{n} a_i \vec{\phi}(\vec{x}^{(i)})$$

Why?

$$\frac{2}{n} \sum_{i=1}^{n} (\vec{\phi}(\vec{x}^{(i)}) \cdot \vec{w} - y_i) \vec{\phi}(\vec{x}^{(i)}) + 2\lambda \vec{w}$$, gradient of regularized risk

↓

Setting to zero, solving for $\vec{w}^*$:

$$\vec{w}^* = \sum_{i=1}^{n} \underbrace{\left(-\frac{1}{n\lambda} \vec{\phi}(\vec{x}^{(i)}) \cdot \vec{w}^* - y_i\right)}_{a_i} \vec{\phi}(\vec{x}^{(i)})$$

↓

$$\vec{w}^* = \phi^T \vec{a} \quad \text{, where } \vec{a} = (a_1 \cdots a_n)^T$$

So, the problem:

$$\underset{\vec{w}}{\operatorname{argmin}} \frac{1}{n} \|\phi \vec{w} - \vec{y}\|^2 + \lambda \vec{w}^T \vec{w}$$

$\Updownarrow$

$$\underset{\vec{a}}{\operatorname{argmin}} \frac{1}{n} \|\phi \phi^T \vec{a} - \vec{y}\|^2 + \lambda \vec{a}^T \phi \phi^T \vec{a}$$

this can be kernalized.

new optimization solution:

inside ϕ, we have

$$\phi = \begin{pmatrix} \vec{\phi}(\vec{x}^{(1)}) & \cdots \\ \vdots & \ddots \\ \vec{\phi}(\vec{x}^{(n)}) & \cdots \end{pmatrix}$$

$$\begin{aligned} (\phi \phi^T)_{ij} &= \phi_i \cdot (\phi^T)_j \\ &= \phi_i \cdot \phi_j \\ &= \vec{\phi}(\vec{x}^{(i)}) \cdot \vec{\phi}(\vec{x}^{(j)}) \\ &= k(\vec{x}^{(i)}, \vec{x}^{(j)}). \end{aligned}$$

And so, (i,j) entry of $\phi\phi^T$ is $\vec{\phi}(\vec{x}^{(i)}) \cdot \vec{\phi}(\vec{x}^{(j)}) = k(\vec{x}^{(i)}, \vec{x}^{(j)})$.

$$\phi \phi^T = \begin{pmatrix} k(\vec{x}^{(1)}, \vec{x}^{(1)}) & \cdots \\ \vdots & \ddots \\ & k(\vec{x}^{(n)}, \vec{x}^{(n)}) \end{pmatrix}$$

kernal matrix

So, we have $\underset{\vec{a}}{\operatorname{argmin}} \frac{1}{n} \|K\vec{a} - \vec{y}\|^2 + \lambda \vec{a}^T K \vec{a}$

$\Downarrow$

$$\vec{a}^* = (K + n\lambda I)^{-1} \vec{y}$$

The kernal ridge regression.

Making a prediction on new point:

$$H(\vec{x}) = \vec{w}^* \vec{\phi}(\vec{x}) = \sum_{i=1}^{n} a_i \vec{\phi}(\vec{x}^{(i)}) \cdot \vec{\phi}(\vec{x})$$

$$= \sum_{i=1}^{n} a_i k(\vec{x}^{(i)}, \vec{x}).$$

A weighted sum of kernal evaluations.

Kernal Soft-SVM

$$\underset{\vec{a}}{\operatorname{argmin}} \left(\lambda \vec{a}^T K \vec{a} + \frac{1}{n} \sum_{i=1}^{n} \max\left\{ 0, 1 - y_i (K\vec{a})_i \right\} \right).$$

make new prediction:

$$H(\vec{x}) = \sum_{i \in S} a_i \, k(\vec{x}^{(i)}, \vec{x})$$

↑ the set of support vector.

Downside:

nxn kernal matrix can be very large.

Kernal Function

valid kernal: must complete the dot produce w.r.t. some mapping, $\vec{\phi}(\vec{x})$.

⇓

$$k(\vec{x}, \vec{x}') = \vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{x}')$$

New kernal constructed from other kernal functions.

- $k(\vec{x}, \vec{x}') = k_1 + k_2$
- $k(\vec{x}, \vec{x}') = k_1 \times k_2$
- $k(\vec{x}, \vec{x}') = k_3(\vec{\phi}(\vec{x}), \vec{\phi}(\vec{x}'))$
- $k(\vec{x}, \vec{x}') = f(\vec{x}) \, k_1 \, f(\vec{x}')$

Theorem:

A symmetric function k is a valid kernal

⇑⇓

the kernal matrix, K, is positive semi-definite for any choice of data $\vec{x}^{(1)} \dots \vec{x}^{(n)}$.

RBF kernal Gaussian kernal

$$k(\vec{x}, \vec{x}') = e^{\frac{-\|\vec{x}-\vec{x}'\|^2}{2\sigma^2}}$$ ← distance between $\vec{x}$ and $\vec{x}'$

$$= e^{-\gamma\|\vec{x}-\vec{x}'\|^2}, \quad \text{where } \gamma = 1/(2\sigma^2)$$

Interpretation: RBF kernal measures similarity of $\vec{x}$ and $\vec{x}'$.

very similar: $k(\vec{x}, \vec{x}) \approx 1$

very different: $k(\vec{x}, \vec{x}) \approx 0$.

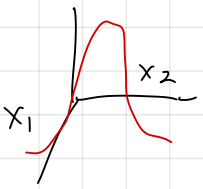
Parameter σ (or γ) controls the scale

⇓

larger σ (smaller γ), wider the Guassian.

---

ex)



$$H(x) = \sum_{i=1}^{n} a_i k(\vec{x}^{(i)}, \vec{x})$$

↓ sum of bumps

$\vec{x}, \vec{x}' \in \mathbb{R}^2$

$\vec{x} = (x_1, x_2)$

RBF kernal

prediction: $H(\vec{x}) = \sum_{i=1}^{n} a_i k(\vec{x}^{(i)}, \vec{x})$.

- One parameter a_i learned for each training point $\vec{x}^{(i)}$.
- $k(\vec{x}^{(i)}, \vec{x}) \approx 0$ for any $\vec{x}^{(i)}$ far from $\vec{x}$.
- $H(\vec{x})$ is largely determined by the training points closest to $\vec{x}$.

↓ associated weights

Since $k(\vec{x}^{(i)}, \vec{x})$ determine the distance

Understanding:

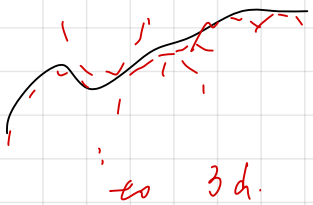
RBF function placed at each training point ⇒ sum of bumps

↓ as a surface

$H(\vec{x})$ largely determined by training point closest to $\vec{x}$.

↓

a generalization of kNN rule



to 3d.

RBF kernal Map

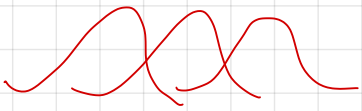
The mapping $\vec{\phi}(\vec{x})$ with entries of the form: $\mathbb{R}^2 \to \mathbb{R}^\infty$

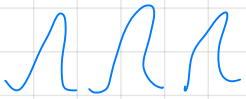
$$e^{-\|\vec{x}\|^2/2} x_i, \quad \frac{1}{\sqrt{2!}} e^{-\|\vec{x}\|^2/2} x_i x_j, \quad \frac{1}{\sqrt{3!}} e^{-\|\vec{x}\|^2/2} x_i x_j x_k, \cdots$$

mapping to an infinite dimensional Hilbert space

Hyperparameter: γ → kernel width.

- → small ↘ → underfitting
- $\gamma \downarrow$ → more wider ⇒ gaussian overlapping
- $\gamma \uparrow$ → more narrower ⇒ gaussian separating
- ↘ sto ↓ → ↓ overfitting





Probabilistic Modeling

ex) Given a new flower with $X = x$ petals, predict its species, Y.

↑ random variable

Assumption: When nature generates a new flower, it picks X & Y from some probability distribution.

- Joint distribution $\mathbb{P}(X = x, Y = y)$ gives us full info.

	$Y=0$	$Y=1$
$X=0$	0%	
	5%	
	10%	
	⋮	
$X=6$	0%	

⇒ Entries of the joint distribution table sum to 100%

$$\sum_{x \in \{0,\cdots,6\}} \sum_{y \in \{0,1\}} \mathbb{P}(X = x, Y = y) = 1.$$

- Marginal distribution for X is found by summing over values of Y

$$\mathbb{P}(X = x) = \sum_{y \in \{0,1\}} \mathbb{P}(X = x, Y = y)$$

or

$$\mathbb{P}(Y = y) = \sum_{x \in \{0,\cdots,6\}} \mathbb{P}(X = x, Y = y)$$

- Conditional Probability :

$$\mathbb{P}(Y=y \mid X=x) = \frac{\mathbb{P}(X=x, Y=y)}{\mathbb{P}(X=x)} \quad \text{or} \quad \mathbb{P}(X=x \mid Y=y) = \frac{\mathbb{P}(X=x, Y=y)}{\mathbb{P}(Y=y)}$$

	$Y=0$	$Y=1$
$X=0$	0%	0%
$X=1$	5%	0%
⋮		
$X=4$	5%	20%
!		

→ $\mathbb{P}(Y=y \mid X=1)$

$Y=0$	100%
$Y=1$	0%

→ $\mathbb{P}(Y=y \mid X=4)$

$Y=0$	20%	← 5% / 25%
$Y=1$	80%	← 20% / 25%

Bayers Theorem

$$\mathbb{P}(Y=y \mid X=x) = \frac{\mathbb{P}(X=x \mid Y=y)\,\mathbb{P}(Y=y)}{\mathbb{P}(X=x)}$$

⇓

Bayes Decision Theorem

Predict the most likely label y given $X=x$

i.e. predict the y that maximizes $\mathbb{P}(Y=y \mid X=x)$

In Binary Probabilistic Classification:

- Predict 1 if $\mathbb{P}(Y=1 \mid X=x) > \mathbb{P}(Y=0 \mid X=x)$; otherwise predict 0.

↑

Bayes Classification rule

- alternative form:

Predict 1 if $\mathbb{P}(X=x \mid Y=1)\,\mathbb{P}(Y=1) > \mathbb{P}(X=x \mid Y=0)\,\mathbb{P}(Y=0)$

otherwise predict 0.

Continuous Distribution

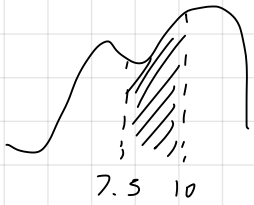
pdf : $p : \mathbb{R} \to \mathbb{R}^+$

$$\mathbb{P}(a < X < b) = \int_a^b p_X(x)\,dx$$

How likely it is to get a value of X in any interval $[a, b]$.

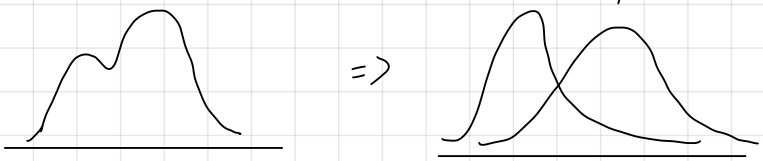
$\mathbb{P}(X = x) = 0$. => prob. of continuous r.v. being exactly a certain value is zero.

ex)



$$\mathbb{P}(7.5 < X < 10) = \int_{7.5}^{10} p_X(x)\,dx$$

- Conditional density $p(x \mid Y = y)$, conditional on Y.

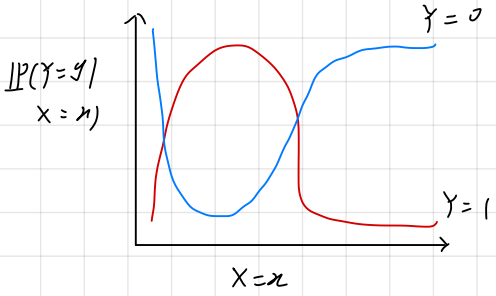


- Conditional on X :

⇓

conditional distribution of Y given X :

Discrete, b/c Y is discrete



· Joint Distribution

$p(x, 0) = p(x \mid Y=0)\,\mathbb{P}(Y=0)$

$p(x, 1) = p(x \mid Y=1)\,\mathbb{P}(Y=1)$

Decision Boundary

predict class 1 if $p(x \mid Y=1)\,\mathbb{P}(Y=1) > p(x \mid Y=0)\,\mathbb{P}(Y=0)$.

· Multi-variate Distribution

random vector $\vec{X}$

pdf: $p_X(\vec{x}) : R^d \to R^+$.

· Bayes Error

· Predict 0, true 1 ①

· predict 1, true 0 ②

$\mathbb{P}(\text{error}) = \mathbb{P}(\text{Case ①}) + \mathbb{P}(\text{Case ②})$.

Case ①: $\mathbb{P}(\text{case ①}) = \mathbb{P}(Y \text{ is actually 1, predict 0})$

$= \mathbb{P}(Y \text{ is actually 1}) \cdot \mathbb{P}(\text{predict 0} \mid Y \text{ is actually 1})$

Case ② $\mathbb{P}(\text{case ②}) = \mathbb{P}(Y \text{ is actually 0, predict 1})$

$= \mathbb{P}(Y \text{ is actually 0}) \cdot \mathbb{P}(\text{predict 1} \mid Y \text{ is actually 0})$

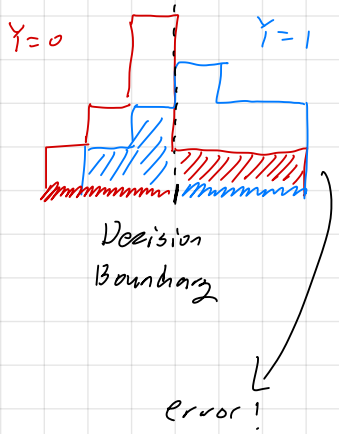
· Bayes classifier has the smallest possible error probability of any classifier.

⇓

the error is called Bayes error

· In most cases, the minimum possible error probability is > 0.

Bayes Error if $\mathbb{P}(Y=0) = \mathbb{P}(Y=1) = 0.5$.



$\sim P_0 = (x \mid Y=0)$

$\sim P_1(x \mid Y=1)$

$\mathbb{P}(Y \text{ actually } 1)\ \mathbb{P}(\text{predict } 0 \mid Y=1)$

$0.5 \cdot 0.3$

↓ area of P_1 under $Y=0$

$\mathbb{P}(Y \text{ actually } 0)\ \mathbb{P}(\text{predict } 1 \mid Y=0)$

$0.3 \cdot 0.3$

↓ area of P_0 under $Y=1$

Bayes classifier is optimal !

⇓

problem it requires knowing the joint distribution.

⇓

Estimate : do not know joint distribution.

we know the data.

X	Y
5	0
3	0
4	1
4	1

estimate joint Prob. : $\mathbb{P}(X=x, Y=y) \approx \frac{\#(X=x \ \& \ Y=y)}{n}$

estimate marginal : $\mathbb{P}(Y=y) \approx \frac{\#(Y=y)}{n}$

estimate conditional : $\mathbb{P}(X=x \mid Y=y) \approx \frac{\#(X=x \ \& \ Y=y)}{\#(Y=y)}$

$$\mathbb{P}(Y=y \mid X=x) \approx \frac{\#(X=x \ \& \ Y=y)}{\#(X=x)}$$

Law of large number : as data size $n \to \infty$, these estimated prob. converges to their true values

Using estimated prob. for Bayes classifier: (no longer optimal)

estimate $\hat{P}(Y=0 \mid X=x)$, $\hat{P}(Y=1 \mid X=x)$ } -> compare and predict based on Bayes decision rule.

Multivariate estimate:

$\hat{P}(Y=y \mid X=x_1, X=x_2 \cdots X=x_n)$

$\hat{P}(X=x_1, \cdots \mid Y=y)$

Histogram Density Estimators

estimate density for continuous distribution

↓ problem: most values never seen in data

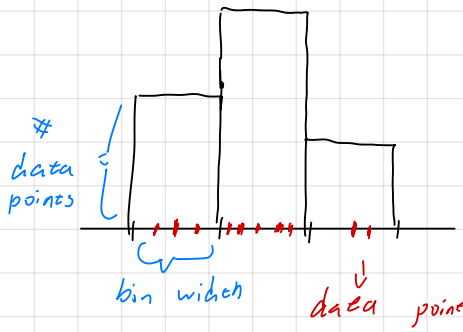
Histogram estimator

- Suppose $X_1 \cdots X_n$ came from density f.
- Divide domain into k bins: $[a_i, b_i)$.
  - within each bin, ↓ often equal-sized.

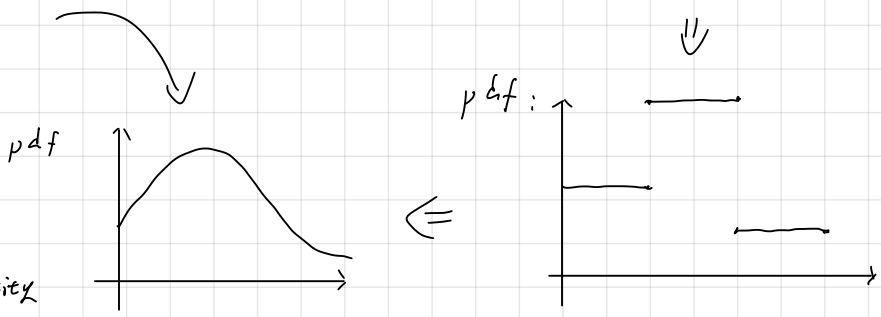
$$f(x) \text{ within bin } i \approx \frac{\#\text{ data points} \in [a_i, b_i)}{n \times (b_i - a_i)}$$

($b_i - a_i$: bin width)

histogram integrals to 1



As we get more data
- decrease bin width
- increase number of bins



As n and # bins $\to \infty$, estimator approaches true density by law of large number.

yet smooth

estimate $p(x \mid Y=y)$

Conditional probability: restrict data where $Y=1/0$ and use histogram estimator

↓

by law of large number, approaches true density

- $Y=1$
- $Y=0$

estimate $\mathbb{P}(Y=y|X=x)$, discrete prob, conditioned on continuous variable

↓

problem: x may not be seen in data

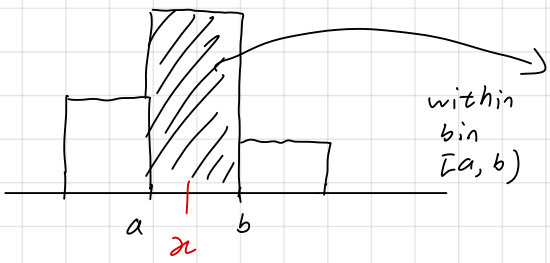
↓

Two Approaches:

① Count $\#(Y=1)$ and $\#(Y=0)$ within bin containing x

- find bin containing x
- estimate

$$\mathbb{P}(Y=y|X=x) \approx \frac{\#(Y=y \text{ within bin})}{\#(\text{points with bin})}$$



X Y

=> count $Y=y$, divide by # points

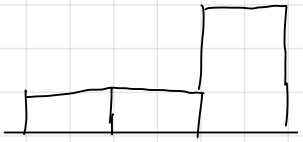
Bayes rule: predict $Y=y$ that occurs most within the bin.

② Compute from Baye's rule

- estimate $p(x|Y=y)$ $\mathbb{P}(Y=y)$ $P_X(x)$
- use baye's rule $\mathbb{P}(Y=y|X=x) = \frac{p(x|Y=y)\mathbb{P}(Y=y)}{P_X(x)}$

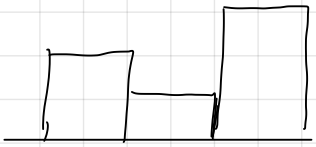
$\mathbb{P}(Y=y)$ -> count

$p(x|Y=y)$ -> restrict data, histogram estimate



$p(x|Y=1)$

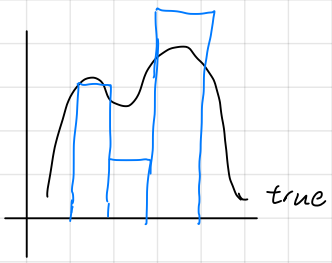
$P_X(x)$ -> histogram estimator



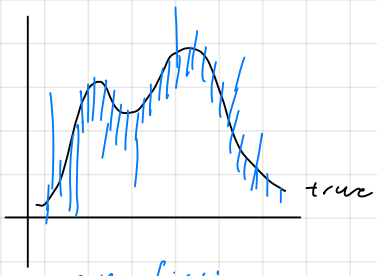
$P_X(x)$

Both approaches the same if we use the same bins

Like KNN, # of bins might result over- or under-fitting.



Under fitting



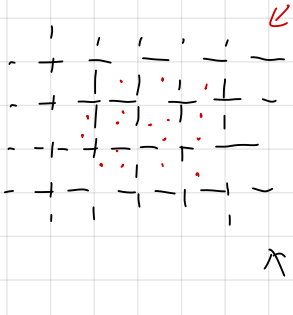
over fitting

Multivariate Histogram Density Estimators

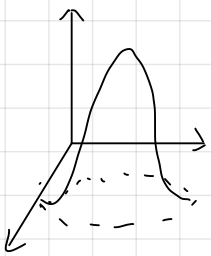
estimate $p(\vec{x})$

- divide R^d into rectangular bins with regular side-length $l_1, \cdots, l_d$
- estimate

$$f(\vec{x}) \text{ within bin} \simeq \frac{\#\text{ data points} \in [a_i, b_i)}{n \times (l_1 \times \cdots l_d)}$$



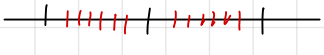
pdf
=>



as $n \to \infty$
bin

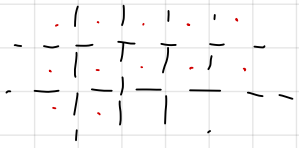
Curse of dimensionality

Not accurate b/c fewer points in each bins as we divide each dimensionality while estimating



6 points in each bin

=>



one data point in each bin.

Parametric Density Estimation

Histogram needs massive data in high dimension.

↓

has no shape of true density / very flexible but yet "data hungry"

↓

idea : assume true density follows some distribution.

ex) Gaussians

$$p(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

↓

parameters

↓ ↓

μ : center σ : width

CTL : sums of indep. random variable are Gaussian.

Parametric Distributions determined by finite number of parameters.

ex) $$p(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln x-\mu)^2}{2\sigma^2}\right)$$ - log normal.

· product of indep. positive random number

· ex) length of comments

$$p(x;k,\theta) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-x/\theta}$$ - Gamma

· ex) wait time, ...

$$p(x;x_m,\alpha) = \frac{\alpha x_m^{\alpha}}{x^{\alpha+1}}$$ - Pareto

· ex) distribution of wealth.

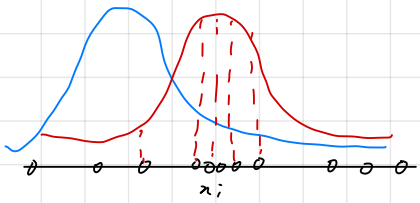
Idea : Assume data comes from some parametric density

↑

Use data to estimate parameter

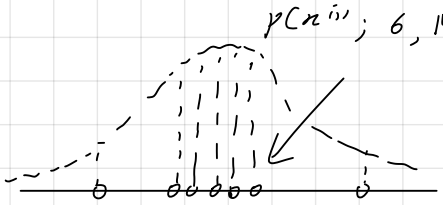
Maximum Likelihood Estimation

Use data to estimate parameter → estimating the parameter



- not likely
- more likely

data → distribution.



Let p be Gaussian prob. density function.

$p(x^{(i)}; \mu, 6)$ quantifies how likely it is to see $x^{(i)}$ if parameter μ and 6 used.

↓

$\prod_{i=1}^{n} p(x_i; \mu, 6)$ quantifies the likelihood of seeing $x^{(1)} \dots x^{(n)}$

· joint density of data

Likelihood of μ and 6, with respect to data $x^{(1)}, \dots x^{(n)}$.

↓

$$\ell(\mu, 6; x^{(1)} \dots x^{(n)}) = \prod_{i=1}^{n} p(x^{(i)}; \mu, 6).$$

Find $\underset{\mu, 6}{\operatorname{argmax}}\ \ell(\mu, 6; x^{(1)} \dots x^{(n)})$. maximize likelihood.

↓

more data points has high values

↓

· Log-likelihood → easier to work with.

$$\tilde{\ell}(\mu, 6) = \ln \ell(\mu, 6).$$

$$= \ln \prod_{i=1}^{n} p(x) \quad \leftarrow \ln(a \cdot b) = \ln(a) + \ln(b)$$

$$= \sum_{i=1}^{n} \ln(p(x))$$

· Derivate and set to 0.

Solution:

MLE for Gaussian Distribution:

$$\mu_{mle} = \frac{1}{n} \sum_{i=1}^{n} x^{(i)} \qquad 6_{mle} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_{mle})^2}$$

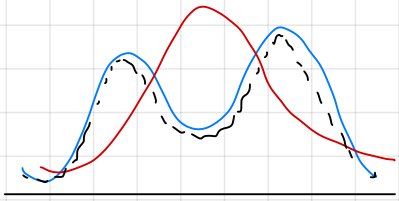
fitting data:

compute μ_{mle}, 6_{mle}.

Parametric vs. non-parametric estimation

- Non-parametic approaches can fit arbitary density, but require lots of data - espically in high dimensions.

- Parametric approaches require less data, provided that they are correctly specified.



- true
- parametric estimation.
--- non-parametic estimation

mis-specified parameter distribution.

Bayes with multiple features

with k feature, pick y maximizing

$$P_{\vec{x}}(\vec{x} \mid Y=y)\, P(Y=y)$$

Multivariate Gaussian density estimation

$$X \sim N(\mu, \sigma^2)$$

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(x-\mu)^2/\sigma^2}.$$

⇓

① d indep. r.v. $X_1 \cdots X_d$ from Gaussian, different mean but same variance.

joint density

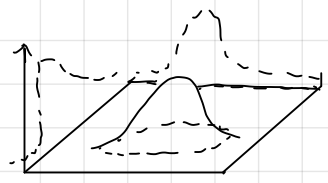
$$p(X_1, \cdots X_d) = p(X_1)\, p(X_2) \cdots p(X_d)$$

$$= \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(x-\mu_1)^2/\sigma^2}\right) \cdots (\cdots).$$

$$= \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(\frac{(x_1-\mu_1)^2 + (x_2-\mu_2)^2 + \cdots + (x_d-\mu_d)^2}{2\sigma^2}\right)$$

$$= \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(\frac{-\|\vec{x}-\vec{\mu}\|^2}{2\sigma^2}\right)$$

↑ controls the width



spherical Gaussians

2-dimesional example

② $X_1 \cdots X_d$ indep Gaussian

different mean and variances

$$p(x_1, \cdots, x_d) = \cdots$$

$$= \frac{1}{(2\pi)^{d/2} \sigma_1 \cdots \sigma_d} \exp\left(-\frac{1}{2}\left[\frac{(x_1-\mu_1)^2}{\sigma_1^2} + \cdots + \frac{(x_d-\mu_d)^2}{\sigma_d^2}\right]\right)$$

Define:

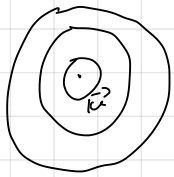
$$C = \begin{pmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_d^2 \end{pmatrix}$$

$$p(\vec{x}) = \frac{1}{(2\pi)^{d/2} |C|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^T C^{-1} (\vec{x}-\vec{\mu})\right)$$

where $|C|$ is the determinant of C.

⇓

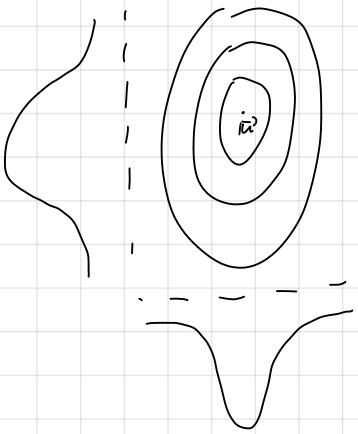
product of diagonal of C.



- Contour are (hyper) spheres
- every slice through middle gives same gaussian

Axis-Aligned Gaussians

- Contours are axis aligned (hyper) ellipses
- C is the covariance matrix
  - Diagonal
  - Entries are variances.



③ $X_1 \cdots X_d$ not indep.

Covariance:

$$Cov(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)]$$

Note: $Var(X_i) = Cov(X_i, X_i)$.

Covariance measure how much two quantities vary together.

$$Cov(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)]$$

Covariance Matrix:

$$C = \begin{pmatrix} Var(X_1) & Cov(X_1, X_2) & \cdots \\ Cov(X_1, X_2) & Var(X_2) & \\ \vdots & & \ddots \\ & & Var(X_d) \end{pmatrix}$$

C symmetry, $Cov(X_i, X_j) = Cov(X_j, X_i)$

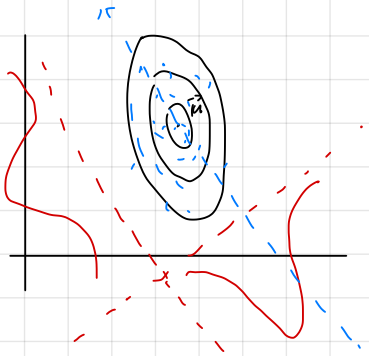
$$p(\vec{x}) = \frac{1}{(2\pi)^{d/2}|C|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^T C^{-1}(\vec{x}-\vec{\mu})\right)$$

rotated axis-aligned Gaussian

C -> covariance matrix

negative

$$\begin{pmatrix} \cdot & Cov \\ & \cdot \\ Cov & & \cdot \end{pmatrix}$$



slope negative

↓

negative covariance

which matrix are valid covariance matrices:

- if C is the rotation of some diagonal covariance matrix C_0. That is $C = RC_0$.
- C is symmetric, positive semi-definite, $x C x^T \geq 0$.

↓

eigenvalues ≥ 0.

3 Cases

① C diagonal, with all same entries

② C diagonal, different entries

③ C not diagonal

MLE for multivariate Gaussian

$\vec{\mu}$: controls Gaussian's location

C: controls Gaussian's shape

$$\vec{\mu}_{mle} = \frac{1}{n}\sum_{i=1}^{n} \vec{x}^{(i)}$$

For C, make assumption.

more parameter estimate ↓

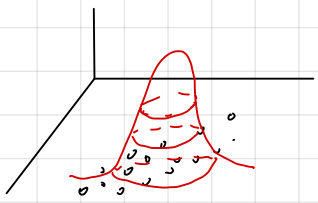
- Spherical
- Axis-aligned
- General

more-likely to overfitting

① Spherical:

variance of the data

$$\sigma^2_{mle} = \frac{1}{n}\sum_{i=1}^{n} \| \vec{x}^{(i)} - \vec{\mu}_{mle} \|^2$$



② Axis-aligned

σ_1^2 = sample variance of heights

σ_2^2 = sample variance of weight

③ General Gaussians.

MLE for covariance i and j:

$$C_{ij} = \left(\frac{1}{n}\sum_{k=1}^{n} \vec{x}_i^{(k)} \vec{x}_j^{(k)} \right) - \mu_i \mu_j$$

How: make matrix

$$\begin{pmatrix} x_1^{(1)} & x_1^{(2)} \\ \vdots & \vdots \\ x_n^{(1)} & x_n^{(2)} \end{pmatrix} \Rightarrow X = \begin{pmatrix} x_1^{(1)} - \bar{x}^{(1)} & x_1^{(2)} - \bar{x}^{(2)} \\ \vdots & \vdots \\ x_n^{(1)} - \bar{x}^{(1)} & x_n^{(2)} - \bar{x}^{(2)} \end{pmatrix} \Rightarrow \quad C = \frac{1}{n} X^T X.$$

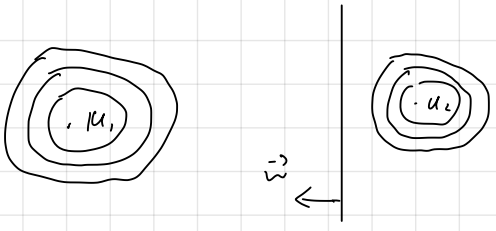
Discriminant Analysis

Bayes classifier

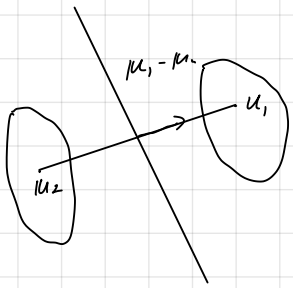
1. fit Gaussian for $p(\vec{x} \mid Y = y)$
2. for new point, predict y maximizing $p(\vec{X} = \vec{x} \mid Y = y)\, p(Y = y)$

Decision Boundary:

surface between different classification



choose class 1 if $\vec{w} \cdot \frac{(\vec{\mu}_1 - \vec{\mu}_2)}{\sigma^2} \geq \theta$



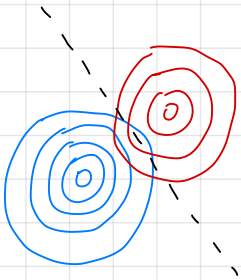
covariance matrix identical, diagonal

↓

axis-aligned Gaussians

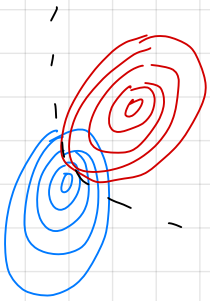
predict class 1 if $\vec{x} \cdot \vec{w} \geq \theta$.

$C = \frac{n_1 C_1 + n_2 C_2}{n_1 + n_2}$ ← compute covariance for each class, perform weighted average



- Covariance matrix equal, decision boundary linear
- Linear Discriminant Analysis (LDA).

Decision boundary



- Covariance matrix C_1, C_2 different, non-diagonal.
- Decision boundary is quadratic

Decision boundary

↓

Quadratic Discriminant Analysis.
(QDA)

Conditional Independence:

A and B independent if

$P(A \cdot B) = P(A) P(B)$

check if $P(A|B) = P(A)$
$P(B|A) = P(B)$

$\Updownarrow$

$P(A|B) = P(A)$.

A and B are independent if learning B does not influence your belief that A happens.

Real world data are always not independent

↓

but some are very close!

A, B conditionally independent given C if

$$\mathbb{P}(A, B \mid C) = \mathbb{P}(A \mid C) \cdot \mathbb{P}(B \mid C)$$

$$\Updownarrow$$

$$\mathbb{P}(A \mid B, C) = \mathbb{P}(A \mid C)$$

learning B will not inflence A given C.

learning B happens in addition to C does not inflence your belief that A happens given C.

eg) Survival and class not indep.

$$\mathbb{P}(\text{Survived} = 1) = 0.408$$
$$\mathbb{P}(\text{Survived} \mid \text{Pclass} = 1) = 0.657$$

they're close to conditionally independent given ticket price:

$$\mathbb{P}(\text{Survived} = 1 \mid \text{Pclass} = 1, \text{Fare} > 50) = 0.708$$
$$\mathbb{P}(\text{Survived} = 1 \mid \text{Fare} > 50) = 0.696$$

learing Pclass = 1 will not inflence since fare already telling Pclass

$X_1 \cdots X_d$ mutually conditionally independent given Y if

$$\mathbb{P}(X_1 \cdots X_d \mid Y) = \mathbb{P}(X_1 \mid Y) \cdot \mathbb{P}(X_2 \mid Y) \cdots \mathbb{P}(X_d \mid Y)$$

Curse of dimensionality

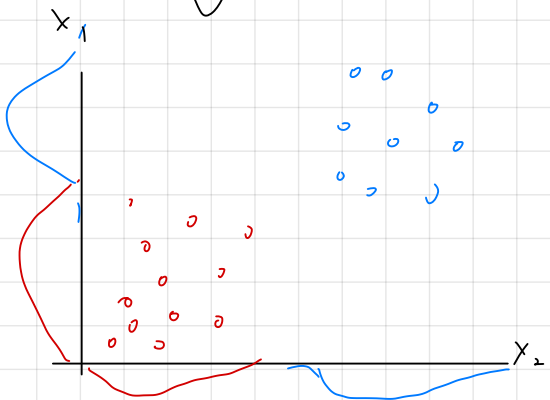
$$\downarrow$$

$$p(\vec{x} \mid Y = y) = p(x_1, \ldots, x_d \mid Y = y)$$

However, if features are mutually conditionally independent given Y

$$\downarrow$$

$$= p_1(x_1 \mid Y = y)\, p_2(x_2 \mid Y = y) \cdots p_d(x_d \mid Y = y)$$



For Bayes classifier,

estimate

$p(x_1, \dots x_d | Y = y_i)$

$= p_1(x_1 | Y = y_i) \cdots p_d(x_d | Y = y_i)$ if $x_1 \dots x_d$ mutually conditionly independent given Y

↓

reduced the number of bins needed to cover the input space for histogram estimate

↓

project points in 1-dimension for each feature

Naive Bayes Algorithm.

(1) Assume $x_1 \cdots x_d$ mutually independent given the class label.

(2) Estimate one-dimensional densities $p_1(x_1 | Y = y_i) \cdots p_d(x_d | Y = y_i)$

(3) pick y_i maximizes

$p_1(x_1 | Y = y_i) \cdots p_d(x_d | Y = y_i) \mathbb{P}(Y = y_i)$.

Gaussian Naive Bayes

$p(x_1 | Y) \cdots p(x_d | Y)$ a product of 1-d Gaussian with different mean and variance

$$C = \begin{pmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_d^2 \end{pmatrix}$$

d-dimensional gaussian with diagonal covariance matrix

each class has own diagonal covariance matrix

↓

$\Leftrightarrow$ QDA with diagonal covariance.

Naive Bayes work well in high-dimensions

practical issues

$\mathbb{P}(x_1 | Y) \cdots \mathbb{P}(x_d | Y)$ multiplying lots of small probability

potential for underflow

Trick: $\log [\mathbb{P}(X_1 = x_1 | Y = y_i) \cdots \mathbb{P}(X_d = x_d | Y = y_i) \mathbb{P}(Y = y_i)$

$= \left(\sum_{j=1}^{d} \log \mathbb{P}(X_j = x_j | Y = y_i) \right) + \log \mathbb{P}(Y = y_i)$

↑ summation not multiplication.

Regression in the sense of Probability

ERM → least square (minimize error)

↓

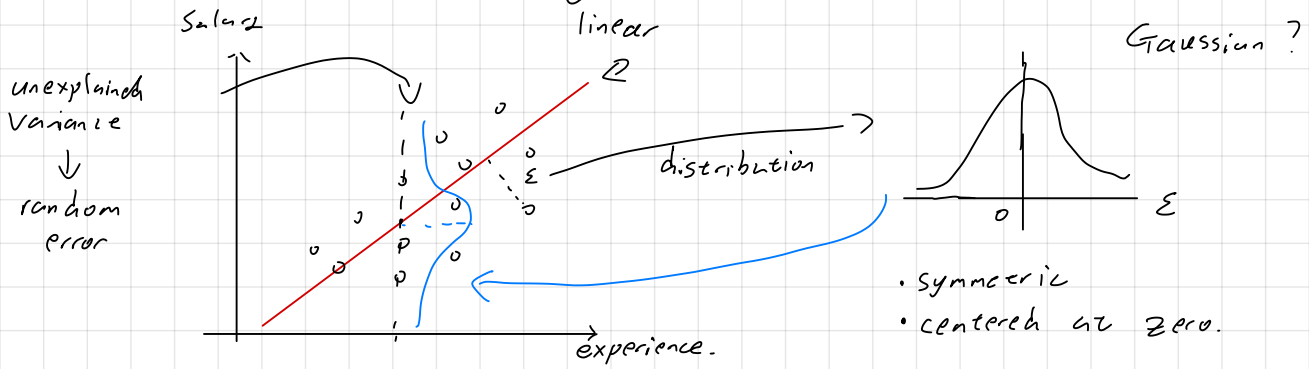
"curve fitting" approach.

From probabilistic view:

There is uncertainty in the data generation process.

Modeling uncertainty

$\text{Salary} = \underbrace{w_0 + w_1 \times (\text{Experience})}_{\text{linear}} + \varepsilon$ ← random error



$\text{Salary} = w_0 + w_1 \times (\text{Experience}) + \underbrace{N(0, \sigma^2)}_{\varepsilon}$

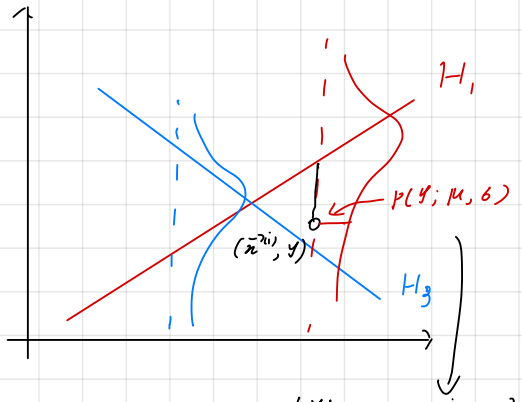
$\Leftrightarrow \text{Salary} \sim N(w_0 + w_1 \times (\text{Experience}), \sigma^2)$

In general, $Y \sim N(\text{Aug}(\vec{x}) \cdot \vec{w}, \sigma^2)$.

for any feature vector $\vec{x}$, target Y is drawn from Gaussian centered at $\text{Aug}(\vec{x}) \cdot \vec{w}$.

Given some data, $\Rightarrow$ parameters

estimate by maximizing likelihood.



$p(y_i; w_0+w_1x^{(i)}, \sigma)$

measure likelihood with respect to $(\vec{x}^{(i)}, y)$

likelihood:

$p(y;\mu,\sigma)$ gaussian pdf:

$$p(y;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(y-\mu)^2/(2\sigma^2)}$$

observe data set $\{(\vec{x}^{(i)}, y_i)\}$.

likelihood

$$p(y;\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y-\mu)^2}{\sigma^2}}$$

$\downarrow$ at training point $\vec{x}^{(i)}$

$$p(y_i;\vec{w},\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_i-\vec{w}\cdot\vec{x}^{(i)})^2}{\sigma^2}}$$

So

$$L(\vec{w},\sigma) = \prod_{i=1}^{n} p(y_i; Aug(\vec{x}^{(i)})\cdot\vec{w}, \sigma).$$

$$L(\vec{w},\sigma) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_i-\vec{w}\cdot\vec{x}^{(i)})^2}{\sigma^2}}$$

$$\tilde{L}(\vec{w},\sigma) = \ln \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_i-\vec{w}\cdot\vec{x}^{(i)})^2}{\sigma^2}}$$

$$= \sum_{i=1}^{n} \ln\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_i-\vec{w}\cdot\vec{x}^{(i)})^2}{\sigma^2}}\right)$$

$$= \sum_{i=1}^{n} \ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \ln\left(e^{-\frac{1}{2}\frac{(y_i-\vec{w}\cdot\vec{x}^{(i)})^2}{\sigma^2}}\right)$$

$\downarrow$

$$= \sum_{i=1}^{n} \ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2}(y_i-\vec{w}\cdot\vec{x}^{(i)})^2/\sigma^2$$

$$= \cdots$$

$$= -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(Aug(\vec{x}^{(i)})\cdot\vec{w}-y_i)^2 + \frac{n}{2}\ln\frac{1}{\sigma^2} - \frac{n}{2}\ln(2\pi)$$

$$\underset{\vec{w}}{\operatorname{argmax}}\left[-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(Aug(\vec{x}^{(i)})\cdot\vec{w}-y_i)^2 + \underbrace{\frac{n}{2}\ln\frac{1}{\sigma^2} - \frac{n}{2}\ln(2\pi)}_{\text{constant}}\right]$$

$$= \underset{\vec{w}}{\operatorname{argmax}}\left[\underbrace{-\frac{1}{2\sigma^2}}_{\text{constant}}\sum_{i=1}^{n}(Aug(\vec{x}^{(i)})\cdot\vec{w}-y_i)^2\right]$$

$$= \underset{\vec{w}}{\operatorname{argmax}}\left[-\frac{1}{n}\sum_{i=1}^{n}(Aug(\vec{x}^{(i)})\cdot\vec{w}-y_i)^2\right]$$

$$= \underset{\vec{w}}{\operatorname{argmin}}\left[\frac{1}{n}\sum_{i=1}^{n}(Aug(\vec{x}^{(i)})\cdot\vec{w}-y_i)^2\right].$$

⇑

minimizing mean squared error.

$$\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$$

Probabilistic view of regularization

recall ridge regression

$$\vec{w}^* = \underset{\vec{w}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (H(\vec{x}^{(i)}; \vec{w}) - y_i)^2 + \lambda \|\vec{w}\|^2.$$

Solution: $\vec{w}^* = (X^T X + n\lambda I)^{-1} X^T \vec{y}$.

helps control overfitting.

In Bayesian perspective.

↑ assign belief and updating belief

Prior on weights.

we believe it is more likely to be small (close to zero) than large.

prior belief captured by assuming

$$w_i \sim N(0, s^2)$$

in truth, w_i not random.

By bayes's rule:

$$P_{\vec{w}}(\vec{w} \mid \vec{x}, y) \propto P_Y(y \mid \vec{w}, \vec{x}) P_{\vec{w}}(\vec{w}).$$

↑ maximize ↑ likelihood ↓ prior belief update

$$\underset{\vec{w}}{\operatorname{argmax}} [P_{\vec{w}}(\vec{w} \mid \vec{x}, y)]$$

$$= \underset{\vec{w}}{\operatorname{argmax}} [P_Y(y \mid \vec{w}, \vec{x}) P_{\vec{w}}(\vec{w})]$$

$$= \underset{\vec{w}}{\operatorname{argmax}} \ln [P_Y(y \mid \vec{w}, \vec{x}) P_{\vec{w}}(\vec{w})]$$

$$= \underset{\vec{w}}{\operatorname{argmax}} [\ln P_Y(y \mid \vec{w}, \vec{x}) + \ln P_{\vec{w}}(\vec{w})]$$

$$= \underset{\vec{w}}{\operatorname{argmin}} [-\ln P_Y(y \mid \vec{w}, \vec{x}) - \ln P_{\vec{w}}(\vec{w})]$$

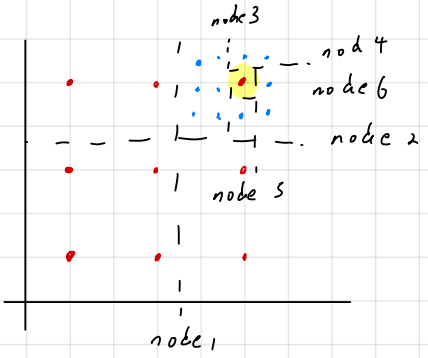
$$= \underset{\vec{w}}{\operatorname{argmin}} \left[MSE(\vec{w}) - \ln p_{\vec{w}}(\vec{w}) \right]$$

$$\Downarrow$$

$$= C + \frac{1}{2s^2} \|\vec{w}\|^2$$

Decision Tree

- A rooted tree
- Internal node ask yes/no question.
- Leaf node are decisions (class labels).
- Path from root is a sequence of "and"s:
- To make prediction → traverse tree.



start with single node containing all data points

- repeat greedy procedure
  - look at all possible questions (splits)
  - pick the one that most reduces uncertainty

$\Downarrow$

intuitive:

a good question splits the data by class.

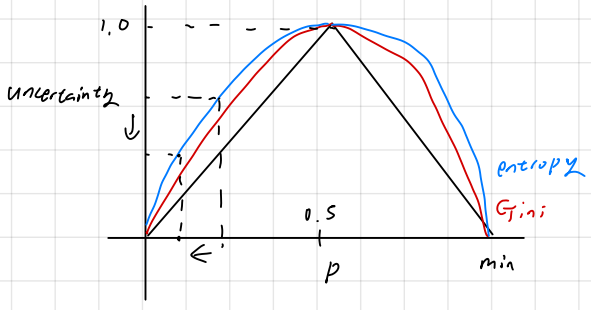
Measuring Uncertainty:

Suppose our node contains proportions

p from class +

$(1-p)$ from class −

- Misclassification rate: $\min\{p, 1-p\}$.
- Gini index: $2p(1-p)$
- Entropy: $p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$



minimizes uncertainty.

$\Downarrow$

minimize proportion in one class

p or $(1-p)$.

Let $\mu(S)$ be the uncertainty score for a set of labeled points, S.

Uncertainty $\mu(S)$

P_L ↙ ↘ P_R

$\mu(S_L)$ $\mu(S_R)$.

$P_L \mu(S_L) + P_R \mu(S_R)$

ex)

[13, 15]

$x_1 < 0.25$

↙ ↘

[4, 0] [9, 15]

initial Gini : $2 \times \frac{13}{28} \times \frac{15}{28}$

$P_L \mu(S_L) + P_R \mu(S_R) = \frac{4}{28} \cdot 0 + \frac{24}{28} \cdot 2 \cdot \frac{9}{24} \cdot \frac{15}{24} = \frac{45}{112}$

[13, 15]

$x_1 < 0.5$ → good question.

↙ ↘

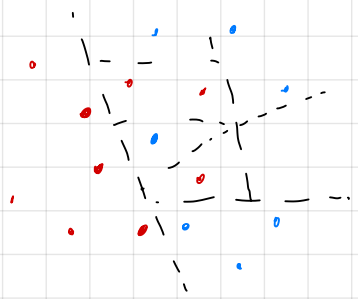
[8, 0] [5, 15]

minimize uncertainty.

$P_L \mu(S_L) + P_R \mu(S_R) = \frac{8}{28} \cdot 0 + \frac{20}{28} \cdot 2 \cdot \frac{5}{20} \cdot \frac{15}{20} = \frac{30}{112}$

Recursively ask questions that minimizes uncertainty.

Overfitting



seperate perfectly for all training points

Regularization

[10, 20]

- Pruning: simplify already-constructed tree

  - replace node by most common class
  - if change reduces validation error. keep it, otherwise, reserve it.

- Early-stopping: stop early.

  ↓

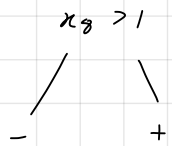
  stop recursion when:

  - node is "pure enough" (uncertainty is low).
  - tree is too deep.

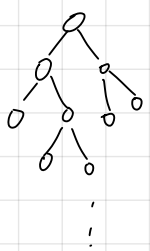
- Decision tree properties:

  - can accommodate any type of data
  - can accommodate any number of classes.
  - can perfectly fit any data see.
  - very expressive.

- Weak learner - one which performs only a little better than chance
- Decision stump - is a weak learner



- Full decision tree is a strong learner

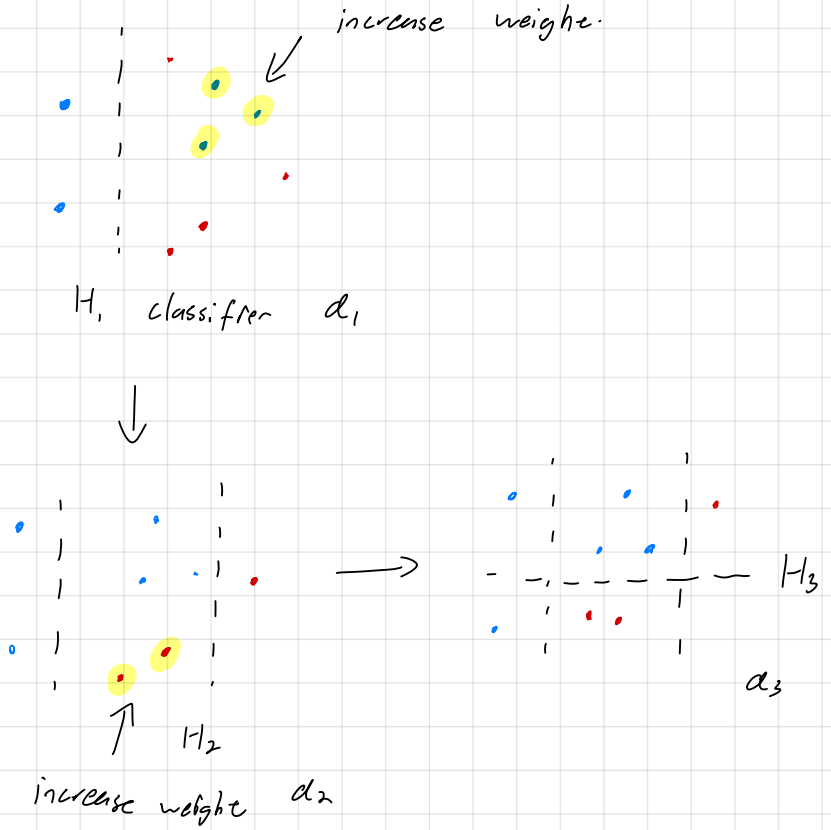


Boosting

- Train a weak classifier, $H_1: x \to [-1, 1]$.
- Increase weight of misclassified points, train another classifier H_2.

- Repeat, creating more classifier, updating weights.
- Final classifier: a linear combination of $H_1 \cdots H_k$.

ex)



Final classifier: $\text{sign}(\alpha_1 H_1(x) + \alpha_2 H_2(x) + \alpha_3 H_3(x))$.

AdaBoost

- Measuring performance:
  - suppose weights at step t are in $\vec{w}^{(t)}$, (normalized)
  - use weights to learn a classifier:
    $H_t : \mathcal{X} \to [-1, 1]$.
  - The "margin":

$$r_t = \sum_{i=1}^{n} w_i^{(t)} y_i H_t(\vec{x}^{(i)}) \in [-1, 1].$$

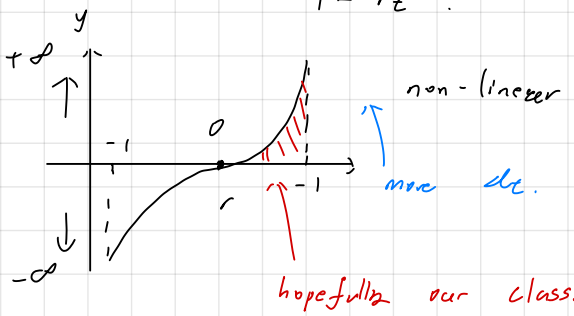
$w_i^{(t)}$: more weight on good prediction → good margin. less weight on bad prediction.

$y_i H_t(\vec{x}^{(i)})$: $[-1, 1]$

→ the larger r_t, the better H_t is doing on "important" points.

- performance of H_t:

$$\alpha_t = \frac{1}{2} \ln \frac{1 + r_t}{1 - r_t}.$$



non-linear

more α_t on larger r_t.

hopefully our classifier is better than chance.

- updating weights:

weight misclassified point more heavily.

$y_i H_t(\vec{x}^{(i)}) < 0.$

$$w_i^{(t+1)} \propto w_i^{(t)} \cdot \exp(\underbrace{-\alpha_t}_{<0} \underbrace{y_i H_t(\vec{x}^{(i)})}_{>0}).$$

$\propto$: b/c normalize w

misclassified $\rightarrow y_i H_t(\vec{x}^{(i)}) < 0$

$\downarrow$

$-\alpha_t y_i H_t(\vec{x}^{(i)}) > 0$

$\downarrow$

more weight

- Final classifier:

$$H_t(\vec{x}) = \sum_{t=1}^{T} \alpha_t H_t(\vec{x}).$$

- Algorithm:

Given data $(\vec{x}^{(1)}, y), \ldots (\vec{x}^{(n)}, y_n)$, labels in $\{-1, 1\}$.

- initialize weight vector $\vec{w}^{(1)} = (\frac{1}{n}, \frac{1}{n} \cdots \frac{1}{n})^T$
- repeat:
  - given data and weights $\vec{w}^{(t)}$ to weak learner. receive a classifier, $H_t: \mathcal{X} \rightarrow [-1, 1]$ back.
  - $\alpha_t = \frac{1}{2} \ln \frac{1 + r_t}{1 - r_t}$
  - $\vec{w}^{(t+1)} \propto \vec{w}_i^{(t)} \cdot \exp(-\alpha_t y_i H_t(\vec{x}^{(i)}))$.
  - $H_t(\vec{x}) = \sum_{t=1}^{T} \alpha_t H_t(\vec{x})$

To learn decision stump.

- try all features, threshold.
- choose that which maximizes the margin:

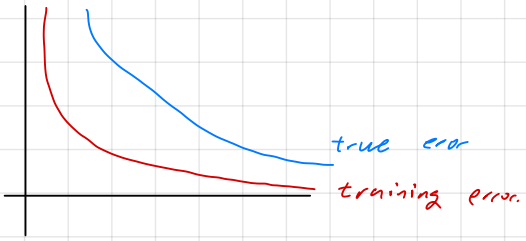
$$r_t = \sum_{i=1}^{n} w_i^{(t)} y_i \, H_t(\vec{x}^{(i)}) \in [-1, 1].$$

$\Leftrightarrow$ maximizes performance

$$\alpha_t = \frac{1}{2} \ln \frac{1+r_t}{1-r_t}$$

Theorem: Suppose on each round t, the weak learner returns a rule H, whose error on step t weighted data is $\leq \frac{1}{2} - \gamma$.
After T rounds, training error is at most $e^{-\gamma^2 T/2}$.

Boosted decision tree resistant to overfitting:



Why weak learner → fast to learn.

Random Forest

Boosted decision tree: → slow, train decision with all data / feature.

Prevent decision tree from overfitting by "hiding data" randomly.

Train a bunch of trees, quickly
Average them to make final prediction

For $t = 1$ to T

Choose n' training point randomly, with replacement

Fit decision tree H_t.

At each node, restrict to one of k features, chosen randomly.

Final classifier: majority vote of $H_1 \cdots H_T$

common setting: $n' = n$ (bootstrap)

$k = \sqrt{d}$.