Example 1) ML Classification task

Pattern $\longrightarrow \boxed{\text{Alg}} \longrightarrow$ label

Notation :

① $\mathcal{X}$: Feature space = Set of all feature vectors
For eg. $\mathcal{X} = \mathbb{R}^d$

② Classifier $h : \mathcal{X} \rightarrow \{1, ---, M\}$

Simple Case : $M=2$, $\{0,1\}$ or $\{-1,+1\}$

How to design classifiers & how to judge their performance?

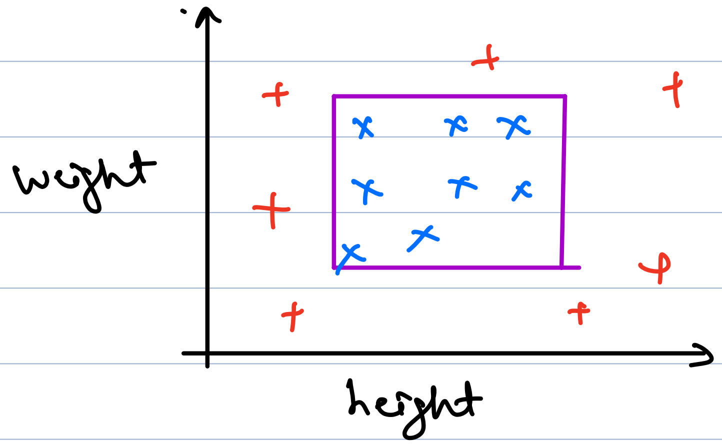
Input : $\{ (x_i, y_i), i=1 --- n \}$ Training dataset

feature vector, class label

Using training data, learn an appropriate classifier h

Test : Test & validate the h on "new" data

Example: Medium-build



Regression task:

Training data: $\{(x_i, y_i), i = 1 \ldots n\}$

$x_i \in \mathbb{R}^d, \ y_i \in \mathbb{R}$

Need is to learn a function $h: \mathcal{X} \to \mathbb{R}$

Examples: prediction of stock prices, etc

Curve-fitting: $\{(x_1, y_1), \ldots (x_n, y_n)\}$

Find a function f s.t. $y = f(x_i)$

Generalization:- Obtain a classifier/regression function using a training dataset s.t. the error on "unseen" (test) data is low.

Assumption: There is a distribution underlying the data.

Linear models

Ref: Bishop's book

Max-likelihood & least-squares

$$(*) \rightarrow y = \theta^T x + \epsilon, \quad \text{where} \quad \epsilon \sim N\left(0, \frac{1}{\beta}\right)$$

unknown

precision parameter

independent & identically distributed

Suppose $\mathcal{D}_n = \{(x_i, y_i), i = 1 \ldots n\}$ iid & satisfying $(*)$.

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} \frac{\sqrt{\beta}}{\sqrt{2\pi}} \exp\left(-\frac{\beta}{2}\left(y_i - \theta^T x_i\right)^2\right)$$

$$\ell(\theta) = \log \mathcal{L}(\theta) = \frac{n}{2}\log\beta - \frac{n}{2}\log 2\pi - \beta\left[\frac{1}{2}\sum_{i=1}^{n}\left(y_i - \theta^T x_i\right)^2\right]$$

To maximize $\ell(\theta)$, it is enough to find

$$\hat{\theta}_{ML} = \arg\min_{\theta} \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \theta^T x_i\right)^2 \leftarrow \text{empirical risk minimization}$$

Linear Regression: Given data $\{(x_i, y_i);\ i=1 \ldots n\}$

$x_i \in \mathbb{R}^d,\ y \in \mathbb{R}$

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(x_i^T\theta - y_i)^2$$

Minimize J: $A = \begin{bmatrix} - & x_1^T & - \\ - & x_2^T & - \\ & \vdots & \\ - & x_n^T & - \end{bmatrix}_{n \times d}$

$$A\theta = \begin{bmatrix} x_1^T\theta \\ \vdots \\ x_n^T\theta \end{bmatrix} \qquad Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$A\theta - Y = \begin{bmatrix} x_1^T\theta - y_1 \\ \vdots \\ x_n^T\theta - y_n \end{bmatrix}$$

$$(A\theta - Y)^T(A\theta - Y) = \sum_{i=1}^{n}(x_i^T\theta - y_i)^2 = 2J(\theta)$$

$$\nabla J(\theta) = \frac{1}{2}\nabla(A\theta - Y)^T(A\theta - Y) = A^T(A\theta - Y)$$

So, $\nabla J(\theta) = 0 \iff (A^TA)\theta = A^TY$

$$\nabla J(\theta) = 0 \iff (A^T A)\theta = A^T Y$$

Can we write $\theta = (A^T A)^{-1} A^T Y$?

Yes, if A is full rank.

since full rank $A \Rightarrow A^T A$ is invertible

(Why? argue $\text{rank}(A) = \text{rank}(A^T A)$ by showing $N(A) = N(A^T A)$)

Lecture-4: A popular ML algorithm for training

Want to Solve $\min_{\theta} f(\theta) = \frac{1}{m}\sum_{i=1}^{m} f_i(\theta)$

Common assumption: f_i smooth & f is convex/ strongly-convex

Batch GD:

$$\theta_{n+1} = \theta_n - a_n \left(\underbrace{\frac{1}{m}\sum_{i=1}^{m} \nabla f_i(\theta_n)}_{\text{computationally expensive for large } m} \right) \leftarrow \text{Noiseless algorithm}$$

(e.g. $m \to$ # training examples in a large dataset)

Alternative: SGD

Pick "i_n" uniformly at random in $\{1 \ldots m\}$

$$i_n = \begin{cases} 1 & \text{w.p. } 1/m \\ \vdots & \\ m & \text{— " — } 1/m \end{cases}$$

$$\theta_{n+1} = \theta_n - a_n \nabla f_{i_n}(\theta_n) \to \text{One sample gradient used for update}$$

"Specializing SGD for linear models":

Linear regression: $$f(\theta) = \frac{1}{2m}\sum_{i=1}^{m} \underbrace{(x_i^T\theta - y_i)^2}_{f_i(\theta)} \quad (1)$$

Let $A = \frac{1}{m}\sum_{i=1}^{m} x_i x_i^t$, $b = \frac{1}{m}\sum y_i x_i$ (assume A has full col-rank)

Then, minimizer $\hat{\theta}$ of (1) is $\hat{\theta} = A^{-1} b$

$A \to d \times d$ matrix

Incremental inverse computation: $O(d^2 m)$

Woodbury's Identity: $(A+BCD)^{-1} = A^{-1} - A^{-1}B(C + DA^{-1}B)DA^{-1}$

Sherman Morrison lemma: $(A + uv^T)^{-1} = A^{-1} - \dfrac{A^{-1}uv^TA^{-1}}{1+v^TA^{-1}u}$

SGD alternative:

$$\theta_{k+1} = \theta_k - a_k \nabla f_{i_m}(\theta_k)$$

pick a sample (x_{i_m}, y_{i_m}) unif. at random from $\{(x_i, y_i), i=1 \ldots m\}$

$\to O(d)$ computation per iteration

$$\theta_{k+1} = \theta_k + a_k \underbrace{(y_{i_m} - \theta_k^T x_{i_m})x_{i_m}}_{(*)}$$

$\theta_k \longrightarrow \hat{\theta}$ as $k \to \infty$.

$(*)$ is a LSA algorithm.

Take expectation & equate to zero

$$\frac{1}{m}\sum_{i=1}^{m}(y_i - \theta^T x_i)x_i = 0 \iff A\hat{\theta} = b$$

Some notation before example 2:

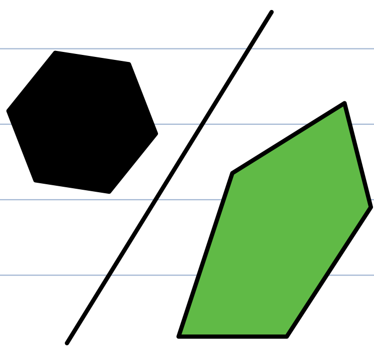
Formally,

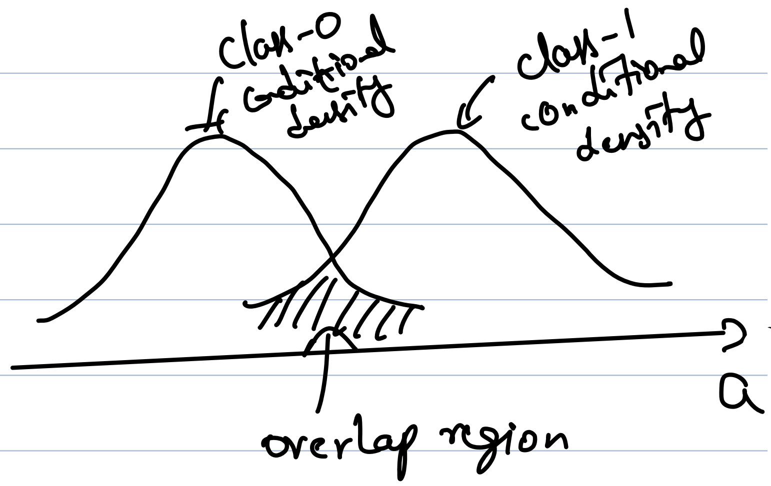
f_i : conditional density of features from class-i

Let $x = (x_1, \ldots x_d) \in \mathbb{R}^d$ represent a feature vector

$f_i(x)$: joint density of $(x_1, \ldots x_d)$ given that x is in class-i

Note: A feature vector "x" can belong to different classes with different probabilities





Example 2: Logistic Regression

Linear models for classification

Ref: Sec 4.2 of Bishop's book

Probabilistic Generative model:

Bayes classifier

Model the class-conditional densities & prior

Then, compute the posterior

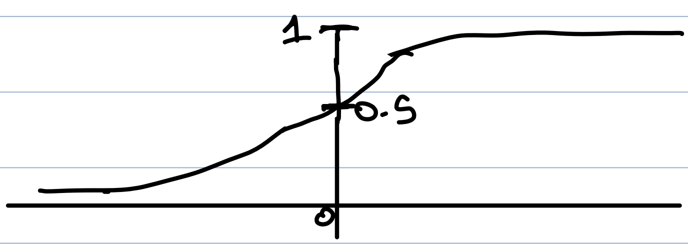
Two-class classification problem:-

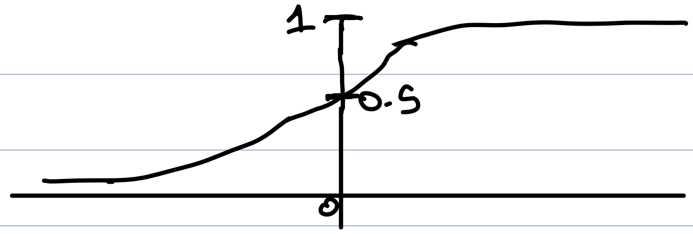
$$q_1(x) = \frac{f_0(x)\, p_0}{f_0(x)\, p_0 + f_1(x)\, p_1}$$

$$= \frac{1}{1 + \frac{f_1(x)\, p_1}{f_0(x)\, p_0}}$$

$$q_1(x) = \frac{1}{1 + \exp(-w)}, \quad w = \log \frac{f_0(x)\, p_0}{f_1(x)\, p_1}$$

$$= \sigma(w)$$





Note: ① $\frac{d\sigma(w)}{da} = \sigma(w)(1-\sigma(w))$ ← check this

② $\sigma(-w) = 1-\sigma(w)$

③ Logit function: $w = \log\left(\frac{\sigma}{1-\sigma}\right)$

Classification task is easier if "a" has a simple functional form, such as "linear".

Example:- Class conditional densities are Gaussian with the same covariance matrix

$$f_i(x) = \frac{1}{(2\pi)^{d/2}} \frac{1}{\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu_i)^T \Sigma^{-1}(x-\mu_i)\right), \quad i=0,1$$

Calculating the sigmoid parameter "a" in this case:

$$a = \log\frac{f_0(x)}{f_1(x)} + \log\frac{p_0}{p_1}$$

$$\log \frac{f_0(x)}{f_1(x)} = -\frac{1}{2}(x-\mu_0)^T \Sigma^{-1} (x-\mu_0) + \frac{1}{2}(x-\mu_1)^T \Sigma^{-1} (x-\mu_1)$$

$$= x^T \Sigma^{-1} (\mu_0 - \mu_1) - \frac{1}{2} \mu_0^T \Sigma^{-1} \mu_0 + \frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1$$

$$= w^T x + w_0,$$ where

$$w = \Sigma^{-1}(\mu_0 - \mu_1),$$ and

$$w_0 = -\frac{1}{2} \mu_0^T \Sigma^{-1} \mu_0 + \frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1$$

So, $\eta_i(x) = \sigma(w^T x + w_0)$

Max-likelihood estimation:

Dataset: $\{(x_i, y_i), i = 1 \ldots n\}$

class labels $\in \{0, 1\}$

$y_i = +1$ if class-0

To estimate: $p_0, \mu_0, \mu_1, \Sigma$

For a data point from class-0,

$$f(x_i, \text{class-0}) = p_0 f_0(x_i)$$

Similarly, $f(x_i, \text{class-1}) = (1 - p_0) f_1(x_i)$

Likelihood function:

$$\mathcal{L}(p_0, \mu_0, \mu_1, \Sigma) = \prod_{i=1}^{n} \left(p_0 \underbrace{f_0(x_i)}_{N(\mu_0, \Sigma)}\right)^{y_i} \left((1-p_0) \underbrace{f_1(x_i)}_{N(\mu_1, \Sigma)}\right)^{1-y_i}$$

Estimate p_0:

log-likelihood: $\ell(p_0, \mu_0, \mu_1, \Sigma) = \log \mathcal{L}(p_0, \mu_0, \mu_1, \Sigma)$

In $\ell(p_0, \mu_0, \mu_1, \Sigma)$, the terms that depend on p_0 are as follows:

$$\nabla_{p_0} \left(\sum_{i=1}^{n} y_i \log p_0 + (1-y_i) \log (1-p_0) \right) = 0$$

$$p_0 = \frac{1}{n} \sum_{i=1}^{n} y_i = \frac{n_0}{n_0 + n_1}, \text{ where}$$

$n_0 = \#$ of points in class-0

$n_1 = \#$ of points in class-1

Estimate μ_0:

$$\nabla_{\mu_0} \left(\ell(p_0, \mu_0, \mu_1, \Sigma) \right) = 0$$

$$\nabla_{\mu_0} \left(-\frac{1}{2} \sum_{i=1}^{n} y_i (x_i - \mu_0)^T \Sigma^{-1} (x_i - \mu_0) \right) = 0$$

$$\mu_0 = \frac{1}{n_0} \sum_{i=1}^{n} y_i x_i$$

$y_i = 1$ if $x_i \in$ class-0.

Similarly $\mu_1 = \frac{1}{\ldots} \sum (1-y_i) x_i$

$\ldots \mu_1 = \frac{1}{n_1} \sum_{i=1} (1-y_i) x_i$

Estimating the covariance matrix Σ:

The terms involving Σ in the log-likelihood function are

$$\Big(-\frac{1}{2} \sum_{i=1}^{n} y_i \log|\Sigma| - \frac{1}{2} \sum_{i=1}^{n} y_i (x_i - \mu_0)^T \Sigma^{-1} (x_i - \mu_0)$$

$$- \frac{1}{2} \sum_{i=1}^{n} (1-y_i) \log|\Sigma| - \frac{1}{2} \sum_{i=1}^{n} (1-y_i)(x_i - \mu_1)^T \Sigma^{-1} (x_i - \mu_1) \Big)$$

$$= -\frac{n}{2} \log|\Sigma| - \frac{n}{2} \mathrm{Tr}(\Sigma^{-1} S), \quad \text{where}$$

$$S = \frac{n_0}{n} S_0 + \frac{n_1}{n} S_1, \quad \text{———} (1)$$

$$S_0 = \frac{1}{n_0} \sum_{i \in \text{class-0}} (x_i - \mu_0)(x_i - \mu_0)^T$$

$$S_1 = \frac{1}{n_1} \sum_{i \in \text{class-1}} (x_i - \mu_1)(x_i - \mu_1)^T$$

The ML-estimate of Σ is "S"

Note: In the univariate case, it is easy to infer the S in (1) is the ML estimate.

$$\pm n \quad = -\frac{1}{2} \sum y_i (x_i - \mu_0)^2 \times \left(-\frac{1}{s^2}\right)$$

$$-\frac{1}{2}\sum(1-y_i)(x_i-\mu_1)^2 \times \left(-\frac{1}{S^2}\right)$$

$$n = \frac{1}{S}\left[\sum y_i (x_i-\mu_0)^2 + \sum(1-y_i)(x_i-\mu_1)^2\right]$$

$$S = \left[\frac{n_0}{n}\times\frac{1}{n_0}\sum_{i\in \text{class-}0}(x_i-\mu_0)^2 + \frac{n_1}{n}\times\frac{1}{n_1}\sum_{i\in\text{class-}1}(x_i-\mu_1)^2\right]$$

$$= \frac{n_0}{n}S_0 + \frac{n_1}{n}S_1, \text{ where } S_0 = \frac{1}{n_0}\sum_{i\in\text{class-}0}(x_i-\mu_0)^2$$

$$S_1 = \frac{1}{n_1}\sum_{i\in\text{class-}1}(x_i-\mu_1)^2$$

Probabilistic discriminative models

We have seen that

(i) For a 2-class classification problem, the posterior probability of each class can be written as a logistic sigmoid acting on a linear function of the feature vector.

ML approach can be adopted to estimate density parameters.

(ii) Alternatively, we the functional form of the generalized linear model & determine the parameters (weights) directly.

$$\eta_0(x) = \sigma(w^T x), \quad \eta_1(x) = 1 - \eta_0(x)$$

ML-approach:- $x \in \mathbb{R}^d$

Take Gaussian case,

parameters to estimate are

(i) $\mu_0, \mu_1 \Rightarrow 2d$ parameters

(ii) Σ (symmetric) $\Rightarrow \frac{d(d+1)}{2}$ parameters

(iii) prior prob p_0

Overall ML approach requires $O(d^2)$ parameters to estimate

If we use the generalized linear model & estimate parameters directly, then the complexity of the task reduces to estimating $O(d)$ # of parameters.

Logistic regression:

Dataset $\{(x_i, y_i), i = 1 \dots n\}$ $y_i \in \{0, 1\}$

Likelihood $\mathcal{L}(w) = \prod_{i=1}^{n} \left(\sigma(w^T x_i)\right)^{y_i} \left(1 - \sigma(w^T x_i)\right)^{1-y_i}$

Cross-entropy loss $\quad l(\omega) = -\log \mathcal{L}(\omega)$

$$l(\omega) = -\left[\sum_{i=1}^{n} y_i \log(\sigma(\omega^T x_i)) + (1-y_i)\log(1-\sigma(\omega^T x_i))\right]$$

$$\nabla l(\omega) = -\sum_{i=1}^{n} \frac{y_i}{\sigma\cancel{(\omega^T x_i)}} \sigma\cancel{(\omega^T x_i)}(1-\sigma(\omega^T x_i))\, x_i$$

$$-\sum_{i=1}^{n} \frac{(1-y_i)}{(1-\cancel{\sigma(\omega^T x_i)})} \sigma(\omega^T x_i)\,(1-\cancel{\sigma(\omega^T x_i)})\, x_i$$

$$= -\sum_{i=1}^{n} \left[y_i (1-\sigma(\omega^T x_i))\, x_i - (1-y_i)\,\sigma(\omega^T x_i)\, x_i \right]$$

$$\nabla l(\omega) = \sum_{i=1}^{n} (\sigma(\omega^T x_i) - y_i)\, x_i$$

The gradient descent update

$$\omega_{k+1} = \omega_k - \alpha_k \left[\sigma(\omega_k^T x(k)] - y(k)\right] x(k)$$

where $(x(k), y(k))$ could be chosen such that we cycle through the dataset or we could pick $(x(k), y(k))$ unif at random from the dataset $\{x_i, y_i\}_{i=1...n}$

Claim: l is strictly convex

Example 3: Matrix factorization

"Sparse" data   Want: Low-rank matrix estimate

Notation: $A_j \leftarrow n \times p$ matrix, $j = 1, \ldots, m$, "Given"
$X \leftarrow n \times p$ matrix "Unknown"

Aim:
$$\min_{X} \frac{1}{2m} \sum_{j=1}^{m} \left(\langle A_j, X \rangle - y_j \right)^2 \quad \text{—— } (*)$$

trace of $A_j^T X$

Regularized variant:

$$\min_{X} \frac{1}{2m} \sum_{j=1}^{m} \left(\langle A_j, X \rangle - y_j \right)^2 + \lambda \|X\|_*$$

nuclear norm = sum of singular values

This is a convex-optimization problem

Another variation: Suppose $X = LR^T$, $L \in n \times r$ matrix
$R \in p \times r$ matrix

"r" $\leftarrow$ rank $\quad r \ll \min(n, p)$

(*) becomes

$$\min_{L, R} \frac{1}{2m} \sum_{j=1}^{m} \left(\langle A_j, LR^T \rangle - y_j \right)^2 \quad \text{—— } (**)$$

In $(**)$, # elements $= (n+p)r \ll np$ (in x)

$(**) \leftarrow$ non-convex optimization problem.

Non-negative matrix factorization requires L & R entries to be ≥ 0.

Example 4: optimization in ML training

$$\min_{x} \sum_{i=1}^{m} f_i(x) + \lambda \underbrace{R(x)}_{\text{regularizer}}$$

$f_i, R \to$ convex in typical training problems (e.g. linear regression, SVM)

$f_i \to$ non-convex in deep learning

e.g. Image dataset

$\{ a_j, j = 1 \ldots m \}$

$a_j \to n$-pixel image

Binary Classification problem Image $\to \boxed{} \to$ Car or chair

Dataset $\{(a_j, b_j), j = 1 \ldots m\}$

$$\min_{x} \quad f(x) = \sum_{j=1}^{m} l\left(f_x(a_j), b_j\right)$$

SVM

$\max\left(0, 1 - b_j\left(x^T a_j\right)\right)$

Hinge-loss

deep-learning

"non-convex"

loss function