Classification as a Regression Problem

- Target variable $y \in \{C_1, C_2, C_3, \cdots, C_K\}$;
- To treat **classification as a regression problem** we should transform the target *y* into numerical values;
- The **choice** of numerical class representation is quite arbitrary;
- Careful numerical class representation is a **critical step**.

Binary Classification

Let us represent the classes C_0 and C_1 with numerical values 0 and 1, i.e., if $y \in C_0$ then *y* = 0, and if $y \in C_1$ then *y* = 1

Since we have assigned numeric values to classes, binary classification can be considered to be a regression problem. The optimal predictor for regression is

$$f^*(\mathbf{x}) = E[y \mid \mathbf{x}] = 0 \cdot P(y \in C_0 \mid \mathbf{x}) + 1 \cdot P(y \in C_1 \mid \mathbf{x}) = P(y \in C_1 \mid \mathbf{x})$$

Therefore, the optimal predictor outputs the posterior probability of class C_1.
By applying the Bayes classification rule we can obtain the Bayes classifier!!

Important Conclusion:
With appropriate class representation, the optimal classification is equivalent to optimal regression.

Multi-class Classification (*K* classes)

Could the previous result be generalized to multi-class classification?

Example.
3 classes: “white”, “black”, and “blue”.
Let us examine the representation: y = -1 (class is “white”), y = 0 (class is “black”), and y = 1 (class is”blue”)
Discussion: The representation is inappropriate since it enforces order; implies that “white” and “blue” are further away then let’s say “black” and “blue.” What if it is evident that an example could be either “white” or “blue,” but definitely not “black”? The proposed representation would probably lead to a completely misleading answer “black”!!

Solution:
Decompose the multi-class problem to *K* binary classification problems:
Problem *i* $(1 \le i \le K)$:

if $y \in C_i$ then *y* = 0
if $y \notin C_i$ then *y* = 1

The regression function $E[y \mid \mathbf{x}]$ on **Problem *i*** will equal the posterior $P(y \in C_i \mid \mathbf{x})$. By repeating for all K classes, all posteriors will be available, which is sufficient to construct the Bayes classifier!!

Approaches to Minimizing MSE from a Finite Dataset

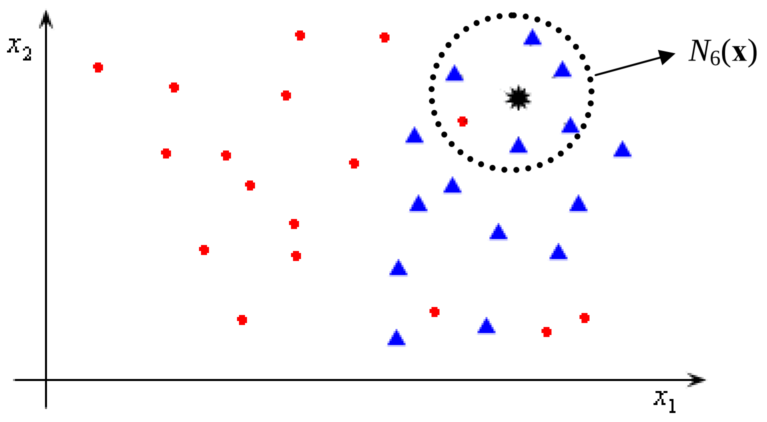
Goal: Given a dataset *D* find a mapping *f* that minimizes MSE.

Two extreme approaches:

1. Nearest neighbor algorithm (non-parametric approach)
2. Linear regression (parametric approach)

Non-parametric approach assumes that data points close in the attribute space are similar to each other. It does not assume any functional form.
Parametric approach assumes a functional form: e.g., the output is a linear function of the inputs.

Nearest Neighbor Algorithms



k-nearest neighbor (k-NN): $f(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} y_i$, where $N_k(\mathbf{x})$ are the k nearest neighbors of **x**

Regression:	The prediction is the **average** y among the *k* nearest neighbors
Classification:	The prediction is the **majority class** among the k nearest neighbors

k-NN is also known as a "lazy learning algorithm" because it does not perform any calculations prior to seeing a data point; it has to analyze the whole dataset for the nearest neighbors every time a new data point appears.
Note: Parametric learning algorithms learn a function from the dataset; they are much faster in giving predictions but need to spend some time beforehand.

Theorem: If $N \to \infty, k \to \infty, \frac{k}{N} \to 0$ **(large number of neighbors in a very tight neighborhood)** k-NN is an optimal predictor.

Practically, if we have 2 dimensions (2 attributes), k-NN is a good try; if we have more attributes, we may run out of data points (in practice, data size is always limited).

Example: Generate an M-dimensional dataset such that all attributes X_i, $i = 1, 2, \ldots M$, are uniformly distributed in interval [0,1]. What is the length r of the size of a hypercube that contains 10% of all data points (i.e. 10% of the hypercube volume)? Answer: $r = 0.1^{1/M}$.

M	*r*=0.1	*r*=0.01	*r*=0.001
1	0.1	0.01	0.001
2	0.31	0.10	0.03
5	0.63	0.40	0.25
10	0.80	0.63	0.50
100	0.98	0.95	0.93
1000	0.998	0.995	0.993

In high dimensions, all neighboring points are far away, and could not be used to accurately estimate values with k-NN!!

Linear Regression

Assumes a **functional form**

$$f(\mathbf{x},\boldsymbol{\theta}) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \ldots + \theta_M \cdot x_M \quad \text{(Eq.2)}$$

where **x** = (x_0, x_1, …x_M) are the attributes and **θ** = (θ_0, θ_1, …θ_M) are the function parameters.

More generally,

$$f(\mathbf{x},\boldsymbol{\theta}) = \theta_0 + \theta_1 \cdot \phi_1(x) + \theta_2 \cdot \phi_2(x) + \ldots + \theta_M \cdot \phi_M(x)$$

where φ's are the so called basis functions

Example:

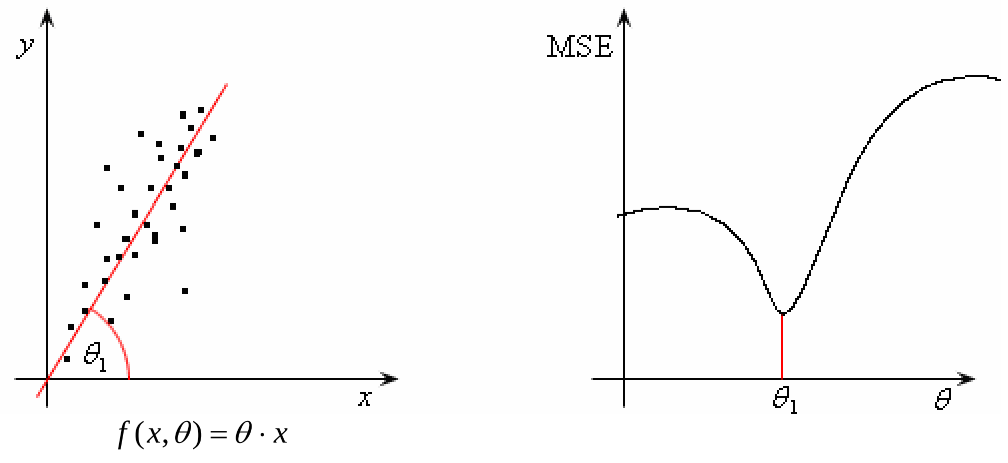
$f(\mathbf{x},\boldsymbol{\theta}) = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2^2 + \theta_3 \cdot x_1^4$, where **x** = ($x_1$, x_2) are the attributes and **θ** = (θ_0, θ_1, θ_2, θ_3) are the function parameters. Note that function *f*(**x**,θ) from the example is linear in the parameters. We can easily transform it into a function from (Eq.2) by introducing new attributes x_0'=1, x_1'=x_1 and x_2'=x_2^2, and x_3'=x_1^4.

Linear regression is suitable for problems where functional form *f*(**x**,θ) is known with sufficient certainty.

Learning goal: Find **θ** that minimizes MSE

MSE is a function of parameters **θ**, so the problem of minimizing MSE can be solved by standard methods of the unconstrained optimization.

Illustration of the regression:



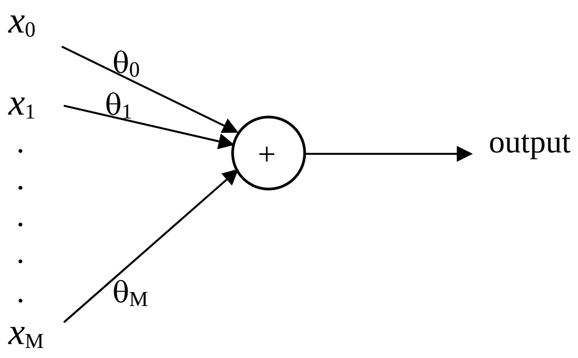
Linear Regression

Linear regression can be represented by a functional form:

$$f(\mathbf{x}; \theta) = \theta_0 x_0 + \theta_1 x_1 + \ldots + \theta_M x_M = \sum_{j=0}^{M} \theta_j x_j$$

Note: x_0 is a **dummy attribute** and its value is a constant equal to 1.

Linear regression can also be represented in a graphic form:



Goal: Minimize Mean Square Error (MSE):

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i;\theta))^2$$

$\Rightarrow$ MSE is a quadratic function in parameters θ
$\Rightarrow$ It is a convex function
$\Rightarrow$ There is only one minimum, it is the **global minimum**

Solution: Sufficient condition is $\frac{\partial MSE}{\partial \theta_j} = 0$, $\forall \theta_j$, $j = 0, 1, \ldots, M$.

Therefore, find θ_j such that

$$\frac{\partial MSE}{\partial \theta_j} = -\frac{2}{N}\sum_i \left(y_i - \sum_k \theta_k x_{ik} \right) \cdot x_{ij} = 0, \ \forall j$$

There are M+1 linear equations with M+1 unknown variables $\Rightarrow$ we can get a closed-form solution.

Special Case: If some attribute is a linear combination of others, there is no unique solution.

$$\frac{\partial MSE}{\partial \theta j} = 0 \Rightarrow \sum_{i=1}^{N} y_i x_{ij} = \sum_{i=1}^{N}\sum_{k=0}^{M} \theta_k x_{ik} x_{ij} \Rightarrow (\textit{in matrix form})\ X^T Y = X^T X \theta,$$

where:

$X_{[N\times(M+1)]} = \{x_{ij}\}_{i=1:N,\, j=1:(M+1)}$, ($x_{ij}$ is jth attribute of ith data point)
$Y_{[N\times 1]} = \{y_i\}_{i=1:N}$,

$\theta_{[(M+1)\times 1]} = \{\theta_j\}_{j=1:(M+1)}$

Note: D = [X Y], i.e., [X Y] is what we defined previously as the data set.

The **optimal parameter choice** is then:

$\theta = (X^T X)^{-1} X^T Y$, which is a closed form solution.

Note: the above solution exists if $X^T X$ is invertible, i.e. if its rank equals M+1, i.e. no attribute is a linear combination of others (in Matlab, use function *rank*).

Note: using matrix derivations we can do the optimization in a more elegant way by defining

$$MSE = \frac{1}{N}(Y - X\theta)^T (Y - X\theta) \Rightarrow \nabla_\theta MSE = -2X^T(Y - X\theta) = \mathbf{0}_{[(M+1)\times 1]}$$
$$\Rightarrow \theta = (X^T X)^{-1} X^T Y$$

Statistical results:

Assumption: the true **data generating process** (DGP) is

$y = \sum_{j=0}^{M} \beta_j x_j + e$, e is noise with $E(e) = 0$, $Var(e) = \sigma^2$

Note: This is a big assumption!

Questions: How close is the estimate θ to the true value β?

Answer 1:

$$E[\theta] = E[(X^T X)^{-1} X^T Y] = (X^T X)^{-1} X^T E[Y] \quad \text{(remember, } Y = X\beta + \mathbf{e}_{[Nx1]})$$
$$\Rightarrow \quad E[\theta] = (X^T X)^{-1} X^T X\beta + (X^T X)^{-1} X^T E[e]$$
$$\Rightarrow \quad E[\theta] = \beta + 0 = \beta$$

Conclusion: if we repeat linear regression on different data sets sampled according to the true DGP, the average θ will equal β (i.e., $E[\theta] = \beta$), which are the true parameters. Therefore, the linear regression is an **unbiased predictor**.

Answer 2: The variance of parameter estimate θ is

$$Var[\theta] = \ldots\text{(after some calculation)}\ldots = (X^T X)^{-1}\sigma^2$$

Conclusion: $Var[\theta]$ is a measure of how different estimation θ is from the true parameters β, i.e. how successful is the linear regression. Therefore, quality of linear regression depends on the **noise level** (i.e. σ^2) and on the **data size**. The variance increases linearly with σ^2 and decreases as $1/N^2$ with the size of the dataset N.

More stringent assumption: the true DGP is

$y = \sum_{j=0}^{M} \beta_j x_j + e$, and $e \sim N(0, \sigma^2)$ (i.e., e is **Gaussian additive noise**)

If the assumption is valid we could: Estimate θ can be considered as a multi-dimensional Gaussian variable with $\theta = N(\beta, (X^T X)^{-1}\sigma^2)$. Therefore, we could do some nice thing such as test the hypothesis that $\beta_j = 0$ (i.e. that attribute j is not influencing the target y).

Nonlinear Regression

Question: What if we know that $f(\mathbf{x};\theta)$ is a non-linear parametric function?
For example: $f(\mathbf{x};\theta) = \theta_0 + \theta_1 x_1 x_2^{\theta 2}$, this is a function nonlinear in parameters.

Solution: Minimize MSE = $\frac{1}{N}\sum (y_i - f(\mathbf{x}_i;\boldsymbol{\theta}))^2$

Start from the necessary condition for minimum:

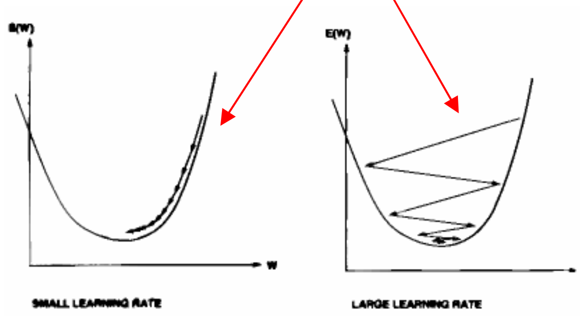
$\Rightarrow$ $$\frac{\partial MSE}{\partial \theta_j} = -\frac{2}{N}\sum (y_i - f(\mathbf{x}_i;\boldsymbol{\theta}))\frac{\partial f(\mathbf{x}_i;\boldsymbol{\theta})}{\partial \theta_j} = 0$$

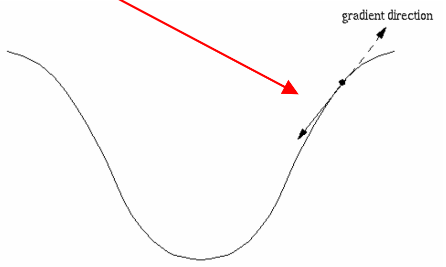
$\Rightarrow$ Again, we have to solve M nonlinear equations with M unknowns.
$\Rightarrow$ But, this time closed-form solution is not easy to derive.

Math Background: Unconstrained Optimization:

Problem: Given $f(\mathbf{x})$, find its minimum.
Popular Solution: Use the **gradient descent** algorithm.
Idea: The gradient of $f(\mathbf{x})$ at the minimum is zero vector. So,

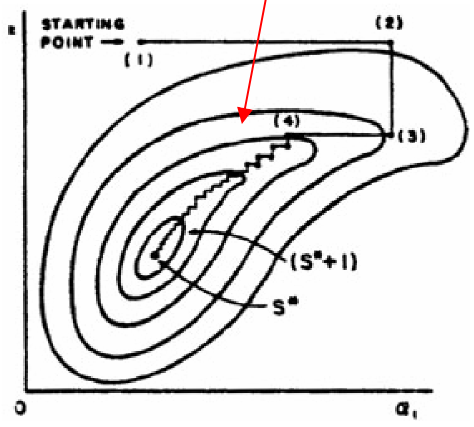
1. start from an initial guess $\mathbf{x}^0$;
2. calculate gradient $\nabla f(\mathbf{x}^0)$;
3. move in the direction opposite of the gradient, i.e., generate new guess $\mathbf{x}^1$ as $\mathbf{x}^1 = \mathbf{x}^0 - \alpha\nabla f(\mathbf{x}^0)$, where α is a properly selected constant;
4. repeat this process until convergence to the minimum.

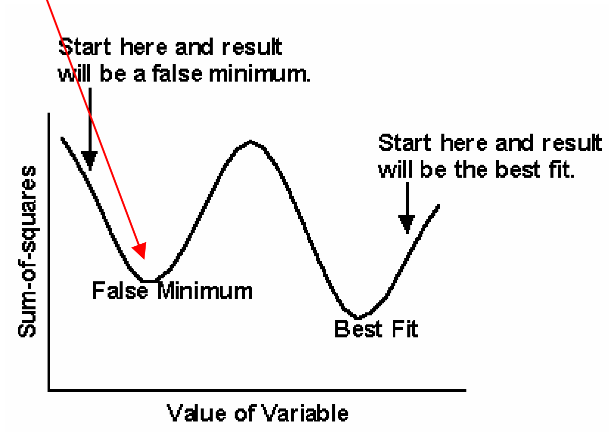




Two problems with gradient descent algorithm:

1. It accepts convergence to a **local minimum**. The simplest solution to avoid the local minimum is to repeat the procedure starting from multiple initial guesses $\mathbf{x}^0$.
2. Possible **slow convergence** to a minimum. There are a number of algorithms providing faster convergence (e.g. conjugate gradient; second order methods such as Newton or quazi-Newton; nonderivative methods)





Back to solving nonlinear regression using gradient descent procedure:

Step 1: Start from an initial guess for parameters θ^0.

Step k: Update the parameters as $\theta^{k+1} = \theta^k - \alpha\nabla f(\theta^k)$

Special Case: For linear prediction the update step would be $\theta^{k+1} = \theta^k + 2\alpha X^T(Y - X\theta^k)$

Logistic Regression – Classification by MSE Minimization

Remember: Classification can be solved by MSE minimization methods (E[y|**x**] can be used to derive posteriors $P(y \in C_j | \mathbf{x})$).

Question: What functional form $f(\mathbf{x};\theta)$ can be an appropriate choice for representing posterior class probabilities?

Option 1: What about linear model $f(\mathbf{x};\theta) = \sum_{j=0}^{M} \theta_j x_j$? The range of the function goes beyond 0-1, so it is not a good choice.

Option 2: We can use sigmoid function to do squeeze the output of a linear model to the range between 0 and 1: $f(\mathbf{x};\theta) = g(\sum_{j=0}^{M} \theta_j x_j)$. If $g(z) = e^{-z}/(1+e^{-z})$, optimizing $f(\mathbf{x};\theta)$ is called **logistic regression**.

Solution: Logistic regression can be solved by minimizing MSE. Derivative $\partial MSE/\partial\theta_j$ is

$$\frac{\partial MSE}{\partial \theta_j} = -\frac{2}{N}\sum_{i=1}^{N}(y - f(\mathbf{x};\boldsymbol{\theta}))x_{ij}\, g'\left(\sum_{k=0}^{M}\theta_k x_{ik}\right)$$

Note: Solving $\nabla_\theta MSE = 0$ results in (M+1) nonlinear equations with (M+1) unknowns $\Rightarrow$ optimization can be done by using gradient descent algorithm.

Maximum Likelihood (ML) Algorithm

Basic Idea: Given a data set D and a parametric model with parameters θ that describes the data generating process, the best solution θ^* is the one that maximizes $P(D|\theta)$, i.e.

$$\theta^* = \arg\max_\theta P(D|\theta)$$

$P(D|\theta)$ is called the **likelihood**, so the name of the algorithm that finds the optimal solution θ^* is called the **maximum likelihood algorithm**. This idea can be applied for both unsupervised and supervised learning problems.

ML for Unsupervised Learning: Density Estimation

Given D = $\{\mathbf{x}_i, i=1, 2, \ldots N\}$, and assuming the functional form $p(x|\theta)$ of the data generating process, the goal is to estimate the optimal parameters θ that maximize likelihood $P(D|\theta)$:

$$P(D|\theta) = P(x_1, x_2, \ldots, x_N|\theta)$$

By assuming that data points $\mathbf{x}_i$ are independent and identically distributed (*iid*)

$$P(D|\theta) = \prod_{i=1}^{N} p(\mathbf{x}_i \mid \boldsymbol{\theta}) \quad (p \text{ is the probability density function.})$$

Since log(x) is monotonically increasing function with x, maximization of $P(D|\theta)$ is equivalent to maximization of $l = \log(P(D|\theta))$. l is called the log-likelihood. So,

$$l = \sum_{i=1}^{N} \log\left(p(\mathbf{x}_i \mid \boldsymbol{\theta})\right)$$

Example: Data set D = $\{\mathbf{x}_i, i=1, 2, \ldots N\}$ is drawn from a Gaussian distribution with mean μ and standard deviation σ, i.e., $X \sim N(\mu,\sigma^2)$. Therefore,

$$p(x_i \mid \mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \Rightarrow l = \sum_{i=1}^{N} \left(\log\frac{1}{\sqrt{2\pi}\sigma} - \frac{(x-\mu)^2}{2\sigma^2} \right)$$

Values μ and σ that maximize the log-likelihood satisfy the necessary condition for local optimum:

$$\frac{\partial l}{\partial \mu} = 0 \Rightarrow \hat{\mu} = \tfrac{1}{N} \sum_{i=1}^{N} x_i\ , \quad \frac{\partial l}{\partial \sigma} = 0 \Rightarrow \hat{\sigma} = \tfrac{1}{N} \sum_{i=1}^{N} (x_i - \hat{\mu})^2$$

ML for Supervised Learning

Given D = $\{(\mathbf{x}_i, y_i), i=1, 2, \ldots N\}$, and assuming the functional form $p(y|\mathbf{x},\theta)$ of the data generating process, the goal is to estimate the optimal parameters θ that maximize likelihood $P(D|\theta)$:

$$P(D|\theta) = P(y_1, y_2, \ldots, y_N|x_1, x_2, \ldots, x_N,\theta) = \quad \text{/if data is } iid$$

$$= \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_i, \boldsymbol{\theta})$$

ML for Regression

Assume the data generating process corresponds to:

$y = f(\mathbf{x}, \boldsymbol{\theta}) + e$, where $e \sim \mathrm{N}(\mu, \sigma^2)$

Note: this is a relatively strong assumption!

$$\Rightarrow\ y \sim N(f(\mathbf{x}, \boldsymbol{\theta}), \sigma^2)$$

$$\Rightarrow\ p(y \mid \mathbf{x}, \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x - f(\mathbf{x}, \boldsymbol{\theta}))^2}{2\sigma^2}}$$

$$\Rightarrow\ l = \log P(D \mid \boldsymbol{\theta}) = \sum_{i=1}^{N} \left(\log \frac{1}{\sqrt{2\pi}\sigma} - \frac{(y_i - f(\mathbf{x}_i, \boldsymbol{\theta}))^2}{2\sigma^2} \right)$$

Since σ is a constant, maximization of l is equivalent to minimization of $\frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i, \boldsymbol{\theta}))^2$

Important conclusion: **Regression using ML** under the assumption of DGP with additive Gaussian noise is equivalent to **regression using MSE minimization**!!

ML for Classification

There are two main approaches to classification involving ML: the Bayesian Estimation approach, and logistic regression.

Bayesian Estimation

Idea: given a dataset *D*, decompose *D* into datasets $\{D_j\}$, $j = 1, \ldots, k$ = # of classes, where $\bigcup D_j = D$ and $D_i \cap D_j = \varnothing$ for all *i*, *j*. For each D_j, we can estimate the pdf $p(\mathbf{x} \mid y \in C_j)$ (the class-conditional density). These densities can be estimated using the ML methods described in Lecture 3, provided we make a (strong) assumption about the functional form of the density (e.g., Gaussian). We also note that this approach is useful theoretically and when the input dimension is low, but density estimation is generally not practical in high dimensions.

In order to obtain a classifier, we want to be able to estimate the probabilities $p(y \in C_j \mid \mathbf{x})$ (the posterior class probabilities). A new input will be assigned to the class with the highest estimated posterior class probability. We can estimate these probabilities by applying Bayes' Theorem:

Bayes Theorem: if A & B are events, then

$$P(A \mid B) = \frac{P(B \mid A) P(A)}{P(B)}$$

So we see that

$$p(y \in C_j \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid y \in C_j) p(y \in C_j)}{p(\mathbf{x})}$$

where $p(y \in C_j)$ (the prior class probability) may be estimated without reference to the inputs as the relative frequency of C_j in the dataset *D*. For the purpose of classification, it is not necessary to compute

$p(\mathbf{x})$, since we are interested only in the relative sizes of the posterior probabilities. Finally, we may define the Bayesian classifier by

$$\hat{y} = \arg\max_{C_k} \ p(y \in C_j \mid \mathbf{x}) = \arg\max_{C_k} \ p(\mathbf{x} \mid y \in C_j)\, p(y \in C_j)$$

We reiterate that this method is only practically applicable in low dimensions, and requires strong assumptions about the functional form of the class distributions.

Logistic Regression

The assumptions involved in logistic regression are similar to those involved with linear regression, namely the existence of a linear relationship between the inputs and the output. In the case of logistic regression, this assumption takes a somewhat different form: we assume that the posterior class probabilities can be estimated as a linear function of the inputs, passed through a sigmoidal function. Parameter estimates (coefficients of the inputs) are then calculated to minimize MSE. For simplicity, assume we are doing binary classification and that $y \in \{0,1\}$. Then the logistic regression model is

$$\mu = p(y \in C_1 \mid \mathbf{x}) = \frac{e^{\eta}}{1+e^{\eta}} \qquad \text{where} \quad \eta = \sum_j \theta_j x_j$$

The likelihood function of the data D is given by

$$p(D \mid \mathbf{\Theta}) = \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_i, \mathbf{\Theta}) = \prod_{i=1}^{N} \mu_i^{y_i} (1-\mu_i)^{1-y_i}$$

Note that the term $\mu_i^{y_i}(1-\mu_i)^{1-y_i}$ reduces to the posterior class probability of class 0 when $y_i = 0$, and the posterior class probability of class 1 otherwise, so this expression makes sense. In order to find the ML estimators of the parameters, we form the log-likelihood

$$\ell = \log p(D \mid \mathbf{\Theta}) = \sum_{i=1}^{N} [y_i \log \mu_i + (1-y_i)\log(1-\mu_i)]$$

The ML estimators require us to solve $\nabla_{\mathbf{\Theta}} \ell = 0$, which is a non-linear system of (M+1) equations in (M+1) unknowns, so we don't expect a closed form solution. Hence we would, for instance, apply the gradient descent algorithm to get the parameter estimates for the classifier.