Solve non-linear problems by (approximate) reduction to linear ones

basic idea: if $Y = X \cdot \beta$ (linear model) is not good enough, add more features until it works, e.g. by non-linear functions of X

e.g. $X = [X_1, X_2, X_3]$ (3 features originally)

$$\tilde{X} = [X_1, X_2, X_3, X_1^2, X_1 \cdot X_2, \log(X_3), \sqrt{X_2} \cdot \exp(X_3), \dots]$$

$\Rightarrow$ $Y = \tilde{X} \cdot \tilde{\beta}$ depends non-linearly on X, despite being a linear fct.

improve by also transforming Y: $\tilde{Y} = \Psi(Y, X)$ $\quad \tilde{X} = \varphi(X)$

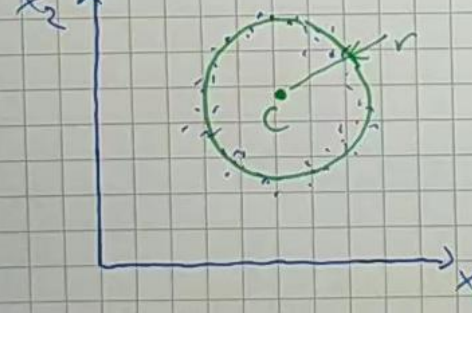
solve $\tilde{Y} = \tilde{X} \cdot \hat{\beta}$

predict: $Y = \Psi^{-1}(\tilde{X} \cdot \hat{\beta}, X)$

↳ Ψ must be invertible in Y

Problem: coming up with good coordinate transforms $\tilde{Y} = \Psi(Y, X)$, $\tilde{X} = \varphi(X)$ is difficult. Which functions to choose is a "hyperparameter"

Example where it works: circle fitting: 3 parameters: $\beta = [c_1, c_2, r]^T$



residual: distance of a point from the circle

$$\delta_i(\beta) = \sqrt{(X_{i1} - c_1)^2 + (X_{i2} - c_2)^2} - r = \epsilon \approx 0$$

$$\Rightarrow \boxed{\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{N} \delta_i(X_i)^2}$$ no analytic solution

$\Rightarrow$ use Levenberg-Marquardt algorithm alternative solution using coordinate transforms

Euclidean residual

if point X_i is on the circle, $\sqrt{(x_{i1}-c_1)^2+(x_{i2}-c_2)^2} - r = 0$

inside the circle < 0

outside the circle > 0

likewise: X_i is on the circle $(x_{i1}-c_1)^2 + (x_{i2}-c_2)^2 - r^2 = 0$

inside < 0

algebraic residual

outside > 0

$\Rightarrow$ find a coordinate transform that measure residuals with the second form (without square root)

$$\tilde{X}_i = [\, x_{i1},\ x_{i2},\ 1\,] \quad = \varphi(X)$$

$$\tilde{Y}_i = x_{i1}^2 + x_{i2}^2 \quad = \psi(\cancel{Y}, X)$$

$$\tilde{\beta} = [\, 2c_1,\ 2c_2,\ r^2 - c_1^2 - c_2^2\,]^T$$

$$= [\, 2\beta_1,\ 2\beta_2,\ \beta_3^2 - \beta_1^2 - \beta_2^2\,]^T \qquad (\beta = [c_1, c_2, r])$$

$$\Rightarrow \tilde{Y}_i - \tilde{X}_i \cdot \tilde{\beta} = (x_{i1} - c_1)^2 + (x_{i2}-c_2)^2 - r^2$$

$$\hat{\tilde{\beta}} = \arg\min_{\tilde{\beta}} \sum_{i=1}^{N} (\tilde{Y}_i - \tilde{X}_i \cdot \tilde{\beta})^2$$

algebraic loss

solve as OLS

reconstruct original β: $\hat{c}_1 = \frac{\hat{\tilde{\beta}}_1}{2}$ $\hat{c}_2 = \frac{\hat{\tilde{\beta}}_2}{2}$ $r = \sqrt{\hat{\tilde{\beta}}_3^2 + \hat{c}_1^2 + \hat{c}_2^2}$

when points are close to circle, both methods give similar results, algebraic fails for part of circles

Non-linear least squares in an augmented feature space

idea: $Y = f(X)$ is non-linear (hard to optimize), find a coordinate transform $\tilde{X} = \varphi(X)$ such that the relationship becomes (approximately) linear: $Y = \tilde{X} \cdot \beta$ => easy to optimize

[sometimes, it helps to transform Y as well: $\tilde{Y} = \psi(Y, X)$, can get desired response by inversion: $Y = \psi^{-1}(\tilde{Y}, X)$, e.g. circle fit]

problem: finding a suitable transformation is hard

Q: Is there a systematic way of doing it? Universal recipe?

A: Yes. Kernel ridge regression, kernel regression

Intuition: if the optimization does not ~~...~~ access $\tilde{X}_i$ directly, but only via scalar products $\tilde{X}_i \tilde{X}_{i'}^T = G_{ii'}$, then we can apply the kernel trick; compute $G_{ii'}$ via an analytic formula ("kernel function") without first computing $\tilde{X}_i$, $\tilde{X}_{i'}$

Matrix of scalar products: Gram matrix, Gramian $\quad G = X X^T \in \mathbb{R}^{N \times N}$

$[\, G = \tilde{X} \tilde{X}^T \,]$

[note: don't confuse with scatter matrix $S = X^T X \in \mathbb{R}^{D \times D}$]

Normal ridge regression does not support the kernel trick:

$$\hat{\beta} = \arg\min_{\beta} (Y - X\beta)^2 + \tau \beta^T \beta \Rightarrow \hat{\beta} = \underbrace{(X^T X + \tau \mathbb{I})^{-1} X^T}_{\text{regularized pseudo-inverse}} Y$$

because we need access to the feature vectors X_i

rewrite ~~rig~~ ridge regression in terms of its <u>dual optimization problem</u>

dual solution vector $\hat{\alpha}$: $$\boxed{\hat{\alpha} = \arg\max_{\alpha} \; -\alpha^T X X^T \alpha - \tau \alpha^T \alpha + 2\alpha^T Y}$$

$$\hat{\alpha} = (\underbrace{X X^T}_{= G} + \tau \mathbb{I})^{-1} Y$$

$\Rightarrow$ access data only via Gram matrix
$\Rightarrow$ kernels can be applied

relation to the primal problem:

$$\hat{\beta} = X^T \hat{\alpha} \quad , \quad \hat{Y}_{test} = X_{test} \cdot \hat{\beta} = \underbrace{X_{test} X^T}_{g} \hat{\alpha}$$

proof: homework

g: scalar products between test and training features

$g_i = X_{test} \cdot X_i^T$

disadvantage of dual formulation: need to store training set also for testing, so that $g = X_{test} X^T$ can be computed (not necessary in primal ridge regression: once $\hat{\beta}$ is known, TS is no longer needed)

<u>derivation of the dual problem</u>

(1) introduce new variable r_i for the residuals: $r_i = Y_i - X_i \beta$ for all i

$\Rightarrow$ primal problem $\hat{\beta} = \arg\min_{\beta} r^T r + \tau \beta^T \beta \quad$ s.t. $\forall i: r_i = Y_i - X_i \beta$

$\Rightarrow$ Langrangian $(\beta, r, \gamma) = r^T r + \tau \beta^T \beta + \sum_{i=1}^{N} \gamma_i (Y_i - X_i \beta - r_i)$

now, it is mathematically more convenient, to factor out 2τ from the Lagrangian multipliers $\Rightarrow$ introduce new multipliers α_i, s.t. $\gamma_i = 2\tau\alpha_i$

[always: $\gamma_i \geq 0$, $\alpha_i \geq 0$] $\Rightarrow$ modified Lagrangian

$$\text{Lagrangian}(\beta, r, \alpha) = r^T r + \tau \beta^T \beta + 2\tau \sum_i \alpha_i (y_i - x_i \beta - r_i)$$

(2) solve for β:

$$\frac{\partial \text{Lagrang.}}{\partial \beta} = 2\tau\beta - 2\tau \sum_{i=1}^{n} \alpha_i x_i^T \overset{!}{=} 0$$

$$\boxed{\hat{\beta} = \sum_i \alpha_i x_i^T = \alpha X^T}$$

$\Rightarrow$ eliminate β from the Lagrangian

$$\text{Lagrangian}(r, \alpha) = r^T r + \tau \alpha^T X X^T \alpha + 2\tau \alpha^T y - 2\tau \alpha^T X^T X \alpha - 2\tau \alpha^T r$$

$$= r^T r - \tau \alpha^T X X^T \alpha + 2\tau \alpha^T y - 2\tau \alpha^T r$$

(3) solve for r:

$$\frac{\partial \text{Lagrang}}{\partial r} = 2 r^T - 2\tau \alpha^T \overset{!}{=} 0 \qquad \boxed{r = \tau \alpha}$$

$\Rightarrow$ eliminate r from Lagrangian:

$$\text{Lagrangian}(\alpha) = \tau^2 \alpha^T \alpha - \tau \alpha^T X X^T \alpha + 2\tau \alpha^T y - 2\tau^2 \alpha^T \alpha$$

$$= -\tau \alpha^T X X^T \alpha + 2\tau \alpha^T y - \tau^2 \alpha^T \alpha$$

$\Rightarrow$ dual optimization problem

$$\boxed{\hat{\alpha} = \arg\max_{\alpha} -\alpha^T X X^T \alpha + 2\alpha^T y - \tau \alpha^T \alpha}$$

(4) solve dual for α: $\frac{\partial \text{Lagrang.}}{\partial \alpha} = -2 X X^T \alpha - 2\tau\alpha + 2Y \overset{!}{=} 0$

$$\boxed{\alpha = (X X^T + \tau \mathbb{I})^{-1} Y}$$

$= (G + \tau \mathbb{I})^{-1} \cdot Y \Rightarrow$ can apply kernel trick

<u>Example</u> for a coordinate transform where the kernel trick works

given: 2-D feature space $X_i = [X_{i1}, X_{i2}]$

augmented feature space: $\tilde{X}_i = \varphi(X_i) \in \mathbb{R}^6$

$= [\sqrt{2} X_{i1}, \sqrt{2} X_{i2}, X_{i1}^2, X_{i2}^2, \sqrt{2} X_{i1} \cdot X_{i2}, 1]$

dot product: $\tilde{X}_i \cdot \tilde{X}_{i'} = 2 X_{i1} \cdot X_{i'1} + 2 X_{i2} \cdot X_{i'2} + X_{i1}^2 \cdot X_{i'1}^2 + X_{i2}^2 \cdot X_{i'2}^2$

$+ 2 X_{i1} \cdot X_{i2} \cdot X_{i'1} \cdot X_{i'2} + 1$

$= ((X_i X_{i'}^T) + 1)^2$

Gram matrix in augmented space: $G = \tilde{X} \cdot \tilde{X}^T$

with $G_{ii'} = (X_i X_{i'}^T + 1)^2 = k(X_i, X_{i'})$

$\Rightarrow$ can calculate $G_{ii'} = \tilde{X}_i \cdot \tilde{X}_{i'}^T$ <u>without</u> computing $\tilde{X}_i$ and $\tilde{X}_{i'}$

a formula with this property is called <u>kernel function</u>: $G_{ii'} = k(X_i, X_{i'})$

<u>Mercer condition</u>: a formula $k(X_i, X_{i'})$ is a valid kernel function (i.e. is identical to a scalar product in some augmented feature space), when for all square-integrable function $f(X)$ it holds: $\int_{X_i} \int_{X_{i'}} k(X_i, X_{i'}) f(X_i) \cdot f(X_{i'}) \, dX_{i'} \, dX_i \geq 0$

Intuition: for any TS X, the Gram matrix $G_{ii'} = k(X_i, X_{i'})$ is positive semi-definite

⇒ when Mercer's condition is fulfilled, the inverse $(G + \tau \mathbb{I})^{-1}$ exists for <u>every</u> training set ($\hat{\alpha} = (G + \tau \mathbb{I})^{-1} y$ exists)

• most popular kernel functions

– polynomial kernels: $k(x_i, x_{i'}) = (x_i x_{i'}^T + 1)^p$ $\quad p \geq 1$

(the example from previous page used $p = 2$)

– squared exponential kernel $k(x_i, x_{i'}) = \exp\left(-\frac{\|x_i - x_{i'}\|^2}{2h^2}\right)$

h: bandwidth of kernel ≙ how the distance $\|x_i - x_{i'}\|$ is scaled

– p or h (and τ from ridge regression) are hyper-parameters that the user must adjust (e.g. cross-validation)

– long list of permissible kernel function by asking Google

– more general treatment: Gaussian processes in Advanced Machine Learning

• heuristic approximation: <u>kernel regression</u> (Nadaraya-Watson regression)

$$\hat{y}_{new} = \hat{f}(x_{new}) = \frac{\sum_{i=1}^{N} y_i \cdot g_i}{\sum_{i=1}^{N} g_i} \qquad g_i \hat{=} k(x_{new}, x_i)$$

weighted sum weights

weight normalization $\quad g_i' = \frac{g_i}{\sum g_i}$

complexity: $O(N \cdot D)$

• in practice (≙ squared-exponential kernel), $g_i \approx 0$ when $\|x_i - x_{new}\| > 3h$

⇒ only the near-neighbors of x_{new} contribute to the sum

⇒ efficient algorithms to sum only over near neighbors (instead of all N training points)

e.g. "fast Gaussian transform"

- prediction of new instance with dual ridge regression.
  - linear case: $\hat{y} = X_{new} X^T \hat{\alpha}$ $\Rightarrow$ pre-compute $\hat{\beta}^T = X^T \alpha$
    $= X_{new} \hat{\beta}$ $\Rightarrow$ do not need to store $\hat{X}$ (TS)
  - non-linear case (with kernel trick): $\hat{y} = \underbrace{k(X_{new}, X)}_{= g} \hat{\alpha}$ with $g_i = k(X_{new}, x_i)$

    $\Rightarrow$ g must be computed again for every x_{new}, pre-computing $X^T \hat{\alpha}$ does not help, because $k(\cdot,\cdot)$ is non-linear, and we cannot factor out $X^T \hat{\alpha}$ as in the linear case $\Rightarrow$ need to store training set
- when kernel trick was invented in the 1990ies, explicitly computing high-dimensional $\tilde{X}$ was impractical (too expensive in time and memory)
  $\Rightarrow$ kernel avoids this
  today, we have GPUs and lots of RAM $\Rightarrow$ we can explicitly compute complex $\tilde{X} = \varphi(X)$, for example many neural nets do this:
  network with L layers: - first $L-1$ layers compute $\tilde{X} = \varphi(X)$
  - last layer computes $\hat{y} = \tilde{X} \cdot \beta$

  penultimate neuron activations form an augmented feature space: "latent representation"
- kernel trick can also be used to make linear classifiers (LDA, LR) non-linear $\Rightarrow$ <u>support vector machine</u> (SVM) $\Rightarrow$ later