Regression

José Dorronsoro
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Contents

1 Machine Learning Modeling Basics

What Is Machine Learning (ML)?

- Lofty definition: make machines learn!!!
- Have to precise "machines" and "learn"
- The machines of ML: mathematical input–output processes that lend themselves to some form of (numerical) parameterization
- The learning process: adjust the machine's parameters until a goal is reached
- New thing: "goal"?
  - At first sight, get something done
  - Ultimately, to minimize some error measure
- Summing things up: a ML process tries to find a concrete mathematical/algorithmic **input–output parameterized transformation** that **minimizes an error measure** by iteratively **adjusting the transformation's parameters**

Where Lies ML?

- In the middle of a possibly long process chain
- Before ML starts we must
  - Go from **raw to organized** data: accesing, gathering, cleaning, formatting, ...
  - Go from **organized to** (potentially) **informative** data: extracting basic and derived features
- After ML finishes we must perform
  - Outcome **evaluation**: how good/actionable it is
  - Outcome **exploitation**: collect, organize, act
  - **Individual model maintenance**: monitor performance, tune hyper–parameters
  - **Modeling life cycle maintenance**: discard old models, introduce new ones ize and **communicate** our work/results
- ML is in the middle of the global process chain but also in the middle of some subchains

Supervised/Unsupervised Models

- Model types: **supervised, unsupervised**
- Supervised models:
  - Targets y^p are known and the model tries to predict or estimate them
  - These known targets guide, or **supervise**, model building
  - Main emphasis here

- Unsupervised models:
  - There are no predetermined or supervising outputs
  - But nevertheless the model is supposed to learn relations or find structure in the data
  - Sometimes as a first step towards a supervised model

Regression and Classification

- Problems (usually) to be solved by models: regression, classification
- Patterns come in pairs (x, y)
  - x: inputs, predictors, features, independent variables
  - y: target, response, dependent variable; numerical in regression, class labels in classification
- **Regression**: the desired output y is regressed into the inputs x to derive a model $\widehat{y} = f(x)$
  - We want $y \simeq \widehat{y}$ so having $y - \widehat{y}$ "small" is the natural goal
- **Classification**: inputs are derived from several classes $C_1, \ldots, C_K$, to which labels ℓ_k are assigned
  - The model now assigns a label $\ell(x)$ to an input x
  - If x is derived from C_k we want to have $\ell(x) = \ell_k$
  - Here having $\ell(x) - \ell_k$ "small" may not make sense

The Boston Housing Problem

- This is a first "toy" problem
- We want to estimate the median of house values over an area from some information about it which we believe relevant
- Features x: several real estate–related variables of Boston areas
  - CRIM: per capita crime rate by town
  - RM: average number of rooms per dwelling
  - NOX: nitric oxides concentration (parts per 10 million)
  - AGE: proportion of owner-occupied units built prior to 1940
  - LSTAT: % lower status of the population
  - ...
- Target y: MEDV, median value of owner-occupied homes in $1,000's

Wind Energy Forecasting

- This is a second, real regression problem
- We want to estimate the hourly energy production of a wind farm from NWP variables which we believe relevant

- The **features** are the NWP variables
  - U, V surface wind components
  - U, V 100–meter wind components
  - Temperature
  - Pressure
  - ...
- The **target** is the energy produced during the outgoing hour

The ML Cycle in Wind Energy

- Raw data: historic wind energy production data plus NWP files from weather forecasters
  - Possibly huge files with special formats
  - We have to extract the relevant NWP information, organize them in a suitable way and pair it with the energy data
- The ML core: whatever set of (non–linear) regression algorithm which you may think useful
- After ML is finished
  - Collect, organize and save the different model outputs
  - Select one single model output or some combination (more ML) of them as your system's output
  - Compute uncertainty estimates
  - Combine your outputs with someone's else
  - And keep up the entire process

How to Build Regression Models

- In general we have a sample $S = \{x^p, y^p\}$, $1 \leq p \leq N$, with x^p the **features** and y^p the **targets**
- We want to build a model $\widehat{y} = f(x)$ so that $\widehat{y}^p = f(x^p) \simeq y^p$; i.e., we want to **regress** y to the x
- The concrete f is chosen within a certain family $\mathcal{F}$
  - Examples here: linear regression, multilayer perceptrons (MLPs), SVMs
  - Other useful models: Random Forests (RF), Gradient Boosting (GB), nearest neighbor (NN)
- Natural option to ensure $f(x^p) \simeq y^p$: choose f to minimize the sample mean square error (MSE)

$$\widehat{e}(f) = \widehat{e}_S(f) = \frac{1}{2N} \sum_{p=1}^{N} (y^p - f(x^p))^2$$

- Thus, the model we select is $\widehat{f} = \widehat{f}_S = \arg\min_{f \in \mathcal{F}} \widehat{e}_S(f)$

Model Parameterization

- Usually individual models are selected through (ideally optimal) **parameter sets**
  - The parameters (weights) $W \in R^M$ select a concrete f in $\mathcal{F}$
- **Parametric** models have a fixed functional form $f(x) = f(x; W)$
- Simplest example: linear regression, where $M = d$ and $W = (w_0, w)$

$$f(x; w_0, w) = w_0 + \sum_{j=1}^{d} w_j x_j = w_0 + w \cdot x$$

- **Semi–parametric** models also use weights but without a predefined functional form; MLPs but also RF or GBR
- **Non parametric** models do not use weights nor follow any broad functional form; NN models

Model Estimation as Error Minimization

- For a parametric or semiparametric $f(x; W)$ we can write $\widehat{e}_S(f) = \widehat{e}_S(W)$
- The problem to solve becomes

$$\widehat{W}^* = \widehat{W}^*_S = \arg\min_W \widehat{e}_S(f(\cdot; W)), \text{ i.e., } \widehat{e}_S(\widehat{W}^*) \leq \widehat{e}_S(W)\ \forall W$$

- In linear regression

$$\widehat{e}(w_0, w) = \frac{1}{2N} \sum_p (y^p - w_0 - w \cdot x^p)^2$$

  which ends up in a simple quadratic form
- The regression problem reduces to **minimize** $\widehat{e}_S(W)$
  - Something in principle well understood in mathematical optimization

2 Basic Regression

Regression Assumptions

- **Key assumption**: x and y are related as $y = \phi(x) + n$ where
  - $\phi(x)$ is the **true** underlying function
  - n is **additive noise** with 0 mean and finite variance σ^2_N
- Our sample is just a particular instance of a deeper **sample generation process**
- Thus x, n are produced by **random variables** X, N
  - And so is y, given by $Y = \phi(X) + N$
- Moreover, X and N are **independent distributions** with densities $q(x)$, $\nu(n)$

- Thus, X and Y (or X and N) have a joint density

$$p(x, y) = p(x, \phi(x) + n) = q(x)\ \nu(n) = q(x)\ \nu(y - \phi(x))$$

but with independent components

Linear Models

- The simplest ones but an indispensable first step
- Assuming $x \in R^d$, the basic linear model is

$$f(x) = w_0 + \sum_1^d w_i x_i = w_0 + w \cdot x$$

- w_0 complicates notation; to drop it we center x and y so that $E[x_i] = E[y] = 0$; then $w_0 = 0$
- Then we are left with the simpler homogeneous model $f(x) = w \cdot x$
- In practice we will always **normalize** x, for instance to have 0 mean and 1 standard deviation (std) on each feature
- But: how do we find w?

1–dimensional Linear Regression (LR)

- Assume that features X and target Y are **centered**, i.e., have 0 means
- For 1-dimensional patterns x the LR model then becomes

$$f(x) = w \cdot x$$

- And the error is then the function $e(w)$

$$\widehat{e}(w) = \frac{1}{2N} \sum_{p=1}^N (w \cdot x^p - y^p)^2 = \frac{1}{2N} \sum_p (r^p)^2$$

with $r^p = w \cdot x^p - y^p$ the **residual** of the p–th element

- The problem has obviously a minimum w^*
- To find it we just solve $\widehat{e}'(w) = 0$

Solving $\widehat{e}'(w) = 0$

- To compute $\widehat{e}'(w)$ we have

$$\begin{aligned} \widehat{e}'(w) &= \frac{1}{2N} \sum_p x^p r^p = \frac{1}{2N} \sum_p \left(w (x^p)^2 - x^p y^p \right) \\ &= w \left(\frac{1}{2N} \sum_p (x^p)^2 \right) - \frac{1}{2N} \sum_p x^p y^p \end{aligned}$$

- The optimal w^* solves $\widehat{e}'(w) = 0$ and is given by

$$w^* = \frac{\frac{1}{2N}\sum_p x^p y^p}{\frac{1}{2N}\sum_p (x^p)^2} \quad = \quad \frac{\frac{1}{2N} X \cdot Y}{\frac{1}{2N} X \cdot X} = \frac{\frac{1}{2N} X \cdot Y}{\text{var}(x)}$$

where X and Y denote the N dimensional vectors $(x^1, \ldots, x^N)^t$, $(y^1, \ldots, y^N)^t$

General Linear Regression

- Assume again that X and Y are centered
- The LR model becomes now $f(x) = \sum_1^d w_i x_i = w \cdot x$
- If Y is the $N \times 1$ **target** vector and we organize the sample S in a $N \times d$ **data matrix** X, the sample mse is given by

$$\begin{aligned} \widehat{e}(w) &= \frac{1}{2N}\sum_p (w \cdot x^p - y^p)^2 = \frac{1}{2N}(Xw - Y)^t (Xw - Y) \\ &= \frac{1}{2N}(w^t X^t X w - 2 w^t X^t Y + Y^t Y) \end{aligned}$$

- Now we have to solve $\nabla \widehat{e}(w) = 0$, i.e., $\frac{\partial \widehat{e}}{\partial w_i}(w) = 0$
- It is easy to see that

$$\nabla \widehat{e}(w) = \frac{1}{N} X^t X w - \frac{1}{N} X^t Y = \widehat{R} w - \widehat{b}$$

Solving the Linear Equations

- The optimal $\widehat{w}^*$ must verify $\nabla \widehat{e}(\widehat{w}) = \widehat{R}\,\widehat{w} - \widehat{b} = 0$, where

$$\widehat{R} = \frac{1}{N} X^t X, \;\; \widehat{b} = \frac{1}{N} X^t Y$$

- Over the original, non–centered data matrix we have

$$\widehat{R} = \frac{1}{N}(X - \overline{X})^t (X - \overline{X});$$

i.e., $\widehat{R}$ is the **sample covariance matrix**

- If $\widehat{R}$ is invertible, we just solve the linear system $\widehat{R}\,\widehat{w} - \widehat{b} = 0$
- And obtain the sample–dependent $\widehat{w}^*$ as

$$\widehat{w}^* = \widehat{R}^{-1}\widehat{b} = \left(X^t X\right)^{-1} X^t Y$$

Finding Optimal Models

- For general regression models it is usually not possible to solve analytically the equation $\nabla \widehat{e}(W) = 0$

  - For LR and big data, covariance matrices over large datasets or dimensions may not be computed
  - Numerical methods are needed

- The simplest numerical alternative is **gradient descent**:
  - Starting from some random W^0 we iteratively compute

$$W^{k+1} = W^k - \rho_k \nabla \widehat{e}(W^k)$$

  - Component wise: $w_i^{k+1} = w_i^k - \rho_k \frac{\partial \widehat{e}}{\partial w_i}(W^k)$
  - ρ_k is the **learning rate**

- If $W^k \to W^*$, then $\nabla \widehat{e}(W^*) = 0$
  - Since our problems have obviously minima, this should be enough

Measuring Model Fit

- First option: **Root Square Error** $RSE = \sqrt{\frac{1}{N}\sum(y^p - \widehat{y}^p)^2} = \sqrt{\frac{1}{N}RSS}$
- OK, but how good is this? We must always have a **base model** to benchmark our results
- Simplest "model": the mean $\overline{y} = \frac{1}{N}\sum_1^N y^p$, with square error

$$\frac{1}{N}\sum(y^p - \overline{y})^2 = \frac{1}{N}TSS = \text{Var}(y)$$

- We can compare our model against our base computing

$$\frac{RSE^2}{\text{Var}(y)} = \frac{\sum(y^p - \widehat{y}^p)^2}{\sum(y^p - \overline{y})^2} = \frac{RSS}{TSS}$$

- The widely used R^2 coefficient is simply $R^2 = 1 - \frac{RSS}{TSS}$

Evaluating Expected Performance

- It is obvious that before we start applying a model, we should have a reasonably accurate idea of its performance in practice
  - I.e., we want to estimate the model's **generalization performance**
  - Estimating the generalization performance **only over the sample S used for training results in misleading error values**

- We apply Cross Validation (CV) by
  - Randomly splitting the sample S in M subsets $S_1, \ldots, S_M$
  - Working with M **folds**: pairs (S_m, S_m^c), with

$$S_m^c = S - S_m = \cup_{i \neq m} S_i$$

  - Building M different models **using the S_m^c as training subsets**

  - Computing their errors e_m on the folds' **validation subsets** S_m
  - Using these errors' average as a first estimate of the true model performance

Regularization

- Our regression solution $\widehat{w}^* = \left(X^t X\right)^{-1} X^t Y$ won't work if $X^t X$ is not invertible
  - For instance, when some features are correlated
- We fix this working instead with $X^t X + \alpha I$ for some $\alpha > 0$
- Then $\widehat{w}^* = \left(X^t X + \alpha I\right)^{-1} X^t Y$ minimizes

$$e_R(w) = \frac{1}{2N} \sum_p (y^p - w \cdot x_p^p)^2 + \frac{\alpha}{2} \|w\|^2,$$

- This is the **Ridge Regression** problem
  - Our first example of **regularization**, a key technique in Machine Learning
  - **All ML models must be regularized in some way**
- We find the optimal α by CV

Grid Hyper–parameter Selection

- Build M **folds**: pairs (S_m, S_m^c) and use S_m^c as training and S_m as the validation subsets
- Fix a hyper–parameter range $[0, A]$
  - $\alpha = 0$: no penalty and, thus, small bias and high variance
  - $\alpha = A$: large penalty and, thus, small variance but high bias
- Select an $L + 1$ point **grid**

$$G = \left\{0, \frac{A}{L}, \frac{2A}{L}, \ldots, \frac{\ell A}{L}, \ldots, \frac{LA}{L} = A\right\}$$

- At each $\alpha_\ell = \frac{\ell}{L} A$, $0 \leq \ell \leq L$
  - Train M models on the S_m^c using the hyper–parameter α_ℓ
  - Average their M validation errors e_m on the S_m to get the error $e(\alpha_\ell)$ at α_ℓ
- Finally choose the (hopefully) optimal hyper–parameter α^* as

$$\alpha^* = \text{arg min}_{0 \leq \ell \leq L} \; e(\alpha_\ell)$$

3 Non–Linear Regression Models

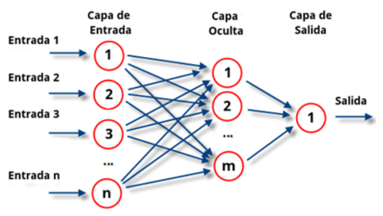
Going Beyond Linearity

- LR is simple, elegant and illuminating, but not too powerful
  - Linear transformations almost never are
- Solution: go for (highly) non–linear transformations $f(x; w)$
- We shall briefly explore some such approaches
  - Multilayer Perceptrons (MLPs, a.k.a. Neural Nets)
  - Deep Neural Nets (DNN): MLPs on steroids
  - Support Vector Machines (SVMs)
  - Regression Tree–based methods: Random Forests, Gradient Boosting
  - Model–free models (??): k–Nearest Neighbor (k–NN) regression

3.1 Multilayer Perceptrons

MLP Architecture

- General layout:
  - An input layer (input)
  - One or several hidden layers
  - One output layer
- Feedforward connections only



- Overall process: $f(x; W)$ with W the set of connecting weights and biases

MLPs for Regression

- MLPs fit nicely in our regression scenario
- Given a sample $S = \{(x^p, y^p)\}$, we define a suitable MLP $f(x; W)$, with W the MLP weight and bias set

- We select the optimal W^* minimizing the sample MSE

$$\widehat{e}(W) = \widehat{e}_S(W) = \frac{1}{2N}\sum_{p=1}^{N}(y^p - f(x^p; W))^2$$

- $f(x; W)$ is highly non–linear and $\widehat{e}_S(W)$ more so
- Thus we must use numerical optimization for which we need to compute $\nabla_W \widehat{e}_S(W)$

The Simplest Regression MLP

- It is a Single Hidden Layer (SHL) MLP
  - D inputs (determined by the problem at hand)
  - One hidden layer with H units (number to be chosen) and a concrete **activation** σ (sigmoid, $\tanh$, ReLU)
  - One linear output
- While the simplest possible MLP, it provides quite powerful regression models
- Usually enough for many applications

Activation Functions

- Choices for f:
  - Heaviside (in Rosenblatt's Perceptrons): $f(a) = 0$ if $a \leq 0$, $f(a) = 1$ if $a > 0$
  - Identity: $f(a) = a$
  - Sigmoid:
$$f(a) = \frac{1}{1 + e^{-a}}$$
  - Hyperbolic tangent:
$$f(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$
  - Rectified Linear Units (ReLUs): $r(x) = \max(0, x)$

The SHL MLP Process

- Input–hidden processing: if O are the hidden unit outputs

$$o^h = \sigma\left(w^H_{h0} + \sum_{j=1}^{D} w^H_{hj} x_j\right)$$

- Hidden–output processing: we have for the outputs y

$$y = w^O_0 + \sum_{h=1}^{H} w^O_h o_h,$$

- Global process:

$$y = f(x; W^O, W^H) = w_0^O + \sum_h w_h^O \sigma \left(w_{h0}^H + \sum_j w_{hj}^H x_j \right)$$

MLPs and Universal Approximation

- We say that $\mathcal{F} = \{f(X; W)\}$ is a **Universal Approximation Family** over a domain $\mathcal{R}$ if

  For any $\epsilon > 0$ and any reasonable ϕ, we can find a weight set $W_{\phi,\epsilon}$

$$\int \|\phi(x) - f(x; W_{\phi,\epsilon})\|^2 p(x) dx \leq \epsilon$$

- Notice that Universal Approximation is just what we need in regression
- In fact a **Single Hidden Layer MLP with enough hidden units is an effective universal approximator**
- But we have to be able to build them

MLP Training

- Usually done by some form of **gradient descent** over **minibatches**
  - **Small random** subsets of the entire sample
- Gradients are computed through the **backpropagation** algorithm
- Gradient iterations start from a **random** weight set W_0
  - Different initial W_0 result in different final MLPs
  - Which one to use? If possible, train several and average their predictions
- In general, MLP training is quite costly
  - For a SHL MLP each iteration has a cost of $O(N \times d \times n_H)$, with N sample size, d dimension, n_H the number of hidden units
  - And more if we want several MLPs (although this falls into the embarrassingly parallel category)

MLP Regularization

- MLPs are powerful approximators so we risk overfitting
- Regularization is **mandatory** so we actually minimize

$$\widehat{e}_R(W) = \widehat{e}(W) + \frac{\alpha}{2}\|W\|^2$$

- The new gradient is simply $\nabla\widehat{e}_R(W) = \nabla\widehat{e}(W) + \alpha W$

- We must carefully choose an adequate α (plus the number of hidden units, plus the learning rate, plus . . .)
- Hyper–parameter selection tools (plus careful hyper–parameter analysis) are indispensable

MLP Ensembles

- Recall that $e(W)$ does not have a single minimum
- Moreover, the final MLP depends on the random initial W^0
- And mini–batch training adds extra randomness to the final model
- This suggests
  - To start from M independent initial weights and get M optimal weight sets W_m^*
  - To output the average $f_e(X) = \frac{1}{M}\sum_1^M f(x; W_m^*)$
- We expect outputs of the form $\widehat{y}_k^p = y^p + \epsilon_k^p$ with the ϵ_k^p independent
- Hence $\frac{1}{M}\sum_m \epsilon_m^p \simeq 0$ and $\frac{1}{M}\sum_m \widehat{y}_m^p \simeq y^p$

Summing Things Up

1. MLPs improve on linear regression by using highly non–linear models
2. MLPs are **universal approximators**
   - Just what we need for regression, but high overfitting risks
3. Regularization is crucial
4. The definition of MLPs requires to decide on an architecture
   - Can be done by CV but this may be very costly
5. MLP training usually done by **gradient descent** (or some variant) over **minibatches**
6. Gradients are computed using the **backpropagation** algorithm
7. Training large MLPs is costly
8. We can exploit randomness in MLP training working with MLP ensembles

Deep Neural Nets

- There was a very intense academic interest in the (by now) standard MLPs in the 1990's
  - Several NN conferences and journals appear
- MLP working and training became well understood
  - Although losing much of neuronal plausibility
- MLPs found relevant applications in many fields

    - They were incorporated into data science tools and products
    - Although hyper–parameter selection was (is) costly and had (has) to be done very carefully

NN's Golden Autumn?

- This went on strongly until the late 90's when
    - New relevant contributions decreased
    - New competitors appeared: Boosting, SVMs, Random Forests, Gradient Boosting Regression, ...
- A nagging issue were deeper MLPs
    - One hidden layer MLPs were enough for most applications at the time
    - But nobody knew how to train MLPs with three or more hidden layer

The Deep Neural Network Boom

- Breakthroughs by G. Hinton and Y. Bengio around 2007 rekindled the interest in NNs
- Around 2010 the floodgates opened:
    - Large nets with huge number of weights
    - New convolutional layers, regularizations, initializations or activations
    - New techniques appear ... that were not that different from the old ones
- **New mood**: what was impossible before is now much easier and leads to better results
- Major breakthroughs were achieved in significant problems in computer vision and speech recognition
    - Applied in autonomous vehicles, medical diagnosis, speech transcription, machine translation

What Is New In DNNs?

- New and fancy network structures:
    - Convolutional layers (with non–differentiable components)
    - More flexible feedforward connections
- **Automated symbolic backprop derivation**
- Network size: large number of layers and huge number of weights
- Very large sample size (sometimes)
- New cost functions and new ways to combine them
- New (non differentiable) activations: ReLUs
- New regularization: dropout, dropconnect
- Recognition that a good weight initialization is critical

3.2 Support Vector Regression

Support Vector Regression

- In SV regression (SVR) we begin with a linear model and try to minimize another regularized error function

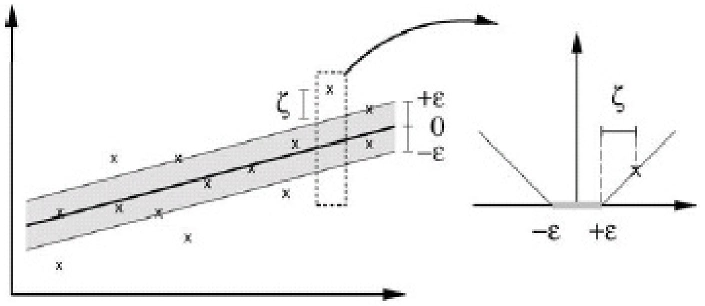
$$\sum_p \left[y^p - (w \cdot x^p + w_0)\right]_\epsilon + \frac{\alpha}{2}\|w\|^2$$

where $[z]_\epsilon = \max(0, |z| - \epsilon)$ is the ϵ–**insensitive** loss:

- This is a convex optimization problem with a **unique solution**
- Notice that an error $|y^p - f(x^p; w, w_0)|$ is only penalized if it is $> \epsilon$
- Thus, we keep the regularization of Ridge regression but introduce a new (non–differentiable) penalty

The ϵ Error Tube

- We only penalize errors that fall **outside an ϵ–wide tube** around the true function



- So far we have a possibly non–powerful linear model
  - We must introduce some form of non–linearity

Solving the SVR Minimization Problem

- SVR's penalty function is not differentiable
  - In principle gradient descent would be problematic
- Standard approach:
  - Write it as a **constrained optimization problem**
  - Apply tools of the general theory for such problems to write an equivalent but simpler **dual** problem

  - Solve the dual problem

- While everything is so far linear, analyzing the dual problem will suggest **how to develop non linear SVR**

SVR as a Constrained Problem

- We have $f(w, w_0) = \ell_\epsilon(w, w_0) + \frac{\alpha}{2}\|w\|^2$
  - f is convex but $\ell_\epsilon = \sum_p \left[y^p - (w \cdot x^p + w_0)\right]_\epsilon$ is not smooth
  - Direct minimization of $f(w, w_0)$ is difficult, so we re–formulate the unconstrained SVR problem as a constrained one

- If $C = 1/\alpha$, we rewrite f as the **primal problem**

$$f(w, w_0, \xi, \eta) = \frac{1}{2}\|w\|^2 + C\sum_p(\xi_p + \eta_p)$$

subject to the following constraints on the errors $w \cdot x^p + w_0 - y^p$:

$$\begin{aligned}
-\xi_p - \epsilon &\leq w \cdot x^p + w_0 - y^p, \\
\eta_p + \epsilon &\geq w \cdot x^p + w_0 - y^p, \\
\xi_p, \eta_p &\geq 0
\end{aligned}$$

The SVR Lagrangian

- We put together the error function and the constrains in the **Lagrangian**

$$\begin{aligned}
L(w, w_0, \xi, \eta, \alpha, \beta, \gamma, \delta) &= \frac{1}{2}\|w\|^2 + C\sum_p(\xi_p + \eta_p) \\
&- \sum_p \alpha_p(w \cdot x^p + w_0 - y^p + \xi_p + \epsilon) \\
&+ \sum_q \beta_q(w \cdot x^q + w_0 - y^q - \eta_q - \epsilon) - \sum_p \gamma_p \xi_p - \sum_q \delta_q \eta_q
\end{aligned}$$

with $\alpha_p, \beta_q, \gamma_r, \delta_s$ all ≥ 0

- Notice that we have by construction

$$L(w, w_0, \xi, \eta, \alpha, \beta, \gamma, \delta) \leq f(w, w_0, \xi, \eta)$$

SVR's Dual Problem

- We define the dual function as

$$\Theta(\alpha, \beta, \gamma, \delta) = \min_{w, w_0, \xi, \eta} L(w, w_0, \xi, \eta, \alpha, \beta, \gamma, \delta)$$

- Thus, again by construction, we have

$$\Theta(\alpha, \beta, \gamma, \delta) \leq L(w, w_0, \xi, \eta, \alpha, \beta, \gamma, \delta) \leq f(w, w_0, \xi, \eta)$$

- And the dual problem is

$$\max_{\alpha,\beta,\gamma,\delta \geq 0} \Theta(\alpha, \beta, \gamma, \delta)$$

- At first sight the constraints **are now much simpler**
- If Θ is simple enough, we will end up with a dual problem simpler than the primal
  - This will be the case but new constraints will be added to the previous simple ones

SVR's Dual Function

- We derive the dual function solving the equations

$$\frac{\partial L}{\partial w_i} = 0, \ \frac{\partial L}{\partial w_0} = 0, \ \frac{\partial L}{\partial \xi_p} = 0, \ \frac{\partial L}{\partial \eta_p} = 0$$

- Plugging the results back in L, the dual function Θ becomes

$$\begin{aligned} \Theta(\alpha, \beta) &= -\frac{1}{2} \sum_{p,q} (\alpha_p - \beta_p)(\alpha_q - \beta_q) x^p \cdot x^q - \\ &\quad \epsilon \sum_p (\alpha_p + \beta_p) + \sum_p y^p (\alpha_p - \beta_p) \end{aligned}$$

  which is (minus) a semidefinite positive quadratic form

- And the final constraints become $0 \leq \alpha_p, \beta_q \leq C$, $\sum \alpha_p = \sum \beta_q$
- γ and δ disappear but the new constraint $\sum \alpha_p = \sum \beta_q$ is tricky

The Primal, the Lagrangian and the Dual

- Summing things up, we started with SVR's **primal problem** but we much prefer to solve the **dual problem** provided **both solutions are equivalent**
- This indeed so: if $w^*, w_0^*, \xi^*, \eta^*$ and α^*, β^* are the primal and dual solutions respectively, we have $\Theta(\alpha^*, \beta^*) = f(w^*, w_0^*, \xi^*, \eta^*)$
- Moreover, once we know the optimal α^*, β^* we have $w^* = \sum_p (\alpha_p^* - \beta_p^*) x^p$
- And the Lagrangian $L(w^*, w_0^*, \xi^*, \eta^*, \alpha^*, \beta^*)$ gets squeezed in the middle
- In particular, if $0 < \alpha_p^*, \beta_q^* < C$,

$$\begin{aligned} 0 &= \alpha_p^* (w^* \cdot x^p + w_0^* - y^p + \epsilon), \\ 0 &= \beta_q^* (w^* \cdot x^q + w_0^* - y^q - \epsilon) \end{aligned}$$

  from which we can derive w_0^*

The Kernel Trick for SVR

- Summing things up, the optimal dual solutions α^*, β^* enable us to recover the optimal primal solutions w^*, w_0^*
- It turns out that we only need to compute dot products among patterns to apply the model

$$f(x; w^*, w_0^*) = w_0^* + w^* \cdot x = w_0^* + \sum_p (\alpha_p^* - \beta_p^*) x^p \cdot x$$

- Moreover, we just also need to compute dot products $x^p \cdot x^q$ to set up and solve the dual problem
- We can extend this replacing $x^p \cdot x^q$ by $k(x^p, x^q)$ where k is an appropriate **kernel**
- Standard choice: the Gaussian kernel $k(x, x') = e^{-\gamma \|x - x'\|^2}$
- This leads to a very powerful non–linear **Gaussian SVR** model $f_G(x) = w_0^* + \sum_p (\alpha_p^* - \beta_p^*) e^{-\gamma \|x - x^p\|^2}$

Gaussian SVR Hyper–parameters

- The linear SVR problem requires to fine tune two hyperparameters: the penalty $C = \frac{1}{\alpha}$ and the tube's width ϵ
- Gaussian SVR adds the kernel's width γ
- As usual, optimal hyper–parameters are selected by CV
  - Useful values for ϵ can be obtained as $2^{-k}\sigma_y$, with σ_y the standard deviation of the target y
  - Gaussian SVR inputs are usually scaled to a $[-1, 1]$ range, which suggests to consider γ values of the form $\frac{2^k}{d}$ with $-K_L \leq k \leq K_R$ and d pattern's dimension
  - Customary values for C are $0.1, 1, 10, \ldots, 10,000$
- But SVR training is costly and we have to work in a 3–dimensional grid ...
- SVR hyper–parameterization requires both patience and computing power!!

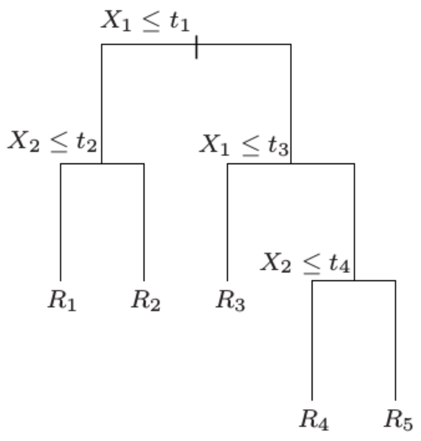
3.3 RFR, GBR and k–NN Regression

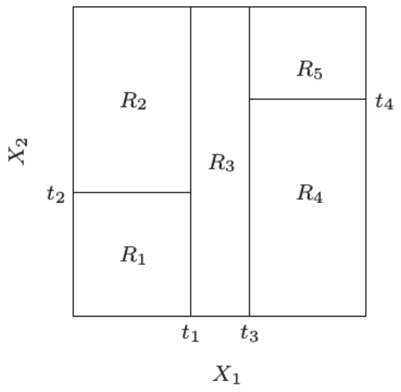
Decision Trees

- Decision trees are built following a sample split procedure according to most relevant variables
- Variables and split values are selected according to some **gain criterion**

Regression Trees

- These splits divide feature space into rectangular regions
- In regression trees a **single prediction value** is assigned to each region R





- Usual choice: the average of y^p over the samples $x^p \in R$

Random Forest Regression

- Single trees may not yield good models
- In Random Forest Regression
  - Many trees T_k are **randomly** built (the forest)
  - The final model is their average $\mathcal{T}_M(x) = \frac{1}{M}\sum_1^M T_k(x)$
- The forest‘s trees must also be built **independently**
- Randomness and independence are achieved by
  - Randomly and independently select a subsample when building each tree
  - Randomly and independently select a subset of features for each split
- Pros: good, easy to build models which allow some interpretation of variable relevance; can handle categorical features

- Cons: random final model, several hyper–parameters to be tuned

Gradient Boosting Regression

- A forest is also built with random trees but each new tree tries to **reduce the error of the combined previous trees**
- Very different from Random Forests
  - In RFR each tree is built the same way with the same targets
  - In GBR each tree amends the error of the previous ensemble and has a new specific target
- Moreover the outcomes of a new tree G_k are **shrunk** when it is added: $\mathcal{G}_M(x) = \epsilon G_M(x) + \mathcal{G}_{M-1}(x)$
- Good selection of ϵ and M are crucial

k–Nearest Neighbor Regression

- Sometimes the relation between features x and targets y doesn't justify a strong model $y = \phi(x) + n$
- k–NN Regression relies on a reasonable assumption: **Predictors that are close should give predictions that are also close**
- In k–NN Regression we fix a number k of neighbors to be considered and for an input x set

$$\hat{y} = Y_{kNN}(x) = \frac{1}{k} \sum_{x^p \in N_k(x)} y^p$$

  where $N_k(x)$ denotes the k sample points closest to x
- **Weighted variants**: for instance, $Y^w_{kNN}(x) = \frac{1}{C_k(x)} \sum_{x^p \in N_k(x)} \frac{1}{\|x^p - x\|} y^p$
  - $C_k(x) = \sum_{x^p \in N_k(x)} \frac{1}{\|x^p - x\|}$ is a normalizing constant

4 Modeling in Practice

4.1 The Modeling Cycle

Before Modeling Starts

- Two key tasks before modeling can start
  - Data capture
  - Data cleaning and organization
- Can be long, hard and costly
- Even more so in Big Data
- Always needed but we leave out of this discussion

- Our set up here: supervised modeling
- Starting point: we have received a sample

$$S = \{(x^1, y^1), \dots, (x^N, y^N)\}$$

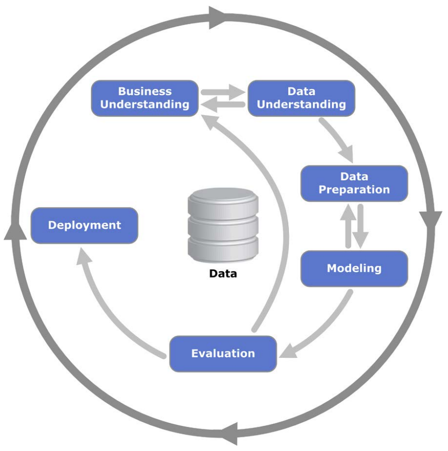
from which we build the $N \times d$ **data matrix** $X = (X_{pi}) = (x_i^p)$ and the N–th dimensional **target vector** $Y = (Y_p) = (y^p)$

Modeling Cycle

- Modeling goal: to build a model $\hat{y} = f(x; W^*)$ such that $f(x^p; W^*) \simeq y^p$ for all p
- Cycle phases:
  1. Data visualization
  2. Feature and target analysis
  3. Definition of training, validation and test sets or folds
  4. Selection of the (first) models to use
  5. Selection of optimal model hyperparameters
  6. Definitive (for the time being!) model building (training)
  7. Starting model application and real use data collection
  8. Follow up of the model in exploitation, analysis of its results and, if needed,
  9. Start all over again, often from step 4

CRISP–DM

- The latest formalizatiof of this is the **Cross Industry Standard Process for Data Mining** (CRISP–DM)



From Wikipedia

Data Exploration

- Individual feature visualization and analysis
  - Simple feature graphics, feature–target graphics
  - Histograms, boxplots, scatterplots
  - Outlier identification
  - And whatever else we figure out that may help
- Correlations
  - Are there strong correlations among features? Are some of them redundant?
  - Have features a predictive value? Are there correlated with the target?
- And, again, whatever else we figure out that may help

Model and Hyperparameter Selection

- There are more or less clear starting elections for medium size problems
  - Linear (linear, logistic regression) models, as a first reference and as a follow up of data analysis
  - SVC, SVR
  - Batch MLPs, always with weight decay, second order training and averaging different MLPs
  - Random Forests, Gradient Boosting Regression
- For large size problems
  - Minibatch MLPs, specialized SVC/SVR using Dual Coordinate Descent (DCD) or online SVMs: Pegasos, ...
  - Dimensionality reduction using Principal Components if $N \gg d$, or sparse regression (Lasso, Elastic Net) if $d \gg N$
- Use CV for model evaluation and hyperparameter selection
- Adapt CV to the problem at hand (for instance if it has a time structure)

Train and test

- Keep training information: error evolution at least, partial models if possible, . . .
  - Training is usually costly: better stop it if it doesn't advance
  - Also, retain intermediate models just in case some incidence happens
- Over test data
  - Measure error measures: **MSE or MAE in regression**, accuracy, recall, precision in classification

  - Analyze **error distribution** in regression, posterior class probabilities in classification
  - Detect and correct biases and systematic errors

- If possible, visualize features and errors

How to Make All This?

- The whole process requires a lot of work but also has a partially repetitive nature: it fits in a **data flow** scenario
- The best model is not clear beforehand: we need to have as many implementations available as possible
- That was very difficult, say, about the year 2000
- Now it is much easier, as we have many tools

4.2 Tools for ML

Overview

- Reference tools
  - Commercial: SAS Enterprise Miner, IBM's SPSS, Matlab
  - Academic: Matlab/Octave, Weka, R, **Python libraries**
- SAS Enterprise Miner, IBM's SPSS
  - Full range business inteligence software with several machine learning techniques
  - Needs professional installation and support
- Weka, Knime, RapidMiner: easy to use, good for a first try (and even a second one!!), perhaps not for application development
- R, Matlab: excellent packages in most areas of statistics and engineering, respectively
  - Very strong on RFR, GBR; rather weak in MLPs
  - For SVC/SVR all give wrappers around LIBSVM or LIBLINEAR
  - Elementary script languages under the hood; not for product development

Python

- Lingua franca of Big Data (Hadoop) and (increasingly) Data Science and Machine Learning
- Over–simple characterization: object–oriented pseudo C with tons of good libraries
- More seriously: simple, easy to learn **general purpose** language
  - Succinct and $\pm$ clean basic code with indentation–based blocks
  - Handy for data preparation and model exploitation
- Very strong community

  - But also many strong critics
- Many environments and IDEs
- Very strong support network: stackoverflow, tutorials, MOOCS, ...

Python for ML

- Best tool for exploratory data analysis, initial model building and simple programming: IPython QT Console
- Best tool for data flow and analysis description: IPython Notebooks
- A lot of very good libraries for a very wide range of applications
- Numerical and Data Science libraries: Numpy, Scipy, Statsmodels, Matplotlib
- Machine Learning: Scikit–learn (sklearn)
- Nice overview in The Python Ecosystem for Data Science

Scikit–learn

- Reference library for ML in Python (and probably the best overall)
- Implements all the approaches mentioned above (only partially for deep networks)
  - Keras + TensorFlow are the DNN reference
- Model work follows a common **define–fit–predict** approach
- Sklearn has also tools for data pre–processing, cross validation–based model performance estimation and hyper–parameter selection
- Can be put together under a pipeline framework
- A first step toward automated ML?

IPython Notebooks

- Excellent for summarizing data information and model definition, and reporting model results
- Notebooks are organized in **code and text cells**
- Code cells have Python code that can be edited and executed with `Ctrl+Enter`
- Text cells have either raw text or (much better) **markdown** text: headings, lists, other formatting or LaTeXformulae
- Notebooks can be saved as such, as Python code and as html or pdf files
- Main use: tell the story in the data
  - Should have mostly explanatory information
  - Code should be written into modules to be imported

4.3 ML Algorithms and Big Data

What Is Big Data?

- Many definitions but a simple operative one may be: problems whose datasets do not fit in memory of single–server hardware
- Of course, this changes with the hardware we can work with:
  - Desktop PCs: up to 100 GB of RAM
  - Rack–based servers: up to 1 TB of RAM
- Datasets smaller than this can be handled with "standard" computing equipment
- Other considerations aside, larger datasets have to go to the cloud
- Thus, we can take as a first approximation: Big Data = Cloud Data
- Cloud offers data and process parallelization
  - Essentially no restrictions on data sizes plus highly parallel computing structures
  - This somehow decides which algorithms can more easily go to the Cloud

ML and the Cloud: Processes

- Process parallelization is usually simpler
  - In fact, more or less straightforward for single multicore machines
  - But may impose communication costs when working with different nodes
- Some ML algorithms are better suited for process parallelization than others
- Simplest for Random Forest Regression: individual trees are built independently
- Simple for ensembles of MLPs–DNNs or of GBRs
  - Different networks can be trained in parallel
- SVM models are essentially unique, so single sample ensembles do not make sense
- **Parallelization crucial for hyper–parameter selection**
  - Very costly computationally but embarrassingly parallel

ML and the Cloud: Data

- Distributed data handling is trickier
- Again simplest for Random Forest Regression: individual trees are built on independent subsamples
- Harder but possible for single MLP–DNN training:
  - Minibatches for training can be sent to different machines (map)

  - But some coordination is needed (reduce?)
  - More or less the same situation for GBR
- Quite harder for SVM models:
  - Samples have to be much reduced when using dual solvers (LIBSVM)
  - Better situation for primal linear solvers (LIBLINEAR) but effective models are only obtained in large dimension problems
- Ad–hoc cloud tools indispensable
- And Big Data problems usually impose very simple, most likely linear, algorithms

Summing Things Up

- There is no best overall ML model for regression
- Linear regression should be the first model to try on a new problem
  - It sets a first benchmark
- MLPs/DNNs, SVR, RFR, GBR are independent approaches to attack a given problem
  - If possible, they all should be considered
- Cloud tools for ML are becoming available, such as Spark
- But at the end ML models are just tools to be added to a problem's solution:
  - They have to be adequately tuned
  - The problem must be amenable to them and the features used the best possible