Regression and Classification Basics

José R. Dorronsoro
Dpto. de Ingeniería Informática, Escuela Politécnica Superior
Universidad Autónoma de Madrid
28049 Madrid, Spain

Machine Learning Modeling Basics

What Is Machine Learning (ML)?

- Lofty definition: make machines learn!!!
  - Have to make “machines" and “learn" more precise
- 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 and we have a model, we must perform
  - Outcome **evaluation**: how good/actionable the model is
  - Outcome **exploitation**: collect, organize, act
  - **Individual model maintenance**: monitor performance, tune hyper–parameters
  - **Modeling life cycle maintenance**: discard old models, introduce new ones and **communicate** our work/results
- ML is in the middle of the global process chain

Supervised/Unsupervised Models

- First modeling step: get a **sample** $S = \{z^1, \ldots, z^N\}$ with $z^p \in \mathbf{R}^D$ the **patterns** and N the sample size
- Model types: **supervised, unsupervised**
- Supervised models:
  - Patterns $z^p = (x^p, y^p)$, with the y^p being the **targets** that the model tries to predict
  - These known targets guide, or **supervise**, model building
  - Main emphasis here
- Unsupervised models:
  - Patterns $z^p = x^p$ without any known targets
  - 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

How to Build Regression Models

- In general we have a sample $S = \{x^p, y^p\}$, $1 \leq p \leq N$, with $x^p \in \mathbf{R}^d$ 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
  - And also: 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 $M = d + 1$ 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; Nearest Neighbor 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

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

- In linear regression

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

  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

Regression Basics

Linear Models

- Assuming $x \in \mathbf{R}^d$, 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 and, 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 not y if we may help it
- 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

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

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

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

- 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

$$\widehat{e}'(w) \quad = \quad w\left(\frac{1}{N}\sum_p (x^p)^2\right) - \left(\frac{1}{N}\sum_p x^p\, y^p\right)$$

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

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

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

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

- Thus, the optimal $\widehat{w}^*$ must solve the **normal equation** $\widehat{R}\,\widehat{w} - \widehat{b} = 0$, where we recall that

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

- 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 optimal $\widehat{w}^*$ as

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

Alternative: Gradient Descent

- Computing the covariance matrix has a $O(N \times d^2)$ cost and invert it has a $O(d^3)$ cost
  - For big data problems it may not possible to solve analytically the normal equation $\nabla \widehat{e}(w) = 0$
- The simplest numerical alternative is **gradient descent**:
  - Starting from some random w^0 we iteratively compute

$$w^{k+1} = w^k - \rho_k \nabla \widehat{e}(w^k) = w^k - \frac{\rho}{n_B} \left(\widehat{X}_B^t \widehat{X}_B w^k - \widehat{X}_B^t Y \right)$$

    over a **mini-batch** B with n_B samples
  - 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 is enough

Measuring Model Fit

- First option: **Root Square Error** $RSE = \sqrt{\frac{1}{N}\sum(y^p - \widehat{y}^p)^2}$
- OK, but how good is this? We must always have a **base model** to benchmark our results
- Simplest “model": a constant w_0, which yields the mean $\bar{y} = \frac{1}{N}\sum_1^N y^p$, with square error

$$\frac{1}{N}\sum(y^p - \bar{y})^2 = \mathrm{Var}(y)$$

- We can compare our model against this base by computing

$$\frac{\sum(y^p - \widehat{y}^p)^2}{\sum(y^p - \bar{y})^2} = \frac{RSE^2}{\mathrm{Var}(y)}$$

- The widely used R^2 coefficient is simply $R^2 = 1 - \frac{RSE^2}{\mathrm{Var}(y)}$

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 could fix this working instead with $X^t X + \alpha I$ for some $\alpha > 0$
- To make this practical, one can show that $\widehat{w}^* = \left(X^t X + \alpha I\right)^{-1} X^t Y$ 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**
- Important issue: how to find the right choice for α?

Takeaways on Linear Regression

1. We introduced **supervised** models
2. We have reviewed the essentials of the **linear regression model** (always the first thing to try)
3. We have considered model estimation as a problem on **error minimization**
4. We have seen how to build linear models **analytically and numerically**
5. We have defined how to **measure model fit**
6. We have introduced **regularization**

Bias, Variance and Cross Validation

Sample Dependence I

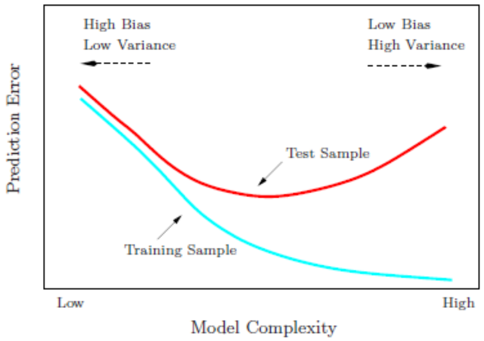
- **Key assumption**: x and y are related as $y = \phi(x) + n$ where
  - $\phi(x)$ is the **true** underlying function
  - n is **additive noise** with 0 mean and finite variance σ_N^2
- We can't do much about the noise n, so we concentrate in getting a model $\widehat{f}$ such that $\widehat{f} \simeq \phi(x)$
- That is, we want our models to have a small **bias**, i.e., get as close as possible to the “true" model ϕ
  - Intuitively this can be achieved using highly flexible models with many parameters
- But **everything is sample dependent**: changing S gives a different model
  - To emphasize this we write $\widehat{f}_S(x)$

Sample Dependence II

- But we also want some model stability with respect to the different samples
- In other words, for different samples S, S' we want our models to verify
$$\widehat{f}_S(x) \simeq \widehat{f}_{S'}(x)$$
- That is, we want our models to have small **variance** with respect to sample changes
  - Intuitively this can be achieved using simple models with few parameters
- But obviously the goals of small bias and small variance are contradictory to a large extent

The Bias–Variance Tradeoff

- There is thus a **tradeoff** between bias (low for complex models) and variance (low for simple models)



Taken from *Hastie et al., p. 38*

Evaluating Expected Performance

- Samples will change anyway: the sample with which we train our model will be different from the one on which we exploit it
  - And before we start applying a model, we should have a reasonably accurate idea of its performance in practice
- To get it over a single S, we apply **Cross Validation** (CV), where we
  - Randomly split the sample S in M subsets $S_1, \ldots, S_M$
  - Work with M **folds**: pairs (S_m, S_m^c), with

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

  - Build M different models **using the S_m^c as training subsets**
  - Compute their errors e_m on the folds' **validation subsets** S_m
  - Use these **errors' average** as a first estimate of the true model performance
- CV can and **must be used** in any model building procedure
- We will also use CV to find an **optimal value** for the hyper–parameter α in Ridge Regression

Grid Hyper–parameter Selection

- Consider for Ridge regression a hyper–parameter range $[0, A]$
  - $\alpha = 0$: no penalty and, thus, small bias but possibly high variance
  - $\alpha = A$: large penalty and, thus, small variance but high bias
- Select an $L + 1$ point **grid** $[\alpha_0, \ldots, \alpha_L]$
  - For instance a uniform one $\alpha_\ell = \ell \, \frac{A}{L}, \ell = 0, 1, \ldots, L$
- Build M **folds**: pairs (S_m, S_m^c) and for each α_ℓ
  - Train M Ridge 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 CV error $e(\alpha_\ell)$ for α_ℓ
- Finally choose the (hopefully) optimal hyper–parameter α^* as

$$\alpha^* = \arg\min_{0 \le \ell \le L} \; e(\alpha_\ell)$$

- α^* gives the model with the **best expected generalization among all possible α choices**

Takeaways on Bias, Variance and CV

1. We have stressed that **any model estimation is sample–dependent** and that this has to be controlled
2. We have introduced the **bias** and **variance** as the two key components of any model error
3. We have discussed **bias–variance trade–off**
4. We have introduced **Cross Validation** here as a tool to estimate a **model's generalization performance**
5. We have also introduced **Cross Validation** as a tool to estimate a **model's hyper–parameters**

Basic Classification

Regression vs Classification

- Recall that in regression we have numerical continuous targets y and want our predictions $\widehat{y}$ to be as close to y as possible
- But in classification we have a finite number of labelled targets for which “selection by closeness" doesn’t make sense
- Natural alternative: select the **most probable label given the pattern** we have just received
  - The concrete labels used for targets do not matter anymore
  - Model learning should thus be “target" agnostic
  - And good probability estimates should be quite useful
- Let’s analyze this in an example

A First Problem: Pima Indian Diabetes

- We want to diagnose whether a person may have diabetes from some clinical measures
- Features x: clinical measures
  - `numPregnant`
  - `bloodPress`
  - `massIndex`
  - `age ...`
- Target y: 0 (no diabetes), 1 (diabetes)
- Clear goal but perhaps too radical
- Better: try to estimate **the probability $P(1|x)$ of having diabetes depending on the features x we measure**

Classification Setup

- We have random patterns ω from M classes, $C_1, \dots C_M$
- Over each pattern we "measure" d features $x = x(\omega) \in \mathbb{R}^d$
  - x inherits the randomness in ω and becomes a **random variable**
- A ω has a **prior probability** π_m of belonging to C_m
- Inside each class C_m there is a **conditional class density** $f(x|m)$ that "controls" the appearance of a given x
- The π_m and $f(x|m)$ determine the **posterior probability** $P(m|x)$ that x comes from class C_m
- **Intuition:** we should assign x to the class with the **largest** $P(m|x)$, that is, work with the classifier

$$\delta(x) = \arg\max_m P(m|x)$$

The Obviously Optimal Classifier

- It can be shown that $P(m|x) = \frac{\pi_m f(x|m)}{f(x)}$
- Thus, we should decide according to a **classifier** function δ_B

$$\begin{aligned}\delta_B(x) &= \arg\max_m P(m|x) = \arg\max_m \frac{\pi_m f(x|m)}{f(x)} \\ &= \arg\max_m \pi_m f(x|m)\end{aligned}$$

- With some extra work we can show that this **Bayes Classifier** δ_B defines an optimal solution (in some precise sense) of the classification problem
- But ... this doesn't look too practical for we do not know either π_m or (much harder) $f(x|m)$
- We will focus first on estimating **directly** the label generating distribution $P(m|x)$

Logistic Regression (LR)

Logistic Regression (LR)

- We assume

$$P(1|x) = P(1|x; w_0, w) = \frac{1}{1 + e^{-(w_0 + w \cdot x)}}$$

  - Then $0 \leq P(1|x) \leq 1$ for any x
- We then have

$$P(0|x) = 1 - P(1|x) = \frac{e^{-(w_0 + w \cdot x)}}{1 + e^{-(w_0 + w \cdot x)}} = \frac{1}{1 + e^{w_0 + w \cdot x}}$$

  - Notice that if $w_0 + w \cdot x = 0$, $P(1|x) = P(0|x) = 0.5$
- The ratio $\frac{P(1|x)}{P(0|x)} = e^{w_0 + w \cdot x}$ is called the **odds** of x and its log the **log odds** or **logit**
- Thus, the basic assumption in LR is that the **logit is a linear function** $w_0 + w \cdot x$ of x
- We have the model $f(x; w)$; we need a **loss** function $L(w_0, w)$ to minimize for which we use the sample's **likelihood**

Estimating w_0^*, w^*

- Assume a single sample x, y and two possible model coefficients w_0, w and w_0', w'
- Denoting by $p = P(y|x; w_0, w)$ and $p' = P(y|x; w_0', w')$, it is clear that we should prefer w_0, w if $p > p'$ and w_0', w' if not
    - In other words, we prefer the coefficients that give a **higher posterior probability** to the sample (x, y)
- For an independent sample $S = \{(x^p, y^p)\}$, its joint probability under a posterior model $p = P(y|x; w_0, w)$ is

$$P(Y|X; w_0, w) = \prod_{p=1}^{N} P(y^p|x^p; w_0, w)$$

- And, again, given two possible model coefficients w_0, w and w_0', w', we should prefer w_0, w iff

$$P(Y|X; w_0, w) > P(Y|X; w_0', w')$$

Sample's Likelihood

- Therefore, we can estimate the optimal w_0^*, w^* as

$$w_0^*, w^* = \arg\max_{w_0, w} P(Y|X; w_0, w)$$

- By the independence assumption we have

$$\begin{aligned} P(Y|X; w_0, w) &= \left\{ \prod_{y^p=1} P(1|x^p) \right\} \left\{ \prod_{y^p=0} P(0|x^p) \right\} \\ &= \prod_{p=1}^{N} P(1|x^p)^{y^p} \; P(0|x^p)^{1-y^p} \end{aligned}$$

with the last equality follows from
  - If $y^p = 1$, $P(1|x^p) = P(1|x^p)^{y^p} = P(1|x^p)^{y^p} \; P(0|x^p)^{1-y^p}$, and
  - If $y^p = 0$, $P(0|x^p) = P(0|x^p)^{1-y^p} = P(1|x^p)^{y^p} \; P(0|x^p)^{1-y^p}$

Max Log–Likelihood Estimation

- The log–likelihood of w_0, w given S is then

$$\begin{aligned}
\ell(w_0, w; S) &= \log P(Y|X; w_0, w) \\
&= \sum_{p=1}^{N} \{ y^p \ \log P(1|x^p) + (1 - y^p) \ \log P(0|x^p) \} \\
&= \sum_p y^p \ \log \frac{P(1|x^p)}{P(0|x^p)} + \sum_p \log P(0|x^p) \\
&= \sum_p y^p \ (w_0 + w \cdot x^p) - \sum_p \log(1 + e^{w_0 + w \cdot x^p})
\end{aligned}$$

- We can thus estimate the optimal $\widehat{w}_0^*, \widehat{w}^*$ as

$$\widehat{w}_0^*, \widehat{w}^* = \text{arg min}_{w_0, w} \ -\ell(w_0, w; S)$$

- Extra bonus: $-\ell$ is a convex differentiable function of (w_0, w) and, thus, it is enough to solve $\nabla \ell(w_0, w) = 0$

Multinomial Logistic Regression

- It is applied to multiclass problems
- Assuming K classes, the posterior probabilities are assumed to be

$$\begin{aligned} P(c|x) &= \frac{e^{w_0^c + w^c \cdot x}}{1 + \sum_1^{K-1} e^{w_0^k + w^k \cdot x}}, \quad 1 \le c \le K-1 \\ P(K|x) &= \frac{1}{1 + \sum_1^{K-1} e^{w_0^k + w^k \cdot x}} \end{aligned}$$

- Given a sample S, we can define a variant $\ell(W_0, W)$ of the log likelihood approach followed in logistic regression
  - Here $W_0 \in \mathbf{R}^K$ and W is a $K \times d$ matrix, with d the x dimension
- Again, we can estimate the optimal $\widehat{W}_0^*, \widehat{W}^*$ as

$$\widehat{W}_0^*, \widehat{W}^* = \arg\min_{W_0, W} \; -\ell(W_0, W; S)$$

  - Again $-\ell$ is a convex differentiable function and we just have to solve $\nabla \ell(W_0, W) = 0$

Newton–Raphson Solution

- However, neither in logistic nor in multinomial regression does $\nabla \ell(\mathcal{W}) = 0$ admit a closed form solution but only an iterative, numerical one
  - Here we write $\mathcal{W}$ for either (w_0, w) or (W_0, W)
- We solve it with the **Newton–Raphson** iterative method (equivalent here to Newton's method for minimization)
- Starting from a random $\mathcal{W}^0$, Newton's iterations are

$$\mathcal{W}^{k+1} = \mathcal{W}^k + (\mathcal{H}_\ell(\mathcal{W}^k))^{-1} \nabla \ell(\mathcal{W}^k)$$

  - $\mathcal{H}_\ell(\mathcal{W}^k)$ denotes the Hessian of ℓ at $\mathcal{W}^k$ (which may or may not be invertible)
  - Everything is fine if the $\mathcal{W}^k$ are close enough to the optimum $\mathcal{W}^*$ but far away things may get tricky
- Just as before, we can add regularization terms $\frac{\alpha}{2}\|w\|^2$ or $\frac{\alpha}{2}\|W\|^2$
- The iterations in Logistic and Multinomial Regression are again typical of many of the model building methods used in Machine

Learning in ML

- The general approach to **learning** is usually the following:
  - A **model** $f(x; W)$ is chosen
  - Given a sample $S = \{(x^1, y^1), \dots, (x^N, y^N)\}$ and a loss function $\ell(y, \widehat{y})$, we define a **sample dependent loss function**

$$L(W) = L(W|S) = \sum \ell\left(y^p, \widehat{y}^p = f(x^p; W)\right)$$

- $L(W)$ is often minimized from some W^0 by **iterations**

$$W^{k+1} = W^k - \rho_k G(W^k, S)$$

  with ρ_k a **learning rate** and G some vectorial function
  - When $G(W) = \nabla L(W)$ we have **gradient descent**
  - When $G(W) = \mathcal{H}(W)^{-1} \nabla L(W)$ we obtain **Newton's method**
- In **batch learning** the entire sample S is used at each iteration
- **On–line** or **minibatch learning**: we use either a single patterns (x^p, y^p) or small subsample
- Several such procedures will appear here in the coming weeks

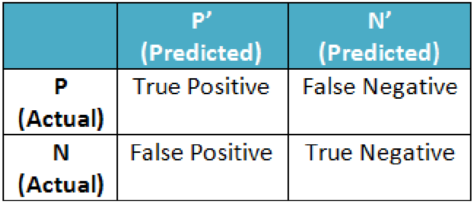
Practical Classification

True/False Positives/Negatives

- Consider a two class problem with labels $y = 0, 1$
- We will call patterns with label 1 **positive** and those with label 0 **negative**
  - Usually the positive patterns are the interesting ones: sick people, defaulted loans, . . .
- Let $\widehat{y} = \widehat{y}(x)$ the label predicted at x; we say that x is a
  - **True Positive** (TP) if $y = \widehat{y} = 1$
  - **True Negative** (TN) if $y = \widehat{y} = 0$
  - **False Positive** (FP) if $y = 0$ but $\widehat{y} = 1$
  - **False Negative** (FN) if $y = 1$ but $\widehat{y} = 0$
- The standard way of presenting these data is through the **confusion matrix**

The Confusion Matrix

- Standard layout



	P' (Predicted)	N' (Predicted)
P (Actual)	True Positive	False Negative
N (Actual)	False Positive	True Negative

- Other layouts:
  - Positives(with label 1) at bottom, as done in `confusion_matrix` of `sklearn`
  - Predicted values in rows, real values in columns

Classifier Metrics

- The classifier **accuracy** is $acc = \frac{TP+TN}{N}$
- acc is the first thing to measure but it may not be too significant: if the number N_0 of negatives is $\gg N_1$, the number of positives
  - The classifier $\delta(x) = 0$ will have a high accuracy $N_0/N \simeq 1$
  - But it will also be useless!!
- First variant in two class problems: Precision, Recall
  - **Recall** or **sensitivity** or **true positive rate** : $TP/(TP + FN)$, i.e., the fraction of positives detected
  - **Precision** or **positive predictive value**: $TP/(TP + FP)$, i.e., the fraction of true alarms issued
  - Also **specificity** or **true negative rate**: $TN/(TN + FP)$,
- Recall measures how many positive cases we recover, i.e., how **effective** is our method
- Precision measures the effort we need for that, i.e., its **efficiency**
- Ideal classifier: high recall, high precision (i.e., effective and efficient!!)

How to Handle Posterior Probabilities

- If possible, we want **posterior probabilities** as model outputs better than labels
- Most models give them as pairs

$$(\widehat{P}(0|x), \widehat{P}(1|x)) = (\widehat{P}(0|x), 1 - \widehat{P}(0|x))$$

- In principle we would decide 1 if $\widehat{P}(1|x) > 0.5$ and viceversa, but this may be too crude
- It may be advisable to set a **decision threshold** κ and decide 1 if $\widehat{P}(1|x) > \kappa$ and 0 if $\widehat{P}(1|x) < \kappa$
- For **imbalanced** problems where $\pi_0 \gg \pi_1$ (usually the interesting ones) we would have $\widehat{P}(1|x) \simeq 0$ for most x
  - In this case we may choose a $\kappa < 0.5$ and **suggest** 1 if $\widehat{P}(1|x) > \kappa$

What's New from Regression?

- Some things change from regression, some don't
- We should check feature correlations, if only to remove too similar features
- Important: **positive and negative–class feature histograms**
  - Scatter plots (x_i, y) are usually less informative
- The **bias–variance trade–off** is subtler in classification
- Accuracy, recall, precision are the usual model quality measures
- We use CV with **stratified folds** to estimate generalization performance
- We also use CV for hyperparameter estimation, as regularization will also be needed
  - In LR we should minimize $-\ell(w_0, w; S) + \frac{\alpha}{2}\|w\|^2$

Takeaways on Basic Classification

1. We have introduced the classification problem as one of computing **posterior probabilities**
2. We have defined the **optimal Bayes classifier**
3. We have introduced **Logistic Regression** to estimate posterior probabilities, and the numerical minimization of its (minus) log–likelihood
4. We have introduced **accuracy, recall, precision** as first classification metrics
    - Accuracy also works for multiclass problems; recall or precision do not extend so well
5. We have reviewed some practical issues in classification

Nearest Neighbor Classification

Approximating the Bayes Classifier

- Starting with the approximation $P(m|x) \simeq P(m|B_r(x))$, Bayes formula gives

$$P(m|x) \simeq P(m|B_r(x)) = \frac{P(C_m \cap B_r(x))}{P(B_r(x))} = \frac{\pi_m P(B_r(x)|m)}{P(B_r(x))}$$

- Fix x and k and assume a N pattern sample with
  - N_m patterns are in C_m
  - $B(x, r)$ the smallest ball containing k sample patterns
  - $B(x, r)$ has k_m patterns from class C_m
- We then have the following approximations

$$\pi_m \simeq \frac{N_m}{N}, \quad P(B_r(x)) \simeq \frac{k}{N}, \quad P(B_r(x)|m) \simeq \frac{k_m}{N_m}$$

The k–NN Classifier

- We can thus approximate $P(m|x)$ as

$$P(m|x) \simeq \frac{\pi_m P(B_r(x)|m)}{P(B_r(x))} \simeq \frac{k_m}{N_m} \frac{N_m}{N} \frac{1}{\frac{k}{N}} = \frac{k_m}{k}$$

- And the optimal δ_B as

$$\delta_B(x) \simeq \text{arg max}_m \, P(m|B_r(x)) = \text{arg max}_m \, \frac{k_m}{k}$$

- We have thus arrived to the k–**Nearest Neighbor** classifier

$$\delta_k^{NN}(x) = \text{arg max}_m \, \frac{k_m}{k} = \text{arg max}_m \, k_m$$

- Thus $\delta_{kNN}(x)$ assigns x to the class that **has more patterns in $N_k(x)$, the subset of the k neighbors that are closest to x**

By the Way: *k*–NN Regression

- *k*–NN Classification assumes that a pattern should belong to the class with the closest patterns
- *k*–NN Regression also relies on a similar 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

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

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

Some k–NN Issues

- **Q1: How do we choose k?** Using CV, of course, and balancing again the bias–variance tradeoff
  - Small variance with large k: if $k = N$, k–NN regression returns the mean
  - Small bias with small k: if $k = 1$ a very close point should give a very close prediction
  - But also large variance: the nearest point to x in another sample may have a quite different target or belong to another class
- **Q2: Is k–NN meaningful?** Yes but only if predictors that are close give predictions that are also close, and **provided that there are enough of them close by**
- This will not be easy because of the **curse of dimensionality**:
  - Even for low dimensions and large samples, **the sample space is essentially empty**
  - And for most problems, **there never will be close enough points**
  - Thus, to get k observations we may go too far away from x and the k-NN predictions will not be meaningful