Master-MIRI
Topics on Optimization and Machine Learning (TOML)

José M. Barceló Ordinas
Departament d'Arquitectura de Computadors
(UPC)

- **Machine Learning (ML) in this course:**
  - **Objective:** this course is focus on some ML algorithms and its relation to optimization. Moreover, this course is a continuation of other courses in which several concepts related to statistical analysis and machine learning is given (SMDE-MIRI and SANS-MIRI).
    - E.g. SANS-MIRI: Bayesian posterior, maximum likelihood, linear regression, Markov chain monte carlo (MCMC), probabilistic graphical models (PGM),
  - We will focus in topic 2 in some supervised techniques:
    - multiple linear regression, nearest neighbor algorithms,
    - random forest,
    - kernels and Gaussian processes,
    - support-vector machines/regression (SVM/SVR).
  - We will focus in topic 3:
    - Deep learning (neural networks)

- **Machine Learning (ML) along time:**
  - **Pattern recognition:** look for patterns and regularities in data (term from 70's-80's) and use statistics and signal processing machinery (mostly used in electrical engineering), and it was not necessary that a machine performed automatically this process,
  - **Machine learning (ML):** study of systems that can learn from data (term from the 90's-00's), and implied to fed a computer with data and let the computer to make its own decisions, and then becoming part of computer science,
  - **Deep Learning (DL):** use of large amounts of data (big data) in the process of automatically study systems that can learn from data.
  - **Data science (DS):** scientific field that extract information from data, and can include machine learning, deep learning, big data, among others, so is related to statistics, data analysis, data mining and informatics.
  - **Artificial intelligence (AI):** tries to mimic human reasoning and actions, and includes reasoning, natural language processing and machine learning,

In general all these fields imply the knowledge of **algebra**, **optimization**, **probability** and **programming**, and with DL/DS the use of **specific technologies to handle high amounts of data** (Hadoop, Spark, Cassandra, etc., seen in CCBDA-MIRI course)

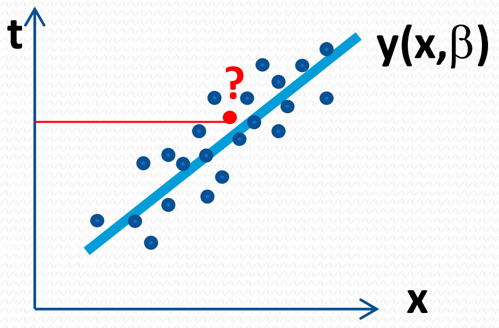
- Machine Learning (ML) approaches:

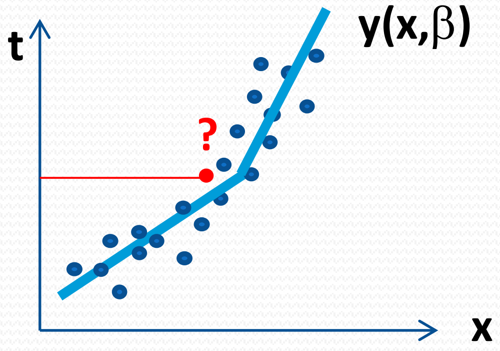
  - **Supervised learning:** the algorithm is given inputs and outputs, and has to look for a mapping between the inputs and the outputs. Examples are multiple linear regression, SVM/SVR, random forest, K-nearest neighbors (KNN), etc.

  - **Unsupervised learning:** the algorithm learns patterns from untagged data, e.g. clustering algorithm (k-means) or PCA (reduction of dimensions), or anomaly detection among others,

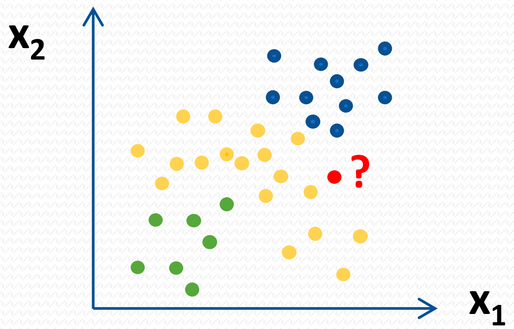
  - **Reinforcement learning:** taking actions to maximize cumulative rewards and normally are processed using dynamic programming for solving optimization problems, such as Markov Decision Processes (MDP's). The user receives feedback and is guided to the best outcome (trial and error, no training phase).

- **Supervised Learning: regression and classification …**
  - **Objective:** we want to find the relationship between a **response** or "dependent variable" **t** and a set of **measurements** or "predictors or features" **x** (called "independent variables").
  - **Regression:** relationships usually are **quantitative**. Example: **t** measures the amount of CO_2 in Barcelona and **x** the number of vehicles (linear on the left and or non-linear on the right).





  - **Classification:** relationships usually are **qualitative**. For example (x_1,x_2)= (#vehicles, CO_2). (H,H) as blue, (H,L) or (L,H) as orange, and (L,L) as green, with H=High, L=Low. Here t={blue, orange, green}.



Unsupervised Learning: clustering …

- **Objective:** we have a vector for every measurement *i*, we observe vector $\mathbf{x}_i=(x_1, \dots, x_M)$, but we don't have a response t, so we can not supervised our analysis using a regression or classification method. However we can do other things such as find clusters in our data, e.g. see "An introduction to statistical learning" (James-Witten-Hastie-Tibshirani).

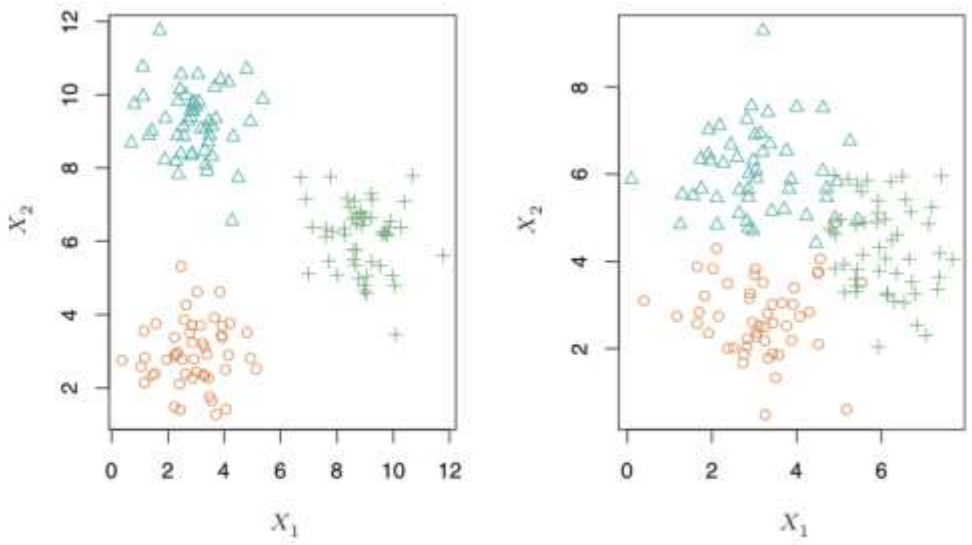


FIGURE 2.8. *A clustering data set involving three groups. Each group is shown using a different colored symbol.* Left: *The three groups are well-separated. In this setting, a clustering approach should successfully identify the three groups.* Right: *There is some overlap among the groups. Now the clustering task is more challenging.*

Multiple Linear Regression (MLR) …

- **Sampling** is typical in many Computer science applications,

For example: consider a wireless sensor network (WSN) that measures air pollution, specifically, **tropospheric ozone (O_3)**, measured in μgr/m^3. Ozone varies (cross-sensitive) with temperature and relative humidity. Thus, when measuring O_3, we have to measure temperature (T) and relative humidity (RH),

- **Metal-oxide (MOX) sensors** measure resistence (Ohms) or voltage (Volts) while **electro-chemical sensors** measure voltage (Volts). These values have to be converted to μgr/m^3, a process that we call **calibration**,
- In order to calibrate, we deploy a node with three sensors on a reference station that gives a reference value (true value) of ozone. We call **t** to the values taken by the reference station (μgr/m^3), $\mathbf{x_1}$ to the values of the resistence MOX O_3 sensor (Ohms), $\mathbf{x_2}$ to the values of the T sensor (ºC), and $\mathbf{x_3}$ to the values of the RH (%) sensors and assume a linear dependence $y(x, \beta)$ among them:

$$t \sim y(x, \beta) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

- The objective is:

1. **to find coefficients β_0, β_1, β_2 and β_3,** given that we know **t, x_1, x_2 and x_3**,
2. if we remove the reference station, **be able to predict values of t** (ozone) given that we are measuring **x_1, x_2 and x_3**.

Multiple Linear Regression (MLR) …

- **In general:** the problem is fitting **y** (dependent variable) with several features **x** (independent variables or regressors). We assume that **x is a vector of dimension M+1** (M=3 in our example):

$$t_i \sim y_i(x_i, \beta) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_M x_{iM} = \Sigma_{j=0\ldots M} \beta_j x_{ij},$$

- Let us assume, in the previous example, that **we take N samples**, 1 every hour, (rows in the matrix) for the reference station and for each of the sensors, calling $X \in R^{N\mathrm{x}M+1}$ to the **design matrix**: $t \sim y = X\beta,$

t (μgr/m^3)
73
96
94
100
102
105
108
…
46
41

$\sim$

offset	x_1 (KOhms)	x_2 (ºC)	x_3 (%)
1	53.28	33	29.33
1	100.35	29.8	28.43
1	268.28	22.1	35.73
1	375.32	23.1	34.3
1	429.10	23.9	32.47
1	483.69	25.0	32.17
1	427.64	25.1	32.17
…	…	…	
1	98.33	15	68.93
1	81.02	14.0	71

β_0
β_1
β_2
β_3

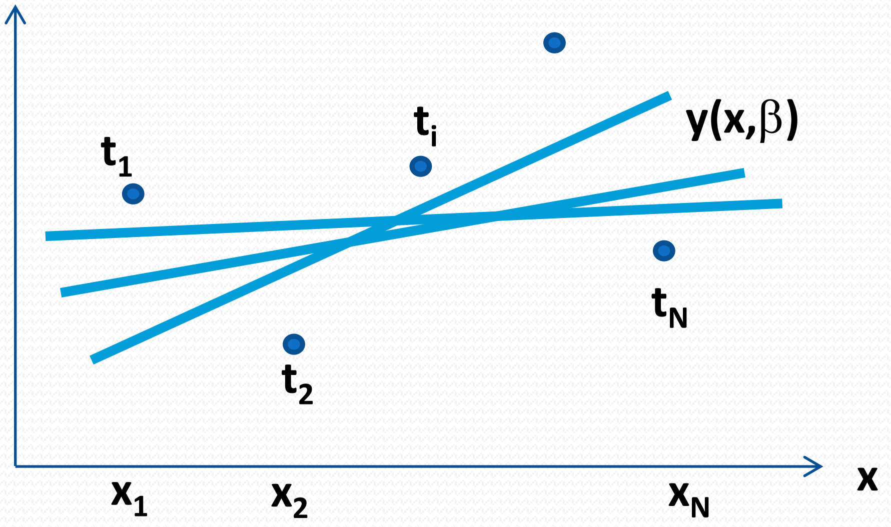
Multiple Linear Regression (MLR) basics …

- **Objective:** find those coefficients β (of dimension M+1 since takes into account the offset β_0) that form an **hyperplane $y(x,\beta) = x^T \beta$**, by fitting the polynomial to the training data given by pairs $(\mathbf{x}_i, t_i)$, i=1,…,N. The coefficients β have to minimize a cost/loss function.
- Let's call **residuals** to $r_i = \mathbf{x}^T_i\beta - t_i$, and the **residual vector** $\mathbf{r} = (r_1, \ldots, r_N) = X\beta - \mathbf{t}$. The regression problem consists in finding a vector β as closed as possible to each point t_i, it is to say, that minimize all the residuals. This can be done by solving the norm approximation problem:

$$\textbf{minimize} \quad J(\beta) = 1/(2N)\, ||X\beta - \mathbf{t}||^2_2 = 1/(2N)\, ||\mathbf{r}||^2_2 = 1/(2N)\, \Sigma_{i=1\ldots N}\, r^2_i$$

$$\textbf{var} \quad \beta$$

Where $J(\beta)$ is a penalty/loss function.



Multiple Linear Regression (MLR) basics …

- The most common norm approximation problem is the **least-squares approximation problem** that involves the Euclidean l-2 norm. This can be done **minimizing** the error/loss function:

$$J(\beta) = 1/(2N)\ ||\mathbf{X\beta - t}||^2_2 = 1/(2N)\ (\mathbf{X\beta - t})^t\ (\mathbf{X\beta - t}) = 1/(2N)\ ||\mathbf{r}||^2_2 = ½\ \Sigma_{i=1\ldots N}\ \mathbf{r}^2_{\mathbf{i}} =$$

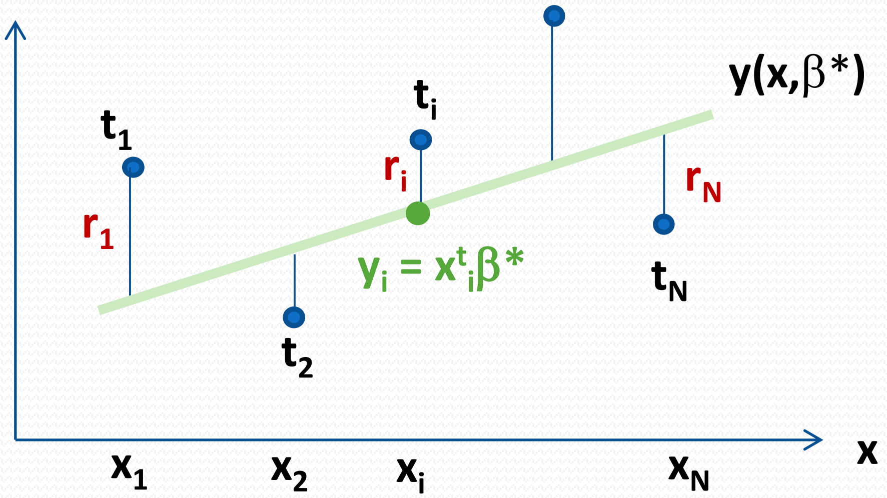
$$= 1/(2N)\ \Sigma_{i=1\ldots N}\ (\mathbf{x}^t_{\mathbf{i}}\ \beta - t_i)^2 = 1/(2N)\ \Sigma_{i=1\ldots N}\ (\Sigma_{j=0\ldots M}\ \beta_j x_{ij} - t_i)^2$$

where r_i are the residuals. We divide by N because we don't want that the loss function depends on the number of samples. This is equivalent to the **norm approximation problem**:

$$\textbf{minimize}\quad J(\beta) = 1/(2N)\ ||\mathbf{X\beta - t}||^2_2 \rightarrow \textbf{minimize}\ 1/(2N)\ (\boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - 2\mathbf{t}^T\mathbf{X}\boldsymbol{\beta} + \mathbf{t}^T\mathbf{t})$$

Note that the problem is convex on β. Since it is an unconstraint COP, The solution of this problem is obtained by equalizing the gradient of $J(\beta)$ to zero and clearing β:

$d(J(\beta))/d\beta = 0 \rightarrow \boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\ \mathbf{X}^T\mathbf{t}$ that are called the **normal equations**.



Multiple Linear Regression (MLR) basics: geometric interpretation …

- **Geometric interpretation**: take the columns of the design matrix **X**: $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_M$, with $\mathbf{x}_i \in \mathbf{R}^N$. These vectors of dimension N, span a subspace of dimension M+1. If we don't consider the offset, then the spanned subspace by $\mathbf{x}_1, \ldots, \mathbf{x}_M$ is of dimension M,
- Now, we have vector $\mathbf{t} \in \mathbf{R}^N$. Thus, any point of the subspace spanned by $\mathbf{x}_0, \ldots, \mathbf{x}_M$ closest to **t is the projection of point t in the subspace X in the norm ||·||**, it is to say, the optimal point of the optimization problem,

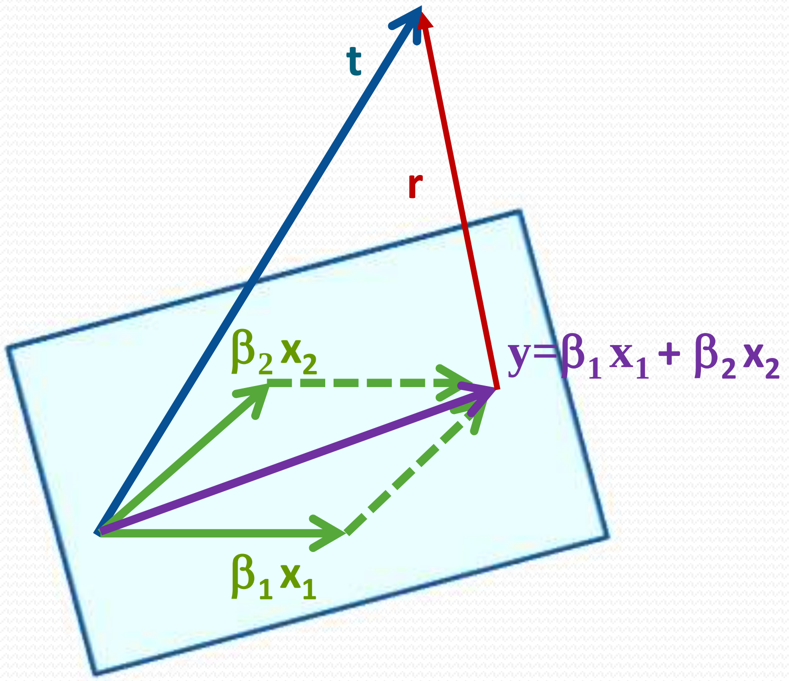
Here, you have an example with M=2 regressors or features (without offset), where the residual **r = y-t = Xβ-t** represents the error committed at each point.

Using the geometry of vectors, **Xβ is the orthogonal projection of t on the subspace spanned by X where** $\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}$,

Then the predictions $y \in R^N$ of a new set of measures **Z** will be calculated as:

$$y = Z\beta = Z(X^TX)^{-1}X^Tt$$

In general there can be numerical difficulties when $\mathbf{X}^T\mathbf{X}$ is close to singular (non-invertible), e.g. colinear column vectors.



least squares approximation problem – gradient descent method revisited

If many samples, the normal equations are difficult to obtain, i.e., (numerical errors when the inverse of a very large matrix is obtained). **Use the gradient descent algorithm to obtain the β's**. Using the Newton's method implies to calculate the inverse of the Hessian (that it is expensive).

Let us assume that $\boldsymbol{\beta}$ is a vector of coefficients ($\beta_0, \ldots, \beta_M$), and the loss function is $J(\beta) = 1/(2N) \sum_{i=1\ldots N} (x_i^T \beta - t_i)^2$. The gradient of $J(\boldsymbol{\beta})$ is:

$$\nabla J(\beta) = \sum_{i=1\ldots N} (x_i^T \beta - t_i)\, x_i/N$$

where $\mathbf{x}_i^T = (x_{i0}, x_{i1}, \ldots, x_{iM})$ for all i=1, …, N, and, the **gradient descent algorithm** tell us that:

$$\beta^{k+1} = \beta^k - \eta \nabla J(\beta^k) = \beta^k - \eta/N \sum_{i=1\ldots N} (x_i^T \beta^k - t_i)\, x_i,$$

with i=1, …, N. This is called the **LMS (Least-Mean-Squares) algorithm**, with some initial $\boldsymbol{\beta}^0 = (\beta^0_0, \ldots, \beta^0_M)$, and iterate in k until some accuracy $\boldsymbol{\varepsilon}$ is achieved.

- The convergence is better **if not all** the N data points are used at each iteration k. Then there are several variations that solve the LMS algorithm: **stochastically** (use one point of the N at each iteration k) or **in mini-batches** (use N_k data points of the N at each iteration k), among others.

least squares approximation problem – gradient descent method revisited

See tutorial "*An overview of gradient descent algorithms*", **Sebastian Ruder**.

We discuss gradient descent methods used with data. Second order methods such as Newton's method are most of the time infeasible with large amounts of data since it has to invert the Hessian. Assume that $J(\beta,x) = 1/(2N) \Sigma_{i=1\dots N} (x_i^T \beta - t_i)^2$, and $J(\beta,x_{j,j+n}) = 1/(2n) \Sigma_{i=j\dots j+n} (x_i^T \beta - t_i)^2$. **Normalize the data** using the mean and standard deviation: $(x_i-\mu_x)/\sigma_x$, and **shuffle the data** before each iteration k to avoid bias the optimization algorithm.

The step size η in machine learning is sometimes called **learning rate**.

1. **Batch gradient descent:** each update takes all samples, not useful with on-line data,

$$\beta^{k+1} = \beta^k - \eta \nabla J(\beta^k,x),$$

2. **Stochastic gradient descent (SGD):** each update takes one samples, good for on-line computation but objective function fluctuates with high variance,

$$\beta^{k+1} = \beta^k - \eta \nabla J(\beta^k,x_{i,i+1}),$$

3. **Mini-batch gradient descent:** each update takes a subset of samples called mini-batches (50-256), reduces the variance of the stochastic improving stability,

$$\beta^{k+1} = \beta^k - \eta \nabla J(\beta^k,x_{i,i+n}),$$

least squares approximation problem – gradient descent method revisited

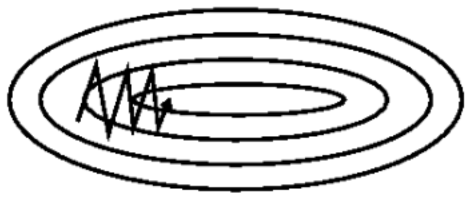
In Batch, mini-batch and stochastic, the learning rate η is constant, e.g. $\eta=0.1$. **Low values** of η make the algorithm to have a low convergence, while **large values** of η can cause the loss function to fluctuate or diverge. There are methods that adjust the learning rate to the data or to pre-defined schedules. Let's see a couple of examples to grasp the idea:

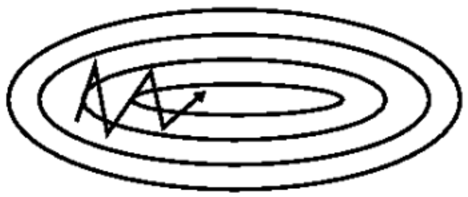
1. **Momentum:** avoid ravines (areas where the surface curves much more steeply in one dimension than in another). To avoid oscillations around the slopes of ravines, momentum accelerate SGD in the relevant direction and dampens oscillations by adding a fraction $\gamma=0.9$ (called the <u>momentum coefficient</u>) and a <u>velocity vector</u> **v**:

$$\mathbf{v}^{k+1} = \gamma \mathbf{v}^{k} - \eta\, \nabla J(\boldsymbol{\beta}^{k},\mathbf{x}),$$

$$\boldsymbol{\beta}^{k+1} = \boldsymbol{\beta}^{k} + \mathbf{v}^{k+1} = \boldsymbol{\beta}^{k} + \gamma \mathbf{v}^{k} - \eta\, \nabla J(\boldsymbol{\beta}^{k},\mathbf{x}),$$

The momentum term $\gamma \mathbf{v}^{k}$ increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions.


(a) SGD without momentum


(b) SGD with momentum

least squares approximation problem – gradient descent method revisited

2. **Nesterov accelerated gradient (NAG):** the main problem with momentum is that we first compute the gradient and then make a big jump (using the momentum term in the gradient direction). NAG makes a big jump in the direction of the previous accumulated gradient and then measures the gradient where it ends up and makes a correction:

$$\mathbf{v_t}^{k+1} = \gamma\, \mathbf{v_t}^{k} - \eta\, \nabla J(\beta^{k} - \gamma\, \mathbf{v_t}^{k}, \mathbf{x}),$$

$$\beta^{k+1} = \beta^{k} + \mathbf{v_t}^{k+1} = \beta^{k} + \gamma\, \mathbf{v_t}^{k} - \eta\, \nabla J(\beta^{k} - \gamma\, \mathbf{v_t}^{k}, \mathbf{x}),$$

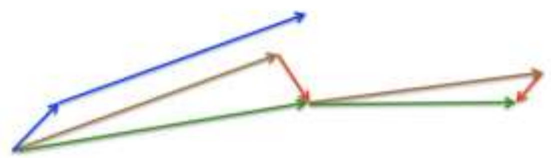


Figure 3: Nesterov update (Source: G. Hinton's lecture 6c)

In blue: momentum that obtains gradient and makes a big jump

In brown: NAG that makes a big jump, then a gradient correction ending in the green line improving the responsiveness.

Other methods are **Adagrad, Adadelta, Adam or AdaMax**. One of the most used is Adam that uses exponentially decaying averages of estimates of the first and second moments of the past gradients. Make a look at S. Ruder tutorial for more info.

Multiple Linear Regression (MLR) basics: accuracy of the model …

We need some quality metrics to measure the error of the fitting, it is to say, how near is each $y_i = \mathbf{x}_i^T \beta$ to t_i or in vector form $\mathbf{y} = \mathbf{X}\beta$ to $\mathbf{t}$.

The **Residual Sum of Squares** is **RSS=** $2N\,J(\beta) = ||r||^2_2 = \Sigma_{i=1\ldots N} (\mathbf{x}_i^t\beta - t_i)^2$.

- **Root Mean Squared Error (RMSE)** allows to compare different sizes of data sets, since the RMSE is measured in the same scale than the target scale values t_i:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N} RSS}$$

Observe that MSE = 1/N RSS = 2 $J(\beta)$

- **R^2 (Coefficient of determination)** measures the proportion of variability in **y** that can be explained using **X** and itis bounded between 0 and 1. When R^2 is close to 1 indicates that a large proportion of variability in the response has been explained by the regression.

Training set, validation and test set, cross-validation …

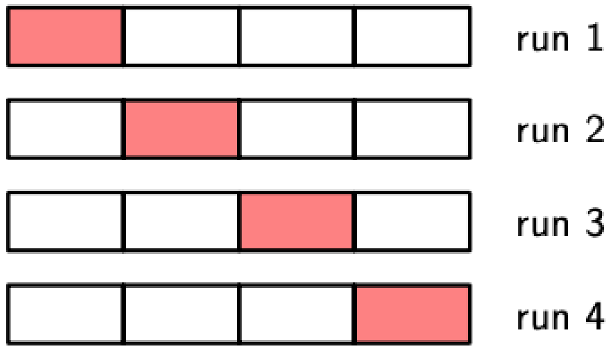
- We assume a data set of size N samples (and M features). In order to asses the data set with a given model, we divide the data set in three parts: the **training set**, the **validation set**, and the **test set**.
- We use the train model to asses the parameters using the training set. We can observe the performance metrics on these set to assess the goodness of the model, but we since these parameters have been adjusted (optimized) to fit the training data, we can not assure that these parameters are going to behave well on new data,
- We use the validation set to tune **hyperparameters**, e.g. regularization parameters, o hidden parameters such the number of layers in a neural network.
- In order to assess how good is the model, estimate over the test set and see how are the performance metrics over the test set to see the goodness of the model studying the bias-variance trade-off., e.g. to avoid over-fitting.

- Sometimes there are no hyperparameters and only training and test set is used.

Training set, validation and test set, cross-validation …

- **Cross-Validation** is a technique use when there is few data to be split for training and validation purposes and we are interested in using as much data as possible for the training. The idea is to split the data set in S partitions which allows (S-1)/S partitions to participate in the training, while the last partition participates in the validation. This is done S times as shown in the figure and results are averaged.

The technique of S-fold cross-validation, illustrated here for the case of $S = 4$, involves taking the available data and partitioning it into S groups (in the simplest case these are of equal size). Then $S - 1$ of the groups are used to train a set of models that are then evaluated on the remaining group. This procedure is then repeated for all S possible choices for the held-out group, indicated here by the red blocks, and the performance scores from the S runs are then averaged.



Bias-Variance trade-off …

Let us consider that our data **t** follows a general function **f(x)** (not necessarily linear) with a Gaussian error: $\mathbf{t = f(x) + \epsilon}$, with $\boldsymbol{\epsilon \sim N(0,\sigma^2)}$, and that we fit our data with a function $\mathbf{y(x, \beta)}$, e.g., $\mathbf{t \sim y(x, \beta)}$,

Take the expectation over a training set $D=\{(x_1,t_1), \dots, (x_N,t_N)\}$ of the square error, **MSE=** $\mathbf{E_D[(y_D(x) - t)^2] = E[(y_D(x) - f(x) - \epsilon)^2]}$, then it can be proof that:

$$E_D[(y_D(x) - t)^2] = \text{Bias}_D[y_D(x)]^2 + \text{Var}_D[y(x)] + \sigma^2,$$

- The **Bias** is the difference between our average predictions over all datasets and the desired regression function (true value).

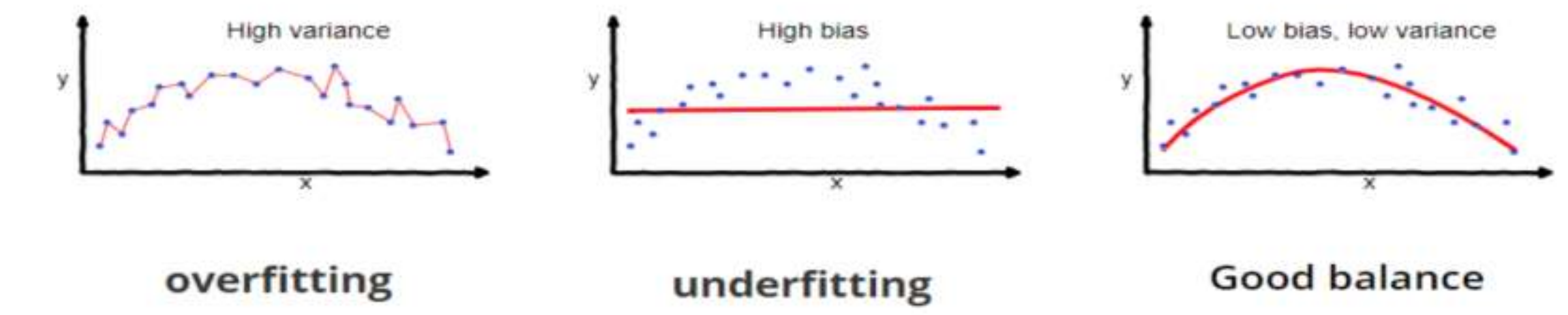
$$\text{Bias}_D[y_D(x)] = E_D[y_D(x)] - f(x),$$

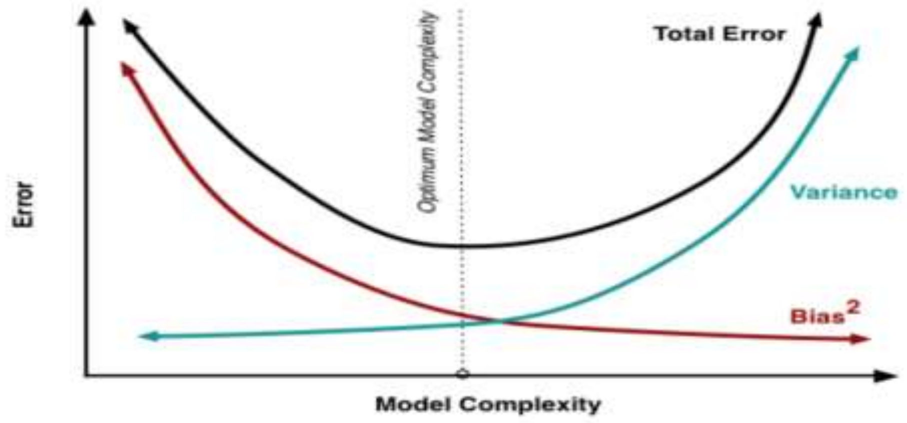
- The **Variance** represents how much the learning methods moves with respect the mean (spread of the data), paying attention to the training data. It is to say, how sensitive is the the function $y_D(x)$ to a particular data set,

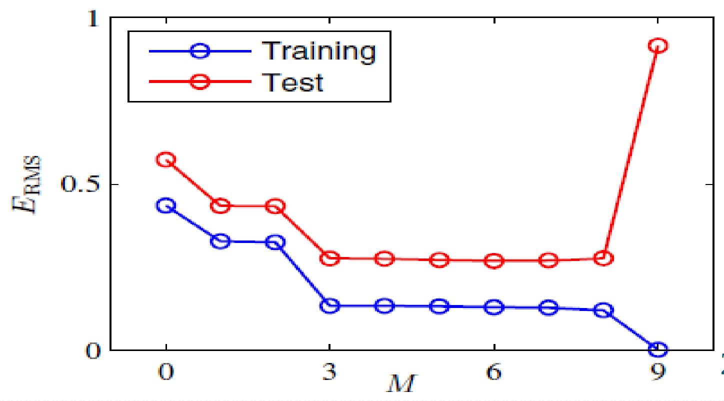
$$\text{Var}_D[y(x)] = E_D[y_D(x)^2] - E_D[y_D(x)]^2,$$

Bias-Variance trade-off …

- The Bias is high when <u>making simple assumptions on the model</u> and does not take in to account too much to the training data, thus high training error corresponds to high bias since our model is **underfitting** the train data.
- High variance produces **overfitting**,
- The **error *σ*** is irreducible and is due to the noise (gaussian distributed),







Multiple Linear Regression basics: Regularization …

- **Regularization:** a technique to fight overfitting:
  - Add a **penalty** λ to the error function and **minimize** $J(\beta)$

$$J(\beta) = 1/(2N)\ ||\mathbf{y(x,\beta) - t}||^2_2 + \lambda/2\ ||\beta||^2_q$$

- Regularization with L2-norm (q=2), $||\beta||^2_2 = \Sigma_{j=1\ldots M} (\beta_j)^2$, is called **ridge regression**.
- Regularization with L1-norm (q=1), $||\beta||^2_1 = \Sigma_{j=1\ldots M} |\beta_j|$, is called **Lasso**.
- For different λ, we may obtain different values of the coefficients β. Be careful to normalize (with respect mean and std) the inputs since penalties are different under scaling,

Multiple Linear Regression basics: Regularization …

- **Ridge regression (l-2 norm)**: $J(\beta) = 1/(2N)\ ||\mathbf{y(x,\beta) - t}||^2_2 + \lambda/2\ ||\beta||^2_2$

We can solve it exactly since in vectorial form:

$$J(\beta) = 1/(2N)\ (\mathbf{X\beta - t})^T(\mathbf{X\beta - t}) + \lambda/2\ \beta^T\beta,$$

and $d(J(\beta))/d\beta = 0$

$$\beta = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})^{-1}\ \mathbf{X}^T\mathbf{t}$$

We can observe that still β is linear in t, and we add a positive constant λ to $\mathbf{X}^T\mathbf{X}$ before inversion, assuring that is non-singular (invertible)

- **Lasso regression (l-1 norm):** $J(\beta) = 1/(2N)\ ||\mathbf{y(x,\beta) - t}||^2_2 + \lambda/2\ ||\beta||^2_1$

We cannot solve it exactly, and we can solve it as a quadratic optimization problem, but there are efficient ways of obtaining a solution with the same cost than ridge regression.

Multiple Linear Regression basics: Regularization, optimization interpretation …

- **Ridge Regression:**

minimize $J(\beta) = 1/(2N)\ ||\mathbf{y}(\mathbf{x}, \boldsymbol{\beta}) - \mathbf{t}||^2_2$

s.t $||\boldsymbol{\beta}||^2_2 = \Sigma_{j=1\ldots M} (\beta_j)^2 \leq s$

var $\boldsymbol{\beta}$

where $s=2/\lambda$. Large λ implies low coefficient estimates, thus minimizing the effect of overfitting (decreasing the variance at the cost of slightly increase the bias) including all coefficients $\boldsymbol{\beta}$'s.

- **Lasso:**

minimize $J(\beta) = 1/(2N)\ ||\mathbf{y}(\mathbf{x},\boldsymbol{\beta}) - \mathbf{t}||^2_2$

s.t $||\boldsymbol{\beta}||^2_1 = \Sigma_{j=1\ldots M} |\beta_j| \leq s$

var $\boldsymbol{\beta}$

Some coefficients ($\boldsymbol{\beta}$'s) estimates will be zero (it is doing variable selection automatically), eliminating the overfitting.

- ## Multiple Linear Regression basics: Regularization …
  - Difference between **Lasso (L1-norm) and Ridge Regression (L2-norm):** see how in the Lasso solution (L1-norm) $\beta_1=0$ and $\beta_2 \neq 0$, while in the Ridge regression (L2-norm) $\beta_1, \beta_2 \neq 0$.

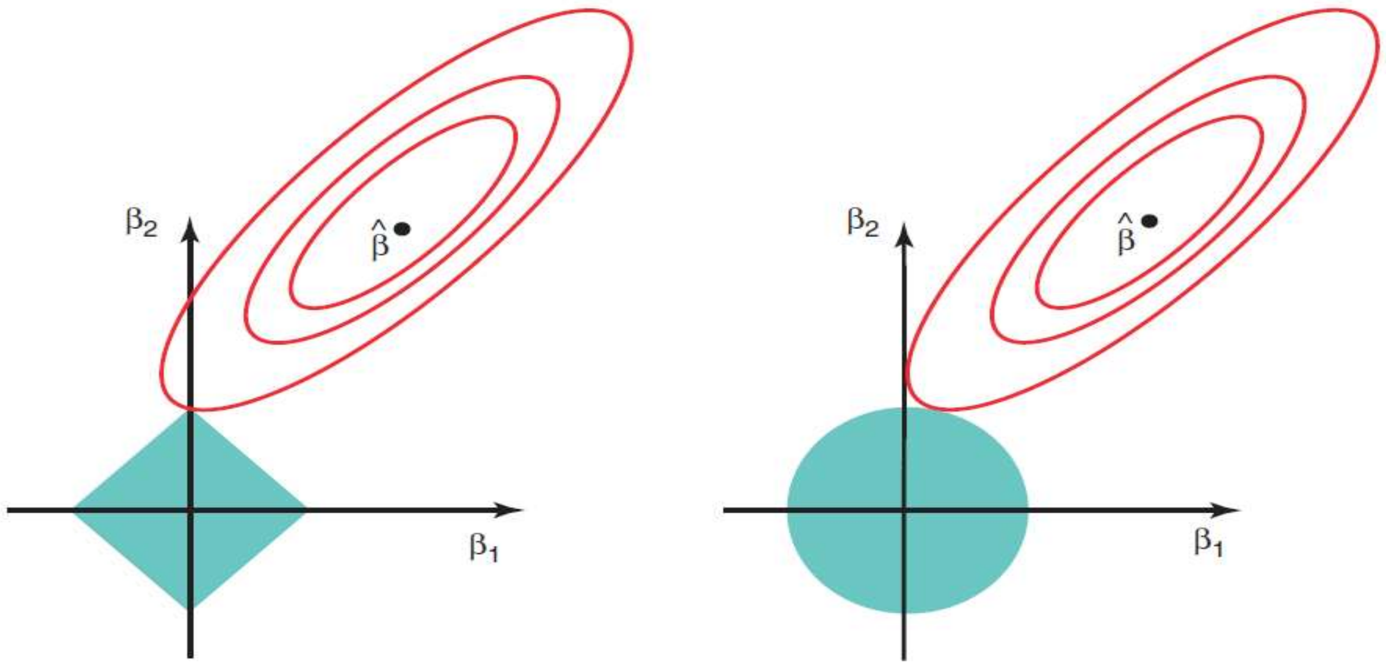


FIGURE 6.7. *Contours of the error and constraint functions for the lasso* (left) *and ridge regression* (right). *The solid blue areas are the constraint regions, $|\beta_1| + |\beta_2| \leq s$ and $\beta_1^2 + \beta_2^2 \leq s$, while the red ellipses are the contours of the RSS.*

Multiple Linear Regression basics: Regularization …

- Ridge regression minimizes the impact of some variables but does not eliminate them. Lasso is good for eliminating some of your variables, i.e, some β coefficients will be zero (sparse solution), while in ridge regression possibly all β contribute.
  - Ridge regression is a shrinkage while Lasso works as subset selection technique since eliminates variables, thus doing shrinkage and variable selection (useful if many variables),
- Ridge regression is computationally better, although there are good methods for solving Lasso, e.g. KKT conditions explain a lot of things here, and leads to algorithms such as **Least Angle Regression** for solving Lasso,
- Thinking also in the Bayesian interpretation, all of them are derived as posterior modes (maximize the posteriors) using different priors, e.g. Lasso assumes a Laplace distribution on β's while ridge regression assume a normal distributions on β's. However it is more common to use the mean of the posterior than the mode, and ridge regression is also the mean, but lasso it is not,
- Smaller values of *q=1* in the regularization term $||\boldsymbol{\beta}||^2_q$, makes them difficult to be solved, since they are not convex. Thus, q=1 is the smallest value of *q* that makes the problem convex,
- **ElasticNet penalty**: it is a weighted (with weight α) solution between lasso and ridge regression, e.g., $\lambda/2\ (\alpha\ ||\boldsymbol{\beta}||^2_2 + (1-\alpha)\ ||\boldsymbol{\beta}||^2_1)$, selects as lasso and shrinks as ridge.

- **Bayesian Multiple Linear Regression basics …**
  - Remember than in the Bayesian formulation, the **posterior** $p(\theta|y, x)$ distribution is obtained from the **likelihood** $p(y|\theta,x)$ and the **prior** $p(\theta)$, it is to say: $\mathbf{p(\theta|y, x) \sim p(y|\theta,x)\, p(\theta)}$. Then, assuming in our previous nomenclature that $y=t$, $x=X$ and $\theta=\beta$,
  - In this case we assume that:

    $$p(t|X, \beta) \sim N(X\beta, \sigma^2 I),$$

  where the errors $\boldsymbol{\varepsilon}$ are also Gaussian distributed with mean zero and identical variance $\boldsymbol{\sigma^2}$, and I is the identity matrix (ones in the diagonal, zeroes otherwise).
  - Now, we assume a non-informative prior for the β's, e.g., uniformly distributed over the error variance: $p(\beta) \sim \sigma^{-2}$,
  - Then, the posterior probability will be:

    $p(\beta|t, X) \sim N(\beta^*, S_0)$, is normal distributed with mean $\boldsymbol{\beta^*}$ and variance S_0.
  - It can be proof then, that:

    $\beta^* = (X^TX)^{-1} X^T t$ mean is the normal equations obtained in the frequentist assumption

    $S_0 = (X^TX)^{-1} \sigma^{-2}$

  And the errors are calculated using marginal distributions, and have scaled inverse-$\mathcal{X}^2$ form:

    $\sigma^2 \sim \text{Inv-}\mathcal{X}^2(N-M, s^2)$ with $s^2=(N-M)^{-1} (X\beta^*-t)^T(X\beta^*-t)$
  - We can also follow a MCMC (Markov Chain Monte Carlo) approach for obtaining simulated data from which to obtain β^* and S_0.

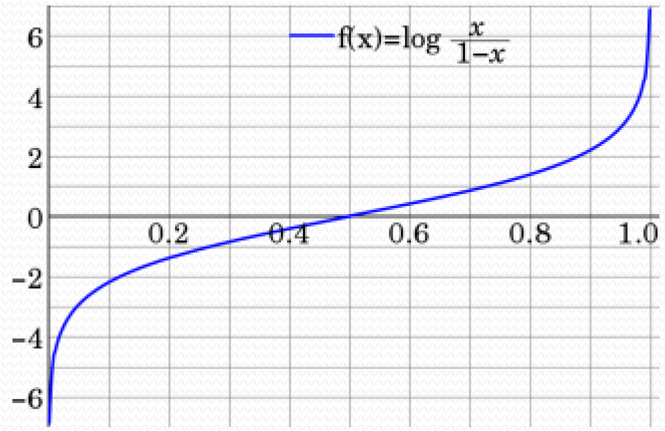
- **Logistic Regression …**
  - **Algorithm for classification:** assume that we have a vector of measurements $\mathbf{x}_i=(\mathbf{x}_{i1}, \ldots, \mathbf{x}_{iM})$, for i=1,…,N, and each is assigned an output label $\mathbf{t}=(t_1,\ldots,t_N)$, where $t_i=\{L_1, \ldots, L_K\}$.
  - The **goal** is to obtain a linear relationship between the predictor variables and the log-odds (**logit**) of the event t=1.
  - If p is a probability, we first define the **logit function x=f(p)** as:

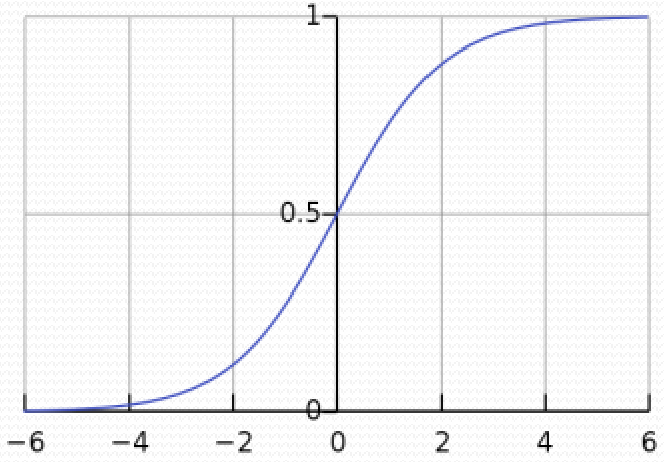
$$f(p) = \text{logit}(p) = \log(\, p/(1-p)\,) = \log(p) - \log(1-p) \rightarrow x = f(p) = \ln(\, p/(1-p)\,)$$

Where the most used logarithm is the natural logarithm in base *e*.

  - We define the **sigmoid function p=S(x)** as the inverse of the logit function

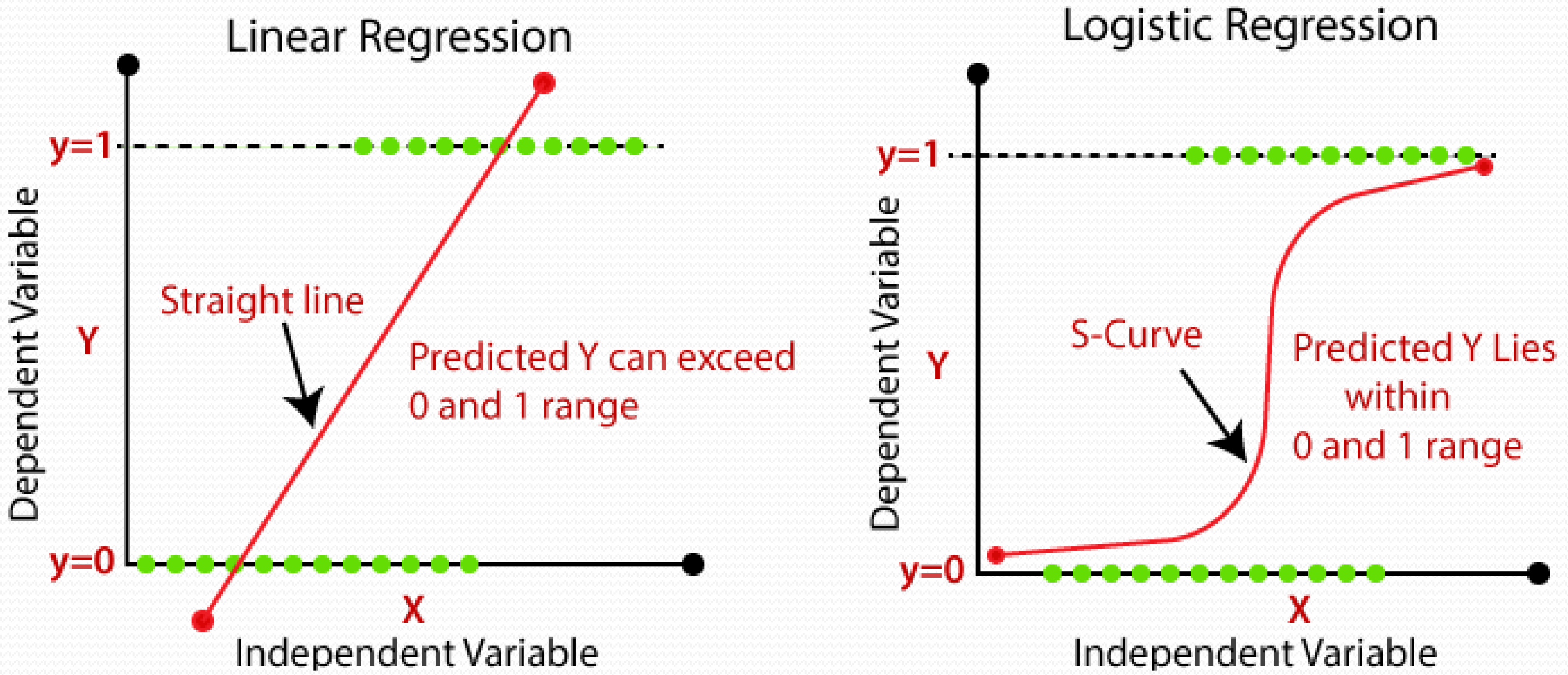
if $x = f(p) = \ln(\, p/(1-p)\,) \quad \Leftrightarrow \quad p = S(x) = 1/(1+e^{-x}) = e^x/(1+e^x) = 1-S(-x)$





Logistic Regression …

- Algorithm for classification: assume two output labels t_i= {L_1,L_2}= {0,1}.
- The **goal** is to obtain a linear relationship between the predictor variables and the log-odds (**logit**) of the event t=1.



- **Logistic Regression …**
  - We assume two labels $t_i=\{0,1\}$ and M features, and we will have to estimate M+1 $\boldsymbol{\beta}$ coefficients $\boldsymbol{\beta}= (\beta_0, \beta_1, …, \beta_M)$.
  - We define the probability of t=1 as:

    $p= p(t=1) = S(\boldsymbol{\beta_0+\beta_1 x_1+ … +\beta_M x_M}) = e^{\beta 0 + \beta 1\, x1 + … + \beta M\, xM} \boldsymbol{/(1+} e^{\beta 0 + \beta 1 x1 + … + \beta M\, xM}\boldsymbol{)}$

  and manipulating the expression, we can obtain :

    $p(t)/(1-p(t)) = e^{\beta 0 + \beta 1x1 + … + \beta M\, xM}$
  - Then, we define the logit as:

    $f(p) = \log (p/(1-p)) = \boldsymbol{\beta_0 + \beta_1 x_1 + … + \beta_M x_M}$

  - We use **maximum likelihood** (better statistical properties than least squares for this problem) for estimating the $\boldsymbol{\beta}$ coefficients. Now, $\boldsymbol{\beta^*}$ are the optimal coefficients and we can predict new values using the following:

    $p= p(t=1) = S(\boldsymbol{\beta^*_0+\beta^*_1 x_1+ … +\beta^*_M x_M})$

  So we now with certain probability if we classify a point $x=(x_1,…,x_M)$ as t=1 or as t=0.

  - **Multi-class** (with K classes) extensions exist for the logistic regression, but they are not commonly used. For multiclass problems use **Linear Discriminant Analysis (LDA),** that uses the Bayes' theorem for classification.

- **Nearest neighbours methods: K-Nearest Neighbour (KNN) …**

  - The goal of nearest neighbour methods is to predict based on the neighbourhood of the data. Let's have a D-dimensional vector of features $\mathbf{x}= (x_1,\ldots,x_D)$, and now $\{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ training data and labels $\{t_1, \ldots, t_N\}$.

$$\tilde{y}(x)= 1/k \, \Sigma_{\mathbf{xi} \in Nk(x)} \, y_i$$

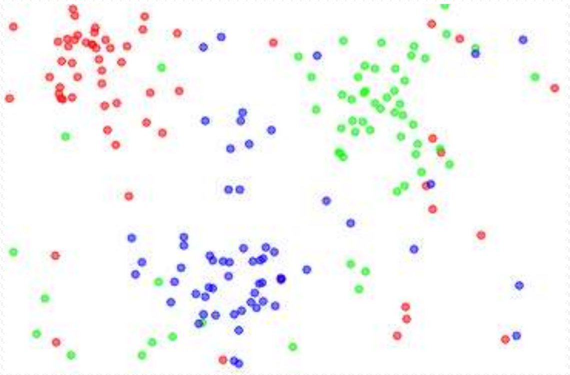
The prediction for a point x is done using the k-neighbours of point x. For that, order distances as $||\mathbf{x_1}- \mathbf{x}|| \leq ||\mathbf{x_2}- \mathbf{x}|| \leq \ldots \leq ||\mathbf{x_{(n)}}- \mathbf{x}||$ for some norm (e.g. l-2 = Euclidean distance).

If regression: take the average of the k neighbours,

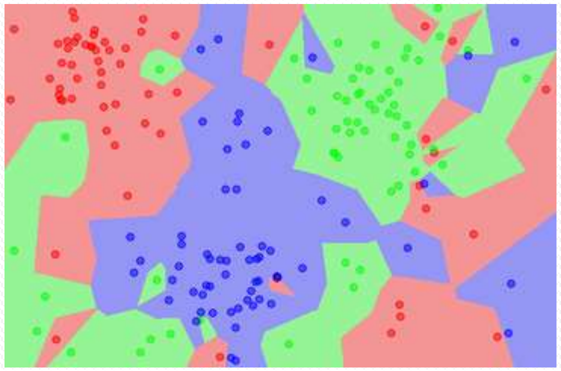
If classification: take the mode (the most voted result) of the k neighbours. To avoid ties in binary classification, it is better to select k odd. If k=1 (special case) it is called nearest-neighbour.

Nearest neighbours methods: K-Nearest Neighbour (KNN) …

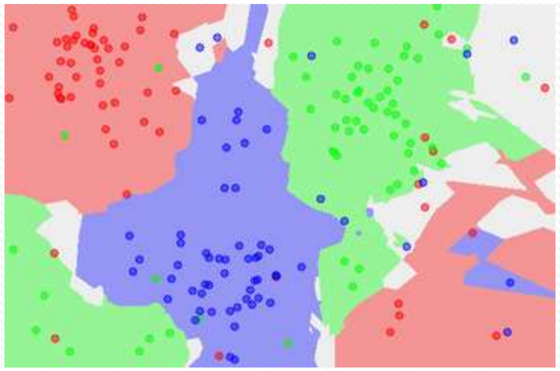
- Example taken from Wikipedia: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm



Data set



1-NN



5-NN

Nearest neighbours methods: K-Nearest Neighbour (KNN) …

- The hyper-parameter *k* has to be selected using hyper-parameter optimization (e.g. using cross-validation),

If k=1, the predictions become more unstable since we predict according to a single neighbor that can be in our class or in any class,

As k grows, the prediction are more stable since we use averages for regression and majority voting (mode) for classification, but at some point we begin to increase the error rate. At this point we have to stop increasing k,

- **Advantages of KNN:** the algorithm is simple and takes few assumptions, and can be used for regression or classification,
- **Disadvantages of KNN:**
  - Computational challenges if the training set is huge (obtain the distances between all pairs) → there are a lot of proposals to reduce the number of distance evaluations,
  - Computational challenges if the number of features D is large → perform feature extraction (or reduction) that consists on reducing the number of features by transforming the input data to a reduced representation of the set of features, e.g. use PCA (Principal Component Analysis), LDA (Linear Discriminant Analysis), etc.