Summary of Topics on Optimization and Machine Learning

# Why do we optimize?

Optimization is useful because it permits us to solve many problems. In general, the optimization allows the saving of resources, for example the battery or the communication in the Iot context.

To give an example, there is a node deployed in some place, it could be ubicated in a city as in a mountain village, in any case the purpose is trying to save energy in term of battery and communication.

How can we do this? We know that a node has four states:

* Carrier sense, the state where the node is visible to the others;
* Receive, in this state the node is receiving data;
* Transmission, here the node is sending data to others;
* Sleep, a state where the node is doing nothing.

The first three states consume 20mAh, instead the last one consumes 0.1 mAh.

So, the purpose is to maximize the sleep time and minimize the others, obviously when it is possible.

Are there other methods to optimize this problem? We can minimize the delay, minimizing it in some problems is important, an example is the nuclear reactor, here the delay is important!

## Convex optimization

Returning to our example, minimizing the delay or maximizing the save of energy will give different results, can we find a balanced solution? Yes, we can find a solution that maximizes the save of energy and, at the same time, minimizes the delay!

This type of optimization is called non-linear or convex.

To explain the concept better, we can imagine that the delay is a function of the energy, so we can represent it in a hyperbole.

<grafico iperbole delay energia>

In the graph, the point p represents the optimal solution on the first problem, instead the point a, represents the optimal solution of the second one.

And in the middle? There are other solution for the same problems!

## Game theory

The game theory could give us a hand for finding the best possible solution, but how does it works? Essentially the game theory is the study of the mathematical interactional models used by rational agent. Precisely, the game theory explain, in base of the behavior of the agents, how we can find the best solution.

How can a game be? A game can be:

* collaborative: all the agents have to collaborate with a sharing of information, the purpose is to find the best possible solution;
* competitive: the agents will compete against the others, so each one will try to maximize their solution, adapting their behavior to the situation.

In the previous example, we can do an agent system where an agent will try to maximize the save of energy and another one will minimize the delay.

# Convex optimization

What is the difference between the non-linear optimization and the convex one?

To explain this, we need an example: we imagine to have a function f(x) like in the image on the right, the purpose is to find the minimum.

<graph on non linear function>

How we can do this? We start from a point x0 (chosen randomly) and we descent until it is possible. When it is not possible to descend anymore, we found a point x1, it is a minimum value! However, x1 is a local minimum and not the global one, so we can retry using another point (like x0’ in the photo) or keep this one.

What happen if we apply the same reasoning with a convex function? Before to answer the question, we have to explain what is a convex function: It is a function with the shape of a parabola. Thank to this shape, The convex functions have only one minimum value so we can always find the global one!

<photo of a convex function>

Moreover, they exist some optimized algorithm allowing to find the global minimum!

In convex optimization problem, we have to identify some convex sets and functions, also we can solve the dual problem and apply the solution to the previous one, it is practically the same.

## Vector space

What is the vector space? It is a space of objects called vectors, they are closed under the finite vector addition and the scalar multiplication. With vectors we can manipulate convex spaces and functions easily.

### Properties

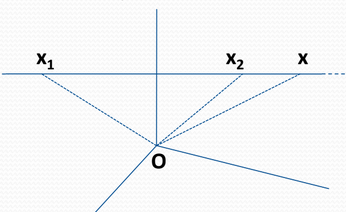
* Commutativity: doing x+y or y+x is equal!
* Associativity: With the vectors x, y and z, it is not important which couple of vectors we sum before. This property works with the scalar multiplication too!
* Identity: If we sum x with the vector 0, we get x again!
* Invertibility: we can invert a vector x negating each coordinate.
* Distributivity: if we have (s+t)\*x where x is a vector and s and t are scalars, we can get the same result multiplicating x with each scalar. We can do more or less the same reasoning with s\*(x+y).

## Affine space

An affine space is essentially a vector space with a variable more called affine coordinate, it is useful for conversions. In fact, operations over to points form a vector and a point-vector operation for a point.

In this way, we can define points and vectors with the same structure.

Mathematically, an affine space A is map function f:AxV→A where V is the vector space. The map function f has the properties of identity, associativity and uniqueness.

This is and example:

Given the graph on the right, the purpose is to calculate the vector , how can we do?

We have a point X1, so we can use it:

Also we have the point X2 and it is in the line between X1 and X, so:

In other words, X is the linear combination between the points X1 and X2. So, in this way we can find all the points between the other two!

A set that respects these rules is called affine set.

### Demostration

We have n+1 points Xi, each one has a λi. What can we do? We know that:

So, we can take the last point and put it out from the sum:

After doing this, we multiply the sum and divide the content:

To simplify the things, we do a change of variable:

So, we can calculate a point y in this way:

### Conclusions

In conclusions, x is a point in A. It doesn’t manner how many point we have, we can find other ones using the linear combination.

### Exercise

Proof that a set of solution of the linear system Ax=b (where A is a n\*m real matrix and b is a real n-vector) is an affine set. How can we do this?

Considering S as the set of solutions of Ax=b, we imagine to have to points x and y in S, so we use these one to produce a new point s:

After we produced s, how do we proof it is a solutions of S? We do the following:

As=b is true if s in in S, so S is an affine set.

So, we apply the definition of affine set on As:

If this equation is true, S is an affine set.

## Convex sets

What is a convex set? A convex set is an affine set where 1 are between 0 and 1. So, if we have to point x1 and x2, the convex combination will produce all the points between these two! In fact:

if 1=1, 2=0 so x=x1. On the other hand, if 1=0, 2=1 and so x=x2.

## Examples of convex sets

### Convex Hull

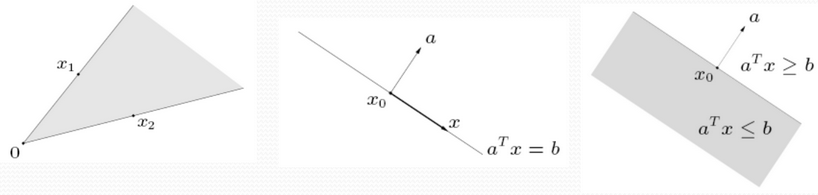
A convex hull is the smallest convex set of a given group of points, how can we get it? We have to combine linearly all the possible couples of points! Moreover, it is possible to get a convex hull from a non-convex set.

### Cones

A cone C is a set where for every x in C and Ө>=0, Ө\*x is a point in the cone. So, a convex cone C is a convex set where for every x1,x2 in C and Ө1, Ө2 >=0, x=Ө1\*x1 + Ө2\*x2 is a point in C.

### Hyperplanes and half spaces

An hyperplanes is an n-1 subspace of a space of n dimensions, so it is a set H formed by all the x where .

The variable a is a normal vector orthogonal to the hyperplane, is the complement vector of a and we get it in this way:

An half space is expressed like a set H that contains all the x where or .

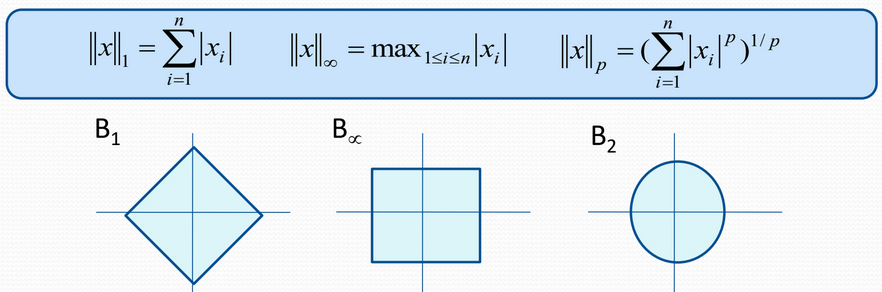
### Norms and inner product

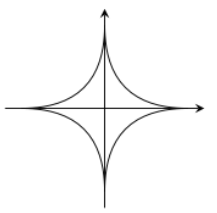
The norm is a function and it represents the length of a vector x, it is:

* non-negative, because f(x)>=0;
* definite, in fact f(x)=0 only if x=0;
* homogeneous, for every t in R, f(t\*x) is equals to t\*f(x)
* satisfied the triangle inequality.

An interest norm is the square norm of a vector x:

### Norm balls

Norm balls are convex sets defined as the sets of all the vector x where the n-norm of x is at most equal to given value r:

In base of the value of n, the norm ball has a different shape and a different method to calculate the norm, as we can see in the image on the right.

And what happens if n is between 0 and 1? The norm ball has a different shape!

Is it a convex set? Yes!

## Convex operations

Are there operations preserving the convexity? Yes, they are:

* the scalar multiplication, it create the same set but larger/smaller;
* the sum of convex sets;
* the cartesian product;
* the intersection of sets;
* the projection.

And the union of sets? In general no because it depends on the sets themselves.

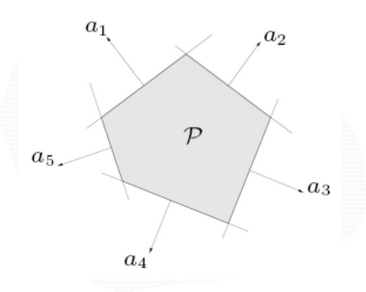
What about function?

A linear function f is convex because we can combine it linearly.

And other functions instead? In general, these functions are not convex!

## Polyhedra

A polyhedra is the intersection of a finite set of halfspaces and hyperspaces, we can resume it as the set P of points x where Ax<=b and Cx=d:

We can see a polyhedra like a set of constraint that all the points of the sets have to satisfy.

For example: given the following constraints:

* x1>=0
* x2>=0

We apply the definition:

a11\*x1+a12\*x2<=0

a21\*x1+a22\*x2<=0

Resolving this disequation give us these results:

* a11\*x1=-1
* a12\*x2=0
* a21\*x1=0
* a22\*x2=0

So, every point in the set has to satisfy these equations.

## Separating hyperplane theorem

The separating hyperplane theorem says that, given two separated convex sets C1 and C2, there is a hyperplane H that separates these sets. How can we find it? we draw a segment that connects the sets, the hyperplane is perpendicular to it.

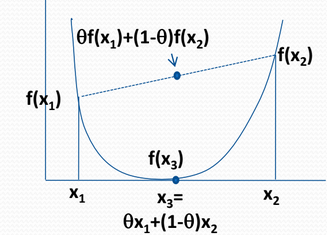
This theorem works only if the two sets are convex and separated, in fact if one of these sets is concave, we can’t find an hyperplane the separate them.

## Supporting hyperplane theorem

The supporting hyperplane theorem says that, given a convex set C, the is a hyperplane H that is tangent to the set.

Also in this case, this theorem doesn’t work with concave set because there are some points where the hyperplane is not tangent.

# Convex function and their domain

A convex function is a function that respects the following conditions:

* Each x is in the domain of f (strictly contained in ) , that is a convex set;
* for each a between 0 and 1,

So, each x in in the line between x1 and x2.

In general, if a function f is convex, the inverse function -f is concave. All the affine functions like f(x)=Ax+b are both convex and concave.

### Linear and affine functions

A linear function is a polynomial, for example f(x)=a\*x+b and it is a mapping between two vector spaces preserving addition and scalar multiplication.

An affine function is a composition of a translation and a linear map. In fact, f(x)=Ax is a linear function and also linear even through the matrix A, instead f(x)=Ax+b is affine but not linear.

### Other things about functions

A function f is differentiable if its domain is open and there exist the partial derivative at each point:

, it is a vector!

A function f is twice differentiable if its domain is open and the Hessian exists in each point:

, it is a specular matrix!

A symmetric matrix A is positive semi-definite if and only if for each x, .

We can check if a matrix is positive semi-definite in this way:

* all eigenvalues of A are >=0;
* All leading principal minors have positive or equal to cero determinants.

For negative semidefinite matrix, we can do a dual reasoning, but in the check:

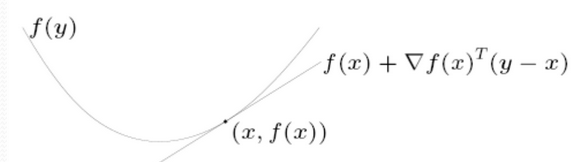
* All eigenvalues of A are <=0.
* All leading odd principal minors have negative or equal to cero determinants and all even principal minors have positive or equal to cero determinants.

### How can we test the convexity in functions?

1. Applying the definition of convexity on the function, for example:

,

1. First order condition: given a differentiable function f, it is convex if:

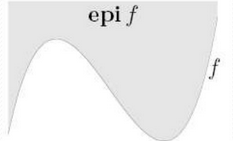
* its domain is a convex set;
* we can define an hyperplane supporting the function:

So, we can say that this approximation of the function f is a global underestimator.

1. Second order condition: Given a twice differentiable function f, it is convex if:

* its domain is convex;
* the Hessian is a positive semi-definite matrix.

1. Epigraphs: Given the graph of a function f defined in this way:

dom(f)}

The epigraph of f is defined as follow:

Also there the hypograph, it is literally an epigraph but f(x)>=t:

So, we can conclude that:

* a function f is convex if and only if its epigraph is a convex set;
* f is concave is its hypograph is a convex set.

## Sub-level sets

An a-sublevel set of a function f is defined as the set of points x in the domain of f such that f(x) is at most equals to a:

So, we define an a-level set in case of equality. A sublevel set if a convex function is convex for any a.

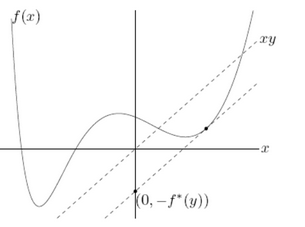
## Jensen inequality

The Jersen inequality is the following:

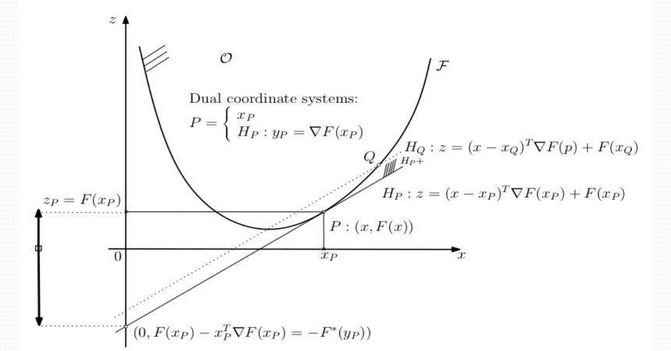
it could be extended to k points with the same rules.

In case of convex sets, this inequality exendes the infinite sums, integrals and expectations:

## Conjugate function

Given a function f, the conjugate function f\* is defined as the following:

The conjugate function f\* is convex and:

* it can be interpreted as the negative of the y-intercept of the tangent line of the function f sloping y. So, it is the largest affine function below f with the largest intercept;
* it can also be interpreted as the encoding of the convex hull of epi(f) in terms of supporting hyperplanes.

if the function f is convex and differentiable, we can apply the Legendre transform: a transform where the x value attaining the maximum satisfies .

So we can say that:

f\*\*=f if and only if f is convex and closed, i.e. epi(f) is closed.

The idea of the conjugation is to encode a function in another way, so we can express it with a different system of coordinates. Moreover, the conjugate allow to calculate the lagrangian function very easily and fast.

Example:

We calculate the derivative of f:

## Composition of functions

Give a function , and another we can get such that . So, the domain of f is the following:

dom(h)}

The composition results useful when a function f is complicated and we have to say if it is convex or not, in fact:

* f is convex if h is convex and non-decreasing and g is convex;
* f is convex if h is convex and non-increasing and g is concave;
* f is convex if h is a non-decreasing concave function and g is concave;
* with a convex g and a non-increasing concave h, we have a concave f.

## Vector composition

The vector composition is a particular composition where the functions take vectors as input.

## Quasi-Convex functions

A function f is quasi convex when its domain and every sublevel set are convex. So, a function f is quasiconcave when -f is quasi convex. If f is both quasiconvex and quasiconcave, it is quasilinear. Moreover, the linear combination of a quasi convex function is at most the max between the two extreme points:

So, if f is quasi concave:

This is the Jersen’s inequation for quasiconvex (or quasi concave) functions.

# Non linear optimization

A non linear optimization problem is defined using a standard:

minimize f0(x)

subject to fi(x)<=0 for i=0 to m

hi(x)=0 for i=0 to p

In the standard form, x is a real n-vector and . Moreover, we can consider the “subject to” section as a set of constraints that every point x have to respect.

When m=p=0, the problem is unconstrained.

## What is a feasible set?

A feasible set is the set of points x respecting the constraints in a optimization problem. His domain D is the intersection of the domain of f0 and the ones of every fi and hi:

The optimal value p\* is the minimum value in the feasible set:

In case of p\*=, the problem is unfeasible, instead it is unbounded below if there are feasible points xk (with k belonging to ) such that f(xk) belongs to , so p\*=.

The optimal point x\* is the point of the feasible set that minimize f0, in other words x\* permits to get p\*:

In general p\* is in the convex hull of a convex set, however it is not important to get x\* in the machine learning context, the purpose is to be closer to this point!

Also we have to pay attention in the function approximation because a little change of p\* can change everything!

A point x is locally optimal if there is an R>0 such that:

Moreover:

* the i-th inequality fi(x)<=0 is active if x is feasible and fi(x)=0;
* the same inequality is inactive if x is feasible and fi(x)>=0;
* The equalities are always active!

### What if do we try to minimize a constant?

Given the problem on the right, what can we get if we try to solve it?

It get that the optimal solution is always p\*=0 if x is a feasible point, instead p= if there aren’t feasible points.

### How to convert a problem in the standard form

Given a problem in this form:

minimize f0(x)

subject that u<=x<=l

We split the inequality:

minimize f0(x)

subject that

u-x<=0

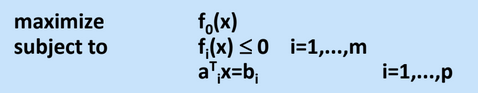
l-x<=0

### Equivalence of problems

Two problems are equivalent if we can get the solution of the second problem from the first one and vice versa.

# Convex optimization problems for non linear functions

Given a problem in the standard form, we can say the following things:

* f0(x) must be convex, such as every fi(x) function in the inequality constraints!
* every hi(x) in equality constraints must be affine;
* The feasible set of the problem is convex, so the domain D is convex.

We the function f0 results concave, the problem is a concave optimization, so we have to maximize it. As we can see in the problem on the right, every fi(x) function remain convex.

### Demonstration

For the first order condition, we know that:

Given a feasible set X, a point x is optimal if it respects the following conditions if and only if:

* it is a point of X;
* for every y in X,

Geometrically )!=0 means that defines a supporting hyperplane to the feasible set of the point x.

# Network Utility maximization

Given a network as in the photo on the right, we can say the following things:

* The network has a set of links L, each link Li has capacity Ci;

We define some routes Si as we can see in the photo, they are subsets of L.

What is the best optimal bandwidth allocation?

To say this, we define the function Ur(xr) as the utility function from transmitting on route r at rate xr.

If Ur(xr)=log(xr), we have a concave function, so we have to maximize it!

Ur(xr) is the utility function at a single rate, what can we do to get the complessive utility?

We define a function , it is concave and so we have to maximize it!

About the constraints instead?

The constraints are inequations about the capacities Ci, for instance in the photo:

* Link L1: x0+x2<=C1
* Link L2: x0+x1<=C2
* Link L3: 0<=C3
* Link L4: 0<=C4
* Link L5: x2<=C5

So we can resume them in a matric A, a vector x and another vector C, so:

<matrix of constraints>

And with wireless nodes? In case of wireless nodes, the reasoning is more or less the same but we have to add some constraints about the allocation. In fact we have to calculate the size of each buffer and verify if each node can contain the packet.

## Example: power control for cellular communication

We have to maximize the Signal Interference Ratio (SIR) of a user i, we can calculate this with the following formula:

In the formula, Pi is the power of the receiver i, Gik is the gain between i and k and is the noise of i.

This value must be at least equals to the minimum SIR i. Also every Power Pi must be between its minimum and maximum.

These two are the constraints of the problem, what is the function f0 insteas?

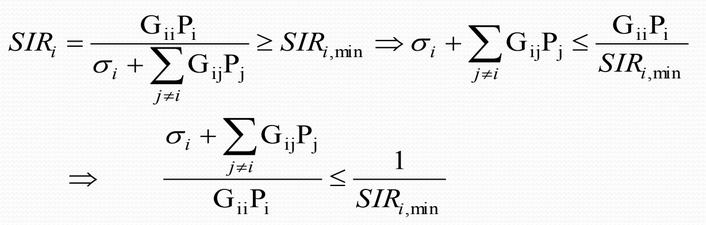
It is the function SIR i(P), it is a concave function and so we have to maximize it!

For CDMA systems, we need another constraint: the product power-gain between transmitter and receinver must be equals:

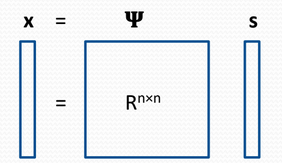
However, the maximization of SIR i is not a convex optimization problem, how can we do? We can convert it to a geometric problem (that is not convex) and convert this one to a convex problem.

So:

* we have to optimize 1/SIR i;
* the constraints are practically the same, we have only to resystem them;

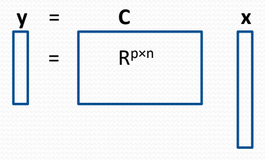


## Example: compressive sensing

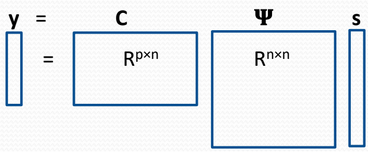
We have to compress a 100x100 image, what can we do?

* We vectorize the image;
* We transform the vector using Fourier;
* ReTransform the vector and reconverting it in an image;

We can write it as the calculus of a vector x given a matrix ψ and a vector s.

The new representation is called k-sparse if the vector s has only k-non zero elements, so it can be compressed in a k-vector if k<<n.

However this method is lossy, is there another way?

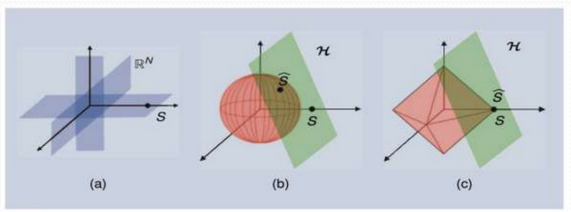
We can choose some matrix allowing us to calculate a vector y in base of a number p<<n that is the number of samples we take from the image.

From the definition of x, we can say this:

From this formula, we can assume that, with p samples of the signal x using a matrix C, we can resume the matrix C and into a matrix ф, their composition.

So, our purpose is to find a matrix ф that produced a k-sparse vector s and, from this, recover the vector x, How can we do that?



In the first problem we have to minimize the 0-norm of s, however the norm of a k-sparse vector is equal to 0 and the optimization problem is non-convex and NP-Hard. In front of this, we try to minimize the 2-norm of s, but it does not produce a sparse vector, so the 1-norm results a way to get a convex sparse representation. ղη

How can we choose p? We use the Restricted isometric property:

where c is a small constant, k is the k-sparse coefficient and N the length of the vector.

# Sensor selection

The sensor selection is a problem similar to Compressing sensing but with some differences. We have m measurements (each one with n feature) corrupted by an additive noise . We have to choose k measurement the resume the m ones.

We can solve this problem using the linear regression:

In the equation ai is a vector characterizing the measurements. So we can write it in form of vector:

And from here, we can find the optimal point x\*:

What is the idea? We know that the estimation error is the difference between a point x respect to the optimal one, so we can define the η-confidence ellipsoid of this difference, we’ll use it to calculate the true value.

What is the η-ellipsoid of a point p? it is a set containing p with the η% of probability!

A scalar quantity of the η-ellipsoid is its volume, we can calculate it with this formula:

We can simplify the calculus in this way:

Our purpose is to minimize the volume or maximize the logarithm of the determinant.

In poor word, the sensor selection problem consists in choosing k measurements that resume the m ones. So the problem is in this form:

maximize

subject to

var S contained in {1,...,m}

However, S is not a convex set because we cannot combine it linearly, how can we relax the problem? We introduce a variable z in {0,1}, then we rewrite the problem:

maximize

subject to

var zi={0,1}

The problem remain concave because of z, that is not convex, We solve this ranking the solutions:

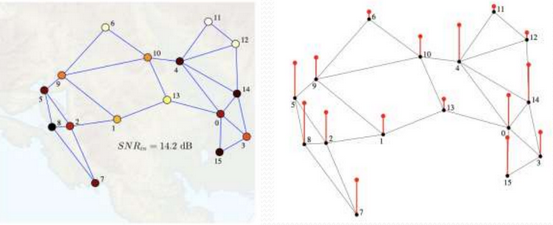
maximize

subject to

var 0<=zi<=1 for each i=1 to m

## Graph Signal Processing (GSP)

The GSP is a problem with the following idea: given a network of n nodes, the purpose is to build a graph whose relationships allow us to build a Lapracian matrix nxn L.

We can get L from the difference between W and D, where W is the weight matrix with the costs of each edge and D is the degree matrix, we this one in this way:

After we got L, we can perform GPS with operations like Fourier analysis, filterning, etc.

How can we calculate L?

For instance, using Fourier, we calculate L in this way:

What is the problem? The matrix U depends from L and this one depends from W.

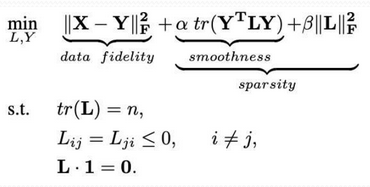
A particular case is the linear graph, in this way the Fourier transform on it is like in a normal signal, and in a normal case? How can we do it?

To solve this problem, we have to calculate the L or W and there is three ways:

1. We use a distance function f(d) to calculate the matrix W, from here we calculate D, L and then U. This is the most used method but it is not very efficient;
2. We use the Graphical Lasso: assuming we have a Gaussian Markov Random FIeld (GMRF), we have to calculate a graph with n node and n measurements, one for each node. We can resume this graph into a matrix ∑, so we have to estimate a sparse precision matrix θ, how? we try to calculate the covariance from ∑, then we invert it:

This is a great result! Even though it is not the best method to estimate L.

1. We estimate directly L imposing the criteria of sparsity and smoothness:

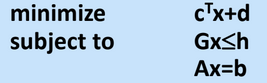


In the problem:

* a\*tr(...) is the smoothness, how we go from a node to another one;
* tr(L)=n avoid trivial solutions, we don’t want these!
* Lij=Lji<=0 indicates that the matrix L is negative semi definite and it is specular.

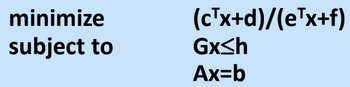
## Linear optimization problems

They are problems where we have to optimize a linear f0, they are convex in every form.



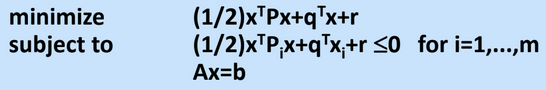
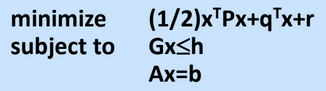
## Linear fractional problems

In this case we have to optimize a problem where the variable x is in the denominator of a fraction, we can convert it into a classical linear problem.



## Quadratic optimization problems

They are problems where the variable x is squared. There is a variant called quadratically constrained QP where the constraints are quadratic.

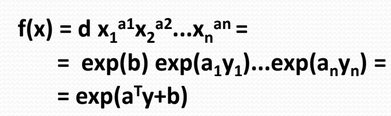


## Geometric optimization problems

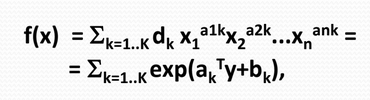
They are problems where the function f0 is defined as a set of monomials and/or posynomials.

With this type of problem, we have resolved the Compressive sensing! However, geometric problems are not convex optimization problems, but we can convert them! How can we do this?

We have to invert the function f0 and all the constraints:



Where b=log(d). And in case of posynomials?



Where is a vector of yi, each Also is a vector and bk=log(dk).

# Duality

What is a dual problem? Give a problem p, the dual problem d is a problem with the same solution of p but we optimize a different function correlated with the first one.

Which are the steps to find the dual problem?

At first we have to calculate the lagrangian, a function L that take three parameters and result all the problem in this way:

In the formula, >0 an v>0 are vectors of lagrangian multiplier respectively associated to the inequalities and the equalities.

Why do we use lagrangian multipliers? To ask this question, we assume to maximize a function f(x,y) subject to a constraint g(x,y)=c, which is the optimal point? It is (x\*,y\*) and it satisfies g(x,y), so imaging to have a hyperplane f(x,y)=x+y=k. If k=0, the hyperplane will be a line passing for (0,0), and when k>0? it will pass in the point (0,k).

We can resume f(x,y)=k with a gradient, here we can see that the constraint g has the same orientation of f but it is smaller, so we rescale it with a value :

After we calculate the lagrangian L, we compute the dual function of L as the minimum of L in base of and v:

Is a concave function? Yes because it is a minimum of a family of affine functions (that are concave).

Moreover, is at least or at most p\*? It is at most p\* because the input is the optimal point x\*, so:

* it satisfies the equalities, every hi(x) is zero;
* it satisfies the inequalities that produce a value that is at most zero;
* f0(x\*) is equal to p\*.

So, our interest is to find a couple ( ) i.e. q() =d\* where d\* is a lower bound of the optimal value p\*, so a couple that allows us to go as close as possible to the optimal value.

We can define the problem in this way:

maximize q()

s.t. >=0

var

There two types of duality:

* the strong duality tells that the value d\* must be equals to p\*;
* In the weak one instead, d\* is strongly minor to p\*

## Strong duality

With strong duality we can solve both the primary problem and the dual one, so it could be useful in the Machine Learning context.

To do an example: we have to solve this system of linear equations:

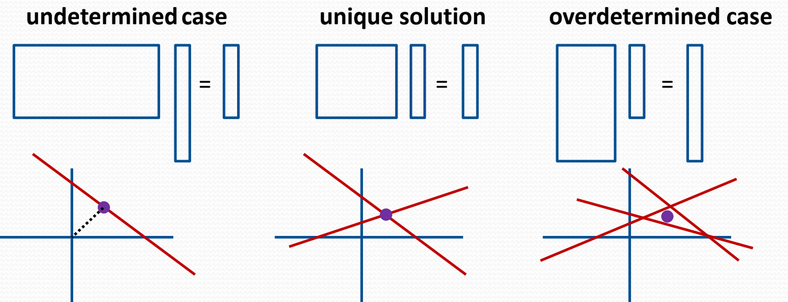
So, there are three possible scenarios:

* the underdetermine case compare when m<n, so there are more variable than equations. In this case we can always find an optimal solution, how? We have to find the x that minimize the 2-norm.

| Primary problem | Dual problem |
| --- | --- |
| minimize  subject to  var x | maximize  var v  Note: is the left pseudo-inverse. |

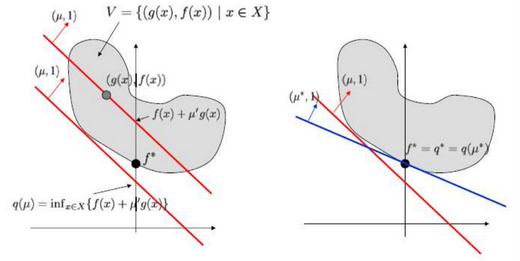
* When m=n, we have a unique solution because we have the same number of variables and equations:
* Instead, with m>n we have the over determined case, where there are no solutions. How can we do in this case? The best we can do is to find the closest point to all the equations:

| Primary problem | Gradient |
| --- | --- |
| minimize  var x | Note: is the right pseudo-inverse. |



## Weak duality

In which cases can we have weak duality? When d\* is a lower bound of p\*, so we can define the duality gap as the difference between these values:

Given one optimization problem and its dual, we consider f\* as the primal optimal, corresponding to the minimum vertical axis of all points on the left half plane. The dual value q(u) for a feasible u>=0 corresponds to the vertical intersection of all hyperplane supporting the problem from below. So the dual optimal q\* is th max intercept of all hyperplanes over all u>=0.

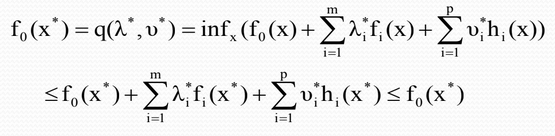
## Slater’s condition

The Slater’s condition tells that the strong duality is guaranteed the primal problem is convex and exists an x in the relative interior of X that satisfy the constraints, we can name x as Slater vector.

The relative interior of a set contains all points that are not in the edge of the set.

## KKT optimality condition

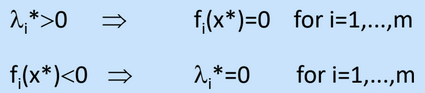
Assuming we have strong duality, we have that x\* is the optimal point of the primary problem and is the dual one. So we can say that:



We can conclude that the optimal point x\* minimize the Lagrangian, so its gradient vanishes.

### Complementary slackness property

For any optimal point x\* and its dual , in case of strong duality, all inequalities goes to zero. So:



In poor word the KKT conditions are:

* the respect of all primary constraints;
* the respect of all dual constraints;
* the complementary slackness;
* the vanish of the lagrangian gradient.

So in convexity we have strong duality if and only if we satisfy the KKT conditions.Instead in concavity we have KKT conditions if we have strong duality but the contrary is not true.

# Descents methods for unconstrained problems

In descent methods, our purpose is to find an optimal point x\* minimizing the function f0, so p\*=f0(x\*).

How can we do it? We do a sequential reasoning:

In poor words, the optimal k+1-th value is computed using the k-th optimal value, the step size t>=0 and the search direction d(k).

Precisely, what is d? it is a descent direction if it satisfies the following condition:

## Lemma 1

Given a function f, we are interested about its gradient, in fact it is a direction (not descent)! We can use the gradient of f to find a descent direction d in this way:

So the angle is less than 90°.

So, applying the definition of descent direction:

## Lemma 2

The gradient of the function f at the step k is a direction maximizing the rate of increase, so its opposite is a direction maximizing the rate of decrease.

## Lemma 3

Given a positive definite matrix A, for every x such that is different to 0, the direction is a descent direction, why? because this reason:

## Step of descent methods

Selecting the step size t, the simplest way is to choose a constant but it is not optimal. Another way is to choose minimizing f along the ray

In this case, the cost of the optimization problem could be high and so we have to approximate the method.

### Backtracking line search

TO do this, we have to choose a minimizing f along the ray, then given descent direction d(k), we continue to calculate a new value of t until it converges:

Precisely, what do we want to do with this search? We try to find the best hyperplane supporting the function f, the one allowing us to minimize better.

At the beginning, we have to choose a vector as starting point, it’s better is is feasible because it allows to find the minimum quickly.

When do we stop the iterations? There are several options:

* When the norm of the gradient of f is less than epsilon, in this case if the gradient is equals to 0, we have found a point satisfying the first order condition;
* When the improvements of function is less than epsilon:
* when the movement in points is less than epsilon:
* Other measures relative to the last two.

## Gradient Descent

The gradient descent is a particular descent method where the descent direction d is the negative gradient. So we stop iterating when the 2-norm of the gradient is less than epsilon.

## Steepest descent

The steepest descent is a descent method where the direction d make the derivative as negative as possible respecting |v|p.

If we use the 2-norm, d will be the negative gradient and so this method belongs to the classic gradient descent.

Other norms produce other steepest descents, for example we’ll get the coordinate descent if we use the 1-norm.

## How can we measure a descent?

We use a condition number k, it measures how a function f change in proportion to small changes of x. A problem is well-conditioned if k is small, instead if it is large, the problem is ill-conditioned.

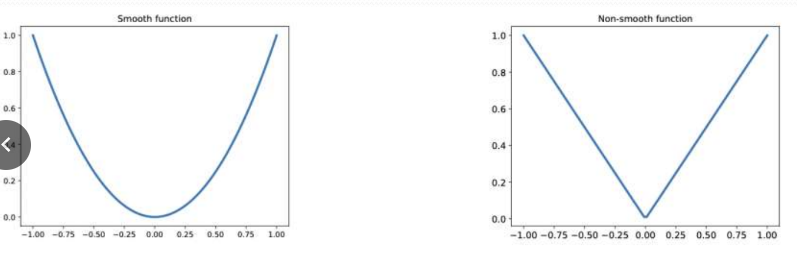
## Newton’s method

The Newton’s method is a gradient descent method where:

* we start choosing a x0 value;
* We define the Newton decrement in this way:
* We define the Newton step:
* When do we stop? when !

It is a good estimate when x is close to x\* because Newton step minimize with a second-order approximation:

## Smooth v non-smooth

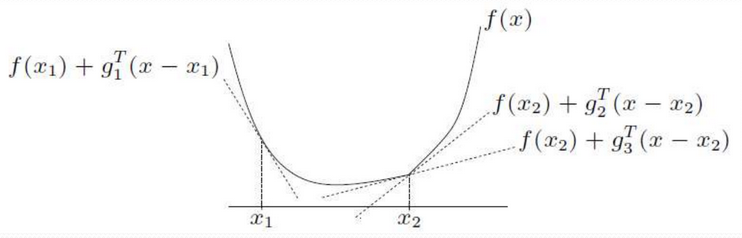
If the function we optimize is smooth, we can continuously differentiate it. We can identity the smoothness using the Lipschitz constant L:

In case of non-smoothness, there are some points where the function is not differentiable, what can we do in these cases? We use subgradients!

## Subgradient methods

Given a not differentiable function f, what can we do to get the gradient? Considering the first-order condition, a function g is a subgradient of f if:

if the function f is convex, it exists at least a subgradient to each point in the relint of the domain. if f is also differentiable, the gradient of f is a subgradient at x.



## Is it possible to descend a subgradient?

Yes and there are little differences respect to the original:

In the formula:

* is a subgradient of f at ;
* t>=0 is the step size;
* k is the iteration time.

In case of -gk is not a descent direction of f at xk, we maintain an fbest keeping track of the lowest purpose function value.

## Adding equality constraints

What if we add equality constraints to our gradient descent? Essentially we can rewrite the problem in this way:

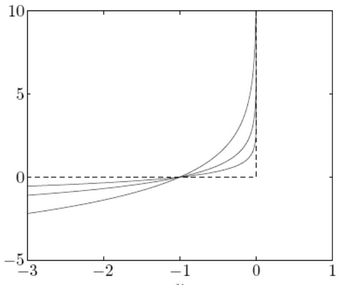
|  |  |
| --- | --- |
|  |  |

So, after choosing a initial value x0, we have to compute delta x nt in order to solve the linear equation system. From this we calculate lambda.

## And for convex optimization problems?

In this case we can rewrite the problem in this way:

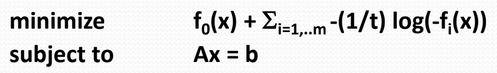
|  |  |
| --- | --- |

Where I(u) is the indicator function, it is 0 if u<=0 and infinite otherwise.

However the function I(u) is not differentiable, so we cannot apply the Newton method, what can we do?

We can approximate I(u) to (-1/t)\*log(-u), in this case I is convex and non-decreasing for t, u >0.

So we rewrite the problem in this way:



From here, we can solve the problem as a linear system.

# Machine learning

The machine learning is a branch of artificial intelligence where a computer learn a model starting with a certain amount of data, it will be used to make predictions.

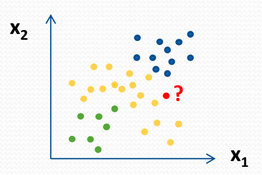
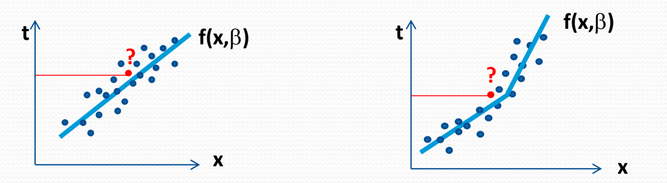
There different approaches of machine learning:

* supervised: the algorithm know the output of data, so it will try to find a mapping between input and output;
* unsupervised: the algorithm know only the input data, so it compute the output basing on the patterns;
* reinforcement: there are some agent that interact with the environment, they have to learn how to move in it in order to solve the problem.

## Supervised learning

There are two types of supervised learning:

* the classification: we have to compute the most likely label basing on the data features;
* the regression: we have to reduce the data in a real number starting from the features.



## Unsupervised learning

The most common unsupervised method is clustering. In poor words, clustering is a way to compute a certain number of points representing a subset of data, so we’ll use them when we have to do some predictions of new data.

## An example

Given a wireless sensor network measuring some values in the air, our purpose is to predict the amount of O3, how can we do this? we have to consider every measure as a coordinate xi, then we multiply it with a value bi, it is an angular coefficient.

What is the value of each bi? We have to compute it from the data!

## Loss and risk function

Our purpose in machine learning is to predict a value the is as close as the real one, so what can we do? We define the following funxtion:

* the loss function estimate how a model approximates a prediction, there is not a best loss but some of there are better than others.
* the risk function is the expected value of the loss function, it indicates if the prediction respect the distribution of probability. In the most of the cases we haven’t this distribution, so we use the empirical risk function.



## Multiple linear regression

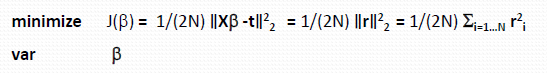
The multiple linear regression is a machine learning method where we have to create a linear function starting from a dataset.

How can we create a function? Our purpose is to compute some value bi, one for each dimension plus one we use as an offset.

So, the array b forms an hyperplane f(x,b)=dot(x,b), we found it fitting the polynomial to the training set.

Considering a quadratic loss function, that b has to minimize, we define the residual vector r as the vector of differences between the predicted value x\*b and the true value t.

In order to find the closest vector b to the true value t, we have to minimize the residual, so we have to oprimize this problem:

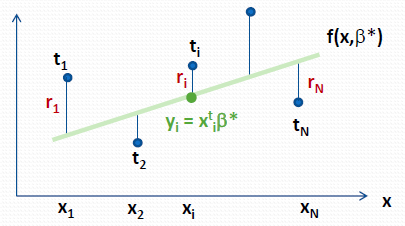


We don’t want that the loss function depends on the number of samples, so we approximate it, the problem became the following:



This is an unconstrained convex optimization problem on b, so we get the solution putting the gradient of J(b) equal to zero, so we invert the formula to compute b\*:





## Geometric interpretation

Considering the image on the right, t is the vector of true value, the green rows are the column of training set X and the rectangle is a subset X. Our purpose is to find a projection of the vector t in the subspace X minimizing the residuals r.

We define the projection of a vector y on a hyperplane X in this way:

Considering the vector geometry, Xb\* is the orthogonal projection of t on the subspace x where:

The prediction y will be computed on a new subspace Z (the test set):

Because of we use a quadratic loss function, if we compute the hessian we should get a positive definite matrix, by the definition of convex optimization problem, if we apply any vector v on the Hessian H:

So the norms are positive if v is not zero, so H is positive definite and there is a unique global minimum in the problem.

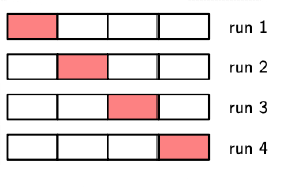
## How can we measure the accuracy of a model?

We can use:

* the Root mean square error (RMSE), it allow to compare different models for the same dataset, we can calculate it in this way:
* The coefficient of determination R^2 that measures the proportion of variability in t using X. This value is between 0 and 1 and, if it is close to this last number, there is much variability we can explain with the regression.
* The mean absolute error indicates the average absolute error.

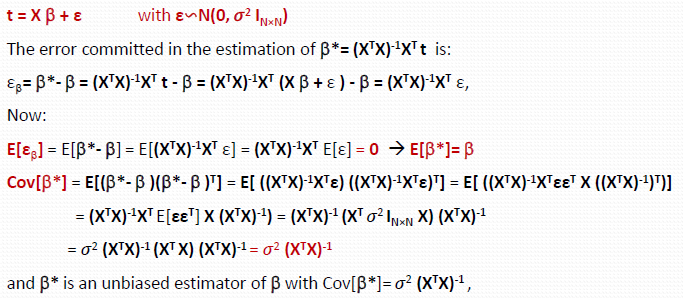
## Other important things

We don’t use all the dataset to create the model , we split it in some parts:

* the training set, that is the part we use to create the model and so compute the vector b. In this part we can use the measure of accuracy to evaluate the goodness of a model;
* the test set for testing the model we created previously and see how it is accurate;
* the cross-validation test, and important part of the dataset that result useful for the tuning of the hyperparameter.

Precisely, we use the cross validation when there are few data and it results difficult to divide them. So, we divide the dataset in K parts, one of these will become the test set, the others will form the training set. We do different run using a different chunk of data as test set every time, at the end we make the average of all the vectors b we got.

## Error of the estimation



# Bayesian multiple linear regression

In the case of Bayesian MLR, we have to compute the posterior probability using the Bayes formula.

So, we can conclude that:

## And in case of a very large matrix X?

In case of the matrix X is very large, the application of the gradient descent algorithm could be very difficult, in fact the matrix H^-1 could be ill-conditioned if we use the Newton’s method.

So, what we can do? In order to avoid numerical errors, we have to be careful about the convergence of the descent method, so we need to check the condition number of a matrix A, defined in this way:

If we use a 2-norm, and so , so a matrix is well conditioned if k is small, this is meaningful because it say the importance of the matrix size.

## Gradient descent for machine learning

In order to get the best vector b, we use the gradient descent algorithm on the problem we defined previously. So we compute the vector b in this way:

This algorithm is the least-mean-square algorithm (LMS) and it works in this way:

* we choose a vector b0;
* we iterate until there is no convergence.

The convergence is better if not all the points are used in each iteration, so there are several variation of this algorithm:

* stohastic LMS: it use one random point for each iteration;
* mini-batches LMS: it uses a subset of data points for each iteration;
* etcetera.

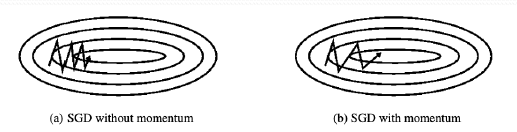
For a better optimization, we have to normalize the data with mean and standard deviation and shuffle it before each iteration, these steps allow us to avoid the bias.

## Using the momentum

What is the momentum? it is a sort of average of a small group of points, it allow a more directed descent and avoid the oscillations.

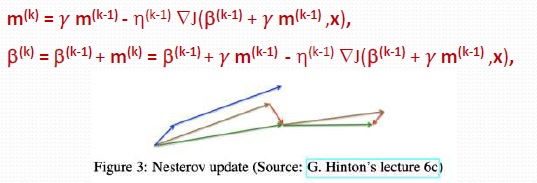
How can we compute the momentum? We use a weight γ, the momentum coefficient:

We can consider the moment as an exponential weighted average of the past gradients.



Are there other types of momentum? Yes, the Nesterov momentum is one of these! In the previous case, we compute first the gradient and the the jump, is there a way?

With the Nesterov momentum, first we make the jump using the previous momentum, then e make correction:



## Using the regularization

In order to evaluate a machine learning model, we need to know the following concepts:

* the bias is the difference between the prediction and the true value, the more this value is large, the more the prediction doesn’t represent the data, this is the reason why a large bias cause underfitting (bad representation);
* the variance is how much the model respect the mean in base of the training set, If this value is high, a small change in the training set produces large changes in the predictions and, for this reason, it can cause overfitting (no generalization).

Is there a way to have a balance? Yes, the regularization! It is a technique to prevent the overfitting, how can we do it?

Because of the number P of feature cause overfitting if it is more than the number of point N, a solution is to decrease this number with the subset selection.

So, how does the subset selection algorithm work?

* First we take one feature, we solve the model an take some statistics;
* We do the same things with two features;
* Same things but with three features;
* Etcetera.

In base on the results we get, we choose the best subset of features. This algorithm gives great solution but its complexity is too high, precisely O(2^n-1) int time.

Can we do better?

We can check the best group of feature every time, then we try with a feature more, etc.

In this way the complexity belongs to O(p^2).

Moreover, we add a penalty in the cost function J to do the regularization better:

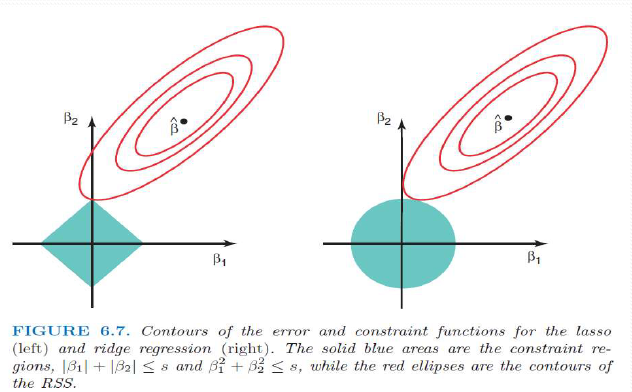


In base on the value of q, we have different kinds of regularization, in fact:

* we have a ridge regression if we qis equals to 2;
* if q is equals to 1, we have a lasso.

What is the difference between Lasso and Ridge regression?

In the Lasso there is only a feasible point we can optimize, it is in the vertex on the square we can see in the image below. Considering the Ridge regression instead, we have more feasible points because the ellipsoid is tangent with the border and not only with a vertex.



## Single value decomposition for MLR

We know that the SVD is a generalization of the eigendecomposition of a rectangular matrix, Considering a NxM matrix, we know that:

if , then:

In case of ridge-regression, we can find the prediction in this way:



We can define zj=dj\*uj as the jth principal componente of the matric X, here uj is the jth normalized principal component with variance dj/N. If we do a PCA, we can observe that the largest singular values provides the largest sample variance of all the combination of features (the column of X). SO, the ridge-regression shrinks the features that have a small variance, then it shrinks the ones explaining less the data.

# Logistic regression

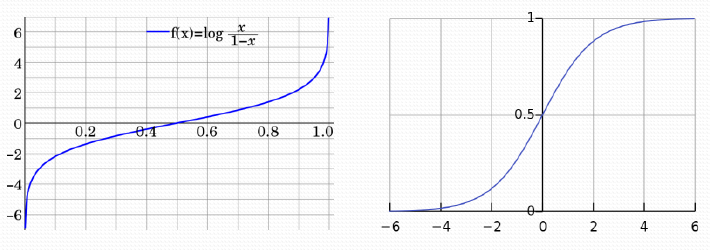
The logistic regression is a binary classification method (WTF). This algorithm assumes that we have a vector of measurements x, each one is assigned to a binary label t. The purpose is to get a linear relationship between variables and log-odds for the label 1.

We can represent the logistic regression as an s-curve called sigmoid, how can we make predictions?

We make the model computing the most-likely class.

So, considering the probability p, we can define the logit function f(p) in this way:

Because of the logit function return the point x, we can define the sigmoid function as its inverse:



To make the model we can consider a vector b of size P+1 where P is the number of features, how can we define the probability of the first class? In this way:

So, we can define the logit function as a polynomial where the vector b contains all the coefficients:

The class assigned to a point is the one which has the maximum likelihood for a vector b. After getting the best vector b, we use it to make predictions, precisely computing the probability for the first class:

There is an extension of the logistic regression allowing the multiclass classification, however it is not used because of better alternatives like the Linear Discriminant Analysis.

## k Nearest Neighbors (k-NN)

The k-NN is a useful method for classification and regression allowing to predict the best k-values more similar to a point. Considering N points x with P features and N labels t, the idea of this algorithm is to take the best k points calculating the distance (like the euclidean one) from the input.

After taking the k points, what do we do?

* in case of regression, we compute the average;
* in case of classification, we take the most common.

This algorithm works better when k is odd, also it must be balanced because a small k belongs to underfitting and a high value causes overfitting.

This algorithm is very simple and takes few assumptions, however it could be huge in time if the training set or the number of features is high.

# Kernel regression

The kernel regression is an extension of MLR for non-linear model. The idea of this approach is to work with a dataset of point X like the normal linear regression but with a different vision: we work basing on the features and not on the data. How can we do this?

We define a mapping converting a point x iof the data space in a point of the feature space, this mapping add d dimensions to the point, this value can belong to infinite. THe purpose of this mapping is to find a way to separate linearly the data, so we have to find a function allowing us to find a function that:

* separate the data in the feature space linearly;
* separate the same data in the data space in a non-linear way.

This function is the kernel and it is defined as following:

## Regression model

The model for the kernel regression uses a different cost function, here we have ot consider that f(b,x) depends on . So, because of we have N points x, we can define the design matrix Ф as a vector of size N containing every .

So, we can express the cost function J(b) in this way:

Calculating the gradient of J respect to b and putting it to 0, we can get the value of b:





So, we can redefine J(b) as following:



In the formula above, is the Gramm matrix, a NxN matrix containing all the points Kij=k(xi,xj), we can consider it as the scalar product of and .

Considering the dual problem, we can observe that J(b)=J(a), so we can use the last one to work in the feature space.

We use J(a) because it is the dual formulation for the least-square problem in the MLR, this is possible thanks to the kernel functions.

We can represent the Gramm matrix K as a function and its points, this is useful when we have infinite vectors, in fact we can’t compute the scalar product in these cases.

So, we can directly use the data points in the feature space! This is the kernel trick!

## Why do we use the dual problem?

We use the dual problem instead of the original one because, even through it is less efficient when N>>P, it is expressed in terms of kernel functions, so we can solve problems with a large N.

## Estimation of data

To estimate the data, first we have to compute the best vector , then we compute the best vector a in this way:

Considering a new point x0, is the estimation in the feature space, However we want to use the matrix K, How can we do?

So:

where k(x) is a vector of k(xi,x)

## What is a kernel?

A kernel is a function k(x,y) defined as we saw previously, it is positive definite if:

* it is symmetric, so k(x,y)=k(y,x);
* K is a positive semi-definite matrix, so the sum of each point is more or equal to zero.

Some examples of kernel are:

* the linear kernel: ;
* the gaussian kernel:

Also we can build our kernels, how? We combine simple ones!

## Spaces in kernel regression

To understand the spaces in the kernel regression, we start from the vector space, it is a set of objects called vectors that is closed under addition and scalar multiplication.

From the vector space we define the normed space, a vector space where the is a norm function for each vector f, it satisfies the following things:

* ;
* ;
* .

A norm function induces a distance, so a normed space is a metric space and a topological vector space.

Now we talk about the Banach space, it is a complete normed space, it means that every Cauchy sequence converges respecting the norm. This convergence is important because it allows us to get the optimal value.

A pre-Hilbert space is a vector space V over a field F where the inner product is defined. The inner product is like a mapping that satisfies the following things:

* <f,f> is a real value and it is positive-definite;
* <f,f> is zero only if f is zero;
* <f,g> is equals to <g,f>;
* <af1+bf2,g>=a<f1,g>+b<f2,g>.

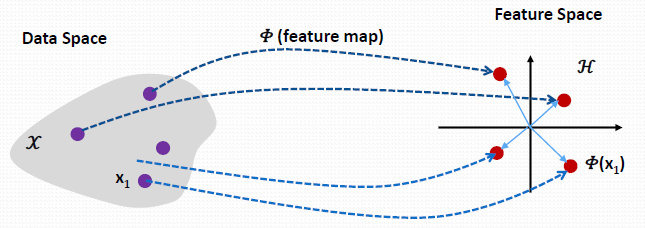
Because of the inner product induces a norm, a pre-hilbert space is a normed space.

A hilbert space is a complete pre-hilbert space, so it is a Banach space.

## Moore-Aronszaj Therem

Given a kernel k, it is positive definite on a set X if and only if it exists an Hilbert space H and a mapping from X to H:

so:



Now we consider as the set of all possible mappings, this is not a Hilbert space and so we take a subset, what can we do?

* We define a mapping;
* We define an inner-product;
* We add the completeness property limiting the Cauchy sequences in the norm;
* show if the inner product is equals to k(x,y), in this case k(x,y) is positive definite.

## What is the idea?

Given N points kxi, we generate a space .

Given f,g in G, we can combinate linearly these points with different hyperparameters:

So:

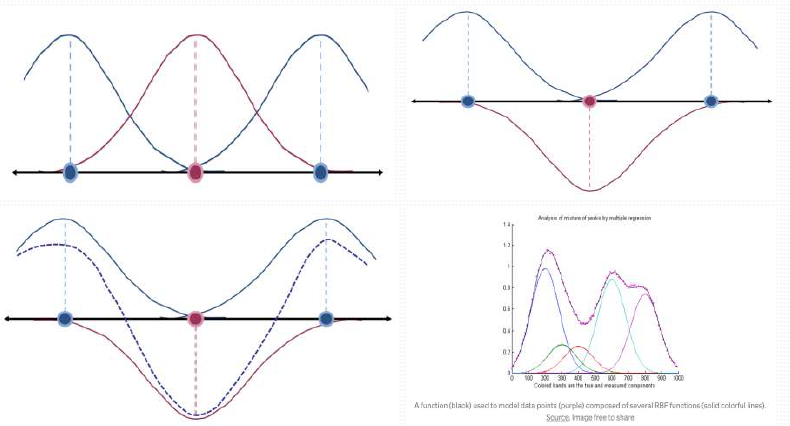
So we can define the inner product in this way:

Our solution is in a subspace, in this subspace we can find it with a linear regression.

Basing on this assumption, we can model the problem:

## Radial basis functions (RBF)

A RBF is a function depending of the distance of x from a fixed point a, the idea is that, if we have a radial function, the result will be a linear combination of some simpler functions.



## Gaussian processes

A gaussian process is an extension of the multivariate Gaussian distribution with infinite dimensions, what is the idea? GIven a dataset following a normal distribution with average μi and covariance Σ, we can consider Σ as the Gramm matrix, so we can do a kernel regression with that one.

Our purpose is to find a function f(xi) to estimate the true data, so we can compute the vector b from it:

Our estimation is gaussian distributed, so:

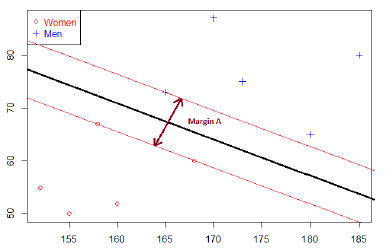
How can we make predictions? Given a new point xn+1, to estimate its tn+1, the idea is that where , so we compute the marginal probability P{t}, then P{t|tn+1} and P{tn+1|t} at the end:

;

How can we get Cn+1? In this way:

# Support vector machine

The support vector machine is a machine learning method following this idea: given some new measurements, the classification can be done in three ways:

* In case of data that are linearly separable, we found the hyperplane that divide these sets;
* In case data are separable but not linearly, we have to find a function that does it;
* in case on data non linear and non separable, we cannot classify, we can try using a kernel.

## Linearly separable data

Considering the first case, we have to find the best hyperplane separating the sets, how can we do this? We make an optimization problem and we solve it, using the KKT conditions.

Where is the problem?

The problem is in the data on the borders, in fact future predictions could influence them. In order to solve it, we use two symmetric margins, so we have to optimize this distance.

### Example

Given n training data xi and n true data ti that can be -1 and +1, how does the hyperplane separate?

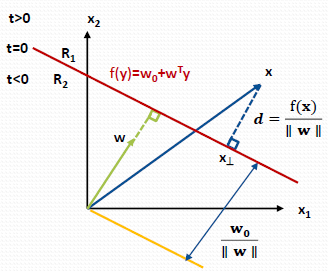
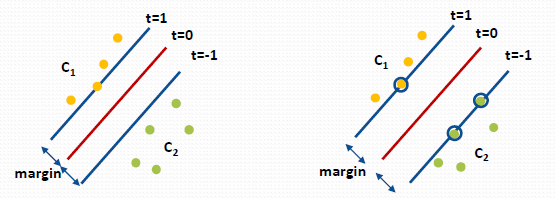
* if ti is -1, ;
* if ti is equals to 1, ;

So we can resume everything with this property:

So we can consider the function as the separating hyperplane.

Now how can we optimize the margins?

The margine are the perpendicular distance between the decision boundaries and the closest data point. The location of these boundaries depends on a subset of points called support vectors.



In order to define the optimization problem, we consider d as the distance between a point x to the margin, so the hyperplane is perpendicular with it and, for this reason, we can compute a vector u in this way:

And so, we can define the perpendicular distance d in this way:

So, these are our purpose:

* we want that all the points are well classified;
* we want that the minimize the distance.

So, we can define the optimization problem in this way:

In order to simplify the problem, we can rescale w and w0, so:

So, for the further points , this will be our new constraint.

Regarding the f0, we have to maximize 1/|w|, that is like to minimize , so we can rewrite the problem in this way:

In this way we get w\* and w0\*, we’ll use them for predictions.

## non overlapping classes

In case of non overlapping classes, we can solve this problem with the dual one, so we define the lagrangian:



a is a n-vector of lagrange multipliers.

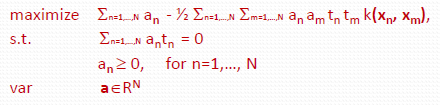
Setting the derivatives to 0 we got that:



Eliminating w0 and w from L(w0,w,a) we got that:



And so, we go the dual problem:



How is the prediction done? The support vector machine consider only the points on the margins for the prediction, we can find them computing an, in fact this value is more than zero for these vectors.

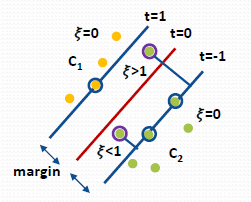
So, the algorithm works in this way:

* first we compute the an of the support vectors, solving the dual problem;
* then we compute w0 as the average of support vectors;
* finally we make the prediction using the function f(x) on the new point.

## Overlapping classes

When there is overlapping between classes, there could be data in the margins, what can we do in these cases?

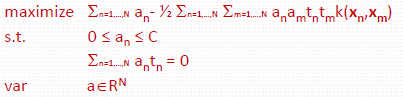
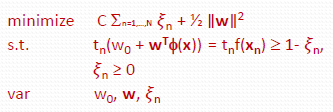
If we try to classify them, there could be misclassifications, instead if we use a linear kernel we can classify them correctly but with more complex one we can risk the overfitting.

In order to solve this problem, we introduce the slack variables, they are n-vectors 𝜉>0 defined as follow:

* if the point xi stays in its margin, it will be well-classified and sol 𝜉i=0;
* if the point xi is between the margins but in the correct side, 𝜉i will be between 0 (excluded) and 1;
* if the point xi is between the margins and in the wrong side, 𝜉i will be more than 1, so it behaves like a penalty because xi will be misclassified.

The introduction of the slack variables changes the optimization problem, in fact we give a soft margin and allow the training data to be misclassified. So, to maximize the margin we have to minimize the square norm of w and the sum of each 𝜉i, moreover we multiplied this last part with a constant C, it works as a regularization factor. If C is large, every 𝜉i will be small and the data will be separable, but with a cost: the overfitting.

So the optimization problem and the dual one become these:



Regarding the algorithm of resolution, it remains the same, with a difference:

* the points that contribute in the prediction have ai >0 and satisfy this:
* if ai<C, ui>0 and so 𝜉i=0 because of the KKT condition, this points lies of the margin;
* if ai=C, the point lie inside the margin and so 𝜉i is between o and 1 if it is classified correctly, in case of classification instead 𝜉i>1.

The support vector machine work for the binary classification, but we can modify it in order to support the multiclassification, there are severla options:

* one vs all: we build K separate SVM using a different C to distinguish the positive classe against the others K-1. In general it is an inconsistent method because the result is assigned multiple class simultaneously. Moreover the training data are unbalanced;
* one vs one: we build K\*(K-1)/2 binary SVM, it results always unconsistent and it is expensive with large Ks.
* other methods.

## Support vector regression

The problem of the support vector regression is to find a function f(x) with an ε-tube deviation from the target t.

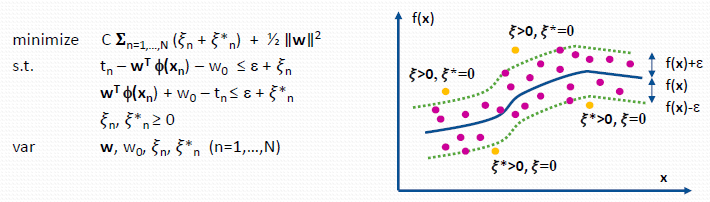
Considering , we can solve the system as an optimization problem, in fact we can compute the hyperplane and the margin ε from the target t.

But instead of using a quadratic problem, we use an ε-insensitive error function, defined as follow:

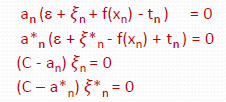
This is something like the red function in the image on the right.

Also in this case we use the Slack variables to account the unfeasible data, the meaning is:

* if ti>f(xi)+ε, 𝜉i>0;
* if ti>f(xi)-ε, 𝜉\*i>0.



From the KKT condition, we know that:



## Hyperparameters

The parameter C tells how much avoid the misclassification, so:

* if C is small, we have underfitting;
* if it is large, we have overfitting.

If we use the radial base function as kernel, we have to consider an hyperparameter gamma, it defines how fare the influence of the training point reaches.

With a low gamma, only the nearest point to the hyperplane will be considered for the compution of the separation line, instead we consider more points with a greater value.

Epsilon is the hyperparameter that define the epsilon tube and so the loss function. If the points are inside the loss function, they will be zero, if we can tolerate high errors, we can increase the epsilon band and viceversa.

## Ensemble methods

The idea of the ensemble method is to combine some methods to get a better classification or regression, how can we do it?

Considering n points of training set, the first purpose is to increase the number of training set:

* We can decide the size m;
* We can choose randomly m points from the principal training set;

A first idea is the committees, that has the following idea:

* we choosen models where we apply different datasets to make predictions;
* the results will be the average for regression or the majority for the classification.

The bagging is designed to improve stability and accuracy, it has two phases:

* A phase of bootstrapping where it produces multiple training sets;
* the phase of aggregation where we get an ensemble method in which each model classify/regret and then compute the best value.

The bagging works starting from the an outcome ym(x) of a model m, there we compute the average error of the model in this way:

So, we compute the total average error:

And the committee error:

Assuming the errors have zero mean and they are uncorrelated, Ex belongs to zero, so:

The boosting is a sequential ensemble method where it adjuct the weight of the observations iteratively, the idea is the following:

* starting from a model, we trained it and produces some results;
* the second model takes the results of the first one as input for the training and the production of results;
* Once that each model produce results, we will combine it!

The decision tree is a model where each node is a condition, so the data points takes a branch or the another one in order to reach the results!

To create the tree structure, there are some methods:

* compute the optimal structure that is infeasible due to the number of combinations, then add the nodes using a greedy algorithm.
* We stop adding nodes when the residual error is below than a threshold;
* Optionally we can prune some branches.

The pruning process start from a tree T0 and a subtree T, if we want divide a node in |T| leaves, we do it with a criterion:

where lambda determine the trade-off between the Q(T) and |T|.

There are some ways to compute Q(T):

* the cross entropy;
* the gini-index

The random forest is an ensemble method taking some decision trees and making some predictions using differents training sets.

We have to be careful because it belongs to overfitting due to the high variance.

The random forest works as follow:

* pick k random points from the training set;
* build a decision tree;
* repeat the previous steps for the other trees;
* make a prediction testing the new point on each tree and combining the results.

# Unsupervised learning

The unsupervised learning is a type of learning where we haven’t classes or values to predict from the beginning, our purpose here is to make a prediction in base of the distances of the data.

The clustering is an algorithm that allows us to find cluster in the dataset, the PCA allow us to compute principal components we can use for the prediction.

## K-mean

The k-mean is a clustering algorithm computing k clusters in base of the dataset. the union of each cluster give the original dataset and their interection is null.

THe purpose is to minimize the within-cluster variation W(Ck), it is the measure of how the points of a cluster differ from each other.

Because of we have K clusters, we have to minimize the sum of the measures:



The algorithm is the following:

* do an initial cluster assignment on the points randomly;
* for each cluster, compute the cetroids;
* for each centroid we compute the distance for each point and we reassign it to the closest one;
* we reiterate from the second step until there are changes.

## Hierarchical clustering

The hierarchical clustering divides the dataset in due cluster and each cluster in two cluster too until we have the number of clusters we need using a cutoff.

To do this, this method uses a structure called dendogram, a tre structure representing the observations, to get this, firstly we have to choose a measure of distance, then an approach:

* the bottom-up approach start from N clusters, each one with one observation, then it merge the similar clusters two by two until there is a single cluster;
* the linkage define the measure of distance between groups of observations.

There are four ways of linkage:

* complete: there is the massimal distance between clusters, it compute all distances and takes the largest;
* single: there is the minimal distance between cluster, it works as the complete one but it takes the smallest cluster;
* average: it computes all the distances and records the average of these;
* centroid: it computes the distance between centroids.

### Algorithm

* Firstly, we have N clusters with N\*(N-1)/2 distances;
* For each i from N to 2, we examine all the distances to identity the minimum one, so we merge these clusters and we recompute the distances.

### Expectation-maximization

The expectation-maximization is an algorithm of clustering for the gaussian mixture, the purpose is to get k clusters using the linear superposition of k gaussian distribuitions in terms of latent variables.

In this way we add covariance informations providing the center ones given by the k-mean.

# Neural networks

A neural network is a structure that computes automatically the hidden features and use them to compute a solution! The unit is the neuron, it receives all inputs from the previous level and computes an out put in this way:

* make a linear combination of all the points, getting a value z;
* compute the output a using a function on z.

the function used by a neuron is the activation function, a function tells if the neuron fire of not.

The logistic and the softmax function are activation function for classification, for hidden nodes we have the ReLU.

logistic and softmax are differentiable in every point and with a range of 0-1, however they risks the saturation and requires the exponential. Moreover there is no sparsity, so they fire a zero very unlikely.

THe reLU is very simple to compute and act as a linear function even through it is not that. Also there is no saturation, so the gradient flows well and we control the sparsicy

In order to accelerate the neural network, there is a way called backpropagation, it consists in:

* a step of forward propagation, computing all the a, z and weights of each layer;
* starting from the end, we compute the gradients of a, z and the weights and, from these, the same gradients of the previous level;
* we apply the gradient descent on the weights using the gradients, the momentum and the learning rate.