Mastering machine learning CH 2

Topics to
Search:
the information Theory
Bernoulli Dist.
Kullback-Leibler divergence

Cost Functions and Optimization

Definitions :

Loss functions: are proxies that allow us to measure the error made by a machine learning model.

We consider the supervised scenario with finite datasets X and y

We can define the generic loss functions for a single data points as:

$$J(\bar{x}_i, \bar{y}_i; \bar{\theta}) = J(f(\bar{x}_i; \bar{\theta}), \bar{y}_i) = J(\tilde{y}_i, \bar{y}_i)$$

J is a function of the whole parameter set and must be proportional to the error between the true label and the predicted label.

Note:
A very important property of a loss function is convexity. In many real cases, this is an almost impossible condition; however, it's always useful to look for convex loss functions, because they can be easily optimized through the gradient descent method.

In many cases,
If the bias is null and the variance is
small enough, the resulting model will show a good generalization ability, with high training and validation accuracy; however, considering the data

generating process, it's useful to introduce another measure called expected risk:

$$E_{Risk}[f] = \int J(f(\bar{x};\bar{\theta}),\bar{y})p_{data}(\bar{x},\bar{y})d\bar{x}d\bar{y}$$

This value can be interpreted as an average of the loss function over all possible samples drawn from p(data). However, as p(data) is generally continuous, it's necessary to consider an expected value and integrate over all possible couples, which is often an intractable problem. The minimization of the expected risk implies the maximization of global accuracy, which, in turn, corresponds to the optimal outcome.

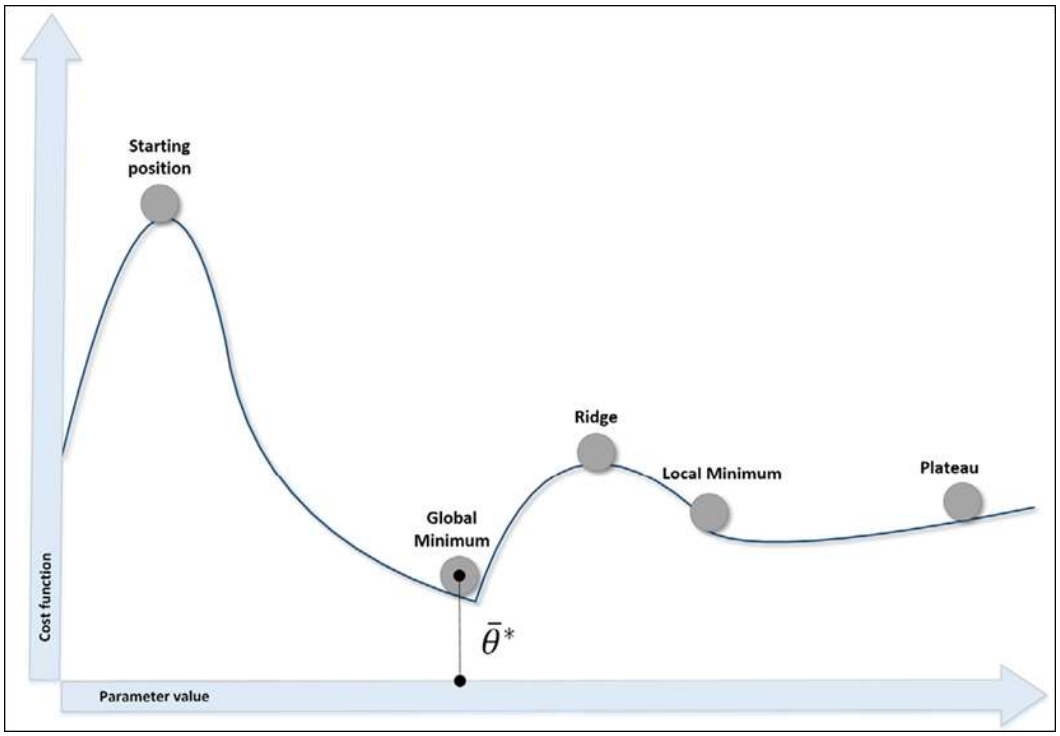
In real world scenarios we works with finite numbers of training samples
So it is preferrable to use a preferable cost functions called a loss function (no not confuse it with the log-likelihood)

$$L(X,Y;\bar{\theta}) = \sum_{i=0}^{N} J(\bar{x}_i,\bar{y}_i;\bar{\theta})$$

the actual function that we're going to minimize. Divided by the number of samples

also called as:
(emperical risk) because it's an approximation, based on a finite sample X, of the expected risk

When the cost function has more than two parameters it is perhaps Impossible to understand its internal structure



In the different situations we observe that :

The starting point the cost function is usually very high due to the error

Local minima:
Where the gradient is nulll and the second derivative is positive
(unfortunately, if the concavity isn't too deep, an inertial movement or
noise can easily move the point away.)

Ridges (local maxima):
where the gradient is null and the second derivative is negative. They are unstable points because even minimal
perturbation allows escape toward lower-cost areas.

Plateaus, or regions:
where the surface is almost flat and the gradient is
close to zero. The only way to escape a plateau is to keep some residual
kinetic energy

Global minimum:
the point we want to reach to optimize the cost
function.

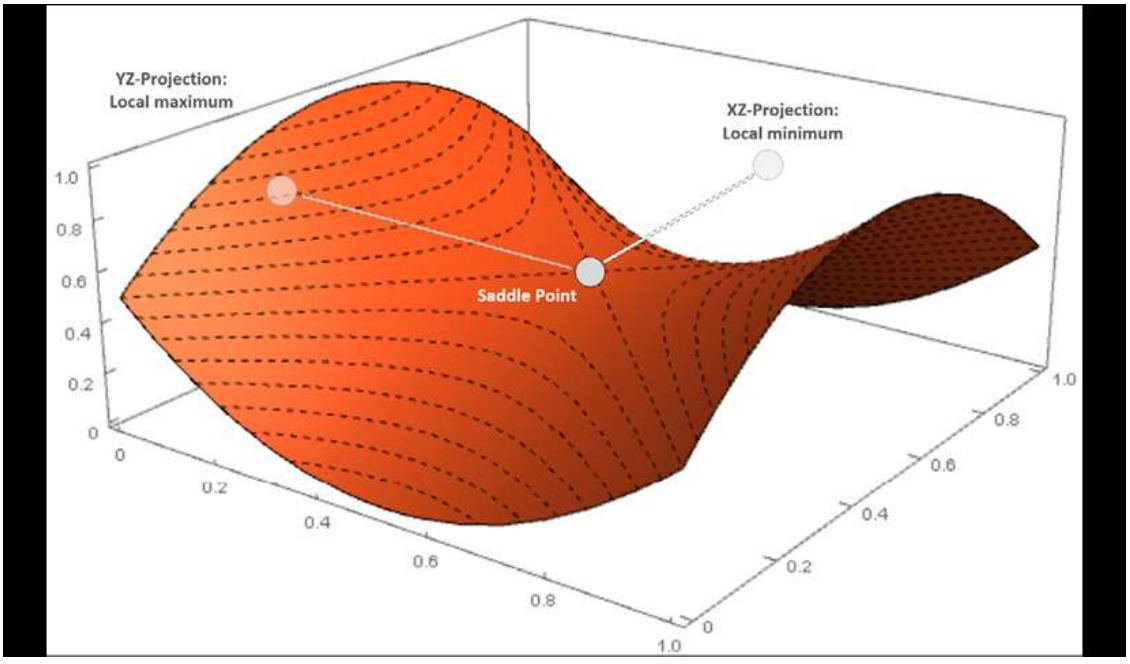
local minima are likely to be found in models with small number parameters and become more unlikely to be found when increasing the parameters

a local minimum for a
convex function (and here, we're assuming L to be convex) only if:

$$\begin{cases} \nabla_{\bar{\theta}} L(\bar{x}^*) = 0 \\ H_{\bar{\theta}} L(\bar{x}^*) \text{ is positive definite} \end{cases}$$

the first n rows and n
columns must be positive—therefore all its eigenvalues must be
positive. This probability decreases with the number of parameters (is
an n × n square matrix and has n eigenvalues), and becomes close to zero in
deep learning models where the number of weights can be in the order of
millions, or even more.

saddle points: where the eigenvalues have different signs and
the orthogonal directional derivatives are null, even if the points are neither
local maxima nor local minima.



The surface is very similar in shape to a horse saddle

saddle points are quite
dangerous, because many simpler optimization algorithms can slow down
and even stop, losing the ability to find the right direction.

Cost functions

Mean squared error: is one of the most common regression cost functions.
Its generic expression is:

$$L(X, Y; \bar{\theta}) = \frac{1}{N+1} \sum_{i=0}^{N} [f(\bar{x}_i; \bar{\theta}) - y_i]^2$$

This function is differentiable at every point of its domain and it's convex,

it can be optimized using the stochastic gradient descent (SGD)
algorithm.
This cost function is fundamental in regression analysis using the
Ordinary or Generalized Least Square algorithms

when employing it in regression tasks with
outliers. Its value is always quadratic, and therefore, when the distance
between the prediction and an actual outlier value is large.
the relative error is large.

mean squared error isn't robust to outliers, because it's always
quadratic, independent of the distance between actual value and prediction.

Huber cost function

is based on threshold T(h), so that for distances less than T(h) its behavior
is quadratic, while for a distance greater than T(h) it becomes linear, reducing
the entity of the error and thereby reducing the relative importance of the
outliers.

The analytical expression is:

$$L(X, Y; \bar{\theta}, t_H) = \begin{cases} \frac{1}{2} \sum_{i=0}^{N-1} [f(\bar{x}_i; \bar{\theta}) - y_i]^2 & if \ |f(\bar{x}_i; \bar{\theta}) - y_i| \leq t_H \\ t_H \sum_{i=0}^{N-1} |f(\bar{x}_i; \bar{\theta}) - y_i| - \frac{t_H}{2} & if \ |f(\bar{x}_i; \bar{\theta}) - y_i| > t_H \end{cases}$$

Hinge cost function

This cost function is adopted by Support Vector Machine (SVM) algorithms, where the goal is to maximize the distance between the separation boundaries. where the support vectors lie.

Its analytic expression is:

$$L(X, Y; \bar{\theta}) = \sum_{i=0}^{N-1} \max(0, 1 - f(\bar{x}_i; \bar{\theta}) y_i)$$

this cost function is not optimized using classic SGD methods
because it's not differentiable when:

$$f(\bar{x}_i; \bar{\theta}) y_i = 1 \Rightarrow \max(0,0)$$

For this reason, SVM algorithms are optimized using quadratic programming techniques

Categorical cross-entropy

Categorical cross-entropy is the most diffused classification cost function adopted by logistic regression and the majority of neural architectures.

The generic analytical expression is:

The Categorical cross-entropy is convex and can be easily optimized using SGD techmiques

if we toss a fair coin, then according to a Bernoulli distribution. Therefore, the entropy of such a discrete distribution is:

$$H(p) = -\sum_i p_i \log_2 p_i = -\frac{1}{2}(-1) - \frac{1}{2}(-1) = 1\ bit$$

The uncertainty is equal to 1 bit, which means that before any experiment there are 2 possible outcomes, which is obvious. What happens if we know that the coin is loaded, and P(Head) = 0.1 and P(Tail) = 0.9? The entropy is now:

$$H(p) = -0.1 \log_2 0.1 - 0.9 \log_2 0.9 \approx 0.47\ bit$$

the entropy is proportional to the variance or to the spread of the distribution.

the larger the variance, the larger the region in which the potential outcomes have a similar probability to be selected.

cross-entropy is defined between two distributions p and q:

P is the data generating process we are working with.