DATA-DRIVEN CLASSIFICATION & REGRESSION

[INTRODUCTION]

Prof. FRANCESCO A.N. PALMIERI
UNIVERSITA' DELLA CAMPANIA
LUIGI VANVITELLI

[CORSO DI SIGNAL PROCESSING & DATA FUSION]
N. Sett. 2023

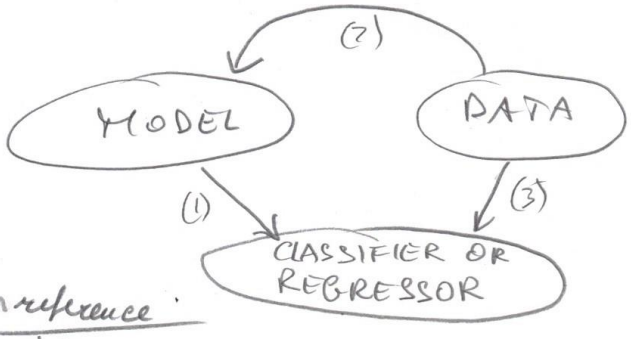


Fig 1

With reference

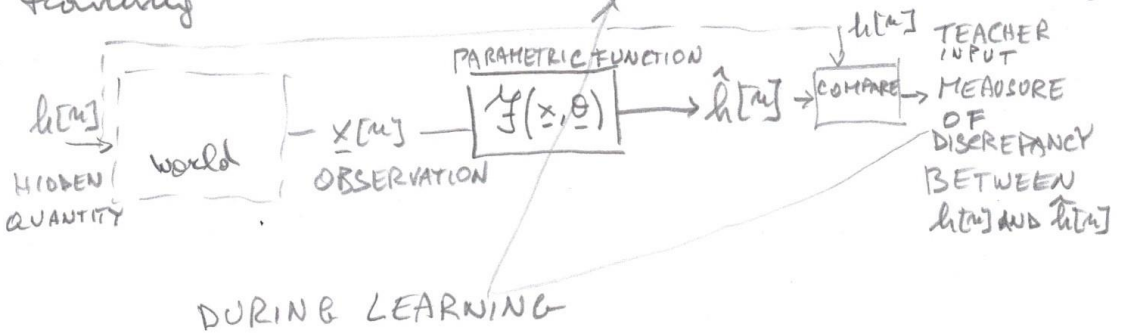
to the above diagram, we have examined in our previous lectures how to design a regressor or a classifier starting from a model (1). More specifically, when we have available a generative model in terms of likelihoods and priors, a regressor, or a classifier, can be computed mathematically (1). We have also pointed out how the model parameters may be estimated from data (2).

In this chapter we want to study path (3) where the design is completely derived, or better "learned", from data.

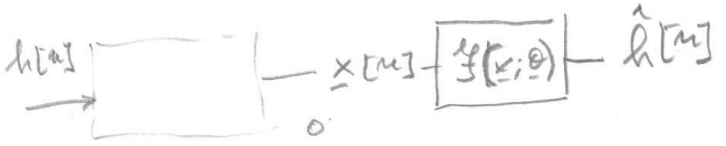
Data-driven solutions are becoming increasingly popular lately for their great success in providing solutions to long-standing problems such as: image recognition, speech synthesis, text processing etc.

The following figure shows our reference diagram where our observations come from the real world where we have no knowledge of the generative mechanism. (*)

There is a hidden quantity $h[n]$ available only during training



DURING LEARNING



AFTER LEARNING (INFERENCE)

In data-driven design we have available a number of "teaching examples" that during the learning phase allow us to determine the best parameters $\underline{\theta}$ for a function $f()$ that acts as a regressor or a classifier.

We refer to regression when $h[n]$ and $\hat{h}[n]$ are continuous variables

We refer to classification when $h[n]$ or $\hat{h}[n]$ belong to finite discrete sets.

The general paradigm presented above is what is referred to as SUPERVISED LEARNING or LEARNING WITH A TEACHER.

The input-output pairs

(input, desired output) — or true value

$(x[n], h[n])$

are usually divided into 3 sets:

TRAINING SET $\mathcal{L} = \{(x[n], h[n]),\ n = 1, \dots, N_L\}$

VALIDATION SET $\mathcal{V} = \{(x[n], h[n]),\ n = 1, \dots, N_V\}$

TEST SET $\mathcal{T} = \{(x[l], h[l]),\ l = 1, \dots, N_T\}$

(*) It is argued sometimes that using a given function class for $f(\underline{x}, \underline{\theta})$ in a way assumes that we know something about how $\underline{x}$ is generated, being $f(\underline{x}, \underline{\theta})$ a sort of "inverse model" of the observations.

This is an interesting (largely open) issue that we leave out for now, concentrating on the pragmatic view that postulates a function $f(\underline{x}, \underline{\theta})$. The choice is considered appropriate if after learning performances are satisfactory.

The designer has usually available a set of pairs that he/she can split as
[Training Set, Validation Set] (for example [80%, 20%]).
Learning is performed on the training set and the validation set is used to check if a good solution has been obtained. A solution that works well on the training set may not be satisfactory if it does not generalize to data unseen in the training process. The validation set is used for this purpose and it is part of the design process. When the parametric function $f(\underline{x}, \underline{\theta})$ has too many free parameters ($|\underline{\theta}|$) results on the training set may be very good, but useless, because the solution may be overfitting the data. The following picture shows a two dimensional space (both x and h are one-dimensional real variables)

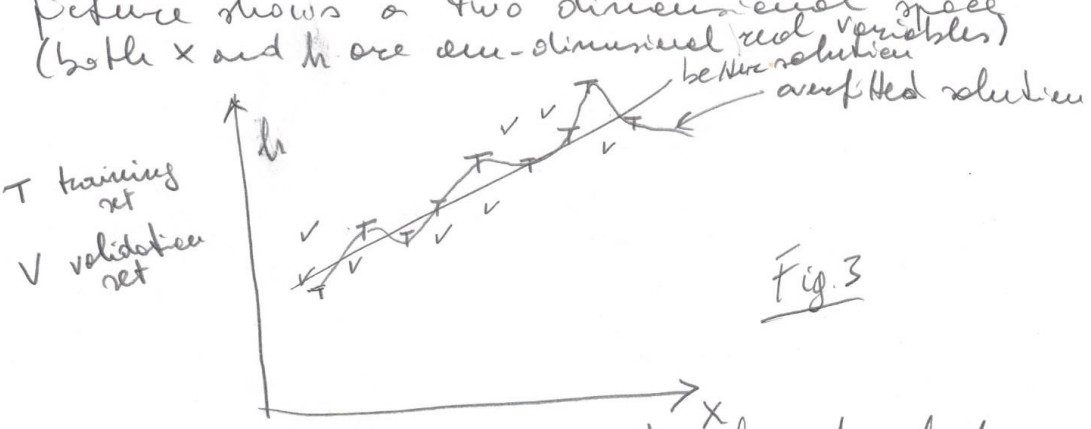


Fig. 3

An overfitted solution may be almost perfect on the training set, but perform poorly outside of it. A better solution can be found by checking for generalization on the validation set.
(smoothed, or regularized)

A typical diagram that reports performance for increasing complexity, or # of parameters, for training set and validation set is depicted in the following figure

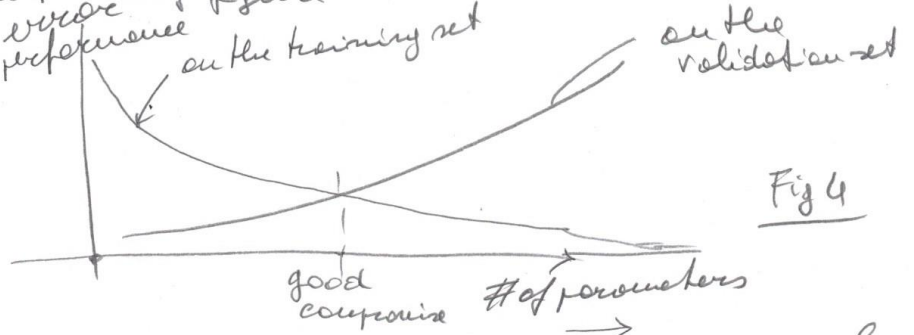


Fig 4

Increasing the # of parameters, the error may become negligible on the training set. However when testing the solution on the validation set, we can get increasingly worse performance because the function is "broken" or "overfitted" (see Eqn 3) the intersection of the two curves may represent a good compromise between performance and generalization.

However, the true final performance of a design must be evaluated on the TEST SET! The Test set should not influence the design. Note that during design there is an exchange of information between the validation set and the solution. After testing on the test set the design should not change!! In machine learning competitions, usually the test set is kept secret until final evaluation. The interested reader can check on Kaggle.com for datasets and competitions (they often award cash prizes!)

By looking at Figure 2,
the estimator $f(\underline{x}, \underline{\theta})$ is a parametric function that depends on a set of parameters $\underline{\theta}$. The size of $\underline{\theta}$ may be extremely large, even in the order of millions, or more, in large neural networks.
Performances for a given $\underline{\theta}$ can be written as the average over the TRAINING SET

$$\mathcal{E}(\underline{\theta}) = \frac{1}{N_z} \sum_{n=1}^{N_z} L\left(f(\underline{x}[n], \underline{\theta}), h[n]\right) \quad (*)$$

Note that on the cost function on the training set we omit the θ dependence. This is done for simplicity of the notation since it will be the most used one in the following.

or on the VALIDATION SET

$$\mathcal{E}^V(\underline{\theta}) = \frac{1}{N_V} \sum_{n=1}^{N_V} L\left(f(\underline{x}[n], \underline{\theta}), h[n]\right)$$

or on the TEST SET

$$\mathcal{E}^T(\underline{\theta}) = \frac{1}{N_T} \sum_{\ell=1}^{N_T} L\left(f(x[\ell], \theta), h[\ell]\right)$$

where $L(\hat{h}, h)$ is a LOSS FUNCTION that measures the discrepancy between the output $\hat{h} = f(\underline{x}, \underline{\theta})$ and h.
(We will be more specific about L in the following)
Learning happens on the training set and seeks the optimal solution

$$\underline{\theta}^\circ = \arg\min_{\underline{\theta}} \frac{1}{N_z} \sum_{n=1}^{N_z} L\left(f(\underline{x}[n], \underline{\theta}), h[n]\right)$$

The solution is then tested on the validation set $\mathcal{E}^V(\theta^\circ)$ and on the test set $\mathcal{E}^T(\theta^\circ)$, as explained above.

The performance function (cost) $E(\underline{\theta})$, (in estimation theory sometimes called "risk", here it would be an "empirical risk", is not necessarily convex in $\underline{\theta}$. This is actually quite rare and actually difficult to assert. The solution to the minimization of $E(\theta)$ is usually a "local minimum". The challenge for any learning system is therefore to find a satisfactory local minimum! We will address this issue later in these notes.

It is evident from the above considerations that a good solution learned from data has to rely on a very large dataset, that both in the training set and in the validation set, contains enough example to span as much as possible the space $(\underline{x}, h)$.

REGRESSION

In regression problems the desired output h is a continuous vector variable $\underline{d}$.

Here $\underline{f}(\underline{x},\underline{\theta})$ is a vector function $\underline{y} = \underline{f}(\underline{x},\underline{\theta})$ where also $\underline{y}$ is a continuous vector variable that can be compared directly to $\underline{d}$ by taking the difference vector $\underline{\varepsilon} = \underline{d} - \underline{y}$

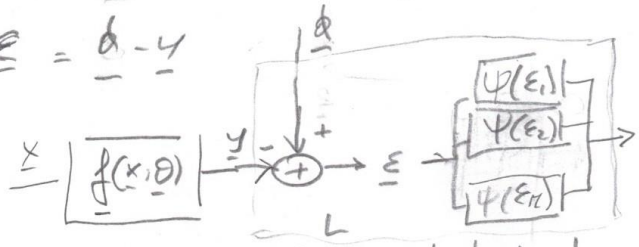


Fig 5

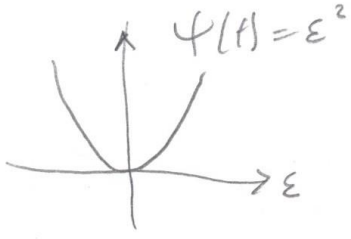
Error values must contribute to the total cost in an additive fashion, therefore a loss function $\psi(\varepsilon)$, where ε is a scalar, must be defined.

For a generic component

$$L(d,y) = \psi(d-y) = \psi(\varepsilon)$$

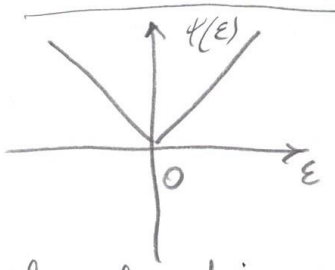
we have many possible choices for $\psi(\cdot)$.

SQUARED LOSS



By far the most popular one. This function is differentiable, and, as we saw in model-based regression, may lead to closed-form solutions

ABSOLUTE VALUE LOSS

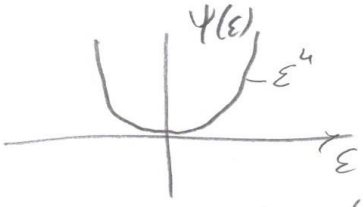


$$\psi(\varepsilon) = |\varepsilon|$$

This function is used to avoid the excessive weight given to large error values in the quadratic loss. Note that this function is discontinuous in its first derivative.

HIGH-ORDER LOSS

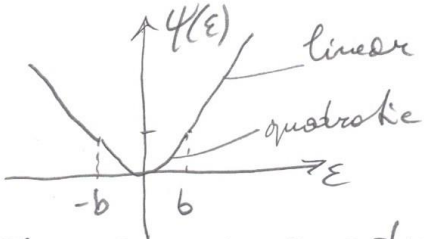
Larger orders in the exponent can be used



$$\psi(\varepsilon^n) \quad n \text{ even}$$

Here continuity and differentiability are ensured, but the effect of weighting more large values of ε is further accentuated.

HUBER LOSS



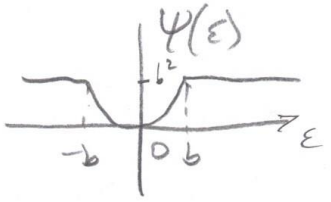
$$\psi(\varepsilon) = \begin{cases} -b\varepsilon - \frac{b^2}{2} & \varepsilon < -b \\ \frac{1}{2}\varepsilon^2 & -b < \varepsilon < b \\ b\varepsilon - \frac{b^2}{2} & \varepsilon > b \end{cases}$$

This loss is quadratic around the origin and linear for larger values.
In its definition attention is paid to ensure continuity and differentiability for $\varepsilon = \pm b$.

Huber loss is common in the statistics literature because it merges the advantages of the quadratic function for small ε and those of the absolute value loss for larger ε.

SATURATED QUADRATIC LOSS

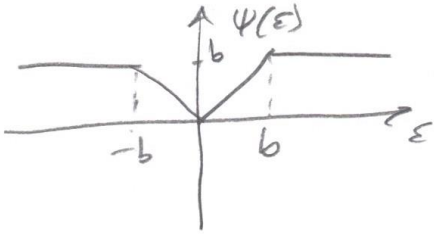
When larger values of ε are to be considered equivalent, or perhaps is necessary to contain large error deviations, the quadratic loss is modified as



$$\Psi(\varepsilon) = \begin{cases} b^2 & \varepsilon < -b \\ \varepsilon^2 & -b < \varepsilon < b \\ b^2 & \varepsilon > b \end{cases}$$

SATURATED ABSOLUTE VALUE LOSS

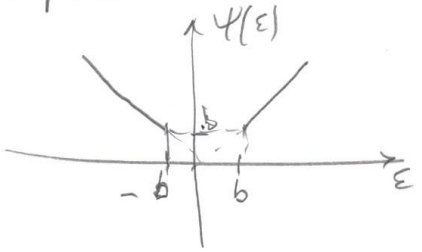
Similarly to the above, the absolute value loss can be modified to be



$$\Psi(\varepsilon) = \begin{cases} b & \varepsilon < -b \\ |\varepsilon| & -b < \varepsilon < b \\ b & \varepsilon > b \end{cases}$$

THRESHOLDED LOSS

When the optimization needs to ignore small values of ε, a loss with a threshold can be adopted

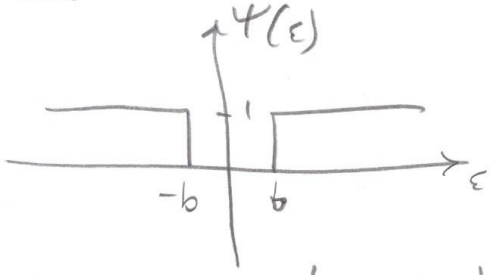


$$\psi(\varepsilon) = \begin{cases} -\varepsilon & \varepsilon < -b \\ 0 & -b < \varepsilon < b \\ \varepsilon & \varepsilon > b \end{cases}$$

The same can be done with any other loss function setting to zero ψ for small values of ε.

THRESHOLDED BINARY

When small values are ignored and larger values are given equal weight we can use the binary threshold function



$$\psi(\varepsilon) = \begin{cases} 1 & \varepsilon < -b \\ 0 & -b < \varepsilon < b \\ 1 & \varepsilon > b \end{cases}$$

The list presented above is by no means exaustive because other loss functions can be built for specific applications.
Their choice determines the landscape of the total cost and the most appropriate search algorithm (gradient-based, combinatorial, etc.)

When $\underline{y}$ and $\underline{d}$ are M-dimensional vectors, as in Figure 5, usually Ψ is applied to all the components of $\underline{\varepsilon} = \underline{d} - \underline{y}$ and the total loss is their sum.

If different components of $\underline{\varepsilon}$ are to be given non equal importance, we can use a convex combination and compute the total loss as

$$L(\underline{\Phi}, \underline{y}) = \sum_{i=1}^{M} \alpha_i \Psi(d_i - y_i) \quad \begin{cases} 0 < \alpha_i < 1 \\ \sum_{i=1}^{M} \alpha_i = 1 \end{cases}$$

Choosing $\alpha_1 \dots \alpha_M$ may be challenging unless the application suggests clearly different importance to the various components.
The most common choice remains $\alpha_i = \frac{1}{M}$ $i = 1, M$ and the cost function to minimize on the training set is

$$\mathcal{E}(\underline{\theta}) = \frac{1}{N_2} \sum_{n=1}^{N_2} \sum_{i=1}^{M} \Psi(d_i[n] - f_i(\underline{x}[n], \underline{\theta}))$$

i.e.

$$\underline{\theta}^{\circ} = \arg\min_{\underline{\theta}} \frac{1}{N_2} \sum_{n=1}^{N_2} \sum_{i=1}^{M} \Psi(d_i[n] - f_i[\underline{x}[n], \underline{\theta})]$$

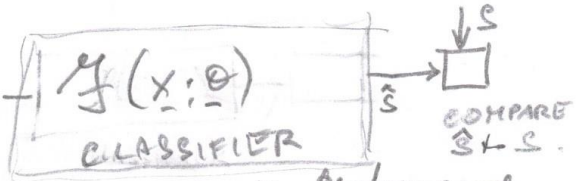
We will use this approach in the following

CLASSIFICATION

In classification problems the hidden quantity s is a symbol which belongs to a discrete set of CLASSES $s \in \mathcal{A} = \{a_1, \dots, a_M\}$

For example in image classification

In our framework the output of our parametric vector function $\underline{f}(\underline{x};\underline{\theta})$ is a label $\hat{s} \in \mathcal{A}$ that should match s as much as possible.



COUNTING ERRORS To evaluate performance we can count how many times $\hat{s} \neq s$, i.e. when we have a classification error (misclassification).

To be more precise, for each class we can count how many times it has been misclassified, and with which one.

We can organize our results in a "confusion matrix"

s \ $\hat{s}$	a_1	a_2	...	a_M
a_1	$n(a_1\|a_1)$	$n(a_2\|a_1)$		$n(a_M\|a_1)$
a_2	$n(a_1\|a_2)$	$n(a_2\|a_2)$		$n(a_M\|a_2)$
a_M	$n(a_1\|a_M)$	$n(a_2\|a_M)$		$n(a_M\|a_M)$

(estimated $\hat{s}$ — true s)

CONFUSION MATRIX

$$N_{\hat{s}|s} = \left[n(\hat{s}=a_j \mid s=a_i) \right]_{i,j=1,\dots,M}$$

# of times class a_i has been classified as a_j

If we normalize to 1 each row

$$p(\hat{s}=a_j \mid s=a_i) = \frac{n(\hat{s}=a_j \mid s=a_i)}{\sum_{\ell=1}^{M} n(\hat{s}=a_\ell \mid s=a_i)}$$

the matrix becomes row-stochastic and its elements can be read frequencies of occurrence, or probabilities (estimated)

s \ $\hat{s}$	a_1	a_2	...	a_M
a_1	$p(a_1\|a_1)$	$p(a_2\|a_1)$		$p(a_M\|a_1)$
a_2	$p(a_1\|a_2)$	$p(a_2\|a_2)$		$p(a_M\|a_2)$
a_M	$p(a_1\|a_M)$	$p(a_2\|a_M)$		$p(a_M\|a_M)$

This is like the channel matrix used in digital communication

Note that the classifier performance, i.e. the confusion matrix, can be computed on the training set, the validation set and the test set, and they can be very different from each other.
Each row of the confusion matrix $N_{\hat{S}|S}$ contains information on how many examples have been included in the set for each class.

$$N(a_i) = \sum_{\ell=1}^{M} n(\hat{S}=a_\ell | S=a_i) \qquad i=1,\dots,M$$

$N(a_i)$: total # of examples for class a_i

The sum of all the elements of $N_{\hat{S}|S}$ is clearly the size of the set (training n_a, evaluation n_v or test n_T): $n = \sum_{i=1}^{M}\sum_{j=1}^{M} n(\hat{S}=a_j | S=a_i)$

The sum of all the off-diagonal elements is the total number of errors

$$N_\varepsilon = \sum_{\substack{i=1 \\ i\neq j}}^{M}\sum_{j=1}^{M} n(\hat{S}=a_j | S=a_i)$$

$$\varepsilon_p = \frac{N_\varepsilon}{n} \times 100 \qquad \text{ERROR PERCENTAGE}$$

If we attribute to the frequency of occurrence of each class in the set, the meaning of a prior

$$\pi(a_i) = \frac{N(a_i)}{n}$$

We can calculate the error probability

$$P(\text{Error}) = 1 - P(c) = 1 - \sum_{i=1}^{M} P(c|a_i)\,\pi(a_i)$$

$$= 1 - \sum_{i=1}^{M} p(\hat{s}=a_i|s=a_i)\,\pi(a_i)$$

We should be aware that this approach may be very misleading: just because in our set a certain class is more present than another, does not necessarily mean that it is more probable.

Usually a more robust approach to this issue is to assume that all classes are equally likely, so that

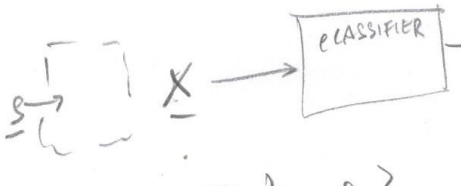
$$P(\text{Error}) = 1 - \frac{1}{M}\sum_{i=1}^{M} P(\hat{s}=a_i|s=a_i) = \frac{N_E}{N}$$

In the statistical literature the estimation of the priors from the data is a controversial issue. The use of conjugate priors, or entropic priors, among others, have been proposed. The interested reader can consult the specific literature.

CONSIDERING ERASURES

A classifier may also "provide" "no decision", i.e. raising hands and saying "I do not know", or "I am not sure". This may happen because the data is insufficient or the posteriors may be too close to each other. (more on posteriors later)

In this case the decision of the classifier may belong to a larger alphabet, augmented with an extra symbol



$\hat{s} \in \mathcal{A}_a = \{a_0\ a_1 \dots a_M\}$

a_0: cancellation or non-decision class

EXAMPLE In this example a classifier looks at an image and has to decide if it is a person, a dog, a horse or else. The alfabeth is

$$s \in \mathcal{A} = \{\text{person, dog, horse, else}\}$$

After training the classifier gives the following confusion matrices

true \ $\hat{s}$	person	dog	horse	else
person	15	0	1	0
dog	1	20	0	3
horse	2	1	10	0
else	1	1	0	13

on the training set ($n_z = 58$)

s \ $\hat{s}$	person	dog	horse	else
person	8	1	0	2
dog	2	3	1	1
horse	1	0	5	0
else	2	1	0	10

on the validation set ($n_z = 37$)

Note how different the results are on the two sets.

Reading the results on the validation set:
A person has been correctly classified 8 times and 1 time it has been confused with a dog, never with a horse and 2 times with else, etc...

We can normalize the confusion matrices to get the row-stochastic matrices $[P(\hat{s}=a_j \mid s=a_i)]$

s \ $\hat{s}$	person	dog	horse	else
person	15/16	0	1/16	0
dog	1/24	20/24	0	3/24
horse	2/13	1/13	10/13	0
else	1/15	1/15	0	13/15

$[P(\hat{s}=a_j \mid s=a_i)]$

s \ $\hat{s}$	person	dog	horse	else
person	8/11	1/11	0	2/11
dog	2/7	3/7	1/7	1/7
horse	1/6	0	5/8	0
else	2/13	1/13	0	10/13

We had in the training set the following examples: 16 for "person", 24 for "dog", 13 for "horse", and 15 for "else" $\Rightarrow n_z = 58$ examples.

We had in the validation set the following examples: 11 for "person", 7 for "dog", 8 for "horse", 13 for "else" $\Rightarrow n_v = 37$ examples.

To be more precise we report the results on a table in percentages

	$n_t = 58$ examples Training set	$n_v = 37$ examples Validation set
Total # of errors	$N_\varepsilon = 10$	$N_\varepsilon = 11$
error percentage	$\left(\frac{N_\varepsilon}{n_t} = 0.17\right)$ 17%	$\left(\frac{N_\varepsilon^v}{n_v} = 0.30\right)$ 30%
correct classification	83%	70%
A "person" has been correctly classified	$\left(\frac{15}{16} = 0.94\right)$ 94% of times	$\left(\frac{8}{11} = 0.73\right)$ 73% of times
A "dog" has been correctly classified	$\left(\frac{20}{24} = 0.83\right)$ 83% "	$\left(\frac{3}{7} = 0.43\right)$ 43% "
A "horse has been correctly classified	$\left(\frac{10}{13} = 0.77\right)$ 77% "	$\left(\frac{5}{6} = 0.83\right)$ 83% "
"Else" has been correctly classified	$\left(\frac{13}{15} = 0.87\right)$ 87% "	$\left(\frac{10}{13} = 0.77\right)$ 77% "

Note how different the performances are on the two sets. This is just a didactic example. Usually datasets are much larger than 58 and 37.

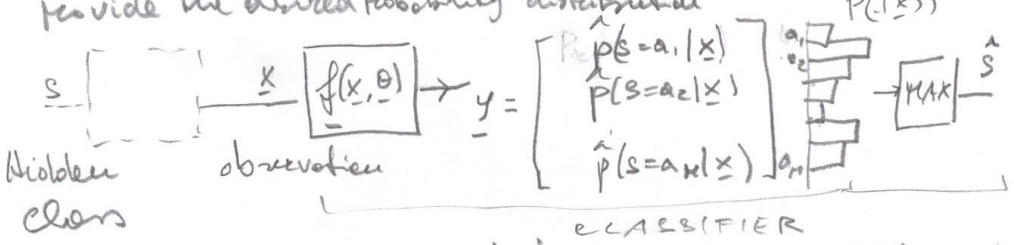
LOSS FUNCTIONS FOR CLASSIFICATION

Counting errors is certainly a measure of performance for our classifier, but it is not clear how this can be translated into a differentiable cost function. As we will see in the following, training a parametric function $\underline{y}(\underline{x},\underline{\theta})$ requires differentiability of $E(\underline{\theta})$.

We adopt a better approach assuming that the outputs $\underline{y}$ of our system are the posterior probabilities for the various classes, given the observation $\underline{x}$

$$y_i = \hat{P}(a_i|\underline{x}) = \text{Prob that given } \underline{x}, \text{ } a_i \text{ is the hidden class.}$$

The estimator is a vector function $\underline{f}(\underline{x},\underline{\theta})$ that has to provide the desired probability distribution

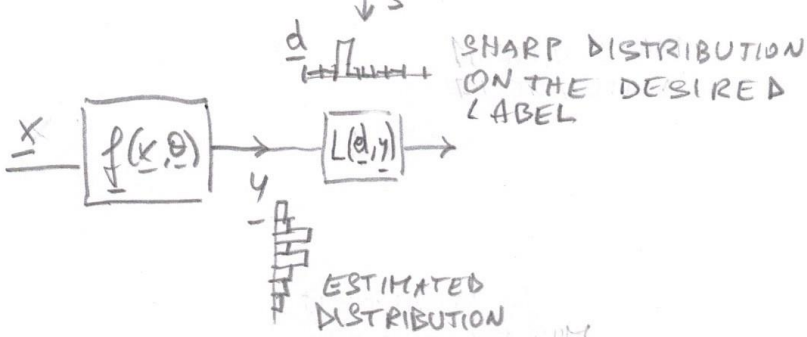


From the posterior distribution we can take a decision on

$$\hat{s} = \underset{i=1,\dots M}{\operatorname{argmax}} \hat{p}(s=a_i|\underline{x})$$

We know from the discussion on model-based solutions, that to maximize the posterior is equivalent to error minimization. Obviously here we are in a data-driven situation where $\hat{p}(s|\underline{x})$ are only possibly

estimates of the true $p(s|\underline{x})$, if they ever exist! In contrast, this framework becomes a good reference for defining a loss and then a differentiable cost function



We map the desired output s to a sharp distribution $\underline{d}$ (one-hot vector)

$$s=a_1 \Rightarrow \underline{d}=\begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix} \quad ; \quad s=a_2 \Rightarrow \underline{d}=\begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix} \quad \cdots \quad s=a_M \Rightarrow \begin{bmatrix}0\\0\\\vdots\\0\\1\end{bmatrix}$$

and we try to match $\underline{y}$ to $\underline{d}$ as much as possible. The vector $\underline{y}$ is constrained to have positive components $0 \le y_i \le 1$ that sum to one $\sum_{i=1}^{M} y_i = 1$. Usually $\underline{y}$ is the output of a softmax function that naturally satisfies the constraints.

This approach is particularly interesting because we can also conceive a situation in which we are not sure (in the learning set) about the label and use a distribution $\underline{d}$ that is not one-hot.

For example, if $n=5$, and the training set has provided a double label, say on the two classes a_1 and a_3, the desired distribution can be taken to be:

$$\underline{d} = \begin{bmatrix} 1/2 \\ 0 \\ 1/2 \\ 0 \\ 0 \end{bmatrix}$$

We need now to define a measure of discrepancy between $\underline{d}$ and $\underline{y}$, i.e. a loss function $L(\underline{d}, \underline{y})$.

Any loss function defined for regression can be applied to this scenario as

$$L(\underline{d}, \underline{y}) = \sum_{i=1}^{n} \psi(d_i - y_i).$$

Recall the squared function $\psi(d_i - y_i) = (d_i - y_i)^2$, the absolute value $\psi(d_i - y_i) = |d_i - y_i|$, etc.

However the peculiarity of $\underline{d}$ and $\underline{y}$ being two discrete distributions suggests that we can use different ideas from information theory to measure the difference between $\underline{d}$ and $\underline{y}$.

CROSS ENTROPY

There are various ways of measuring the difference between two distributions. The most common one is the KL-DIVERGENCE (KL stands for Kullback-Leibler)

$$KL(\underline{d}\|\underline{y}) = \sum_{i=1}^{M} d_i \log \frac{d_i}{y_i}$$ and $KL(\underline{d}\|\underline{y}) \geq 0$

Recall that if $\underline{d}=\underline{y}$, $KL(\underline{d}\|\underline{y})=0$. This definition is well-known in information theory and it is widely used, even if it is not technically a distance because $KL(\underline{d}\|\underline{y}) \neq KL(\underline{y}\|\underline{d})$. If we rewrite it as

$$KL(\underline{d}\|\underline{y}) = \underbrace{-\sum_{i=1}^{M} d_i \log \frac{1}{d_i}}_{-H(\underline{d})} + \underbrace{\sum_{i=1}^{M} d_i \log \frac{1}{y_i}}_{H(\underline{d},\underline{y})}$$

the first term is minus the entropy of $\underline{d}$ and the second term is the so-called CROSS-ENTROPY

$$\boxed{H(\underline{d},\underline{y}) = \sum_{i=1}^{M} d_i \log \frac{1}{y_i}}$$

CROSS-ENTROPY

Note that $H(\underline{d},\underline{y}) = KL(\underline{d}\|\underline{y}) + H(\underline{d})$.

Also since both terms are positive also the cross entropy is positive.

$$H(\underline{d},\underline{y}) \geq 0.$$

Furthermore since $H(\underline{d})$ is fixed, the cross entropy can be used as a loss

$$L(\underline{d},\underline{y}) = H(\underline{d},\underline{y})$$

to minimize and it is equivalent to cf. KL minimization of $KL(\underline{d}\|\underline{y})$.

In learning a classifier on labeled (typically) data, the desired distribution $\underline{d}$ is one-hot encoded and a perfect match between $\underline{d}$ and $\underline{y}$ would give $H(\underline{d},\underline{y})=0$. For example if $\underline{d}=\begin{pmatrix}0\\1\\0\\0\end{pmatrix}$

$$H(\underline{d},\underline{y}) = 0\lg\frac{1}{y_1} + 1\lg\frac{1}{y_2} + 0\lg\frac{1}{y_3} + 0\lg\frac{1}{y_4}$$

$$= -\lg y_2 \qquad \left(-\text{log-likelihood of } y_2\right)$$

$= 0$ if $y_2=1$; if $\underline{d}=\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $H(\underline{d},\underline{y}) = -\lg y_3$, etc...

Minimize $H(\underline{d},\underline{y})$ is also equivalent to maximum log-likelihood learning.

learning a classifier $\underline{y}=f(\underline{x},\underline{\theta})$ on the training set $\mathcal{X}$, can we formulated as the problem

$$\underline{\theta}^* = \arg\min_{\underline{\theta}} \frac{1}{n_2}\sum_{n=1}^{n_2} H(\underbrace{f(\underline{x}^{[n]},\underline{\theta})}_{\underline{y}^{[n]}}, \underline{d}^{[n]})$$

$$= \arg\min_{\underline{\theta}} \frac{1}{n_2}\sum_{n=1}^{n_2}\left(\sum_{i=1}^{M} d_i^{[n]} \lg\frac{1}{y_i^{[n]}}\right)$$

THE BINARY CASE

Let us look at the important binary case where $\Omega = \{a_1, a_2\}$ with $\underline{d} = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$

$\underline{d} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ for a_1 and $\underline{d} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ for a_2

The classifier does not need to produce two outputs y_1 and y_2 because they were complementary i.e. $1: y_1 = 1 - y_2$. Therefore we can have a classifier with only one output y that represents, for example, $y = P(a_2|\underline{x})$ because $1 - y = P(a_1|\underline{x})$.

Also the loss function simplifies because with two outputs we would have

$$L(\underline{d}, \underline{y}) = d_1 \log \frac{1}{y_1} + d_2 \log \frac{1}{y_2}$$

$$= d_1 \log \frac{1}{1-y_2} + d_2 \log \frac{1}{y_2}$$

But d_1 and d_2 are also complementary i.e. $d_2 = 1 - d_1$, therefore

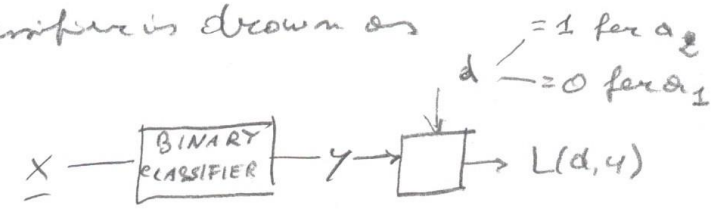
$$L(\underline{d}, \underline{y}) = d_1 \log \frac{1}{1-y_2} + (1-d_1) \log \frac{1}{y_2} \quad \text{or}$$

$$L(\underline{d}, \underline{y}) = (1-d_2) \frac{1}{1-y_2} + d_2 \log \frac{1}{y_2}$$

Taking a scalar desired value d as desired output

$$\begin{cases} d = 0 & \text{for } a_1 \Rightarrow y = 0 \\ d = 1 & \text{for } a_2 \Rightarrow y = 1 \end{cases}$$

The classifier is drawn as



with

$$L(d,y) = d \log\frac{1}{y} + (1-d)\log\frac{1}{1-y} \quad (*)$$

We will see in the following that the last stage of the classifier may be a sigmoidal function $y=\varphi(z)$, usually (not necessarily) chosen to be a logistic function $\left(y=\varphi(z)=\frac{1}{1+e^{-z}}\right)$. The reason for this choice is its relation to the solution for model-based gaussian classifier, but also because the logistic function can be seen as equivalent to a two-class softmax architecture. A detailed discussion about this will be reported in the following.

(*) Verify that L is null for perfect match:

$d=1, y=1 \quad L = 1\log\frac{1}{1} + 0\log\frac{1}{0} = 0$

$d=0, y=0 \quad L = 0\log\frac{1}{0} + 1\log\frac{1}{1} = 0$

For wrong match

$d=1 \quad y=0 \quad L = 1\log\frac{1}{0} + 0\log\frac{1}{1} = \infty$

$d=0 \quad y=1 \quad L = 0\log\frac{1}{0} + 1\log\frac{1}{0} = \infty$

MORE ON BINARY CLASSIFICATION

In binary classification, we have already looked at the various errors with reference to the confusion matrix

S \ $\hat{S}$	P	N
P	TRUE POSITIVE	FALSE NEGATIVE
N	FALSE POSITIVE	TRUE NEGATIVE

and attributed a probability to each of the 4 conditional events.

When running an experiment on data, the results of a classifier is clearly in terms of number of results for each case

S \ $\hat{S}$	P	N
P	$n(\hat{S}=P \mid S=P)$	$n(\hat{S}=N \mid S=P)$
N	$n(\hat{S}=P \mid S=N)$	$n(\hat{S}=N \mid S=N)$

It is common in machine learning to define the following parameters

$$\text{PRECISION} = \frac{\text{TRUE POSITIVE}}{\text{TRUE POSITIVE} + \text{FALSE POSITIVE}} = \frac{n(\hat{S}=P \mid S=P)}{n(\hat{S}=P \mid S=P) + n(\hat{S}=P \mid S=N)}$$

This is often referred to as positive predictive value (PPV)

$$\text{RECALL} = \frac{\text{TRUE POSITIVE}}{\text{TRUE POSITIVE} + \text{FALSE NEGATIVE}}$$

$$= \frac{u(\hat{S}=P \mid S=P)}{u(\hat{S}=P \mid S=P) + u(\hat{S}=N \mid S=P)}$$

This can be associated to the probability of true positive and it is sometimes referred to as true positive rate or sensitivity

Another definition is

$$\text{TRUE NEGATIVE RATE} = \frac{\text{TRUE NEGATIVE}}{\text{TRUE NEGATIVE} + \text{FALSE POSITIVE}}$$

$$= \frac{u(\hat{S}=N \mid S=N)}{u(\hat{S}=N \mid S=N) + u(\hat{S}=P \mid S=N)}$$

This can be associated to the prob. of true negative and is sometimes referred to as specificity

A global measure of performance often used in machine learning is

$$\text{ACCURACY (UNBANCED)} = \frac{\text{TRUE POSITIVE} + \text{TRUE NEGATIVE}}{\text{TRUE POSITIVE} + \text{TRUE NEGATIVE} + \text{FALSE POSITIVE} + \text{FALSE NEGATIVE}}$$

$$= \frac{u(\hat{S}=P \mid S=P) + u(\hat{S}=N \mid S=N)}{u(\hat{S}=P \mid S=P) + u(\hat{S}=N \mid S=N) + u(\hat{S}=P \mid S=N) + u(\hat{S}=N \mid S=P)}$$

This measure on real data may not be appropriate because the data set is positive and negative examples may be unbalanced.

A better accuracy measure is

$$\text{BALANCED ACCURACY} = \frac{\text{TRUE POSITIVE RATE} + \text{TRUE NEGATIVE RATE}}{2}$$

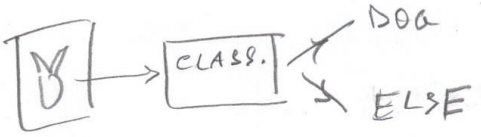
This can be associated to the average probability of correct response with uniform priors:

$$\frac{1}{2}\left(P_r\{\hat{s}=P|s=P\} + P\{\hat{s}=N|s=N\}\right)$$

The interested student may consult the vast statistical literature rich of other definitions that are usually application specific.

EXAMPLE

A pattern recognition system has to distinguish dogs from other animals (else)



The results of the experiment on the test set (27 examples) were reported in the following confusion matrix.

S		DOG	ELSE
	DOG	15	3
	ELSE	2	7

Note that the dataset is quite unbalanced!

The defined parameters are:

$$\text{PRECISION} = \frac{15}{15+2} = 0.8823 \qquad 88\%$$

$$\text{RECALL} \atop (\text{TRUE POSITIVE RATE}) = \frac{15}{15+3} = 0.8333 \qquad 83\%$$

$$\text{TRUE NEGATIVE RATE} = \frac{7}{7+2} = 0.7778 \qquad 78\%$$

$$\text{UNBALANCED ACCURACY} = \frac{15+7}{15+7+3+2} = \frac{22}{27} = 0.8148 \qquad 81\%$$

$$\text{BALANCED ACCURACY} = \frac{1}{2}\left(\frac{15}{18} + \frac{7}{9}\right) = \frac{15+14}{18.2} = \frac{29}{36} = 0.8056 \qquad 80\%$$

—— o ——

The results of this example show that, even if the performance percentages are all in the same range, a comprehensive measure may be misleading. A direct look at the details of the confusion matrix should always be considered.