Extreme Learning Machine in J

Pierre-Edouard Portier

2019

1 Regression

$\boldsymbol{x^{(1)}} \dots \boldsymbol{x^{(P)}}$ are vectors of $\mathbb{R}^{n-1}$ with associated values $y^{(1)} \dots y^{(P)}$ of $\mathbb{R}$. We search a function $f(\boldsymbol{x}) : \mathbb{R}^{n-1} \to \mathbb{R}$ to model the observed relationship between $\boldsymbol{x}$ and y. f can have a fixed parameterized form. For example:

$$f(\boldsymbol{x}) = a_0 + a_1 x_1 + a_2 x_2 + \dots + a_{n-1} x_{n-1}$$

If $P = n$, parameters $a_0 \dots a_{n-1}$ are found by solving a linear system.

$$\begin{cases} y^{(1)} & = a_0 + a_1 x_1^{(1)} + a_2 x_2^{(1)} + \dots + a_{n-1} x_{n-1}^{(1)} \\ \dots & = \dots \\ y^{(P)} & = a_0 + a_1 x_1^{(P)} + a_2 x_2^{(P)} + \dots + a_{n-1} x_{n-1}^{(P)} \end{cases}$$

This system can be written In matrix form.

$$\begin{pmatrix} 1 & x_1^{(1)} & \dots & x_{n-1}^{(1)} \\ 1 & x_1^{(2)} & \dots & x_{n-1}^{(2)} \\ \dots & \dots & \dots & \dots \\ 1 & x_1^{(P)} & \dots & x_{n-1}^{(P)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(P)} \end{pmatrix}$$

Each line of the first term matrix is a vector $\boldsymbol{x^{(i)T}}$ with the addition of a constant coordinate that accounts for parameter a_0. Thus, naming this matrix $\boldsymbol{X}^T$, the linear system can also be written:

$$\boldsymbol{X}^T \boldsymbol{a} = \boldsymbol{y}$$

Consider the special case when x is a number and f is a polynomial of degree $n-1$:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{n-1} x^{n-1}$$

With $P = n$ examples $\left(x^{(k)}, y^{(k)}\right)$, the parameters are found by solving the following linear system:

$$\begin{pmatrix} 1 & x^{(1)} & (x^{(1)})^2 & \dots & (x^{(1)})^{n-1} \\ 1 & x^{(2)} & (x^{(2)})^2 & \dots & (x^{(2)})^{n-1} \\ \dots & \dots & \dots & \dots & \\ 1 & x^{(P)} & (x^{(P)})^2 & \dots & (x^{(P)})^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(P)} \end{pmatrix} \tag{1}$$

Incidentally, the first term is called the Vandermonde Matrix.

1.1 Experiment with a 1-dimensional synthetic dataset

We define a non linear function `f` from which we generate a dataset

2a ⟨*dataset* 2a⟩≡ (10)
```
f=: 3 : '(^y) * cos 2*pi * sin pi * y'
⟨noise 2b⟩
⟨gendat 2d⟩
```

In traditional mathematical form, this function is:

$$f(x) = e^x \times cos\left(2\pi sin\left(\pi x\right)\right)$$

Function `noise` adds some random noise to the values of a vector. For example `0.5 noise v`, will add random values uniformly drawn from interval $[-0.5, 0.5]$ to the terms of vector `v`.

2b ⟨*noise* 2b⟩≡ (2a)
```
noise=: 4 : 'y + -&x *&(+:x) ? (#y) # 0'
```

`0.5 gendat 10` generates from `f` a dataset `(X,Y)` of 10 points with random noise in $[-0.5, 0.5]$ added to `Y`. It also stores in `minmaxX` the minimum and maximum values of `X`. It computes the pair `minmaxf`, where the first term is ten percent smaller than the minimum of `f` on interval $[0, 1]$, and the second term is ten percent bigger than the maximum of `f` on interval $[0, 1]$. `minmaxf` is later used to crop the plots so that extreme values are not visible.

2c ⟨*utils* 2c⟩≡ (10) 3d ▷
```
pushup=:   ] + 0.1 * |
pushdown=: ] - 0.1 * |
```

2d ⟨*gendat* 2d⟩≡ (2a)
```
gendat=: 4 : 0
  X=: ? y $ 0
  Y=: x noise f X
  minmaxX=: (<./ , >./) X
  minmaxf=: (([: pushdown <./) , ([: pushup >./)) f steps 0 1 100
  ⟨testdat 8d⟩
)
```

`plotdat 0` plots the dataset.

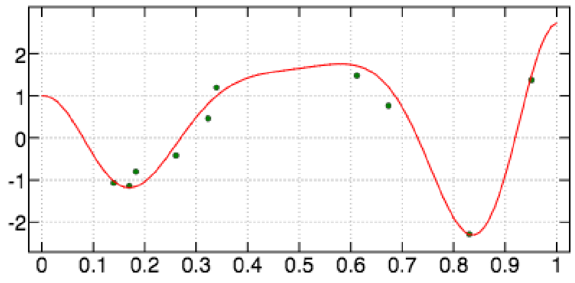
2e ⟨*plotdat* 2e⟩≡ (10)
```
plotdatnoshow=: 3 : 0
  ⟨initplot 3a⟩
  pd X;Y
  ⟨plotf 3b⟩
)
plotdat=: 3 : 0
  plotdatnoshow 0
  pd 'show'
)
```

3a ⟨*initplot* 3a⟩≡ (2e 9c)

```
pd 'reset'
pd 'color green'
pd 'type marker'
pd 'markersize 1'
pd 'markers circle'
```

3b ⟨*plotf* 3b⟩≡ (2e 9c)

```
pd 'color red'
pd 'type line'
pd 'pensize 1'
pd (;f) steps 0 1 100
```



```
   0.5 gendat 10
_2.53128 2.99011
   plotdat 0
```

polyreg 0 solves the linear system (1) and stores the coefficients of the polynomial in variable c.

3c ⟨*polyreg* 3c⟩≡ (10)

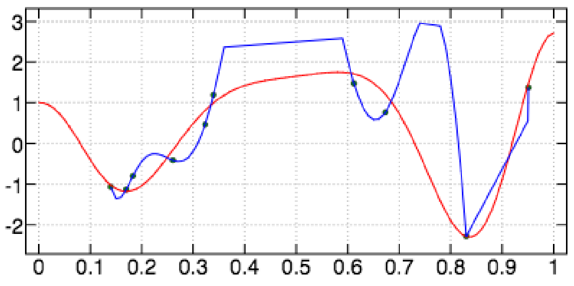
```
polyreg=: 3 : 0
  c=: Y ([ %. ] ^/ i.@#@]) X
  plotpoly 0
)
```

3d ⟨*utils* 2c⟩+≡ (10) ◁2c 5b▷

```
NB. identify the elements with values between {.x and {:x
sel=: (] >: {.@[) *. (] <: {:@[)
```

3e ⟨*plotpoly* 3e⟩≡ (10)

```
plotpoly=: 3 : 0
  plotdatnoshow 0
  pd 'color blue'
  xs=: (] #~ minmaxX"_ sel ]) /:~ X,steps 0 1 100
  pval=: c&p. xs
  crop=: minmaxf sel pval
  pd (crop # xs);(crop # pval)
  pd 'show'
)
```



```
polyreg 0
```

1.2 Generalization to a function space

Given a basis for a function space, we can try to express `f` as a combination of basis functions.

$$f(\boldsymbol{x}) = a_1 f_1(\boldsymbol{x}) + a_2 f_2(\boldsymbol{x}) + \cdots + a_n f_n(\boldsymbol{x})$$

Given a dataset of n pairs $\left(\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}\right)$, the coefficients a_i are found by solving a linear system.

$$\begin{pmatrix} f_1(\boldsymbol{x}^{(1)}) & f_2(\boldsymbol{x}^{(1)}) & \dots & f_n(\boldsymbol{x}^{(1)}) \\ f_1(\boldsymbol{x}^{(2)}) & f_2(\boldsymbol{x}^{(2)}) & \dots & f_n(\boldsymbol{x}^{(2)}) \\ \dots & \dots & \dots & \dots \\ f_1(\boldsymbol{x}^{(n)}) & f_2(\boldsymbol{x}^{(n)}) & \dots & f_n(\boldsymbol{x}^{(n)}) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \dots \\ a_n \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(n)} \end{pmatrix}$$

Let us denote this linear system by $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

1.3 Least squares

With more examples than the number of basis functions, the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ (with $\boldsymbol{A} \in \mathbb{R}^{m \times n}$) doesn't necessarily have a solution. Thus, we want to find an approximate solution $\boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{b}$ that minimizes the squares of the errors: $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$.

$$\begin{aligned}
& \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 \\
= & \{\, \|\boldsymbol{x}\|_2 = \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}} \} \\
& (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \cdot (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \\
= & \{\text{euclidean scalar product}\} \\
& (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^T (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \\
= & \{\text{property of transposition}\} \\
& \left(\boldsymbol{x}^T\boldsymbol{A}^T - \boldsymbol{b}^T\right)(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \\
= & \{\text{multiplication}\} \\
& \boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{b} - \boldsymbol{b}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{b} \\
= & \{\text{Since each element of the sum is a scalar, } \boldsymbol{b}^T\boldsymbol{A}\boldsymbol{x} = \left(\boldsymbol{b}^T\boldsymbol{A}\boldsymbol{x}\right)^T = \boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{b}\} \\
& \boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{b} + \boldsymbol{b}^T\boldsymbol{b}
\end{aligned}$$

To this quadratic expression corresponds a convex surface. Its minimum is found by setting the derivative to zero.

$$\mathbf{0} = 2\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{A}^T\boldsymbol{b}$$
$$=$$
$$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}$$

Thus, when $m > n$, we solve $\boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{b}$ by solving $\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}$. $\boldsymbol{A}^T\boldsymbol{A}$ is called the Gram matrix. `gram y` computes the Gram matrix `S` for a polynomial basis of degree `y-1`.

5a ⟨*gram* 5a⟩≡ (10) 5c ▷

```
gram=: 3 : 0
  A=: X ^/ i.y
  S=: (mp~ |:) A
)
```

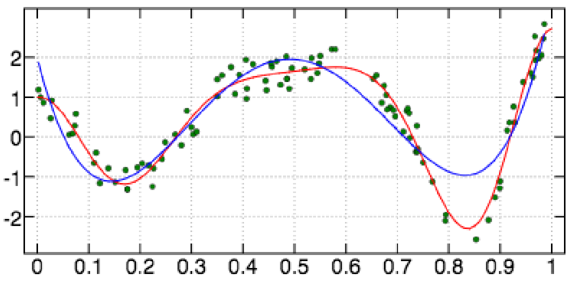
5b ⟨*utils* 2c⟩+≡ (10) ◁3d 6b ▷

```
mp=: +/ . * NB. matrix product
```

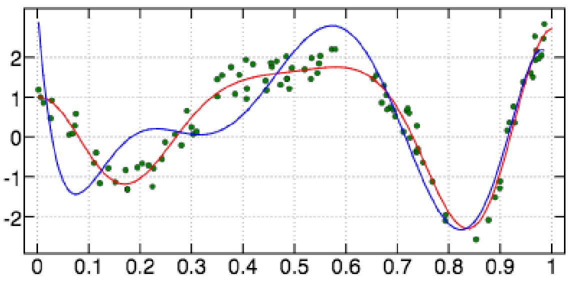
`leastsq y` solves the overdetermined linear system by computing the Gram matrix for a polynomial basis of degree `y-1`.

5c ⟨*gram* 5a⟩+≡ (10) ◁5a

```
leastsq=: 3 : 0
  gram y
  c=: ((|:A) mp Y) %. S
  plotpoly 0
)
```



```
   0.5 gendat 100
_2.53128 2.99011
   leastsq 5
```



```
   leastsq 8
```

1.4 Tikhonov regularization

With less examples than the number of basis functions (i.e. $m < n$, underdetermined system), $\boldsymbol{Ax} = \boldsymbol{b}$ doesn't have a unique solution. Even with $m \geq n$, the linear system can have approximate solutions more desirable than the optimal one. In particular, this is the case when several examples are very similar. For example, the solution to…

$$\begin{pmatrix} 1 & 1 \\ 1 & 1.00001 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0.99 \end{pmatrix}$$

…is $\boldsymbol{x}^T = (1001, -1000)$. However, the approximate solution $\boldsymbol{x}^T = (0.5, 0.5)$ is more suitable. Indeed, the optimal solution is not likely to adapt well to new inputs (e.g., input $(1, 2)$ would be projected onto $-999\ldots$).

Thus, when several solutions are feasible, we want to favor smaller norms $\|x\|_2$ by solving a new minimization problem:

$$\min_{\boldsymbol{x}} \|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2 + \alpha \|\boldsymbol{x}\|_2^2$$
$$\textit{with} \quad 0 < \alpha < 1$$

The minimum of this expression is found by setting its derivative to zero.

$$\begin{aligned} \boldsymbol{0} &= 2\boldsymbol{A}^T\boldsymbol{Ax} - 2\boldsymbol{A}^T\boldsymbol{b} + 2\alpha\boldsymbol{x} \\ &= \\ &\left(\boldsymbol{A}^T\boldsymbol{A} + \alpha\boldsymbol{I}_{n\times n}\right)\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b} \end{aligned}$$

It comes down to adding a small positive value to the diagonal of the Gram matrix. This approach has been given several names: Tikhonov regularization, ridge regression…

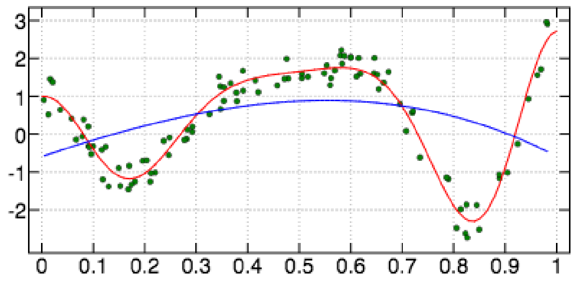
`1E_3 ridge 5` will solve the ridge regression for a polynomial basis of degree 5 and a regularization coefficient equal to 10^{-3}.

6a ⟨*ridge* 6a⟩≡ (10)

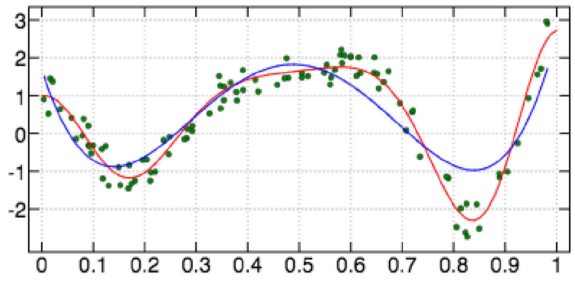
```
ridge=: 4 : 0
  gram y
  c=: ((|:A) mp Y) %.  x addDiag S
  plotpoly 0
)
```

6b ⟨*utils* 2c⟩+≡ (10) ◁5b 9a▷

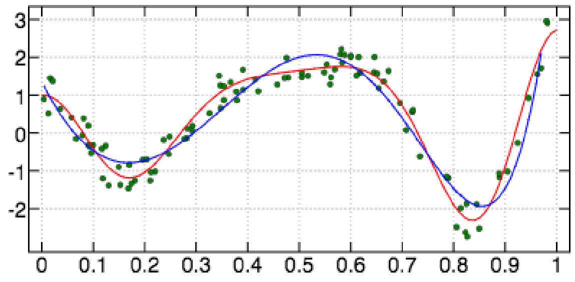
```
diag=: (<0 1)&|: : (([:(>:*i.)[:#])})
addDiag=: ([+diag@]) diag ] NB. add x to the diagonal of y
```



1E_4 ridge 4



1E_4 ridge 5



1E_4 ridge 8

1.5 Extreme Learning Machine

The following parametrized form of f corresponds to a single hidden layer neural network.

$$f(\boldsymbol{x}) = c_1 g(\boldsymbol{w_1} \cdot \boldsymbol{x} + b_1) + c_2 g(\boldsymbol{w_2} \cdot \boldsymbol{x} + b_2) + \cdots + c_M g(\boldsymbol{w_M} \cdot \boldsymbol{x} + b_M)$$

g is a non-linear activation function. We use the rectified linear unit (ReLU): $g(y) = max(0, y)$.

If vectors $\boldsymbol{w_1} \ldots \boldsymbol{w_M}$ and scalars $b_1 \ldots b_M$ are initialized randomly and never modified (i.e., if they are not parameters), we can solve a linear system $\boldsymbol{Hc} = \boldsymbol{y}$ of unknwon $\boldsymbol{c}$.

$$\boldsymbol{H} : \begin{pmatrix} g(\boldsymbol{w_1} \cdot \boldsymbol{x_1} + b_1) & \ldots & g(\boldsymbol{w_M} \cdot \boldsymbol{x_1} + b_M) \\ \ldots & \ldots & \ldots \\ g(\boldsymbol{w_1} \cdot \boldsymbol{x_N} + b_1) & \ldots & g(\boldsymbol{w_M} \cdot \boldsymbol{x_N} + b_M) \end{pmatrix}$$

$$\boldsymbol{c}^T : (c_1 \ldots c_M)$$
$$\boldsymbol{y}^T : (y_1 \ldots y_N)$$

This approach is named *Extreme Learning Machine* [1].

[1] `https://scholar.google.fr/scholar?q=extreme+learning+machine`

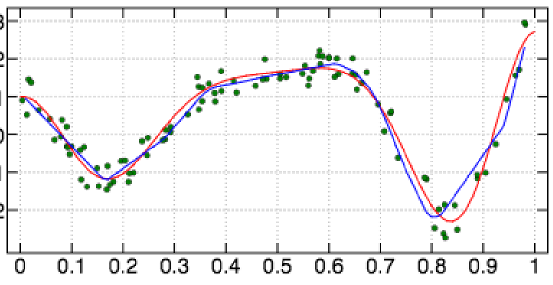
`initelm 100` initializes randomly matrix H with 100 neurons on the hidden layer (i.e., $M = 100$) and computes its Gram form `S`.

8a ⟨*elm* 8a⟩≡ (10) 8b▷
```
initelm=: 3 : 0
  W=: _1 + 2 * ? (y,1) $ 0 NB. input weights
  B=: ? y $ 0 NB. bias
  H=: mkH ,. X
  0 [ S=: (mp~ |:) H
)
mkH=: 3 : '0&>. B +"1 y mp"1/ W'
```

`elm 1E_4` solves the extreme learning machine linear system with a Tikhonov regularization coefficient of 10^{-4}.

8b ⟨*elm* 8a⟩+≡ (10) ◁8a
```
elm=: 3 : 0
  c=: ((|:H) mp Y) %.  y addDiag S
  plotelm 0
)
```

8c ⟨*plotelm* 8c⟩≡ (10)
```
plotelm=: 3 : 0
  plotdatnoshow 0
  pd 'type line'
  pd 'color blue'
  xs=: (] #~ minmaxX"_ sel ]) steps (<.<./X),(>.>./X),100
  pd xs;(mkH ,. xs) mp c
  pd 'show'
)
```



```
    initelm 100
0
    elm 1E_3
```

1.6 Test dataset

A test set is used to assert the capacity of the model to generalize on unseen data. Its size is fixed to 10% of the size of the training set.

8d ⟨*testdat* 8d⟩≡ (2d)
```
XT=: ? (>. 0.1 * y) $ 0
YT=: f XT
```

`test` computes the root mean square error (RMSE) on the test set.

9a ⟨*utils* 2c⟩+≡ (10) ◁6b

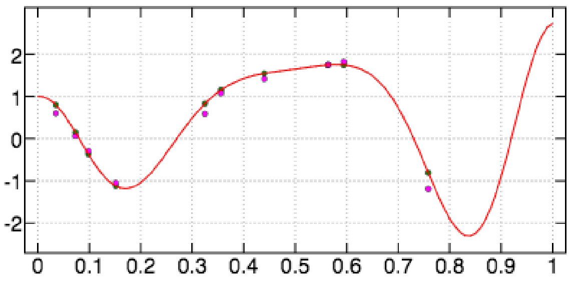
```
mean=: +/ % #
rmse=: [: %: [: mean ([: *: -)
```

9b ⟨*test* 9b⟩≡ (10)

```
test=: 3 : 0
  YThat=: (mkH ,. XT) mp c
  plottest 0
  YT rmse YThat
)
```

9c ⟨*plottest* 9c⟩≡ (10)

```
plottest=: 3 : 0
  ⟨initplot 3a⟩
  pd XT;YT
  pd 'color magenta'
  pd XT;YThat
  ⟨plotf 3b⟩
  pd 'show'
)
```



```
   test 0
0.172019
```

9d ⟨*require* 9d⟩≡ (10)

```
require'trig'
require'plot'
require'numeric'
```

10 ⟨*jelm.ijs* 10⟩≡
```
  ⟨require 9d⟩
  ⟨utils 2c⟩
  ⟨dataset 2a⟩
  ⟨plotdat 2e⟩
  ⟨plotpoly 3e⟩
  ⟨polyreg 3c⟩
  ⟨gram 5a⟩
  ⟨ridge 6a⟩
  ⟨plotelm 8c⟩
  ⟨elm 8a⟩
  ⟨plottest 9c⟩
  ⟨test 9b⟩
```