Primo report

Giacomo Longo (4336477) e Roberta Tassara (4336488)

14 Novembre 2019

1 Analisi di regressione

1.1 Introduzione

Il primo caso che analizzeremo riguarda la realizzazione di una stima mediante un approssimatore universale

$$h(x) = \sum_{i=0}^{p} c_i x^i$$

Tale approssimatore costruisce funzioni del tipo

$$h(x) = c_0$$

$$h(x) = c_1 x + c_0$$

$$h(x) = c_2 x^2 + c_1 x + c_o$$

La funzione ricercata deve minimizzare l'errore empirico

$$\min_{\underline{c}} \{\frac{1}{n} \sum_{i=1}^{n} (y_i - h(x_i))^2\} \qquad \underline{c} = \{c_0, ..., c_p\}$$

che rappresenta la somma delle distanze tra i dati osservati (y_i) e quelli della curva ottima ($h(x_i)$).
Scrivendo la funzione in forma matriciale, derivandola rispetto a c e ponendo il tutto = 0 si ottiene

$$\underline{c} = (X^T X)^+ X \underline{y}$$

che corrisponde al minimo globale

1.2 Analisi al variare di p con $y = 1 - x^2$

1.2.1 Parametri di simulazione

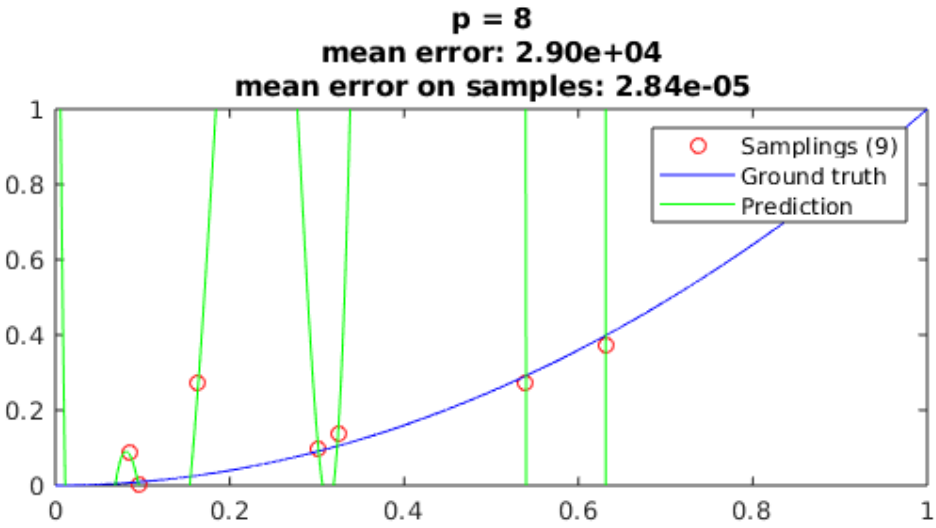
Dimensione dell'asse delle x: 10000 punti
Intervallo dell'asse delle x: $0 \leq x \leq 1$
Punti campione: 9
Intensitá di rumore: 10%

1.2.2 $p > n - 1$

L'analisi é stata omessa in quanto non rilevante: per tracciare n punti é sufficiente un polinomio di grado $n - 1$

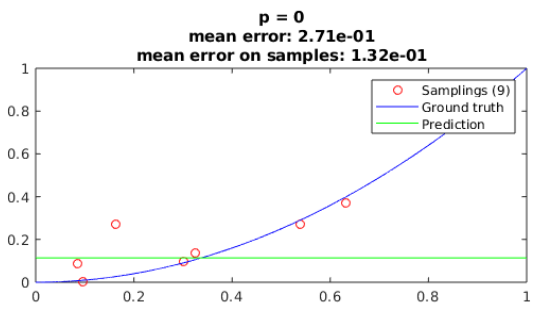
1.2.3 $p = n - 1$

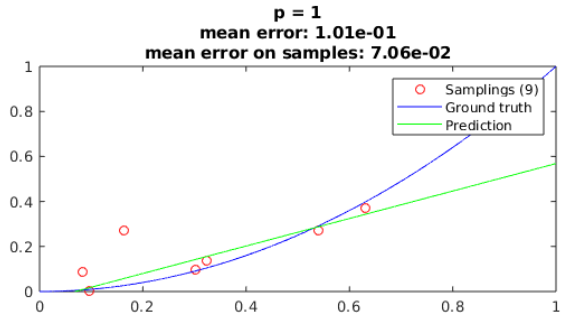
Se $p = n-1$ la funzione estimante(in verde) attraversa perfettamente tutti i punti campionati(in rosso), tuttavia non rappresenta una buona approssimazione della funzione originale (in blu). Infatti l'errore medio individuato é particolarmente alto.
Tale risultato, mostra che anche in prossimitá dei campioni con valori molto simili alla funzione reale, la funzione stimata si comporta in modo molto differente con vistose oscillazioni dovute ai termini di grado elevato.

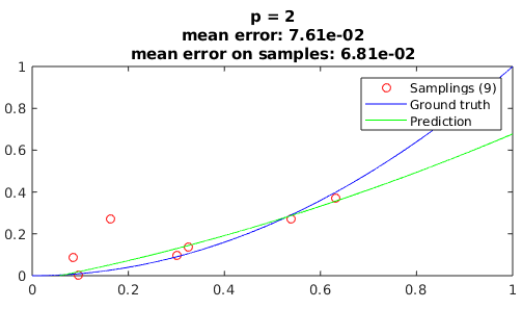


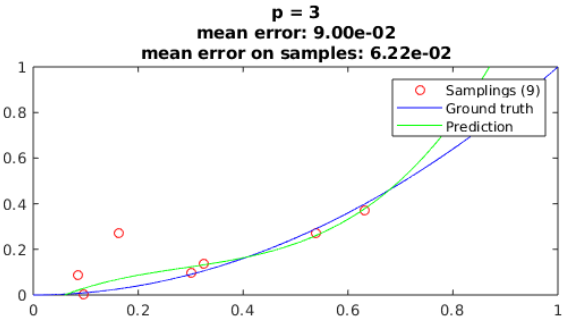
1.2.4 $p < n - 1$

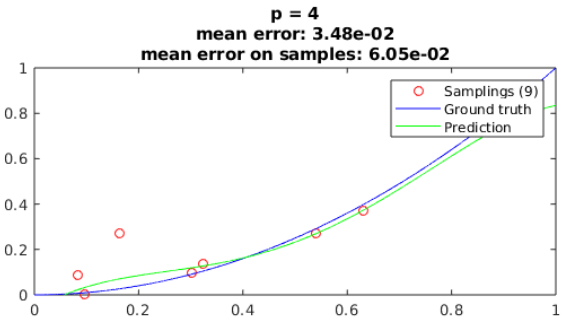
Per bassi valori di p, la funzione risultante é semplice.
Tra i vari valori di p, in questa particolare istanza, il valore che approssima meglio la funzione originale é $p = 4$ per il quale si ha il piú basso errore medio.
Questo significa che per questa funzione, l'errore di approssimazione é inferiore a quello di stima.
Colpisce anche che il miglior stimatore non coincida con il grado della funzione originaria, ció é dovuto al rumore.

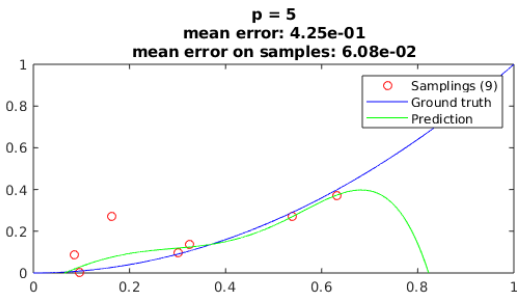


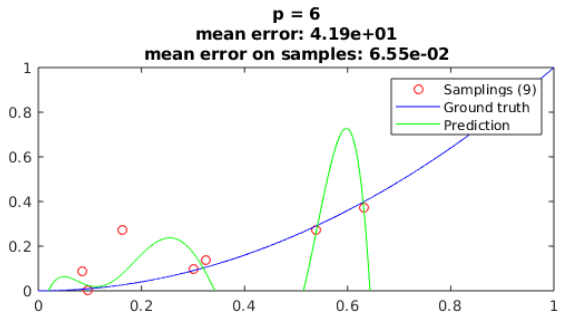


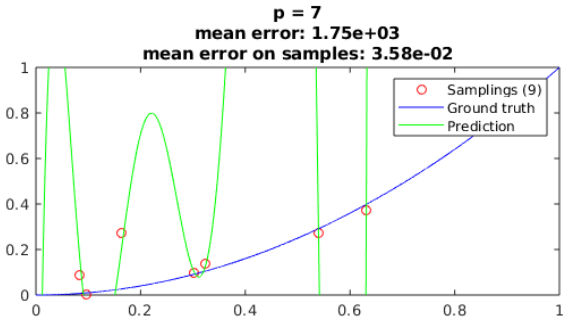





p = 4
mean error: 3.48e-02
mean error on samples: 6.05e-02
Samplings (9)
Ground truth
Prediction


p = 5
mean error: 4.25e-01
mean error on samples: 6.08e-02
Samplings (9)
Ground truth
Prediction


p = 6
mean error: 4.19e+01
mean error on samples: 6.55e-02
Samplings (9)
Ground truth
Prediction


p = 7
mean error: 1.75e+03
mean error on samples: 3.58e-02
Samplings (9)
Ground truth
Prediction

1.3 Codice

```
%% Cleanup
clear % Clear variables
clc % Clean console
close all % Close all plots

%% Original function (ground truth)
num_true = 10000;
x_true = linspace(0,1,num_true)'; % Generation of the X axis
y_true = groundTruth(x_true);

figure
hold on
grid on

tiledlayout(3,3) % Arrange plots in a grid

%% Sampling from ground truth
num_samples = 9; % Number of points to sample from the function
sigma = 0.1; % Noise strength
x_sampling = rand(num_samples, 1); % Take num_samples points on the x-axis
y_sampling = sampleWithGaussianNoise(x_sampling, sigma); % Sample and add gaussian noise

%% Learning from sample data
% We now build a function like sum over i {c * x^i}
p = num_samples - 1; % Degree of the approximating polynomial (should be between 1 and num_samples - 1)

learn_X = zeros(num_samples, p+1); % Matrix initialization
for i = 1:p+1
    learn_X(:,i) = x_sampling .^ (i - 1); % Calculation of x^i for all xs
end

forward_X = zeros(num_true, p+1);
for i = 1:p+1
    forward_X(:,i) = x_true .^ (i - 1);
end

for current_p = 0:p
    % Learning phase
    x_current = learn_X(:,1:current_p+1); % Rescale matrix
    c_current = (x_current' * x_current) \ (x_current' * y_sampling); % Recalculate coefficients

    y_current = forward_X(:,1:current_p+1) * c_current;
    average_error = mean(abs(y_current - y_true));

    y_samples = x_current * c_current;
    average_error_on_sample = mean(abs(y_samples - y_sampling));

    nexttile;
    plot(x_sampling, y_sampling, 'ro', x_true, y_true, 'b', x_true, y_current, 'g-');
    axis([min(x_true), max(x_true), min(y_true), max(y_true)]) % Scale axis to ground truth function
    title(sprintf('p = %d\nmean_error: %.2e\nmean_error_on_samples: %.2e', current_p, average_error, average_error_on_sampl
    legend(sprintf('Samplings (%d)', num_samples), 'Ground_truth', 'Prediction');
end

%% Definitions
function y = groundTruth(x)
    y = x.^2;
end
function y = sampleWithGaussianNoise(x, sigma)
    y = groundTruth(x) + sigma * randn(size(x));
end
```

2 Analisi di regressione in presenza del regolarizzatore

2.1 Introduzione

Per questa analisi si introduce un nuovo parametro λ detto *regolarizzatore*
La funzione da minimizzare diviene

$$\min_{\underline{c}} || X\underline{c} - \underline{y} ||^2 + \lambda || \underline{c} ||^2$$

Portando il calcolo di $\underline{c}$ a:

$$\underline{c} = (X^T X + \lambda I)^+ X\underline{y}$$

Il valore introdotto ha la funzione equivalente a un filtro low pass: i termini di grado piú alto vengono smorzati, "ammorbidendo" la funzione risultante.
$\lambda || \underline{c} ||$ rappresenta circa il numero di funzioni all'interno dello spazio delle ipotesi
e affinché questo valore sia convesso, viene elevato al quadrato.

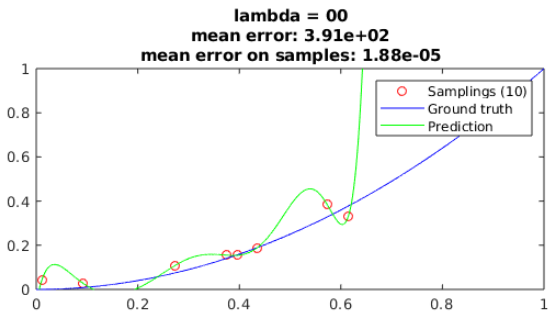
2.2 Parametri di simulazione

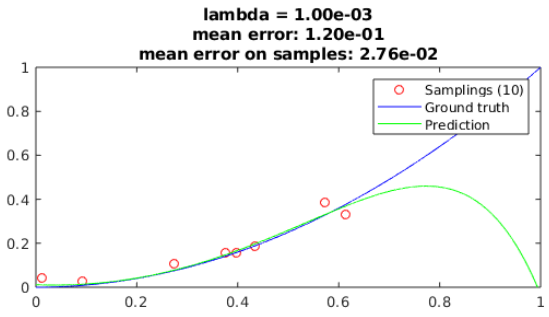
Dimensione dell'asse delle x: 10000 punti
Intervallo dell'asse delle x: $0 \leq x \leq 1$
Punti campione: 10
Intensitá di rumore: 10%

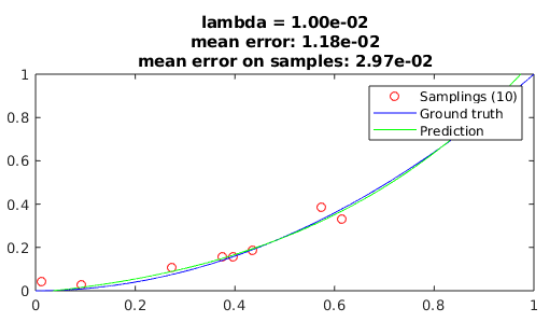
2.3 Analisi al variare di λ

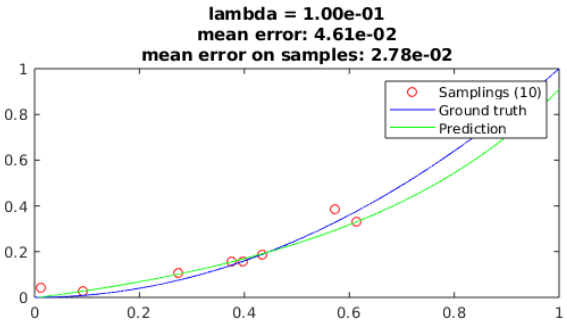
Per valori piccoli di λ si ha overfit (in zero si ha che la funzione attraversa esattamente tutti i punti campionati).
Con il crescere di λ, la struttura della funzione ricostruita si semplifica (i coefficienti associati ai termini piú alti in grado risultano smorzati).
Per valori di λ troppo elevati, la funzione diverge dai dati, tendendo a "appiattirsi".

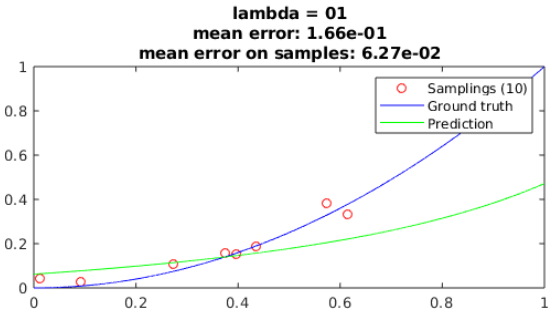
Il valore di λ che fornisce il minor errore rispetto alla funzione reale nel caso considerato é $\lambda = 1 * 10^{-2}$.
Non é possibile trarre una legge generale che associ l'errore medio al variare di λ. Per quanto riguarda l'errore medio rispetto ai campioni, valori di lambda inferiori sono associati a una maggiore vicinanza ai punti campionati, infatti per il caso limite $\lambda = 0$ si ha una funzione che attraversa esattamente tutti i punti utilizzati per la sua costruzione.

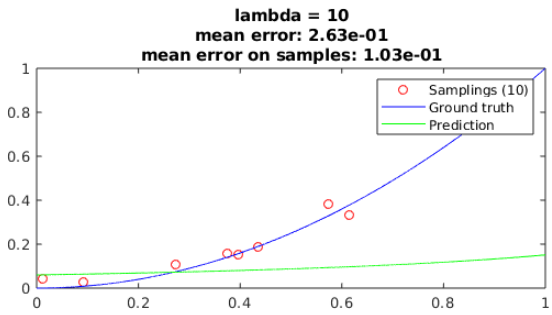





lambda = 1.00e-02
mean error: 1.18e-02
mean error on samples: 2.97e-02
Samplings (10)
Ground truth
Prediction


lambda = 1.00e-01
mean error: 4.61e-02
mean error on samples: 2.78e-02
Samplings (10)
Ground truth
Prediction


lambda = 01
mean error: 1.66e-01
mean error on samples: 6.27e-02
Samplings (10)
Ground truth
Prediction


lambda = 10
mean error: 2.63e-01
mean error on samples: 1.03e-01
Samplings (10)
Ground truth
Prediction

2.4 Codice

```
%% Cleanup
clear % Clear variables
clc % Clean console
close all % Close all plots

%% Original function (ground truth)
num_true = 10000;
x_true = linspace(0,1,num_true)'; % Generation of the X axis
y_true = groundTruth(x_true);

figure
hold on
grid on

tiledlayout(3,3) % Arrange plots in a grid

%% Sampling from ground truth
num_samples = 10; % Number of points to sample from the function
sigma = 0.05; % Noise strength
x_sampling = rand(num_samples, 1); % Take num_samples points on the x-axis
y_sampling = sampleWithGaussianNoise(x_sampling, sigma); % Sample and add gaussian noise

%% Learning phase
lambdas = logspace(-3,1, 9);
lambdas(2) = lambdas(1);
lambdas(1) = 0; % Test lambda = 0
for lambda = lambdas % Iterate over lambdas
    p = num_samples - 1; % Use maximum p
    X_learning = zeros(num_samples, p+1);
    for i = 0:p
        X_learning(:,i + 1) = x_sampling .^ i;
    end
    c = (X_learning' * X_learning + lambda * eye(p+1)) \ (X_learning'*y_sampling);

    % Forward
    X_forward = zeros(num_true, p+1);
    for i = 0:p
        X_forward(:,i + 1) = x_true .^ i;
    end
    Y_forward = X_forward * c;

    average_error = mean(abs(Y_forward - y_true));

    Y_samples = X_learning * c;
    average_error_on_samples = mean(abs(Y_samples - y_sampling));

    nexttile;
    plot(x_sampling, y_sampling, 'ro', x_true, y_true, 'b', x_true, Y_forward, 'g-');
    axis([min(x_true), max(x_true), min(y_true), max(y_true)]) % Scale axis to ground truth function
    title(sprintf('lambda = %.2d\nmean_error: %.2e\nmean_error_on_samples: %.2e', lambda, average_error, average_error_on_s
    legend(sprintf('Samplings (%d)', num_samples), 'Ground_truth', 'Prediction');
end

%% Definitions
function y = groundTruth(x)
    y = x.^2;
end
function y = sampleWithGaussianNoise(x, sigma)
    y = groundTruth(x) + sigma * randn(size(x));
end
```

3 Analisi di Monte Carlo dei due approcci

3.1 Parametri

Numero di trials: 1000

3.2 Regressione

```
p = 0, error = 2.66e-01
p = 1, error = 7.90e-02
p = 2, error = 5.64e-02
p = 3, error = 9.54e-02
p = 4, error = 2.27e-01
p = 5, error = 1.10e+00
p = 6, error = 9.00e+00
p = 7, error = 1.37e+02
p = 8, error = 2.96e+03
```

3.2.1 Codice

```
%% Cleanup
clear % Clear variables
clc % Clean console
close all % Close all plots

%% Original function (ground truth)
num_true = 10000;
x_true = linspace(0,1,num_true)'; % Generation of the X axis
y_true = groundTruth(x_true);

sigma = 0.1; % Noise strength
num_samples = 9; % Number of points to sample from the function
p = num_samples - 1; % Degree of the approximating polynomial (should be between 1 and num_samples - 1)

num_monte_carlo = 1000;
err_monte_carlo = zeros(num_monte_carlo, p);

for mc = 1:num_monte_carlo

    % Sampling from ground truth
    x_sampling = rand(num_samples, 1); % Take num_samples points on the x-axis
    y_sampling = sampleWithGaussianNoise(x_sampling, sigma); % Sample and add gaussian noise

    learn_X = zeros(num_samples, p+1); % Matrix initialization
    for i = 1:p+1
        learn_X(:,i) = x_sampling .^ (i - 1); % Calculation of x^i for all xs
    end

    forward_X = zeros(num_true, p+1);
    for i = 1:p+1
        forward_X(:,i) = x_true .^ (i - 1);
    end

    for current_p = 0:p
        % Learning phase
        x_current = learn_X(:,1:current_p+1); % Rescale matrix
        c_current = (x_current' * x_current) \ (x_current' * y_sampling); % Recalculate coefficients

        y_current = forward_X(:,1:current_p+1) * c_current;
        average_error = mean(abs(y_current - y_true));

        err_monte_carlo(mc, current_p + 1) = average_error;

        y_samples = x_current * c_current;
        average_error_on_sample = mean(abs(y_samples - y_sampling));
    end

end

%% Print results
for i = 0:p
    errors = err_monte_carlo(:, i+1);
    error_on_p = mean(errors);
    fprintf('p = %d, error = %.02e\n', i, error_on_p);
end

%% Definitions
function y = groundTruth(x)
    y = x.^2;
end
function y = sampleWithGaussianNoise(x, sigma)
    y = groundTruth(x) + sigma * randn(size(x));
end
```

3.3 Regressione con regolarizzatore

```
lambda = 0.00e+00, error = 1.00e+04
lambda = 1.00e-03, error = 3.69e-02
lambda = 1.00e-02, error = 2.94e-02
lambda = 3.16e-02, error = 2.80e-02
lambda = 1.00e-01, error = 3.11e-02
lambda = 3.16e-01, error = 4.39e-02
lambda = 1.00e+00, error = 6.99e-02
lambda = 3.16e+00, error = 1.11e-01
lambda = 1.00e+01, error = 1.65e-01
```

Come precedentemente indicato, l'errore maggiore si commette per $\lambda = 0$.
Con la crescita di λ, dopo una certa soglia, l'errore incrementa monotonicamente.

3.3.1 Codice

```
%% Cleanup
clear % Clear variables
clc % Clean console
close all % Close all plots

%% Original function (ground truth)
num_true = 10000;
x_true = linspace(0,1,num_true)'; % Generation of the X axis
y_true = groundTruth(x_true);

%% Learning phase
lambdas = logspace(-3,1, 9);
lambdas(2) = lambdas(1);
lambdas(1) = 0; % Test lambda = 0

num_samples = 10; % Number of points to sample from the function
sigma = 0.05; % Noise strength

num_monte_carlo = 1000;
err_monte_carlo = zeros(num_monte_carlo, length(lambdas));

for mc = 1:num_monte_carlo
    ilambda = 1;

    x_sampling = rand(num_samples, 1); % Take num_samples points on the x-axis
    y_sampling = sampleWithGaussianNoise(x_sampling, sigma); % Sample and add gaussian noise

    for lambda = lambdas % Iterate over lambdas

        p = num_samples - 1; % Use maximum p
        X_learning = zeros(num_samples, p+1);
        for i = 0:p
            X_learning(:,i + 1) = x_sampling .^ i;
        end
        c = (X_learning' * X_learning + lambda * eye(p+1)) \ (X_learning'*y_sampling);

        % Forward
        X_forward = zeros(num_true, p+1);
        for i = 0:p
            X_forward(:,i + 1) = x_true .^ i;
        end
        Y_forward = X_forward * c;

        average_error = mean(abs(Y_forward - y_true));

        err_monte_carlo(mc, ilambda) = average_error;
        ilambda = ilambda + 1;
    end
end

%% Print results
for i = 1:length(lambdas)
    errors = err_monte_carlo(:, i);
    error_on_p = mean(errors);
    fprintf('lambda = %.02e, error = %.02e\n', lambdas(i), error_on_p);
end

%% Definitions
function y = groundTruth(x)
    y = x.^2;
end
function y = sampleWithGaussianNoise(x, sigma)
    y = groundTruth(x) + sigma * randn(size(x));
end
```