Regressione

- Lineare
  - Simple linear regression
  - Multiple linear regression
  - Regression vs Geometrical fitting
- Non lineare
  - Variabile indipendente non lineare
  - Ottimizzazione numerica (metodi iterativi)

Definizioni

- Nei problemi di classificazione supervisionata ad ogni pattern $\mathbf{x}_i$ del training set è associata un'etichetta y_i (classe di appartenenza). Obiettivo del training è l'apprendimento di un mapping dallo spazio dei pattern a quello (discreto) delle etichette.
- Nella regressione i valori y_i sono numerici e continui (in $\Re$), e l'obiettivo del training è l'apprendimento di una funzione $f(\mathbf{x}) \rightarrow y$.
  - f è controllata da un insieme di parametri. Se la dipendenza dai parametri è lineare (es. assenza di elevazioni a potenza dei parametri nella definizione di f), la regressione si definisce lineare.
  - Nella regressione $\mathbf{x}$ è detta variabile indipendente e y variabile dipendente.
  - Si assume che la variabile indipendente sia esatta mentre la variabile dipendente sia affetta da errore (es. imprecisione di misura).
  - Se la variabile indipendente è uno scalare x (i.e., una sola variabile indipendente), parliamo di simple linear regression (retta di regressione)
  - Se la variabile indipendente è un vettore $\mathbf{x}$ (i.e., più variabili indipendenti), parliamo di multiple linear regression (esempio: iperpiano di regressione).

Simple Linear Regression

- Dato un training set TS contenente n pattern $(x_1, y_1) \ldots (x_n, y_n)$, e la seguente dipendenza tra variabile indipendente e dipendente:

$$y_i = \alpha + \beta \cdot x_i + \varepsilon_i$$

dove:

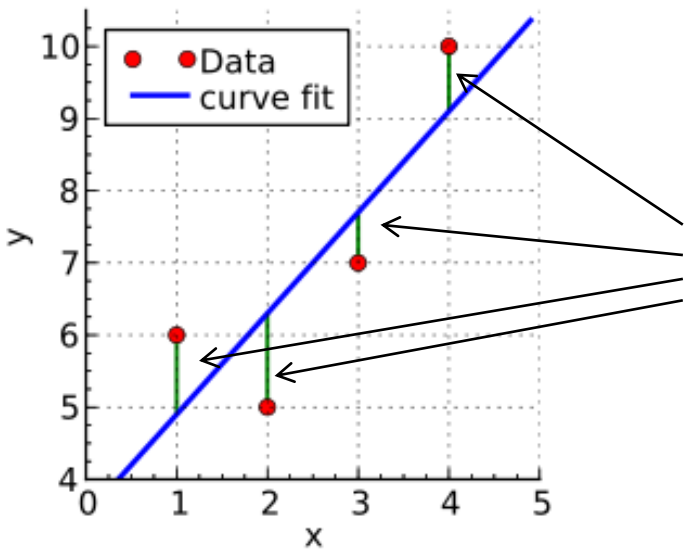
- ε_i è l'errore di misura (incognito) del pattern x_i
- α, β sono parametri da determinare

la soluzione ai minimi quadrati (Least Square) del problema consiste nel determinare l'equazione della retta:

$$f(x) = y = \alpha + \beta \cdot x$$

che minimizza la somma dei quadrati dei residui:

$$\alpha^*, \beta^* = arg \min_{\alpha,\beta} \sum_{i=1\ldots n} (f(x) - y_i)^2 = \sum_{i=1\ldots n} {\varepsilon_i}^2$$



Simple Linear Regression (2)

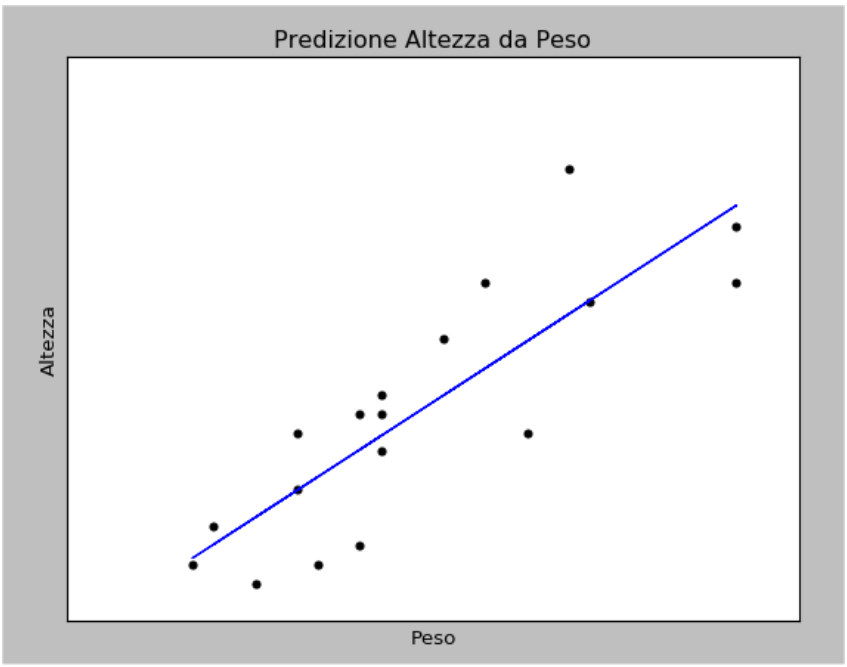
- Il precedente problema di minimizzazione può essere facilmente risolto uguagliando a zero le derivate parziali della funzione obiettivo rispetto ai parametri α, β ottenendo le cosiddette equazioni normali.
  - risolvendo il sistema (due equazioni in due incognite) si ottiene:

$$\beta^* = \frac{\sum_{i=1\ldots n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1\ldots n}(x_i - \bar{x})^2}, \qquad \alpha^* = \bar{y} - \beta^* \bar{x}$$

    dove $\bar{x}$ e $\bar{y}$ sono la media degli x_i e y_i rispettivamente.

    Necessari almeno due punti (con x_i diverse), altrimenti il denominatore si annulla.

- Esempio: predizione dell'altezza a partire dal peso (scikit-learn).



Peso	Altezza
72	173
54	159
65	172
58	170
62	165
72	176
60	173
64	179
55	166
46	158
52	158
47	160
55	167
54	166
55	164
49	157
51	165
51	162

Multiple Linear Regression

- Generalizziamo ora a iperpiani ($\mathbf{x} \in \Re^d$).
- Per semplicità di trattazione sia $\mathbf{X}$ la matrice rettangolare ($n \times (d+1)$) ottenuta inserendo sulle righe gli n pattern del TS, e avendo cura di inserire a destra una colonna con tutti valori 1.

$$\mathbf{X} = \begin{bmatrix} x_{11} & \cdots & x_{1d} & 1 \\ x_{21} & & x_{2d} & 1 \\ \vdots & \ddots & \vdots & \\ x_{n1} & \cdots & x_{nd} & 1 \end{bmatrix}$$

pattern $\mathbf{x}_1$

- Siano:

uno scalare per ogni pattern

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \text{e} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{d+1} \end{bmatrix}$$

termine noto

i vettori costruiti a partire dai valori della variabile indipendente (uno per ogni pattern) e dai $d+1$ parametri da determinare.

- Le relazioni tra variabili indipendenti e dipendente possono essere scritte attraverso le equazioni:

$$y_i = \sum_{j=1\ldots d+1} x_{ij} \cdot \beta_j \qquad \text{per} \quad i = 1 \ldots n$$

indici i, j riferiti alla matrice $\mathbf{X}$

o in forma matriciale come:

$$\mathbf{y} = \mathbf{X}\,\boldsymbol{\beta}$$

Per $n > (d+1)$, $\mathbf{X}$ è rettangolare $\rightarrow$ sistema di equazioni sovradeterminato.

Multiple Linear Regression (2)

- In formato matriciale la funzione obiettivo da minimizzare (Least Square) può essere scritta come:

$$\|\mathbf{y} - \mathbf{X}\,\boldsymbol{\beta}\|^2$$

Ancora una volta le equazioni normali possono essere ottenute derivando rispetto ai parametri. Per la derivazione «elegante» in formato matriciale vedi:

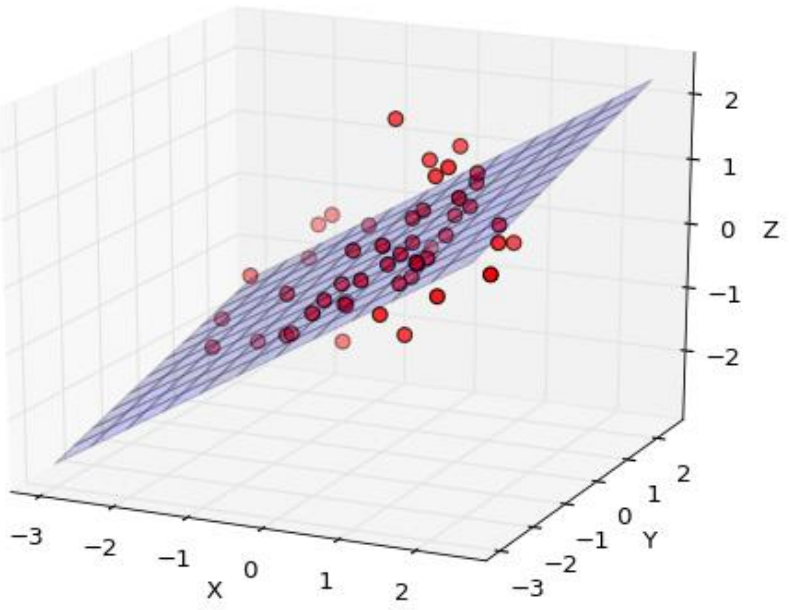
https://en.wikipedia.org/wiki/Linear_least_squares_(mathematics).

$$(\mathbf{X}^t\mathbf{X})\boldsymbol{\beta}^* = \mathbf{X}^t\mathbf{y}$$

Infine, invertendo otteniamo i parametri del modello:

$$\boldsymbol{\beta}^* = (\mathbf{X}^t\mathbf{X})^{-1}\,\mathbf{X}^t\mathbf{y}$$

Esempio d = 2:



Multiple Linear Regression (3)

- Nota bene:
  - Se $n = (d+1)$, la matrice $\mathbf{X}$ è quadrata e l'iperpiano di regressione tocca tutti i punti (interpolazione)
  - Se $n > (d+1)$, il numero di equazioni è superiore al numero di incognite e l'iperpiano approssima i punti.
  - Per essere invertibile la matrice $\mathbf{X}^t\mathbf{X}$ deve essere a rango massimo: problemi in caso di punti collineari (colonne di $\mathbf{X}$ linearmente dipendenti).
  - La matrice $\mathbf{X}^t\mathbf{X}$ è spesso «mal condizionata» e il calcolo dell'inversa può risultare numericamente instabile. Il problema può essere risolto in modo numericamente più stabile attraverso decomposizione SVD (Singular Value Decomposition) di $\mathbf{X}$:

$$\mathbf{X} = \mathbf{U}\boldsymbol{\Gamma}\mathbf{V}^t$$

Dove $\mathbf{U}$ è una matrice ortogonale $n \times n$, $\mathbf{V}$ è una matrice ortogonale $(d+1) \times (d+1)$ e $\boldsymbol{\Gamma}$ una matrice diagonale rettangolare $n \times (d+1)$ con tutti gli elementi fuori diagonale uguali a 0.

Si dimostra che:

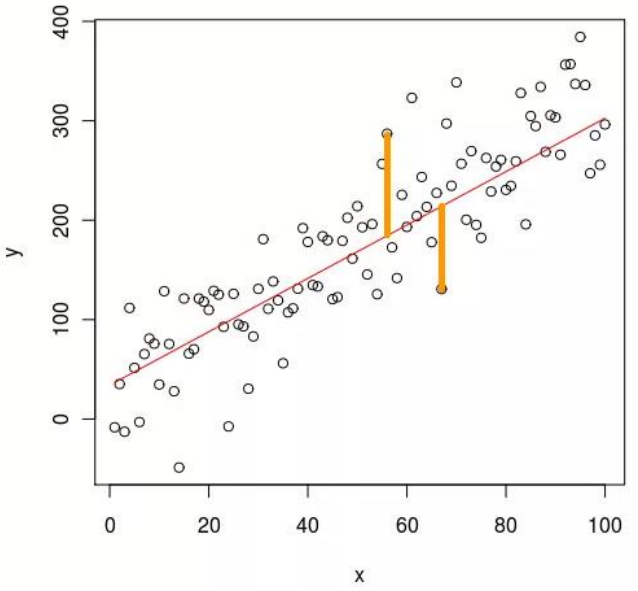
$$\boldsymbol{\beta}^* = \mathbf{V}\boldsymbol{\Gamma}^+\mathbf{U}^t\,\mathbf{y}$$

dove $\boldsymbol{\Gamma}^+$ (denominata pseudoinversa di $\boldsymbol{\Gamma}$), vista la particolare forma di $\boldsymbol{\Gamma}$, si ottiene semplicemente invertendo gli elementi non nulli di $\boldsymbol{\Gamma}$ e calcolando la trasposta.

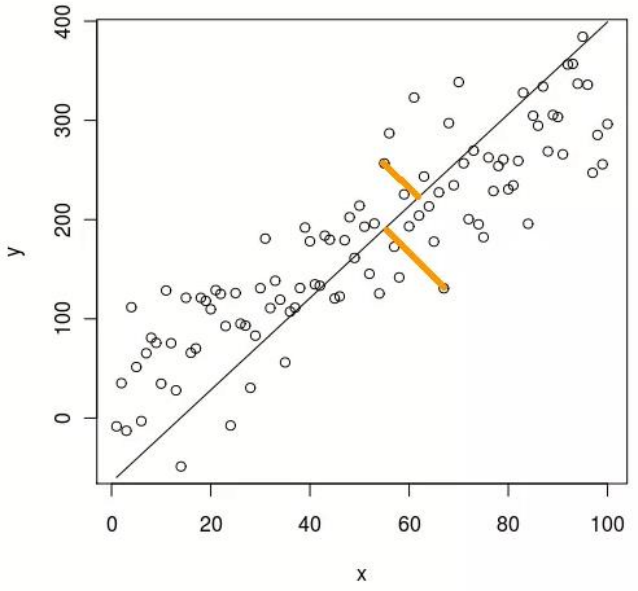
Regression vs Geometrical fitting

- *La regressione lineare (least square estimator) è un buon metodo per il fitting geometrico di nuvole di punti 2D o 3D?*

Linear Regression
(Least Square)



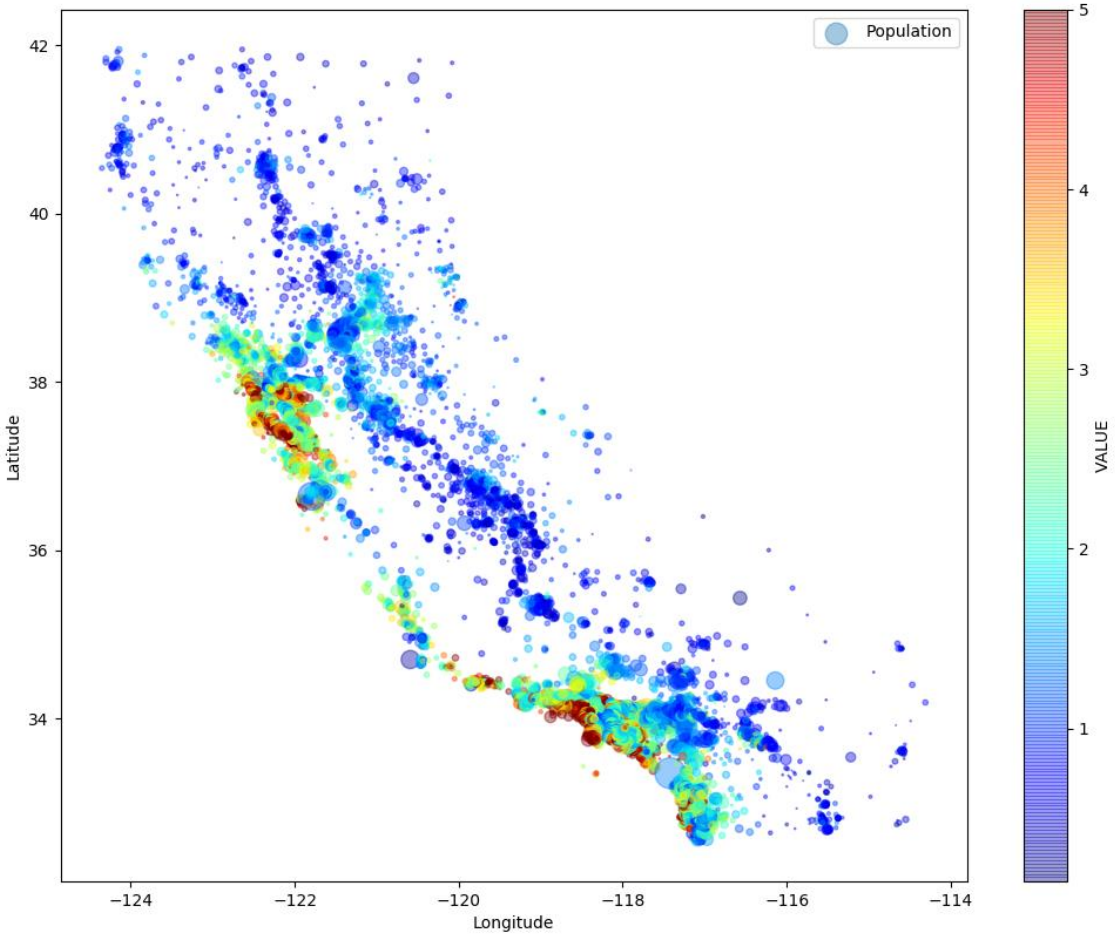
PCA
(Principal Component Analysis)



- La regressione lineare (least square) minimizza le distanze «verticali» dall'iperpiano e non le distanze euclidee (cosa semplicemente ottenibile, come vedremo, attraverso PCA).
- Ciò è conseguenza del fatto che le variabili indipendente e dipendente non hanno ruolo simmetrico (la variabile indipendente si considera esatta, mentre quella dipendente affetta da errore). Nota: scambiando le due variabili il risultato cambia.
- Nel caso di nuvole di punti geometrici le coordinate hanno lo stesso ruolo e PCA fitting è preferibile.

Esempio: stima prezzo case

- California housing dataset contiene 20640 pattern, ciascuno riferito a un quartiere (gruppo di case) in California. I pattern sono 9 dimensionali, per ciascuno di essi:
  - la variabile indipendente è costituita dal vettore 8-dimensionale [MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup, Latitude, Longitude]
  - la variabile dipendente (valore target) dal costo medio delle case nel quartiere [Value] (1 ≡100K$).
- Pandas è un tool Python utile per esplorare i dati:



Stima con Multiple Linear Regression

- Split dei dati in training e test (80 - 20%):

```
X_train, X_test, y_train, y_test = train_test_split(housing.data,
                                   housing.target,test_size = 0.2)
```

- Con la funzione Scikit-Learn di regressione lineare:

```
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_train_predicted = lin_reg.predict(X_train)
rmse = np.sqrt(mean_squared_error(y_train, y_train_predicted))
print('Train RMSE: ', rmse)
y_test_predicted = lin_reg.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_test_predicted))
print('Test RMSE: ', rmse)
```

- Con implementazione esplicita ($\boldsymbol{\beta}^* = (\mathbf{X}^t\mathbf{X})^{-1}\,\mathbf{X}^t\mathbf{y}$):

```
X_train_b = np.c_[np.ones((X_train.shape[0],1)), X_train]
X_test_b = np.c_[np.ones((X_test.shape[0],1)), X_test]
beta_star = np.linalg.inv(X_train_b.T.dot(X_train_b)).dot(
X_train_b.T).dot(y_train)
y_train_predicted = X_train_b.dot(beta_star)
…
```

Stesso risultato (ultimi decimali a parte)

Train RMSE: 0.7233

Test RMSE: 0.7274 (72K$ di scostamento medio)

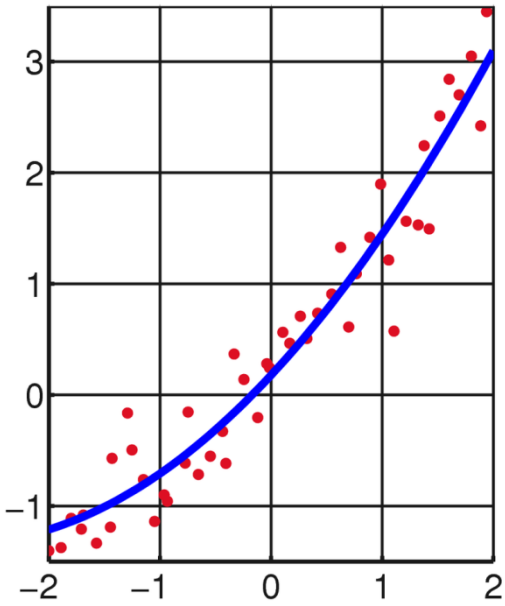
Variabile indipendente non lineare

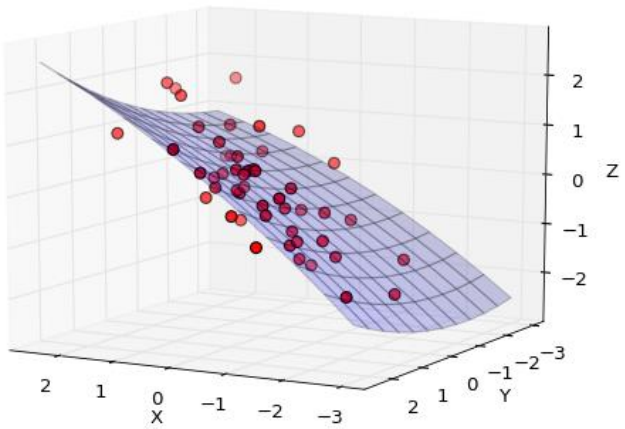
- Con un semplice accorgimento è possibile continuare ad utilizzare la regressione lineare per approssimare i dati con curve e superfici:
  - Nella teoria introdotta infatti l'importante è che la funzione di regressione sia lineare rispetto ai parametri e non rispetto alla variabile indipendente, i cui elementi possono comparire elevati a potenza, combinati tra loro, come argomento di funzioni trascendenti, ecc.

- Esempio: per $d = 1$ e modello quadratico $y = \beta_1 x^2 + \beta_2 x + \beta_3$ scriviamo $\mathbf{X}$ come:

$$\mathbf{X} = \begin{bmatrix} {x_1}^2 & x_1 & 1 \\ {x_2}^2 & x_2 & 1 \\ \vdots & \vdots & \vdots \\ {x_n}^2 & x_n & 1 \end{bmatrix}$$

ottenendo →





← Esempio $d = 2$

Regressione non lineare

- Nel caso di dipendenza non lineare del modello dai parametri non esiste soluzione least square in «forma-chiusa».

  Esempio dalla biologia (enzyme-mediated reaction)

$$y = \frac{\beta_1 x}{\beta_2 + x}$$

- Per la risoluzione si applicano quindi tecniche iterative di ottimizzazione numerica:

  - È possibile utilizzare il classico algoritmo di Gradient Descent (metodo del primo ordine che esegue passi in direzione contraria al gradiente).

  - L'algoritmo più noto e utilizzato è quello di Gauss-Newton (metodo del secondo ordine che richiede stima Hessiana).

    https://en.wikipedia.org/wiki/Gauss–Newton_algorithm

In pratica

- Per problemi di regressione di piccole-medie dimensioni:
  - Nel caso lineare (rispetto ai parametri) la soluzione least square in forma chiusa (con fattorizzazione SVD) è l'approccio più utilizzato.
  - Nel caso non lineare Gauss-Newton è la prima scelta
- Per problemi di regressione di grandi dimensioni (anche lineari) l'approccio iterativo Gradient Descent è spesso preferibile per motivi di trattabilità numerica e complessità computazionale.
- Per alcune tecniche di classificazione studiate esiste una variante efficace per problemi di regressione. Casi notevoli sono:
  - Support Vector Machine
  - Random Forest
- Nell'ambito del deep learning, reti neurali addestrate con gradient descent sono state utilizzate con successo per risolvere problemi di regressione di enormi dimensioni come la stima della posizione di un oggetto in un'immagine a partire da milioni di esempi (vedi Faster R-CNN)
  - la variabile indipendente è un immagine (pixel)
  - La variabile dipendente la bounding box dell'oggetto (trattasi in questo caso di un vettore $\rightarrow$ multivariate multiple regression).

Prezzi case: modelli non lineari

- Random Forest Regressor (da Scikit-Learn):

```
forest_reg = RandomForestRegressor(n_estimators = 30,
                                   max_features = 8)

forest_reg.fit(X_train, y_train)

y_train_predicted = forest_reg.predict(X_train)

…
```

Train RMSE: 0.1970

Test RMSE: 0.5080 (il modello lineare era troppo semplice …)

- Rete Neurale 4 livelli + Gradient Descent (da TensorFlow):

```
…

hidden1 = tf.layers.dense(X, 100, name = "hidden1", activation =
                          tf.nn.relu)

hidden2 = tf.layers.dense(hidden1, 100, name = "hidden2",
                          activation = tf.nn.relu)

y_pred = tf.layers.dense(hidden2, 1, name = "logits")

…
```

Train RMSE: 0.4647

Test RMSE: 0.5198 (sembra non si possa fare molto meglio …)