

# Regressione Non Lineare

## Analisi Predittiva di Variabili Continue

### Programmazione di Applicazioni Data Intensive

Laurea in Ingegneria e Scienze Informatiche  
DISI – Università di Bologna, Cesena

Proff. Gianluca Moro, Roberto Pasolini  
*nome.cognome@unibo.it*



## Regressione Polinomiale

- La *regressione polinomiale* è una generalizzazione di quella lineare con altri **termini di grado superiore**
  - per ottenere modelli capaci di descrivere data set più complessi
  - altre funzioni non lineari: e.g. *Gaussian Radial Basis Function*
- Ad esempio, con una sola variabile indipendente, un modello polinomiale di grado 3 ha 4 termini con altrettanti parametri  

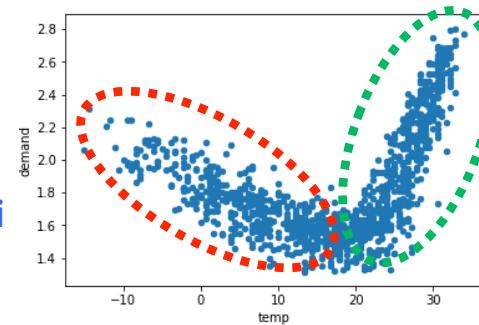
$$h(x) = \theta_0 + \theta_1 \cdot x + \theta_2 \cdot x^2 + \theta_3 \cdot x^3$$
 dove  $x$  è sempre il dato di input
- Su 2 variabili  $a$  e  $b$  un modello di grado 2 ha invece 6 termini  

$$h(a, b) = \theta_0 + \theta_1 \cdot a + \theta_2 \cdot a^2 + \theta_3 \cdot b + \theta_4 \cdot a \cdot b + \theta_5 \cdot b^2$$
- la regressione polinomiale è **ancora lineare, rispetto ai parametri  $\theta$** , non nelle variabili dei dati  $a, b$  che però sono noti
  - la funzione d'errore è definita in uno spazio a maggiori dimensioni  $\theta$
  - l'algoritmo di regressione rimane il medesimo di quella lineare



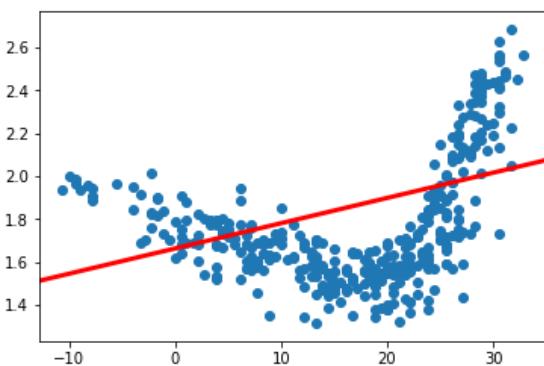
## Esempio: Predizione del Consumo di Elettricità sull'Intero Anno

- Abbiamo visto come esempio la **predizione dei consumi nei mesi estivi**, durante i quali si usa aria condizionata
- Ci si aspetta però un aumento dei consumi anche nei mesi più freddi, per via dell'uso di sistemi di riscaldamento elettrici
- Visualizzando i dati di tutto l'anno in un grafico a dispersione, si nota un **aumento dei consumi anche con temperature basse**
- Possiamo addestrare un **modello di regressione unico** che permetta di effettuare predizioni sui consumi per tutto l'anno ?



## Esempio: Predizione Consumi su tutto l'anno con Regressione Lineare

- Addestrando un **modello lineare** su questi dati, l'approssimazione è **inaccurata**
  - l'err. quadratico medio sui dati di validazione è 0,081
  - l'errore relativo è del 14,4%
  - il coefficiente  $R^2$  è 0,028
- Come si vede dal grafico, i dati non sono approssimabili in modo soddisfacente con una retta
- Possiamo ottenere un'approssimazione migliore con un modello polinomiale ?



## Esempio: Predizione Consumi su tutto l'anno con Regressione Polinomiale di 2° grado (i)

- Con un modello polinomiale di secondo grado, la formula che stima il consumo  $y$  dalla temperatura  $x$  diventa:

$$\hat{y} = \theta_0 + \theta_1 \cdot x + \theta_2 \cdot x^2$$

- L'errore quadratico medio è quindi:

$$E(\theta) = \frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 \cdot x_i + \theta_2 \cdot x_i^2 - y_i)^2$$

data	temp. media	picco consumo
01/06/2016	$x_1 = 25,2^\circ\text{C}$	$y_1 = 2,13 \text{ GW}$
02/06/2016	$x_2 = 27,1^\circ\text{C}$	$y_2 = 2,21 \text{ GW}$
03/06/2016	$x_3 = 26,9^\circ\text{C}$	$y_3 = 2,22 \text{ GW}$

- Ad esempio, con le 3 osservazioni viste in precedenza la funzione di errore da minimizzare è

$$\begin{aligned} E(\theta) &= \frac{1}{3} \left( (\theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_1^2 - y_1)^2 + (\theta_0 + \theta_1 \cdot x_2 + \theta_2 \cdot x_2^2 - y_2)^2 + (\theta_0 + \theta_1 \cdot x_3 + \theta_2 \cdot x_3^2 - y_3)^2 \right) \\ &= \frac{1}{3} \left( (\theta_0 + 25,2\theta_1 + 635,04\theta_2 - 2,13)^2 + (\theta_0 + 27,1\theta_1 + 734,41\theta_2 - 2,21)^2 + (\theta_0 + 26,9\theta_1 + 723,6\theta_2 - 2,22)^2 \right) \end{aligned}$$

- calcoliamo il vettore gradiente di  $\theta_0, \theta_1, \theta_2$



## Esempio: Predizione Consumi su tutto l'anno con Regressione Polinomiale di 2° grado (ii)

- Per calcolare il vettore gradiente, calcoliamo le derivate parziali di  $E(\theta)$

$$E(\theta) = \frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 \cdot x_i + \theta_2 \cdot x_i^2 - y_i)^2$$

$$\frac{\partial E(\theta)}{\theta_0} = \frac{2}{m} \sum_{i=1}^m (\theta_0 + \theta_1 \cdot x_i + \theta_2 \cdot x_i^2 - y_i)$$

$$\frac{\partial E(\theta)}{\theta_1} = \frac{2}{m} \sum_{i=1}^m (\theta_0 + \theta_1 \cdot x_i + \theta_2 \cdot x_i^2 - y_i) \cdot x_i$$

$$\frac{\partial E(\theta)}{\theta_2} = \frac{2}{m} \sum_{i=1}^m (\theta_0 + \theta_1 \cdot x_i + \theta_2 \cdot x_i^2 - y_i)^2 \cdot x_i^2$$



## Esempio: Predizione Consumi su tutto l'anno con Regressione Polinomiale di 2° grado (iii)

- funzione d'errore da minimizzare

$$E(\boldsymbol{\theta}) = \frac{1}{3} \left( (\theta_0 + 25.2\theta_1 + 635.04\theta_2 - 2.13)^2 + (\theta_0 + 27.1\theta_1 + 734.41\theta_2 - 2.21)^2 + (\theta_0 + 26.9\theta_1 + 723.6\theta_2 - 2.22)^2 \right)$$

- sostituiamo i dati noti  $x$  e  $y$  nelle derivate e otteniamo

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_0} = -4.37 + 2\theta_0 + 52.8\theta_1 + 1395.37\theta_2$$

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_1} = -115.52 + 52.8\theta_0 + 1395.37\theta_1 + 36913.6\theta_2$$

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_2} = -3054.72 + 1395.37\theta_0 + 36913.6\theta_1 + 977487\theta_2$$

- quindi il gradiente (l'apice  $k$  è lo step  $k$ -esimo di discesa per  $\theta_0, \theta_1, \theta_2$ )

$$\theta_0^{k+1} = \theta_0^k - \eta(-4.37 + 2\theta_0^k + 52.8\theta_1^k + 1395.37\theta_2^k)$$

$$\theta_1^{k+1} = \theta_1^k - \eta(-115.52 + 52.8\theta_0^k + 1395.37\theta_1^k + 36913.6\theta_2^k)$$

$$\theta_2^{k+1} = \theta_2^k - \eta(-3054.72 + 1395.37\theta_0^k + 36913.6\theta_1^k + 977487\theta_2^k)$$

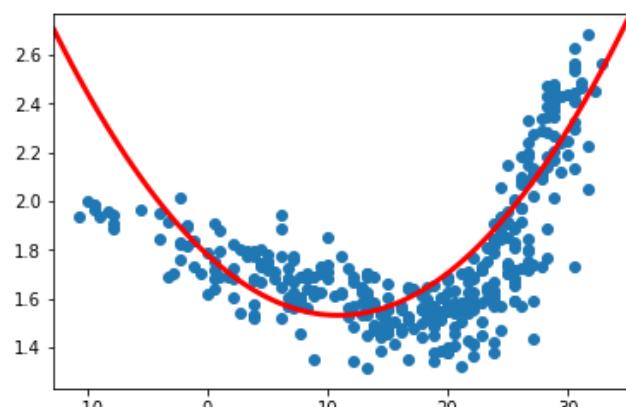
Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

7



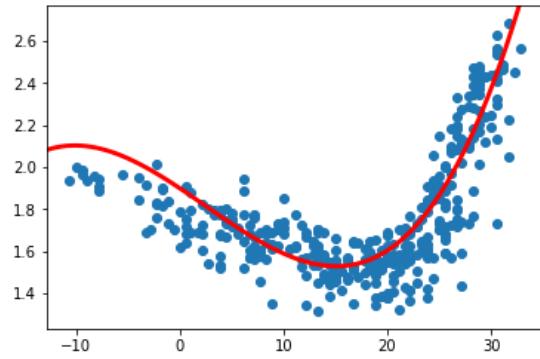
## Esempio: Risultato della Predizione Consumi su tutto l'anno con Regressione Polinomiale di 2° grado

- Il modello di secondo grado corrisponde ad una parabola, che visivamente approssima in modo migliore i dati
- Questo emerge anche misurando l'errore
  - l'errore quadratico medio sul validation set è 0,036
  - l'errore relativo è 8,8%
  - il coefficiente  $R^2$  è 0,566
- Possiamo fare meglio ?

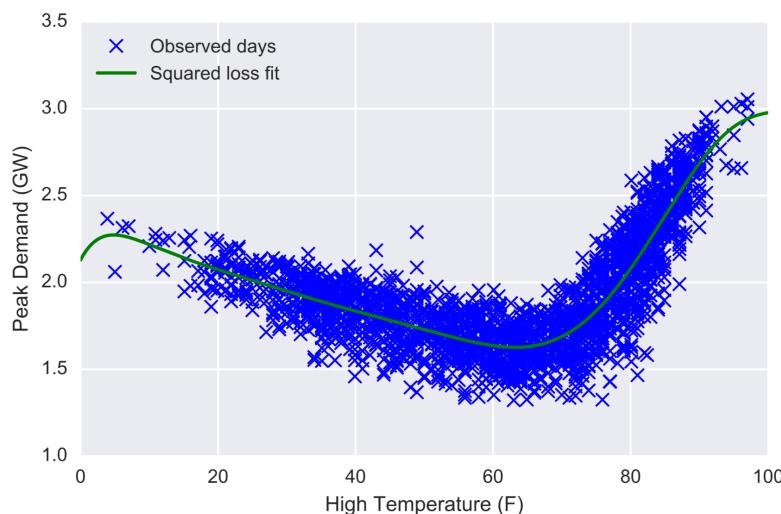


## Esempio: Predizione Consumi su tutto l'anno con Regressione Polinomiale di 3° grado

- Cosa succede con un modello di 3° grado ? → 4 variabili θ
 
$$\hat{y} = \theta_0 + \theta_1 \cdot x + \theta_2 \cdot x^2 + \theta_3 \cdot x^3$$
- La forma del modello è quella di una curva cubica, che visivamente sembra approssimare ancora meglio i dati
- Infatti misurando l'errore...
  - l'errore quadratico medio sul validation set è 0,025
  - l'errore relativo è **6,9%**
  - il coefficiente  $R^2$  è 0,703



## Esempio: Predizione Consumi su tutto l'anno con Regressione Polinomiale di grado 10



- aumentando il grado fino a 10 si ottiene ancora un miglioramento



# scikit-learn: Pre-processing

- scikit-learn offre diverse classi per pre-processare i dati, con la possibilità di cambiare l'insieme delle feature
  - queste offrono una API comune, simile agli algoritmi di learning
- Per usare una classe di pre-processing, questa va dapprima creata indicando eventuali parametri
- Creato l'oggetto che definisce la trasformazione da applicare, ne usiamo i metodi per applicarla su insiemi di dati
  - alla prima applicazione va usato il metodo `fit_transform` per far sì che l'oggetto apprenda la struttura dei dati
  - in genere, come per i modelli, il `fit` è eseguito sui soli dati di training
  - alle successive applicazioni si usa il metodo `transform`, che presume che i dati abbiano la stessa struttura (cioè le stesse variabili)



## scikit-learn: Aggiunta di Feature Polinomiali

- La classe `PolynomialFeatures` aggiunge alle variabili di grado 1 tutte quelle fino al grado  $N$

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)
```

- Ad esempio la trasformata di questa matrice con 2 variabili (colonne) e 2 osservazioni è...

```
poly.fit_transform([[2, 10], [-3, 20]])
array([[ 1.,  2., 10.,  4., 20., 100.],
       [ 1.,  3., 20.,  9., 60., 400.]])
```

- Per escludere il termine di grado 0 (ridondante con l'intercetta) va indicato `include_bias=False` nel costruttore

```
poly = PolynomialFeatures(degree=2, include_bias=False)
```



<i>a</i>	<i>b</i>
2	10
-3	20



1	<i>a</i>	<i>b</i>	<i>a</i> <sup>2</sup>	<i>a</i> · <i>b</i>	<i>b</i> <sup>2</sup>
1	2	10	4	20	100
1	-3	20	9	-60	400



## scikit-learn: Regressione Polinomiale

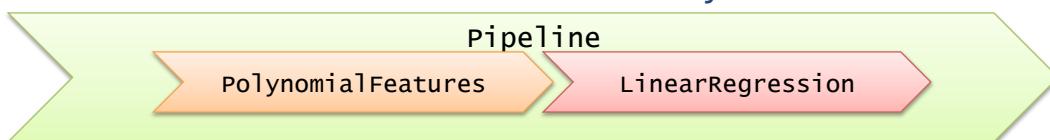
- Per eseguire la regressione polinomiale in scikit-learn possiamo quindi creare un filtro per l'aggiunta delle feature...
- ```
poly = PolynomialFeatures(degree=2, include_bias=False)
```
- ...e usare ancora un modello `LinearRegression`, ma stavolta convertendo i dati in input
- ```
prm = LinearRegression()
prm.fit(poly.fit_transform(x_train), y_train)
```
- Per utilizzare il modello, i dati vanno trasformati utilizzando lo stesso filtro

```
>>> prm.predict(poly.transform(30))
array([2.2923595])
>>> prm.score(poly.transform(x_val), y_val)
```



## scikit-learn: Pipeline

- Nella pratica è spesso necessario addestrare ed utilizzare un modello applicando trasformazioni a tutti dati in input
  - si può avere una sequenza più o meno lunga di trasformazioni
- Una *pipeline* incapsula una o più trasformazioni e un modello, permettendo di interagire con essi come un'entità unica
  - l'API dell'oggetto pipeline è la stessa di un modello “semplice”, con i metodi `fit`, `predict`, `score`, ... che funzionano allo stesso modo
  - ad ogni chiamata di questi metodi, le trasformazioni sono applicate automaticamente ai dati in input prima di essere passate al modello
- Ad es. un modello di regressione polinomiale si può costruire abbinandone a uno lineare un filtro `PolynomialFeatures`



## scikit-learn: Definizione ed Uso di una Pipeline

- Per costruire una pipeline va passata una lista degli elementi che la compongono, con un nome associato a ciascuna

```
from sklearn.pipeline import Pipeline
prm = Pipeline([
    # nome   elemento
    ("poly", PolynomialFeatures(degree=2, include_bias=False)),
    ("linreg", LinearRegression())
])
```

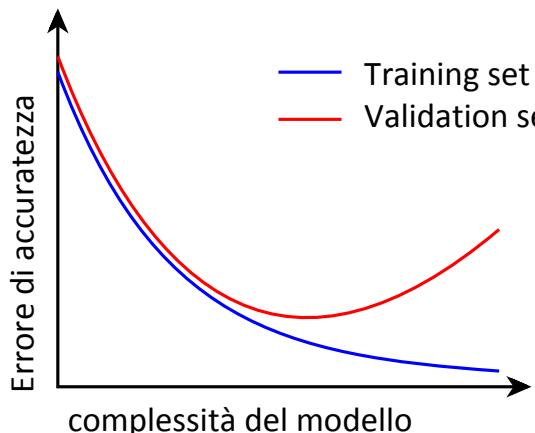
- Una volta creata, la pipeline può essere utilizzata come un modello singolo, con i filtri applicati in automatico

```
>>> prm.fit(X_train, y_train)
>>> prm.predict(30)
>>> prm.score(X_val, y_val)
```

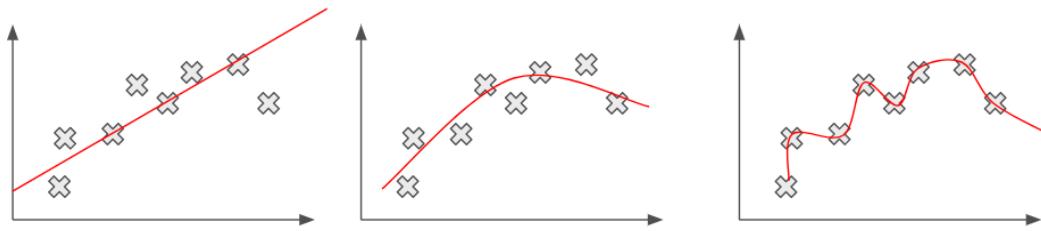


## Complessità del Modello: *Overfitting Vs Generalizzazione*

- i dati devono essere divisi in training set e validation set
- qualsiasi modello di learning si estraе dal training set
- all'aumentare della complessità del modello di learning si riduce l'errore
- ma dopo una certa soglia di complessità, l'errore sul validation set torna a crescere
- questa è la soglia di overfitting che indica che la maggior complessità dei modelli descrive meglio il training set ma non il validation set
  - il modello ha perso di generalità
- scegliamo il modello che minimizzi l'errore sul validation set



# Complessità del Modello: Underfitting ed Overfitting



Underfitting

Optimal

Overfitting

- **Underfitting**

- il modello è troppo semplice e quindi inadeguato a rappresentare i dati, è insoddisfacente sia l'errore sul training, sia sul validation

- **Overfitting**

- il modello è troppo complesso, l'errore sul training è significativamente inferiore a quello sul validation, il modello non generalizza dal training e non rappresenta adeguatamente dati ignoti

- **Modello ottimale: minimizza l'errore sul validation set**

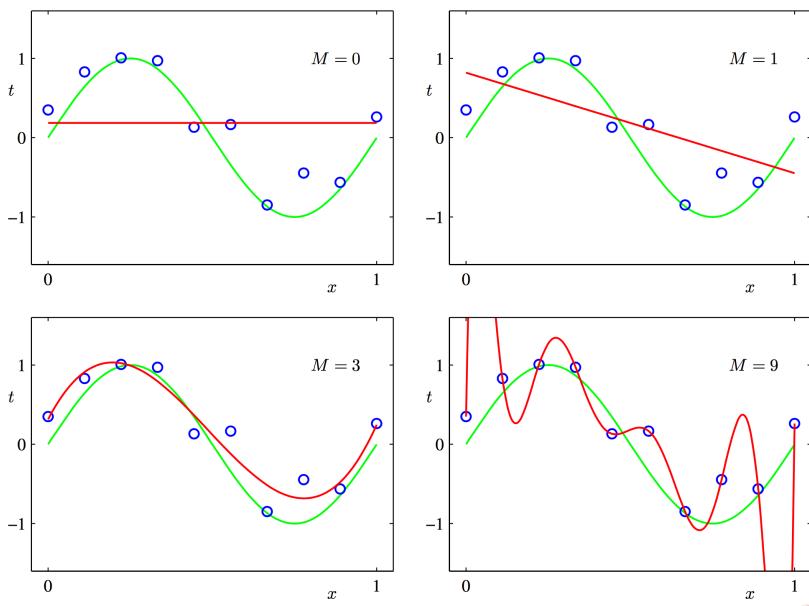
Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

17



# Complessità del Modello: Underfitting ed Overfitting in base al Grado del Polinomio

- in blu i dati
- in verde la regressione ideale
- in rosso la regressione all'aumentare del grado M
- Underfitting con  $M < 3$
- Overfitting con  $M \geq 9$

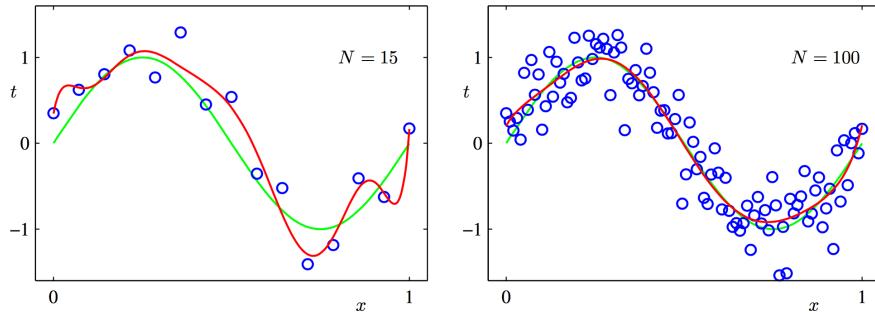


Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

18



# Complessità del Modello e Quantità dei Dati



- Nella figura di sinistra, con polinomio di grado 9 e N=15 dati, il modello (curva rossa) è in overfitting
  - differenza significativa con la curva verde ottimale
- Con N=100 e stesso grado si ottiene un modello ottimale
  - quantità di dati e complessità del modello sono interdipendenti
  - con pochi dati la regressione può migliorare aggiungendo dati casuali generati tenendo conto della distribuzione di quelli reali

Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

19



## Training, Validation Set e Parametri e Iperparametri

Training set (e.g. 70%)	Holdout / validation set (e.g. 30%)
-------------------------	-------------------------------------

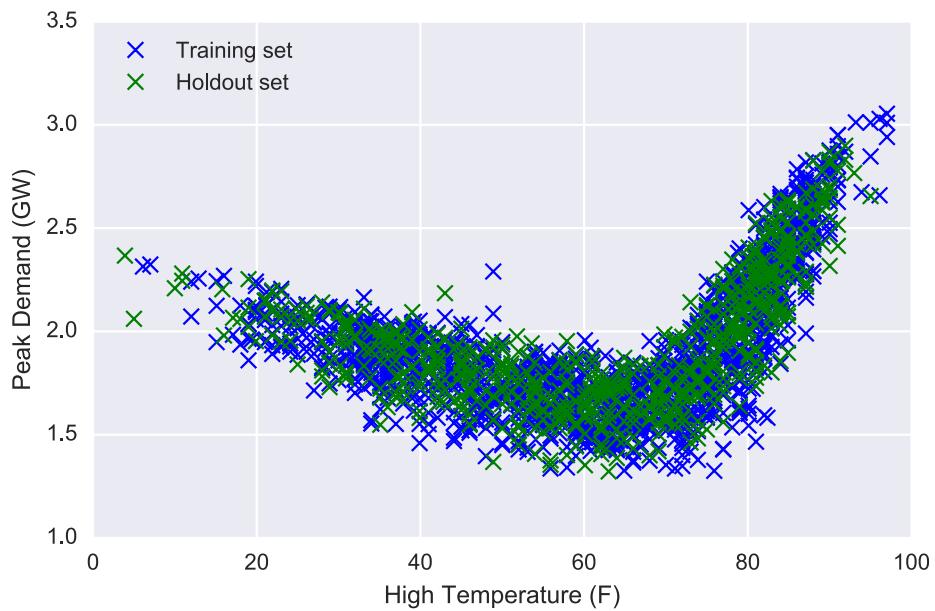
- suddividiamo i dati in training e validation set (holdout)
  - di norma la suddivisione è casuale; esistono metodi più complessi
- suddivisioni classiche 70-30, 2/3-1/3, ma anche 50-50
- Dal training set si determinano i parametri  $\theta$  del modello di learning, e.g. i coefficienti  $\alpha$  e  $\beta$  nella regressione
- Gli iperparametri sono tutte le altre scelte e si determinano dal validation set:
  - Il tipo di funzione di regressione, lineare o polinomiale, il grado, la normalizzazione, regolarizzazione etc.

Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

20



# Esempio di Holdout: Training e Validation



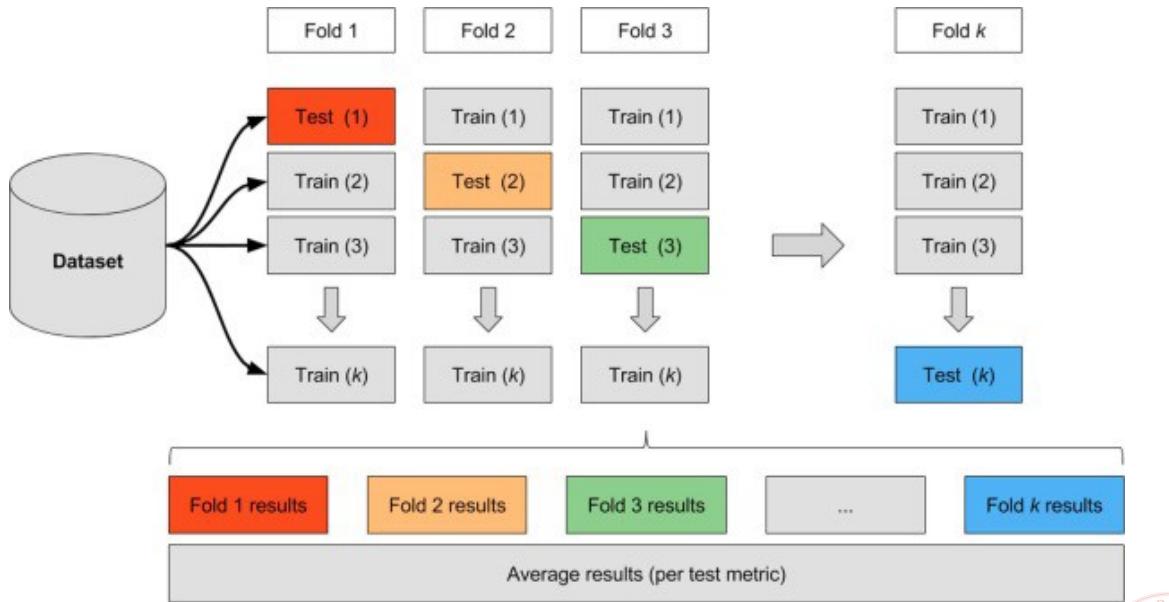
## K-Fold Cross Validation (i)



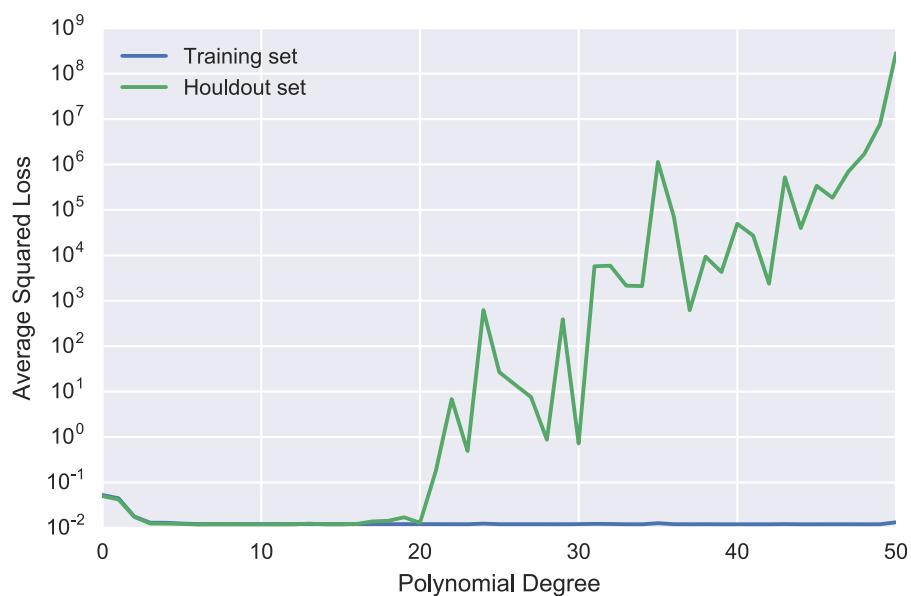
- si suddividono i dati in  $k$  sottoinsieme disgiunti
- un sottoinsieme è usato come validation set e gli altri  $k-1$  come training set
- si ripete il procedimento  $k$  volte con ciascuno dei  $k$  subset
- si ottengono  $k$  modelli di learning, e.g.  $k$  regressioni
- l'accuratezza è la media delle accuratezze dei  $k$  modelli
- k-fold cross validation stratificata
  - stessa distribuzione e caratteristiche dei dati in ogni fold



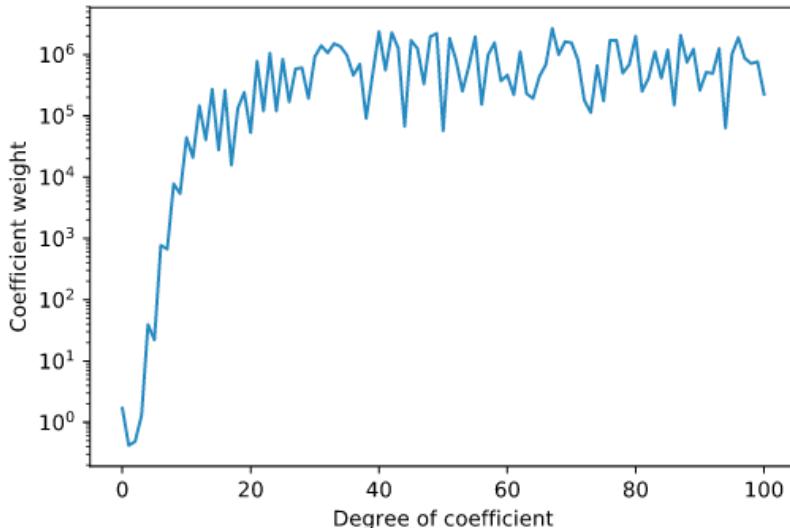
## K-Fold Cross Validation (ii)



## Regressione Polinomiale: l'Errore Aumenta al variare del grado (con Holdout), perché ?



## Regressione Polinomiale: Aumento dei coefficienti $\theta$ all'aumentare del grado



- esempio sulla previsione del consumo di elettricità
- aumentando il grado del polinomio, aumenta rapidamente il valore di  $\theta$



## Regolarizzazione: *Ridge Regression*

- Il grado nella regressione polinomiale misura la complessità del modello di learning
  - con grado teoricamente infinito possiamo modellare qualsiasi data set
- ma i coefficienti del polinomio, e.g. grado 50, diventano grandi

$$\theta = -3.88 \times 10^6, 7.60 \times 10^6, 3.94 \times 10^6, -2.60 \times 10^7, \dots$$

ciò provoca forti oscillazioni nella regressione peggiorandone l'accuratezza

- regolarizzare significa ridurre il valore dei coefficienti, come ?
- si aggiunge alla funzione d'errore da minimizzare anche  $\lambda \cdot \theta$ 
  - $0 \leq \lambda < \infty$  (iperparametro), con  $\lambda = 0$  nessuna regolarizzazione

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^m (\theta^T x^{(i)} - y^{(i)})^2 + \lambda \|\theta\|_2^2 \quad \|\theta\|_2^2 = \sum_{i=0}^{n-1} \theta_i^2 \quad n = \text{num. parametri } \theta$$



# Minimizzazione dell'errore con Regolarizzazione

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^m (\theta^T x^{(i)} - y^{(i)})^2 + \lambda \|\theta\|_2^2$$

- per minimizzare come al solito calcoliamo la derivata prima

$$\nabla_{\theta} \left( \sum_{i=1}^m (\theta^T x^{(i)} - y^{(i)})^2 + \lambda \|\theta\|_2^2 \right) = 2X^T(X\theta - y) + 2\lambda\theta$$

- e troviamo il gradiente ponendo la derivata a zero

$$2X^T X\theta + 2\lambda\theta = 2X^T y \implies \theta = (X^T X + \lambda I)^{-1} X^T y$$

- la soluzione ha un termine nuovo  $\lambda I$  dove  $I$  è la matrice identità
- perciò più  $\lambda$  è grande, più si riducono i coefficienti  $\theta$

- nota importante sul termine  $\|\theta\|_2^2 = \sum_{i=0}^{n-1} \theta_i^2$

- la scala delle variabili di input si riflette sui parametri  $\theta$  e poiché ora si sommano, incidentalmente predominerebbero quelli con scala maggiore → **standardizzare le variabili di input**

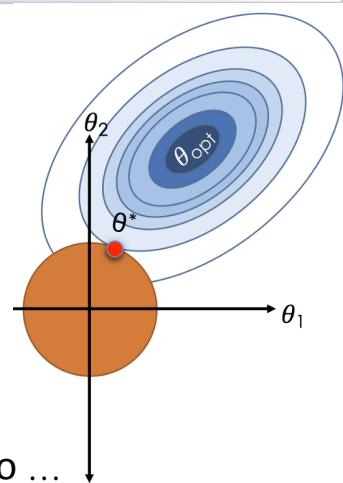
Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

27



## Regolarizzazione: Interpretazione Geometrica

- Senza perdita di generalità consideriamo una regressione con parametri  $\theta$
- $\hat{y} = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2$
- $\theta_{\text{opt}}$  è il punto che minimizza la funzione di errore ( $\theta_0$  è costante con dati standardizzati)
  - in blue le curve di livello della funzione d'errore
- minimizzare la funzione d'errore aggiungendo  $\lambda \|\theta\|_2^2$  corrisponde a trovare una soluzione  $\theta^*$  (punto rosso) di minimo vincolato ...
- ... all'interno della relativa disequazione  $\theta_1^2 + \theta_2^2 \leq r^2$  i.e. ipersfera centrata in 0,0 con raggio  $r$  che dipende da  $\lambda$ 
  - all'aumentare di  $\lambda$ , si riduce  $r$  e la soluzione tende ad avvicinarsi all'origine con  $\theta$  minori

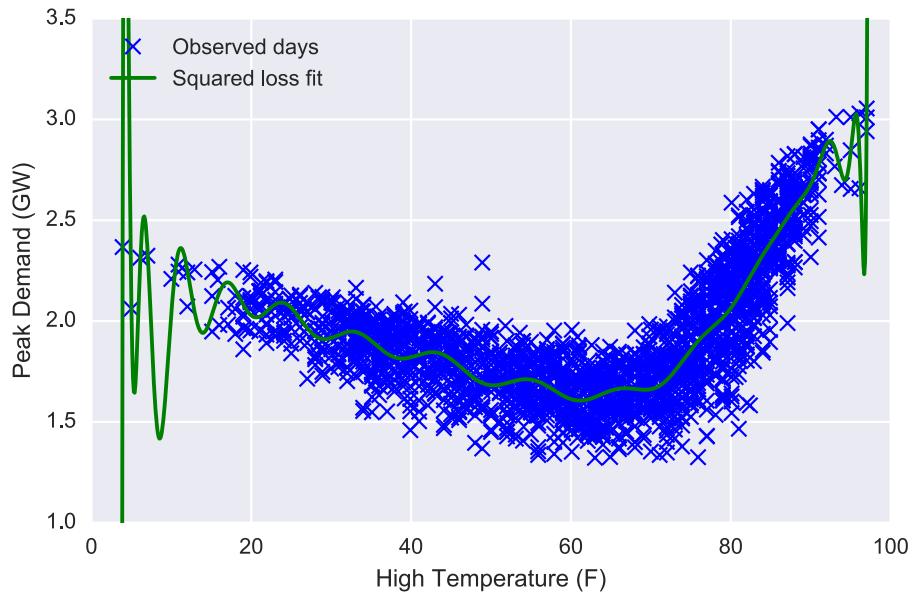


Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

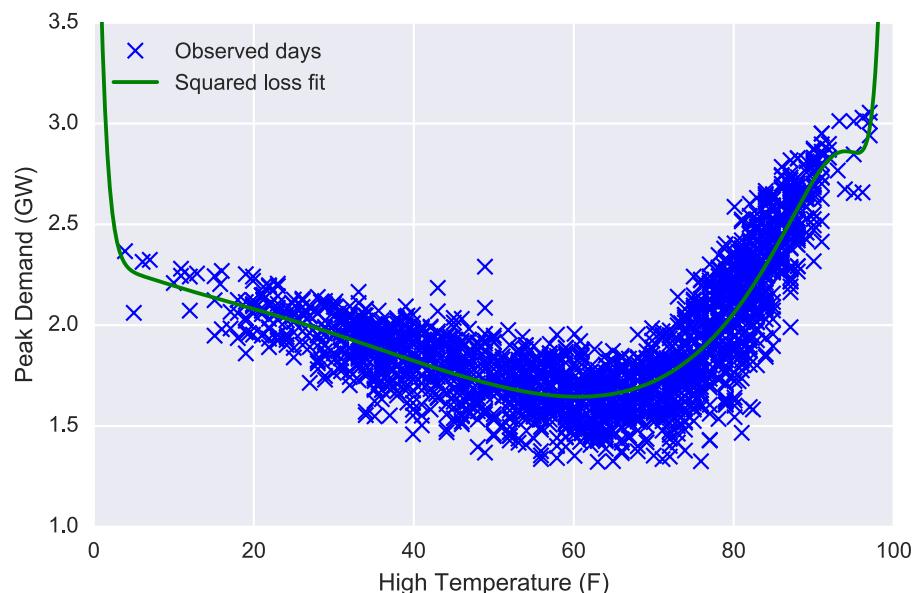
28



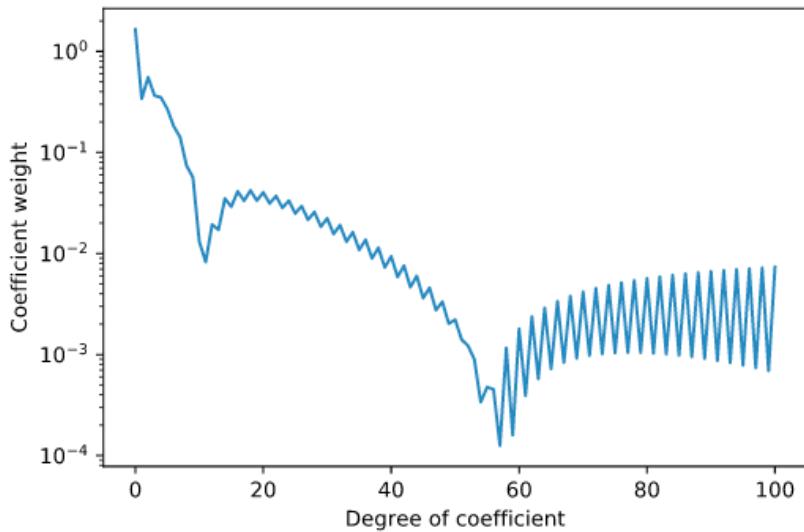
## Regressione Polinomiale: grado 50 senza regolarizzazione



## Regressione Polinomiale: grado 50 con regolarizzazione $\lambda = 1$



## Regressione Polinomiale: Effetto della Regolarizzazione al variare del grado



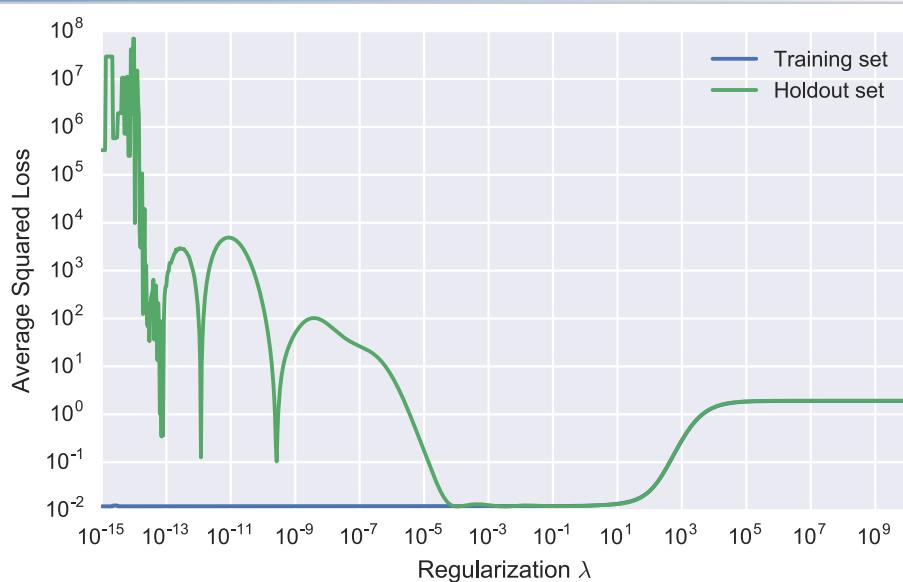
- esempio sulla previsione del consumo di elettricità
- Pur aumentando il grado, la regolarizzazione riduce molto il valore di  $\theta$ 
  - da  $10^6$  a  $10^{-3}$

Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

32



## Regressione Polinomiale: Accuratezza con grado 50 al variare di $\lambda$



la regolarizzazione è un altro iperparametro da ottimizzare rispetto al validation set; lo si trova con cross validation

Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

33



# Individuazione del Migliore $\lambda$

- Il metodo più semplice per determinarlo è usare la k-fold cross validation: in python l'iperparametro  $\lambda$  è chiamato  $\alpha$

```
from sklearn.linear_model import RidgeCV
...
# 25 ascisse  $\lambda$  del grafico precedente da  $10^{-15}$  a  $10^9$ 
alphas = 10**np.linspace(-15, 9, 25)
reg_cv = RidgeCV(alphas, cv=5) # 5-cross fold
reg_cv.fit(poly.fit_transform(X_train), y_train)
reg_cv.score(poly.transform(X_val), y_val)
>>> 0.01102
reg_cv.alpha_ # alpha migliore
>>> 0.1
```

- determinare gli iperparametri migliori con gli stessi dati usati per estrarre il modello migliore genera score ottimisti, usare **Nested Cross Validation** in questo caso per stimare correttamente anche lo score su dati ignoti

Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

34

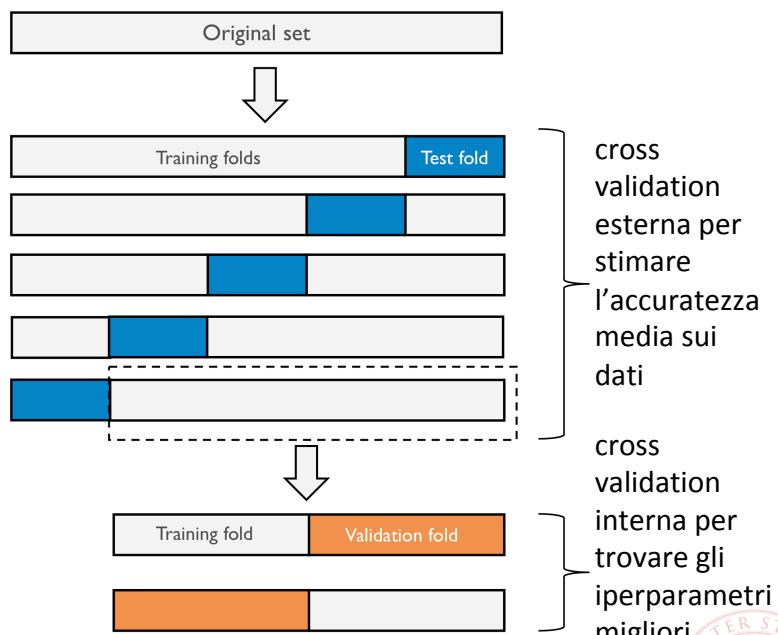


## Nested Cross Validation (i)

Ogni parte di training (grigia) della k-fold cross validation esterna è suddivisa nella cross validation interna in m-subfold che sono usati per individuare gli iperparametri migliori

Gli iperparametri migliori sono poi usati per addestrare e **testare** il modello nella relativa parte della validation esterna

Non è un metodo per estrarre un modello predittivo migliore di un altro, ma per **stimare l'accuratezza a regime** sui dati

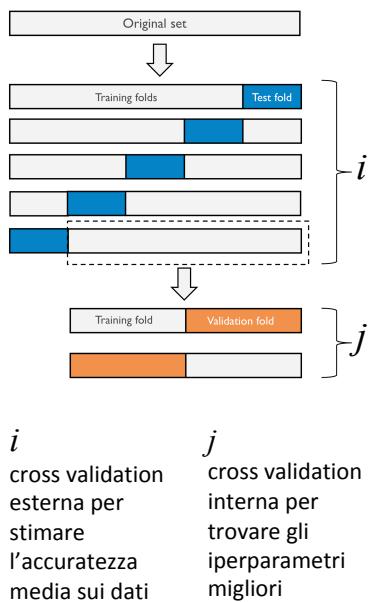


Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

35



# Nested Cross Validation (ii)



## Pseudocodifica del Funzionamento

```

Require:  $K_1, K_2$ , where  $K_1$  is number of outer folds and  $K_2$  inner folds
Require:  $\mathcal{D}$ , dataset containing input features  $X$  and output feature  $y$ 
Require:  $P_{sets}$ , set of hyperparameters with different values
Require:  $\mathcal{M}$ , a single estimator, model.

for  $i = 1$  to  $K_1$  splits do
    Split  $\mathcal{D}$  into  $\mathcal{D}_i^{train}, \mathcal{D}_i^{test}$  for the  $i$ 'th split
    for  $j = 1$  to  $K_2$  splits do
        Split  $\mathcal{D}_i^{train}$  into  $\mathcal{D}_j^{train}, \mathcal{D}_j^{test}$  for the  $j$ 'th split
        foreach  $p$  in  $RandomSample(P_{sets})$  do
            Train  $\mathcal{M}$  on  $\mathcal{D}_j^{train}$  with hyperparameter set  $p$ 
            Compute test error  $E_j^{test}$  for  $\mathcal{M}$  with  $\mathcal{D}_j^{test}$ 
    Select optimal hyperparameter set  $p^*$  from  $P_{sets}$ , where  $E_j^{test}$  is best
    Train  $\mathcal{M}$  with  $\mathcal{D}_i^{train}$ , using  $p^*$ 
    Compute test error  $E_i^{test}$  for  $\mathcal{M}$  with  $\mathcal{D}_i^{test}$ 

```

Al termine il modello migliore si ottiene addestrando su tutti i dati utilizzando per ogni iperparametro il valore che ha dato il risultato medio migliore, e.g.  $\lambda = 0.1$ , grado polinomio = 10



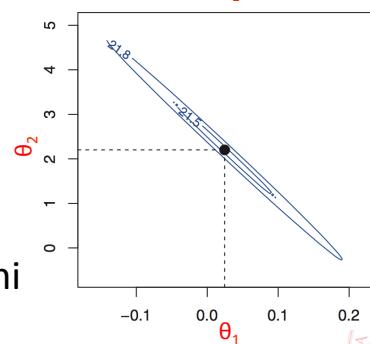
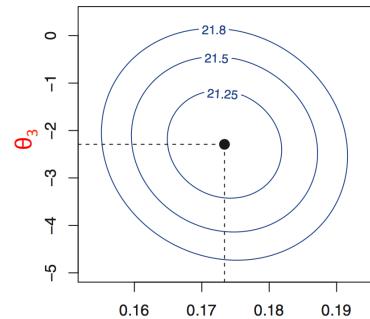
# Collinearità: Dipendenze tra Variabili di Input (i)

- Nella regressione ordinaria si assume che le variabili di input siano indipendenti tra loro
- Con dipendenze la regressione è instabile, i.e. piccole variazioni nei dati generano modelli molto diversi e inaffidabili
  - un'azienda scopre dai dati di campagne pubblicitarie su giornali, radio e TV, che l'aumento maggiore delle vendite si ottiene con radio e TV  
 $Vendite = \theta_0 + \theta_1 \cdot TV + \theta_2 \cdot Radio + \theta_3 \cdot Giornali$  i.e.  $\theta_1 > \theta_2 > \theta_3$
  - scopre anche che maggiore è la pubblicità in radio, maggiore è il ritorno della pubblicità in TV → c'è qualche dipendenza tra TV e radio
  - quindi investendo 100 solo in TV o solo in radio, si ottiene un aumento delle vendite inferiore rispetto ad investire 50 in TV e 50 in radio
  - Effetto noto nel marketing come *sinergia*, nel Statistical Learning come *interazione*: una soluzione è aggiungere un nuovo parametro che combina le due variabili →  $\theta_4 \cdot TV \cdot radio$



# Collinearità: Dipendenze tra Variabili di Input (ii)

- Nella fig. in alto l'errore di regressione minimo usando solo  $\theta_1, \theta_3$  è ben definito (e.g.  $\theta_{\text{radio}}, \theta_{\text{giornali}}$ )
- sotto invece con  $\theta_1, \theta_2$  un ampio num. di loro valori produce lo stesso minimo → ampio n. di soluzioni (e.g.  $\theta_{\text{radio}}, \theta_{\text{TV}}$ )
  - training set con differenze trascurabili generano modelli di regressione con coefficienti molto diversi -> soluzione instabile causata da dipendenza tra  $\theta_1$  e  $\theta_2$
  - accuratezza imprevedibile su dati ignoti
- la regolarizzazione risolve il problema poiché vincola/riduce le possibili soluzioni
  - soluzioni vincolate intorno all'origine



Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

39

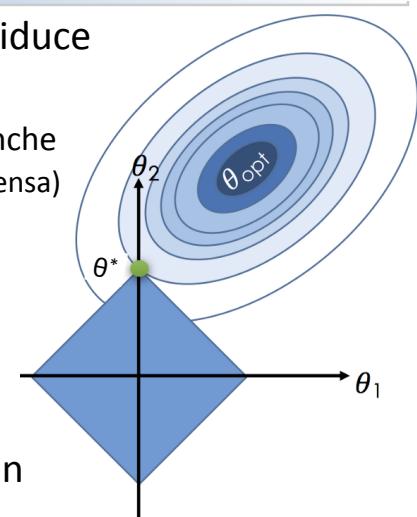


# Regressione LASSO

- La regressione Ridge, all'aumentare di  $\lambda$  riduce ma non azzerà il valore dei parametri  $\theta$ 
  - perciò la soluzione utilizza tutte le variabili, anche quelle irrilevanti per la predizione (soluzione densa)
- Se penalizziamo i  $\theta$  nella minimizzazione dell'errore con norma L1 invece che L2

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^m (\theta^T x^{(i)} - y^{(i)})^2 + \lambda \|\theta\|_1 : \|\theta\|_1 = \sum_{j=1}^n |\theta_j|$$

- la soluzione  $\theta^*$  è vincolata all'interno di un ipercubo centrato sull'origine
  - maggior è  $\lambda$** , più è probabile che  $\theta^*$  cada sugli spigoli azzerando diversi  $\theta$ , più **variabili irrilevanti si eliminano**, **soluzione sparsa**
  - modello predittivo più interpretabile; tecnica per selezionare variabili

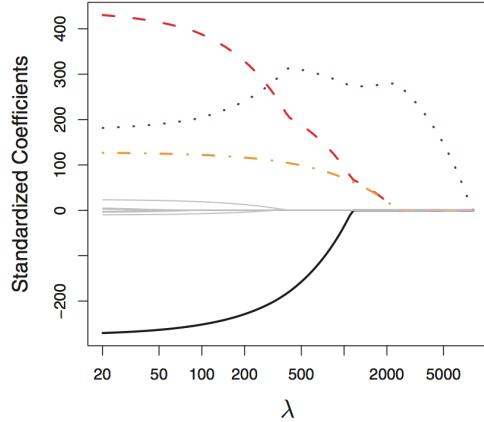
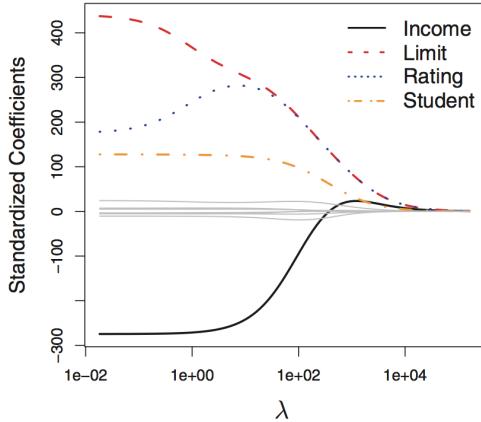


Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

40



# Esempio Grafico di Selezione di Variabili con Lasso



- I grafici mostrano la riduzione dei valori dei parametri di regressione **Ridge (sx)** e **LASSO (dx)** all'aumentare di lambda
  - Previsione dell'insolvenza con carte di credito
  - con Ridge tutte le variabili, comprese quelle irrilevanti in grigio chiaro si riducono ma non si azzerano, con LASSO le irrilevanti si azzerano

Applicazioni Data Intensive - G. Moro, R. Pasolini - DISI, Università di Bologna, Cesena

41

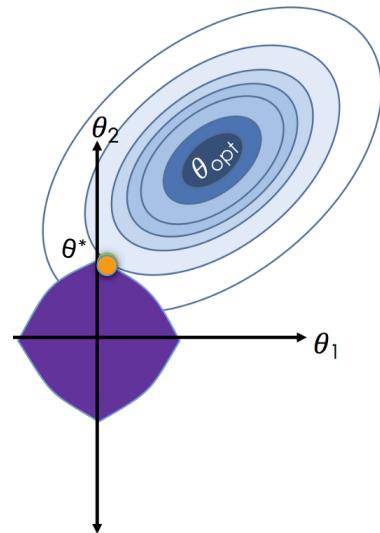


## Regressione Elastic Net: L1+L2

- Generalizza Ridge e LASSO con entrambe le penalizzazioni dei  $\theta$  con norma L1 e L2

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^m (\theta^T x^{(i)} - y^{(i)})^2 + \lambda (\alpha \|\theta\|_1 + (1-\alpha) \|\theta\|_2^2)$$

- introduce un secondo iperparametro  $\alpha$  per pesare le penalizzazioni L1 e L2
  - con  $\alpha = 0$  è la regressione Ridge, con  $\alpha = 1$  è la regressione LASSO; valori intermedi combinano le due penalizzazioni ed i loro pro e contro
  - in python l'iperparametro  $\alpha$  è **`l1_ratio`** e **`lambda`** è  $\lambda$



# Regressione Polinomiale: Quanti parametri $\theta$ con **10** variabili indipendenti e grado **2** ?

- Il numero di variabili indipendenti sono quelle del data set
  - e.g. nella previsione del prezzo delle case le variabili sono una decina: *num. stanze, diversi quartieri (var binarie), metri quadri, num. piano ...*
  - $$h_{\theta}(x_1, \dots, x_{10})^2 = \theta_1 + x_1 \theta_2 + x_2 \theta_3 + x_3 \theta_4 + x_4 \theta_5 + x_5 \theta_6 + x_6 \theta_7 + x_7 \theta_8 + x_8 \theta_9 + x_9 \theta_{10} + x_{10} \theta_{11} + x_1^2 \theta_{12} + x_1 x_2 \theta_{13} + x_1 x_3 \theta_{14} + x_1 x_4 \theta_{15} + x_1 x_5 \theta_{16} + x_1 x_6 \theta_{17} + x_1 x_7 \theta_{18} + x_1 x_8 \theta_{19} + x_1 x_9 \theta_{20} + x_1 x_{10} \theta_{21} + x_2^2 \theta_{22} + x_2 x_3 \theta_{23} + x_2 x_4 \theta_{24} + x_2 x_5 \theta_{25} + x_2 x_6 \theta_{26} + x_2 x_7 \theta_{27} + x_2 x_8 \theta_{28} + x_2 x_9 \theta_{29} + x_10 x_2 \theta_{30} + x_3^2 \theta_{31} + x_3 x_4 \theta_{32} + x_3 x_5 \theta_{33} + x_3 x_6 \theta_{34} + x_3 x_7 \theta_{35} + x_3 x_8 \theta_{36} + x_3 x_9 \theta_{37} + x_10 x_3 \theta_{38} + x_4^2 \theta_{39} + x_4 x_5 \theta_{40} + x_4 x_6 \theta_{41} + x_4 x_7 \theta_{42} + x_4 x_8 \theta_{43} + x_4 x_9 \theta_{44} + x_{10} x_4 \theta_{45} + x_5^2 \theta_{46} + x_5 x_6 \theta_{47} + x_5 x_7 \theta_{48} + x_5 x_8 \theta_{49} + x_5 x_9 \theta_{50} + x_{10} x_5 \theta_{51} + x_6^2 \theta_{52} + x_6 x_7 \theta_{53} + x_6 x_8 \theta_{54} + x_6 x_9 \theta_{55} + x_{10} x_6 \theta_{56} + x_7^2 \theta_{57} + x_7 x_8 \theta_{58} + x_7 x_9 \theta_{59} + x_{10} x_7 \theta_{60} + x_8^2 \theta_{61} + x_8 x_9 \theta_{62} + x_{10} x_8 \theta_{63} + x_9^2 \theta_{64} + x_{10} x_9 \theta_{65} + x_{10} x_2 \theta_{66}$$
  - 66 termini** le dimensioni del data set aumentano di quasi 7 volte !!
  - Problemi ad elevata dimensionalità:** se il numero di variabili è maggiore, o dello stesso ordine di grandezza, rispetto al numero delle istanze



# Regressione Polinomiale: Quanti parametri $\theta$ con **n** variabili e grado **g**?

- $h_{\theta}(x_1, \dots, x_n)^g = \dots ??$
- il polinomio di grado 2 in  $n$  variabili genera  $\frac{(n+1)(n+2)}{2} = \binom{n+2}{2}$  termini
- il polinomio di grado  $g$  in  $n$  variabili genera  $\binom{n+g}{g}$  termini
  - e.g. previsione del consumo energetico dalla **temperatura e dal tipo del giorno** della settimana:
    - con 2 variabili e grado 10 otteniamo l'accuratezza migliore → 66 variabili, ma la quantità di dati aumenta però di 33 volte
  - e.g. **10 variabili e grado 10** generano **184756** parametri prima ancora di effettuare la regressione → **approccio non scalabile**
    - Regola generale: all'aumentare di n diminuire g e viceversa**
    - prima di procedere calcolare il numero di variabili derivate da  $n$  e  $g$
  - E' possibile mappare i dati originali in un nuovo spazio ad elevata dimensionalità senza creare le relative nuove variabili ? Fantascienza ??**



# Soluzione all'Esplosione della Dimensionalità (i)

*La Scienza a volte Supera la Fantascienza*

- sembra assurdo, ma possiamo portare i dati in nuovi spazi ad elevata dimensionalità senza creare nuove variabili, **come?**
- e.g. con questo polinomio  $(1+x_1z_1+x_2z_2)^2$ 
  - dove  $\mathbf{x} = (x_1, x_2)$  e  $\mathbf{z} = (z_1, z_2)$  2 dati di input con le loro 2 dimensioni

$$(1 + x_1z_1 + x_2z_2)^2 = x_1^2z_1^2 + 2x_1x_2z_1z_2 + x_2^2z_2^2 + 2x_1z_1 + 2x_2z_2 + 1$$

- sviluppando abbiamo 6 termini che corrispondono anche al prodotto scalare dei due vettori seguenti

$$(x_1^2, x_1x_2\sqrt{2}, x_2^2, x_1\sqrt{2}, x_2\sqrt{2}, 1) \bullet (z_1^2, z_1z_2\sqrt{2}, z_2^2, z_1\sqrt{2}, z_2\sqrt{2}, 1)$$

- ciò corrisponde al mapping in 6 dimensioni dei 2 vettori  $\mathbf{x}$  e  $\mathbf{z}$  in 2D

$$(x_1, x_2) \xrightarrow{\phi} (x_1^2, x_1x_2\sqrt{2}, x_2^2, x_1\sqrt{2}, x_2\sqrt{2}, 1) \quad (z_1, z_2) \xrightarrow{\phi} (z_1^2, z_1z_2\sqrt{2}, z_2^2, z_1\sqrt{2}, z_2\sqrt{2}, 1)$$

- costi: triplicate le dimensioni iniziali dei dati ... ANDIAMO AVANTI



# Soluzione all'Esplosione della Dimensionalità (ii)

## Funzioni Kernel

- sviluppiamo interamente il polinomio di prima

$$\begin{aligned} (1 + x_1z_1 + x_2z_2)^2 &= x_1^2z_1^2 + 2x_1x_2z_1z_2 + x_2^2z_2^2 + 2x_1z_1 + 2x_2z_2 + 1 \\ &= (x_1z_1 + x_2z_2)^2 + 2(x_1z_1 + x_2z_2) + 1 \\ &= (x^T z)^2 + 2(x^T z) + 1 \\ &= (x^T z + 1)^2 \end{aligned}$$

- cosa c'è di interessante nel risultato finale ?

- il quadrato del prodotto scalare dei vettori iniziali  $x$  ed  $y$  in 2 dimensioni + 1 è uguale al prodotto scalare dei vettori trasformati in 6 dimensioni  
 $((x_1, x_2) \bullet (z_1, z_2) + 1)^2 = (x_1^2, x_1x_2\sqrt{2}, x_2^2, x_1\sqrt{2}, x_2\sqrt{2}, 1) \bullet (z_1^2, z_1z_2\sqrt{2}, z_2^2, z_1\sqrt{2}, z_2\sqrt{2}, 1)$
- ciò dimostra che  $((x_1, x_2) \bullet (z_1, z_2) + 1)^2$  equivale a lavorare in uno spazio a 6 dimensioni senza creare nuove variabili dallo spazio 2D iniziale
  - 2 moltiplicazioni producono lo stesso effetto di 6 moltiplicazioni
- il metodo, **KERNEL TRICK**, vale per ogni grado  $g$  e num.  $n$  di dimensioni
  - $\mathbf{x} = (x_1, \dots, x_n)$   $\mathbf{z} = (z_1, \dots, z_n)$   $Kernel(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^g$



## Soluzione all'Esplosione della Dimensionalità (iii)

### Costi Computazionali con e senza Kernel

- costi senza e con  $Kernel(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^g$  polinomiale
  - senza kernel, con grado  $g$  ed  $n$  dimensioni il costo è  $O(r^2 n^g / 2g!)$ 
    - e.g.  $n = 100, g = 4, 1000$  istanze  $\rightarrow 1000^2/2 \times 4421275 \approx 2.21 \times 10^{12}$
  - con kernel il costo è costante all'aumentare del grado  $g \rightarrow O(r^2 n^2 / 2)$ 
    - come sopra  $n = 100, g = 4, 1000$  istanze  $\rightarrow 1000^2/2 \approx 5 \times 10^5$
- e.g. num. di termini polinomiali con alcune dimensioni e gradi

Grado del polinomio di trasformazione	$\phi(\mathbf{x}) \cdot \phi(\mathbf{z})$	$\phi(\mathbf{x})$ : num. dimensioni con $n$ dimensioni iniziali	E.g. con 100 dimensioni ed $r^2/2$	Con Kernel Trick	E.g. con 100 dimensioni
2	$(\mathbf{x} \cdot \mathbf{z})^2$	$n(n+1)/2$	$2525 r^2$	$n r^2/2$	$50 r^2$
3	$(\mathbf{x} \cdot \mathbf{z})^3$	$n(n+1)(n+2)/6$	$85850 r^2$	$n r^2/2$	$50 r^2$
4	$(\mathbf{x} \cdot \mathbf{z})^4$	$n(n+1)(n+2)(n+3)/24$	$2.21 \times 10^6 r^2$	$n r^2/2$	$50 r^2$



## Altre Funzioni Kernel

- Kernel (non lineari) possono modellare dati complessi con distribuzioni non lineari
  - senza aumentare le dimensioni dei dati e dei relativi costi impossibili da sostenere per operare nello spazio ad elevata dimensionalità
- Limite: possono generare modelli affetti da overfitting

Proprietà delle Funzioni Kernel       $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})$

Polinomiale:                           $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + \theta)^g$

Gaussian Radial Basis:             $K(\mathbf{x}, \mathbf{y}) = e^{(-\gamma \|\mathbf{x} - \mathbf{y}\|^2)}$  con  $\gamma = \frac{1}{2\sigma^2}$

Sigmoidale:                           $K(\mathbf{x}, \mathbf{y}) = \tanh(k \mathbf{x} \cdot \mathbf{y} - \delta)$

dove  $0 \leq \theta \in R, g \in N, k, \delta \in R$



# la Magia delle Funzioni Kernel

- Il caso della previsione del consumo di elettricità dimostra che
  - la regressione lineare è inadeguata e che un polinomio di grado 10 o 50 ottiene risultati molto più accurati
  - ma un polinomio di **grado 10 con 2 variabili** (temperatura e giorno della settimana) genera 66 variabili → **33 volte i dati e costi della reg. lineare**
  - con grado 50** l'accuratezza migliora ancora ma servono **1326 variabili** e la quantità di dati aumenta di oltre **600 volte**
  - col **kernel polinomiale di grado 50** ogni dato ha **1326 dimensioni, senza creare nemmeno una** delle 1326 variabili
- Come sfruttiamo il kernel nella regressione ?

Esempio di regressione quadratica in una variabile

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 = (\theta_0, \theta_1, \theta_2) \cdot (1, x, x^2)$$

$$= \boldsymbol{\theta}^T \phi(x) \quad \text{dove } \phi(x) \text{ è la trasformazione di } x$$

Gianluca Moro - DISI, Università di Bologna

50



## Come Incorporare le Funzioni Kernel nella Regressione ?

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 = (\theta_0, \theta_1, \theta_2) \cdot (1, x, x^2) \quad \text{es. di Regressione quadratica in una variabile}$$

$$= \boldsymbol{\theta}^T \phi(x) \quad \text{dove } \phi(x) \text{ è la trasformazione di } x, \quad \text{ma } \boldsymbol{\theta}^T = \sum_{i=1}^m \alpha_i \phi(x^{(i)})$$

i coefficienti ottimali  $\theta$  sono combinazioni lineari dei dati

$$= \sum_{i=1}^m \alpha_i \phi(x) \phi(x^{(i)}) \quad \phi(x^{(i)}) \text{ è la trasformazione di ogni istanza } x^{(i)} \text{ diversa da } x$$

$$= \sum_{i=1}^m \alpha_i K(x, x^{(i)}) \quad K \text{ è il kernel che sostituisce le due trasformazioni } \phi(x) \phi(x^{(i)})$$

- perchè c'è la sommatoria benché ci sia 1 sola variabile  $x$  indipendente ?
  - la funzione kernel polinomiale moltiplica due dati di input  $x$  e  $z$
- $K(x, z) = (1 + x^T z)^d$
- la sommatoria qui introduce il prodotto tra il dato di input  $x$  e tutti gli altri  $m$  dati  $x^{(i)}$   $i=1,..,m$  del data set e per ciascuno c'è un proprio parametro  $\alpha_i$  da apprendere
- **complessità di training circa quadratica rispetto al num. istanze**

Gianluca Moro - DISI, Università di Bologna

51



# Minimizzare la Funzione di Errore con Kernel

- Ricordiamo la funzione di errore da minimizzare  $\underset{\theta}{\text{minimize}} \sum_{i=1}^m (\theta^T x^{(i)} - y^{(i)})^2 + \lambda \|\theta\|_2^2$

- Come diventa con Kernel ? Iniziamo dal quadrato della norma di  $\theta$

– il vettore  $\theta^*$  dei parametri ottimali è  $\theta^* = \sum_{i=1}^m \alpha_i \phi(x^{(i)})$   
perciò la sua norma al quadrato è

$$\|\theta\|_2^2 = \theta^T \theta = \left( \sum_{i=1}^m \alpha_i \phi(x^{(i)})^T \right) \left( \sum_{j=1}^m \alpha_j \phi(x^{(j)}) \right) = \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j K(x^{(i)}, x^{(j)})$$

$$= \alpha^T K \alpha$$

– dove  $K \in \mathbb{R}^{m \times m}$  e la matrice quadrata  $K_{ij} = K(x^{(i)}, x^{(j)})$  con i valori kernel di tutte le coppie dei dati

$$\underset{\alpha}{\text{minimize}} \quad \frac{1}{m} \sum_{i=1}^m \left( \sum_{j=1}^m \alpha_j K(x^{(j)}, x^{(i)}) - y^{(i)} \right)^2 + \lambda \alpha^T K \alpha$$

$$\underset{\alpha}{\text{minimize}} \quad \frac{1}{m} \|K\alpha - y\|_2^2 + \lambda \alpha^T K \alpha$$

l'i-esimo elemento del vettore  $K\alpha - y$  è il quadrato della sommatoria interna; il risultato è la sommatoria di quadrati di un vettore, i.e. una norma al quadrato

Gianluca Moro - DISI, Università di Bologna

52



# Gradiente della Regressione con Kernel

- derivata della funzione d'errore rispetto ai parametri  $\alpha$

$$\nabla_{\alpha} \left( \frac{1}{m} \|K\alpha - y\|_2^2 + \lambda \alpha^T K \alpha \right) = \frac{2}{m} K(K\alpha - y) + 2\lambda K\alpha$$

- poniamola a zero e otteniamo il vettore gradiente  $\alpha$

$$2KK\alpha + 2\lambda m K\alpha = 2Ky$$

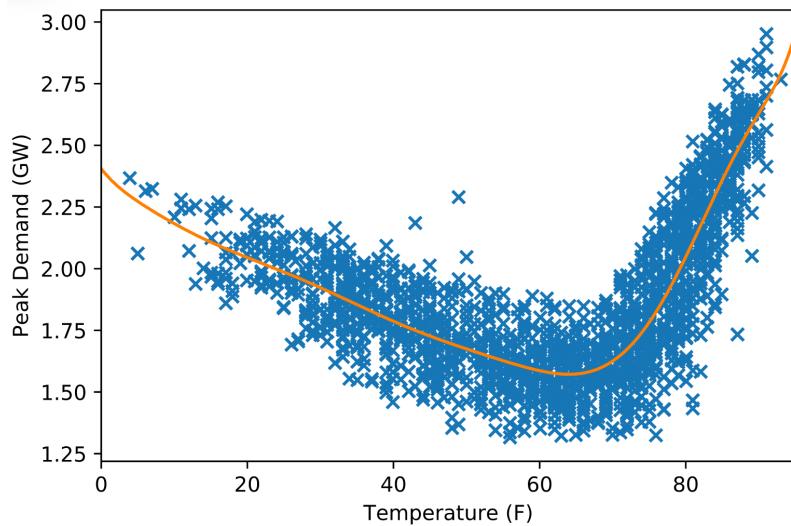
$$(K + \lambda m I)\alpha = y$$

$$\alpha = (K + \tilde{\lambda} I)^{-1}y$$

- dove il termine  $m$  è stato incorporato in  $\tilde{\lambda}$ . Attenzione: la matrice  $K$  è calcolata prima di effettuare la regressione
- Questa regressione è nota come **KERNEL RIDGE REGRESSION** ed è implementata in Python nel package scikit-learn
  - nell'implementazione Python il parametro  $\lambda$  è chiamato  $\alpha$



# Kernel Regression Ridge Applicata alla Previsione del Consumo Elettrico



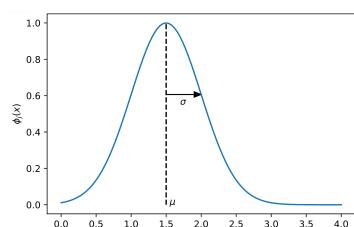
Con polinomio di grado 10 e  $\lambda = 10^{-6}$  otteniamo lo **stesso risultato** di prima ma **senza generare alcuna nuova variabile** rispetto al data set



## Gaussian Radial Basis Function (RBF)

- E' una funzione non lineare più complessa della polinomiale
  - Restituisce un vettore di  $k$  (**parametro**) componenti con valore in  $[0, 1]$ , 1 se  $x$ , il dato in input, coincide con  $\mu^{(i)}$  (**parametri**). Più  $x$  è distante da  $\mu^{(i)}$ , più il valore si avvicina a zero secondo la relativa distribuzione a campana con ampiezza  $\sigma$  (**parametro**)
  - RBF modella i dati sottostanti con combinazioni lineari di queste  $k$  funzioni a campana simili a Gaussiane (può approssimare qualsiasi funzione)
  - Più aumenta  $k$ , più il modello diventa complesso  
Più aumenta  $\sigma$  più il modello diventa semplice
  - Scelta di  $\sigma$ : uguale alla mediana della distanza tra i dati ed i centri  $\mu$ , in Python
  - $D = sqdist(X, X); sigma = np.median(np.sqrt(D)); K = np.exp(-D/(2*sigma**2))$
  - I  $\mu$  centri sono scelti di norma a caso tra i dati per migliorarne la distribuzione nello spazio dei dati

$$\phi(x): \mathbb{R} \rightarrow \mathbb{R}^k = \begin{bmatrix} \exp\left(\frac{-(x-\mu^{(1)})^2}{2\sigma^2}\right) \\ \exp\left(\frac{-(x-\mu^{(2)})^2}{2\sigma^2}\right) \\ \vdots \\ \exp\left(\frac{-(x-\mu^{(k-1)})^2}{2\sigma^2}\right) \\ 1 \end{bmatrix}$$



# Gaussian Radial Basis Function (RBF)

## Kernel ad Infinite Dimensioni

- la versione RBF multi-dimensionale:  $\phi(x) = \left\{ \exp\left(\frac{-\|x - \mu^{(j)}\|_2^2}{2\sigma^2}\right) : i = 1, \dots, k-1 \right\} \cup \{1\}$ 
  - $x$  e  $\mu^{(i)}$  qui sono vettori perciò il quadrato della loro differenza si ottiene con il quadrato della norma euclidea delle loro differenze
  - a differenza di quella mono-dimensionale, qui occorre normalizzare i dati
- la versione Kernel si applica come quella polinomiale senza modificare l'algoritmo  $K(x, z) = \exp\left(-\frac{\|x - z\|_2^2}{2\sigma^2}\right)$
- *perché è un Kernel ad infinite dimensioni ?*
  - RBF corrisponde al prodotto scalare tra 2 vettori ad infinite dimensioni  $\phi(x)^T \phi(z)$ , i.e. ognuno ha un centro in ogni punto dello spazio dei dati
  - infatti secondo l'espansione di Taylor  $e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i = \frac{1}{1!} x^1 + \frac{1}{2!} x^2 + \frac{1}{3!} x^3 + \frac{1}{4!} x^4 + \dots$  delle funzioni esponenziali,  $e^x$  corrisponde ad un polinomio di grado infinito; poichè il grado determina la dimensionalità, allora la dimensionalità è infinita

