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`

La regressione polinomiale è un'estensione della regressione lineare, che consente di modellare relazioni non lineari tra variabili. Mentre la regressione lineare si limita a trovare una retta che meglio si adatta ai dati, la regressione polinomiale introduce termini aggiuntivi, con potenze delle variabili indipendenti, permettendo di ottenere curve che si adattano meglio a set di dati più complessi.

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

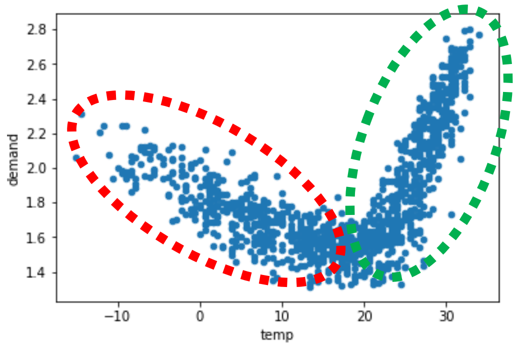
  $h(x) = \theta_0 + \theta_1 \cdot x + \theta_2 \cdot x^2 + \theta_3 \cdot x^3$ dove x è sempre il dato di input
- Su 2 variabili *a* e *b* un modello di grado 2 ha invece 6 termini

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

l'algoritmo di ottimizzazione (come il metodo dei minimi quadrati) può essere usato per trovare i parametri.

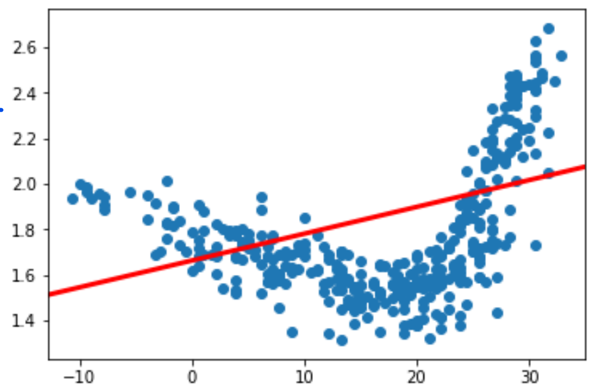
Esempio: Predizione del Consumo di Elettricità sull'Intero Anno

- 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 °C	y_1 = 2,13 GW
02/06/2016	x_2 = 27,1 °C	y_2 = 2,21 GW
03/06/2016	x_3 = 26,9 °C	y_3 = 2,22 GW

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

$$E(\boldsymbol{\theta}) = \tfrac{1}{3}\left(\left(\theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_1^2 - y_1\right)^2 + \left(\theta_0 + \theta_1 \cdot x_2 + \theta_2 \cdot x_2^2 - y_2\right)^2 + \left(\theta_0 + \theta_1 \cdot x_3 + \theta_2 \cdot x_3^2 - y_3\right)^2\right)$$

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

- calcoliamo il vettore gradiente di θ_0, θ_1, θ_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(θ)

$$E(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 \cdot x_i + \theta_2 \cdot x_i^2 - y_i)^2 \rightarrow \left[\begin{array}{l} \frac{\partial E(\theta)}{\theta_0} = \frac{2}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 \cdot x_i + \theta_2 \cdot x_i^2 - y_i) \\ \frac{\partial E(\theta)}{\theta_1} = \frac{2}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 \cdot x_i + \theta_2 \cdot x_i^2 - y_i) \cdot x_i \\ \frac{\partial E(\theta)}{\theta_2} = \frac{2}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 \cdot x_i + \theta_2 \cdot x_i^2 - y_i)^2 \cdot x_i^2 \end{array}\right] \nabla$$

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

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

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

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

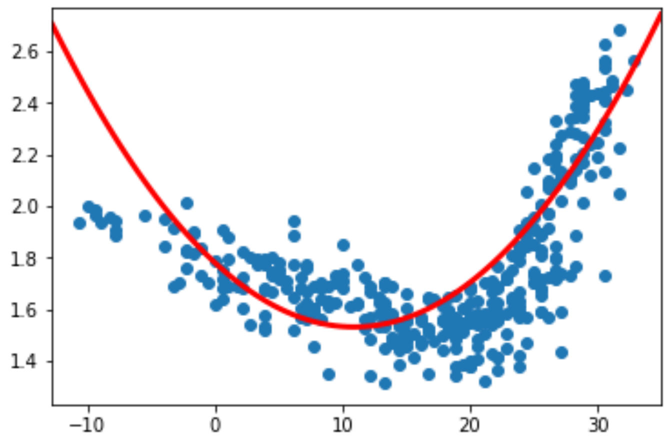
- quindi il gradiente (l’apice *k* è lo step *k*-esimo di discesa per θ_0 ,θ_1 ,θ_2)

Aggiornamento dei parametri

$$\begin{aligned}
\theta_0^{k+1} &= \theta_0^k - \eta(-\,4.37 + 2\theta_0^k + 52.8\theta_1^k + 1395.37\theta_2^k) \\
\theta_1^{k+1} &= \theta_1^k - \eta(-115.52 + 52.8\theta_0^k + 1395.37\theta_1^k + 36913.6\theta_2^k) \\
\theta_2^{k+1} &= \theta_2^k - \eta(-3054.72 + 1395.37\theta_0^k + 36913.6\theta_1^k + 977487\theta_2^k)
\end{aligned}$$

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



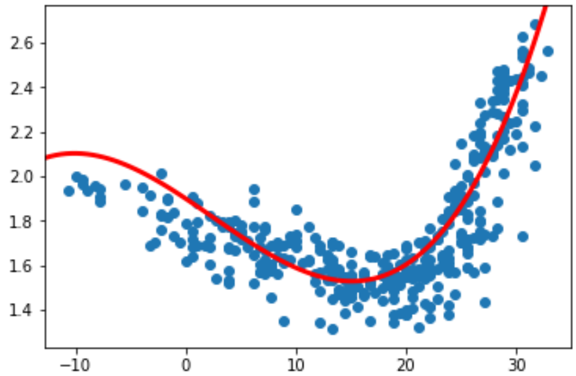
- Possiamo fare meglio ?

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

- Cosa succede con un modello di 3° grado ? → 4 variabili θ

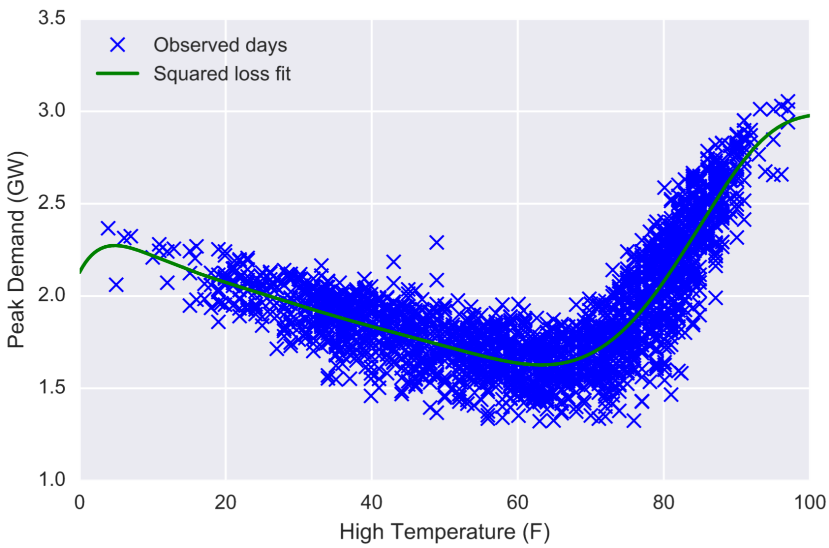
$$\hat{y} = \theta_0 + \theta_1 \cdot x + \theta_2 \cdot x^2 + \theta_3 \cdot x^3$$

- 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-elaborare 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

... e quante nuove variabili con grado 10 ??

- 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.]])
```

a	b
2	10
-3	20

1	a	b	a^2	$a \cdot b$	b^2
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)
```

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

fa il fit sulle variabili di training e le trasforma

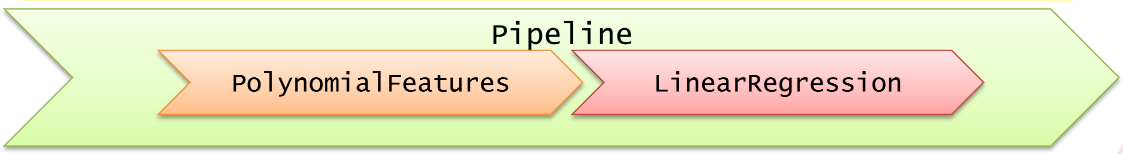
```
prm = LinearRegression()
prm.fit(poly.fit_transform(X_train), y_train)
```

- 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 forniamo una lista degli elementi che la compongono, con un nome associato a ciascuna

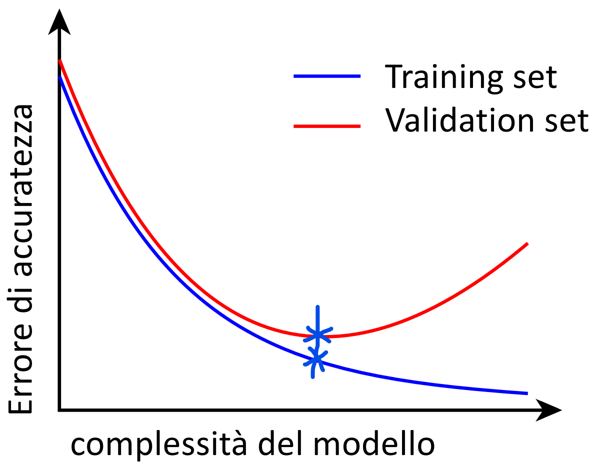
```
from sklearn.pipeline import Pipeline
prm = Pipeline([
  # nome     elemento
  ("poly",
     PolynomialFeatures(degree=2, include_bias=False)),
  ("linreg", LinearRegression())
])
```

- 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)
```

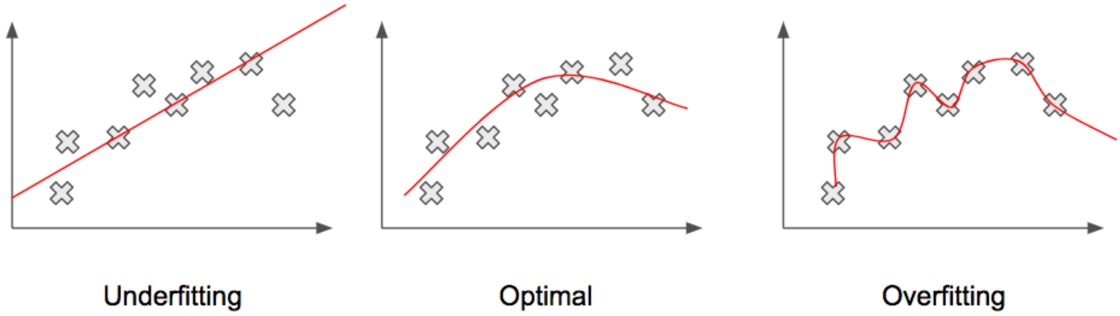

Complessità del Modello: *Overfitting* Vs *Generalizzazione*

- i dati devono essere divisi in training set e validation set
- qualsiasi modello di learning si estrae 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

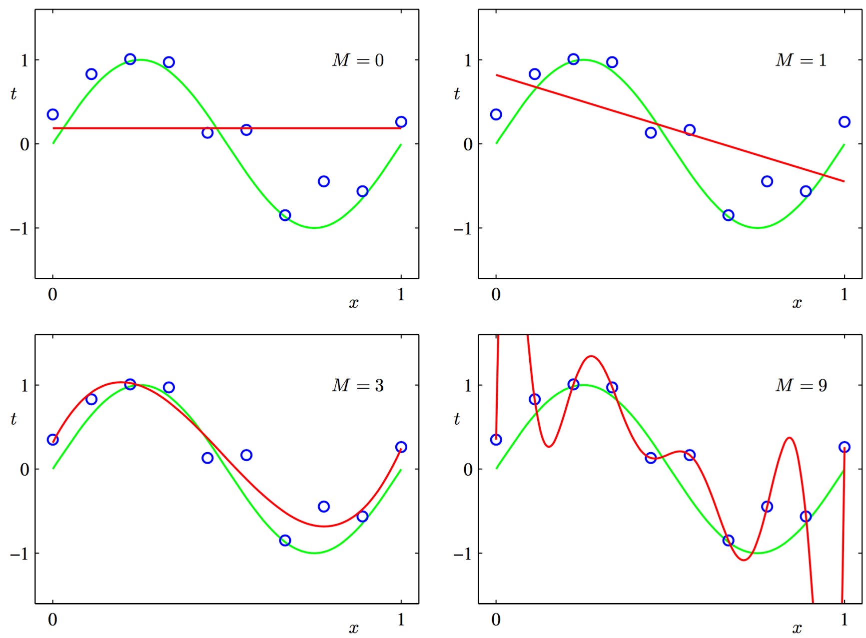
Complessità del Modello: Underfitting ed Overfitting



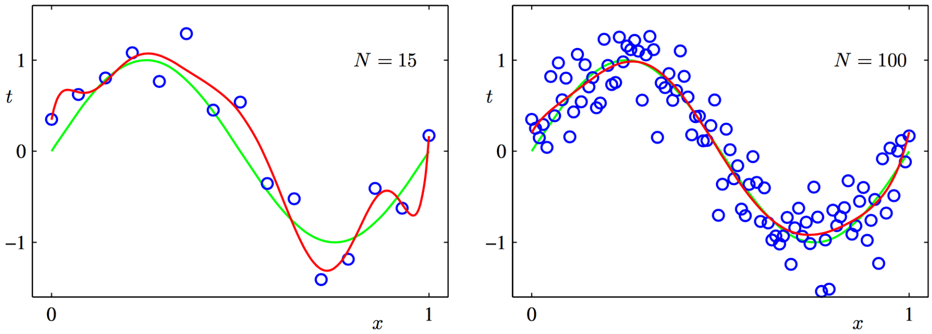
- Underfitting
  - il modello è troppo semplice e quindi inadeguato a rappresentare i dati, è insoddisfacente sia l'errore sul training, sia sul validation
- Overfitting
  - il modello è troppo complesso, l'errore sul training è significativamente inferiore a quello sul validation, il modello non generalizza dal training e non rappresenta adeguatamente dati ignoti
- Modello ottimale: minimizza l'errore sul validation set

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

- in blue i dati
- in verde la regressione ideale
- in rosso la regressione all'aumentare del grado M
- Underfitting con M < 3
- Overfitting con M >= 9 (diventa una interpolazione dei dati, non piú regressione)



Complessità del Modello e Quantità dei Dati



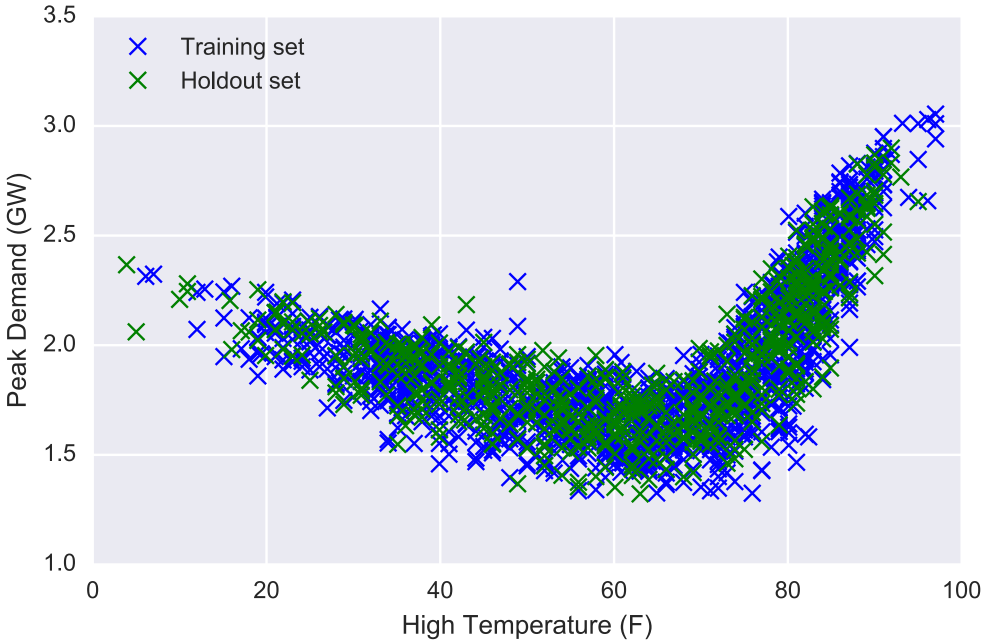
- Nella figura di sinistra, con polinomio di grado 9 e N=15 dati, il modello (curva rossa) è in overfitting
  - differenza significativa con la curva verde ottimale
- Con N=100 e stesso grado si ottiene un modello ottimale
  - quantità di dati e complessità del modello sono interdipendenti
  - con pochi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

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 **θ** del modello di learning, e.g. i coefficienti α e β 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, il learning rate etc.

Esempio di Holdout: Training e Validation

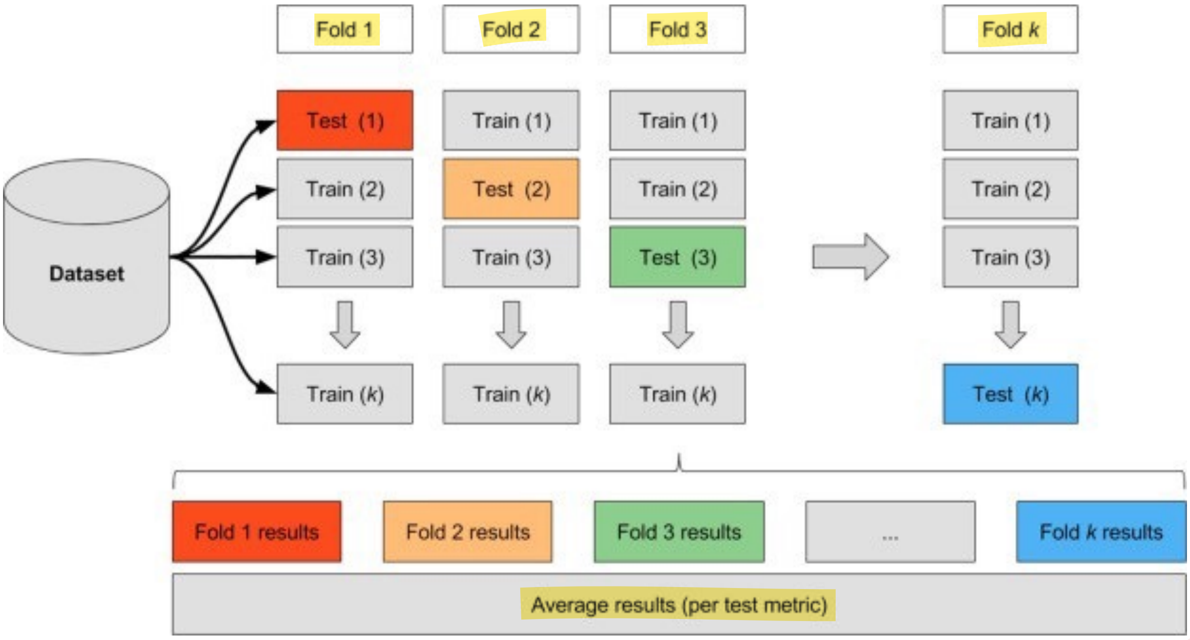


K-Fold Cross Validation (i)

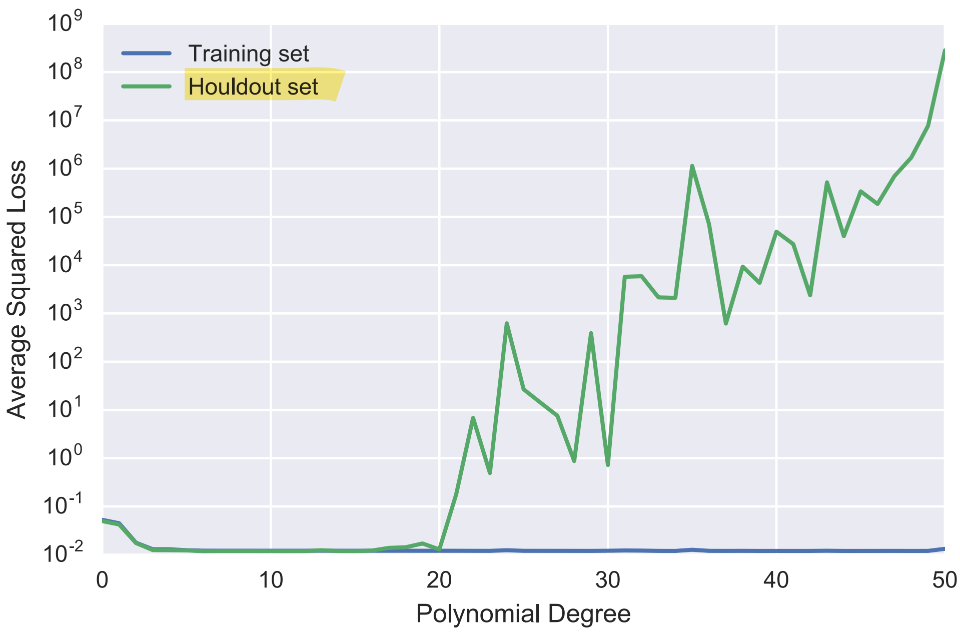
Fold 1	Fold 2	…	Fold k

- 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

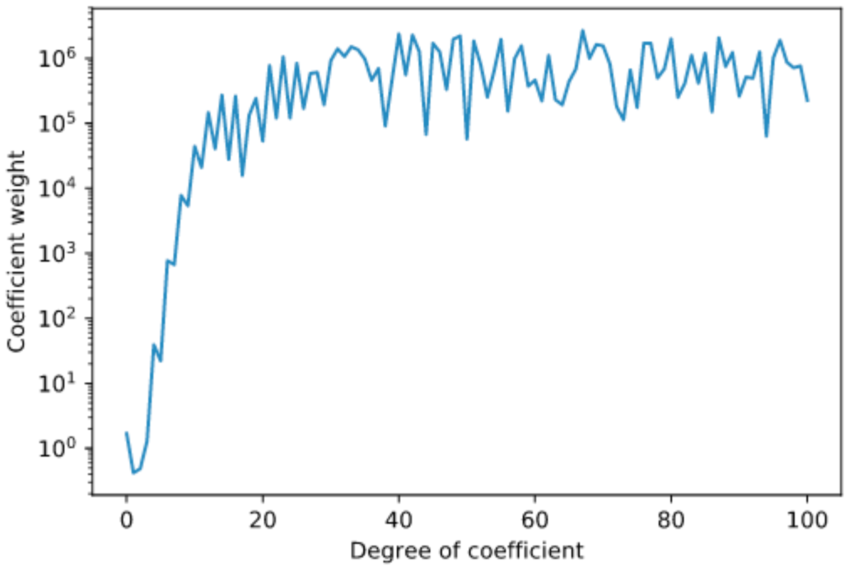
K-Fold Cross Validation (ii)



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



Regressione Polinomiale: Aumento dei coefficienti θ all'aumentare del grado



Mentre l'aumento del grado del polinomio può migliorare l'adattamento ai dati di addestramento, è fondamentale monitorare la crescita dei coefficienti e considerare l'uso di tecniche di regolarizzazione per prevenire l'overfitting e garantire che il modello generalizzi efficacemente su dati non visti.

- esempio sulla previsione del consumo di elettricità
- aumentando il grado del polinomio, aumenta rapidamente il valore assoluto di **θ** Il valore assoluto di un coefficiente indica quanto "grande" è, indipendentemente dal segno.

regolarizzazione L2

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

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

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 λ·θ
  - $0 \leq \lambda < \infty$ (iperparametro), con λ = zero nessuna regolarizzazione

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

n = num. parametri θ

è un termine di penalizzazione che aggiunge il quadrato della norma 2 dei coefficienti, pesato da un parametro lambda.

Minimizzazione dell’errore con Regolarizzazione

Obiettivo della regolarizzazione:
- Ridurre i coefficienti, evitando che diventino troppo grandi.
- Evitare l’overfitting, migliorando la capacità del modello di generalizzare su dati nuovi.

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

- 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

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

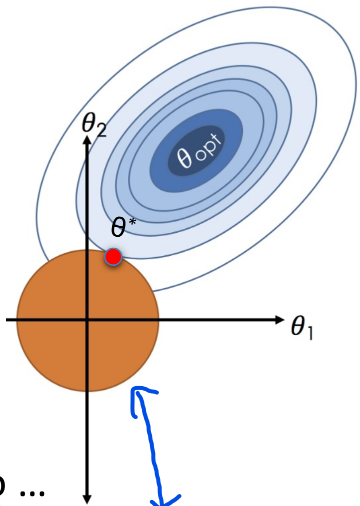
  - la soluzione ha un termine nuovo λI dove I è la matrice identità
  - perciò più λ è grande, più si riducono i coefficienti θ
- *nota importante sul termine* → $\|\theta\|_2^2 = \sum_{i=0}^{n-1} \theta_i^2$
  - la scala delle variabili di input si riflette sui parametri θ e poiché ora si sommano, incidentalmente predominerebbero quelli con scala maggiore → **standardizzare le variabili di input**

Se le variabili di input hanno scale molto diverse (ad esempio, una è tra 0 e 1 e un’altra tra 1000 e 2000), quelle con scala maggiore domineranno il termine di penalizzazione.
Soluzione: Standardizzare le variabili di input, in modo che abbiano media 0 e varianza 1.

Regolarizzazione: Interpretazione Geometrica

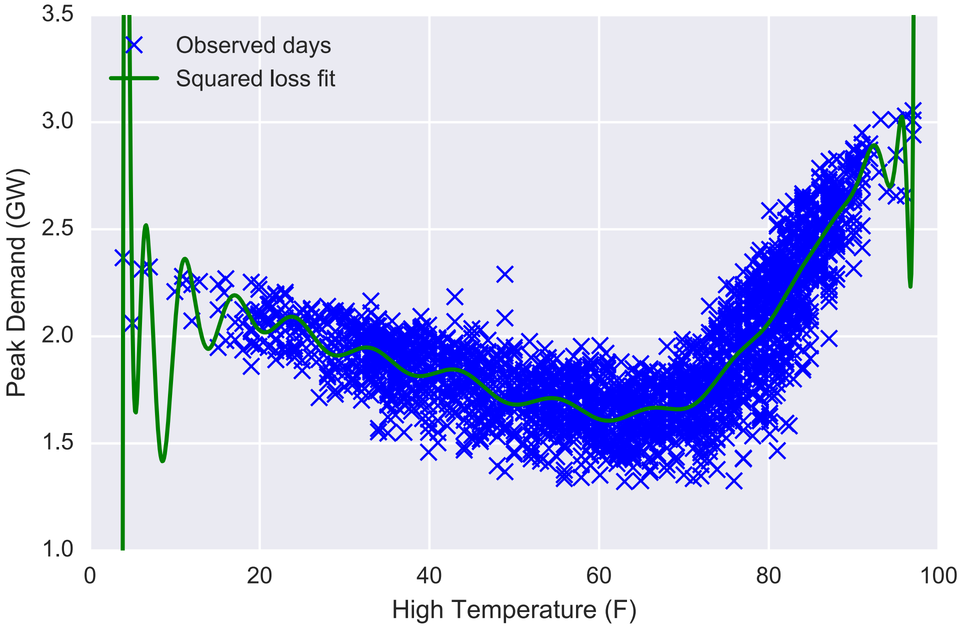
- Senza perdita di generalità consideriamo una regressione con parametri θ

  $\hat{y} = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2$
- θ_{opt} è il punto che minimizza la funzione di errore (θ_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 θ^* (punto rosso) di minimo vincolato …

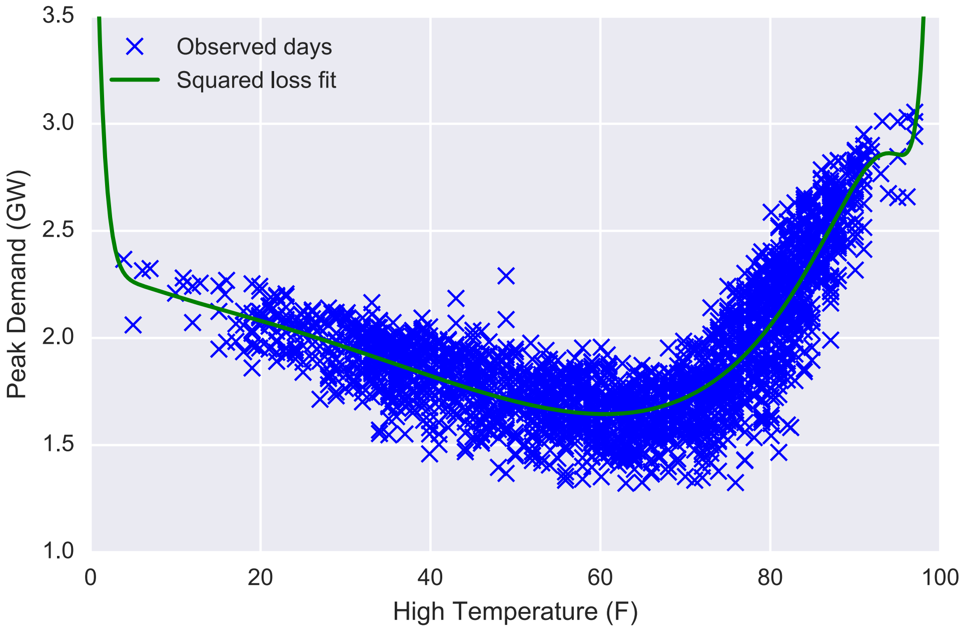


- … all’interno della relativa disequazione $\theta_1^2 + \theta_2^2 \leq r^2$ i.e. ipersfera centrata in 0,0 con raggio r che dipende da λ
  - all’aumentare di λ, si riduce r e la soluzione tende ad avvicinarsi all’origine con θ minori

Regressione Polinomiale: grado 50 senza regolarizzazione



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

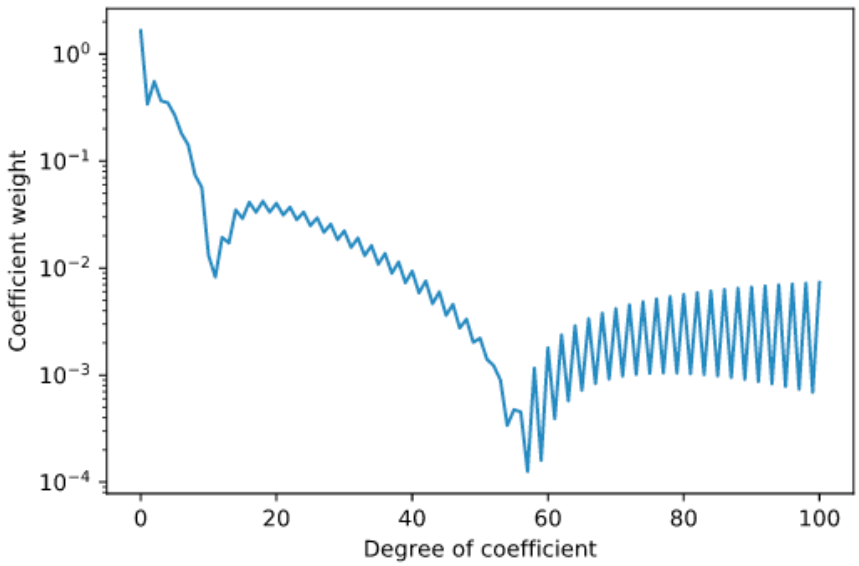


Regressione Polinomiale: Effetto della Regolarizzazione al variare del grado

Per gradi bassi, i coefficienti sono relativamente grandi.

Dopo un certo punto, il valore dei coefficienti diminuisce drasticamente grazie alla regolarizzazione.

Anche per gradi molto elevati, i coefficienti restano piccoli (circa 10^−3) rispetto al caso senza regolarizzazione (dove potrebbero arrivare a 10^6).



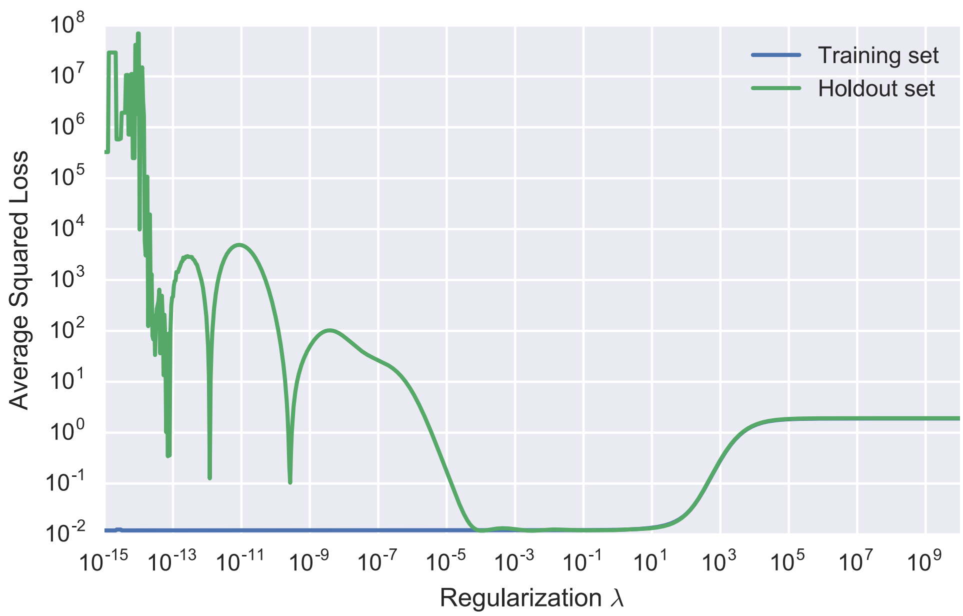
Senza regolarizzazione, i coefficienti della regressione polinomiale tendono a crescere molto rapidamente all'aumentare del grado, portando a overfitting.

Con la regolarizzazione (L2 - Ridge Regression), i coefficienti vengono ridotti per evitare che diventino troppo grandi.

Questo grafico dimostra che, anche se il grado del polinomio aumenta, la regolarizzazione mantiene i coefficienti molto più piccoli rispetto al caso senza regolarizzazione.

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

Regressione Polinomiale: Accuratezza con grado 50 al variare di λ



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

Individuazione del Migliore λ

- Il metodo più semplice per determinarlo è usare la k-fold cross validation: in python l'iperparametro λ è chiamato α

```
from sklearn.linear_model import RidgeCV
...
# 25 ascisse λ 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

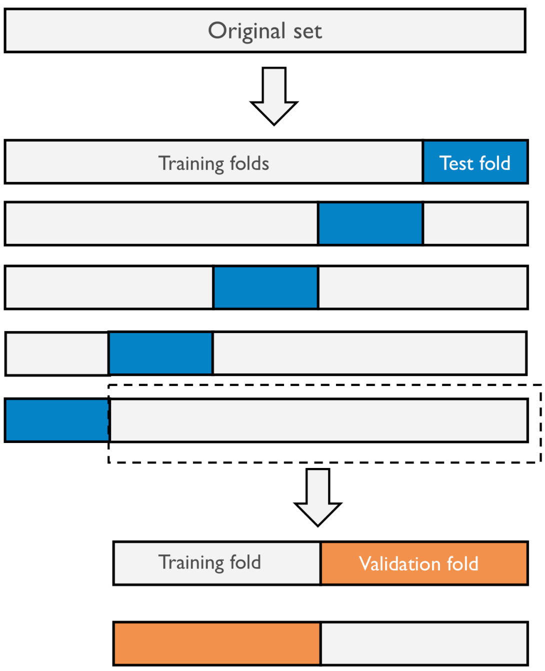
(Non fatto)

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 estrae un modello migliore di un altro, ma **stima gli iper-parametri** per estrarre il modello migliore

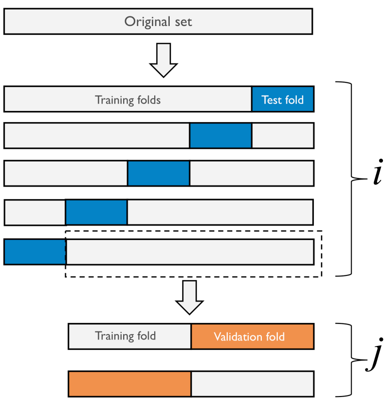


cross validation esterna per stimare l'accuratezza media sui dati

cross validation interna per trovare gli iperparametri migliori

(Non fatto)

Nested Cross Validation (ii)



i
cross validation esterna per stimare l'accuratezza media sui dati

j
cross validation interna per trovare gli iperparametri migliori

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 → $\theta_4 \cdot TV \cdot radio$

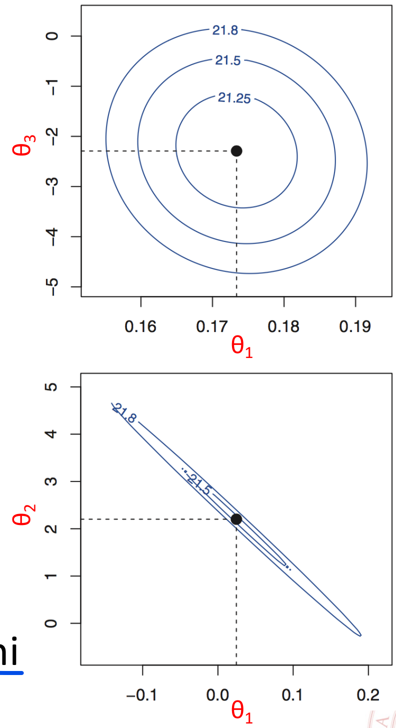
Collinearità: Dipendenze tra Variabili di Input (ii)

Questo significa che esiste un'unica combinazione ottimale di coefficienti per minimizzare l'errore. La soluzione è stabile, ossia piccoli cambiamenti nel training set non alterano molto i valori dei coefficienti.

Se scegliamo due variabili fortemente correlate, il modello trova molte combinazioni diverse di coefficienti che danno lo stesso errore minimo. Questo significa che il modello ha un ampio numero di soluzioni equivalenti.

- Soluzioni instabili: anche piccole variazioni nei dati di training possono portare a grandi cambiamenti nei valori dei coefficienti.
- Generalizzazione scarsa: il modello si comporta in modo imprevedibile su nuovi dati.
- Difficoltà nell'interpretazione: se i due parametri sono collineari, il modello potrebbe assegnare un peso molto grande a uno e un peso negativo all'altro, senza una chiara giustificazione.

- Nella fig. in alto l'errore di regressione minimo usando solo θ_1, θ_3 è ben definito (e.g. θ_{radio} , $\theta_{giornali}$)
- sotto invece con θ_1, θ_2 un ampio num. di loro valori produce lo stesso minimo → ampio n. di soluzioni (e.g. θ_{radio} , θ_{TV})
  - training set con differenze trascurabili generano modelli di regressione con coefficienti molto diversi -> soluzione instabile causata da dipendenza tra θ_1 e θ_2
  - accuratezza imprevedibile su dati ignoti
- la regolarizzazione risolve il problema poiché vincola/riduce le possibili soluzioni
  - soluzioni vincolate intorno all'origine



- Penalizza i coefficienti troppo grandi, costringendoli a rimanere vicini allo zero.
- Elimina la dipendenza tra le variabili collineari, garantendo che il modello abbia una soluzione più stabile.
- Previene cambiamenti drastici nei coefficienti dovuti a piccole variazioni nei dati.

Ridge Regression
Penalizza la somma dei quadrati dei coefficienti.
Riduce i coefficienti ma non li azzera completamente.
Tutte le variabili restano nel modello allora soluzione densa.

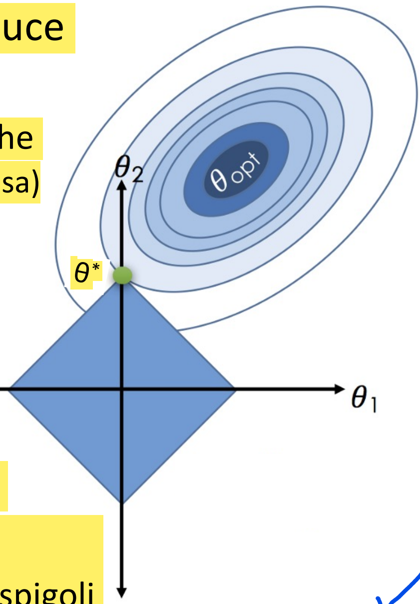
regolarizzazione L1

Regressione LASSO

Penalizza la somma dei valori assoluti dei coefficienti.
Può azzerare completamente alcuni coefficienti, eliminando le variabili meno importanti.
Soluzione più interpretabile allora seleziona automaticamente le feature più rilevanti.

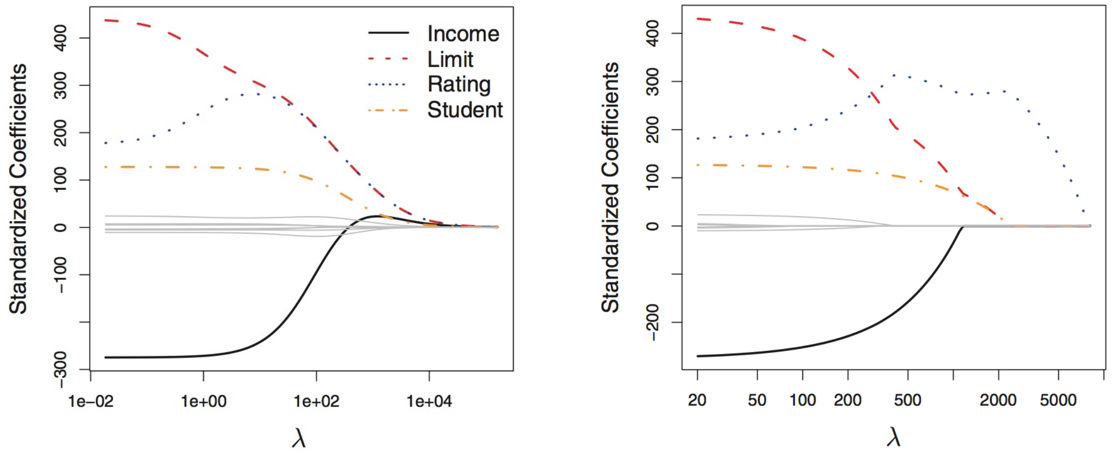
- La regressione Ridge, all'aumentare di λ riduce ma non azzera il valore dei parametri θ
  - perciò la soluzione utilizza tutte le variabili, anche quelle irrilevanti per la predizione (soluzione densa)
- Se penalizziamo i θ nella minimizzazione dell'errore con norma L1 invece che L2

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



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

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

Il p-value è una misura statistica che indica la probabilità che un certo coefficiente sia dovuto al caso.
Se un coefficiente ha un p-value alto, significa che potrebbe non essere significativo per la predizione e può essere eliminato.

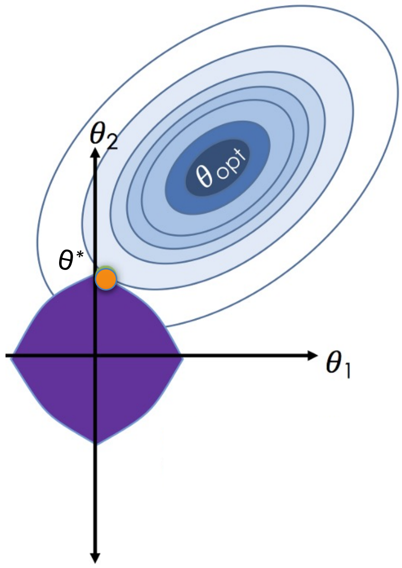
Selezione di Feature con P-Value in Python

- p-value indica la probabilità che il valore del coefficiente sia casuale
  - E.g. p-value 0.06 del coefficiente theta_1
- p-value è impiegato in combinazione con un livello di confidenza
  - E.g. con confidenza dello 0.95, il coefficiente theta_1 è scartato
- In python
  - from sklearn import linear_model
  - from regressors import stats
  - ols = linear_model.LinearRegression()
  - ols.fit(X, y)
  - stats.coef_pval(ols, X, y) Questa funzione calcola i p-value dei coefficienti della regressione. Aiuta a identificare quali variabili sono significative per la predizione.

Regressione Elastic Net: L1+L2

- Generalizza Ridge e LASSO con entrambe le penalizzazioni dei θ con norma L1 e L2

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



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

Regressione Polinomiale: Quanti parametri θ 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 ...*
  - $h_\theta(x_1, \ldots, x_{10})^2$ = θ1 + x1 θ2 + x2 θ3 + x3 θ4 + x4 θ5 + x5 θ6 + x6 θ7 + x7 θ8 + x8 θ9 + x9θ10 + x10 θ11 + x1^2 θ12 + x1 x2 θ13 + x1 x3 θ14 + x1 x4 θ15 + x1 x5 θ16 + x1 x6 θ17 + x1 x7 θ18 + x1 x8 θ19 + x1 x9 θ20 + x1 x10 θ21 + x2^2 θ22 + x2 x3 θ23 + x2 x4 θ24 + x2 x5 θ25 + x2 x6 θ26 + x2 x7 θ27 + x2 x8 θ28 + x2 x9 θ29 + x10 x2 θ30 + x3^2 θ31 + x3 x4 θ32 + x3 x5 θ33 + x3 x6 θ34 + x3 x7 θ35 + x3 x8 θ36 + x3 x9 θ37 + x10 x3 θ38 + x4^2 θ39 + x4 x5 θ40 + x4 x6 θ41 + x4 x7 θ42 + x4 x8 θ43 + x4 x9 θ44 + x10 x4 θ45 + x5^2 θ46 + x5 x6 θ47 + x5 x7 θ48 + x5 x8 θ49 + x5 x9 θ50 + x10 x5 θ51 + x6^2 θ52 + x6 x7 θ53 + x6 x8 θ54 + x6 x9 θ55 + x10 x6 θ56 + x7^2 θ57 + x7 x8 θ58 + x7 x9 θ59 + x10 x7 θ60 + x8^2 θ61 + x8 x9 θ62 + x10 x8 θ63 + x9^2 θ64 + x10 x9 θ65 + x10^2 **θ66**
  - ***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 θ con ***n*** variabili e grado ***g***?

- $h_\theta(x_1, \ldots, x_n)^g = \ldots.$??
- 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 (feriale o festivo) 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. consideriamo questo polinomio $(1+x_1z_1+x_2z_2)^2$
  - dove $x = (x_1, x_2)$ e $z = (z_1, z_2)$ 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 x e z in 2D

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

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

Soluzione all'Esplosione della Dimensionalità (ii)
Funzioni Kernel

- sviluppiamo interamente il polinomio di prima

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

- cosa c'è di interessante nel risultato finale ?
  - il quadrato del prodotto scalare dei vettori iniziali *x* ed *z* in 2 dimensioni + 1 è uquale 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
    - Il quadrato della somma di **2 moltiplicazioni** x_1z_1 , x_2z_2 produce lo stesso effetto della somma di **6 moltiplicazioni**
  - **KERNEL TRICK:** vale per ogni grado *g* e num. *n* di dimensioni
    - **x**=(x_1 ,…, x_n) **z**=(z_1 ,…, z_n) *Kernel*(**x**, **z**) = (**x** · **z** + 1)^g

Soluzione all'Esplosione della Dimensionalità (iii)
Costi Computazionali con e senza Kernel

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

Grado del polinomio di trasformazione	*ϕ(x)• ϕ(z)*	*ϕ(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	$(\mathbf{x}\bullet\mathbf{z})^2$	n(n+1)/2	2525 r^2	$n\,r^2/2$	50 r^2
3	$(\mathbf{x}\bullet\mathbf{z})^3$	n(n+1)(n+2)/6	85850 r^2	$n\,r^2/2$	50 r^2
4	$(\mathbf{x}\bullet\mathbf{z})^4$	n(n+1)(n+2)(n+3)/24	2.21x10^6 r^2	$n\,r^2/2$	50 r^2

Altre Funzioni Kernel

- Kernel (non lineari) possono modellare data set 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 $\quad K(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y})$

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

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

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

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

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 ?

*E*sempio di regressione quadratica in una variabile

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

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

Come Incorporare le Funzioni Kernel nella Regressione ?

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

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

i coefficienti ottimali θ sono combinazioni lineari dei dati

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

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

- perché 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 α_i da apprendere
    → complessità di training circa **quadratica** rispetto al num. istanze

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

i coefficienti ottimali θ* sono combinazioni lineari dei dati

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

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

notazione vettoriale

  - dove $K \in \mathbb{R}^{m \times m}$ è 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*α – *y* è il quadrato della sommatoria interna; il risultato è la sommatoria di quadrati di un vettore, i.e. una norma al quadrato

Gradiente della Regressione con Kernel

- derivata della funzione d'errore rispetto ai parametri α

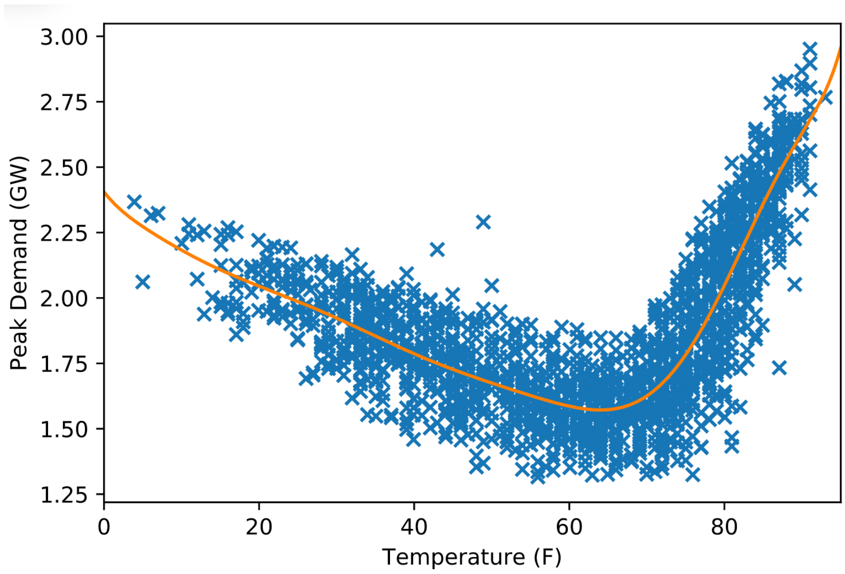
$$\nabla_\alpha \left(\frac{1}{m} \|K\alpha - y\|_2^2 + \lambda \alpha^T K \alpha \right) = \frac{2}{m} K(K\alpha - y) + 2\lambda K\alpha$$

- poniamola a zero e otteniamo il vettore gradiente α

$$2KK\alpha + 2\lambda m K\alpha = 2Ky$$
$$(K + \lambda m I)\alpha = y$$
$$\alpha = (K + \tilde{\lambda} I)^{-1} y$$

- 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 Pyhton nel package scikit-learn
  - nell'implementazione Python l'iperparametro **λ** è chiamato α

Kernel Ridge Regression 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

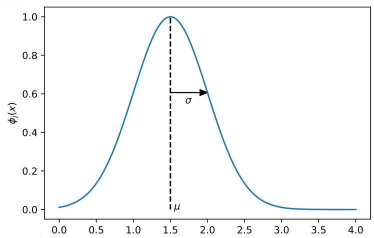
Gaussian Radial Basis Function (RBF)

- 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 σ (*parametro*)
  - RBF modella i dati sottostanti con combinazioni lineari di queste *k* funzioni a campana simili a Gaussiane (può approssimare qualsiasi funzione)
  - Piu aumenta *k*, più il modello diventa complesso
    Più aumenta σ più il modello diventa semplice
  - Scelta di σ: uguale alla mediana della distanza tra i dati ed i centri μ, in Python

```
D = sqdist(X,X); sigma = np.median(np.sqrt(D)); K = np.exp(-D/(2*sigma**2))
```

  - I μ centri sono scelti di norma a caso tra i dati per migliorarne la distribuzione nello spazio dei dati

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



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 $\boldsymbol{\mu}^{(j)}$ 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 delle funzioni esponenziali, e^x corrisponde ad un polinomio di grado infinito; poiché il grado determina la dimensionalità, allora la dimensionalità è infinita

$$e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i = \frac{1}{1!}x^1 + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 + \dots..$$