Applicazioni Predittive di Variabili Continue con Regressione Lineare

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

Applicazioni Predittive con Regressione Lineare

Machine Learning

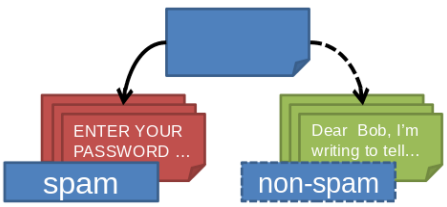
- Il *Machine Learning (ML)* fornisce metodi generali per estrarre modelli di conoscenza da insiemi di dati, esempi:
  - classificare i clienti di un'azienda in fedeli ed a rischio di abbandono (acquisire un nuovo cliente costa 6 volte di più di un cliente esistente)
  - classificare recensioni in positive, negative e neutre
  - classificare immagini di neoplasie in benigne e maligne
  - predire le propensioni di acquisto di ogni persona
  - predire quali clienti saranno buoni clienti (i.e. assicurazioni e sinistri)
  - predire l'andamento delle vendite di ogni prodotto/servizio
  - predire il consumo energetico, l'andamento di un titolo di borsa etc.
- Il ML è applicabile a qualsiasi problema in cui esistono dati sufficienti con l'obiettivo di scoprire un modello di conoscenza
  - funzione che associ ad ogni dato di input una classe/valore numerico

Machine Learning: Tipi di Apprendimento

- Tipi di learning in base al tipo di variabile in output da predire
  - prevedere il valore di variabili discrete
    - e.g. classificare i clienti in fedeli e non fedeli, polarità recensioni ...
  - prevedere il valore di variabili continue
    - e.g. predire l'andamento dei consumi energetici, delle vendite di un prodotto, il valore di un titolo di borsa ...
- tipi di learning in base al tipo di variabili in input di training
  - variabili strutturate, e.g. tabellari con domini noti per ogni attributo
    - e.g. classificare i clienti in fedeli e non fedeli in base agli acquisti, prevedere il consumo energetico, il valore di titoli di borsa etc.
  - variabili destrutturate, e.g. frasi, documenti, post, email ...
    - e.g. classificare email spam/non spam, recensioni positive/negative
- tipi di algoritmi di learning: supervisionati, non supervisionati:
  - non supervisionati quando i dati non sono preclassificati, e.g. **clustering**

Processo di Estrazione della Conoscenza

- Vale per ogni sistema di intelligenza artificiale basato sul learning
- E' costituito da queste fasi (learning supervisionato):
  - raccolta dei dati, analisi esplorativa e della loro qualità
  - trattamento di valori mancanti, normalizzazione, standardizzazione
  - selezione di feature, ossia variabili rilevanti rispetto agli obiettivi
  - divisione dei dati in training set, validation set e test set
  - estrazione di modelli di conoscenza dal training set con algoritmi di ML
    - individuando i parametri che massimizzino l'accuratezza sul validation set; il test set misura l'accuratezza sui dati nuovi/ignoti a regime
  - interpretazione dei migliori modelli di conoscenza estratti
  - deployment della conoscenza in applicazioni
- Esempio, riconoscere email di spam:
  - tipo di learning: *supervisionato con variabile di output discreta, dati di input destrutturati*



Regressione Lineare: Predire Variabili Continue

- ***Regressione*** lineare: stima da un data set una funzione lineare
  - che associ variabili di input (Domini), i.e. variabili indipendenti, ad una variabile di output (Co-dominio), i.e. dipendente
  - con una sola variabile indipendente si parla di regressione *univariata*
  - con più variabili indipendenti si ha la regressione *multivariata*
- Esistono diversi metodi di regressione, che estraggono funzioni con andamenti diversi (polinomiali, logaritmiche, …)
- In ogni caso, la funzione ha dei parametri i cui valori devono essere determinati in modo diretto o numericamente
  - metodo diretto: soluzione ottima ma solo per problemi convessi, richiede l'inversione di matrici con costo $O(N^3)$ e intera allocazione in memoria
    - inadeguato con ampie quantità di dati, impossibile con big data
  - metodo numerico: discesa sul gradiente, non garantisce soluzioni ottime, ma è parallelizzabile, incrementale e anche per problemi non-convessi

Caso di Studio 1: Previsione del Consumo di Elettricità

- Una compagnia elettrica deve dimensionare in modo ottimale la produzione giornaliera di energia elettrica
  - producendo poca energia non si soddisferebbe la domanda
  - un eccesso di produzione comporterebbe costi inutili perché è difficile e costoso immagazzinarla
- Per questo c'è interesse nel prevedere il consumo di energia elettrica in una determinata area nel giorno successivo
- Questo consumo dipende da molti fattori ed è difficile da determinare con sufficiente certezza
- Possiamo però fare ipotesi su quali siano i fattori più importanti e verificare se possiamo effettuare stime su di essi

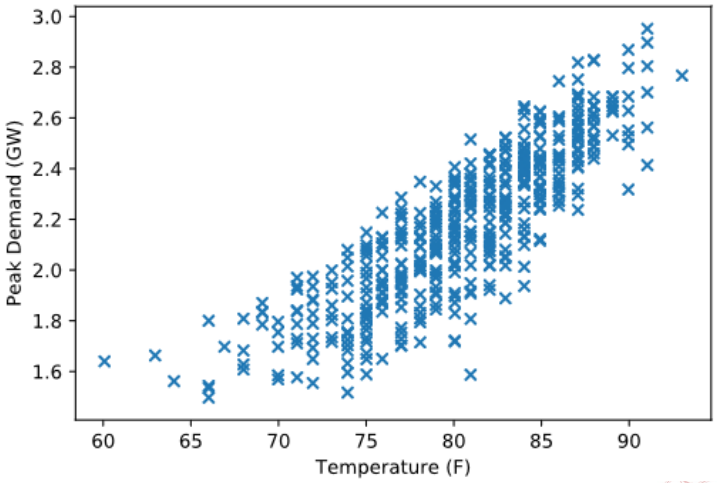
Previsione del Consumo di Elettricità: Analisi dei Dati Storici

- La compagnia ha a disposizione uno storico dei picchi di consumo di corrente registrati giornalmente
- Possiamo confrontare questi dati con altri dati storici per cercare fenomeni correlati su cui effettuare le previsioni
- Ad esempio si ipotizza che, nei mesi estivi, una quota consistente di energia sia consumata per l'aria condizionata
- In tal caso, il consumo nell'arco di una giornata può essere correlato alla temperatura: più fa caldo, più sarà elevato
- Per verificarlo, la compagnia confronta i dati storici sulle temperature registrate nella zona le scorse estati con quelli sui consumi …

data	temperatura media	picco consumo
01/06/2016	25,2 °C	2,13 GW
02/06/2016	27,1 °C	2,21 GW
…	…	…

Previsione del Consumo di Elettricità: Correlazione con la Temperatura

- Rappresentiamo i dati in un *grafico a dispersione* (scatter plot)
  - ogni punto corrisponde ad una data: la coordinata x ne indica la temperatura, la coordinata y ne indica il picco di consumo
- A prima vista si nota che effettivamente i consumi appaiono dipendere in modo particolare dalla temperatura
  - nei giorni più caldi il consumo massimo è stato circa il doppio rispetto ai più freddi



Regressione Lineare Univariata

- La **regressione *univariata*** consente di stimare una variabile dipendente *y* sulla base di un'unica variabile indipendente *x*
- In un modello di regressione ***lineare*** si assume che *y* vari in modo direttamente proporzionale ad *x*
- Il valore di *y* in funzione di *x* è quindi stimato come

$$\hat{y} = \alpha \cdot x + \beta$$

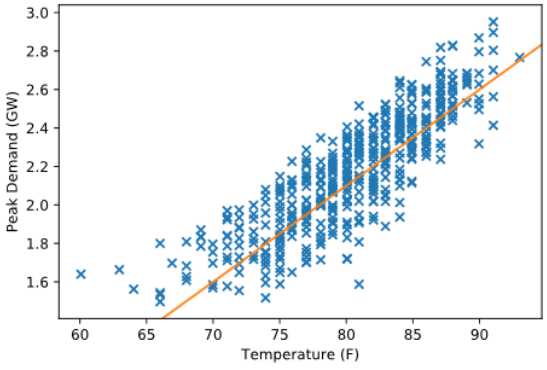
  - $\hat{y}$ indica una stima sul valore effettivo *y*
  - il coefficiente angolare α indica quanto *y* vari al variare di *x*
  - l'*intercetta* β determina il valore "base" di *y* quando *x* è nullo
- L'obiettivo dell'analisi di regressione è trovare i valori dei parametri α e β che rendano la stima più accurata possibile

Previsione del Consumo di Elettricità con Regressione Lineare

- Si può usare la regressione lineare univariata per modellare la dipendenza tra temperatura e consumo di elettricità
- Assumiamo quindi che il consumo si possa stimare come:

$$consumo = \alpha \cdot temperatura + \beta$$

  - α rappresenta il tasso con cui il consumo cresce al crescere della temperatura
  - β rappresenta la quota "base" di consumo non dipendente dall'aria condizionata



- La funzione si può rappresentare nel grafico di dispersione come una retta che approssima i dati effettivamente rilevati

Determinare i Parametri del Modello

- Se abbiamo una previsione affidabile della temperatura *t* del giorno dopo, possiamo quindi stimare il picco di consumo con

$$\alpha \cdot t + \beta$$

- … ma quali valori dovrebbero avere i parametri α e β?
- Non esiste una combinazione di valori "perfetta"
  - in quanto non esiste alcuna retta che modelli esattamente tutti i dati esistenti
- Dobbiamo quindi individuare i valori che diano la migliore approssimazione possibile
- Per poterlo fare occorre definire formalmente cosa intendiamo con la "migliore" approssimazione possibile

Errori del Modello

- Si supponga di fissare empiricamente i valori di α e β
- Stimando il consumo in base alla temperatura sui giorni passati, otterremo valori diversi da quelli reali misurati
- Possiamo misurare la differenza (*residuo*) tra consumo reale *y* e predetto $\hat{y}$ su ciascun caso dove i consumi reali sono noti

data	temp. media	picco consumo	picco stimato	differenza
01/06/2016	25,2 °C	2,13 GW	2,15 GW	+0,2 GW
02/06/2016	27,1 °C	2,21 GW	2,18 GW	-0,3 GW
…	…	…	…	…

- Minori sono le differenze, migliore è l'approssimazione

Misura dell'Errore del Modello

- Dati gli errori su *m* singoli casi, come misurare l'errore complessivo sull'insieme di predizioni ?
- Esistono diverse misure standard per l'errore nella regressione
- La più comune è la media dei quadrati dei singoli errori

$$errore = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

- Per ottimizzare l'accuratezza della predizione, dobbiamo trovare i parametri per cui l'errore sia il minimo possibile
- Come trovare una combinazione ottima di parametri ?
  - è possibile testare combinazioni casuali e tenere quella con errore minore, ma sarebbe inefficiente e inaccurato

Errore come Funzione dei Parametri del Modello

- Fissato un set di dati, l'errore è calcolato come una funzione continua sui parametri del modello di predizione
- Denotando con **θ** la combinazione di valori dei parametri incogniti e con h_θ il modello prescelto che li utilizza, scriviamo:

$$E(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

- Nel caso della regressione lineare univariata, $E(\theta)$ è una funzione sui due parametri α e β

$$E(\alpha,\beta) = \frac{1}{m}\sum_{i=1}^{m}(\underbrace{\alpha \cdot x_i + \beta}_{\text{predizione}} - y_i)^2$$

Errore in Funzione dei Parametri: Esempio

- Ipotizziamo di avere queste tre osservazioni su cui il modello deve essere addestrato …
  - in questo esempio sono tre per semplicità, in genere possono essere migliaia o anche milioni

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

- Per un modello lineare $\hat{y} = \alpha x + \beta$, con α e β ignoti, l'errore è:

$$E(\alpha,\beta) = \frac{1}{3}\left((x_1\alpha+\beta-y_1)^2+(x_2\alpha+\beta-y_2)^2+(x_3\alpha+\beta-y_3)^2\right)$$

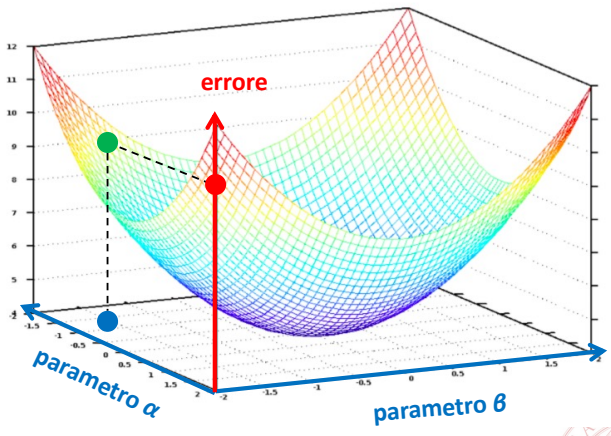
$$= \frac{1}{3}\left((25.2\alpha+\beta-2.13)^2+(27.1\alpha+\beta-2.21)^2+(26.9\alpha+\beta-2.22)^2\right)$$

$$\approx 697.69\alpha^2+52.8\alpha\beta-115.52\alpha+\beta^2-4.37\beta+4.78$$

  - idealmente vorremmo un errore pari a 0, ma non esiste una combinazione di valori di α e β per cui lo sia

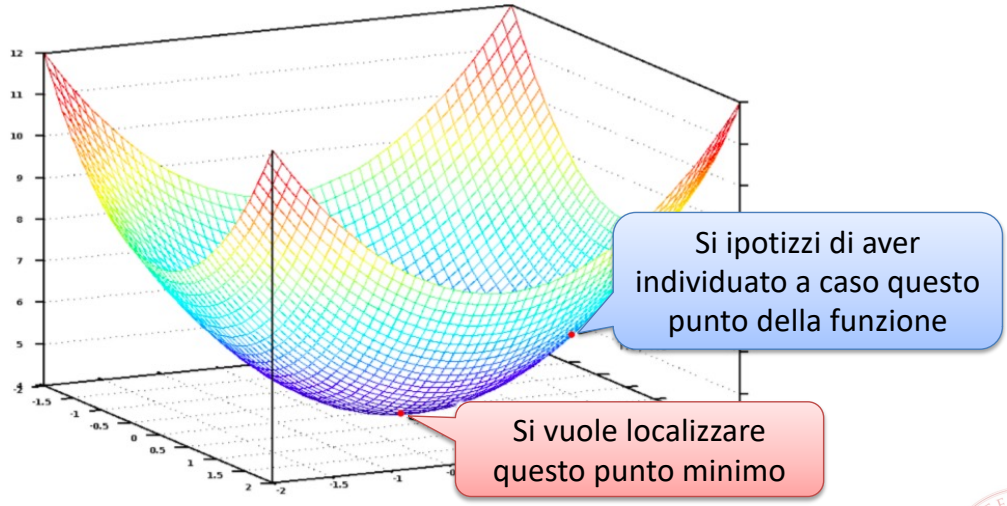
Curva della Funzione d'Errore

- Tracciando la funzione dell'errore *E* su *n* parametri, otteniamo una curva in *n*+1 dimensioni
  - nell'esempio corrente abbiamo 3 dimensioni (2 parametri + errore)
  - in generale possiamo averne un numero arbitrario
- Testando una combinazione di parametri arbitraria, l'errore calcolato su di essa fornisce un campione di tale funzione, ovvero un punto sulla curva
- Perché è una parabola ?



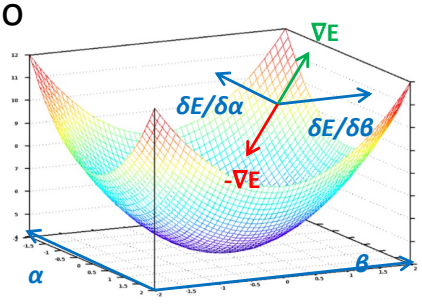
Trovare l'Errore Minimo

- Obiettivo della regressione è quindi trovare i parametri per cui il valore di tale funzione sia minimo



Gradiente di una Funzione

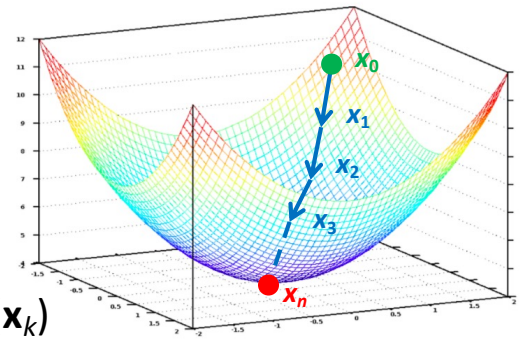
- Data una funzione *f* a più variabili, la ***derivata parziale*** su una di esse è la derivata di *f* considerando le altre come costanti
  - esempio: $f(a,b) = 3a + b^2 \quad \frac{\delta f}{\delta a} = 3 \quad \frac{\delta f}{\delta b} = 2b$
- Il ***gradiente*** ∇f è il vettore delle derivate parziali della funzione *f* per ciascuna delle sue variabili
- Intuitivamente, il gradiente $\nabla f(\boldsymbol{x})$ calcolato in un punto $\boldsymbol{x}$ indica l'inclinazione della curva nel punto stesso
  - dal gradiente si può quindi determinare la direzione in cui la curva sale o scende più ripidamente da un punto determinato



Discesa del Gradiente

- La ***discesa del gradiente*** è un metodo iterativo per trovare un minimo locale di una funzione "seguendo" il suo gradiente

1. Si parte da un punto $\mathbf{x}_k = \mathbf{x}_0$ a caso
2. Si valuta la funzione f ed il suo gradiente ∇f nel punto $\mathbf{x}_k$
3. Ad $\mathbf{x}_k$ si sottrae un vettore proporzionale al gradiente per ottenere un punto $\mathbf{x}_{k+1}$ con $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k)$



4. Si pone $k \leftarrow k+1$ e si esegue l'iterazione successiva dal punto 2, ripetendo fino alla convergenza ad un minimo

- La discesa del gradiente è un metodo generale impiegato da avanzati algoritmi di machine e di deep learning con reti neurali

Discesa del Gradiente: Iterazioni

- Al passo *i* dell'algoritmo di discesa del gradiente, calcolato il gradiente $\nabla f(\mathbf{x}_k)$, il punto successivo $\mathbf{x}_{k+1}$ è calcolato così:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \cdot \nabla f(\mathbf{x}_k)$$

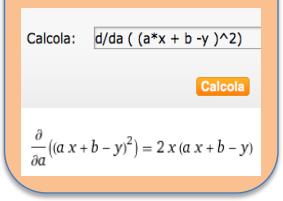
- η costituisce la lunghezza del passo di discesa (*step size*), il rapporto tra gradiente e spostamento ad ogni passo
- η è un **iperparametro** dell'algoritmo di discesa da impostare
  - noto nelle reti neurali come **learning rate**
  - se troppo basso l'algoritmo impiega molte iterazioni per convergere
  - se troppo alto l'algoritmo potrebbe rallentare nel trovare il minimo per via di avanzamenti troppo ampi da un punto all'altro della curva
  - in alcuni metodi η non è costante e viene fatto variare da un'iterazione all'altra

Regressione Lineare Univariata con Discesa del Gradiente

- Applicando la discesa del gradiente sulla funzione d'errore, possiamo trovare i valori dei parametri per cui è minima

$$E(\alpha,\beta)=\frac{1}{m}\sum_{i=1}^{m}(\underbrace{\alpha\cdot x_i+\beta}_{\text{predizione}}-y_i)^2$$

e.g. youmath.it



- Le derivate parziali che compongono il gradiente della funzione d'errore sono:

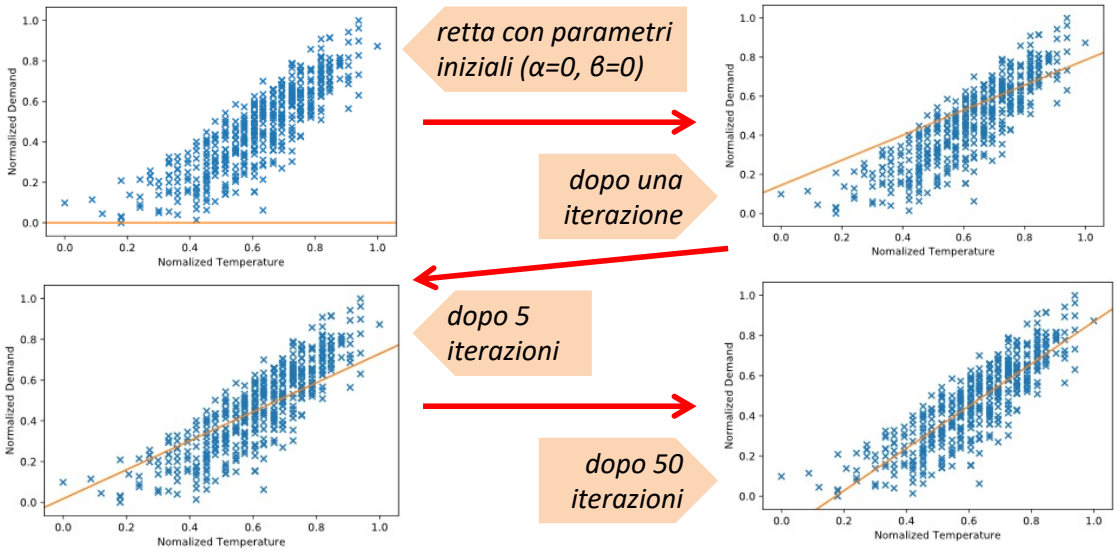
$$\frac{\delta E}{\delta\alpha}=\frac{1}{m}\sum_{i=1}^{m}2(\alpha\cdot x_i+\beta-y_i)\cdot x_i \qquad \frac{\delta E}{\delta\beta}=\frac{1}{m}\sum_{i=1}^{m}2(\alpha\cdot x_i+\beta-y_i)$$

- Ad ogni k-esima iterazione, i parametri sono aggiornati così:

$$\alpha_{k+1}\leftarrow\alpha_k-\frac{2\eta}{m}\sum_{i=1}^{m}(\alpha_k\cdot x_i+\beta_k-y_i)\cdot x_i \qquad \beta_{k+1}\leftarrow\beta_k-\frac{2\eta}{m}\sum_{i=1}^{m}(\alpha_k\cdot x_i+\beta_k-y_i)$$

Discesa del Gradiente: Progressione del Modello di Regressione

- Visualizzando la retta di regressione con i parametri alle varie iterazioni, vediamo come approssimi sempre meglio i dati



Implementazione in Python e NumPy: Discesa del Gradiente from Scratch

- Siano `temp` e `peak` vettori NumPy (o serie pandas) con i dati raccolti su **temperature** e **picchi dei consumi** su diverse date
  - temp è x ossia i dati noti, peak è y ossia i valori da prevedere

```
# inizializziamo i due parametri e impostiamo un passo
alpha, beta = 0, 0  # parametri da apprendere
step_size = 0.001   # iperparametro η
# per un numero fisso di iterazioni...
for it in range(50):
  # calcolo i residui e da questi le derivate parziali
  error = alpha * temp + beta - peak # errore della previsione
  d_alpha = 2 * (error * temp).mean() # derivata per alpha
  d_beta = 2 * error.mean()           # derivata per beta
  # aggiorno i parametri con discesa del gradiente
  alpha -= step_size * d_alpha
  beta -= step_size * d_beta
```

sono evidenziati in rosso i vettori

Implementazione in Python e NumPy: Predizione

- Una volta calcolati i valori di `alpha` e `beta`, data una temperatura `temp`, il picco di consumo stimato sarà:

  `alpha * temp + beta`

- Il procedimento mostrato può essere modificato in diversi modi:
  - invece di avere un numero fisso di iterazioni, fermare l'algoritmo quando gli spostamenti da un'iterazione all'altra sono (quasi) nulli
  - modificare il passo dinamicamente da un'iterazione all'altra
  - normalizzare i dati per compensare differenze di scala tra i valori delle temperature e quelli dei picchi di consumo

Caso di Studio 2: Prezzi delle Case

- Un'agenzia immobiliare deve determinare i prezzi ideali per le case in vendita
- L'agenzia vuole creare un'applicazione per stimare in automatico tale prezzo per **massimizzare i profitti e ridurre i tempi di vendita**, oltre a fornire supporto a non esperti
- Sono disponibili i dati sulle case vendute in precedenza (numero di stanze, metratura, …), insieme al prezzo determinato da un agente esperto

Bedrooms	Sq. feet	Neighborhood	Sale price
3	2000	Normaltown	$250,000
2	800	Hipsterton	$300,000
2	850	Normaltown	$150,000
1	550	Normaltown	$78,000
4	2000	Skid Row	$150,000

Stima del Prezzo di una Casa tramite Regole

- Come stimare il prezzo di una casa date le sue caratteristiche?

Bedrooms	Sq. feet	Neighborhood	Sale price
3	2000	Hipsterton	???

- Un addetto stima il valore usando regole empiriche, basate sulla propria esperienza pregressa
  - il prezzo è tendenzialmente proporzionale alla metratura …
  - … ma alcuni quartieri sono più costosi di altri …
  - … e anche il numero di stanze incide …
  - … e tanti altri fattori, che qui ignoriamo per semplicità
- Queste regole possono essere codificate in un programma …

Implementazione in Python: Stima del Prezzo tramite Regole

```
def estimate_home_price(bedrooms, size, area):
  # il costo medio delle case è circa $200/sqf.
  price_per_sqf = 200
  # alcuni quartieri costano più, altri meno
  if area == "hipsterton":
    price_per_sqf = 400
  elif area == "skid row":
    price_per_sqf = 100
  # si stima un prezzo base in base alla grandezza
  price = price_per_sqf * size
  # si aggiusta la stima sul numero di camere da letto
  if bedrooms == 0:
    price -= 20000
  else:
    price += bedrooms * 1000
  return price
```


Stima del Prezzo tramite Regressione

- Un approccio hand-coded come questo richiede molto lavoro
  - lo sforzo per codificare un insieme complesso di regole è ingente
  - va ripetuto periodicamente per far fronte a cambiamenti
    - oscillazioni dei prezzi, riqualificazione di un quartiere, …
  - il risultato finale può non essere accurato e contenere errori grossolani
- Con la regressione, possiamo usare i dati disponibili per ottenere in automatico una funzione che predica i prezzi
  - basta fornire i dati disponibili in input all'algoritmo
  - si può riaddestrare periodicamente su dati aggiornati e misurarne l'accuratezza prima di farne il deployment
  - in basa alla natura dei dati ed al modello di regressione scelto, la previsione della regressione può migliorare le stime fatte da esperti

Regressione Lineare Multivariata

- Con la regressione ***multivariata*** si stima una variabile dipendente *y* sulla base di più variabili indipendenti $x_1, \ldots, x_n$
- La previsione avviene a partire da un numero arbitrario di variabili numeriche e categoriche
  - ogni variabile categorica si converte in più variabili ***binarie*** con valore 0 o 1, creandone tante quante sono i valori della variabile categorica
  - e.g. quartieri = {q1, q2, q3} → q1 = {0, 1}, q2 = {0, 1}, q3 = {0, 1}
- Nel caso della regressione lineare, la forma della funzione θ stimata è un iperpiano in uno spazio a *n*+1 dimensioni

$$h_\theta(x_1,\ldots, x_n) = \theta_0 + \theta_1 \cdot x_1 + \ldots + \theta_n \cdot x_n$$

  - ogni *coefficiente angolare* θ_i indica il "peso" della variabile x_i nella previsione
  - l'*intercetta* θ_0 indica il valore previsto quando **x** = 0

Stima del Prezzo come Somma di Fattori

- Con la regressione lineare, si considera il prezzo della casa come una somma pesata delle sue caratteristiche
  - La regressione determina il peso di ogni variabile numerica (es. metratura) ed ogni ogni variabile categorica (es. **quartiere**)
  - il prezzo è la somma delle variabili numeriche note moltiplicate per i rispettivi pesi dei parametri, compresi quelli categorici
- Il prezzo *P* di una casa di s metri quadri con n_b stanze da letto in un quartiere $\boldsymbol{q}$ si stima quindi come:

$$P(n_b, s, q) = \theta_0 + \theta_1 n_b + \theta_2 s + \Sigma_i (\theta_i \gamma(\mathbf{q}))$$

  - con parametri θ_0 , θ_1 , θ_2 e $\theta_{i1,..,iq}$ da apprendere (iq = numero dei quartieri)
  - con valori noti s, n_b e quartiere $\boldsymbol{q}$ dato dalla funzione γ di binarizzazione di $\boldsymbol{q}$

Implementazione in Python: Stima del Prezzo per Somma Pesata di Fattori

```
def estimate_home_price(
    bedrooms, size, area):
  # il costo stimato è la somma di
  price = (
    # numero di stanze * coefficiente
        2.2342 * bedrooms
    # metratura * altro coefficiente
    +   0.8978 * size
    # coefficiente basato sul quartiere
    +  12.4543 * (area == "hipsterton")
    -   6.4632 * (area == "skid row")
    # valore base fisso
    + 123.5434
  )
  return price
```

- Nella funzione mostrata, i pesi sono stati appresi con la regressione e sono fissati al suo interno
- Con la regressione scopriamo i loro valori ottimali

Regressione Lineare Multivariata: Notazione Vettoriale

- Il modello generale di regressione lineare multivariata

$$h_\theta(x_1,\ldots, x_n) = \theta_0 + \theta_1 \cdot x_1 + \ldots + \theta_n \cdot x_n$$

- riscritto in forma vettoriale abbiamo il vettore **x** delle variabili dipendenti e quello **θ** dei parametri da determinare (i.e. pesi):

$$h_\theta(\mathbf{x}) = \theta_0 + \boldsymbol{\theta} \cdot \mathbf{x} \qquad \text{con } \mathbf{x} = [x_1 \ldots x_n]$$

- Per ottenere una formulazione più compatta, si può inserire anche l'intercetta θ_0 nel vettore **θ** aggiungendo $x_0 = 1$ ad **x**
- Il modello si riduce così ad un prodotto scalare tra vettori

$$h_\theta(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

$$h_\theta(x_1,\ldots,x_n) = \begin{bmatrix}\theta_0 & \theta_1 & \cdots & \theta_n\end{bmatrix} \cdot \begin{bmatrix}1 & x_1 & \cdots & x_n\end{bmatrix}$$

Implementazione in Python: Stima del Prezzo in Notazione Vettoriale

```
def estimate_home_price(
  theta, bedrooms, size, area):
  # per la stima del costo contano
  features = np.array([
    # valore fisso che moltiplica l'intercetta
    1,
    # numero di stanze
    bedrooms,
    # metratura
    size,
    # coefficiente basato sul quartiere
    area == "hipsterton",# se True 1, altrimenti 0
    area == "skid row" # se True 1, altrimenti 0
  ])
  price = theta.dot(features)
  return price
```

- Con questo approccio i pesi sono raccolti in un vettore **θ** apposito
- Il risultato è il semplice prodotto scalare tra i pesi e il vettore delle caratteristiche

```
theta = np.array([123.5434, 2.2342, 0.8978, 12.4543, -6.4632])
estimate_home_price(theta, 3, 100, "hipsterton")
232.4803
```


Regressione Lineare Multivariata: Funzione d'Errore

- Come già visto, fissato un set di dati noti, possiamo valutare l'errore di un modello come media dei residui quadrati

$$E(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(\mathbf{x}_i) - y_i\right)^2$$

  - dove ricordiamo che *m* è il numero dei dati, *x* la variabile dipendente e *y* la variabile indipendente da predire
- Nel modello lineare multivariato ad *n* dimensioni si ha:

$$E(\theta_0,\theta_1,..,\theta_n) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta_o + \left(\sum_{j=1}^{n}\theta_j \cdot x_{i,j}\right) - y_i\right)^2$$

- … oppure denotato in forma vettoriale:

$$E(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}\cdot\mathbf{x}_i - y_i\right)^2$$

Regressione Lineare Multivariata: Discesa del Gradiente

- Dalla formula precedente calcoliamo la derivata parziale della funzione d'errore, con *n* parametri, per ogni parametro θ_p

$$\frac{\delta E(\boldsymbol{\theta})}{\delta \theta_p} = \frac{2}{m}\sum_{i=1}^{m}\left(\theta_0 + \left(\sum_{j=1}^{n}\theta_j \cdot x_{i,j}\right) - y_i\right)\cdot x_{i,p} = \frac{2}{m}\sum_{i=1}^{m}\left(\boldsymbol{\theta}\cdot\mathbf{x}_i - y_i\right)\cdot x_{i,p}$$

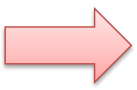
- Applicando la formula su tutti i parametri otteniamo il gradiente della funzione d'errore
- Nella discesa del gradiente ciascun parametro θ_p sarà quindi aggiornato iterativamente ad ogni passo ***k*** in questo modo:

$$\theta_{p,k+1} \leftarrow \theta_{p,k} - \frac{2\eta}{m}\sum_{i=1}^{m}\left(\theta_{0,k} + \left(\sum_{j=1}^{n}\theta_{j,k}\cdot x_{i,j}\right) - y_i\right)\cdot x_{i,p}$$

Regressione Lineare Multivariata: Notazione Matriciale

- Finora abbiamo rappresentato ciascuna delle *m* osservazioni su cui calcolare l'errore con un vettore $\mathbf{x}_i$ e un valore atteso y_i
- L'intero insieme delle *m* osservazioni può essere raccolto in una matrice **X** degli input e un vettore **y** degli output attesi
  - **X** ha una riga per ogni osservazione e una colonna per ogni variabile, più una colonna relativa all'intercetta con valori 1
  - **y** ha un elemento per ogni osservazione (corrispondenti alle righe)

Bedrooms	Sq. feet	Neighborhood	Sale price
3	2000	Normaltown	$250,000
2	800	Hipsterton	$300,000
2	850	Normaltown	$150,000
1	550	Normaltown	$78,000
4	2000	Skid Row	$150,000



variabili binarie per il quartiere

$$\mathbf{X} = \begin{bmatrix} 1 & 3 & 2000 & 0 & 0 \\ 1 & 2 & 800 & 1 & 0 \\ 1 & 2 & 850 & 0 & 0 \\ 1 & 1 & 550 & 0 & 0 \\ 1 & 4 & 2000 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 250000 \\ 300000 \\ 150000 \\ 78000 \\ 150000 \\ \vdots \end{bmatrix}$$

Regressione Lineare Multivariata: Discesa del Gradiente in Notazione Matriciale

- In questo modo, al posto delle singole derivate parziali, possiamo esprimere il gradiente dell'errore in forma vettoriale
  - nel prodotto con matrici considerare **θ** e **y** come matrici colonna (…×1)

$$\nabla E(\boldsymbol{\theta}) = \frac{2}{m}\mathbf{X}^T(\underbrace{\mathbf{X}\boldsymbol{\theta}-\mathbf{y}}_{\text{vettore residui}})$$

- Possiamo esprimere così l'aggiornamento dell'intero vettore dei parametri ad ogni iterazione ***k*** della discesa del gradiente

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \frac{2\eta}{m}\mathbf{X}^T(\mathbf{X}\boldsymbol{\theta}_k - \mathbf{y})$$

- Per trovare il vettore di parametri **θ** ottimo è sufficiente quindi applicare iterativamente questa formula

Implementazione in Python e NumPy

- Una volta costruiti la matrice `X` e il vettore `y` con i dati per il calcolo dell'errore, per ottimizzare il modello è sufficiente:

```
# estraggo numero di osservazioni e di feature
m, n = X.shape
# imposto la lunghezza del passo ed i parametri iniziali
step_size = 0.001
theta = np.zeros(n+1) # inizializza a 0 i parametri da
apprendere
# per un numero fisso di iterazioni ...
for it in range(50):
  # aggiorno i parametri in base al gradiente
  error = X.dot(theta) - y
  theta -= 2 * step_size / m * (X.T.dot(error))
```

- Possiamo quindi usare il vettore `theta` ricavato nella funzione mostrata in precedenza per effettuare stime su nuovi dati

Normalizzazione dei Dati

- In generale le variabili coinvolte in un modello di regressione possono utilizzare scale di valori molto diverse
  - ad es. nel primo caso di studio abbiamo temperature in °C sull'ordine delle decine e consumi in GW sull'ordine delle unità
- Questo rende difficile la convergenza della discesa del gradiente
- Una soluzione consisterebbe nell'usare lunghezze di passo differenziate per parametro
- Una soluzione più semplice è però normalizzare i dati in modo che tutte le variabili abbiano valori in un medesimo intervallo
- Possiamo eseguire la discesa del gradiente su dati normalizzati e "denormalizzare" i parametri ottenuti

Normalizzazione di una Variabile

- Dato un valore *x* di una generica variabile *X*, possiamo calcolarne il valore normalizzato nell'intervallo 0-1 così:

$$\tilde{x} = \frac{x - \min(X)}{\max(X) - \min(X)}$$

- Dato un vettore NumPy o una serie pandas x di valori, possiamo ottenerne la versione normalizzata in questo modo:

```
x_norm = (x - x.min()) / (x.max() - x.min())
```

- Per addestrare efficacemente un modello di regressione, è comune normalizzare tutte le variabili (anche la dipendente)

Uso del Modello su Dati Normalizzati

- Addestrando un modello su dati normalizzati, i suoi parametri saranno ottimizzati su di essi

$$h_{\tilde{\boldsymbol{\theta}}}(\tilde{\mathbf{x}}) = \tilde{\boldsymbol{\theta}} \cdot \tilde{\mathbf{x}}$$

- Per usare il modello occorre quindi coerentemente normalizzare gli input e denormalizzare gli output ottenuti
- In alternativa possiamo trasformare i parametri del modello su dati normalizzati per ottenere quelli validi sui dati originali

$$\theta_0 = \tilde{\theta}_0 \cdot \big(\max(y) - \min(y)\big) + \min(y) - \sum_{i=1}^{n} \tilde{\theta}_i \cdot \min(x_i)$$

$$\theta_i = \tilde{\theta}_i \cdot \frac{\max(y) - \min(y)}{\max(x_i) - \min(x_i)}$$

scikit-learn

- Abbiamo visto come implementare facilmente da zero il metodo di discesa gradiente in NumPy
  - ciò permette di comprendere profondamente i concetti visti
- Esistono però librerie che implementano già algoritmi efficienti per vari tipi di regressione
- Tra queste è nota ***scikit-learn***, una libreria general-purpose per il machine learning
  - usa le strutture dati di NumPy e pandas per rappresentare i dati
  - offre diversi algoritmi basati su interfacce simili, rendendoli facilmente intercambiabili
  - su ogni algoritmo permette di impostare diversi parametri, definendone valori di default di uso comune

Costruire un Modello di Predizione in scikit-learn

- I passi generali per usare scikit-learn sono:
  1. creare un oggetto modello "vuoto", definendo i parametri dell'algoritmo di addestramento (detti anche *iperparametri*), gli iperparametri sono definiti dal data scientist
  2. addestrare il modello fornendo un insieme di dati da cui apprendere i migliori parametri del modello, i parametri sono appresi
  3. effettuare predizioni fornendo nuovi dati al modello appreso e verificando le risposte che fornisce
- Per l'addestramento di un modello vanno forniti
  - una matrice NumPy o un DataFrame pandas **X** con una riga per ogni osservazione e una colonna per ogni attributo
  - un vettore NumPy o una serie pandas **y** che indica per ciascuna osservazione il valore "target" che il modello deve prevedere

Regressione Lineare in scikit-learn

- Vediamo ad esempio come costruire un modello di regressione lineare ed usarlo per effettuare stime o previsioni

1. Si istanzia un oggetto `LinearRegression` che rappresenta un modello "vuoto" (non addestrato)

```
from sklearn.linear_model import LinearRegression
lrm = LinearRegression()
```

2. Se ne invoca il metodo `fit` passando i dati su cui addestrarsi
   - matrice delle osservazioni X e vettore degli output attesi y

```
lrm.fit(X, y)
```

3. Col metodo `predict` sarà quindi possibile ottenere la stima della variabile dipendente per una o più osservazioni nuove

```
y_pred = lrm.predict(X_new)
```

Esempio: Previsione del Consumo di Elettricità in scikit-learn (1)

- Una volta caricati i dati su cui addestrare il modello…

```
temp = ...
peak = ...
```

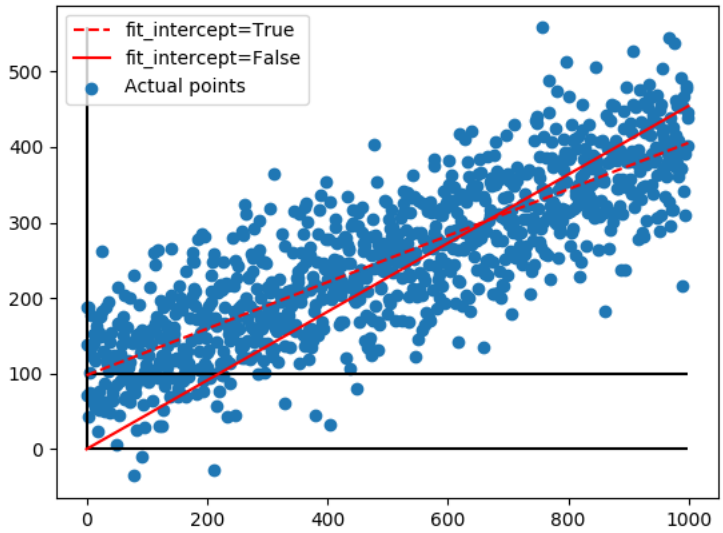
- … creiamo un'istanza di modello di regressione, specificando eventuali parametri
  - si può specificare `fit_intercept=False` per fissare l'intercetta a 0 o `normalize=True` per normalizzare i dati in norma 2

```
from sklearn.linear_model import LinearRegression
lrm = LinearRegression()
```

- … e addestriamo il modello sui dati caricati
  - converto il vettore di *m* elementi in matrice *m*×1 (*m* dati, 1 variabile)

```
lrm.fit(temp[:, None], peak)
```


Esempio: Previsione del Consumo di Elettricità in scikit-learn (1) – fit_intercept



- `impatto di fit_intercept = True oppure False`

Esempio: Previsione del Consumo di Elettricità in scikit-learn (2)

- Dopo aver addestrato il modello possiamo utilizzarlo:

```
>>> lrm.predict(25)
array([1.985])
```

  - con una temperatura di 25 °C il picco di consumo previsto è 1,985 GW
- Si possono ottenere più previsioni in una volta
  - come per la chiamata a `fit`, si fornisce una matrice *m*×1

```
>>> lrm.predict([[23], [25], [27]])
array([1.81431424, 1.9849483 , 2.15558237])
```

- Per vedere il vettore dei coefficienti angolari e l'intercetta …
  - con una sola variabile indipendente abbiamo un unico coefficiente

```
>>> lrm.coef_, lrm.intercept_
(array([0.08531703]), -0.14797750194929682)
```


Valutazione del Modello di Regressione

- Un modello di regressione è addestrato in modo da minimizzare l'errore su un insieme di dati noti
- L'obiettivo della regressione è ottenere un modello generale
  - i.e. capace di fare previsioni sui nuovi dati con la stessa accuratezza mostrata sui dati impiegati per ottenere il modello stesso
- Se invece l'accuratezza su nuovi dati è significativamente più bassa allora ci possono essere due problemi
  - il modello è affetto da overfitting, i.e. è troppo aderente ai dati di addestramento perdendo generalità -> ridurre il numero di variabili o regolarizzare il modello (vedremo come)
  - il training set impiegato non è statisticamente rappresentativo delle peculiarità dei nuovi dati
- Per scongiurare questi problemi occorre verificare/validare l'errore del modello su dati *non* usati per addestrarlo

Validazione con Metodo Hold-Out

- Il metodo *hold-out* prevede di dividere i dati a disposizione in due insiemi, secondo una proporzione predefinita (es. 70-30)
- Il *training set* è utilizzato per addestrare il modello di regressione, minimizzando l'errore su di esso
- Il *validation set* è usato dopo l'addestramento per verificare l'errore del modello su dati ignoti (capacità di generalizzazione)
- Se l'errore sul validation set è simile a quello sul training set, si assume che il modello abbia generalizzato bene i dati
- Un errore sul validation set significativamente maggiore indica overfitting o il training set non rappresenta il dataset
  - nel primo caso cambiare parametri, aggiungere o togliere variabili …
  - nel secondo ripetere la suddivisione dei dati con diverso seed random

scikit-learn: Addestramento Modello e Validazione con Hold-Out

- scikit-learn fornisce una funzione `train_test_split` per partizionare casualmente un set di dati in due set disgiunti
  - vanno specificati la matrice degli input **X** e il vettore output attesi **y**
  - si può specificare un numero esatto o una proporzione dei dati da inserire in uno dei due set (l'altro è calcolato di conseguenza)
  - si può specificare un seed con `random_state`
  - viene restituita una tupla composta da: matrice input training, matrice input validation, vettore output training, vettore output validation
- Ad es. per partizionare un set di dati riservandone il 70% al training set e il restante 30% al validation set:

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_val, y_train, y_val = \
      train_test_split(X, y, train_size=0.7)
```

Esempio: Validazione del Modello di Predizione del Consumo di Elettricità

- Dai vettori `temp` e `peak` con i dati storici, creiamo training e validation set con 70% e 30% dei dati

```
X_train, X_val, y_train, y_val = \
    train_test_split(temp[:, None], peak,
    test_size=0.3, random_state=42)
```

- Addestriamo un modello sul solo training set

```
lrm = LinearRegression()
lrm.fit(X_train, y_train)
```

- Valutiamo l'errore quadratico medio su entrambi i dataset

```
>>> np.mean((lrm.predict(X_train) - y_train)**2)
0.022915822490891O7
>>> np.mean((lrm.predict(X_val) - y_val)**2)
0.019989933582418993
```


Errore Relativo: Mean Absolute Percentage Error (MAPE)

- L'errore quadratico medio è funzionale all'addestramento del modello di regressione tramite discesa gradiente
- Possiamo in aggiunta valutare l'errore secondo altri criteri
- Ad esempio l'*errore relativo* tra un valore *y* reale e la stima y è definito da abs((*ŷ* – *y*) / *y*), *perciò il MAPE* = $\frac{1}{m}\sum_1^m \left|\frac{(\hat{y}i - yi)}{yi}\right|$
- Calcoliamo ad esempio l'errore relativo medio sia su training set che validation set del modello di previsione dei consumi

```
np.mean(np.abs((lrm.predict(X_train) - y_train)/ y_train))
0.05899891472169821
np.mean(np.abs((lrm.predict(X_val) - y_val) / y_val))
0.05170223216469662
```

- Gli errori relativi sui due dataset sono 5,9% e 5,1% circa
- Altre misure di errore SMAPE, WAPE, WMAPE etc.

Coefficiente di Determinazione R²

- indica la proporzione di variazione della variabile dipendente che è predicibile dalle variabili indipendenti
  - $\hat{y}$ previsione; $\bar{y}$ valore medio y; y_i valore y dell'istanza i-esima
- Il suo valore è compreso tra 0 e 1
  - 1 indica che il modello descrive perfettamente i dati
  - 0 indica che non c'è alcuna relazione tra modello e dati
  - valori intermedi indicano diversi gradi di efficacia del modello

$$R^2 = \frac{\sum_{i=1}^{m}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{m}(y_i - \bar{y})^2}$$

- In scikit-learn, dato un modello di regressione addestrato, può essere calcolato su un set di dati col metodo `score`
- Ad es., sempre sul modello di predizione dei consumi si ha:

```
>>> lrm.score(X_train, y_train)
0.7689433269368844
>>> lrm.score(X_val, y_val)
0.7508669565801295
```


Coefficiente di Determinazione R² aggiustato

- R² tende a crescere all'aumentare del numero di variabili dipendenti, indipendentemente dalla bontà del modello
- Sono state definite correzioni e la più usata è la seguente

$$\bar{R}^2 = 1 - (1 - R^2)\frac{n-1}{n-p}$$

- dove p è il numero di variabili indipendenti (di input) ed n la quantità di dati, R² è il coefficiente di determinazione
- $\bar{R}^2$ può essere negativo ed è sempre minore uguale ad R²

Intervallo di Confidenza del Coefficiente di Determinazione R²

- nella regressione la formula per l'intervallo di confidenza (CI) del coefficiente R^2 ha la stessa forma di quella presentata a lezione:
- CI = R^2 -+ t x sqrt(VR) # VR = varianza
- dove t (con distribuzione Student) è dato dal livello di confidenza e dal numero di istanze, e.g. con confidenza del 95% ed almeno 500 istanze di test esso vale
- 1.965
- all'aumentare del numero di istanze la distribuzione t di student tende alla distribuzione normale standard, come abbiamo visto anche per la binomiale a lezione; il valore esatto di t si ottiene con
- stats.t.ppf(1 - 0.025, 500) # alpha 0.05 va dimezzato perché è a due code
- ed sqrt(VR) è l'errore standard, i.e. deviazione standard
- VR = (4R^2 x (1-R^2)^2 x (n-k-1)^2)/((n^2 - 1)(n+3))
- con n il num. di istanze del test set e k il numero di variabili di input.

Intervallo di Confidenza della differenza tra due Coefficienti di Determinazione R²

3.6.3 Confidence Intervals for Differences Between Independent R^2s

For our running example of 62 cases (University V), we found the $R^2_V = .5032$ for the $k = 4$ IVs. For the same IVs in University W, where $n = 143$, assume that $R^2_W = .2108$. The difference is $.5032 - .2108 = .2924$. Since these are different groups that were independently sampled, we can find CIs and perform null hypothesis significance tests on this difference, using the SE of the difference. As we have seen for other statistics, this is simply the square root of the sum of the SE^2s of the two R^2s. We found the SE^2 for V to be .006461 in the previous section, and assume we find the SE^2 for W to be .003350. Substituting,

$$SE_{R^2_V - R^2_W} = \sqrt{SE_{R^2_V} + SE_{R^2_W}} \tag{3.6.4}$$

$$= \sqrt{.006461 + .003350} = \sqrt{.006811} = .0825.$$

The approximate 95% $me = 2(.0825) = .1650$, so the approximate 95% CI for a *nil* hypothesis significance test $= .2924 \pm .1650$, from .13 to .46. Since the 95% CI does not include 0, the difference between the universities' R^2s is significant at the $\alpha = .05$ level.

Stima dei Prezzi delle Case: Misure di Valutazione

- Effettuiamo la stessa divisione 70-30 dei dati sulle case

```
X_train, X_val, y_train, y_val = \
  train_test_split(X, y, test_size=0.3, random_state=42)
lrm = LinearRegression()
lrm.fit(X_train, y_train)
```

- Usando lo stesso codice visto sopra per l'esempio del consumo di elettricità, otteniamo queste valutazioni

	training set	validation set
errore quadrato medio	22,5455	21,5174
errore relativo medio	16,57%	16,52%
coefficiente R^2	0,7435	0,7112

Caso di Studio 3: Classificazione di Opinioni Positive e Negative

- Le persone pubblicano comunemente sul Web opinioni riguardo a prodotti, ristoranti, hotel, etc.
- In alcuni casi (es. recensioni Amazon) l'utente può associare al testo un voto complessivo (es. da 1 a 5 stelle)
- Dalle opinioni di cui c'è solo il testo (es. da un forum), vorremmo stimare automaticamente il voto
  - per capire la popolarità di un prodotto date migliaia di opinioni
- Esistono diversi metodi di *sentiment analysis* per capire se un testo esprima un'opinione positiva o negativa
- Qui vediamo come etichettare recensioni tramite un semplice modello di regressione basato sul conteggio di parole chiave

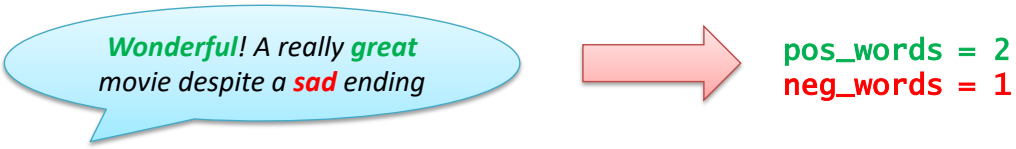
Classificazione di Opinioni: Dizionari di Parole Positive e Negative

- Dal Web possiamo reperire liste compilate manualmente di parole con connotazione positiva o negativa

positive words	
great	superior
wonderful	illuminating
awesome	...

negative words	
horrible	nightmare
useless	terrifying
sad	...

- Dato il testo di una recensione, possiamo contare quante parole di ciascuna delle liste sono presenti
- Otteniamo quindi conteggi di parole positive e negative come variabili che caratterizzano ciascuna recensione



Classificazione di Opinioni: Definizione del Modello di Regressione

- Possiamo addestrare un modello di regressione su queste due variabili a stimare il voto a cui corrisponde un'opinione
- Avendo un dataset con testi e voti delle recensioni …

oid	text	stars
527	Christopher Reeve is the definitive screen "Superman" in my opinion. And this is the edition of ...	5.0
540	Sorry, never watched the movie. If it has ANYTHING to do wit that book, it's an abomination. Psy...	2.0
577	I won't bother going over the plot, as others have done so very ably before me. Actually this is...	4.0

- … possiamo associare a ciascuna i conteggi delle parole positive e negative e usarle così per addestrare il modello

oid	text	stars	pos_words	neg_words
527	Christopher Reeve is the definitive screen "Superman" in my opinion. And this is the edition of ...	5.0	10.0	3.0
540	Sorry, never watched the movie. If it has ANYTHING to do wit that book, it's an abomination. Psy...	2.0	1.0	3.0
577	I won't bother going over the plot, as others have done so very ably before me. Actually this is...	4.0	41.0	33.0

Classificazione di Opinioni: Addestramento del Modello di Regressione

- Dividiamo i dati 70-30 in training e validation come sopra
  - come matrice **X** selezioniamo i conteggi di parole
  - come vettore **y** i numeri di stelle

```
X_train, X_val, y_train, y_val = train_test_split(
    reviews[["pos_words", "neg_words"]],
    reviews["stars"], test_size=0.3, random_seed=42)
```

- Addestriamo il modello di regressione sul training set

```
lrm = LinearRegression()
lrm.fit(X_train, y_train)
```


Classificazione di Opinioni: Lettura del Modello di Regressione

- Osservando i parametri del modello addestrato, possiamo capire come il modello predica il voto …
  - la capacità di un modello di essere interpretabile è nota come explainability
  - i metodi avanzati di learning come le reti neurali non offrono questa capacità che invece è determinante per problemi critici come diagnosi di malattie, auto a guida autonoma, robotica, controllo industriale

```
lrm.coef_, lrm.intercept_
(array([ 0.03439177, -0.04167603]), 3.99541639633)
```

  - partendo da un voto base di circa 4 stelle (bias, i.e. intercetta)
  - aggiunge circa 0,035 stelle per ogni parola positiva
  - e sottrae circa 0,042 stelle per ogni parola negativa

Accuratezza della Classificazione di Opinioni

- Un modello di regressione può essere usato come *classificatore* in un numero finito di classi
- Si considerino ad esempio *positive* le recensioni con almeno 3.5 stelle e *negative* quelle con un numero inferiore

```
labels_test = np.where(y_test >= 3.5, "pos", "neg")
labels_pred = np.where(
  lrm.predict(X_test) >= 3.5, "pos", "neg")
```

- Possiamo valutare in quanti casi il modello di regressione riesce a prevedere la classe corretta nel validation set

```
(labels_test == labels_pred).mean()
0.736533333333334
```

- Il 73,65% circa delle recensioni è classificata correttamente
  - senza recensioni neutre, ossia con differenza tra parole positive e negative nell'intervallo [-2, 2] l'accuratezza sale al **76%**