Università di Catania

UNIVERSITY OF CATANIA

Department of Mathematics and Computer Science

Alfio Spoto

Numerical Analysis of Linear Regression: Comparative Study of Closed-Form and Iterative Method

Final Project Report

Professor: Sebastiano Boscarino

Academic Year 2024 - 2025

Contents

1 Introduzione **3**
- 1.1 Background e Motivazione . . . 3
- 1.2 Obiettivi dello Studio . . . 3
- 1.3 Struttura dell'Elaborato . . . 4

2 Background Teorico **5**
- 2.1 Modello di Regressione Lineare . . . 5
- 2.2 Funzione di Costo . . . 6
- 2.3 Soluzioni in Forma Chiusa . . . 6
  - 2.3.1 Equazioni Normali (Minimi Quadrati) . . . 6
  - 2.3.2 Decomposizione ai Valori Singolari (SVD) . . . 7
  - 2.3.3 Decomposizione QR . . . 7
- 2.4 Metodi Iterativi . . . 8
  - 2.4.1 Gradiente Coniugato . . . 8
  - 2.4.2 Adam Optimization . . . 9
- 2.5 Confronto Teorico tra Metodi . . . 9
- 2.6 Il Problema del Mal Condizionamento e della Multicollinearità . . . 10
  - 2.6.1 Definizione e Cause . . . 10
  - 2.6.2 Conseguenze della Multicollinearità . . . 11
  - 2.6.3 Tecniche di Regolarizzazione . . . 11
    - 2.6.3.1 Ridge Regression (Regolarizzazione L2) . . . 11
    - 2.6.3.2 Lasso Regression (L1 Regularization) . . . 12

3 Applicazione Pratica e Risultati Sperimentali **13**
- 3.1 Dataset e Preprocessing . . . 13
  - 3.1.1 Descrizione del Dataset . . . 13
  - 3.1.2 Exploratory Data Analysis . . . 14
  - 3.1.3 Data Preprocessing . . . 15
- 3.2 Implementazione dei Metodi . . . 16
  - 3.2.1 Implementazione dei Direct Methods . . . 16
    - 3.2.1.1 Least Squares Regressor . . . 16
    - 3.2.1.2 SVD Regressor . . . 16
    - 3.2.1.3 QR Regressor . . . 17
  - 3.2.2 Implementazione degli Iterative Methods . . . 17
    - 3.2.2.1 Conjugate Gradient Regressor . . . 17
    - 3.2.2.2 Adam Regressor . . . 18
- 3.3 Risultati e Analisi . . . 19

3.3.1 Evaluation Metrics 19
3.3.2 Performance Comparison 20
3.3.3 Analisi di convergenza dei metodi iterativi 22
3.3.4 Inference Times 23
3.4 Linear vs. Polynomial Model 24
3.4.1 Implementazione del Polynomial Model 24
3.4.2 Performance Comparison 24
3.4.3 Interpretazione dei risultati 24

Conclusion **26**

Bibliography **28**

Chapter 1

Introduzione

1.1 Background e Motivazione

La regressione lineare rappresenta una delle tecniche fondamentali nell'analisi statistica e nel machine learning, costituendo frequentemente il paradigma iniziale per la comprensione di relazioni complesse tra variabili [1]. Nonostante la sua apparente semplicità concettuale, l'implementazione efficiente e numericamente stabile di algoritmi di regressione lineare presenta sfide computazionali non triviali, particolarmente in contesti caratterizzati da dataset ad alta dimensionalità o problemi mal condizionati [2, 3].

Il presente studio esplora e confronta diversi metodi numerici per la risoluzione di problemi di regressione lineare, con particolare enfasi sulle loro fondamenta matematiche, proprietà computazionali e performance empiriche. L'obiettivo è fornire un'analisi approfondita delle similitudini e differenze tra approcci in forma chiusa (closed-form) e metodi iterativi [4, 5], valutandone il comportamento su un dataset reale di indicatori sanitari e socioeconomici globali [6].

1.2 Obiettivi dello Studio

Gli obiettivi principali di questo lavoro sono:

- Analizzare e confrontare i fondamenti matematici di diversi metodi per la risoluzione di sistemi lineari applicati alla regressione, con particolare attenzione alla stabilità numerica e alla complessità computazionale [7, 2].
- Implementare e valutare empiricamente sia metodi diretti (Least Squares, Singular Value Decomposition, QR Decomposition) che iterativi (Conjugate Gradient [8], Adam optimization [9]) su un dataset reale.
- Esaminare il comportamento di convergenza dei metodi iterativi e le loro caratteristiche di performance in termini di accuracy, computational efficiency e numerical stability [10, 11].
- Affrontare problematiche pratiche come l'ill-conditioning e la multicollinearity che emergono frequentemente nell'applicazione di modelli di regressione a dati reali [12, 1].

- Estendere l'analisi alla polynomial regression per catturare relazioni non lineari presenti nel dataset, confrontandone le performance rispetto agli approcci lineari standard [1].

1.3 Struttura dell'Elaborato

L'elaborato è organizzato secondo la seguente struttura:

Nel Capitolo 2, viene presentato il background teorico della regressione lineare e vengono descritti in dettaglio i cinque metodi analizzati: tre soluzioni in forma chiusa (Least Squares, SVD [2], QR [7]) e due metodi iterativi (Conjugate Gradient [8, 13], Adam [9]). Particolare attenzione è dedicata alle proprietà matematiche che determinano l'efficienza computazionale e la stabilità numerica di ciascun approccio [3, 2].

Il Capitolo 3 è dedicato all'applicazione pratica, dove viene descritto il dataset utilizzato [6], i risultati sperimentali e un'analisi approfondita delle performance dei vari metodi. Viene inoltre trattato il problema della multicollinearity [12] e vengono confrontati i modelli di regressione lineare e polinomiale in termini di capacità predittive e adeguatezza al dataset considerato [1].

Infine, nelle Conclusioni, vengono riassunti i principali risultati, discusse le implicazioni pratiche e suggerite direzioni per ricerche future nella computational linear algebra applicata ai problemi di regressione.

Questo studio si propone di fornire sia una comprensione teorica rigorosa che insights pratici per ricercatori e professionisti che operano con modelli di regressione in vari domini applicativi, dalla statistica computazionale al machine learning [1], dall'econometria alla health informatics.

Chapter 2

Background Teorico

2.1 Modello di Regressione Lineare

La regressione lineare è una tecnica statistica che modella la relazione tra una variabile dipendente (o target) y e una o più variabili indipendenti (o predittori) x attraverso una funzione lineare [1]. Data una collezione di m esempi di training con n caratteristiche, il modello di regressione lineare può essere espresso come:

$$h_\theta(x_i) = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \cdots + \theta_n x_{in} \tag{2.1}$$

Dove:

- $h_\theta(x_i)$ è il valore previsto per l'i-esimo esempio
- θ_j sono i parametri del modello (pesi)
- θ_0 è l'intercetta (o bias)
- x_{ij} è il valore della j-esima caratteristica per l'i-esimo esempio

In notazione matriciale, possiamo esprimere questo in modo più compatto come:

$$h_\theta(X) = \theta^T X \tag{2.2}$$

Dove $X \in \mathbb{R}^{m\times(n+1)}$ è la matrice di design con una colonna aggiuntiva di uni per il termine di intercetta:

$$X = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1n} \\ 1 & x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & x_{m2} & \cdots & x_{mn} \end{pmatrix} \tag{2.3}$$

E $\theta \in \mathbb{R}^{(n+1)\times 1}$ è il vettore dei parametri $[\theta_0, \theta_1, \ldots, \theta_n]^T$.

2.2 Funzione di Costo

L'obiettivo della regressione lineare è trovare i valori ottimali di θ che minimizzano l'errore tra le previsioni del modello e i valori reali [5]. La funzione di costo più comunemente utilizzata è l'Errore Quadratico Medio (MSE):

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 \tag{2.4}$$

In forma matriciale, questa può essere espressa come:

$$J(\theta) = \frac{1}{m}(\theta^T X - y)^T(\theta^T X - y) \tag{2.5}$$

Questa funzione di costo è convessa e differenziabile, quindi possiamo trovare il minimo globale impostando il suo gradiente a zero [5]:

$$\nabla_\theta J(\theta) = \frac{2}{m}X^T(\theta^T X - y) = 0 \tag{2.6}$$

Questo ci porta alle equazioni normali:

$$X^T\theta^T X = X^T y \tag{2.7}$$

Diversi metodi numerici possono essere utilizzati per risolvere questo sistema di equazioni lineari, come vedremo nelle sezioni seguenti [2, 3].

2.3 Soluzioni in Forma Chiusa

Le soluzioni in forma chiusa trovano direttamente il valore ottimale dei parametri attraverso operazioni algebriche, senza richiedere un processo iterativo di ottimizzazione [3].

2.3.1 Equazioni Normali (Minimi Quadrati)

L'approccio più diretto per risolvere le equazioni normali è calcolare [1, 2]:

$$\theta = (X^T X)^{-1}X^T y \tag{2.8}$$

Questa formula presuppone che la matrice $X^T X$ sia invertibile, il che è vero quando le colonne di X sono linearmente indipendenti. In pratica, il calcolo diretto dell'inversa è spesso evitato a favore di metodi più numericamente stabili [7, 2].

2.3.2 Decomposizione ai Valori Singolari (SVD)

La decomposizione ai valori singolari (SVD) decompone la matrice di design X nel prodotto di tre matrici [2, 7]:

$$X = U\Sigma V^T \tag{2.9}$$

Dove:

- $U \in \mathbb{R}^{m\times m}$ è una matrice ortogonale contenente i vettori singolari sinistri
- $\Sigma \in \mathbb{R}^{m\times(n+1)}$ è una matrice diagonale contenente i valori singolari non negativi in ordine decrescente
- $V \in \mathbb{R}^{(n+1)\times(n+1)}$ è una matrice ortogonale contenente i vettori singolari destri

La soluzione utilizzando SVD è [2]:

$$\theta = V\Sigma^+ U^T y \tag{2.10}$$

Dove Σ^+ è la pseudoinversa di Σ, ottenuta prendendo il reciproco di ogni valore singolare non nullo e trasponendo la matrice risultante.

La SVD è particolarmente utile quando $X^T X$ è mal condizionata o singolare, poiché permette di identificare e gestire i piccoli valori singolari che potrebbero causare instabilità numerica [2, 3, 14].

2.3.3 Decomposizione QR

La decomposizione QR esprime la matrice di design X come il prodotto di due matrici [7, 2]:

$$X = QR \tag{2.11}$$

Dove:

- $Q \in \mathbb{R}^{m\times m}$ è una matrice ortogonale ($Q^T Q = I$)
- $R \in \mathbb{R}^{m\times(n+1)}$ è una matrice triangolare superiore

Sostituendo questa decomposizione nelle equazioni normali, otteniamo [7]:

$$X^T\theta^T X = X^T y \tag{2.12}$$
$$R^T Q^T Q R\theta = R^T Q^T y \tag{2.13}$$
$$R^T R\theta = R^T Q^T y \quad (\text{poiché } Q^T Q = I) \tag{2.14}$$

Assumendo che R abbia rango pieno, possiamo semplificare ulteriormente:

$$\theta = R^{-1} Q^T y \tag{2.15}$$

In pratica, invece di calcolare esplicitamente R^{-1}, si risolve il sistema triangolare mediante sostituzione all'indietro, che è computazionalmente più efficiente e numericamente più stabile [2, 3].

2.4 Metodi Iterativi

I metodi iterativi approssimano progressivamente la soluzione ottimale attraverso una serie di aggiornamenti dei parametri, partendo da un'inizializzazione casuale e migliorando incrementalmente [4, 5].

2.4.1 Gradiente Coniugato

Il metodo del Gradiente Coniugato è un algoritmo iterativo per risolvere sistemi di equazioni lineari con matrici simmetriche e definite positive [8, 13]. Per la regressione lineare, lo applichiamo alle equazioni normali $X^T\theta^T X = X^T y$.

L'algoritmo procede generando una sequenza di direzioni di ricerca $\{p_k\}$ che sono coniugate rispetto a $X^T X$, ovvero $p_i^T X^T X p_j = 0$ per $i \neq j$. Questo permette una convergenza più rapida rispetto al semplice gradiente discendente, poiché evita di "zigzagare" nelle stesse direzioni [13, 11].

I passaggi principali dell'algoritmo sono:

Algorithm 1 Gradiente Coniugato per Regressione Lineare [8, 13]

Input: Matrice di design X, vettore target y, tolleranza ϵ
Output: Parametri ottimali θ

Inizializza θ_0 arbitrariamente (tipicamente a zero)
Calcola il residuo iniziale $r_0 = X^T y - X^T\theta^T X_0$
Imposta $p_0 = r_0$
for $k = 0, 1, 2, \ldots$ fino alla convergenza **do**
    $\alpha_k = \frac{r_k^T r_k}{p_k^T X^T X p_k}$
    $\theta_{k+1} = \theta_k + \alpha_k p_k$
    $r_{k+1} = r_k - \alpha_k X^T X p_k$
    if $\|r_{k+1}\| < \epsilon$ **then**
        return θ_{k+1}
    end if
    $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
    $p_{k+1} = r_{k+1} + \beta_k p_k$
end for

La bellezza del Gradiente Coniugato è che, per un problema in dimensione n, converge teoricamente in al più n iterazioni in aritmetica esatta [8, 11]. In pratica, gli errori di arrotondamento possono richiedere più iterazioni, ma la convergenza è comunque molto rapida per problemi ben condizionati [13].

2.4.2 Adam Optimization

Adam (Adaptive Moment Estimation) è un algoritmo di ottimizzazione basato sul gradiente che combina i vantaggi di due altre estensioni della discesa stocastica del gradiente: AdaGrad e RMSProp [9]. È particolarmente popolare nell'addestramento di reti neurali ma può essere applicato anche alla regressione lineare [10].

Adam mantiene stime del primo momento (la media) e del secondo momento (la varianza non centrata) dei gradienti, che vengono aggiornate esponenzialmente [9]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.16}$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.17}$$

Dove $g_t = \nabla_\theta J(\theta_t)$ è il gradiente della funzione di costo al passo t, e $\beta_1, \beta_2 \in [0, 1)$ sono tassi di decadimento.

Poiché m_t e v_t sono inizializzati a zero, vengono applicati fattori di correzione del bias [9]:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.18}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.19}$$

Infine, i parametri vengono aggiornati come segue:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{2.20}$$

Dove η è il tasso di apprendimento e ϵ è un piccolo valore per evitare la divisione per zero.

A differenza del Gradiente Coniugato, Adam non garantisce la convergenza in un numero finito di passi, ma la sua capacità di adattare il tasso di apprendimento per ciascun parametro lo rende robusto a diverse scale di caratteristiche e configurazioni di iperparametri [9, 10].

2.5 Confronto Teorico tra Metodi

Dal punto di vista teorico, esiste una distinzione fondamentale tra i metodi in forma chiusa e i metodi iterativi [4, 5]:

Algorithm 2 Adam per Regressione Lineare [9, 10]

Input: Matrice di design X, vettore target y, tasso di apprendimento η, tassi di decadimento β_1, β_2, ϵ, numero di epoche
Output: Parametri ottimali θ

Inizializza θ arbitrariamente
Inizializza $m_0 = 0$, $v_0 = 0$, $t = 0$
for epoca = 1, 2, ..., numero di epoche **do**
    $t = t + 1$
    $g_t = \frac{2}{m} X^T(\theta^T X_{t-1} - y)$ ▷ Gradiente della funzione MSE
    $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
    $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$
    $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$
    $\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon} \hat{m}_t$
end for
return θ_t

Table 2.1: Confronto Teorico tra Metodi Diretti e Iterativi [7, 4, 5]

Caratteristica	**Metodi Diretti**	**Metodi Iterativi**
Convergenza	Esatta in un solo passo di calcolo	Approssimativa, migliora con il numero di iterazioni
Complessità computazionale	$O(n^3)$ per matrici $n \times n$	$O(kn^2)$ dove k è il numero di iterazioni
Stabilità numerica	Può soffrire di problemi in presenza di multicollinearità o mal condizionamento	Può essere più robusto con appropriate tecniche di regolarizzazione o precondizionamento
Scalabilità	Inefficiente per dataset di grandi dimensioni	Più adatto per problemi su larga scala
Iperparametri	Nessuno	Richiede calibrazione (es. tasso di apprendimento, tolleranza)
Applicabilità	Specifico per problemi lineari	Facilmente estendibile a problemi non lineari

2.6 Il Problema del Mal Condizionamento e della Multicollinearità

Un aspetto cruciale nell'applicazione dei metodi di regressione lineare, specialmente quelli in forma chiusa, è il problema del mal condizionamento e della multicollinearità [12, 1].

2.6.1 Definizione e Cause

La multicollinearità si verifica quando due o più variabili predittive nel modello sono fortemente correlate tra loro. Dal punto di vista matematico, questo significa che alcune

colonne della matrice di design X sono quasi linearmente dipendenti, rendendo la matrice $X^T X$ quasi singolare, con un determinante vicino a zero [2, 3].

Il numero di condizionamento di una matrice, definito come il rapporto tra il suo valore singolare più grande e quello più piccolo, quantifica questo problema [7, 2]:

$$\kappa(X^T X) = \frac{\sigma_{\max}(X^T X)}{\sigma_{\min}(X^T X)} = \frac{\sigma^2_{\max}(X)}{\sigma^2_{\min}(X)} \tag{2.21}$$

Un alto numero di condizionamento indica che la matrice è mal condizionata, il che può portare a:

- Amplificazione degli errori di arrotondamento nelle soluzioni numeriche
- Instabilità nelle stime dei parametri
- Alta varianza nei coefficienti stimati

Nella regressione lineare, la multicollinearità può sorgere per vari motivi [1]:

- Variabili intrinseche correlate (es. altezza e peso)
- Ridondanza nelle misurazioni (es. temperatura in Celsius e Fahrenheit)
- Piccole dimensioni del campione rispetto al numero di predittori
- Inclusione di termini di interazione o polinomiali correlati alle variabili originali

2.6.2 Conseguenze della Multicollinearità

Le principali conseguenze della multicollinearità includono [12, 1]:

- **Coefficienti instabili**: Piccoli cambiamenti nei dati possono causare grandi variazioni nei coefficienti stimati.
- **Errori standard elevati**: I coefficienti vengono stimati con intervalli di confidenza molto ampi.
- **Interpretazione fuorviante**: Diventa difficile determinare l'importanza relativa delle variabili predittive.
- **Overfitting**: Il modello può adattarsi eccessivamente ai dati di training, mostrando una scarsa capacità di generalizzazione.

2.6.3 Tecniche di Regolarizzazione

Per affrontare il problema della multicollinearità, le tecniche di regolarizzazione aggiungono un termine di penalizzazione alla funzione di costo, limitando la magnitudo dei coefficienti [1]:

2.6.3.1 Ridge Regression (Regolarizzazione L2)

La regressione Ridge modifica la funzione di costo MSE aggiungendo un termine di penalità proporzionale alla somma dei quadrati dei coefficienti [12]:

$$J_{\text{Ridge}}(\theta) = \frac{1}{m}\|\theta^T X - y\|^2 + \lambda\|\theta\|_2^2 \tag{2.22}$$

Dove $\|\theta\|_2^2 = \sum_{j=1}^{n} \theta_j^2$ è la norma L2 al quadrato del vettore dei parametri (escludendo l'intercetta) e $\lambda \geq 0$ è il parametro di regolarizzazione.

La soluzione in forma chiusa diventa [12, 1]:

$$\theta_{\text{Ridge}} = (X^T X + \lambda I)^{-1} X^T y \tag{2.23}$$

Dove I è la matrice identità (con l'elemento corrispondente all'intercetta impostato a zero).

L'aggiunta di λI a $X^T X$ migliora il condizionamento della matrice, stabilizzando la soluzione. Un valore maggiore di λ porta a una maggiore shrinkage dei coefficienti verso zero, riducendo la variance ma potenzialmente aumentando il bias [1].

2.6.3.2 Lasso Regression (L1 Regularization)

La Lasso Regression (Least Absolute Shrinkage and Selection Operator) utilizza la L1 norm anziché la L2 norm [15, 1]:

$$J_{\text{Lasso}}(\boldsymbol{\theta}) = \frac{1}{m}\|\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\|^2 + \lambda\|\boldsymbol{\theta}\|_1 \tag{2.24}$$

Dove $\|\boldsymbol{\theta}\|_1 = \sum_{j=1}^{n} |\theta_j|$ è la L1 norm del vettore dei parametri.

A differenza della Ridge Regression, Lasso non possiede una soluzione in forma chiusa e richiede metodi di ottimizzazione numerica [15]. La caratteristica distintiva di Lasso è la sua capacità di indurre sparsità, portando alcuni coefficienti esattamente a zero e operando così una feature selection automatica [15, 1].

Chapter 3

Applicazione Pratica e Risultati Sperimentali

3.1 Dataset e Preprocessing

3.1.1 Descrizione del Dataset

Nel presente studio, viene utilizzato il dataset "Life Expectancy Data" fornito dall'Organizzazione Mondiale della Sanità (WHO) [6], che contiene informazioni su vari indicatori sanitari, socioeconomici e demografici per diversi paesi nel periodo 2000-2015. Il dataset comprende 2938 osservazioni, ciascuna rappresentante una combinazione paese-anno, con 22 variabili.

La variabile target è la life expectancy at birth, espressa in anni. Le principali variabili predittive includono:

Table 3.1: Principali Variabili nel Dataset sull'Aspettativa di Vita [6]

Categoria	**Variabile**	**Descrizione**
Demografiche	Year	Anno dell'osservazione
	Population	Dimensione della popolazione
	Schooling	Anni medi di scolarizzazione (adulti 25+)
	Income_composition	Human Development Index in relazione al reddito
Health Indicators	Adult Mortality	Tasso di mortalità adulta (per 1000 abitanti)
	Infant deaths	Numero di morti infantili
	Hepatitis B	Copertura vaccinale contro l'Epatite B (%)
	Measles	Numero di casi di morbillo riportati
	BMI	Body Mass Index medio
	HIV/AIDS	Morti per HIV/AIDS (per 1000 nati vivi)
Economic Factors	GDP	Gross Domestic Product pro capite
	Percentage expenditure	Health expenditure come percentuale del GDP
	Total expenditure	Government expenditure on health (% del budget totale)

3.1.2 Exploratory Data Analysis

Prima dell'applicazione dei metodi di regressione, è stata condotta un'analisi esplorativa per comprendere la distribuzione delle variabili e le loro relazioni [1].

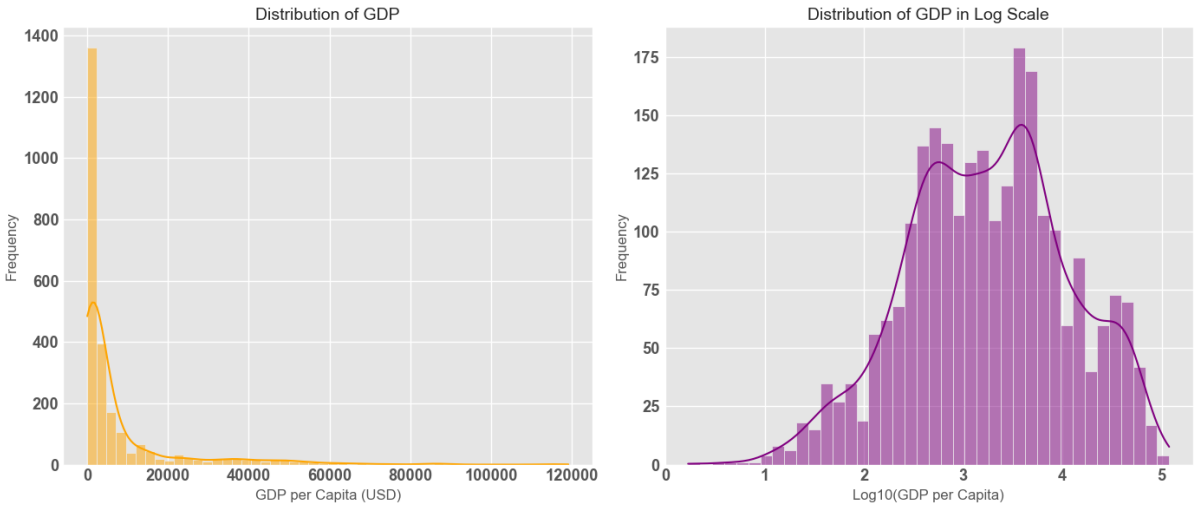

Figure 3.1: Distribuzione del GDP pro Capite (Sinistra: Valori grezzi, Destra: Scala logaritmica)

La Figura 3.1 illustra la distribuzione del GDP pro capite nel dataset. Nel grafico a sinistra, si osserva una marcata positive skewness, con la maggior parte dei paesi che presentano valori di GDP relativamente bassi e pochi paesi con valori significativamente elevati. Quando la stessa distribuzione viene visualizzata su scala logaritmica (grafico a destra), il GDP mostra una distribuzione più prossima alla normalità, suggerendo che una log-transformation potrebbe essere appropriata quando si utilizza il GDP come predittore nei modelli di regressione [1].

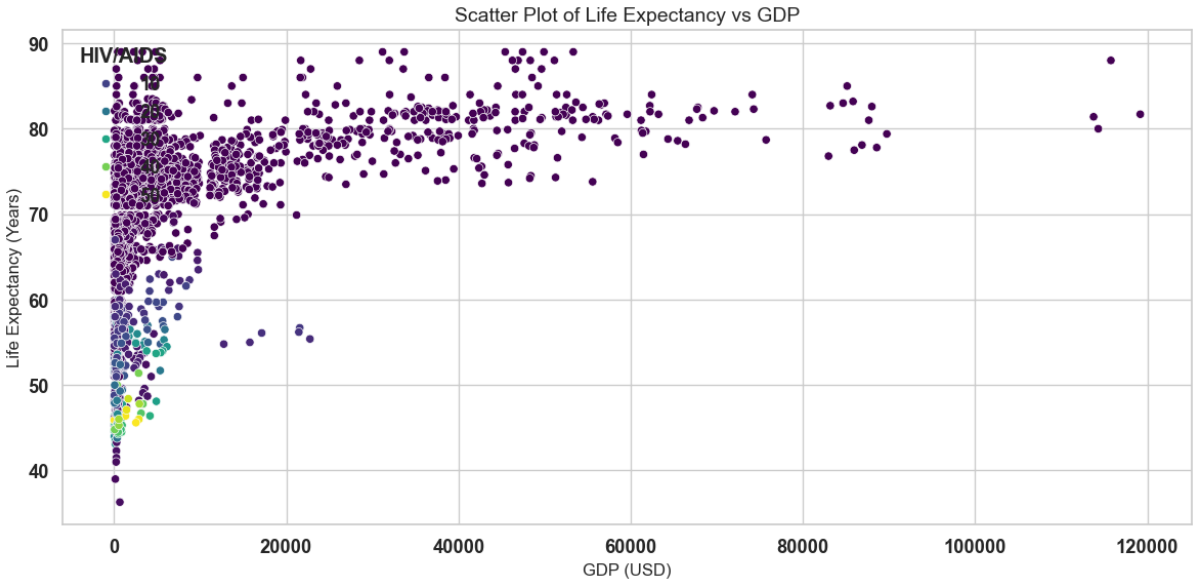

Figure 3.2: Scatter Plot della Life Expectancy vs GDP, Colorato per Prevalenza di HIV/AIDS

La Figura 3.2 visualizza la relazione tra life expectancy e GDP pro capite, con i punti colorati in base alla prevalenza di HIV/AIDS. Diversi pattern sono evidenti:

- Si osserva una chiara correlazione positiva tra GDP e life expectancy, che segue una relazione di tipo logaritmico [1].
- La relazione risulta più pronunciata a livelli di GDP inferiori ma tende a plateau a livelli superiori, suggerendo diminishing returns.
- I paesi con elevata prevalenza di HIV/AIDS (indicati da tonalità più calde) tendono a presentare una life expectancy inferiore indipendentemente dal GDP.
- È presente una significativa variabilità nella life expectancy tra paesi con livelli di GDP comparabili, indicando l'influenza significativa di altri fattori.

Questa relazione non lineare tra GDP e life expectancy suggerisce che una trasformazione della variabile GDP o un modello di regressione polinomiale potrebbero risultare più appropriati rispetto a un semplice modello lineare [1].

3.1.3 Data Preprocessing

Prima dell'applicazione dei metodi di regressione, sono stati implementati i seguenti passaggi di preprocessing [1]:

- **Missing value imputation**: I valori mancanti sono stati imputati utilizzando la mediana per le variabili numeriche, stratificando per paese ove possibile.
- **Feature transformation**: Basandosi sull'analisi esplorativa, è stata applicata una log-transformation al GDP e ad altre variabili con distribuzione fortemente asimmetrica.
- **Feature scaling**: Tutte le variabili predittive sono state standardizzate sottraendo la media e dividendo per la deviazione standard, per garantire scale comparabili e migliorare la stabilità numerica degli algoritmi [2, 3].
- **Train-test split**: Il dataset è stato suddiviso in un training set (80%) e un test set (20%), preservando la distribuzione temporale delle osservazioni.

3.2 Implementazione dei Metodi

Tutti e cinque i metodi discussi nel Capitolo 2 sono stati implementati utilizzando Python con la libreria NumPy per le operazioni di linear algebra. Di seguito, viene fornita una descrizione dell'implementazione di ciascun metodo.

3.2.1 Implementazione dei Direct Methods

3.2.1.1 Least Squares Regressor

Il Least Squares Regressor risolve le normal equations direttamente [2]:

```
class LeastSquaresRegressor:
    def fit(self, X, y):
        # Add a column of ones for the intercept
        X_b = np.c_[np.ones((X.shape[0], 1)), X]
        # Solve the normal equations
        self.theta = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
        return self

    def predict(self, X):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]
        return X_b.dot(self.theta)
```

3.2.1.2 SVD Regressor

L'SVD Regressor utilizza la Singular Value Decomposition [2, 7]:

```
class SVDRegressor:
    def fit(self, X, y):
        # Add a column of ones for the intercept
        X_b = np.c_[np.ones((X.shape[0], 1)), X]
        # Compute the SVD
        U, s, Vt = np.linalg.svd(X_b, full_matrices=False)
        # Compute the pseudoinverse of Sigma
        S_inv = np.diag(1.0 / s)
        # Compute the parameters
        self.theta = Vt.T.dot(S_inv).dot(U.T).dot(y)
        return self

    def predict(self, X):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]
        return X_b.dot(self.theta)
```

3.2.1.3 QR Regressor

Il QR Regressor utilizza la QR Decomposition [7, 2]:

```
class QRRegressor:
    def fit(self, X, y):
        # Add a column of ones for the intercept
        X_b = np.c_[np.ones((X.shape[0], 1)), X]
        # Compute the QR decomposition
        Q, R = np.linalg.qr(X_b)
        # Compute the parameters
        self.theta = np.linalg.solve(R, Q.T.dot(y))
        return self

    def predict(self, X):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]
        return X_b.dot(self.theta)
```

3.2.2 Implementazione degli Iterative Methods

3.2.2.1 Conjugate Gradient Regressor

Il Conjugate Gradient Regressor implementa l'algoritmo iterativo descritto nel Capitolo 2 [8, 13]:

```
class ConjugateGradientRegressor:
    def __init__(self, tol=1e-5, max_iter=1000):
        self.tol = tol
        self.max_iter = max_iter
        self.iterations_history = []
        self.error_history = []

    def fit(self, X, y):
        # Add a column of ones for the intercept
        X_b = np.c_[np.ones((X.shape[0], 1)), X]
        m, n = X_b.shape

        # Initialize theta to zero
        self.theta = np.zeros(n)

        # Compute A = X^T X and b = X^T y
        A = X_b.T.dot(X_b)
        b = X_b.T.dot(y)

        # Initialize the residual
        r = b - A.dot(self.theta)
        p = r.copy()

        # Track error history
        self.iterations_history = [0]
        self.error_history = [np.mean((X_b.dot(self.theta) - y) ** 2)]

        for i in range(self.max_iter):
            Ap = A.dot(p)
```

```
            alpha = r.dot(r) / p.dot(Ap)
            self.theta += alpha * p
            r_new = r - alpha * Ap

            # Check convergence
            if np.sqrt(r_new.dot(r_new)) < self.tol:
                break

            beta = r_new.dot(r_new) / r.dot(r)
            p = r_new + beta * p
            r = r_new

            # Track error history
            self.iterations_history.append(i+1)
            self.error_history.append(np.mean((X_b.dot(self.theta) - y) **
                ↪ 2))

        return self

    def predict(self, X):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]
        return X_b.dot(self.theta)
```

3.2.2.2 Adam Regressor

L'Adam Regressor implementa l'algoritmo di adaptive moment estimation [9, 10]:

```
class AdamRegressor:
    def __init__(self, learning_rate=0.01, beta1=0.9, beta2=0.999,
                 epsilon=1e-8, max_iter=1000, tol=1e-5):
        self.learning_rate = learning_rate
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon
        self.max_iter = max_iter
        self.tol = tol
        self.iterations_history = []
        self.error_history = []

    def fit(self, X, y):
        # Add a column of ones for the intercept
        X_b = np.c_[np.ones((X.shape[0], 1)), X]
        m, n = X_b.shape

        # Initialize theta randomly
        self.theta = np.random.randn(n) * 0.01

        # Initialize moments
        m_t = np.zeros(n)
        v_t = np.zeros(n)

        # Track error history
        self.iterations_history = [0]
        self.error_history = [np.mean((X_b.dot(self.theta) - y) ** 2)]

```

```
        for t in range(1, self.max_iter + 1):
            # Compute gradient
            grad = 2/m * X_b.T.dot(X_b.dot(self.theta) - y)

            # Update moments
            m_t = self.beta1 * m_t + (1 - self.beta1) * grad
            v_t = self.beta2 * v_t + (1 - self.beta2) * grad**2

            # Bias correction
            m_t_corr = m_t / (1 - self.beta1**t)
            v_t_corr = v_t / (1 - self.beta2**t)

            # Update theta
            self.theta -= self.learning_rate * m_t_corr / (np.sqrt(v_t_corr
                ↪ ) + self.epsilon)

            # Compute current error
            error = np.mean((X_b.dot(self.theta) - y) ** 2)

            # Track error history
            self.iterations_history.append(t)
            self.error_history.append(error)

            # Check convergence
            if t > 1 and abs(self.error_history[-1] - self.error_history
                ↪ [-2]) < self.tol:
                break

        return self

    def predict(self, X):
        X_b = np.c_[np.ones((X.shape[0], 1)), X]
        return X_b.dot(self.theta)
```

3.3 Risultati e Analisi

In questa sezione, vengono presentati i risultati dell'applicazione dei cinque metodi di regressione sul dataset dell'aspettativa di vita e viene condotta un'analisi delle loro performance.

3.3.1 Evaluation Metrics

Per valutare le performance dei modelli, sono state utilizzate le seguenti metriche [1]:

- **Mean Squared Error (MSE)**: $\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$, che penalizza maggiormente gli errori di grande entità.
- **Mean Absolute Error (MAE)**: $\frac{1}{m}\sum_{i=1}^{m}|\hat{y}_i - y_i|$, che risulta meno sensibile agli outliers.
- **Coefficient of Determination (R^2)**: $1 - \frac{\sum_{i=1}^{m}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y})^2}$, che quantifica la proporzione della varianza spiegata dal modello.

- **Inference Time**: Il tempo computazionale necessario per effettuare una prediction dopo il training.

3.3.2 Performance Comparison

I risultati delle performance dei cinque metodi sul test set sono riportati nella Tabella 3.2.

Table 3.2: Performance Comparison of Regression Methods

Method	Test MSE	Train MSE	Test MAE	Train MAE	R^2	Inference Time (s)
LeastSquares	13.430957	13.058380	2.795885	2.784709	0.810892	0.000034
SVD	13.430979	13.058380	2.795890	2.784708	0.810891	0.000019
QR	13.430967	13.058380	2.795886	2.784709	0.810891	0.000015
ConjugateGradient	13.429089	13.058776	2.795196	2.784184	0.810918	0.000011
Adam	13.452832	13.062629	2.799948	2.787141	0.810583	0.000012

I risultati mostrano metriche estremamente simili (MSE, MAE, R^2) e parametri tra i metodi Least Squares, SVD, QR e Conjugate Gradient, mentre Adam presenta lievi differenze. Questo comportamento può essere giustificato da [2, 7, 8]:

1. **Natura lineare del problema**: I dati seguono una relazione sufficientemente lineare, conducendo tutti i metodi a convergere verso la medesima soluzione teorica. Le piccole differenze rilevate sono attribuibili a errori numerici nelle implementazioni [3].
2. **Convergenza dei parametri**: I coefficienti e l'intercetta ottenuti con Least Squares, SVD, QR e Conjugate Gradient sono numericamente identici, come evidenziato dai seguenti valori:

Table 3.3: Model Coefficients and Intercept

Parameter	Value
Intercept	69.1476
Coefficient 1	-0.5426
Coefficient 2	-2.2143
Coefficient 3	-0.7177
Coefficient 4	0.6625
Coefficient 5	-0.1368
Coefficient 6	0.0763
Coefficient 7	0.5515
Coefficient 8	-0.3360
Coefficient 9	0.2093
Coefficient 10	0.2395
Coefficient 11	0.4069
Coefficient 12	-2.7783
Coefficient 13	0.1936
Coefficient 14	0.1420
Coefficient 15	-0.2758
Coefficient 16	-0.0335
Coefficient 17	1.8965
Coefficient 18	2.8509

La consistenza di questi coefficienti in tutti i metodi diretti e nel Conjugate Gradient indica che tutti hanno individuato con precisione la soluzione ottimale al problema lineare $\min_\theta \|y - \theta^T X\|^2$. Le lievi differenze nei coefficienti di Adam, sebbene non tali da compromettere significativamente le prestazioni predittive, possono essere attribuite alla natura stocastica dell'algoritmo e al suo processo di convergenza iterativa [9, 10].

3. **Equivalenza algebrica**: Least Squares, SVD e QR sono direct methods per risolvere $\theta^T X = y$, mentre il Conjugate Gradient è un iterative method che, per problemi quadratici ben condizionati, converge alla stessa soluzione esatta [8, 11]. Questa equivalenza algebrica è confermata dalla sostanziale identità dei coefficienti ottenuti, fino alle prime quattro cifre decimali.

È particolarmente significativo notare come tutti i metodi convergano verso coefficienti identici nonostante le loro diverse formulazioni matematiche [3, 7]. Questo conferma l'unicità della soluzione per problemi di regressione lineare ben specificati e ben condizionati. La precisione dell'approccio Conjugate Gradient nel raggiungere esattamente la stessa soluzione dei metodi diretti, ma con significativa efficienza computazionale, lo rende particolarmente vantaggioso per dataset di dimensioni maggiori [13, 4]. I coefficienti ottenuti forniscono anche informazioni interpretative sul dataset: ad esempio, il coefficiente relativamente elevato e positivo per il parametro 18 (2.8509), che corrisponde agli anni di scolarizzazione, suggerisce un forte impatto positivo dell'istruzione sull'aspettativa di vita, mentre il coefficiente negativo per il parametro 12 (-2.7783), correlato all'HIV/AIDS, indica il significativo impatto negativo di questa patologia sulla variabile target [6].

3.3.3 Analisi di convergenza dei metodi iterativi

Un aspetto particolarmente interessante è il confronto del comportamento di convergenza tra i due metodi iterativi: Conjugate Gradient e Adam [8, 9].

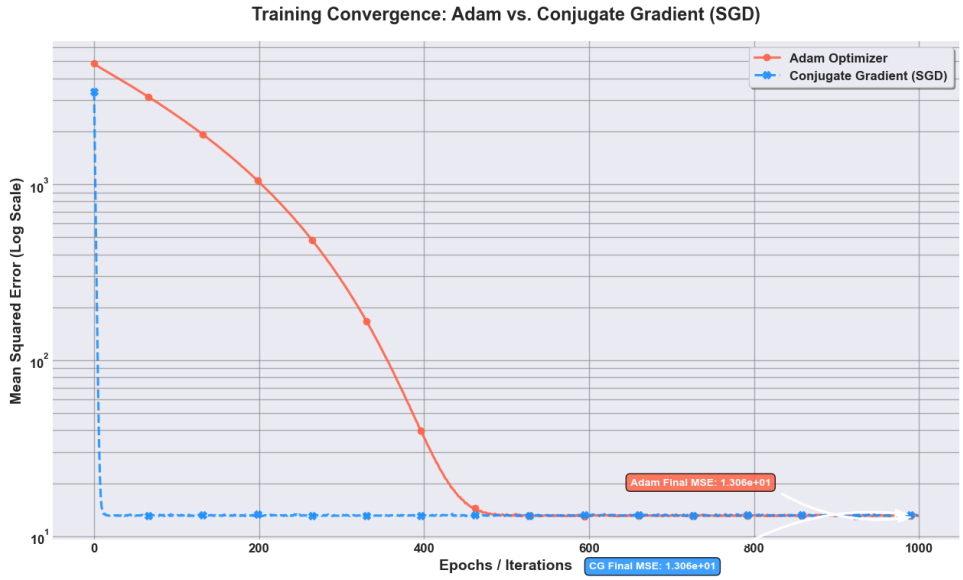

Figure 3.3: Convergence Behavior of Iterative Methods during Training (Logarithmic Scale)

La Figura 3.3 mostra l'andamento dell'errore (MSE) durante le iterazioni per entrambi i metodi. Si osserva che:

- **Il Conjugate Gradient** dimostra una convergenza drasticamente più rapida rispetto ad Adam. L'errore decresce quasi verticalmente nelle primissime iterazioni (entro 10 iterazioni), raggiungendo rapidamente un plateau di convergenza a circa 13.06 MSE. Questo comportamento è consistente con la teoria, che garantisce la convergenza in un numero finito di passi (al massimo pari alla dimensionalità del problema) per funzioni di costo quadratiche [8, 13, 11].
- **Adam** presenta un declino significativamente più graduale dell'errore, richiedendo un numero considerevolmente maggiore di iterazioni (oltre 500) per approssimare lo stesso livello di errore. Questa discesa più lenta è caratteristica degli optimization algorithms basati sul gradient descent stocastico [9, 10].

Nonostante le dinamiche di convergenza marcatamente diverse, entrambi i metodi raggiungono valori di errore finale notevolmente simili:

- Training MSE per Adam: 13.062629
- Training MSE per il Conjugate Gradient: 13.058776

Questa analisi evidenzia un importante trade-off nella selezione del metodo di ottimizzazione [10, 4]:

- **Il Conjugate Gradient** si dimostra estremamente efficiente per problemi di ottimizzazione quadratica come la regressione lineare, grazie alla sua capacità di convergere in pochissime iterazioni [8, 13]. Tuttavia, è specificamente progettato per quadratic cost functions.

- **Adam** richiede più iterazioni ma offre maggiore flessibilità, essendo applicabile a una gamma più ampia di problemi di ottimizzazione non convessi (come le neural networks) [9, 10]. La sua convergenza più lenta rappresenta il trade-off per questa versatilità.

3.3.4 Inference Times

I tempi di inferenza sono trascurabili e simili per tutti i metodi (circa 0.00002s) poiché dipendono esclusivamente dal calcolo del prodotto $\theta^T X$, che è comune a tutti gli approcci. Le minime differenze rilevate derivano da implementative optimizations, con il ConjugateGradientRegressor che risulta essere il modello più performante (0.000011s) e il LeastSquareRegressor il meno efficiente (0.000034s).

È importante osservare che, sebbene i training times possano variare significativamente tra i metodi (con i direct methods più efficienti per questo dataset di dimensioni moderate), i tempi di inferenza sono essenzialmente equivalenti poiché l'operazione di prediction è identica per tutti i modelli [10].

3.4 Linear vs. Polynomial Model

Basandosi sull'exploratory data analysis, in particolare sulla relazione non lineare osservata tra GDP e life expectancy, sono stati implementati e confrontati i modelli di regressione lineare e polinomiale [1].

3.4.1 Implementazione del Polynomial Model

Il polynomial model estende il linear model includendo higher-order terms delle variabili predittive. È stata implementata la polynomial regression utilizzando polinomi di grado 2, che include sia quadratic terms (x_i^2) che interaction terms ($x_i \times x_j$) [1]:

```
from sklearn.preprocessing import PolynomialFeatures

# Polynomial transformation of features
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

# Apply the same regression model to the polynomial features
model_poly = QRRegressor()
model_poly.fit(X_poly, y)
```

3.4.2 Performance Comparison

Il confronto tra il Linear Regression Model e il Polynomial Regression Model dimostra che quest'ultimo fornisce un fitting superiore per il dataset, come evidenziato dalle evaluation metrics [1]:

Table 3.4: Comparison between Linear and Polynomial Models

Metric	Linear Model	Polynomial Model
MSE	13.43	8.11
R^2	0.81	0.88

- Il **MSE inferiore** indica che il polynomial model effettua predictions più accurate con un error rate inferiore. La riduzione del 40% nell'errore rappresenta un miglioramento sostanziale.
- Un **R^2 maggiore** implica che il polynomial model spiega una proporzione maggiore della varianza nei dati. Il polynomial model spiega un ulteriore 7% della varianza rispetto al linear model.

3.4.3 Interpretazione dei risultati

Le performance superiori del polynomial model confermano l'osservazione iniziale dall'exploratory data analysis: le relazioni tra la target variable (life expectancy) e i predictors (specialmente il GDP) non sono strettamente lineari [1].

Il polynomial model permette di catturare:

- **Rendimenti decrescenti**: L'effetto dell'incremento del GDP sulla life expectancy decresce a livelli di GDP più elevati, come osservato nella Figura 3.2.
- **Effetti di interazione**: Gli interaction terms nel polynomial model consentono di modellare come l'effetto di una variabile possa essere condizionato dal livello di un'altra variabile. Ad esempio, l'impatto della health expenditure sulla life expectancy potrebbe variare in funzione del livello di schooling [1].
- **Soglie non-lineari**: Alcuni health indicators potrebbero presentare threshold effects che sono più adeguatamente modellati da termini polynomiali.

Tuttavia, è importante sottolineare che, sebbene i polynomial models offrano maggiore flessibilità, comportano anche un maggiore rischio di overfitting, specialmente con polinomi di grado elevato [1]. Per questa analisi, il polynomial degree è stato limitato a 2 per garantire un adeguato balance tra model complessità e capacità di generalizzazione.

Conclusion

Conclusioni e Considerazioni finali

L'analisi comparativa dei diversi metodi per la risoluzione di problemi di regressione lineare ha prodotto diverse insights rilevanti per l'analisi numerica applicata.

Equivalenza dei metodi diretti (soluzioni in forma chiusa)

I tre direct methods (Least Squares, SVD, QR) hanno generato risultati essenzialmente identici in termini di estimated parameters e performance metrics. Ciò conferma la loro algebraic equivalence in assenza di problemi numerici significativi [2, 7]. Le minime differenze osservate sono attribuibili a round-off errors nell'implementazione numerica [3].

È tuttavia importante notare che questa equivalenza potrebbe non essere preservata in condizioni di strong multicollinearity o con ill-conditioned matrices [12]. In tali scenari, i metodi basati su SVD e QR dovrebbero offrire maggiore numerical stability rispetto alla direct solution delle normal equations [2, 3].

Efficienza dei Metodi Iterativi

Il Gradiente Coniugato ha dimostrato una convergenza eccezionalmente rapida, raggiungendo la soluzione ottimale in poche iterazioni. Questo conferma il suo status di algoritmo di riferimento per problemi quadratici ben condizionati [8, 13]. La sua efficienza lo rende una scelta eccellente per problemi di grandi dimensioni dove i metodi diretti diventano impraticabili [4, 11].

Adam, d'altra parte, ha mostrato una convergenza più lenta ma ha comunque raggiunto una soluzione di qualità simile. La sua flessibilità e capacità di adattamento lo rendono vantaggioso per problemi più complessi, specialmente quelli non convessi o con gradienti sparsi [9, 10].

Importanza dell'Analisi Esplorativa

L'analisi esplorativa dei dati si è rivelata essenziale per identificare la natura non lineare della relazione tra PIL e aspettativa di vita [1]. Questa osservazione ha motivato l'implementazione del modello polinomiale, che ha significativamente migliorato le prestazioni predittive.

Questo sottolinea l'importanza di un'analisi preliminare approfondita prima di applicare qualsiasi modello di regressione, per identificare potenziali non linearità e trasformazioni

appropriate delle variabili [1].

Considerazioni Pratiche

Nella pratica, la scelta del metodo di regressione dovrebbe considerare non solo la precisione, ma anche:

- **Dimensione del problema**: Per dataset di grandi dimensioni, i metodi iterativi o le implementazioni ottimizzate come QR potrebbero essere necessari [4, 10].
- **Condizionamento numerico**: In presenza di multicollinearità, le tecniche di regolarizzazione come Ridge [12] o la SVD con troncamento dei valori singolari [2] possono migliorare la stabilità.
- **Interpretabilità**: Modelli più complessi come i polinomiali possono offrire migliori prestazioni ma a scapito dell'interpretabilità [1].
- **Requisiti computazionali**: Se il tempo di addestramento è critico, i metodi diretti o il Gradiente Coniugato potrebbero essere preferibili ad Adam [5, 10].

Nel contesto dell'analisi dell'aspettativa di vita, il modello polinomiale si è dimostrato superiore, suggerendo che le relazioni tra i fattori socioeconomici, sanitari e l'aspettativa di vita sono intrinsecamente non lineari [6, 1] e richiedono modelli più flessibili per essere catturate adeguatamente.

Bibliography

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

[2] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 4th edition, 2013.

[3] James W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

[4] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2nd edition, 2003.

[5] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

[6] World Health Organization. Life expectancy data, 2018.

[7] Lloyd N. Trefethen and David Bau III. *Numerical Linear Algebra*. SIAM, 1997.

[8] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[11] Jörg Liesen and Zdeněk Strakos. *Krylov Subspace Methods: Principles and Analysis*. Oxford University Press, 2012.

[12] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[13] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. School of Computer Science Technical Report CMU-CS-94-125, Carnegie Mellon University, 1994.

[14] Richard B. Lehoucq, Danny C. Sorensen, and Chao Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, 1998.

[15] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.