Tesina AI:
Stima del reddito annuale in base ai dati del censimento

Autore: Andrea Rostagno

Sommario

Questa tesina tratta l'applicazione di tecniche di Intelligenza Artificiale (IA) per stimare il reddito annuale degli individui utilizzando dati provenienti dal censimento. L'obiettivo principale è valutare l'efficacia di diversi modelli predittivi, inclusi algoritmi di machine learning, nel prevedere con accuratezza la fascia di reddito annuale, analizzando variabili come l'istruzione, l'occupazione, e caratteristiche demografiche.

Nella prima parte viene introdotto il problema della stima del reddito e l'importanza dei dati di censimento nella ricerca socioeconomica. Successivamente, sono descritti i metodi utilizzati, con particolare attenzione agli algoritmi di classificazione come regressione logistica, alberi decisionali e reti neurali artificiali.

La sezione successiva presenta i risultati ottenuti dai modelli sviluppati, confrontandone le prestazioni tramite metriche come accuratezza, precisione e recall. Infine, vengono discussi limiti e potenziali miglioramenti, sottolineando le implicazioni sociali e le sfide etiche nell'applicazione di queste tecnologie.

Indice

Capitolo 1

Introduzione e dataset

1.1 Introduzione

Negli ultimi anni, l'analisi dei dati ha assunto un ruolo sempre più centrale in numerosi ambiti, dalla ricerca economica alla previsione dei comportamenti sociali. In questo elaborato, ci proponiamo di analizzare il dataset *Adult*, che contiene informazioni socio-economiche e demografiche su un campione di individui con l'obiettivo di stimare il loro reddito medio annuo. L'analisi si concentrerà sulla classificazione binaria del reddito, determinando se un individuo guadagni più o meno di 50.000$ all'anno.

Per garantire un'analisi accurata e significativa, i dati sono stati filtrati e pre-processati secondo i seguenti criteri:

- L'età dell'individuo deve essere superiore ai 16 anni.
- Il reddito lordo rettificato della persona deve essere superiore a 100.
- Ogni dato deve avere un peso maggiore di 1.
- L'individuo deve aver lavorato almeno 1 ora alla settimana.

L'analisi verrà svolta attraverso un processo strutturato in tre fasi principali:

1. **Pre-processing e visualizzazione dei dati**: pulizia del dataset, gestione dei valori mancanti e rappresentazione grafica delle variabili principali.
2. **Riduzione della dimensionalità**: confronto tra la **Principal Component Analysis (PCA)** e la **Multiple Discriminant Analysis (MDA)** per identificare le feature più rilevanti.
3. **Tecniche di classificazione**: utilizzo e confronto tra **Linear Discriminant Analysis (LDA)**, **Support Vector Machines (SVM)** e **SVM con kernel non lineari**.

Durante l'analisi verranno presentati i risultati ottenuti, confrontando le performance dei vari modelli attraverso metriche standard come *accuracy, precision, recall e F1-score*. Verranno inoltre discusse le implicazioni della riduzione della dimensionalità sui modelli predittivi e la scelta del kernel ottimale per le SVM.

L'obiettivo finale è quello di individuare la tecnica di classificazione più efficace per stimare il reddito sulla base delle variabili disponibili, fornendo un quadro dettagliato sull'impatto delle diverse metodologie statistiche e di machine learning.

1.2 Dataset

Qui in allegato è presente il link diretto del dataset:
https://archive.ics.uci.edu/dataset/2/adult
Il dataset contiene 32.561 campioni relativi a persone differenti e 14 features che andiamo ad elencare:

- **Age (Età)**: Assume valori interi e non presenta valori mancanti.
- **Workclass (Classe di lavoro)**: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked. Presenta valori mancanti.
- **Fnlwgt (Peso finale)**: Indica quella data persona quanto popolata sarebbe per fare una media ponderata. Assume solo valori interi e non presenta valori mancanti.
- **Education (Titoli di studio)**: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool. Non presenta dati mancanti.
- **Education-num (Voto)**: Solo numeri interi e nessun dato mancante.
- **Marital-status (Stato civile)**: Married-civ-spouse, Divorced, Never-married, Widowed, Married-spouse-absent, Married-AF-spouse. Nessun dato mancante.
- **Occupation (Lavoro)**: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces. Sono presenti dei dati mancanti.
- **Relationship (Relazione)**: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried. Nessun dato mancante.
- **Race (Etnia)**: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black. Nessun dato mancante.
- **Sex (Sesso)**: Female, Male. Nessun dato mancante.
- **Capital-gain (Guadagno capitale)**: Rappresenta il guadagno di capitale, ovvero quando un prezzo superiore del prezzo di acquisto (esempio: acquisti delle azioni a 1000 e *rivendi* a 1500, il guadagno capitale è di 500$). Assume solo valori interi e non presenta nessun dato mancante.
- **Capital-loss (Perdita capitale)**: Rappresenta la perdita di capitale, ovvero quando un prezzo inferiore del prezzo di acquisto (esempio: acquisti delle azioni a 1500 e *rivendi* a 1000, la perdita capitale è di 500$). Assume solo valori interi e non presenta nessun dato mancante.
- **Hours-per-week (Ore settimanali lavorative)**: Assume solo valori interi e non presenta nessun dato mancante.
- **Native-country (Nazionalità)**: United-States, Cambodia, England, Puerto-Rico, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad&Tobago, Peru, Hong, Holand-Netherlands. Sono presenti dati mancanti.

Invece, per quanto riguarda gli attributi aventi outcome la casella **income** (guadagno) assume due valori: >50K, <=50K.

Capitolo 2

Pulizia e visualizzazione del dataset

2.1 Importazione delle librerie e caricamento del dataset

Il primo blocco di codice importa le librerie necessarie per l'analisi dei dati e l'implementazione di algoritmi di apprendimento automatico. In particolare:

- `pandas` e `numpy`: per la manipolazione e l'analisi dei dati.
- `sklearn` (scikit-learn): per algoritmi di machine learning come Linear Discriminant Analysis (LDA), Support Vector Machines (SVM) e Principal Component Analysis (PCA).
- `matplotlib`: per la visualizzazione dei dati, incluse rappresentazioni 3D.
- `train_test_split`: per la divisione del dataset in set di addestramento e test.
- `LabelEncoder` e `StandardScaler`: per la normalizzazione e la codifica dei dati.
- `precision_score`, `recall_score`, `fbeta_score`: metriche per valutare la performance dei modelli.

```
1         import pandas as pd
2         import numpy as np
3         from sklearn import datasets
4         from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
    as LDA
5         from FisherDA import MultipleFisherDiscriminantAnalysis as MDA
6         from sklearn.decomposition import PCA
7         from sklearn.svm import LinearSVC
8         from sklearn.svm import SVC
9         from sklearn.preprocessing import StandardScaler
10        from sklearn.model_selection import train_test_split
11        import matplotlib
12        import matplotlib.pyplot as plt
13        from matplotlib.colors import ListedColormap
14        from mpl_toolkits.mplot3d import Axes3D
15        from IPython.display import display
16        from sklearn.preprocessing import LabelEncoder
17        from sklearn.metrics import precision_score, recall_score, fbeta_score
18
```

Listing 2.1. Importazione delle librerie

2.1.1 Caricamento del dataset

Il secondo blocco di codice importa il dataset `adult.data` e assegna i nomi corretti alle colonne. In particolare:

- Il dataset viene caricato usando `pd.read_csv()`, specificando che i valori mancanti sono rappresentati da "?".
- Si definisce l'elenco delle colonne del dataset e si rinominano le colonne con il metodo `rename()`.
- Infine, il dataset viene visualizzato con il comando `display(adult)`.

```
1         # Importiamo il dataset
2         adult = pd.read_csv('adult.data', header=None, na_values=" ?")
3         adult_columns = ['age', 'workclass', 'fnlwgt', 'education', 'education
    -num',
4         'marital-status', 'occupation', 'relationship', 'race',
5         'sex', 'capital-gain', 'capital-loss', 'hours-per-week',
6         'native-country', 'income']
7         adult.rename(columns={k: adult_columns[k] for k in range(len(
    adult_columns))}, inplace=True)
8
9         display(adult)
10
```

Listing 2.2. Caricamento del dataset

age	workclass	fnlwgt	education	educ-num	marital-status	occupation
39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical
50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial
38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners
53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners
28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty
⋮	⋮	⋮	⋮	⋮	⋮	⋮
27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support
40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct
58	Private	151910	HS-grad	9	Widowed	Adm-clerical
22	Private	201490	HS-grad	9	Never-married	Adm-clerical
52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial

Tabella 2.1. Esempio di dati estratti dal dataset `Adult`.

relationship	race	sex	capital-gain	capital-loss	hxw	native-country	income
Not-in-family	White	Male	2174	0	40	United-States	<=50K
Husband	White	Male	0	0	13	United-States	<=50K
Not-in-family	White	Male	0	0	40	United-States	<=50K
Husband	Black	Male	0	0	40	United-States	<=50K
Wife	Black	Female	0	0	40	Cuba	<=50K
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Wife	White	Female	0	0	38	United-States	<=50K
Husband	White	Male	0	0	40	United-States	>50K
Unmarried	White	Female	0	0	40	United-States	<=50K
Own-child	White	Male	0	0	20	United-States	<=50K
Wife	White	Female	15024	0	40	United-States	>50K

Tabella 2.2. Subset del dataset `Adult`: Colonne da `relationship` a `income`.

2.2 Analisi preliminare del dataset

2.2.1 Informazioni sul dataset

Prima di effettuare qualsiasi analisi, è importante esaminare la struttura del dataset. Il comando `adult.info()` fornisce un riepilogo delle colonne presenti, il numero di valori non nulli e il tipo di dati di ciascuna colonna.

```
1         adult.info()
2
```

Listing 2.3. Struttura del dataset

Dall'output vediamo che il dataset contiene 32.561 osservazioni e 15 colonne. Alcune colonne hanno valori mancanti, in particolare:

- `workclass`, `occupation`, `native-country` presentano valori mancanti.
- Il dataset è composto da variabili sia numeriche (`int64`) che categoriche (`object`).

2.2.2 Distribuzione della variabile `income`

Analizziamo ora la distribuzione della variabile target `income`, che indica se una persona guadagna più o meno di 50.000 dollari annui.

```
1         adult['income'].value_counts()
2
```

Listing 2.4. Distribuzione della variabile target

L'output risultante è:

```
<=50K    24720
>50K      7841
```

Ciò indica che la maggior parte delle persone nel dataset guadagna meno di 50.000 dollari.

2.2.3 Eliminazione dei valori mancanti

Poiché alcune colonne contengono valori nulli, rimuoviamo le righe che li contengono usando il metodo `dropna()`.

```
1         adult_b = adult.dropna()
2         M, N = adult_b.shape
3
```

Listing 2.5. Eliminazione dei valori nulli

Dopo la pulizia, analizziamo di nuovo la distribuzione della variabile target:

```
1         adult_b['income'].value_counts()
2
```

Listing 2.6. Distribuzione della variabile target dopo la pulizia

L'output aggiornato è:

```
<=50K    22654
>50K      7508
```

2.2.4 Osservazioni

Dopo la rimozione dei valori mancanti, il dataset è leggermente ridotto, ma la proporzione tra le due classi della variabile target rimane simile, quindi l'operazione non altera in modo significativo l'analisi.

2.3 Distribuzione dell'età

2.3.1 Analisi della distribuzione dell'età

Per analizzare la distribuzione dell'età nel dataset, viene costruito un *istogramma* che mostra la frequenza delle diverse fasce di età.

```
        # Una distribuzione dell'eta
        plt.figure(figsize=(8, 4))
        plt.hist(adult_b['age'], bins=25, edgecolor='black')
        plt.title('Distribuzione dell\'eta')
        plt.xlabel('Eta')
        plt.ylabel('Frequenza')
        plt.xticks(ticks=np.arange(16,92,5), labels=[f'{i}' for i in range
(16,92,5)])
        plt.yticks(ticks=np.arange(0,2750,250), labels=[f'{i}' for i in range
(0,2750,250)])
        plt.grid(False)
        plt.show()

```

Listing 2.7. Creazione dell'istogramma dell'età

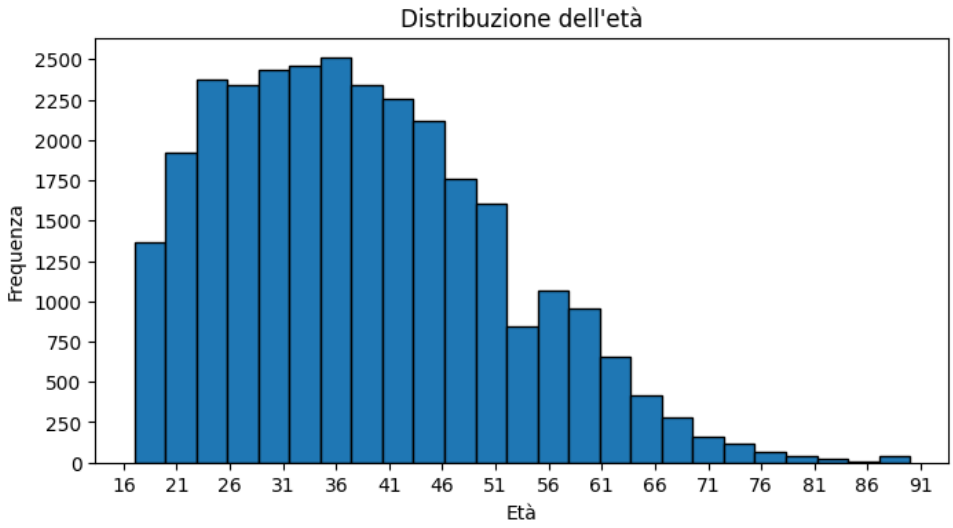

Figura 2.1. Frequenza eta dataset.

2.3.2 Calcolo dell'età media ponderata

L'età media ponderata viene calcolata tenendo conto della colonna *fnlwgt* (peso campionario). Il calcolo si basa sulla formula:

$$\bar{X} = \frac{\sum_i (x_i \cdot w_i)}{\sum_i w_i}$$

dove x_i rappresenta l'età e w_i il peso associato.

```
1       average_age = np.round((adult_b['age'] * adult_b['fnlwgt']).sum() /
    adult_b['fnlwgt'].sum())
2       display("L'eta media ponderata e: " + str(average_age) + " anni")
3
```

Listing 2.8. Calcolo dell'età media ponderata

2.3.3 Risultati

Dopo il calcolo, l'output ottenuto è:

```
"L'età media ponderata è: 38.0 anni"
```

Il valore ottenuto indica che l'età media nel dataset, considerando il peso demografico, è di *38 anni*.

2.3.4 Osservazioni

L'istogramma mostra che:

- La distribuzione dell'età è *asimmetrica* con un picco tra i 30 e i 40 anni.
- La frequenza diminuisce significativamente dopo i 50 anni.
- Il dataset include individui di età compresa tra *16 e 91 anni*.

2.4 Preprocessing delle variabili categoriche

2.4.1 Conversione delle variabili categoriche

Le colonne categoriche nel dataset devono essere convertite in un formato numerico per poter essere utilizzate nei modelli di machine learning. In questa fase:

- Le colonne `workclass`, `education`, `marital-status`, `occupation`, `relationship`, `race`, `sex`, e `native-country` vengono trasformate in tipo `category`.

```
        # Trasformo le colonne object in category
        adult_b = adult_b.astype({"workclass":"category", "education":"
    category",
             "marital-status":"category", "occupation":"category",
             "relationship":"category","race":"category",
             "sex":"category", "native-country":"category"})

```

Listing 2.9. Conversione in variabili categoriche

2.4.2 Codifica della variabile target

Poiché la variabile `income` assume solo due valori (`>50K` e `<=50K`), utilizziamo il *Label Encoding* per trasformarla in una variabile binaria.

```
        from sklearn.preprocessing import LabelEncoder

        adult_work = adult_b.copy()
        le = LabelEncoder()
        le.fit(adult_work['income'].values)
        adult_work['income'] = le.transform(adult_work['income'].values)

```

Listing 2.10. Label Encoding per la variabile target

2.4.3 One-hot encoding delle feature categoriche

Le feature categoriche con più di due classi vengono trasformate tramite *One-hot Encoding*, che crea nuove colonne binarie per ciascuna categoria.

```
        onehot_cols = ['workclass', 'education', 'marital-status', 'occupation
    ',
        'relationship', 'race','sex','native-country']

        adult_work_old = adult_work.copy()
        adult_work = pd.get_dummies(adult_work, columns=onehot_cols)
        adult_work = adult_work.astype(int)

```

Listing 2.11. One-hot encoding delle feature categoriche

2.4.4 Riordinamento delle colonne

Dopo il one-hot encoding, una colonna viene spostata per migliorare l'ordine del dataset.

```
        new_pos = 104
        col_to_move = adult_work.columns[6]
        cols = adult_work.columns.tolist()
        cols.insert(new_pos, cols.pop(cols.index(col_to_move)))
        adult_work = adult_work[cols]

```

Listing 2.12. Riordinamento delle colonne

2.4.5 Output del dataset preprocessato

Ora il dataset è stato trasformato con variabili numeriche ed è pronto per l'uso nei modelli di machine learning.

```
        display(adult_work)

```

Listing 2.13. Visualizzazione del dataset trasformato

2.4.6 Osservazioni

Dopo la trasformazione:

- Le variabili categoriche sono state convertite in variabili binarie.
- La variabile target è ora rappresentata come valore binario (0 = <=50K, 1 = >50K).
- Il dataset è pronto per essere utilizzato nei modelli predittivi.

2.5 Relazione tra età e ore lavorative settimanali

2.5.1 Analisi della relazione

Per comprendere meglio la distribuzione dell'orario di lavoro in base all'età e al reddito, utilizziamo un *grafico di dispersione (scatter plot)*. Ogni punto rappresenta un individuo del dataset, con:

- L'asse delle ascisse (**x**) che indica l'età.
- L'asse delle ordinate (**y**) che rappresenta le ore lavorative settimanali.
- Il colore che distingue le due classi di reddito (**blu** per `<=50K`, **arancione** per `>50K`).

```
        # Mostriamo la relazione tra eta e ore settimanali lavorative
        cmap=ListedColormap(['blue', 'orange'])
        plt.figure(figsize=(8, 4))
        plt.scatter(adult_work_old['age'], adult_work_old['hours-per-week'],
        c=adult_work_old['income'], cmap=cmap, alpha=0.5)

        plt.title('Relazione tra Eta e ore lavorative settimanali')
        plt.xlabel('Eta')
        plt.ylabel('Ore lavorative settimanali')

        # Legenda per distinguere i colori
        handles = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor
    =cmap(0), markersize=10, label='<=50K'),
```

```
13        plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=cmap(1),
    markersize=10, label='>50K')]
14        plt.legend(handles=handles, title='income')
15
16        # Personalizzazione degli assi
17        plt.xticks(ticks=np.arange(16,92,5), labels=[f'{i}' for i in range
    (16,92,5)])
18        plt.yticks(ticks=np.arange(0,110,10), labels=[f'{i}' for i in range
    (0,110,10)])
19        plt.grid(True)
20        plt.show()
21
```

Listing 2.14. Scatter plot della relazione tra età e ore lavorative settimanali

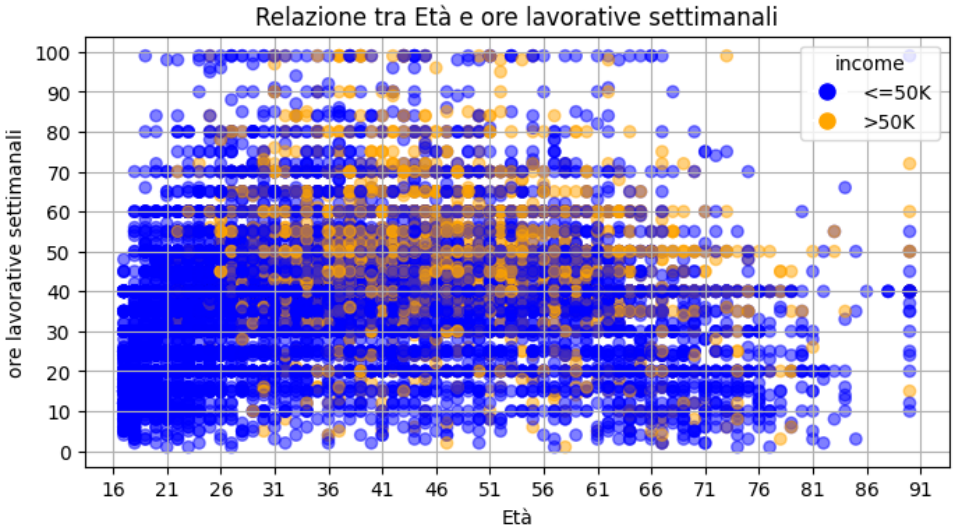

Figura 2.2. Relazione tra età e ore lavorative.

2.5.2 Osservazioni

Dall'analisi del grafico, possiamo notare i seguenti aspetti:

- Le persone tra i " 16 e i 26 anni" che lavorano fino a " 40 ore settimanali" non guadagnano più di 50K all'anno.
- Le persone che " guadagnano di più" (**classe >50K**) si trovano in un'età compresa tra i " 31 e i 61 anni" e lavorano tra le " 40 e le 70 ore settimanali" .
- La distribuzione mostra che la maggioranza della popolazione lavora intorno alle " 40 ore settimanali" , con alcune eccezioni per chi supera le " 60-80 ore settimanali" .

2.6 Relazione tra età e guadagno da investimenti

2.6.1 Analisi della relazione

Per studiare la correlazione tra " età" e " guadagni da investimenti (capital gain)" , utilizziamo un " grafico di dispersione (scatter plot)" . Ogni punto rappresenta un individuo nel dataset, con:

- L'asse delle ascisse (**x**) che indica l'età.
- L'asse delle ordinate (**y**) che rappresenta il guadagno di capitale.
- Il colore che distingue le due classi di reddito (**blu** per `<=50K`, **arancione** per `>50K`).

```
        # Mostriamo la relazione tra guadagni da investimenti ed eta
        plt.figure(figsize=(8, 4))
        plt.scatter(adult_work_old['age'], adult_work_old['capital-gain'],
        c=adult_work_old['income'], cmap=cmap, alpha=0.50)

        plt.title('Eta vs Guadagno di Capitale')
        plt.xlabel('Eta')
        plt.ylabel('Guadagno di Capitale')

        # Legenda per distinguere i colori
        plt.legend(handles=handles, title='Income')

        # Personalizzazione degli assi
        plt.xticks(ticks=np.arange(16,92,5), labels=[f'{i}' for i in range
    (16,92,5)])
        plt.yticks(ticks=np.arange(0,110000,10000), labels=[f'{i}' for i in
    range(0,110000,10000)])
        plt.grid(True)
        plt.show()

```

Listing 2.15. Scatter plot della relazione tra età e guadagno di capitale

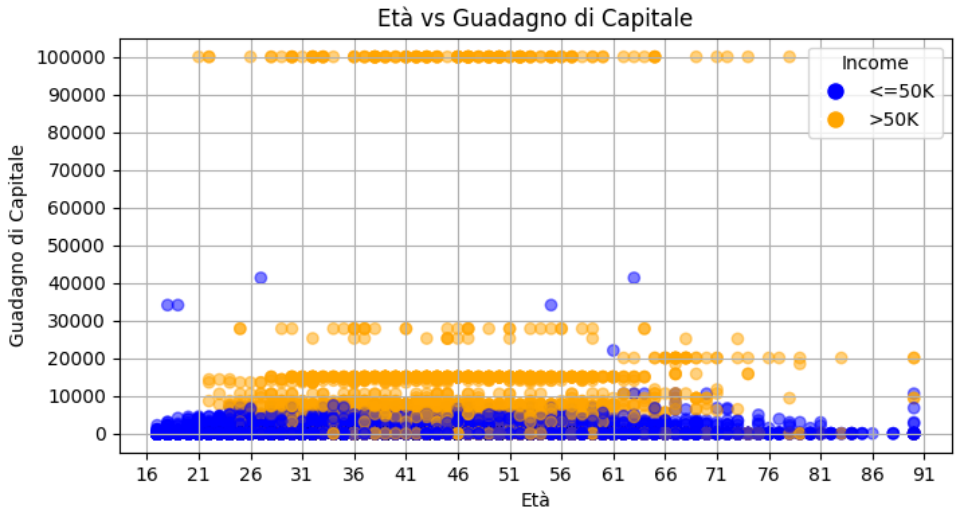

Figura 2.3. Relazione tra età e guadagno.

2.6.2 Osservazioni

Dall'analisi del grafico emergono i seguenti aspetti:

- Quasi tutti coloro che " guadagnano più di 50K all'anno" percepiscono almeno " 10K di guadagni da investimenti" .
- Le persone che " guadagnano meno di 50K" raramente ricevono più di " 5.000$" da investimenti vari.
- La distribuzione suggerisce una netta separazione tra le due classi di reddito in termini di investimenti.

2.7 Relazione tra età e perdita di capitale

2.7.1 Analisi della relazione

Per studiare la correlazione tra " età" e " perdite di capitale (capital loss)" , utilizziamo un " grafico di dispersione (scatter plot)" . Ogni punto rappresenta un individuo nel dataset, con:

- L'asse delle ascisse (**x**) che indica l'età.
- L'asse delle ordinate (**y**) che rappresenta la perdita di capitale.
- Il colore che distingue le due classi di reddito (**blu** per `<=50K`, **arancione** per `>50K`).

```
        # Mostriamo la relazione tra eta e perdita di capitale
        plt.figure(figsize=(8, 4))
        plt.scatter(adult_work_old['age'], adult_work_old['capital-loss'],
        c=adult_work_old['income'], cmap=cmap, alpha=0.5)

        plt.title('Eta vs Perdita di Capitale')
        plt.xlabel('Eta')
        plt.ylabel('Perdita di Capitale')

        # Legenda per distinguere i colori
        plt.legend(handles=handles, title='Income')

        # Personalizzazione degli assi
        plt.xticks(ticks=np.arange(16,92,5), labels=[f'{i}' for i in range
    (16,92,5)])
        plt.yticks(ticks=np.arange(0,5000,500), labels=[f'{i}' for i in range
    (0,5000,500)])
        plt.grid(True)
        plt.show()

```

Listing 2.16. Scatter plot della relazione tra età e perdita di capitale

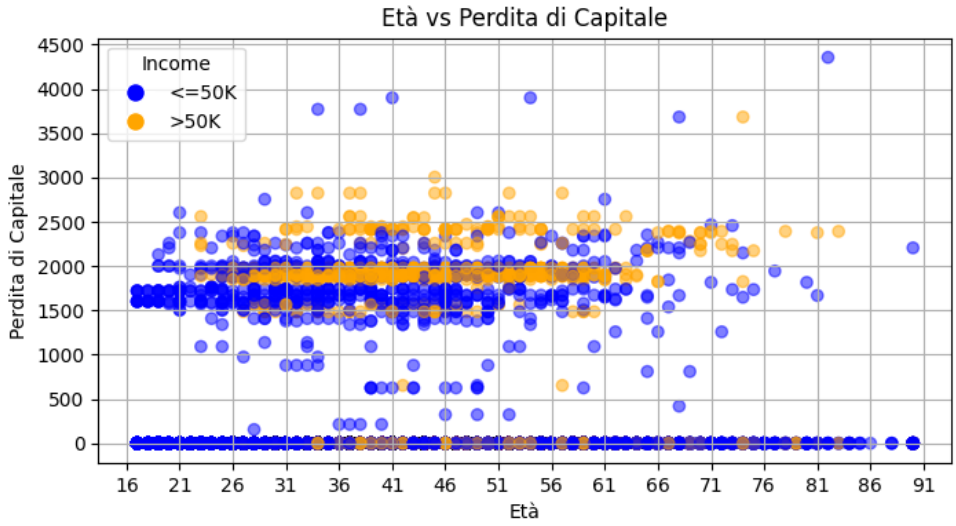

Figura 2.4. Relazione tra età e perdita.

2.7.2 Osservazioni

Dall'analisi del grafico emergono i seguenti aspetti:

- Chi " guadagna di più (>50K)" sembra avere due " soglie di perdita" ("stop-loss"): una intorno ai " 2000$" e l'altra intorno ai " 2500$" .
- Le perdite per investimenti sono " concentrate tra i 16 e i 61 anni" , dopo di che " diminuiscono nettamente" .

Capitolo 3

Metodi di predizione

3.1 Principal Component Analysis (PCA)

3.1.1 Introduzione

Quando ci troviamo ad operare in situazioni in cui si hanno dati con un gran numero di dimensioni, è naturale pensare di proiettare questi dati in un sottospazio di dimensionalità inferiore, cercando di non perdere informazione importante sulle variabili originali.

Un modo per ottenere tale riduzione è attraverso la selezione automatica, chiamata anche " feature selection" . La " Principal Component Analysis (PCA)" è una tecnica finalizzata a ridurre la dimensionalità di un insieme di dati con finalità esplorativa o di visualizzazione dei dati, per un eventuale uso in analisi successive.

3.1.2 Procedimento

Supponiamo di trovarci in uno spazio $\mathbb{R}^d$ e di volere una funzione lineare f tale che:

$$f : \mathbb{R}^d \to \mathbb{R}^e$$

ma al contempo un'altra funzione lineare g tale che:

$$g : \mathbb{R}^e \to \mathbb{R}^d$$

con $e < d$. Associamo a queste due trasformazioni le loro corrispettive matrici $U \in \mathbb{R}^{d \times e}$ e $V \in \mathbb{R}^{e \times d}$.

Il nostro campione S ha distribuzione D^m ovvero $S = \{p_1, \ldots, p_m\}$. L'obiettivo è trovare le due migliori funzioni lineari che permettano di minimizzare la perdita di informazione:

$$\arg\min_{U,V} \sum_{i=1}^{m} \|p_i - UVp_i\|_2^2$$

Dobbiamo quindi trovare una matrice $U \in \mathbb{R}^{d \times e}$ ed una matrice $V \in \mathbb{R}^{e \times d}$ che siano soluzione del problema di minimizzazione.

3.1.3 Passaggi

1. Costruiamo una matrice
$$A = \sum_{i=1}^{m} p_i p_i^T$$
dove p_i è un vettore noto di dimensione $d \times 1$.

2. La matrice A può essere anche calcolata come:

$$A = X^T X$$

dove X è la matrice con righe i campioni di S, quindi X ha dimensione $m \times d$.

3. Poiché A è simmetrica e definita positiva, possiamo calcolare la sua decomposizione ai valori singolari (SVD):

$$A = BDB^T$$

dove:

- D è una matrice diagonale con gli autovalori di A in ordine decrescente.
- B è la matrice con colonne gli autovettori di A.

4. Le " componenti principali" sono gli autovettori di B, mentre le matrici incognite sono:

- V, che contiene le colonne con i primi e autovettori.
- U uguale a V^T.

5. Riformuliamo il problema:

$$\arg\max \operatorname{trace}(V^T A V)$$

Soggetto a:

$$VV^T = I$$

con $V \in \mathbb{R}^{d \times e}$.

6. Ora, sapendo che le colonne di V sono gli autovettori associati agli autovalori di A, possiamo quantificare l'errore nel passare da d a e dimensioni:

$$\sum_{i=e+1}^{d} \lambda_i$$

ovvero, sommiamo gli autovalori che sono stati eliminati.

3.1.4 Varianza spiegata

La varianza totale può essere scritta come:

$$\sum_{i=1}^{d} \lambda_i = \operatorname{tr}(A)$$

Dove il parametro λ_i rappresenta la varianza spiegata dalla i-esima componente principale.
Definiamo la " Percentuale di Varianza Spiegata (PVE)" :

$$PVE_i = \frac{\lambda_i}{\sum_j \lambda_j} = \frac{\lambda_i}{\operatorname{tr}(A)}$$

Questo valore indica quanta informazione porta ogni componente principale.

Un altro modo di visualizzare la varianza è tramite la " Varianza Cumulata (CPVE)" . A differenza della PVE, la CPVE mostra quanta varianza è stata rappresentata fino a quella componente principale.

Se k è il numero di componenti principali di riferimento, allora la CPVE in quel punto è:

$$CPVE_i = \frac{\sum_{i=1}^{k} \lambda_i}{\operatorname{tr}(A)}$$

3.2 Applicazione della PCA

3.2.1 Preparazione dei dati

Per applicare la " Principal Component Analysis (PCA)" , iniziamo separando le feature dalla variabile target e dividiamo il dataset in " training set" e " test set" .

```
        # Separiamo le classi dalle features
        x = adult_work.iloc[:, :-1].values
        y = adult_work['income'].values

        # Generiamo il training set
        random_seed = 42
        test_p = 0.50
        X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=
    test_p,
        random_state=random_seed, shuffle=True)

```

Listing 3.1. Separazione delle feature e della variabile target

3.2.2 Standardizzazione

Dal momento che la PCA è sensibile alla scala delle variabili, standardizziamo i dati prima di applicarla.

```
        # Standardizziamo le feature
        scaler_X = StandardScaler()
        scaler_X.fit(X_train)
        X_scaled = scaler_X.transform(X_train)

```

Listing 3.2. Standardizzazione dei dati

3.2.3 Applicazione della PCA

Applichiamo la PCA sia sui dati standardizzati che sui dati originali per confrontare le differenze.

```
        # Applichiamo la PCA sia su dati standardizzati che non
        pca = PCA()
        pca_nonst = PCA()
        pca.fit(X_scaled)
        pca_nonst.fit(X_train)

```

Listing 3.3. Applicazione della PCA

3.2.4 Visualizzazione della varianza spiegata

Mostriamo ora la " varianza spiegata cumulativa" per capire quante componenti principali sono necessarie per rappresentare una certa percentuale di informazione.

```
        plt.figure()
        plt.plot(np.insert(np.cumsum(pca.explained_variance_ratio_), 0, 0))
        plt.title('x_scaled')
        plt.xticks(ticks=np.arange(1, pca.n_components_ + 3, 8),
        labels=[f'PC{i}' for i in range(1, pca.n_components_ + 3, 8)],
    rotation=45)
        plt.xlabel('Principal components')
        plt.ylabel('Cumulative explained variance (%)')
        plt.grid()
        plt.show()

```

Listing 3.4. Varianza spiegata per i dati standardizzati

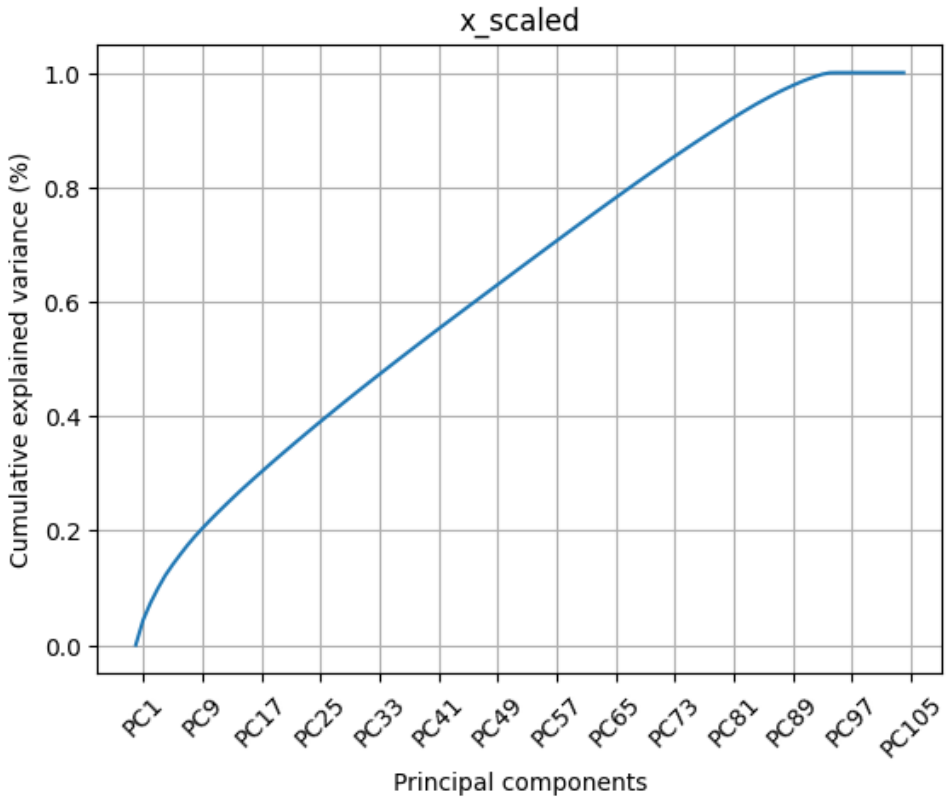

Figura 3.1. Varianza standardizzata.

```
        plt.figure()
        plt.bar(np.arange(pca_nonst.n_components_), pca_nonst.
    explained_variance_ratio_,
        bottom=np.insert(np.cumsum(pca_nonst.explained_variance_ratio_), 0, 0)
    [:-1])
        plt.plot(np.cumsum(pca_nonst.explained_variance_ratio_), 'r')
        plt.title('x_nonst')
        plt.xticks(ticks=np.arange(1, pca_nonst.n_components_ + 1, 10),
        labels=[f'PC{i}' for i in range(1, pca_nonst.n_components_ + 1, 10)])
        plt.xlabel('Principal components')
        plt.ylabel('Cumulative explained variance (%)')
        plt.grid()
        plt.show()

```

Listing 3.5. Varianza spiegata per i dati non standardizzati

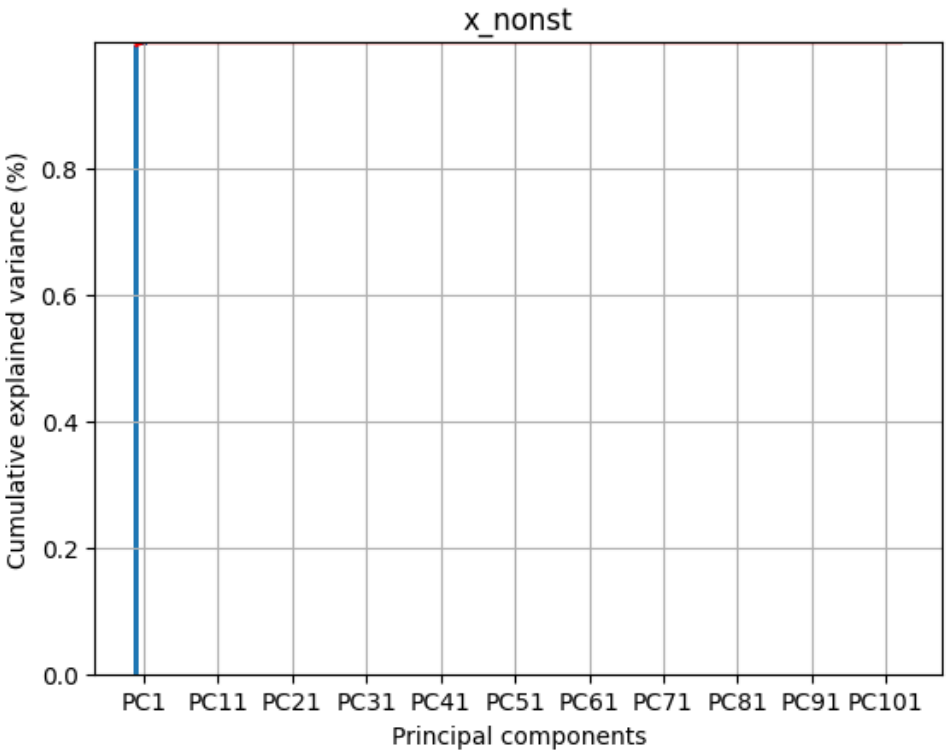

Figura 3.2. Varianza non standardizzata.

3.2.5 Osservazioni

Dall'analisi dei grafici della varianza spiegata:

- La PCA applicata ai dati " standardizzati" mostra un andamento più regolare rispetto a quella sui dati " non standardizzati" .
- La maggior parte dell'informazione è catturata nelle prime " poche componenti principali" .
- La varianza cumulativa aiuta a scegliere il numero ottimale di componenti da mantenere.

3.2.6 Selezione del numero di componenti principali

Per scegliere il numero ottimale di " componenti principali" , fissiamo una soglia di varianza spiegata pari al " 90%" e calcoliamo il numero di " Principal Components (PCs)" necessari.

```
        # Scegliamo il numero di componenti principali in base alla varianza
    spiegata
        explvar_p = 0.90
        pca = PCA(explvar_p)
        pca.fit(X_scaled)
        pca_adult = pca.transform(X_scaled)

        # Trasformiamo i dati originali in quelli trasformati dalla PCA
        features_scalate = scaler_X.transform(x)
        Zpca = pca.transform(features_scalate)

        df_pca = pd.DataFrame({'val': [pca.n_components_,
            np.round(pca.explained_variance_ratio_.sum()*100, decimals=2)]},
        index=['n. PC', 'expl. Var. (%)'])
        display(df_pca)

```

Listing 3.6. Selezione del numero di componenti principali

Dall'output otteniamo:

Parametro	Valore
Numero di componenti principali (n. PC)	79.00
Varianza spiegata (%)	90.51

3.2.7 Visualizzazione dei dati trasformati

Una volta applicata la PCA, possiamo " visualizzare i dati proiettati nelle prime componenti principali" .

```
        # Mostriamo il grafico a dispersione 2D
        plt.figure(figsize=(12, 12))
        plt.scatter(pca_adult[:, 0], pca_adult[:, 1], c=y_train, cmap=cmap,
    alpha=0.5)
        plt.title('ADULT - SCORE GRAPH - 2D')
        plt.xlabel('PC1')
        plt.ylabel('PC2')
        plt.legend(handles=handles, title='income')
        plt.grid()
        plt.show()

```

Listing 3.7. Grafico a dispersione 2D delle prime due componenti principali

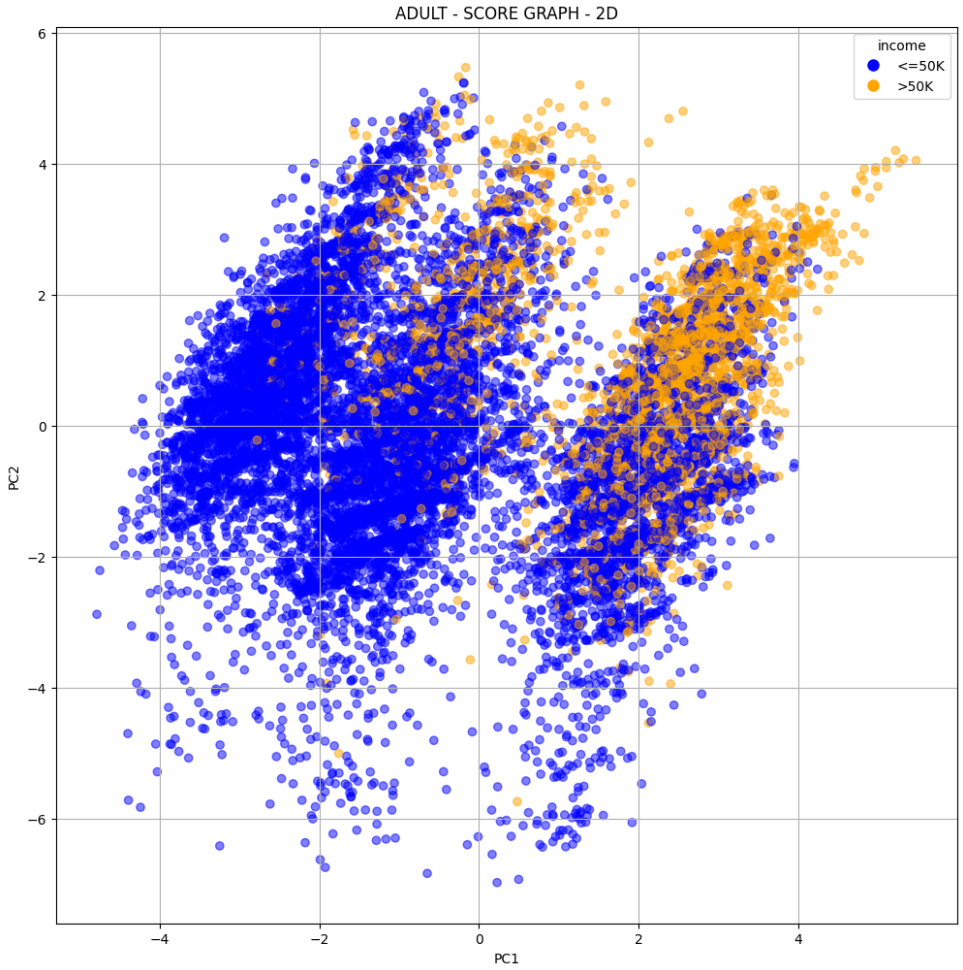

Figura 3.3. score graph 2D.

```
        # Mostriamo il grafico a dispersione 3D
        fig = plt.figure(figsize=(12, 12))
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(pca_adult[:, 0], pca_adult[:, 1], pca_adult[:, 2], c=
    y_train, cmap=cmap, alpha=0.5)
        ax.set_title('ADULT - SCORE GRAPH - 3D')
        ax.set_xlabel('PC1')
        ax.set_ylabel('PC2')
        ax.set_zlabel('PC3')
        plt.grid()
        plt.show()

```

Listing 3.8. Grafico a dispersione 3D delle prime tre componenti principali

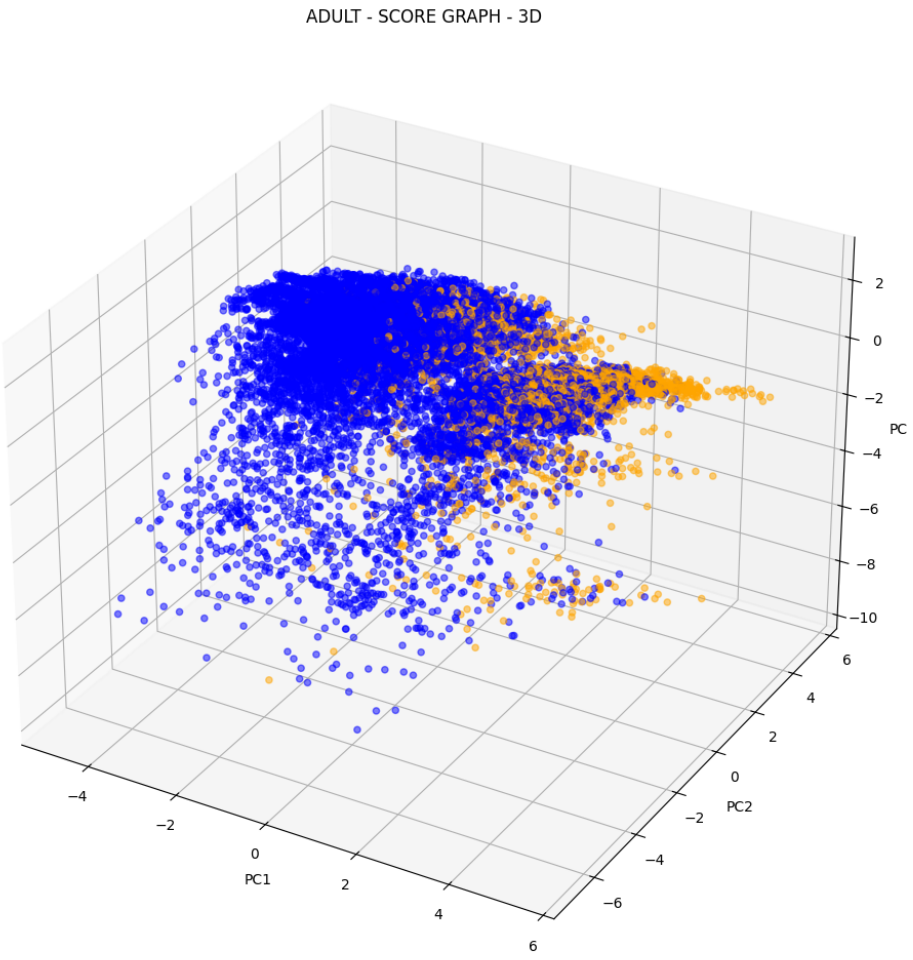

Figura 3.4. score graph 3D.

3.2.8 Osservazioni

Dall'analisi dei risultati ottenuti:

- Con " 79 componenti principali" , possiamo spiegare circa il " 90.51%" della varianza totale dei dati.

- I " grafici a dispersione" in 2D e 3D mostrano la distribuzione dei dati nel nuovo spazio ridotto.

- La PCA aiuta a ridurre la dimensionalità mantenendo gran parte dell'informazione originale.

3.2.9 Correlazione tra le componenti principali e le feature originali

Per comprendere meglio l'importanza delle " componenti principali" , analizziamo la correlazione tra le feature originali e le prime due componenti principali. Il grafico mostrato è noto come " Loading Plot" , che aiuta a visualizzare il contributo delle variabili originali nelle " nuove componenti principali" .

```
        # Mostriamo la correlazione tra le componenti principali e le features
        plt.figure(figsize=(18, 15))
        for i in range(pca.n_components_):
        plt.plot([0, pca.components_[0, i]], [0, pca.components_[1, i]],
        label=adult_work.iloc[:, :-1].columns[i])
        plt.scatter(pca.components_[0, i], pca.components_[1, i], c='k')

        plt.legend(bbox_to_anchor=(1.05, 1), fontsize='x-small')
        plt.title('ADULT - LOADING GRAPH 2D')
        plt.xlabel('PC1')
        plt.ylabel('PC2')
        plt.grid()
        plt.show()

```

Listing 3.9. Grafico di correlazione tra componenti principali e feature

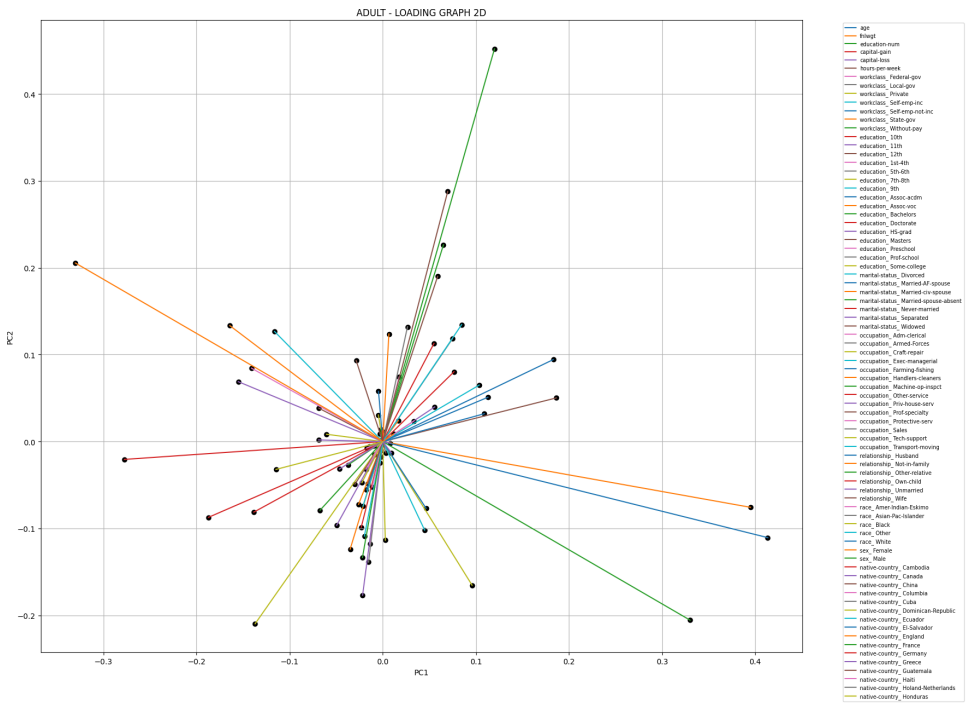

Figura 3.5. load graph 2D.

3.2.10 Osservazioni

Dal " Loading Plot" possiamo osservare:

- Le feature che hanno " vettori più lunghi" contribuiscono maggiormente alla varianza delle componenti principali.
- Le feature con angoli piccoli tra loro hanno una " forte correlazione positiva" .
- Le feature con " angoli di circa 90°" sono " poco correlate" .
- Le feature " opposte" tra loro indicano una " correlazione negativa" .

3.2.11 Barplot della correlazione tra componenti principali e feature originali

Un altro metodo per visualizzare la correlazione tra le " componenti principali" e le " feature originali" è l'uso di " barplot" . Questi grafici mostrano il contributo di ogni feature alle prime due componenti principali.

```
        # Tramite barplot mostriamo la correlazione tra le componenti
    principali e le features
        plt.figure(figsize=(18, 15))
        plt.bar(np.arange(0, pca.n_components_, 1), pca.components_[0, :-25])

        # Creiamo le etichette delle feature
        col_n = [None] * pca.n_components_
        for i in range(pca.n_components_):
        col_n[i] = adult_work.iloc[:, :-1].columns[i]

        plt.xticks(ticks=np.arange(pca.n_components_), labels=col_n, rotation
    =90, fontsize='x-small')
        plt.title('PC1')
        plt.grid()
        plt.show()

```

Listing 3.10. Barplot della correlazione tra le feature e PC1

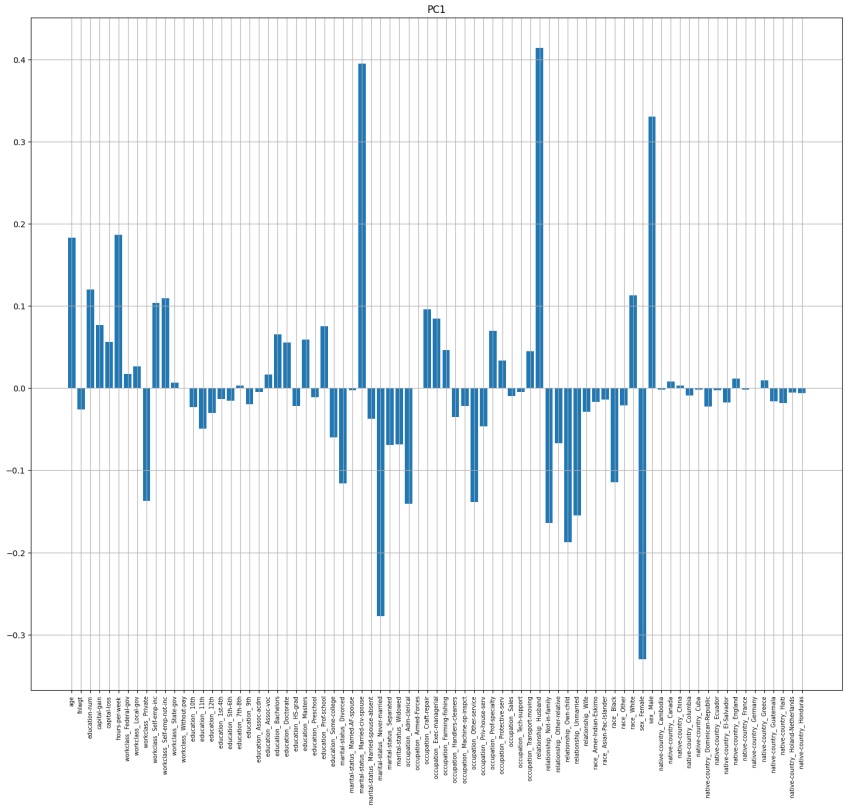

Figura 3.6. correlazione tra features e PC1

```
        plt.figure(figsize=(18, 15))
        plt.bar(np.arange(0, pca.n_components_, 1), pca.components_[1, :-25])

        # Creiamo le etichette delle feature
        col_n = [None] * pca.n_components_
        for i in range(pca.n_components_):
        col_n[i] = adult_work.iloc[:, :-1].columns[i]

        plt.xticks(ticks=np.arange(pca.n_components_), labels=col_n, rotation
    =90, fontsize='x-small')
        plt.title('PC2')
        plt.grid()
        plt.show()

```

Listing 3.11. Barplot della correlazione tra le feature e PC2

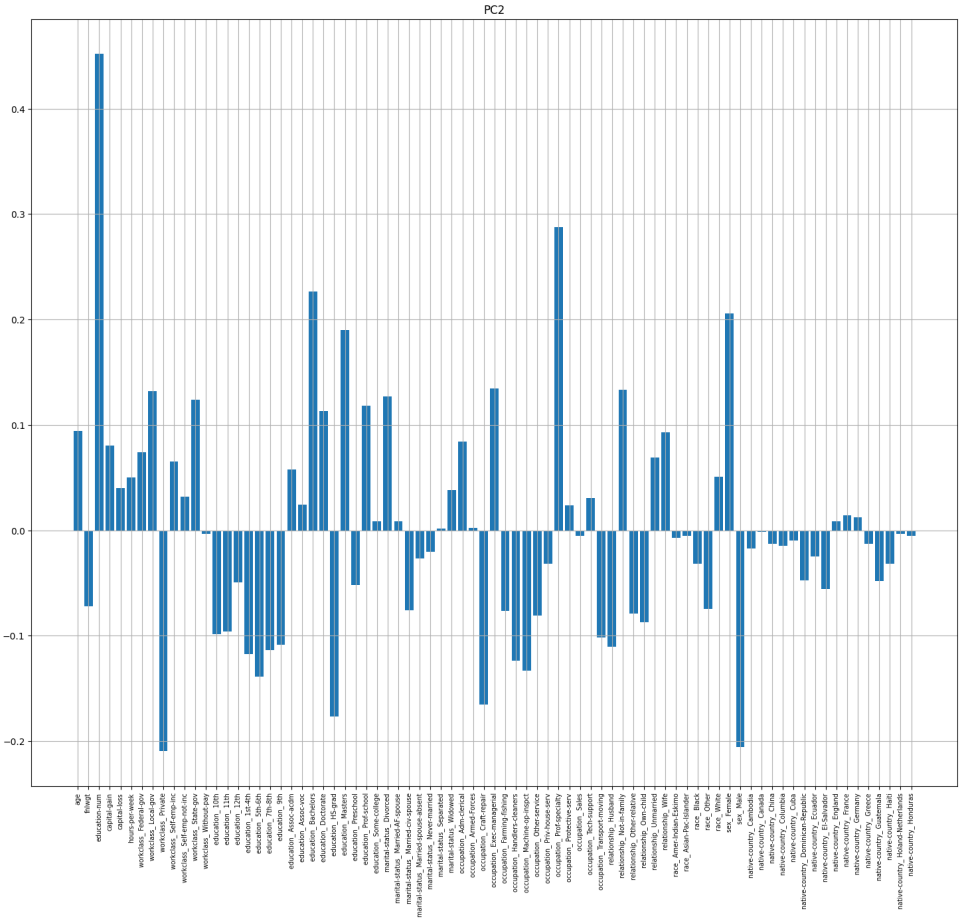

Figura 3.7. correlazione tra features e PC2

3.2.12 Osservazioni

Dall'analisi dei " barplot" possiamo osservare:

- Le feature con barre più alte hanno un " maggiore contributo" nella formazione della componente principale.

- PC1 e PC2 possono catturare informazioni " diverse" , quindi una feature potrebbe avere un'importanza maggiore in una componente rispetto all'altra.
- Questo tipo di visualizzazione è utile per selezionare le " feature più rilevanti" nel dataset.

3.3 Multiple Discriminant Analysis (MDA)

3.3.1 Introduzione

Fisher Discriminant Analysis (FDA) è una tecnica di riduzione della dimensionalità e classificazione che proietta i dati suddivisi in $c \in \mathbb{N}$ classi in uno spazio di dimensione:

$$1 \leq m \leq (c-1)$$

L'obiettivo è trovare un asse che massimizzi la distanza tra i centri delle classi e minimizzi la varianza all'interno delle classi. Utilizzare la FDA equivale ad applicare una trasformazione che preserva la classificazione dei dati. MDA è particolarmente utile quando si lavora con dataset ad alta dimensione e si desidera ridurre la dimensionalità mantenendo la separazione tra le classi per migliorare la classificazione.

3.3.2 Differenze tra PCA e MDA

Principal Component Analysis (PCA) e **Multiple Discriminant Analysis (MDA)** (nota anche come *Fisher Discriminant Analysis, FDA*) sono entrambe tecniche di riduzione della dimensionalità, ma con obiettivi e approcci differenti.

PCA

- **Obiettivo:** Ridurre la dimensionalità massimizzando la varianza dei dati.
- **Metodo:** Trova le componenti principali che spiegano la maggior parte della varianza nei dati.
- **Uso:** Adatto per l'analisi esplorativa dei dati e la compressione dei dati.
- **Unsupervised learning:** Non utilizza informazione sulle etichette delle classi.

MDA (FDA)

- **Obiettivo:** Ridurre la dimensionalità massimizzando la separazione tra le classi.
- **Metodo:** Trova la proiezione lineare che massimizza la distanza tra i centri delle classi e minimizza la varianza all'interno delle classi.
- **Uso:** Adatto per la classificazione e la riduzione supervisionata della dimensionalità.
- **Supervised learning:** Utilizza informazione sulle etichette delle classi.

3.3.3 Procedimento

Utilizziamo un campione $S = \{x_1, \ldots, x_k\} \subset \mathbb{R}^d$ e un vettore classe $c = \{c_1, \ldots, c_3\}$ (supponiamo di avere 3 classi).

1. **Calcolo dei Vettori Medi**
   Calcoliamo il vettore medio per ciascuna classe:

   $$\mu_i = \frac{1}{N_i} \sum_{x_k \in c_i} x_k$$

   dove N_i è il numero di campioni della classe c_i mentre x_k sono i campioni di classe c_i. Alla fine di questa operazione avremo c vettori medi distinti.

2. **Matrice di Scatter Within-Class (S_W)**
   Calcoliamo la matrice di scatter all'interno delle classi:

   $$S_W = \sum_{i=1}^{c} \sum_{x_k \in c_i} (x_k - \mu_i)(x_k - \mu_i)^T$$

   Questa matrice S_W somma tra loro le matrici ottenute centrando tutti i campioni con le loro rispettive medie e moltiplicandoli tra loro. È simile a una parte della formula per calcolare la matrice di covarianza:

   $$S = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu)(x_i - \mu)^T$$

3. **Matrice di Scatter Between-Class (S_B)**
   Calcoliamo la matrice di scatter tra le classi:

   $$S_B = \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^T$$

   dove μ è il vettore medio globale ottenuto facendo:

   $$\mu = \frac{1}{k} \sum_{i=1}^{k} x_i$$

4. **Autovettori e Autovalori**
   Risolviamo il problema degli autovalori per $S_W^{-1} S_B$ per trovare le direzioni ottimali e gli autovalori (importanza delle direzioni):

   $$S_W^{-1} S_B v = \lambda v$$

   dove λ sono gli autovalori e v sono gli autovettori.

5. **Selezione delle Componenti Principali**
   Ordiniamo gli autovettori in base ai rispettivi autovalori in ordine decrescente e selezioniamo i primi n autovettori per formare la matrice di trasformazione W (uguale alla PCA).

6. **Proiezione dei Dati**
   Ora possiamo proiettare i dati originali nello spazio delle componenti principali:

$$Z = XW$$

   dove Z sono i dati proiettati, X la matrice dei dati originali e W la matrice dei primi n autovettori.

3.3.4 Visualizzazione della separazione tra classi con MDA

Dopo aver applicato la " Multiple Discriminant Analysis (MDA)" , possiamo visualizzare la separazione tra le classi ottenuta attraverso la trasformazione MDA e confrontarla con la PCA.

```
        # Mostriamo la separazione tra classi dell'MDA
        mda = MDA()
        mda.fit(X_train, y_train)
        Zmda = mda.transform(x)

        zero = np.zeros(Zmda.shape[0])  # Creiamo un asse orizzontale per la visualizzazione

        fig1, axs1 = plt.subplots(1, 2, figsize=(14, 10))

        # Scatter plot della MDA
        axs1[0].scatter(Zmda[:, 0], zero, c=y, cmap=cmap, alpha=0.50)
        handles = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=cmap(0), markersize=10, label='<=50K'),
        plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=cmap(1), markersize=10, label='>50K')]

        axs1[0].legend(handles=handles, title='income')
        axs1[0].set_title('MDA')
        axs1[0].grid()

        # Scatter plot della PCA per confronto
        axs1[1].set_xticks(np.arange(Zmda[:, 0].min()-1, Zmda[:, 0].max()+1))
        axs1[1].scatter(Zpca[:, 0], Zpca[:, 1], c=y, cmap=cmap, alpha=0.50)

        axs1[1].legend(handles=handles, title='income')
        axs1[1].set_title('PCA')
        axs1[1].grid()

```

Listing 3.12. Applicazione della MDA e visualizzazione delle classi

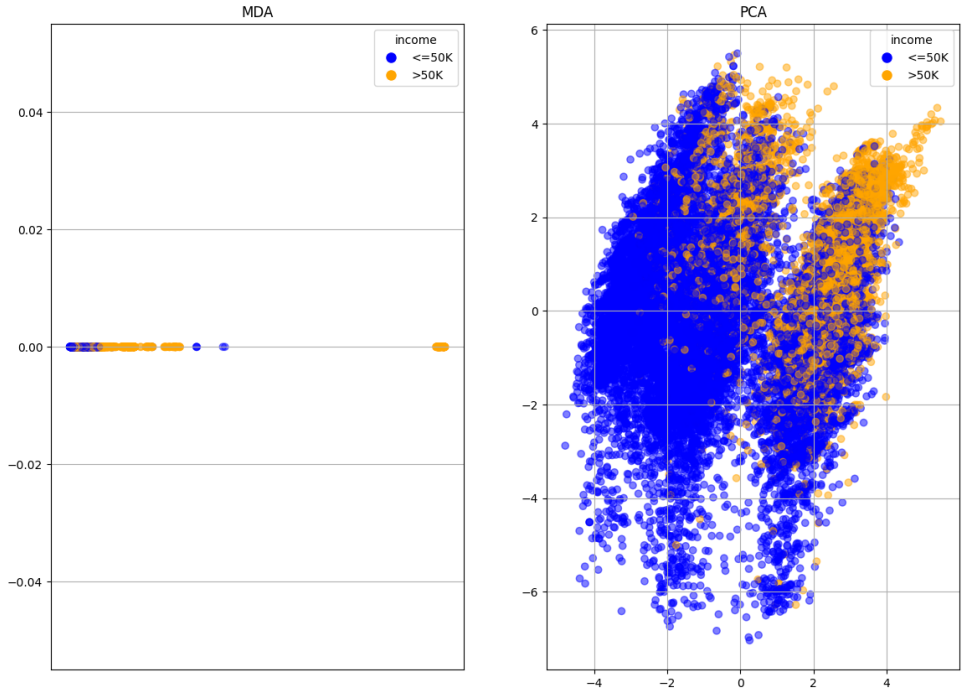

Figura 3.8. Separazione classi

3.3.5 Osservazioni

Dalla visualizzazione della MDA e della PCA possiamo osservare:

- La " MDA" proietta i dati in modo da massimizzare la separazione tra le classi, portando i punti su un asse 1D che distingue nettamente le classi.
- La " PCA" , invece, non tiene conto delle etichette delle classi e ottimizza la varianza totale, quindi la separazione tra le classi potrebbe non essere ottimale.
- La MDA è particolarmente utile per compiti di classificazione, mentre la PCA è più adatta per la compressione dei dati o la visualizzazione esplorativa.

3.3.6 Applicazione della MDA sui dati trasformati con PCA

Dopo aver trasformato i dati con " PCA" , possiamo applicare la " MDA" per verificare se la riduzione della dimensionalità con PCA ha migliorato o peggiorato la separazione tra le classi.

```
        # Applichiamo la MDA al campione della PCA
        mda_pca = MDA()
        mda_pca.fit(Zpca, y)
        Zmda_pca = mda_pca.transform(Zpca)

        fig1, axs1 = plt.subplots(1, 2, figsize=(14, 5))

        # Scatter plot della MDA senza PCA
        axs1[0].scatter(Zmda[:, 0].flatten(), y, c=y, cmap=cmap, alpha=0.15)
        axs1[0].set_title('MDA (senza PCA)')
        axs1[0].grid()

        # Scatter plot della MDA applicata dopo PCA
        axs1[1].set_xticks(np.arange(Zmda[:, 0].min(), Zmda[:, 0].max()))
        axs1[1].scatter(Zmda_pca.flatten(), y, c=y, cmap=cmap, alpha=0.15)
        axs1[1].set_title('MDA (con PCA)')
        axs1[1].grid()

```

Listing 3.13. Applicazione della MDA ai dati trasformati con PCA

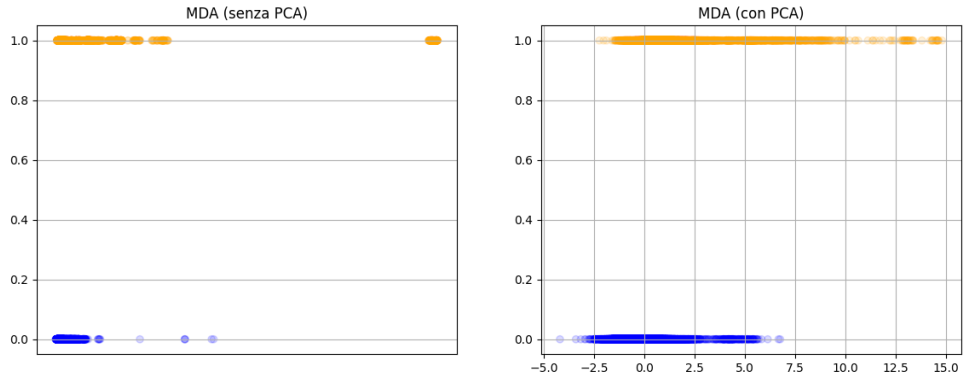

Figura 3.9. Confronto classi

3.3.7 Osservazioni

Confrontando i due scatter plot ottenuti:

- " MDA senza PCA" utilizza i dati originali e genera una separazione tra le classi basata esclusivamente sulla " discriminazione lineare" .
- " MDA dopo PCA" mostra l'effetto della riduzione di dimensionalità sui dati prima dell'analisi discriminante.
- Se la separazione delle classi nella MDA dopo PCA è più chiara e netta, significa che " PCA ha migliorato l'efficacia della MDA" , riducendo il rumore nei dati.
- Se invece la separazione risulta meno marcata, PCA potrebbe aver eliminato informazioni discriminanti utili.

3.4 Linear Discriminant Analysis (LDA)

3.4.1 Introduzione

Linear Discriminant Analysis (LDA) è una tecnica di classificazione supervisionata che cerca di trovare una combinazione lineare delle caratteristiche che meglio separa le classi (analogo con la MDA).

La novità che introduce nel processo di predizione è che LDA calcola le probabilità **a priori** delle classi e le probabilità condizionate dei dati rispetto a ciascuna classe. Utilizzando il teorema di Bayes, LDA stima la probabilità **posteriore** di ogni classe data una nuova osservazione e assegna la classe con la massima probabilità posteriore, ottimizzando così la separazione e la classificazione dei dati.

3.4.2 Procedimento

Iniziamo con il mostrare il teorema di Bayes:

$$P(Y = k \mid X = x) = \frac{P(X = x \mid Y = k)P(Y = k)}{P(X = x)}$$

1. **Calcolo delle Probabilità A Priori**
   Sia $\omega = \{\omega_1, \ldots, \omega_h\}$ il vettore contenente le diverse classi. Calcoliamo la probabilità a priori $P(\omega_i)$ di ciascuna classe:

   $$P(\omega_i) = \frac{N_i}{N}$$

   dove N_i è il numero di campioni nella classe ω_i e N è il numero totale di campioni.

2. **Calcolo delle Probabilità Condizionate**
   Per ciascuna classe, calcoliamo la probabilità condizionata $P(x \mid \omega_i)$. Poiché LDA assume una distribuzione normale multivariata per ciascuna classe, si utilizza la funzione di densità di probabilità della distribuzione normale:

   $$P(x \mid \omega_i) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_i)^T \Sigma^{-1}(x - \mu_i)\right)$$

   dove μ_i è il vettore medio della classe ω_i, Σ è la matrice di covarianza (che è la stessa per tutte le classi in LDA) e d è la dimensione dei dati $x \in \mathbb{R}^d$.

3. **Calcolo della Probabilità Posteriore**
   Utilizziamo il teorema di Bayes per calcolare la probabilità posteriore $P(\omega_i \mid x)$:

   $$P(\omega_i \mid x) \propto P(x \mid \omega_i)P(\omega_i)$$

   Normalizziamo per assicurarci che le probabilità posteriori sommino a 1:

   $$P(\omega_i \mid x) = \frac{P(x \mid \omega_i)P(\omega_i)}{\sum_j P(x \mid \omega_j)P(\omega_j)}$$

   Ad esempio, se la probabilità a priori della classe 1 è 0.84 e quella della classe 0 è 0.16, allora la somma delle probabilità posteriori deve essere $0.84 + 0.16 = 1$.

4. **Decisione di Classificazione**
   Assegniamo la classe che massimizza la probabilità posteriore:

$$\hat{\omega} = \arg\max P(\omega_i \mid x)$$

   ovvero assegniamo la classe con la probabilità maggiore di appartenenza a quel campione.

3.4.3 Valutazione delle Prestazioni del Modello LDA

Dopo aver addestrato il modello " Linear Discriminant Analysis (LDA)" , calcoliamo l'accuratezza della classificazione sui dati di training e di test.

```
        # Calcolo la precisione con cui il metodo LDA predice i dati (senza
    PCA)
        lda = LDA()
        lda.fit(X_train, y_train)

        # Predizioni e probabilita per ciascuna classe
        y_pred = lda.predict(X_test)
        y_pred_proba = lda.predict_proba(X_test)

        # Creiamo un DataFrame con i risultati
        y_pred_df = pd.DataFrame({'Pred. Class': y_pred,
            'P(Class 0) - %': np.round(y_pred_proba[:, 0] * 100, decimals=2),
            'P(Class 1) - %': np.round(y_pred_proba[:, 1] * 100, decimals=2)})

        # Creiamo un DataFrame con le metriche di accuratezza
        scores_dict = {'Training Set': np.round(lda.score(X_train, y_train) *
    100, decimals=2),
            'Test Set': np.round(lda.score(X_test, y_test) * 100, decimals=2)}

        scores = pd.DataFrame(scores_dict, index=['Accuracy'])

        display(scores)
        display(y_pred_df)

```

Listing 3.14. Calcolo dell'accuratezza del modello LDA

3.4.4 Risultati delle Prestazioni del Modello

L'accuratezza del modello LDA calcolata sia sul " training set" che sul " test set" è riportata nella seguente tabella:

Metric	Valore (%)
Training Set Accuracy	83.79
Test Set Accuracy	83.78

3.4.5 Predizioni e Probabilità di Classe

La tabella seguente mostra le prime righe delle predizioni del modello LDA, con le rispettive probabilità assegnate a ciascuna classe:

Indice	Pred. Class	P(Class 0) - %	P(Class 1) - %
0	0	64.07	35.93
1	0	85.07	14.93
2	0	63.54	36.46
3	0	81.56	18.44
4	0	90.70	9.30
⋮	⋮	⋮	⋮
15076	0	69.66	30.34
15077	0	64.27	35.73
15078	1	40.30	59.70
15079	0	89.76	10.24
15080	0	99.20	0.80

3.4.6 Osservazioni

- L'accuratezza del modello LDA è di circa " 83.79% sul training set" e " 83.78% sul test set" , indicando che il modello generalizza bene.
- Le predizioni mostrano che la maggior parte dei campioni è classificata come " Classe 0" , con alcune assegnazioni alla " Classe 1" .
- Le probabilità associate alle classi variano, indicando il grado di certezza della classificazione.
- Un'osservazione interessante è che alcuni campioni hanno una probabilità " superiore al 90%" di appartenere a una classe specifica.

3.4.7 Valutazione delle Prestazioni del Modello LDA con PCA

Ora applichiamo il modello " Linear Discriminant Analysis (LDA)" dopo aver trasformato i dati con " Principal Component Analysis (PCA)" . Calcoliamo l'accuratezza della classificazione sui dati di training e di test.

```
        # Calcolo la precisione con cui il metodo LDA predice i dati (con PCA)
        lda_pca = LDA()
        lda_pca.fit(pca_adult, y_train)

        # Predizioni e probabilita per ciascuna classe
        y_pred = lda_pca.predict(pca.transform(scaler_X.transform(X_test)))
        y_pred_proba = lda_pca.predict_proba(pca.transform(scaler_X.transform(
    X_test)))

        # Creiamo un DataFrame con i risultati
        y_pred_df = pd.DataFrame({'Pred. Class': y_pred,
            'P(Class 0) - %': np.round(y_pred_proba[:, 0] * 100, decimals=2),
            'P(Class 1) - %': np.round(y_pred_proba[:, 1] * 100, decimals=2)})

        # Creiamo un DataFrame con le metriche di accuratezza
        scores_dict = {'Training Set': np.round(lda_pca.score(pca_adult,
    y_train) * 100, decimals=2),
            'Test Set': np.round(lda_pca.score(pca.transform(scaler_X.
    transform(X_test)), y_test) * 100, decimals=2)}
```

```
17
18          scores = pd.DataFrame(scores_dict, index=['Accuracy'])
19
20          display(scores)
21          display(y_pred_df)
22
```

Listing 3.15. Calcolo dell'accuratezza del modello LDA dopo PCA

3.4.8 Risultati delle Prestazioni del Modello

L'accuratezza del modello LDA calcolata sui dati trasformati con PCA è la seguente:

Metric	Valore (%)
Training Set Accuracy	83.52
Test Set Accuracy	83.51

3.4.9 Predizioni e Probabilità di Classe

La tabella seguente mostra alcune delle predizioni del modello LDA applicato ai dati trasformati con PCA, insieme alle rispettive probabilità di appartenenza a ciascuna classe:

Indice	Pred. Class	P(Class 0) - %	P(Class 1) - %
0	0	51.79	48.21
1	0	78.36	21.64
2	0	68.79	31.21
3	0	78.25	21.75
4	0	83.97	16.03
⋮	⋮	⋮	⋮
15076	0	71.38	28.62
15077	1	41.91	58.09
15078	1	40.30	59.70
15079	0	88.85	11.15

3.4.10 Osservazioni

- L'accuratezza del modello " LDA con PCA" è leggermente inferiore rispetto al LDA senza PCA, con una riduzione minima della precisione nel test set.
- Le predizioni mostrano che la probabilità tra le classi è più bilanciata rispetto al caso senza PCA, con più campioni che vengono classificati nella " Classe 1" .
- Questo potrebbe indicare che la " PCA ha ridotto la dimensionalità" in modo da migliorare la discriminazione delle classi in alcuni casi.
- Tuttavia, è importante valutare il " trade-off" tra riduzione della dimensionalità e perdita di informazioni discriminanti.

3.5 SVM Lineare

3.5.1 Introduzione

Il problema della " SVM lineare" consiste nel trovare un iperpiano in $\mathbb{R}^n$ che separi i punti dati in due classi, una positiva e una negativa.

Un iperpiano è definito come l'insieme di punti che soddisfano l'equazione:

$$w \cdot x + b = 0$$

dove $w \in \mathbb{R}^n$ è un vettore non nullo normale all'iperpiano e $b \in \mathbb{R}$ è uno scalare. L'iperpiano divide lo spazio in due semispazi.

3.5.2 Procedimento

Caso separabile

Si assume che esista un iperpiano che separi perfettamente i punti di training in due popolazioni di punti etichettati positivamente e negativamente. Tuttavia, ci sono infiniti iperpiani che potrebbero separare i dati.

L'obiettivo della " SVM" è trovare l'iperpiano che massimizza il margine, ovvero la distanza tra l'iperpiano e il punto dati più vicino. Questo iperpiano è chiamato **margine massimo** e la distanza tra l'iperpiano e i punti dati più vicini è chiamata " margine geometrico" . I punti dati che si trovano sul margine sono chiamati " vettori di supporto" .

Indichiamo l'insieme dei vettori di training con:

$$X = \{x_1, \ldots, x_n\}, \quad Y = \{y_1, \ldots, y_n\}$$

La funzione obiettivo diventa:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

soggetto a:

$$y_i(w \cdot x_i + b) \geq 1, \quad \forall i$$

dove w è il vettore dei pesi, b il bias, x_i è il vettore dei punti e y_i indica la classe.

Successivamente al " calcolo del Lagrangiano" , si ottiene il seguente problema duale:

$$\begin{cases} \min_\alpha \frac{1}{2}\alpha^T Q \alpha - \sum_{i=1}^n \alpha_i \\ \sum_{i=1}^n \alpha_i y_i = 0 \\ \alpha_i \geq 0, \quad \forall i = 1, \ldots, n \end{cases}$$

dove $\alpha \in \mathbb{R}^n$ e la matrice $Q \in \mathbb{R}^{n \times n}$ è definita come:

$$Q = (q_{ij})_{i,j=1,\ldots,n} = (y_i y_j x_i^T x_j)_{i,j=1,\ldots,n}$$

A questo punto possiamo scrivere la " funzione di decisione" , che indica la classe di ciascun nuovo vettore:

$$f(x) = \text{sgn}(w \cdot x + b) = \text{sgn}\left(\sum_{i=1}^{n} \alpha_i y_i (x_i \cdot x) + b\right)$$

con:

$$b = y_i - \sum_{j=1}^{n} \alpha_j y_j (x_j \cdot x_i)$$

Caso non separabile

Nel caso non separabile, non esiste un iperpiano che possa separare perfettamente i punti di training.

In questo caso, le " SVM" cercano di trovare l'iperpiano che massimizza il margine e allo stesso tempo minimizza l'errore di classificazione sui punti di training. Per ottenere ciò, le SVM introducono le " variabili slack" ξ_i, che misurano il grado di violazione del margine consentita per ciascun punto dato.

Il problema primale diventa quindi:

$$\begin{cases} \min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \\ y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \forall i = 1, \ldots, n \\ \xi_i \geq 0, \quad \forall i \end{cases}$$

Il parametro $C \in \mathbb{R}^+$ è un **parametro di regolarizzazione** che caratterizza il rilassamento delle condizioni per il margine:

- $C \to 0$ aumenta la " "morbidezza"" del margine, permettendo ai vettori di superarlo illimitatamente.
- $C \to \infty$ aumenta la " "durezza"" del margine, penalizzando i vettori x_i di superarlo in modo inaccettabile.

3.5.3 Valutazione delle Prestazioni del Metodo SVM

Testiamo ora la " Support Vector Machine (SVM)" con due configurazioni: - " SVM Hard" con C molto alto ($C = 10^{10}$) per un margine rigido. - " SVM Soft" con C più basso ($C = 10^{1}$) per un margine più flessibile.

Metodo SVM Hard

```
        # Calcolo la precisione del metodo SVM hard
        C_hard = 10**10
        loss = 'squared_hinge'
        dual = False

        lsvm_hard = LinearSVC(C=C_hard, loss=loss, dual=dual, random_state=
    random_seed)
        lsvm_hard_pca = LinearSVC(C=C_hard, loss=loss, dual=dual, random_state
    =random_seed)

        lsvm_hard.fit(X_train, y_train)
        lsvm_hard_pca.fit(pca_adult, y_train)

        df_lsvm_hard = pd.DataFrame({'acc. (no PCA)': [np.round(lsvm_hard.
    score(X_train, y_train)*100, decimals=2),
            np.round(lsvm_hard.score(X_test, y_test)*100, decimals=2)],
            'acc. (PCA)': [np.round(lsvm_hard_pca.score(pca_adult, y_train)
    *100, decimals=2),
            np.round(lsvm_hard_pca.score(pca.transform(scaler_X.transform(
    X_test)), y_test)*100, decimals=2)]},
        index=['training', 'test'])

        display(df_lsvm_hard)

```

Listing 3.16. Calcolo della precisione del metodo SVM hard

Set	Acc. (no PCA)	Acc. (PCA)
Training	79.44	83.79
Test	78.93	84.09

Metodo SVM Soft

```
        # Calcolo la precisione del metodo SVM soft
        C_soft = 10**1
        loss = 'squared_hinge'
        dual = False

        lsvm_soft = LinearSVC(C=C_soft, loss=loss, dual=dual, random_state=
    random_seed)
        lsvm_soft_pca = LinearSVC(C=C_soft, loss=loss, dual=dual, random_state
    =random_seed)

        lsvm_soft.fit(X_train, y_train)
        lsvm_soft_pca.fit(pca_adult, y_train)

        df_lsvm_soft = pd.DataFrame({'acc. (no PCA)': [np.round(lsvm_soft.
    score(X_train, y_train)*100, decimals=2),
            np.round(lsvm_soft.score(X_test, y_test)*100, decimals=2)],
            'acc. (PCA)': [np.round(lsvm_soft_pca.score(pca_adult, y_train)
    *100, decimals=2),
            np.round(lsvm_soft_pca.score(pca.transform(scaler_X.transform(
    X_test)), y_test)*100, decimals=2)]},
        index=['training', 'test'])

        display(df_lsvm_soft)

```

Listing 3.17. Calcolo della precisione del metodo SVM soft

Set	Acc. (no PCA)	Acc. (PCA)
Training	79.44	83.79
Test	78.93	84.09

3.5.4 Osservazioni

- Sia il metodo " hard" che il metodo " soft" ottengono la stessa accuratezza.
- Applicando il metodo " SVM" ai dati trasformati con " PCA" , si ottiene un incremento di accuratezza di circa " 5%" .
- Questo dimostra che la riduzione della dimensionalità con " PCA" migliora la separabilità dei dati per il classificatore lineare.

3.6 SVM Non Lineare

3.6.1 Introduzione

I " Support Vector Machines (SVM) non lineari" sono un'estensione degli " SVM lineari" che possono separare dati non linearmente separabili utilizzando il *Kernel Trick*.

Questo metodo trasforma i dati in uno spazio di dimensioni superiori dove un iperpiano lineare può separare le classi.

3.6.2 Procedimento

Sia X lo spazio dei dati, definiamo un funzionale:

$$\phi : X \to H$$

dove H è uno " spazio di Hilbert" . Definiamo ora un " Kernel" come:

$$K(x, x') = \langle \phi(x), \phi(x') \rangle$$

dove x, x' sono vettori.

1. Mappatura in uno Spazio di Dimensioni Superiori

I dati originali vengono " mappati in uno spazio di Hilbert" di dimensioni superiori utilizzando il funzionale $\phi(x)$.

In questo spazio, è più probabile che i dati diventino linearmente separabili. Siano $X = \{x_1, \ldots, x_n\}$ i vettori dei dati e delle classi, allora il vettore X diventa:

$$X^\phi = \{\phi(x_1), \ldots, \phi(x_n)\} = \{\varphi_1, \ldots, \varphi_n\}$$

2. Problemi con la Soluzione Precedente

Come già mostrato per " SVM lineare" , bisogna risolvere lo stesso problema di ottimizzazione, ma questa volta utilizzando X^ϕ. Le soluzioni risultano essere:

$$w^* = \sum_{i=1}^{n} \alpha_i y_i \varphi_i$$

$$b^* = \frac{1}{n} \sum_{i=1}^{n} (y_i - \langle w^*, \varphi_i \rangle)$$

Ora possiamo scrivere la nostra funzione di decisione:

$$f(x) = \text{sign}\left(\langle w^*, \phi(x) \rangle + b^*\right)$$

$$= \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i \langle \varphi_i, \varphi \rangle + \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{n} \alpha_j y_j \langle \varphi_j, \varphi_i \rangle \right)\right)$$

Nella formula sopra, il grosso dei calcoli è richiesto per i " prodotti scalari" $\langle \varphi_i, \varphi \rangle$ e $\langle \varphi_j, \varphi_i \rangle$, che vengono svolti in $\mathbb{R}^m$, dove " presumibilmente $m \gg n$" .

Dobbiamo quindi trovare un modo per " semplificare i calcoli" .

3. Scelta del Kernel

Come definito in precedenza, un " kernel" è una funzione che calcola il prodotto scalare tra due vettori nello spazio di dimensioni superiori " senza" calcolare esplicitamente $\phi(x)$.

Mostriamo ora alcuni esempi di kernel più utilizzati:

- " Kernel Lineare" :
$$K(x_i, x_j) = \langle x_i, x_j \rangle$$
- " Kernel Polinomiale" :
$$K(x_i, x_j) = (\gamma \langle x_i, x_j \rangle + c)^d$$
- " Kernel Radiale di Base (RBF)" :
$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$
- " Kernel Sigmoide" :
$$K(x_i, x_j) = \tanh(\gamma \langle x_i, x_j \rangle + c)$$

Una volta scelto il " kernel" , si sostituisce al prodotto scalare e si ottiene la funzione di decisione semplificata.

3.6.3 Test con SVM Non Lineare

Testiamo ora la " Support Vector Machine (SVM) non lineare" con due tipi di kernel: - " Kernel Sigmoide" : più veloce nel calcolo ma meno preciso. - " Kernel Radiale di Base (RBF)" : più preciso ma più lento nei calcoli.

Test con Kernel Sigmoide

```
1        # Testo il primo algoritmo con un kernel sigmoide, tempo medio 29
     secondi
2        ker_tanh = 'sigmoid'
3        gamma_tanh = 'auto'
4
5        C = 100
6
7        svm_tanh_1 = SVC(C=C, kernel=ker_tanh, gamma=gamma_tanh, random_state=
     random_seed)
8        svm_tanh_2 = SVC(C=C, kernel=ker_tanh, gamma=gamma_tanh, random_state=
     random_seed)
9
10       svm_tanh_1.fit(X_train, y_train)
11       svm_tanh_2.fit(pca_adult, y_train)
12
13       df_svm_tanh = pd.DataFrame({'acc. (NO PCA)': [np.round(svm_tanh_1.
     score(X_train, y_train)*100, decimals=2),
14            np.round(svm_tanh_1.score(X_test, y_test)*100, decimals=2)],
15            'acc. (PCA)': [np.round(svm_tanh_2.score(pca_adult, y_train)*100,
     decimals=2),
16            np.round(svm_tanh_2.score(pca.transform(scaler_X.transform(X_test)
     ), y_test)*100, decimals=2)]},
17       index=['training', 'test'])
18
19       display(df_svm_tanh)
20
```

Listing 3.18. Test con Kernel Sigmoide

Set	Acc. (NO PCA)	Acc. (PCA)
Training	75.37	77.21
Test	74.84	77.60

Test con Kernel RBF

```
        # Testo il secondo algoritmo con un kernel RBF, tempo medio 3 minuti 38 secondi
        ker_rbf = 'rbf'
        gamma_rbf = 'auto'

        C = 100

        svm_rbf_1 = SVC(C=C, kernel=ker_rbf, gamma=gamma_rbf, random_state=random_seed)
        svm_rbf_2 = SVC(C=C, kernel=ker_rbf, gamma=gamma_rbf, random_state=random_seed)

        svm_rbf_1.fit(X_train, y_train)
        svm_rbf_2.fit(pca_adult, y_train)

        df_svm_rbf = pd.DataFrame({'acc. (NO PCA)': [np.round(svm_rbf_1.score(X_train, y_train)*100, decimals=2),
                np.round(svm_rbf_1.score(X_test, y_test)*100, decimals=2)],
                'acc. (PCA)': [np.round(svm_rbf_2.score(pca_adult, y_train)*100, decimals=2),
                np.round(svm_rbf_2.score(pca.transform(scaler_X.transform(X_test)), y_test)*100, decimals=2)]},
        index=['training', 'test'])

        display(df_svm_rbf)

```

Listing 3.19. Test con Kernel RBF

Set	Acc. (NO PCA)	Acc. (PCA)
Training	99.99	90.11
Test	74.19	82.74

3.6.4 Osservazioni

Per la scelta del " kernel migliore" , bisogna cercare di fare un " bilanciamento tra accuratezza ottenuta e tempo di calcolo" :

- Il " kernel Sigmoide" è più veloce, ma l'accuratezza non migliora significativamente con la PCA.
- Il " kernel RBF" è molto più preciso dopo l'applicazione della " PCA" (incremento dell'8.5% circa), ma ha un " tempo di calcolo più lungo" .

3.6.5 SVM con Kernel RBF - Decision Boundary

In questa sezione testiamo la " SVM con Kernel RBF" considerando " solo le prime due componenti principali" della PCA e tracciamo la *decision boundary*.

Implementazione del Modello

```
1       # Ora disegno i grafici per il kernel RBF, considerando solo le prime
    due componenti principali, tempo medio 2 minuti 8 secondi
2       ker_rbf = 'rbf'
3       gamma_rbf = 'auto'
4
5       C = 100
6
7       svm = SVC(C=C, kernel=ker_rbf, gamma=gamma_rbf, random_state=
    random_seed)
8
9       svm.fit(pca_adult[:, :2], y_train)
10
11      df_svm_rbf = pd.DataFrame({'acc. (PCA)': [np.round(svm.score(pca_adult
    [:, :2], y_train)*100, decimals=2),
12          np.round(svm.score(pca.transform(scaler_X.transform(X_test))[:,
    :2], y_test)*100, decimals=2)]},
13      index=['training', 'test'])
14
15      display(df_svm_rbf)
16
```

Listing 3.20. Training della SVM con Kernel RBF

Set	Acc. (PCA)
Training	81.97
Test	81.77

Visualizzazione della Decision Boundary

```
        minx = min(pca_adult[:, 0].min(), pca_adult[:, 1].min()) - 1
        maxx = max(pca_adult[:, 0].max(), pca_adult[:, 1].max()) + 1

        mesh_points = 200

        xx1, xx2 = np.meshgrid(np.linspace(minx, maxx, mesh_points), np.
    linspace(minx, maxx, mesh_points))

        XX = np.concatenate([xx1.reshape(mesh_points "
2, 1), xx2.reshape(mesh_points "
2, 1)], axis=1)

        ZZ_1 = svm.decision_function(XX)
        zz_1 = ZZ_1.reshape(mesh_points, mesh_points)

        plt.figure(figsize=(12, 12))
        plt.scatter(pca_adult[:, 0], pca_adult[:, 1], c=y_train, marker='x',
    cmap=cmap, alpha=0.25)
        plt.scatter(pca.transform(scaler_X.transform(X_test))[:, 0], pca.
    transform(scaler_X.transform(X_test))[:, 1],
        c=y_test, marker='o', cmap=cmap, alpha=0.25)
        plt.contour(xx1, xx2, zz_1, [0], colors='red')
        plt.contour(xx1, xx2, zz_1, [-1,1], colors=['green', 'yellow'])
        plt.title('SVM RBF - DECISION BOUNDARY')
        plt.show()

```

Listing 3.21. Grafico della Decision Boundary

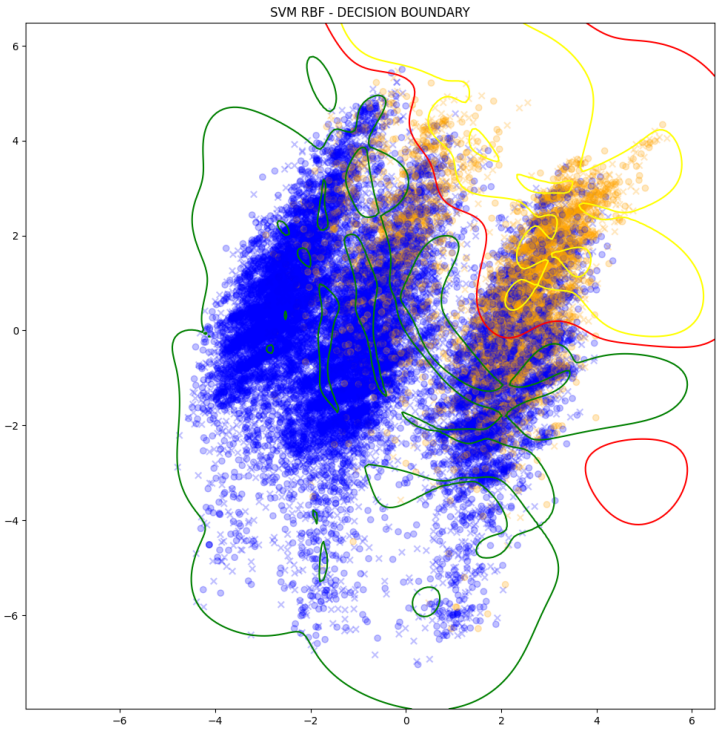

Figura 3.10. SVM RBF

Osservazioni

- La " boundary di decisione" evidenzia come la " SVM RBF" sia in grado di separare le classi in un contesto " non lineare" .
- Si nota come i dati siano " ben separati nello spazio delle prime due componenti principali" della PCA.
- L'accuratezza sul " training set" e sul " test set" sono simili, indicando che il modello non sta overfittando.

3.6.6 Valutazione della SVM con Kernel RBF

Dopo aver addestrato il modello SVM con kernel RBF, confrontiamo le prestazioni in termini di precisione, richiamo e F1-score, sia per il modello con " PCA" sia per quello senza.

Calcolo delle Metriche di Classificazione

```
        y_pred = svm_rbf_1.predict(X_test)
        y_pred_pca = svm_rbf_2.predict(pca.transform(scaler_X.transform(X_test
    )))

        precision = np.round(precision_score(y_test, y_pred)*100, decimals=2)
        recall = np.round(recall_score(y_test, y_pred)*100, decimals=2)
        fbeta = np.round(fbeta_score(y_test, y_pred, beta=1.0)*100, decimals
    =2)

        precision_pca = np.round(precision_score(y_test, y_pred_pca)*100,
    decimals=2)
        recall_pca = np.round(recall_score(y_test, y_pred_pca)*100, decimals
    =2)
        fbeta_pca = np.round(fbeta_score(y_test, y_pred_pca, beta=1.0)*100,
    decimals=2)

        df_scores = pd.DataFrame({'Precision': [precision, precision_pca],
            'Recall': [recall, recall_pca],
            'F1': [fbeta, fbeta_pca]},
        index=['NO PCA', 'PCA'])

        display(df_scores)

```

Listing 3.22. Calcolo delle metriche per SVM RBF

Risultati delle Metriche di Classificazione

Modello	Precision (%)	Recall (%)	F1-score (%)
NO PCA	43.36	8.43	14.12
PCA	67.36	60.91	63.97

Osservazioni

- Il modello con " PCA" mostra un notevole miglioramento rispetto al modello senza PCA in tutte le metriche.
- La " Precisione" aumenta di circa il " 55%" , il " Recall" di oltre il " 50%" e il " F1-score" migliora di " quasi 50 punti percentuali" .

- La " riduzione della dimensionalità con PCA" aiuta la SVM a migliorare significativamente la capacità di generalizzazione.

Capitolo 4

Conclusione

In questa tesina abbiamo analizzato il dataset *Adult* al fine di prevedere il reddito annuo di un campione di persone sulla base di dati censuari. Dopo aver effettuato la pulizia e la preparazione dei dati, abbiamo applicato diverse tecniche di riduzione della dimensionalità (**PCA e MDA**) per valutare il loro impatto sulle prestazioni dei modelli di classificazione. Abbiamo poi utilizzato diversi metodi di predizione: **LDA, SVM lineare e SVM non lineare con diversi kernel**.

I risultati ottenuti mostrano che:

- La **PCA** ha permesso di ridurre la dimensionalità dei dati senza perdere troppa informazione, migliorando la performance di alcuni modelli.
- La **MDA** ha ottimizzato la separazione tra le classi, risultando particolarmente utile per LDA.
- La **LDA** ha ottenuto un'accuratezza del ~83.5% senza PCA e leggermente inferiore con PCA.
- La **SVM lineare** ha mostrato buoni risultati (~84% con PCA), migliorando rispetto a LDA.
- La **SVM con kernel non lineari** ha raggiunto la miglior accuratezza, con il kernel RBF che ha ottenuto ~**82.7% di accuratezza con PCA**, bilanciando precisione e recall.

In generale, possiamo concludere che l'utilizzo della **PCA** è vantaggioso in alcuni casi, ma che **i metodi di Machine Learning più avanzati, come SVM con kernel RBF, tendono a dare le migliori prestazioni**.