

UNIVERSITÀ DEGLI STUDI DI FIRENZE

CORSO DI LAUREA MAGISTRALE IN INFORMATICA

MULTIVARIATE ANALYSIS AND STATISTICAL LEARNING

Principal Component Analysis: dalla teoria alla pratica

Autore:

Marco BURACCHI

Docente:

Prof.ssa Anna GOTTARD

10 febbraio 2018

Indice

| | | |
|----------|--|-----------|
| 1 | Principal Component Analysis | 2 |
| 1.1 | Subsection 1 | 2 |
| 1.2 | Subsection 2 | 2 |
| 2 | Implementazione Python | 3 |
| 2.1 | PANDAS | 3 |
| 2.2 | Implementazione | 3 |
| 2.2.1 | Preparazione dataset | 3 |
| 2.2.2 | Calcolo autovalori e autovettori | 6 |
| 2.2.3 | Selezione delle componenti | 6 |
| 2.2.4 | Matrice di proiezione | 8 |
| 2.2.5 | Proiezione nel nuovo spazio | 8 |
| 3 | Caso di studio | 9 |
| 4 | Codice Python completo | 10 |

1 Principal Component Analysis

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.1 Subsection 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.2 Subsection 2

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2 Implementazione Python

Passiamo adesso all'implementazione di un piccolo esempio di *Principal Component Analysis* (PCA) sul dataset *IRIS*. Per questa realizzazione è stato scelto il linguaggio *Python* con la relativa libreria *pandas*.

2.1 PANDAS

pandas è una libreria Python open-source, ad alte prestazioni e con licenza BSD che fornisce strutture dati e strumenti per l'analisi facili da usare.



È un progetto sponsorizzato da NumFocus. Questo assicura lo sviluppo continuo e a livello mondiale e permette anche di aiutare gli sviluppatori con donazioni o citazioni. Permette di lavorare con dati organizzati in maniera *relazionale* o *etichettata* in maniera intuitiva. Le sue due strutture dati principali sono le serie (unidimensionali) e i dataframe (bidimensionali). Per il nostro

esempio utilizzeremo questa seconda struttura.

2.2 Implementazione

Come già detto, utilizzeremo il dataset *IRIS* che contiene 150 misurazioni di fiori iris di tre specie diverse. La figura 1 schematizza le variabili del dataset che sono:

- Larghezza dei sepali in cm
- Lunghezza dei sepali in cm
- Larghezza dei petali in cm
- Lunghezza dei petali in cm
- Specie (setosa, versicolor, virginica)

2.2.1 Preparazione dataset

Caricamento dataset

Come prima cosa, scarichiamo il dataset dall'archivio della UCI (Università della California) e carichiamolo assegnando i nomi desiderati alle colonne e rimuovendo i valori NA.

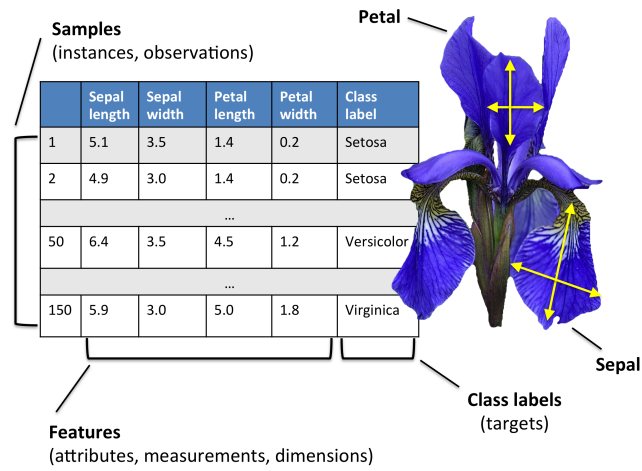


Figura 1: Struttura dataset

```
# download dataset
df = pd.read_csv(
    filepath_or_buffer='https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',
    header=None,
    sep=',')

# scelgo solamente le colonne con i valori di interesse
df.columns=['sepal_len', 'sepal_wid', 'petal_len', 'petal_wid', 'class']
df.dropna(how="all", inplace=True) # Elimina i valori NA
print(df.tail()) #visualizza ultime 5 righe
```

```
|   sepal_len  sepal_wid  petal_len  petal_wid      class
145         6.7         3.0         5.2         2.3  Iris-virginica
146         6.3         2.5         5.0         1.9  Iris-virginica
147         6.5         3.0         5.2         2.0  Iris-virginica
148         6.2         3.4         5.4         2.3  Iris-virginica
149         5.9         3.0         5.1         1.8  Iris-virginica
```

Figura 2: Dataset memorizzato

Dopo questi comandi otteniamo il risultato visibile in figura 2. Dividiamo i valori numerici dalla specie creando una matrice $X \in \mathcal{M}^{150 \times 4}$ e un vettore $y \in \mathcal{M}^{150 \times 1}$

```
# X = tabella con valori, y = etichette
X = df.ix[:,0:4].values
y = df.ix[:,4].values
```

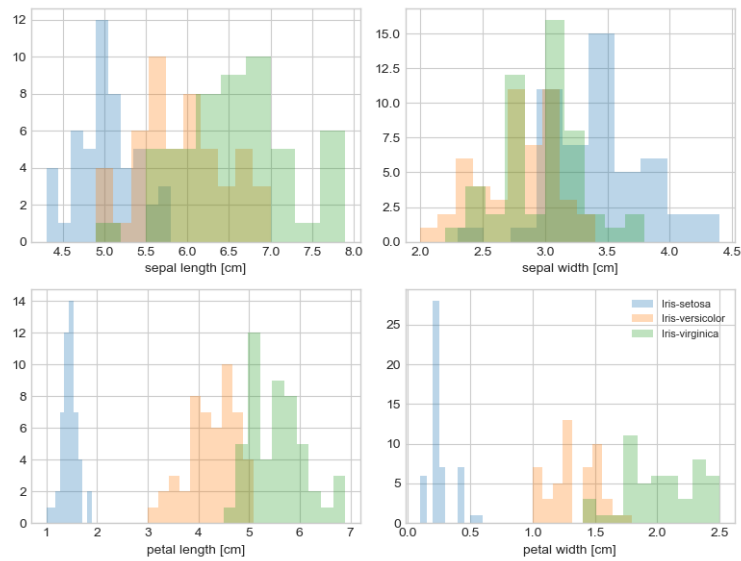


Figura 3: Analisi descrittiva

Analisi descrittiva

Per avere un'idea di come si distribuiscono le tre classi di fiori sulle varie caratteristiche, creiamo qualche istogramma.

```
# creazione istogrammi
feature_dict = {0: 'sepal length [cm]',
                1: 'sepal width [cm]',
                2: 'petal length [cm]',
                3: 'petal width [cm]'}

with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(8, 6))
    for cnt in range(4):
        plt.subplot(2, 2, cnt+1)
        for lab in ('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'):
            plt.hist(X[y==lab, cnt],
                    label=lab,
                    bins=10,
                    alpha=0.3,)
        plt.xlabel(feature_dict[cnt])
    plt.legend(loc='upper right', fancybox=True, fontsize=8)

plt.tight_layout()
plt.show()
```

Il risultato è visibile in figura 3

Standardizzazione

Standardizziamo i dati in scala unitaria (media = 0 e varianza = 1) per migliorare le performance.

```
# normalizzazione dati
X_std = StandardScaler().fit_transform(X)
```

2.2.2 Calcolo autovalori e autovettori

Calcoliamo gli autovettori e gli autovalori della matrice di covarianza e della matrice di correlazione. Possiamo calcolarli utilizzando la formula teorica, utilizzando una funzione di libreria oppure applicando la decomposizione a valori singolari. In ogni caso, i risultati saranno gli stessi (tranne nel caso della SVD che differisce solo per alcuni segni dato che la scomposizione è unica a meno di matrici di fase).

```
# vettore delle medie e matrice di covarianza
mean_vec = np.mean(X_std, axis=0)
cov_mat = (X_std - mean_vec).T.dot((X_std - mean_vec)) / (X_std.shape[0]-1)
print('Matrice di covarianza calcolata: \n%s\n' %cov_mat)

# funzione di libreria
print('Matrice di covarianza NumPy: \n%s\n' %np.cov(X_std.T))
separate()

# calcolo autovalori e autovettori su matrice di covarianza
cov_mat = np.cov(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
print('Autovettori cov: \n%s\n' %eig_vecs)
print('Autovalori cov: \n%s\n' %eig_vals)
separate()

# calcolo autovalori ed autovettori su matrice di correlazione dati standardizzati
cor_mat1 = np.corrcoef(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(cor_mat1)
print('Autovettori corrSTD: \n%s\n' %eig_vecs)
print('Autovalori corrSTD: \n%s\n' %eig_vals)
separate()

# calcolo autovalori ed autovettori su matrice di correlazione dati grezzi
cor_mat2 = np.corrcoef(X.T)
eig_vals, eig_vecs = np.linalg.eig(cor_mat2)
print('Autovettori corr: \n%s\n' %eig_vecs)
print('Autovalori corr: \n%s\n' %eig_vals)
separate()

# decomposizione ai valori singolari
u,s,v = np.linalg.svd(X_std.T)
print('Autovettori SVD: \n%s\n' %u)
separate()
```

La matrice che si ottiene è la seguente:

$$\begin{bmatrix} 0.52237162 & -0.37231836 & -0.72101681 & 0.26199559 \\ -0.26335492 & -0.92555649 & 0.24203288 & -0.12413481 \\ 0.58125401 & -0.02109478 & 0.14089226 & -0.80115427 \\ 0.56561105 & -0.06541577 & 0.6338014 & 0.52354627 \end{bmatrix}$$

2.2.3 Selezione delle componenti

Vogliamo ridurre le dimensioni dello spazio delle variabili proiettandolo in un sottospazio più piccolo. I suoi assi saranno gli autovettori appena calcolati. Purtroppo, in questo momento definiscono solamente la direzione perché sono tutti a lunghezza unitaria. Per stabilire quale può essere abbandonato senza perdere troppa informazione controlliamo ed ordiniamo i rispettivi autovalori.

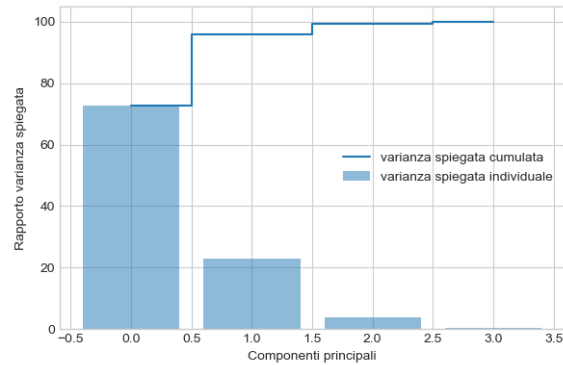


Figura 4: Varianza spiegata

```
# ordinamento degli autovalori
## creazione coppie (autovalore, autovettore)
eig-pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

## Ordinamento dal maggiore al minore
eig-pairs.sort(key=lambda x: x[0], reverse=True)
print('Autovalori ordinati:')
for i in eig-pairs:
    print(i[0])
```

A questo punto calcoliamo la varianza spiegata delle singole componenti e mostriamola in un grafico (Figura 4).

```
# varianza spiegata
tot = sum(eig_vals)
var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(6, 4))

    plt.bar(range(4), var_exp, alpha=0.5, align='center',
            label='varianza spiegata individuale')
    plt.step(range(4), cum_var_exp, where='mid',
            label='varianza spiegata cumulata')
    plt.ylabel('Rapporto varianza spiegata')
    plt.xlabel('Componenti principali')
    plt.legend(loc='best')
    plt.tight_layout()
    plt.show()
```

Dal grafico possiamo dedurre che la maggior parte della varianza può essere spiegata dalla prima componente (72.77%). Anche la seconda porta un po' di informazione (23.03%) mentre la terza e la quarta possono tranquillamente essere rimosse senza perdere troppa informazione.

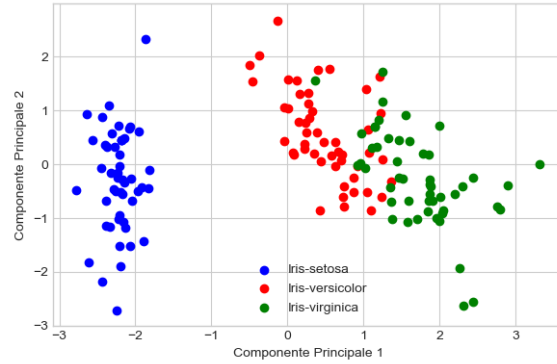


Figura 5: Distribuzione campioni dopo PCA

2.2.4 Matrice di proiezione

Per costruire la matrice di proiezione che ci permetterà di passare da uno spazio 4-dimensionale ad uno 2-dimensionale concateniamo le prime due componenti degli autovettori.

```
# matrice di proiezione
matrix_w = np.hstack((eig_pairs[0][1].reshape(4,1),
                      eig_pairs[1][1].reshape(4,1)))
```

La matrice risultante sarà quindi:

$$W = \begin{bmatrix} 0.52237162 & -0.37231836 \\ -0.26335492 & -0.92555649 \\ 0.58125401 & -0.02109478 \\ 0.56561105 & -0.06541577 \end{bmatrix}$$

2.2.5 Proiezione nel nuovo spazio

Come ultimo passo, utilizzeremo la matrice W per trasformare i nostri campioni nel nuovo sottospazio con l'equazione

$$Y = X \times W \quad Y \in \mathcal{M}^{150 \times 2}$$

```
# proiezione nel nuovo spazio
Y = X_std.dot(matrix_w)
```

Dopo l'applicazione della PCA otteniamo uno spazio bidimensionale dove i campioni sono distribuiti lungo i nuovi assi (Figura 5).

3 Caso di studio

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

4 Codice Python completo

```
'''
Created on 08 feb 2018

@author: marco
'''

import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA as sklearnPCA

def separate():
    print('\n#####\n')

# download dataset
df = pd.read_csv(
    filepath_or_buffer='https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.
    data',
    header=None,
    sep=',')

# scelgo solamente le colonne con i valori di interesse
df.columns=['sepal_len', 'sepal_wid', 'petal_len', 'petal_wid', 'class']
df.dropna(how="all", inplace=True) # Elimina i valori NA
print(df.tail()) #visualizza ultime 5 righe
separate()

# print(df)

# X = tabella con valori, y = etichette
X = df.ix[:,0:4].values
y = df.ix[:,4].values

# print(X)
# print(y)

# creazione istogrammi
feature_dict = {0: 'sepal length [cm]',
                 1: 'sepal width [cm]',
                 2: 'petal length [cm]',
                 3: 'petal width [cm]'}

with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(8, 6))
    for cnt in range(4):
        plt.subplot(2, 2, cnt+1)
        for lab in ('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'):
            plt.hist(X[y==lab, cnt],
                    label=lab,
                    bins=10,
                    alpha=0.3)
        plt.xlabel(feature_dict[cnt])
    plt.legend(loc='upper right', fancybox=True, fontsize=8)

    plt.tight_layout()
    plt.show()

# normalizzazione dati
X_std = StandardScaler().fit_transform(X)
```

```

# vettore delle medie e matrice di covarianza
mean_vec = np.mean(X_std, axis=0)
cov_mat = (X_std - mean_vec).T.dot((X_std - mean_vec)) / (X_std.shape[0]-1)
print('Matrice di covarianza calcolata: \n%s\n' %cov_mat)

# funzione di libreria
print('Matrice di covarianza NumPy: \n%s\n' %np.cov(X_std.T))
separate()

# calcolo autovalori e autovettori su matrice di covarianza
cov_mat = np.cov(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
print('Autovettori cov: \n%s\n' %eig_vecs)
print('Autovalori cov: \n%s\n' %eig_vals)
separate()

# calcolo autovalori ed autovettori su matrice di correlazione dati standardizzati
cor_mat1 = np.corrcoef(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(cor_mat1)
print('Autovettori corrSTD: \n%s\n' %eig_vecs)
print('Autovalori corrSTD: \n%s\n' %eig_vals)
separate()

# calcolo autovalori ed autovettori su matrice di correlazione dati grezzi
cor_mat2 = np.corrcoef(X.T)
eig_vals, eig_vecs = np.linalg.eig(cor_mat2)
print('Autovettori corr: \n%s\n' %eig_vecs)
print('Autovalori corr: \n%s\n' %eig_vals)
separate()

# decomposizione ai valori singolari
u,s,v = np.linalg.svd(X_std.T)
print('Autovettori SVD: \n%s\n' %u)
separate()

# ordinamento degli autovalori

## creazione coppie (autovalore, autovettore)
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

## Ordinamento dal maggiore al minore
eig_pairs.sort(key=lambda x: x[0], reverse=True)
print('Autovalori ordinati:')
for i in eig_pairs:
    print(i[0])
separate()

# varianza spiegata
tot = sum(eig_vals)
var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(6, 4))

    plt.bar(range(4), var_exp, alpha=0.5, align='center',
            label='varianza spiegata individuale')
    plt.step(range(4), cum_var_exp, where='mid',
            label='varianza spiegata cumulata')
    plt.ylabel('Rapporto varianza spiegata')
    plt.xlabel('Componenti principali')
    plt.legend(loc='best')
    plt.tight_layout()
    plt.show()

# matrice di proiezione
matrix_w = np.hstack((eig_pairs[0][1].reshape(4,1),

```

```

print('Matrice W:\n', matrix_w)
separate()

# proiezione nel nuovo spazio
Y = X_std.dot(matrix_w)
print(Y)
with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(6, 4))
    for lab, col in zip(('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'),
                        ('blue', 'red', 'green')):
        plt.scatter(Y[y==lab, 0],
                    Y[y==lab, 1],
                    label=lab,
                    c=col)

    plt.xlabel('Componente Principale 1')
    plt.ylabel('Componente Principale 2')
    plt.legend(loc='lower center')
    plt.tight_layout()
    plt.show()

# pacchetto scikit-learn
sklearn_pca = sklearnPCA(n_components=2)
Y_sklearn = sklearn_pca.fit_transform(X_std)
print(Y_sklearn)
for i in Y_sklearn:
    i[1] = -1*i[1]
print(Y_sklearn)
with plt.style.context('seaborn-whitegrid'):
    plt.figure(figsize=(6, 4))
    for lab, col in zip(('Iris-setosa', 'Iris-versicolor', 'Iris-virginica'),
                        ('blue', 'red', 'green')):
        plt.scatter(Y_sklearn[y==lab, 0],
                    Y_sklearn[y==lab, 1],
                    label=lab,
                    c=col)

    plt.xlabel('Componente Principale 1')
    plt.ylabel('Componente Principale 2')
    plt.legend(loc='lower center')
    plt.tight_layout()
    plt.show()

if __name__ == '__main__':
    pass

```

Riferimenti bibliografici

[Figueredo and Wolf, 2009] Figueredo, A. J. and Wolf, P. S. A. (2009). Assortative pairing and life history strategy - a cross-cultural study. *Human Nature*, 20:317–330.