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

La *Principal Component Analysis* (PCA) è una trasformazione lineare della matrice dei dati il cui scopo è rappresentare la variazione presente nelle tante variabili utilizzando un numero k molto più piccolo di *"fattori"* o *"componenti principali"*. Si costruisce un nuovo spazio su cui rappresentare i campioni ridefinendo gli assi utilizzando le componenti principali al posto delle variabili originali. L'uso di questi nuovi assi permette di rappresentare la vera natura multivariata dei dati in un numero relativamente piccolo di dimensioni e di usare questa rappresentazione per identificare la struttura dei dati stessi.

1.1 Cosa fa?

Supponiamo di aver misurato il valore di due variabili su 40 campioni come in Figura 1

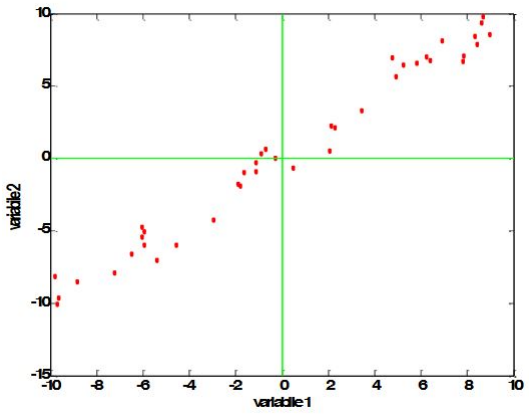


Figura 1: Plot variabili A e B

Vogliamo studiare le relazioni tra i campioni. Le distanze tra i campioni sono usate per definire similarità e differenze. In altre parole, lo scopo della PCA è di descrivere le distanze fra i punti (distribuzione, variabilità) utilizzando il minor numero di dimensioni possibili. Questo scopo si raggiunge costruendo assi che si allineano coi dati. Come si vede dal grafico nessuna delle variabili originali descrive completamente la variabilità all'interno dei dati stessi.

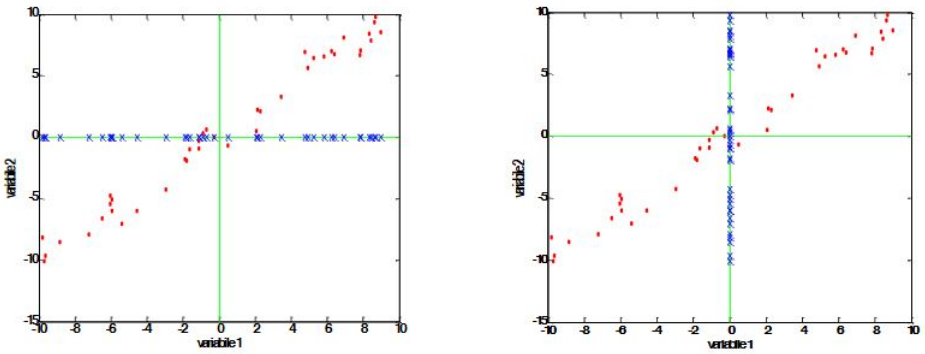


Figura 2: Proiezione variabili A e B

Se prendiamo come componenti principali però, le linee blu in figura 3, la prima componente principale descrive una quantità della variabilità originale maggiore di quella spiegata da ciascuna delle due variabili prese singolarmente. Essa spiega la massima percentuale della variabilità presente nei dati rappresentabile in una sola dimensione. Questa percentuale di variabilità spiegata può essere calcolata attraverso la varianza. La varianza è infatti un indice della dispersione dei dati lungo una particolare direzione ed è indipendente dal sistema di riferimento: una rotazione degli assi mantiene inalterata la varianza totale all'interno dei dati.

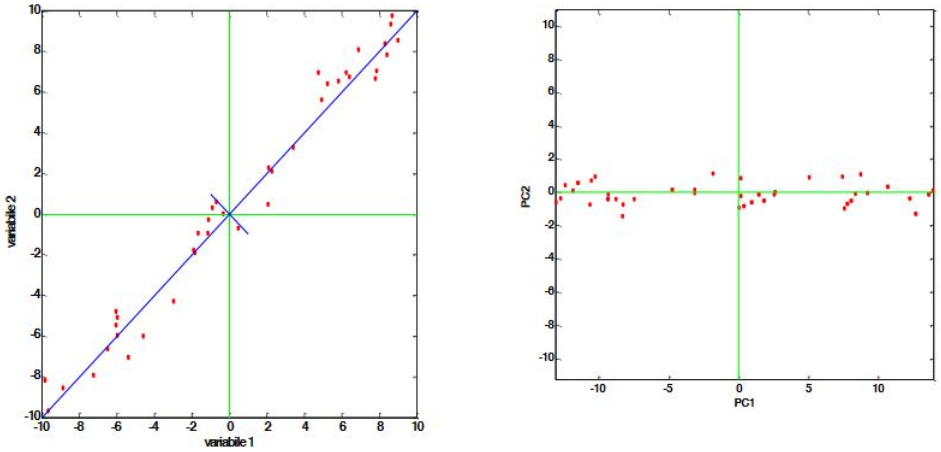


Figura 3: Componenti principali

In questo esempio, la prima componente principale cattura praticamente tutta la variabilità presente nei dati (99.83%) mentre la seconda descrive la rimanente variazione (0.17%). Questa considerazione può essere generalizzata: le componenti principali successive spiegano una sempre minore percentuale della variabilità originale. Seguendo questo principio è possibile dire che le ultime componenti principali descrivono principalmente *rumore*.

1.2 Come funziona?

Standardizzazione:

Standardizziamo i dati (media = 0 e varianza = 1). In questo modo possiamo lavorare con variabili che hanno scale e unità di misura differenti.

Calcolo della covarianza/correlazione:

Calcoliamo la matrice $\mathcal{S}$ di covarianza. La covarianza di due variabili indica quanto le due variabili si muovono insieme.

$$\mathcal{S} = \frac{1}{n-1}\sum_{1}^{n}(x-\mu)(x-\mu)^T$$

In alternativa possiamo utilizzare anche la matrice di correlazione.

Calcolo autovalori/autovettori:

Calcoliamo gli autovalori e gli autovettori della matrice di covarianza. Per definizione

$$\mathcal{S} \times v = \lambda \times v$$

Ognuno di questi autovettori corrisponde ad un autovalore la cui grandezza indica quanta varianza viene spiegata da quell'autovettore. Ordiniamo quindi gli autovalori in ordine decrescente e prendiamo i primi k. Chiamiamo $\mathcal{V}$ la matrice degli autovettori risultante.

Ruotiamo i dati:

Per convertire le osservazioni nel nuovo sistema di assi, moltiplichiamo i dati originali per gli autovettori che indicano la direzione dei nuovi assi (componenti principali). Questi dati ruotati vengono chiamati *score*.

$$Sc = X \times \mathcal{V}$$

Dati ridotti:

A questo punto è possibile utilizzare i nuovi dati.

2 Implementazione Python

Passiamo adesso all'implementazione di un piccolo esempio di 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

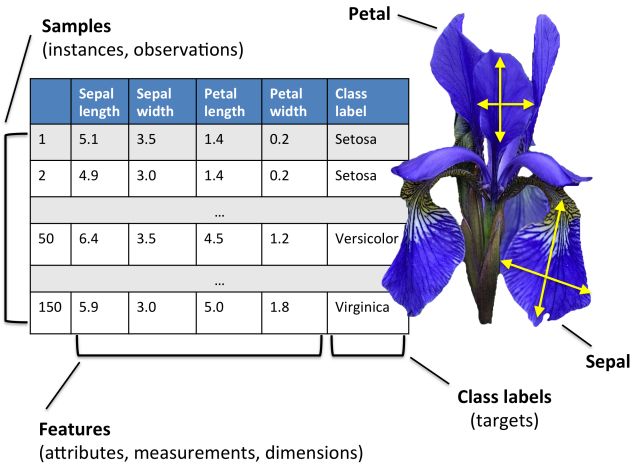
Come già detto, utilizzeremo il dataset IRIS che contiene 150 misurazioni di fiori iris di tre specie diverse. La figura 4 schematizza le variabili del dataset che sono:

1. Larghezza dei sepali in cm
2. Lunghezza dei sepali in cm
3. Larghezza dei petali in cm
4. Lunghezza dei petali in cm
5. 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.



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Figura 4: 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 5: Dataset memorizzato

Dopo questi comandi otteniamo il risultato visibile in figura 5. Dividiamo i valori numerici dalla specie creando una matrice $X \in \mathcal{M}^{150x4}$ e un vettore $y \in \mathcal{M}^{150x1}$

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

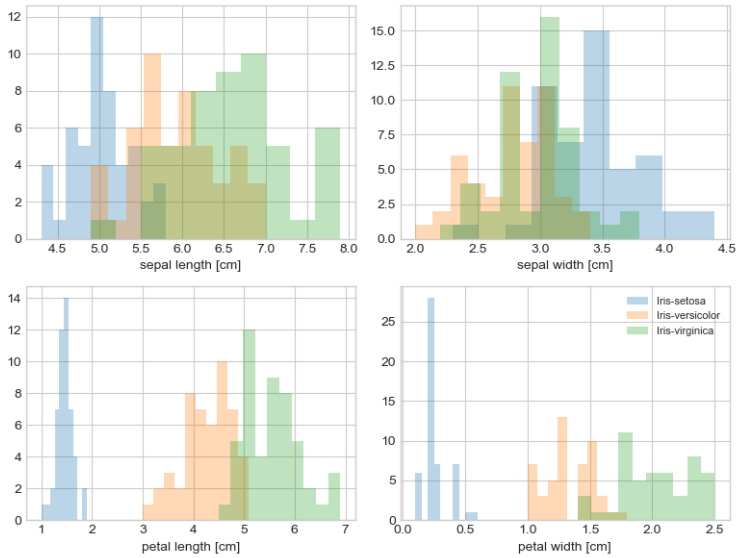


Figura 6: Analisi descrittiva

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 6

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.

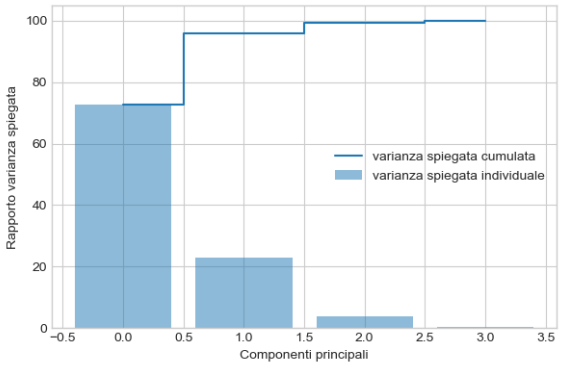


Figura 7: Varianza spiegata

```
# 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 7).

```
# 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.

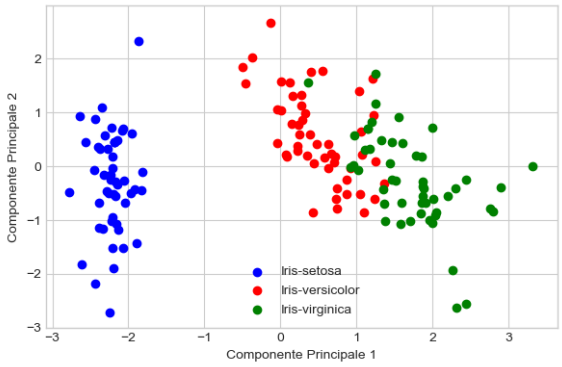


Figura 8: Distribuzione campioni dopo PCA

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

$$Y = X \times W \qquad Y \in \mathcal{M}^{150x2}$$

```
# 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 8).

Esiste anche una funzione di libreria che calcola direttamente dal dataset la PCA con il numero di componenti scelto.

```
# pacchetto scikit-learn
sklearn_pca = sklearnPCA(n_components=2)
Y_sklearn = sklearn_pca.fit_transform(X_std)
```

3 Caso di studio

I dati sul traffico di rete raccolti per l'analisi delle intrusioni hanno in genere molte dimensioni che rendono difficile sia l'analisi che la semplice visualizzazione. La PCA viene utilizzata per ridurre il numero di variabili dai campioni misurati per consentire analisi del traffico e visualizzazioni più semplici.

3.1 Il problema

L'enorme diffusione di sistemi informatici connessi tramite dispositivi di telecomunicazione ha aumentato in maniera esponenziale il problema della sicurezza delle reti. Di conseguenza sono stati sviluppati Sistemi di Rilevamento delle Intrusioni (IDS) sempre più sofisticati. In generale gli IDS si dividono in due macro-categorie:

- Signature based
- Anomaly based

La differenza tra i due è nella modalità di rilevamento delle intrusioni. In quelli signature based, si inseriscono le *firme* di attacchi noti (sequenze di operazioni, di comandi,...) che, quando vengono riconosciuti, provocano l'allarme. Gli IDS anomaly based sono i duali di questi in quanto conoscono il comportamento "normale" del sistema e fanno scattare l'allarme quando riconoscono un comportamento diverso da questo. I primi sono in grado di riconoscere solamente attacchi noti ma con il 100% di successo sulle rilevazioni. I secondi sono in grado potenzialmente di riconoscere attacchi nuovi a costo di ottenere saltuariamente qualche falso allarme. Adesso vedremo un metodo alternativo che utilizza la PCA per raggiungere gli stessi obiettivi.

I tipi di attacchi possibili su una rete sono moltissimi e molto diversi tra di loro. Ci focalizzeremo su due grandi classi chiamate *Denial of Service* (DoS) e Network Probe (NP) ed analizzeremo due tipi di attacchi per ognuna di queste classi.

Negli attacchi DoS l'attaccante cerca di rendere la risorsa sotto attacco troppo occupata (in caso di una rete) o piena (in caso di una memoria) perché l'utente legittimo possa utilizzarla. Prima di effettuare un attacco DoS viene generalmente effettuato un test della rete sulla quale l'attaccante agirà. Questo test viene

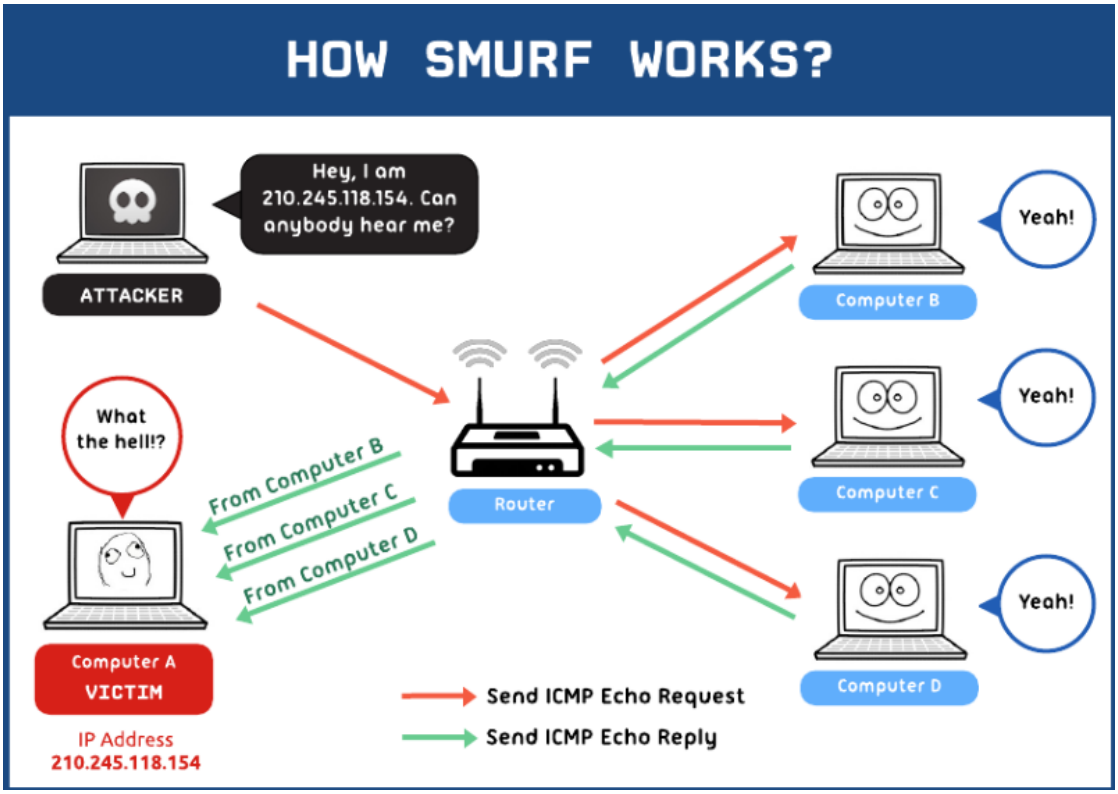


Figura 9: Schema di Smurf attack

eseguito andando a scansionare tutti i computer connessi alla rete sotto attacco e, per ognuno di questi, controllare tutte le porte di comunicazione con la rete in cerca di qualche vulnerabilità. Questo è un classico esempio di Network Probe.

3.1.1 Attacchi di tipo DoS

In questa categoria andremo ad analizzare due tipi di attacchi chiamati *Smurf* e *Neptune*.

Un attacco Smurf si basa semplicemente su una configurazione sbagliata di un computer che permette la creazione di un numero molto grande di messaggi che verranno inviati al computer della vittima con l'invio di un solo messaggio da parte dell'attaccante (Figura 9). Tutti questi messaggi andranno a saturare la rete della vittima che non potrà più usarla in alcun modo.

Un attacco Neptune si basa su una cattiva configurazione di un server in attesa di connessioni. Per creare una connessione, un utente deve mandare prima un

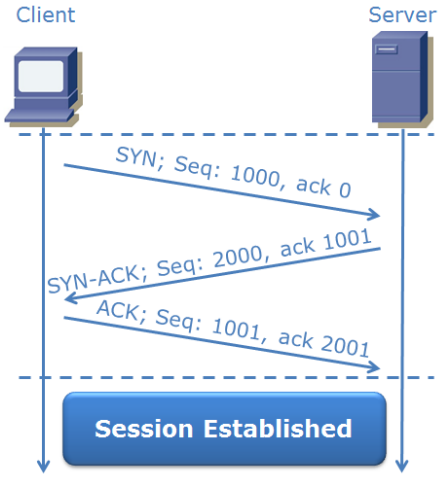


Figura 10: Neptune attack

messaggio per richiedere il collegamento, aspettare una risposta positiva dal server e infine mandare una conferma di ricevuta risposta. Questo procedimento si chiama *three-way handshake* (Figura 10). Le connessioni stabilite vengono poi salvate dal server su una tabella delle connessioni. Quello che fa un attaccante è mandare il primo pacchetto ma non mandare mai il terzo lasciando il server in attesa di effettuare la connessione senza mai potersi liberare.

3.1.2 Attacchi di tipo Network Probe

Gli attacchi di questo tipo sono molto più semplici. Quelli che andremo ad approfondire sono il PortSweep e l'IPSweep.

Ogni computer connesso ad una rete è raggiungibile attraverso un indirizzo chiamato indirizzo IP. Un attacco di IPSweep non fa altro che cercare sequenzialmente tutti gli IP connessi ad una rete. Una volta trovati gli indirizzi IP, per ognuno di questi esegue un PortSweep cioè una scansione sequenziale di tutte le porte di comunicazioni accessibili su ogni computer. Il grosso problema nel riconoscimento di questo tipo di attacchi è che a volte la scansione della rete o delle porte di un computer vengono eseguite legittimamente dagli amministratori di sistema per necessità di manutenzione o di controllo e quindi non possono essere impedite a priori.

3.2 Analisi dei dati

I dati per effettuare lo studio sono presi dal "1998 DARPA Intrusion Detection Evaluation data sets" e contiene dati di utilizzo della rete in sette settimane in caso di traffico normale e in caso di vari attacchi (tra i quali quelli di nostro interesse).

3.2.1 Preprocessing dei dati

Il dataset è stato preprocessato andando ad estrarre soltanto le informazioni riguardanti le intestazioni ("header") dei pacchetti passati dalla rete. Queste intestazioni sono divise in 12 parti (che saranno le nostre variabili).

- IPSource - IP mittente diviso in 4 parti (un generico indirizzo IP può essere 192.168.1.1)
- SPort - Porta di comunicazione del mittente
- IPDest - IP destinatario diviso in 4 parti
- DPort - Porta di comunicazione del destinatario
- Prot - Protocollo di comunicazione utilizzato
- Dim - Dimensioni del pacchetto

Sono stati creati 7 dataset, uno per ognuno dei quattro attacchi e tre per il normale traffico di rete (per cercare di comprendere diversi utilizzi benigni della rete). Data la grande dimensione dello spazio delle variabili (12 dimensioni) è molto difficile capire le relazioni che possono esserci tra di loro. Per questo motivo verrà utilizzata la PCA con la prima e la seconda componente che descriveranno la maggior parte della varianza di qusti dati. Da notare chele prime due componenti forniscono solamente la direzione della massima varianza ma non forniranno una chiara indicazione sulla malignità/benignità del traffico ma i coefficienti della trasformazione (loading) ci forniranno questa indicazione.

3.3 Risultati

Come accennato, i *principal component loading* sono i coefficienti della trasformazione delle componenti principali. Forniscono informazioni su quanto una certa variabile influisce sulla relativa componente principale. Un coefficiente alto indicherà che quella variabile ha una forte influenza su quella componente principale e

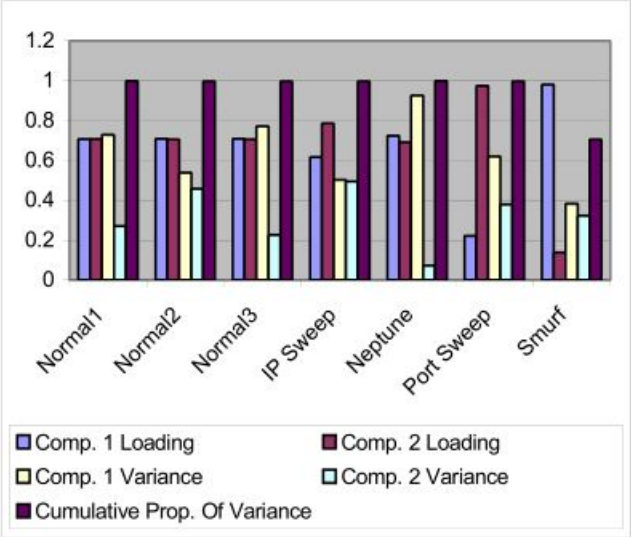


Figura 11: Loading e varianza delle componenti

viceversa. Nel nostro caso ci focalizzeremo sulle prime due componenti principali e sui relativi loadings (specialmente sulla variabile con il loading più alto). Nella Figura 11 possiamo vedere i due loadings più grandi in valore assoluto (SPort e DPort) e la varianza delle prime due componenti nei sette dataset.

3.3.1 Rilevare le intrusioni

L'idea fondamentale è la seguente. Come si vede dal grafico in Figura 11 i valori dei loadings delle prime due componenti principali in caso di traffico regolare sono quasi uguali mentre in caso di attacco differiscono considerevolmente (meno nell'attacco Neptune). Si potrebbe pensare di sfruttare questa conoscenza per riconoscere gli attacchi. Se i valori dei due loadings differiscono più di una certa soglia, facciamo scattare l'allarme.

A seconda della grandezza di questa soglia si può avere un comportamento più "difensivo" facendo scattare falsi allarmi in caso, ad esempio, di traffico benigno straordinario (manutenzione, nuovi incarichi temporanei,...) o un comportamento un po' più "permissivo" magari perdendosi qualche attacco Neptune.

Un secondo livello di analisi potrebbe però essere fatto sula varianza. Si può notare infatti che l'attacco di tipo Neptune ha come caratteristica una enorme differenza sulla varianza delle due componenti. Si può quindi pensare di affinare il rilevamento utilizzando due livelli. Se i loadings delle due componenti differi-

scono di molto dalla soglia, facciamo scattare l'allarme. Se sono vicini alla soglia controlliamo anche la varianza delle due componenti che, in caso di grandissima differenza, potrebbero suggerire un traffico malevolo.

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
```