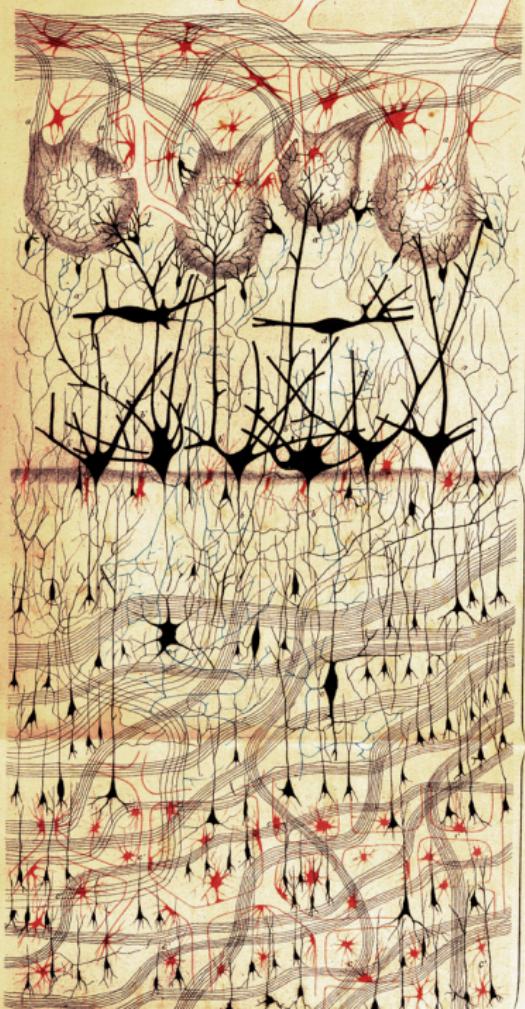


# Principal Component Analysis: dalla teoria alla pratica

Marco Buracchi

Università degli studi di Firenze

26 febbraio 2018



# Sommario

## Principal Component Analysis

Cenni di teoria

Funzionamento

## Implementazione Python

Strumenti

Implementazione

## Caso di studio

Problema

Esperimento

Risultati



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

# Cenni di teoria

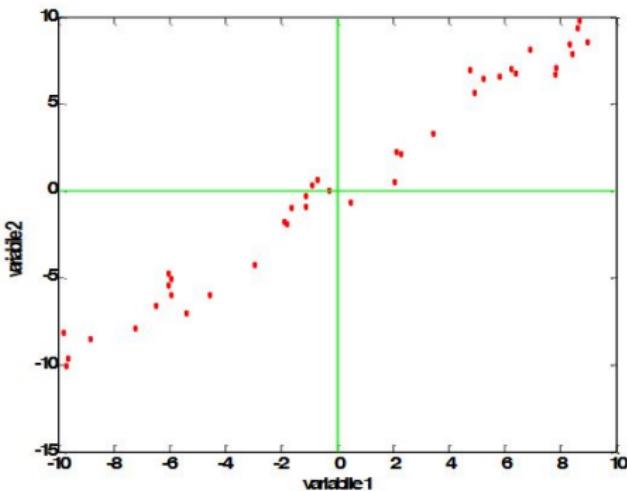


UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

- Trasformazione lineare della matrice dei dati  $\mathcal{X}$
- Misurazione della variazione delle variabili utilizzando un numero minore di "fattori"
- Trasportare il problema in uno spazio  $k$ -variato (generalmente bi-trivariato)
- Semplificazione di visualizzazione e lettura dei dati



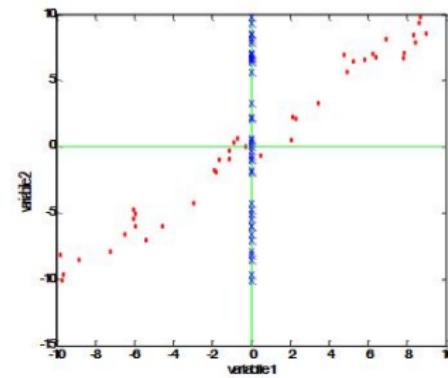
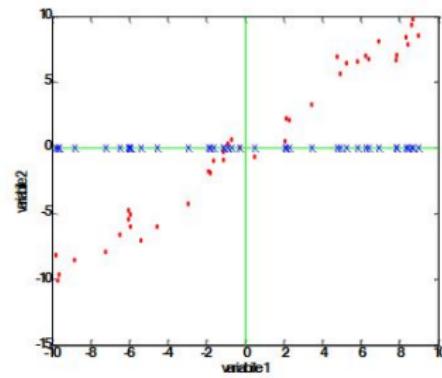
# Esempio



- 40 campioni
- 2 variabili



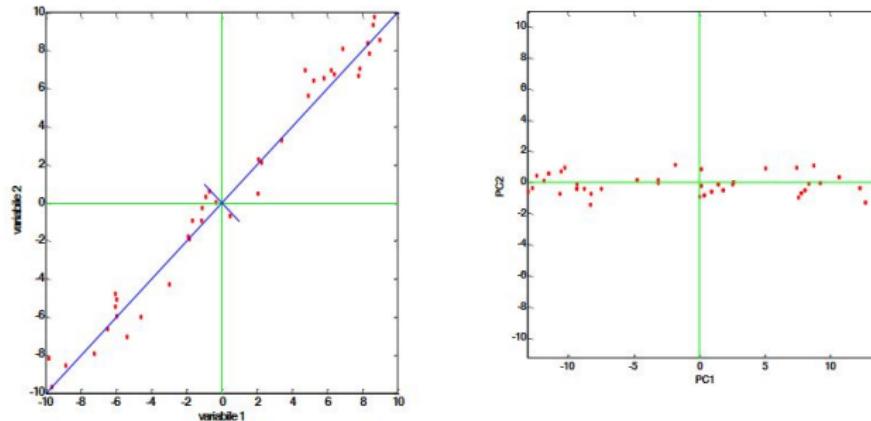
# Esempio - 2



- Nessuna delle due variabili descrive completamente la variabilità dei dati



# Componenti



- Prendiamo come componenti principali le linee blu
- La prima componente spiega la massima percentuale di variabilità rappresentabile in una dimensione



# Varianza

- Questa percentuale di variabilità può essere calcolata tramite la varianza
- La varianza è un indice della dispersione dei dati lungo una particolare direzione
- La varianza è indipendente dal sistema di riferimento
- Ruotare gli assi mantiene inalterata la varianza totale



# Componenti

- La prima componente cattura quasi tutta la variabilità presente nei dati (99.83%)
- La seconda descrive la rimanente (0.17%)
- Generalizzando, le componenti principali successive spiegano una sempre minore percentuale della variabilità originale
- Le ultime componenti principali descrivono principalmente rumore



# Funzionamento

## 1 Standardizzazione

- Standardizzare i dati (media = 0, varianza = 1)
- Possiamo lavorare con variabili su scale e unità di misure differenti

## 2 Calcolo covarianza/correlazione

- Calcoliamo la matrice  $S$  di covarianza

$$S = \frac{1}{n-1} \sum_1^n (x - \mu)(x - \mu)^T$$

- Possiamo usare anche la matrice di correlazione

## 3 Calcolo autovalori/autovettori

- $S \times v = \lambda \times v$



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

# Funzionamento

## 4 Scelta delle componenti

- Ordiniamo in maniera decrescente gli autovalori ottenuti
- Selezioniamo i primi  $k$
- Costruiamo  $\mathcal{W}$ , la matrice dei rispettivi autovettori

## 5 Rotazione dei dati

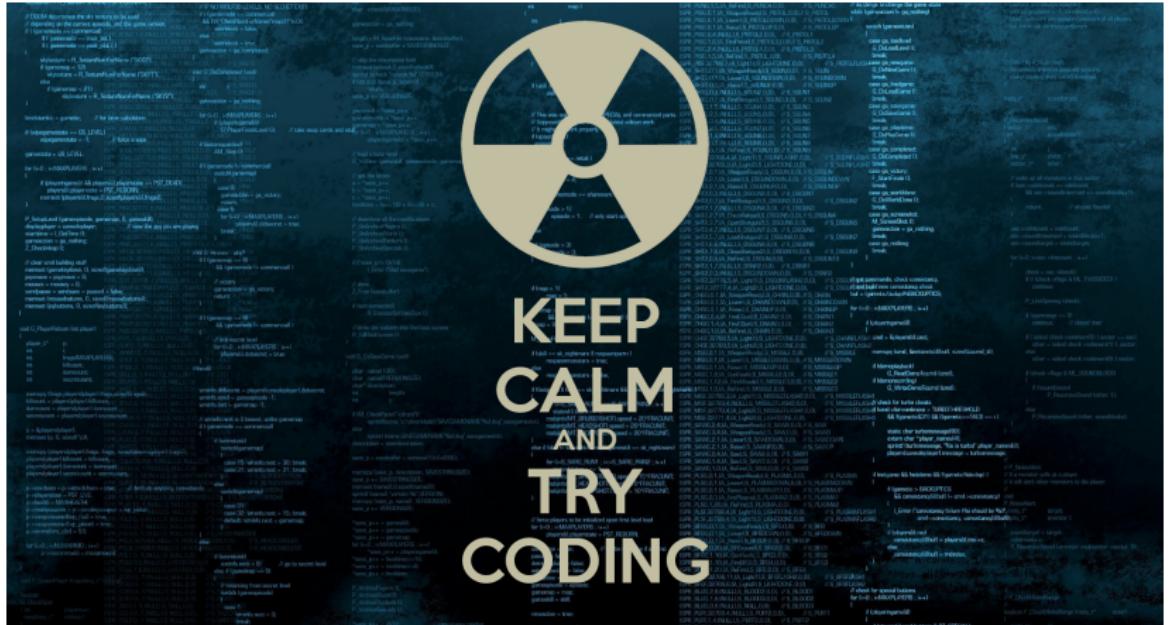
- Moltiplichiamo i dati originali per gli autovettori che indicano le direzioni dei nuovi assi (componenti principali)
- I dati ruotati vengono chiamati *score*

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

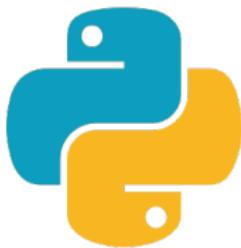


UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

# Implementazione

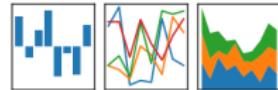


# Strumenti utilizzati



pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

# PANDAS

- Libreria Python, open source, ad alte prestazioni e con licenza BSD<sup>1</sup>
- Strutture dati e strumenti per l'analisi facili da usare (R-like)
  - Serie (unidimensionali)
  - Dataframe (bidimensionali)
- Dati organizzati in maniera *relazionale* o *etichettata*
- Sponsorizzato da *NumFocus* (*matplotlib*, *simpy*, *julia*...)

A Fiscally Sponsored Project of

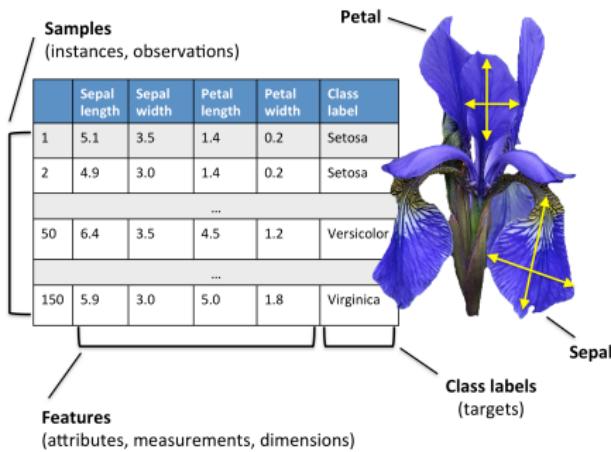


- sviluppo continuo, a livello mondiale e sistema di donazioni a supporto

---

<sup>1</sup>Berkeley Software Distribution

# Dataset



- Dataset IRIS
- 150 misurazioni di fiori iris
- 3 diverse specie

# Caricamento 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



# Divisione valori

- Matrice valori numerici  $X \in \mathcal{M}^{150 \times 4}$
- Vettore specie  $y \in \mathcal{M}^{150 \times 1}$

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



# Analisi descrittiva

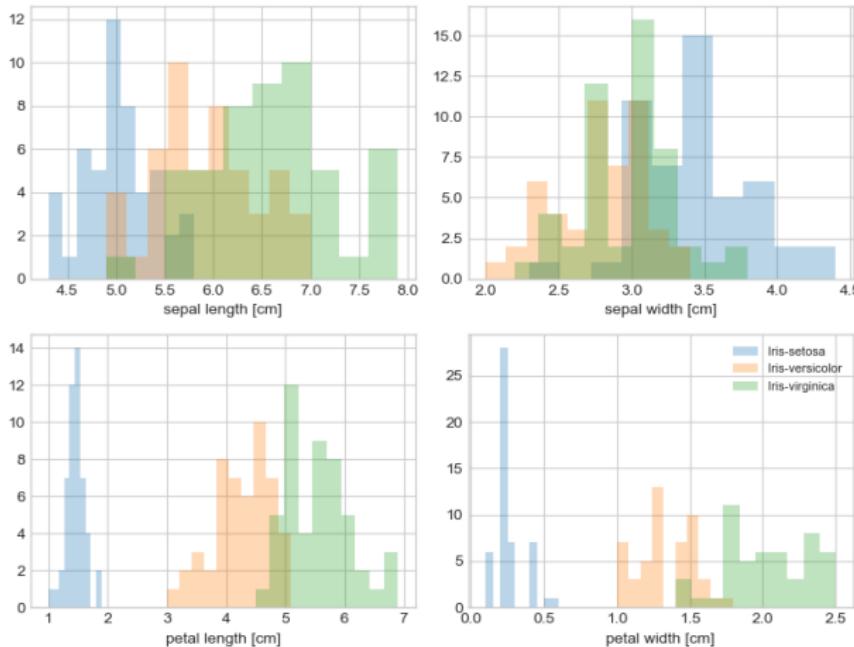
```
# creazione istogrammi
feature_dict = {0: 'sepal length [cm]',
                 1: 'sepal width [cm]',
                 2: 'petal length [cm]',
                 3: 'petal width [cm]'}

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()
```



# Analisi descrittiva



# Standardizzazione



- Valori su scale diverse richiedono standardizzazione
- Usiamo la funzione di libreria

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



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

# Autovalori e autovettori

```
# 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()
```



# Autovalori e autovettori

```
# 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()
```

- In tutti i casi otteniamo la matrice degli autovettori

$$\begin{bmatrix} 0.522 & -0.372 & -0.721 & 0.261 \\ -0.263 & -0.925 & 0.242 & -0.124 \\ 0.581 & -0.021 & 0.140 & -0.801 \\ 0.565 & -0.065 & 0.633 & 0.523 \end{bmatrix}$$

con i rispettivi autovalori

$$[2.910 \quad 0.921 \quad 0.147 \quad 0.020]$$



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

# Selezione componenti

- Gli autovettori calcolati forniscono solamente la direzione perché sono tutti a norma unitaria
- Per scegliere i più interessanti ordiniamo i rispettivi autovalori

```
# 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)
```



# Varianza spiegata

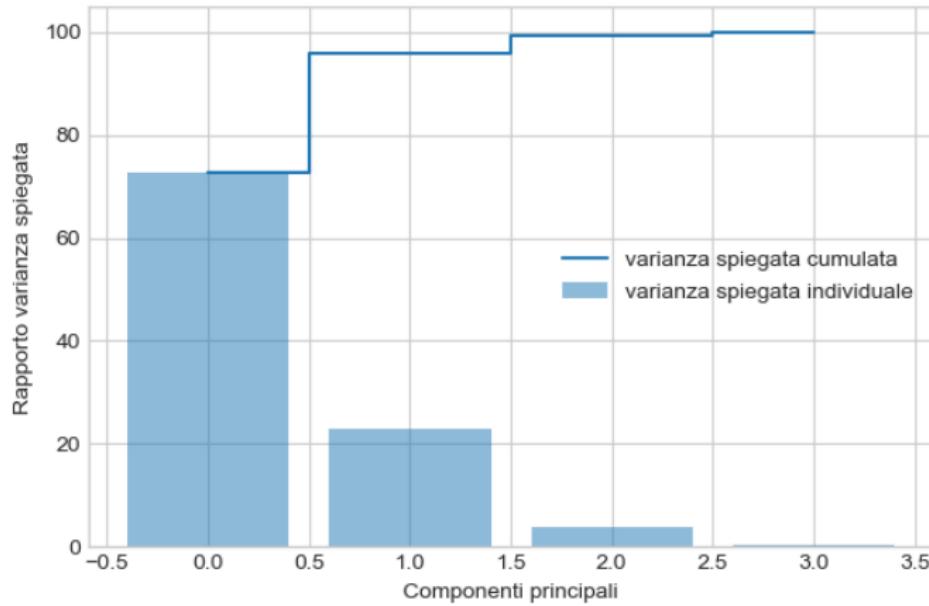
- Calcoliamo la varianza spiegata dalle singole componenti
- Visualizziamo il risultato su un grafico

```
# 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()
```



# Varianza spiegata



# Matrice di proiezione

- Passiamo da 4 a 2 dimensioni
- Estraiamo le prime due componenti dalla matrice degli autovettori

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

- Creiamo la matrice di proiezione  $\mathcal{W}$

$$\begin{bmatrix} 0.522 & -0.372 & -0.721 & 0.261 \\ -0.263 & -0.925 & 0.242 & -0.124 \\ 0.581 & -0.021 & 0.140 & -0.801 \\ 0.565 & -0.065 & 0.633 & 0.523 \end{bmatrix} \Rightarrow \mathcal{W} = \begin{bmatrix} 0.522 & -0.372 \\ -0.263 & -0.925 \\ 0.581 & -0.021 \\ 0.565 & -0.065 \end{bmatrix}$$



# Proiezione

- Proiettiamo nel nuovo sottospazio

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

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



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

# Proiezione

- Proiettiamo nel nuovo sottospazio

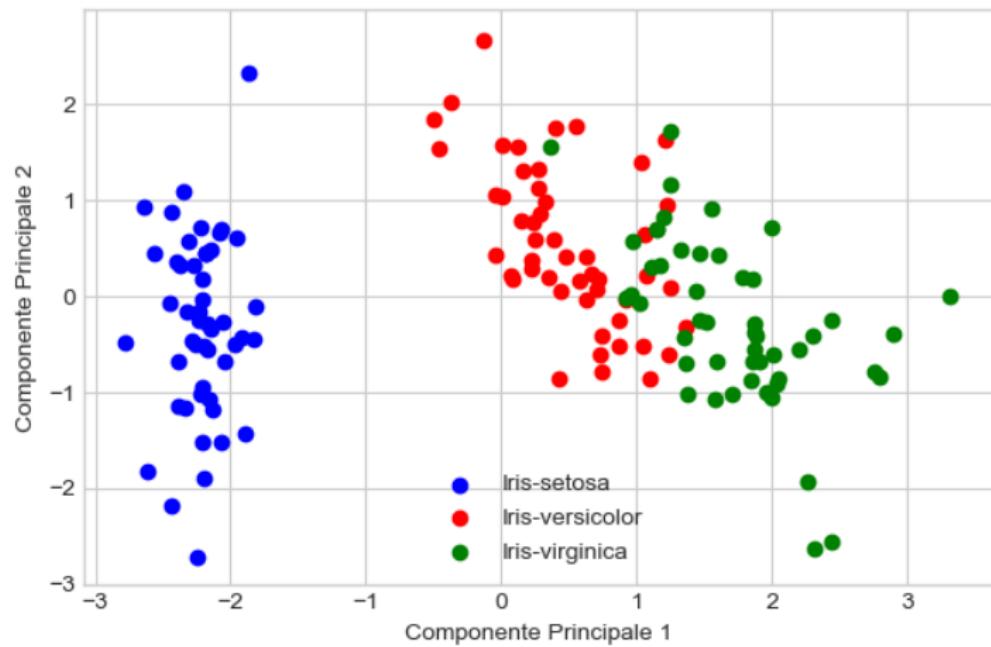
$$\mathcal{Y} = \mathcal{X} \times \mathcal{W} \quad \mathcal{Y} \in \mathcal{M}^{150 \times 2}$$

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



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

# Risultato



# Post Scriptum

- Esiste una funzione di libreria che calcola  $\mathcal{Y}$  direttamente da  $\mathcal{X}$
- L'unico parametro da fornire è il numero di componenti che si vogliono prendere in considerazione

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



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

# Caso di studio



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

# Problema



- Miliardi di dispositivi informatici connessi in rete nel mondo
- Innumerevoli casi di intrusioni malevoli in tali sistemi
- Il problema della sicurezza delle reti è molto serio
- I dati relativi al traffico di rete hanno in genere molte dimensioni
- Difficili da visualizzare e analizzare



# Intrusion Detection Systems



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

- Gli IDS si dividono in due grandi categorie

- ① Signature based

- Riconoscono le *firme* degli attacchi noti
- 100% di successo delle rilevazioni
- Riconoscono solo attacchi noti

- ② Anomaly based

- Riconoscono un comportamento diverso dal *normale*
- Possono riconoscere attacchi nuovi
- Possono creare falsi allarmi o non riconoscere qualche attacco



# Idea



- Possiamo costruire un IDS basato sulla PCA<sup>2</sup>?
- Capiamo un po' meglio come funzionano i principali attacchi

---

<sup>2</sup>2006, Labib Khaled and Vemuri V. Rao: An application of PCA to the detection and visualization of computer network attacks

# Tipi di attacchi

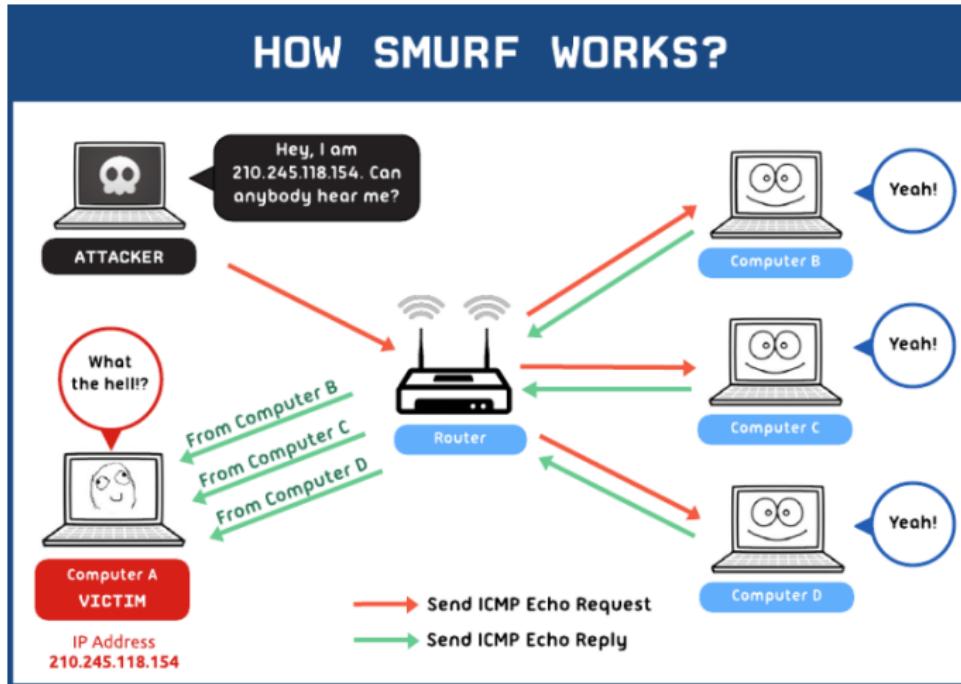
- Ci focalizzeremo su due tipi di attacchi
  - ① Denial of Service (DoS)
    - Si cerca di rendere la risorsa sotto attacco troppo occupata (rete) o piena (memoria) per essere utilizzata
  - ② Network Probe (NP)
    - Scansione sistematica di tutti i dispositivi connessi alla rete e delle loro porte di comunicazione



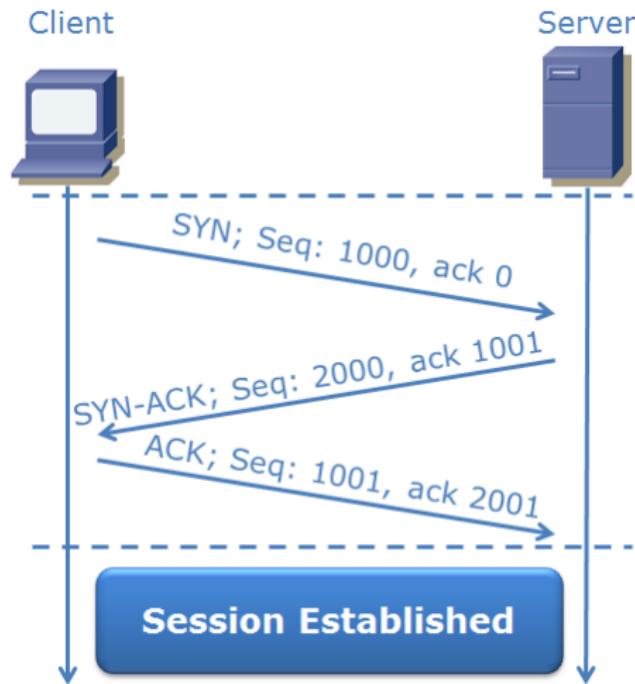
# Attacchi DoS



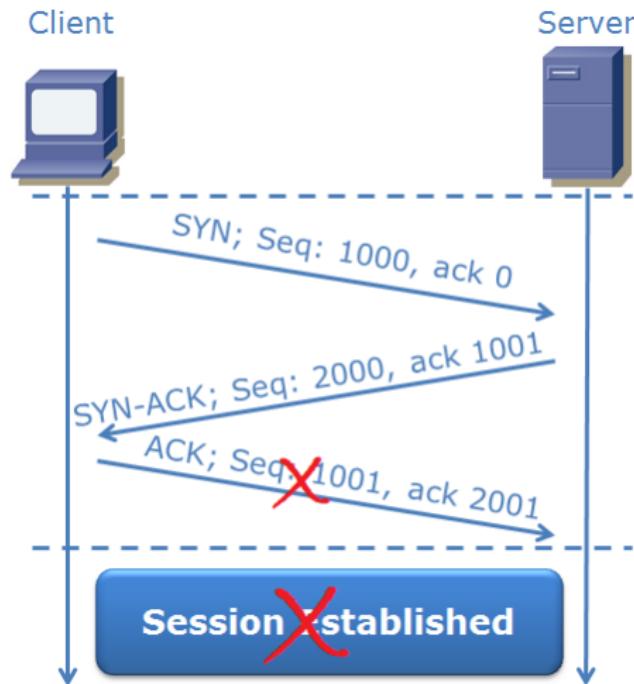
# Attacco SMURF



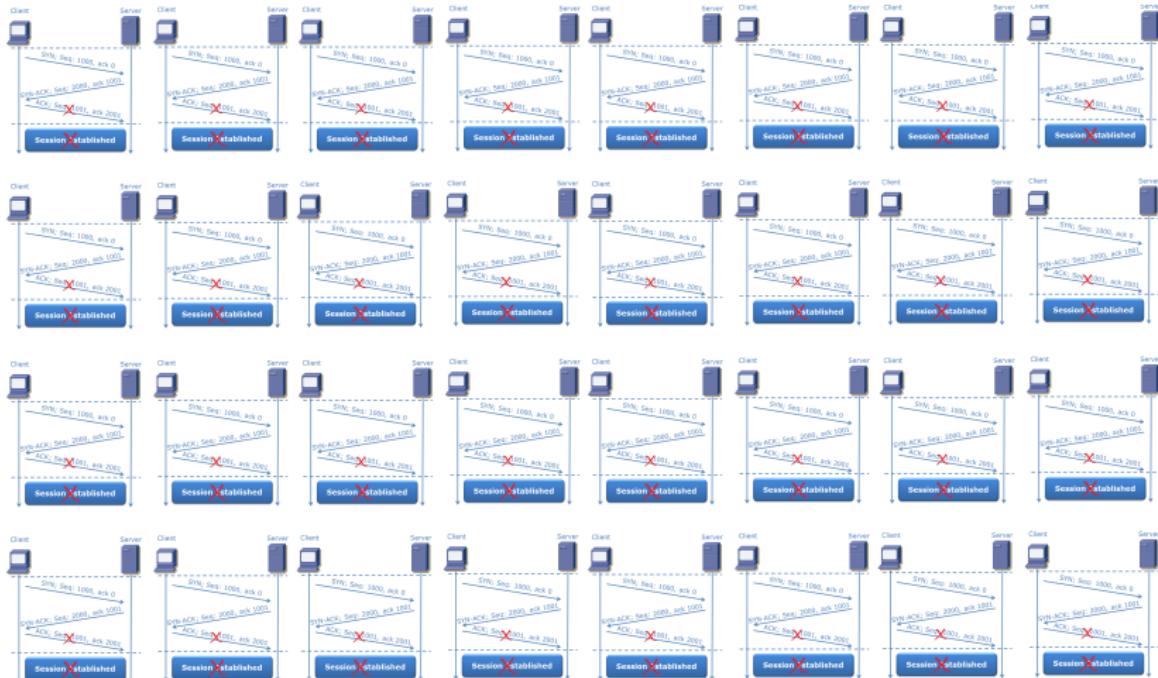
# Attacco Neptune



# Attacco Neptune



# Attacco Neptune



# Attacchi NP

- IPSweep
  - Scansione sistematica di tutti gli IP connessi alla rete
- PortSweep
  - Scansione sistematica di tutte le porte di comunicazione dei sistemi scoperti al passo precedente



# Analisi dei dati

- 1998 DARPA<sup>3</sup> Intrusion Detection Evaluation data sets
- Utilizzo della rete in 7 settimane
- Presente traffico normale e vari attacchi (tra cui quelli ai quali siamo interessati)
- Preprocessing (estrazione delle intestazioni o *header* dei pacchetti di rete)
- 12 variabili rilevanti

---

<sup>3</sup>Defense Advanced Research Projects Agency

# Esperimento

- Creati 7 dataset (uno per ogni attacco e 3 di traffico normale)
- Applicata la PCA ad ognuno di essi per passare da 12 a 2 dimensioni
- Le componenti forniranno la direzione della massima varianza
- I coefficienti della trasformazione (*loading*) ci indicheranno la malignità/benignità del traffico

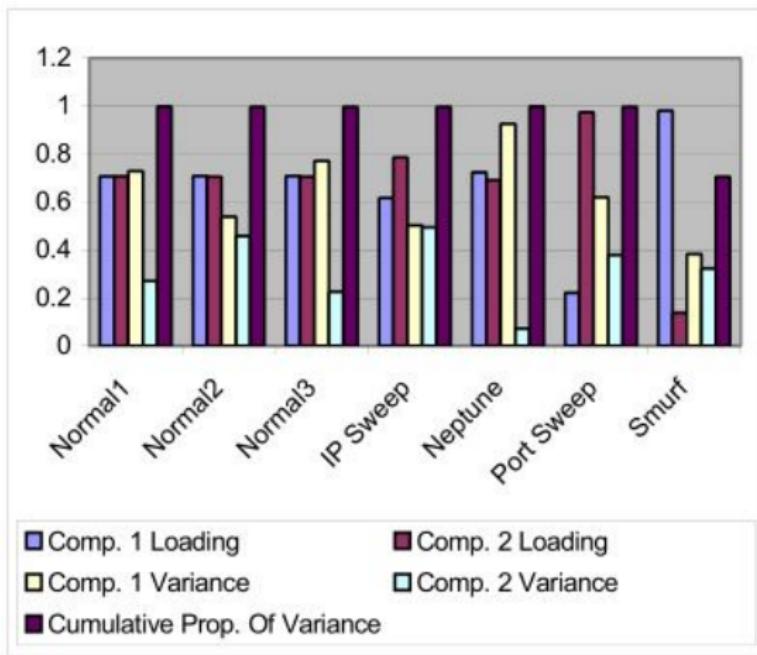


# Loadings

- I loadings forniscono informazioni su quanto ogni variabile influisce sulla relativa componente principale (che è una loro combinazione lineare)
- Focalizzeremo l'attenzione sulle prime due componenti principali e sui relativi loadings



# Loadings



The image features a iconic title card from a Looney Tunes cartoon. The background consists of several concentric, slightly curved bands of color transitioning from dark red at the edges to bright orange in the center. Overlaid on this background is the phrase "That's all Folks!" written in a white, flowing, cursive font. The text is positioned centrally and is partially cut off on the right side of the frame.

That's all Folks!