**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **ІКНІ**

Кафедра **СШІ**

**ЗВІТ**

До лабораторної роботи №2

**З дисципліни:**

**«Машинне навчання»**

**Виконав:**

ст. гр. КН-310

Шиманський П.С.

**Прийняла:**

Якимишин Х.М.

Львів - 2020

**Мета:** ​засвоїти основні відомості про роботу з алгоритмом метод головних компонент(PCA). Навчитись реалізовувати даний алгоритм.

Завдання: ​Реалізувати метод головних компонент і використати його для аналізу даних згідно варіантів, а саме:

1. Частина 1. Реалізація алгоритму

a. Згідно варіанту, обрахувати PCA використовуючи sklearn. Звізуалізувати основні компоненти згідно прикладу.

b. Реалізувати PCA алгоритм використовуючи PyTorch. Візуалізувати результати алгоритму.

c. Порівняти результати отримані компоненти, порівняти їхні результати. Вони повинні співпадати при правильній реалізації.

2. Частина 2. Обезшумлення

a. Добавити до даних шум.

b. Обрахувати на нових зашумлених даних PCA.

c. Реконстроювати дані використовуючи PCA для знешумлення.

d. Звізуалізувати зашумлені і знешумлені дані





**Хід роботи:**

1. Обрахувати РСА використовуючи sklearn:

def draw\_vector(v0, v1, ax=None):

ax = ax or plt.gca()

arrowprops=dict(arrowstyle='->',

linewidth=3,

shrinkA=0, shrinkB=0)

ax.annotate('', v1, v0, arrowprops=arrowprops)

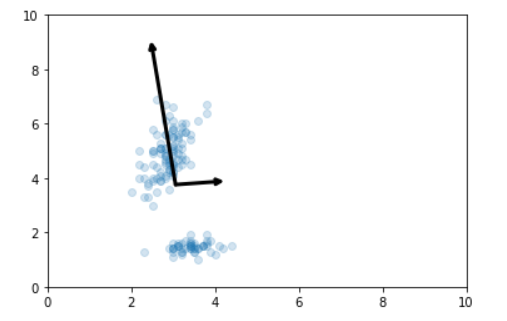
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)

for length, vector in zip(pca.explained\_variance\_, pca.components\_):

v = vector \* 3 \* np.sqrt(length)

draw\_vector(pca.mean\_, pca.mean\_ + v)

plt.axis([0, 10, 0, 10])



**Рис.1.** Візуалізація обрахованого РСА, використовуючи функції sklearn

1. Реалізувати PCA алгоритм використовуючи PyTorch. Візуалізувати результати алгоритму:

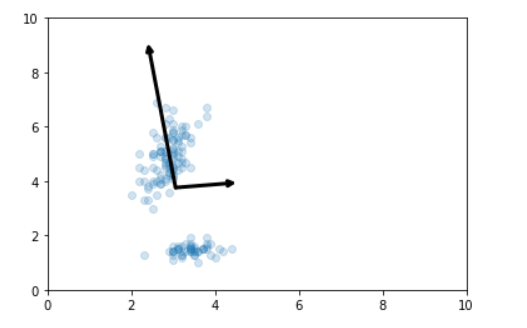
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)

for length, vector in zip(pca2['explained\_variance'], pca2['components']):

v = vector \* 3 \* np.sqrt(length)

draw\_vector(pca.mean\_, pca.mean\_ + v.data.numpy())

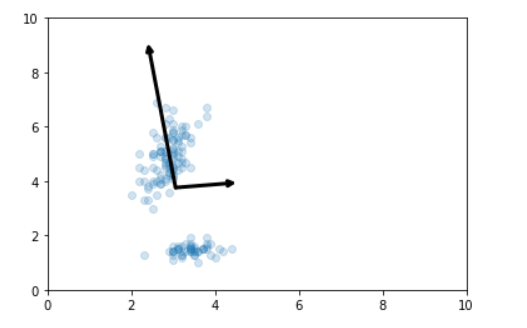
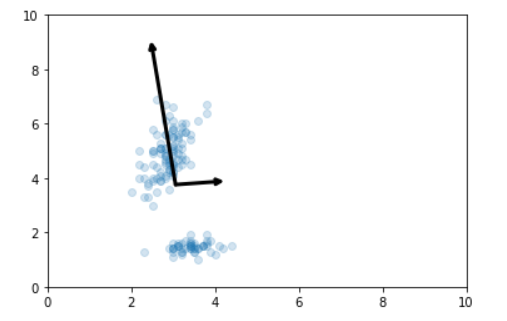
plt.axis([0, 10, 0, 10])



**Рис.2.** Візуалізація обрахованого РСА з використанням функції sklearn, але з використання PyTorch.

1. Порівняти результати отримані компоненти, порівняти їхні результати. Вони повинні співпадати при правильній реалізації.

Після ретельного порівняння результатів можна стверджувати, що вони співпадають. Отже реалізація вірна.



**Рис.3.** Порівняння результатів із різною реалізацією(зліва з використанням sklearn, справа – PyTorch)

1. Добавити до даних шум:

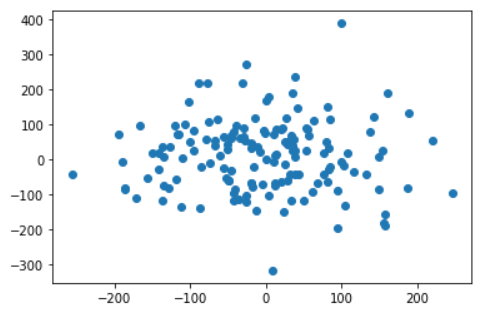
np.random.seed(42)

noisy = X

noisy[:, 0] = np.random.normal(X[:, 0], 45)

noisy[:, 1] = np.random.normal(X[:, 1], 45)

plt.scatter(noisy[:, 0], noisy[:, 1])

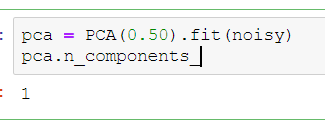


**Рис.4.** Дані після добавлення шуму

1. Обрахувати на нових зашумлених даних PCA:

pca = PCA(0.50).fit(noisy)

pca.n\_components\_



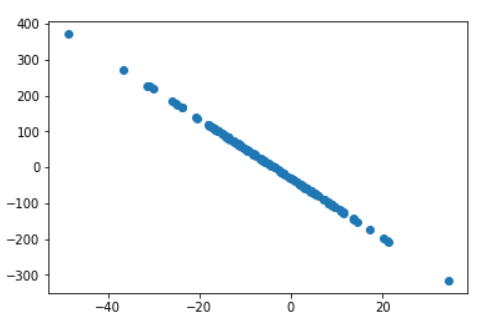
**Рис.5.** Обраховане РСА після зашумлення

1. Реконстроювати дані використовуючи PCA для знешумлення:

components = pca.transform(noisy)

filtered = pca.inverse\_transform(components)

plt.scatter(filtered[:, 0], filtered[:, 1])



**Рис.6.** Реконструйовані дані з використанням РСА для знешумлення

**Висновок:**

На лабораторній роботі було засвоєно основні відомості про роботу з алгоритмом РСА та реалізовано даний алгоритм.

**Код програми:**

import torch

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

from sklearn.decomposition import PCA

iris = sns.load\_dataset("iris")

features = ['sepal\_width', 'petal\_length']

X = iris.loc[:, features].values

Iris

plt.scatter(X[:, 0], X[:, 1])

pca = PCA(n\_components=2)

pca.fit(X)

print(pca.components\_)

print(pca.explained\_variance\_)

def draw\_vector(v0, v1, ax=None):

ax = ax or plt.gca()

arrowprops=dict(arrowstyle='->',

linewidth=3,

shrinkA=0, shrinkB=0)

ax.annotate('', v1, v0, arrowprops=arrowprops)

# plot data

plt.scatter(X[:, 0], X[:, 1], alpha=0.2)

for length, vector in zip(pca.explained\_variance\_, pca.components\_):

v = vector \* 3 \* np.sqrt(length)

draw\_vector(pca.mean\_, pca.mean\_ + v)

plt.axis([0, 10, 0, 10])

def PCA2(X, k, center=True):

n = X.size()[0]

ones = torch.ones(n).view([n,1])

h = ((1/n) \* torch.mm(ones, ones.t())) if center else torch.zeros(n\*n).view([n,n])

H = torch.eye(n) - h

X\_center = torch.mm(H.double(), X.double())

u, s, v = torch.svd(X\_center)

explained\_variance = torch.mul(s[:k], s[:k])/(n-1)

return {'components':components,

'explained\_variance':explained\_variance }

pca2 = PCA2(torch.IntTensor(X), 2)

pca2

plt.scatter(X[:, 0], X[:, 1], alpha=0.2)

for length, vector in zip(pca2['explained\_variance'], pca2['components']):

v = vector \* 3 \* np.sqrt(length)

draw\_vector(pca.mean\_, pca.mean\_ + v.data.numpy())

plt.axis([0, 10, 0, 10])

plt.scatter(X[:, 0], X[:, 1])

np.random.seed(42)

noisy = X

noisy[:, 0] = np.random.normal(X[:, 0], 1)

noisy[:, 1] = np.random.normal(X[:, 1], 1)

plt.scatter(noisy[:, 0], noisy[:, 1])

pca = PCA(0.50).fit(noisy)

pca.n\_components\_

components = pca.transform(noisy)

filtered = pca.inverse\_transform(components)

plt.scatter(filtered[:, 0], filtered[:, 1])