Algebra liniară pentru învăţarea automată

Algebra liniară este o ramură a matematicii, de fapt, algebra liniară este matematica datelor. Matricele și vectorii sunt limbajul datelor. Algebra liniară înseamnă de fapt combinații liniare. Asta înseamnă, că folosim aritmetica pe coloane de numere, care sunt numite vectori și tablouri 2D, numite matrici, pentru a crea noi coloane și matrici de numere.

Aplicarea algebrei liniare în informatică este numită algebră liniară numerică. Aceasta este mai mult decât codificarea operațiilor algebrei liniare în biblioteci de cod, ci include, de şi manipularea atentă a problemelor matematicii aplicate, cum ar fi implementarea operaţiilor cu virgulă.

Uneltele puse la dispoziţie de algebra liniară sunt folosite în multe domenii, cum ar fi:

- Matricile în inginerie.
- Grafice și rețele, cum ar fi analizarea rețelelor.
- Şirurile Markov sunt folosite în economie, şi în analiza creșterii populației.
- Programarea liniară, metoda de optimizare simplex.
- Serii Fourier, algebra liniară pentru funcții utilizată pe scară largă în procesarea semnalelor.
- Algebra liniară pentru statistică și probabilități, de exemplu, metoda celor mai mici pătrate pentru regresie.
- Grafică, cum ar fi diverse translatări, scalări și rotiri ale imaginilor.

Vectori

Un vector este un grup de una sau mai multe valori numite scalari. Vectorii sunt adesea reprezentați folosind a caractere mici, cum ar fi v; de exemplu:

$$v = (v_1, v_2, v_3)$$

unde v1, v2, şi v3 sunt valori reale.

Definirea unui vector

Putem reprezenta un vector în Python ca un obiect NumPy. Se poate crea un obiect array NumPy pornind de la o listă de numere. De exemplu, mai jos avem un vector cu lungimea de 3 şi format din valorile întregi: 1, 2 și 3.

```
# create a vector
from numpy import array
v = array([1, 2, 3])
print(v)
```

Înmulţirea vectorilor

Doi vectori de aceeaşi lungime pot fi înmulţiţi:

$$c = a \times b$$

La fel ca şi adunarea şi scăderea, această operaţie se face element-cu-element:

$$a \times b = (a_1 \times b_1, a_2 \times b_2, a_3 \times b_3)$$

Această operaţie se poate face direct în Numpy:

```
# multiply vectors
from numpy import array
a = array([1, 2, 3])
print(a)
b = array([1, 2, 3])
print(b)
c = a * b
print(c)
```

Exerciţiu

Implementaţi alte operaţii cu vectori: adunarea, scăderea şi produsul scalar.

Matrici

O matrice este un tablou bidimensional de scalari, cu una sau mai multe coloane și unul sau mai multe rânduri.

Notația pentru o matrice este de obicei o literă majusculă, cum ar fi A, iar intrările sunt menționate de către indicele bidimensional pentru rând (i) și coloană (j), cum ar fi a_ij. De exemplu:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix}$$

Definirea unei matrici

Putem reprezenta o matrice în Python folosind un tablou numeric bidimensională. O matrice NumPy poate fi construită pe baza unei liste de liste. De exemplu, mai jos este definită o matrice de 2 rânduri, 3 coloane:

```
# create matrix
from numpy import array
A = array([[1, 2, 3], [4, 5, 6]])
print(A)
```

Adunarea matricilor

Două matrici de aceeaşi dimensiune pot fi adunate pentru a crea o a treia matrice:

$$C = A + B$$

Elementele scalare din matricea rezultată sunt calculate prin adunarea elementelor de pe aceeaşi poziţie din cele două matric care se adună. Putem implementa acest lucru în Python folosind operatorul plus direct pe cele două matrice numerice.

```
# add matrices
from numpy import array
A = array([[1, 2, 3], [4, 5, 6]])
print(A)
B = array([[1, 2, 3], [4, 5, 6]])
print(B)
C = A + B
print(C)
```

Înmulţirea matriceală

Are loc doar în anumite condiţii: numărul de coloane din prima matrice trebuie să fie egal cu numărul de linii din a doua matrice:

$$C(m, k) = A(m, n) \times B(n, k)$$

De fapt, înmulţirea matriceală calculează produsul scalar dintre fiecare rând din matricea A cu fiecare coloană din matricea B.

Înmulţirea matriceală se implementează în NumPy folosind funcţia: dot()

```
# matrix dot product
from numpy import array
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
B = array([[1, 2], [3, 4]])
print(B)
C = A.dot(B)
print(C)
```

Exerciţiu

Implementaţi alte operaţii cu matrici: scăderea, împărţirea, produsul Hadamard, şi înmulţirea vector-matrice.

Tipuri de matrici şi operaţii cu matrici

Transpusa

O matrice definită poate fi transpusă, ceea ce creează o nouă matrice cu coloanele și rândurile inversate. Acest lucru este notat de superscriptul T:

$$C = A^T$$

În NumPy acest lucru se realizează prin apelarea operatorului .T:

```
# transpose matrix
from numpy import array
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
C = A.T
print(C)
```

Inversa

Operaţia de inversare a unei matrici este indicată de un superscript -1 lângă matrice. Rezultatul operației este denumita inversa matricei inițiale, de exemplu, B este inversa lui A.

$$B = A^{-1}$$

Nu toate matricile sunt invertibile!
În NumPy, inversa unei matrici se calculează folosind funcţia inv():

```
# invert matrix
from numpy import array
from numpy.linalg import inv
# define matrix
A = array([[1.0, 2.0], [3.0, 4.0]])
print(A)
# invert matrix
B = inv(A)
print(B)
```

Matrici pătratice

O matrice pătrată este o matrice în care numărul de rânduri (n) este egal cu numărul de coloane (m).

$$n = m$$

Spre deosebire de matricea pătratică, în cazul matricei oarecare numărul de rânduri și numărul de coloane nu sunt egale.

Matrice simetrică

O matrice simetrică este un tip de matrice pătratică în care partea de deasupra diagonalei principale (triunghiul dreapta-sus) este egală cu partea de dedesubtul diagonalei principale (triunghiul stânga-jos). Pentru a fi simetrică, axa simetriei este întotdeauna diagonala principală a matricei, din colţul stânga-sus la colţul dreapta-jos. O matrice simetrică este întotdeauna pătratică și egală cu propria transpunere.

$$M = M^T$$

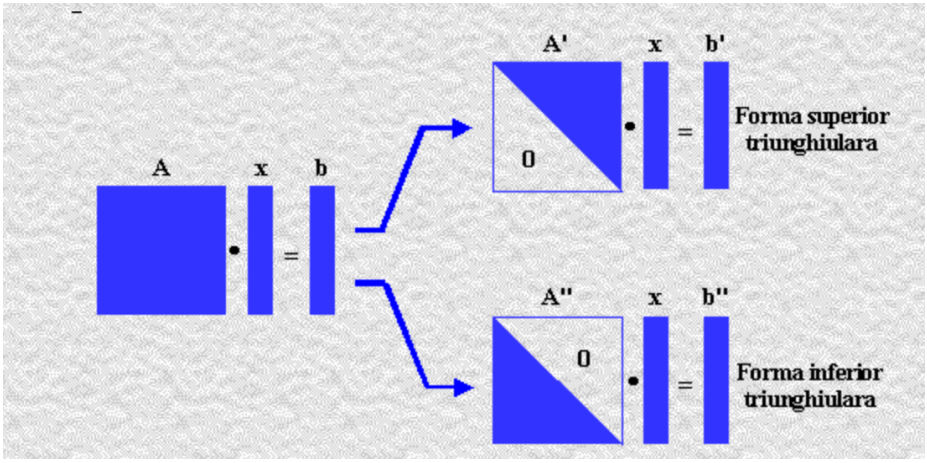
Matrice trianghiulară

O matrice triunghiulară este un tip de matrice pătratică, care are toate valorile din partea superioară dreaptă sau din stânga inferioară a matricei, zero. O matrice triunghiulară cu valori numai deasupra diagonalei principale se numește o matrice triunghiulară superioară. O matrice triunghiulară cu valori numai sub diagonala principală se numește o matrice triunghiulară inferioară.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$

$$A = \begin{pmatrix} a_{11} & 0 & 0 & \dots & 0 \\ a_{21} & a_{22} & 0 & \dots & 0 \\ a_{31} & a_{32} & a_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

Unde se folosesc matricile triunghiulare:



Matrice diagonală

O matrice diagonală este una în care valorile din afara diagonalei principale au o valoare zero, unde diagonala principală este luată din partea stângă sus a matricei în dreapta jos. O matrice diagonal este deseori desemnată cu variabila D și poate fi reprezentată ca o matrice completă sau ca un vector conţinând valorile de pe diagonala principală.

Exerciţiu

Exemple pentru alte operații cu matrici, cum ar fi determinantul, urma și rangul unei matrici.

Factorizarea matricilor

Se mai numeşte şi descompunerea matricilor. Descompunerea matricei este o modalitate de a reduce o matrice în părțile componente ale acesteia. Este o abordare care poate simplifica operațiile mai complexe cu matrici care pot fi efectuate mai uşor pe matricea descompusă decât pe matricea originală. O analogie pentru descompunerea matricilor este factorizarea numerelor, cum ar fi factorizarea lui 25 în 5 şi 5. Din acest motiv, descompunerea matricilor mai este numită și factorizare. La fel ca și factorii valorilor reale, există mai multe metode de a descompune o matrice, deci există o serie de tehnici diferite de descompunere a matricilor.

Descompunerea LU

Descompunerea LU este pentru matricile pătratice și descompune o matrice în componentele L și U.

$$A = L \cdot U$$

Unde A este matricea pătratică pe care dorim să o descompunem, L este matricea triunghiulară inferioară și U este matricea triunghiulară superioară. O variație a acestei descompuneri care este mai stabile numeric în practică se numește descompunerea LUP sau descompunerea LU cu pivotare parțială.

$$A = P \cdot L \cdot U$$

Rândurile matricei originale A sunt rearanjate pentru a simplifica procesul de descompunere și matricea suplimentară P specifică o modalitate de a permuta rezultatul sau de a returna rezultatul original.

Descompunerea LU este adesea folosită pentru simplificare rezolvării sistemelor de ecuații liniare, cum ar fi calcularea coeficienților într-o regresie liniară.

Descompunerea LU poate fi implementată în Python cu funcția lu ().

```
# LU decomposition
from numpy import array
from scipy.linalg import lu
# define a square matrix
```

```
A = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(A)
# LU decomposition
P, L, U = lu(A)
print(P)
print(L)
print(U)
# reconstruct
B = P.dot(L).dot(U)
print(B)
```

Exerciţiu
Implementaţi alte exemple de factorizare a matricilor cum ar fi descompunerea QR, descompunerea Cholesky.

Descompunerea valorilor singulare

Descompunerea valorilor singulare, sau descompunerea SVD, este o metodă de descompunere a unei matrici în părţile ei constituente, cu scopul de a simplifica anumite operaţii cu matrici:

$$A = U \cdot \Sigma \cdot V^T$$

unde A este matricea originală, de dimensiuni *mxn*, pe care dorim să o descompunem. U este o matrice *mxm*, Sigma este o matrice *mxn* şi V este o matrice *nxn.*

Descompunerea SVD se face prin apelarea funcţiei svd():

```
# Singular-value decomposition
from numpy import array
from scipy.linalg import svd
# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
# SVD
U, s, V = svd(A)
print(U)
print(s)
print(V)
```

unde s reprezintă vectorul valorilor proprii ale matricei A.

Reconstruirea matricei originale

```
# Reconstruct SVD
from numpy import array
from numpy import diag
from numpy import dot
from numpy import zeros
from scipy.linalg import svd
# define a matrix
A = array([[1, 2], [3, 4], [5, 6]])
print(A)
# Singular-value decomposition
U, s, VT = svd(A)
# create m x n Sigma matrix
Sigma = zeros((A.shape[0], A.shape[1]))
# populate Sigma with n x n diagonal matrix
Sigma[:A.shape[1], :A.shape[1]] = diag(s)
# reconstruct matrix
B = U.dot(Sigma.dot(VT))
print(B)
```

În cazul în care A este o matrice pătratică, există o metodă mai simplă pentru reconstrucţie:

```
# Reconstruct SVD
from numpy import array
from numpy import diag
from numpy import dot
from scipy.linalg import svd
# define a matrix
A = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(A)
# Singular-value decomposition
U, s, VT = svd(A)
# create n x n Sigma matrix
Sigma = diag(s)
# reconstruct matrix
B = U.dot(Sigma.dot(VT))
print(B)
```

Descompunerea SVD pentru reducerea dimensionalităţii

Reducerea dimensionalităţii este o aplicație populară a descompunerii SVD.

Datele cu un număr mare de caracteristici, adică în cazul în care numărul de caracteristici (coloane) este mai mare decât numărul de observații (rânduri), pot fi reduse la un subset mai mic de caracteristici relevante pentru problema de predicție. Rezultatul este o matrice cu un rang inferior despre care se spune că aproximează matricea originală.

Pentru a face acest lucru, putem efectua o operație de descompunere SVD pe datele originale și selectăm cele mai mari k valori singulare din Sigma. Coloanele vor fi selectate din Sigma și rândurile vor fi selectate din V^T. După care, poate fi reconstruită o aproximație B a matricei originale A, considerând doar primele k coloane din matricea Sigma şi primele k rânduri din matricea V^T:

```
B = U . Sigmak . V^Tk
```

În practică, putem lucra cu un set descriptiv de date pe care îl numim T, care reprezintă un rezumat dens al datelor:

```
T = U . Sigmak
```

Această transformare T poate fi aplicată matricei:

```
T = V^Tk . A
```

Exemplu:

Mai întâi este definită o matrice de 3 × 10, cu mai multe coloane decât rânduri. Descompunerea SVD este calculată și sunt selectate numai primele două caracteristici. Elementele sunt recombinate pentru a da o reproducere exactă a matricei originale. În cele din urmă, transformarea este calculată în două moduri diferite.

```
from numpy import array
from numpy import diag
from numpy import zeros
from scipy.linalg import svd
# define a matrix
A = array([
 [1,2,3,4,5,6,7,8,9,10],
 [11,12,13,14,15,16,17,18,19,20],
 [21,22,23,24,25,26,27,28,29,30]])
print(A)
# Singular-value decomposition
U, s, VT = svd(A)
# create m x n Sigma matrix
Sigma = zeros((A.shape[0], A.shape[1]))
# populate Sigma with n x n diagonal matrix
Sigma[:A.shape[0], :A.shape[0]] = diag(s)
# select
n_elements = 2
Sigma = Sigma[:, :n_elements]
```

```
VT = VT[:n_elements, :]
# reconstruct
B = U.dot(Sigma.dot(VT))
print(B)
# transform
T = U.dot(Sigma)
print(T)
T = A.dot(VT.T)
print(T)
```

Rularea exemplului de mai sus afişează mai întâi matricea iniţială, apoi aproximarea acesteia, urmată de două transformări echivalente ale matricei originale.

Biblioteca Scikit-learn oferă o clasă TruncatedSVD care implementează această reducere a dimensionalităţii direct. Se poate apela clasa TruncatedSVD în care trebuie să specificate numărul de caracteristici sau componente dorite, de ex. 2. Odată apelată această clasă, se poate obţine transformarea apelând funcția fit(), apoi se aplică la matricea originală apelând funcția transform(). Rezultatul este transformarea lui A, numită T mai sus.

Exemplul de mai jos demonstrează utilizarea clasei TruncatedSVD.

```
from numpy import array
from sklearn.decomposition import TruncatedSVD
# define array
A = array([
 [1,2,3,4,5,6,7,8,9,10],
 [11,12,13,14,15,16,17,18,19,20],
 [21,22,23,24,25,26,27,28,29,30]])
print(A)
# svd
svd = TruncatedSVD(n_components=2)
svd.fit(A)
result = svd.transform(A)
print(result)
```

Rularea exemplului de mai sus afişează mai întâi matricea originală, urmată de transformarea acesteia într-o o matrice cu 2 coloane.

Putem observa că valorile se potrivesc cu valorile calculate mai sus, cu excepția semnului pentru anumite valori. Ne putem aștepta la o anumită instabilitate în ceea ce privește semnul, data fiind natura calculelor implicate și diferențele dintre bibliotecile și metodele utilizate. Această instabilitate a semnului nu ar trebui să fie o problemă în practică.

Exerciţiu

Descărcaţi un set de date de pe UCI Repository (https://archive.ics.uci.edu/ml/index.php) pentru care numărul de caracteristici este mai mare decât numărul de observaţii. Faceţi o reducere a dimensionalităţii folosind metoda SVD astfel încât datele obţinute să aibă un număr de caracteristici mai mic decât numărul de observaţii.

Bibliografie pentru lucrul cu matici:

http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/imm3274.pdf

Statistica pentru învăţarea automata

Statistica reprezintă un domeniu al matematicii care conţine o colecție de metode de lucru cu asupra datelor, scopul fiind acela de a folosi datele pentru a răspunde la diferite întrebări.
În funcţie de scopul pentru care sunt folosite, metodele statistice se împarte în două categorii:

- **Statistici descriptive** se referă la metode folosite pentru sintetizarea datelor, adică la metode de rezumare a datelor brute în informații pe care le putem înțelege și împărtăși.
- **Statistici inferențiale** folosite pentru a trage concluzii din eșantioane de date. Sunt folosite pentru a cuantifica proprietățile unei populații pe baza unui set mai mic de observații numit eșantion.

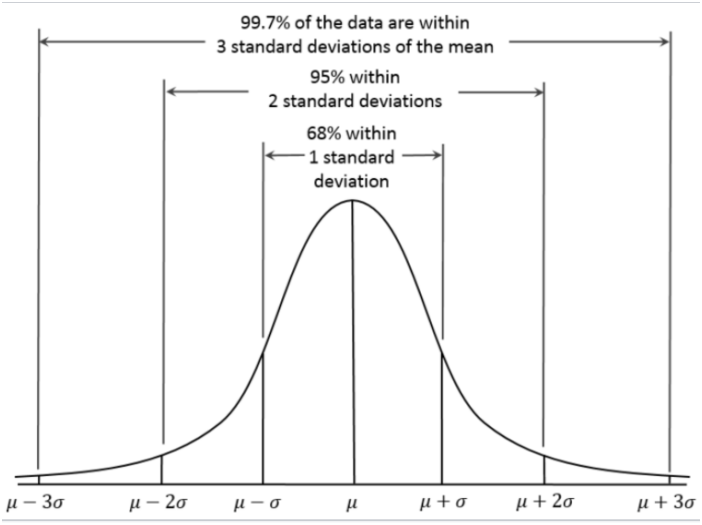
Distribuţia Gaussiană şi statisticile descriptive

Un eșantion de date este un instantaneu dintr-o populație mai mare de posibile observații dintr-un domeniu sau care ar putea fi generate de un proces. În mod interesant, multe observații urmează un model comun sau o distribuție, numită distribuție normală sau mai formal distribuția Gaussiană.

Se cunosc foarte multe detalii despre distribuția Gaussiană și, ca atare, există multe metode statistice care pot fi utilizate cu date care urmează o distribuţie Gaussiană. Orice distribuție Gaussiană, și de fapt, orice eşantion de date extras dintr-o distribuție Gaussiană, poate fi rezumat doar de doi parametri:

- Media: tendința centrală sau valoarea cea mai probabilă în distribuție
- Varianţa: diferența medie pe care o au observațiile de la valoarea medie(răspândirea).

Unitățile de măsură pentru medie sunt aceleași ca și unitățile de măsură pentru varianţă, deși unitățile de măsură pentru varianță sunt luate la pătrat și, prin urmare, mai greu de interpretat. O alternativă populară pentru variață este deviația standard, care este pur și simplu rădăcina pătrată a varianței.



Media, varianța și deviaţia standard pot fi calculate direct pe eșantioane de date în NumPy. Exemplul de mai jos generează un eșantion de 100 de numere aleatorii extrase dintr-o distribuție Gaussiană cu o valoare mediei de 50 și o abatere standard de 5 și calculează statisticile desciptive.

```
# calculate summary stats
from numpy.random import seed
from numpy.random import randn
from numpy import mean
from numpy import var
from numpy import std
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# calculate statistics
print('Mean: %.3f' % mean(data))
print('Variance: %.3f' % var(data))
print('Standard Deviation: %.3f' % std(data))
```

Exerciţiu
Implementaţi în Python funcţii pentru calcularea statisticilor descriptive: media, varianța și deviaţia standard.

Corelaţia dintre variabile
Este util în analiza și modelarea datelor să înțelegem relațiile dintre variabile. Relația statistică dintre două variabile este denumită corelația. Corelația este pozitivă, adică ambele variabile cresc sau descresc în aceeași direcție sau negativă, ceea ce înseamnă că atunci când valoarea unei variabile crește, valorile celeilalte variabile scad.

- Corelație pozitivă: ambele variabile se modifică în aceeași direcție.
- Corelație neutră: nici o relație în schimbarea variabilelor.
- Corelație negativă: variabilele se schimbă în direcții opuse.

Performanța unor algoritmi poate fi scăzută dacă două sau mai multe variabile sunt strânse corelate, fapt numit multicoliniaritate. În acest caz, unele dintre variabilele corelate ar trebui eliminate pentru a îmbunătăți performanţa modelului.
Cuantificarea corelaţiei dintre eșantioanele a două variabile se face utilizând o metodă statistică numită Coeficientul de corelație al lui Pearson, numit aşa după cel care a dezvoltat metoda, Karl Pearson. Funcția NumPy Pearsonr() poate fi folosită pentru a calcula coeficientul de corelație Pearson pentru două eşantioane. Exemplul de mai jos arată calculul corelaţiei în cazul în care o variabilă depinde de a doua.

```
# calculate correlation coefficient
from numpy.random import seed
from numpy.random import randn
from scipy.stats import pearsonr
# seed random number generator
seed(1)
# prepare data
data1 = 20 * randn(1000) + 100
data2 = data1 + (10 * randn(1000) + 50)
# calculate Pearson's correlation
corr, p = pearsonr(data1, data2)
# display the correlation
print('Pearsons correlation: %.3f' % corr)
```

Exerciţiu

Încărcaţi setul de date Iris de pe UCI repository şi calculaţi corelaţia dintre cele 4 variabile (lungimea petalei, lăţimea petalei, lungimea sepalei, lăţimea sepalei)

Teste statistice

Testele statistice sunt folosite pentru a compara datele provenite din două eşantioane. Datele trebuie interpretate pentru a le înţelege. Putem interpreta datele presupunând o structură specială a rezultatelor obţinute și utilizarea metodelor statistice pentru a accepta sau a respinge prezumpţia noastră. Prezumția noastră se numește ipoteză, iar testele statistice utilizate în acest scop sunt numite teste statistice.

Presupunerea unui test statistic se numește ipoteza nulă sau ipoteza zero (H0 pentru scurt). Acesta este adesea numit ipoteza implicită, sau ipoteza că nimic nu a fost schimbat. O încălcare a ipotezei testului statistic este adesea numită prima ipoteză, sau H1 pe scurt.

- Ipoteza 0 (H0): Ipoteza testului este validă și nu a fost respinsă.
- Ipoteza 1 (H1): Ipoteza testului nu este validă și este respinsă la un anumit nivel de semnificație.

Putem interpreta rezultatul unui test statistic folosind valoare p. Valoarea p este probabilitatea de observare a datelor, în condiţiile în care ipoteza nulă este adevărată. O probabilitate mare înseamnă că ipoteza H0 sau ipoteza implicită este probabilă. O valoare mica a probabilităţii p, cum ar fi sub 5% (0,05) sugerează că este puţin probabil și că putem respinge H0 în favoarea ipotezei H1.

Un test de ipoteză statistică utilizat pe scară largă este Testul-T Student care este folosit pentru compararea mediilor a două eșantioane independente. Ipoteza implicită este că nu există nici o diferență între eșantioane, în timp ce o respingere a acestei ipoteze sugerează o diferență semnificativă între eşantionae. Testul presupun că ambele eșantioane au fost extrase dintr-o distribuție Gaussiană și au aceeași varianță.

Testul-T Student poate fi implementat în Python prin intermediul funcţiei ttest_ind (). Mai jos este un exemplu de calculare și interpretare a testului-t Student pentru două eşantioane diferite.

```
# student's t-test
from numpy.random import seed
from numpy.random import randn
from scipy.stats import ttest_ind
# seed the random number generator
seed(1)
# generate two independent samples
data1 = 5 * randn(100) + 50
data2 = 5 * randn(100) + 50
# compare samples
stat, p = ttest_ind(data1, data2)
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
    print('Same distributions (fail to reject H0)')
else:
    print('Different distributions (reject H0)')
```

Exerciţiu

Aplicaţi alte 2 teste statistice care pot fi folosite pentru a testa diferenţele dintre două eşantioane (sugestie: reamintire de la cursul de Software matematic: Funcţii statistice)