**Universitatea “DUNĂREA DE JOS” Galați**

Facultatea de Automatică, Calculatoare, Inginerie Electrica și Electronică,

Specializarea Calculatoare

**Tema 1**

**kNN**

**Căutarea liniara vs. k-D Tree**

S.l.dr.inf. Student:

Crăciun Viorel Marian Abeaboeru Alexandru Cristian

Cuprins

[Introducere 3](#_Toc150115352)

[Implementarea metodelor 4](#_Toc150115353)

[Setul de date 5](#_Toc150115354)

[Rezultate și analiză 5](#_Toc150115355)

[Concluzie 6](#_Toc150115356)

[Bibliografie: 7](#_Toc150115357)

# Introducere

KNN

K-Nearest Neighbors este un algoritm de învățare supervizată folosit în principal pentru clasificare și regresie. Acest algoritm se bazează pe ideea că obiectele similare sunt grupate împreună în spațiul caracteristic.

KNN este adesea utilizat pentru clasificare. Atunci când se dă un nou punct de date, algoritmul încearcă să determine clasa sa în baza etichetelor claselor din setul de date de antrenament. Acest lucru se face prin găsirea K cele mai apropiate puncte din setul de date de antrenament în funcție de o metrică de distanță (de obicei, distanța euclidiană sau distanța Manhattan).

KNN poate fi folosit și pentru regresie. În acest caz, în loc de clasificare, algoritmul estimează o valoare numerică pentru punctul de interogare, pe baza valorilor din setul de date de antrenament pentru cele mai apropiate K puncte, K fiind numărul de vecini apropiați pe care algoritmul îi va lua în considerare pentru a lua o decizie.

KNN are avantaje, cum ar fi ușurința de învățare și capacitatea de a se adapta la date non-lineare, dar are și dezavantaje, cum ar fi sensibilitatea la datele zgomotoase și necesitatea de a stoca întregul set de date de antrenament în memorie pentru a face predicții. Este potrivit pentru probleme mici până la medii și este utilizat în diverse domenii, inclusiv recunoaștere de imagini, clasificare de texte, filtrarea de recomandări, analiza datelor și multe altele.

A screenshot of a video game

Description automatically generated

Exemplu de clasificare k-NN. Punct verde trebuie clasificat

fie în pătrate albastre, fie în triunghiuri roșii. Dacă k = 3 (cerc linie continuă) se atribuie triunghiurilor roșii deoarece există 2 triunghiuri și doar 1 pătrat în interiorul cercului interior. Dacă k = 5 (cerc cu linie întreruptă) se atribuie pătratelor albastre (3 pătrate vs. 2 triunghiuri în interiorul cercului exterior).

Căutarea liniara

Căutarea liniară (Linear Search) este o metodă simplă de găsire a unui element într-un set de date, constând în parcurgerea tuturor elementelor din setul de date în ordine și compararea fiecărui element cu valoarea căutată până când se găsește o potrivire sau se ajunge la sfârșitul setului de date.

Eficiența căutării liniare depinde de dimensiunea setului de date și poate deveni ineficientă pentru seturi de date mari.

Căutarea liniară este utilă atunci când setul de date este mic sau când se efectuează căutări ocazionale într-un set de date.

K-D Tree

K-Dimensional Tree este o structură de date utilizată pentru a accelera căutarea în spații multidimensionale, precum în KNN. K-D Tree porționează spațiul multidimensional în subspații mai mici, ceea ce reduce numărul de comparații necesare pentru a găsi vecinii apropiați.

K-D Tree poate fi eficient pentru seturi de date de dimensiuni mai mari, dar necesită construirea și menținerea unui arbore, ceea ce poate consuma resurse suplimentare.

# Implementarea metodelor

# Extrage datele și etichetele din setul de date

array = dataset.values

X\_train = array[0:1700, 0:8]

X\_validation = array[0:17898, 0:8]

Y\_train = array[0:1700, 8]

Y\_validation = array[0:17898, 8]

# Aplică scalarea Min-Max la datele de antrenare și validare

minmaxscale = MinMaxScaler().fit(X\_train)

X\_train = minmaxscale.transform(X\_train)

X\_validation = minmaxscale.transform(X\_validation)

# Cautare liniara, măsoară timpul de execuție pentru evaluarea modelului cu validare încrucișată

model = KNeighborsClassifier(n\_neighbors=41)

kfold = StratifiedKFold(n\_splits=10, random\_state=1, shuffle=True)

start\_time = time.time()

cv\_results = cross\_val\_score(model, X\_train, Y\_train, cv=kfold, scoring='accuracy')

end\_time = time.time()

execution\_time = end\_time - start\_time

# Afișează scorul de acuratețe și timpul de execuție pentru KNN cu căutare liniară

print('%s (Cautare Liniara): Acuratețe - %f (%f)' % ("kNN", cv\_results.mean(), cv\_results.std()))

print('%s (Cautare Liniara): Timpul de Execuție - %f secunde\n' % ("kNN", execution\_time))

# k-D Tree, măsoară timpul de execuție pentru evaluarea modelului k-D Tree cu validare încrucișată

model = KNeighborsClassifier(n\_neighbors=41, algorithm='kd\_tree')

kfold = StratifiedKFold(n\_splits=10, random\_state=1, shuffle=True)

start\_time = time.time()

cv\_results = cross\_val\_score(model, X\_train, Y\_train, cv=kfold, scoring='accuracy')

end\_time = time.time()

execution\_time = end\_time - start\_time

# Afișează scorul de acuratețe și timpul de execuție pentru KNN cu algoritm k-D Tree

print('%s (k-D Tree): Acuratețe - %f (%f)' % ("kNN", cv\_results.mean(), cv\_results.std()))

print('%s (k-D Tree): Timpul de Execuție - %f secunde\n' % ("kNN", execution\_time))

# Setul de date

HTRU2 este un set de date din cadrul Studiului Universului cu Înaltă Rezoluție Temporală (High Time Resolution Universe Survey - South) care conține informații despre candidații pentru pulsari. În total, setul de date conține 17.898 de exemple, dintre care 1.639 sunt pulsari reali, iar 16.259 sunt semnale de zgomot sau interferență. Acest set de date ajută la identificarea pulsarilor reali printre candidații falși, pe baza acestor valori numerice.

Fiecare candidat este descris printr-o serie de 8 valori numerice:

1. Media profilului de impuls integrat.

2. Deviația standard a profilului de impuls integrat.

3. Kurtosisul excesiv al profilului de impuls integrat.

4. Asimetria profilului de impuls integrat.

5. Media curbei DM-SNR (raport semnal-zgomot în funcție de dispersia în timp).

6. Deviația standard a curbei DM-SNR.

7. Kurtosisul excesiv al curbei DM-SNR.

8. Asimetria curbei DM-SNR.

# Rezultate și analiză

A close up of a paper

Description automatically generated

Durata de antrenare in weka experiment enviroment:

Căutare binara: 1.25 ms

K-D Tree: 37.81 ms

A close up of a number

Description automatically generated

Durata de testare weka experiment enviroment:

Căutare binara: 1335.63 ms

K-D Tree: 134.06 ms

A close up of a computer code

Description automatically generated

Acuratețea algoritmilor in weka experiment enviroment:

Căutare binara: 97.73%

K-D Tree: 97.73%

A close-up of a number

Description automatically generated Acuratețea si durata de testare in Google Colab:

Căutare binara: 97.11% 230.53 ms

K-D Tree: 97.11% 155.56 ms

# Concluzie

Căutarea liniară este simplă și ușor de implementat, dar poate fi ineficientă pentru seturi de date mari, în timp ce K-D Tree poate accelera căutarea pentru dimensiuni mai mari.

K-D Tree necesită construirea unei structuri de date complexe înainte de a putea fi utilizată, în timp ce căutarea liniară nu necesită construcția unui arbore.

Căutarea liniară poate fi potrivită pentru seturi de date mici sau pentru scenarii unde eficiența nu este o preocupare majoră, în timp ce K-D Tree este utilă pentru seturi de date mari sau atunci când vrei să optimizezi timpul de căutare.

# Bibliografie:

https://archive.ics.uci.edu/dataset/372/htru2

https://en.wikipedia.org/wiki/K-nearest\_neighbors\_algorithm

https://en.wikipedia.org/wiki/K-d\_tree

https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/