**MINISTERUL EDUCAŢIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**Departamentul Ingineria Software și Automatică**

**Programul de studii: Tehnologia informației**



**RAPORT**

**Disciplina „Inteligenta Artificiala”**

**Tema:** **Tehnici de clasificare supervizată**

|  |  |  |
| --- | --- | --- |
| **Student(ă):** | **\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Vlași**ț**chi Ștefan , TI-216** |
|  |  |  |
| **Coordonator universitate:** | **\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Viorel Rusu , asist.univ.** |

**Chișinău, 2024**

**Sarcini practice:**

**Clasificare Supervizată - Implementare în Python**

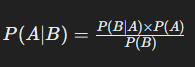
**Bazate pe criteriul Bayes - Clasificator Naiv Bayes:**

Concept: Clasificatorul Naiv Bayes se bazează pe teorema lui Bayes și presupune independența condiționată a fiecărui atribut față de ceilalți, dată fiind clasa.

Pași de bază:

1. Calculul probabilităților a priori pentru fiecare clasă.
2. Calculul probabilităților condiționate pentru fiecare atribut dată fiind clasa.
3. Utilizarea teoremei lui Bayes pentru a calcula probabilitatea condiționată inversă.
4. Clasificarea se realizează prin selectarea clasei cu cea mai mare probabilitate a posteriori.

Set de date: Poate fi utilizat setul de date Iris sau altul potrivit pentru clasificare.



P(A∣B) este probabilitatea condiționată că A să se întâmple dat fiind B,

P(B∣A) este probabilitatea condiționată că B să se întâmple dat fiind A,

P(A) este probabilitatea anterioară (prior) a lui A,

P(B) este probabilitatea anterioară (prior) a lui B.

# Importăm bibliotecile necesare  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.datasets import make\_classification  
from sklearn.model\_selection import train\_test\_split  
# Importăm biblioteca pentru evaluare  
from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score  
  
# Generăm un set de date sintetic pentru scopuri de demonstrație  
X, y = make\_classification(n\_samples=1000, n\_features=20, n\_classes=2, random\_state=42)  
  
# Împărțim setul de date în set de antrenare și set de testare  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Inițializăm clasificatorul Clasificator Naiv Bayes  
naive\_bayes\_clf = GaussianNB()  
  
# Antrenăm clasificatorul Clasificator Naiv Bayes pe setul de antrenare  
naive\_bayes\_clf.fit(X\_train, y\_train)  
  
# Facem predicții pe setul de testare  
y\_pred = naive\_bayes\_clf.predict(X\_test)  
  
# Calculăm acuratețea predicțiilor  
accuracy = accuracy\_score(y\_test, y\_pred)  
print("Acuratețea clasificatorului Naiv Bayes:", accuracy)  
  
# Calculăm alte metrici de evaluare  
print("\nMatricea de Confuzie:")  
print(confusion\_matrix(y\_test, y\_pred))  
  
print("\nRaportul de Clasificare:")  
print(classification\_report(y\_test, y\_pred))

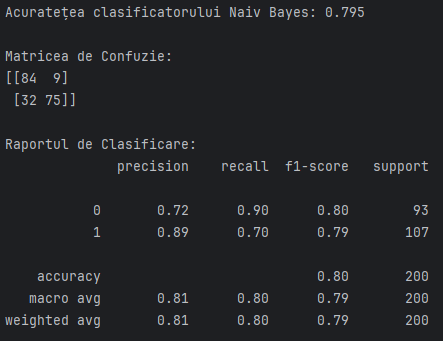


Figura 1 – rezultatele metodei Clasificator Naiv Bayes

**Bazate pe Funcții - Support Vector Machines (SVM):**

Concept: SVM optimizează un hiperplan în spațiul de caracteristici care maximizează separarea între clasele de date.

Pași de bază:

1. Reprezentarea datelor într-un spațiu de caracteristici.
2. Identificarea hiperplanului care maximizează marginea (distanța) între cele două clase.
3. Optimizarea hiperplanului prin minimizarea erorii și maximizarea marginii.

Set de date: Poate fi utilizat setul de date Iris sau alte seturi de date cu probleme de clasificare.

# Importăm bibliotecile necesare  
from sklearn.svm import SVC  
from sklearn.datasets import make\_classification  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score  
  
  
# Generăm un set de date sintetic pentru scopuri de demonstrație  
X, y = make\_classification(n\_samples=1000, n\_features=20, n\_classes=2, random\_state=42)  
  
# Împărțim setul de date în set de antrenare și set de testare  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Inițializăm clasificatorul Support Vector Machine (SVM)  
svm\_clf = SVC(kernel='linear', random\_state=42)  
  
# Antrenăm clasificatorul SVM pe setul de antrenare  
svm\_clf.fit(X\_train, y\_train)  
  
# Facem predicții pe setul de testare  
y\_pred = svm\_clf.predict(X\_test)  
  
# Calculăm acuratețea predicțiilor  
accuracy = accuracy\_score(y\_test, y\_pred)  
print("Acuratețea clasificatorului Support Vector Machine (SVM):", accuracy)  
  
# Calculăm alte metrici de evaluare  
print("\nMatricea de Confuzie:")  
print(confusion\_matrix(y\_test, y\_pred))  
  
print("\nRaportul de Clasificare:")  
print(classification\_report(y\_test, y\_pred))

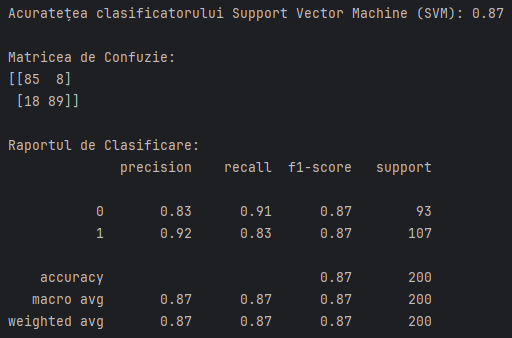


Figura 2 – Rezultatele metodei Support Vector Machines (SVM)

**Bazate pe Reguli de Decizie - Arbori de Decizie:**

Concept: Arborii de decizie utilizează o structură arborescentă pentru a reprezenta regulile de decizie în funcție de caracteristicile datelor de intrare.

Pași de bază:

1. Selectarea atributului cel mai informativ pentru a separa datele în subgrupuri.
2. Divizarea datelor în funcție de valoarea atributului selectat.
3. Repetarea procesului pentru fiecare subgrup până când toate subgrupurile sunt pure sau un alt criteriu de oprire este îndeplinit.

Set de date: Pot fi utilizate seturi de date precum Iris sau alte seturi de date pentru probleme de clasificare.

# Importăm bibliotecile necesare  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.datasets import make\_classification  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score  
  
  
# Generăm un set de date sintetic pentru scopuri de demonstrație  
X, y = make\_classification(n\_samples=1000, n\_features=20, n\_classes=2, random\_state=42)  
  
# Împărțim setul de date în set de antrenare și set de testare  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Inițializăm clasificatorul Arbori de Decizie  
decision\_tree\_clf = DecisionTreeClassifier(random\_state=42)  
  
# Antrenăm clasificatorul Arbori de Decizie pe setul de antrenare  
decision\_tree\_clf.fit(X\_train, y\_train)  
  
# Facem predicții pe setul de testare  
y\_pred = decision\_tree\_clf.predict(X\_test)  
  
# Calculăm acuratețea predicțiilor  
accuracy = accuracy\_score(y\_test, y\_pred)  
print("Acuratețea clasificatorului Arbori de Decizie:", accuracy)  
  
# Calculăm alte metrici de evaluare  
print("\nMatricea de Confuzie:")  
print(confusion\_matrix(y\_test, y\_pred))  
  
print("\nRaportul de Clasificare:")  
print(classification\_report(y\_test, y\_pred))

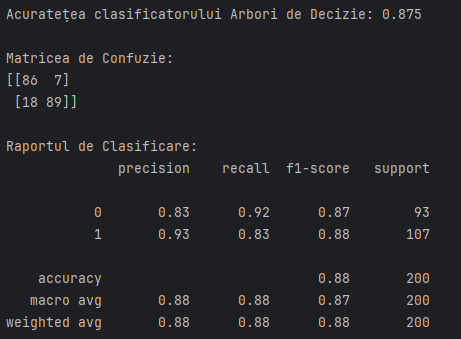


Figura 3 – Rezultatele metodei Arbori de Decizie

**Bazate pe Arbori - Random Forest:**

Concept: Random Forest este o metodă de ansamblu care folosește mai mulți arbori de decizie și combină rezultatele acestora pentru a obține o clasificare mai robustă și mai generalizabilă.

Pași de bază:

1. Construirea unui set de arbori de decizie pe baza unor subseturi aleatorii ale setului de date de antrenare.
2. Clasificarea fiecărui punct de date de intrare prin votul majorității arborilor.

Set de date: Poate fi utilizat setul de date Iris sau alte seturi de date cu probleme de clasificare.

# Importăm bibliotecile necesare  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.datasets import make\_classification  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score  
  
  
# Generăm un set de date sintetic pentru scopuri de demonstrație  
X, y = make\_classification(n\_samples=1000, n\_features=20, n\_classes=2, random\_state=42)  
  
# Împărțim setul de date în set de antrenare și set de testare  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Inițializăm clasificatorul Random Forest  
random\_forest\_clf = RandomForestClassifier(n\_estimators=100, random\_state=42)  
  
# Antrenăm clasificatorul Random Forest pe setul de antrenare  
random\_forest\_clf.fit(X\_train, y\_train)  
  
# Facem predicții pe setul de testare  
y\_pred = random\_forest\_clf.predict(X\_test)  
  
# Calculăm acuratețea predicțiilor  
accuracy = accuracy\_score(y\_test, y\_pred)  
print("Acuratețea clasificatorului Random Forest:", accuracy)  
  
# Calculăm alte metrici de evaluare  
print("\nMatricea de Confuzie:")  
print(confusion\_matrix(y\_test, y\_pred))  
  
print("\nRaportul de Clasificare:")  
print(classification\_report(y\_test, y\_pred))

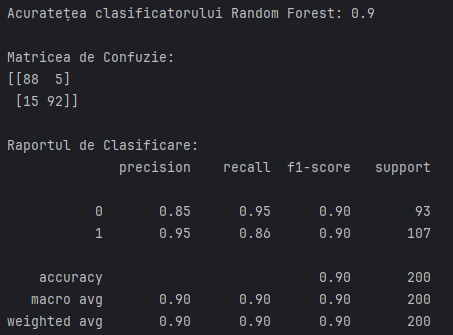


Figura 4 – Rezultatele metodei Random Forest

**Meta-metode – AdaBoost:**

Concept: AdaBoost este o meta-metodă de învățare care combină mai mulți clasificatori slabi într-un clasificator puternic prin reponderarea datelor în funcție de performanța clasificatorilor anterioare.

Pași de bază:

1. Inițializarea ponderilor pentru fiecare punct de date.
2. Antrenarea unui clasificator slab pe setul de date ponderat.
3. Actualizarea ponderilor pentru punctele de date clasificate incorect.
4. Repetarea procesului pentru un număr predeterminat de iterații sau până când clasificatorul are performanța dorită.

Set de date: Poate fi utilizat setul de date Iris sau alte seturi de date cu probleme de clasificare.

# Importăm bibliotecile necesare  
from sklearn.ensemble import AdaBoostClassifier  
from sklearn.datasets import make\_classification  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score  
  
  
# Generăm un set de date sintetic pentru scopuri de demonstrație  
X, y = make\_classification(n\_samples=1000, n\_features=20, n\_classes=2, random\_state=42)  
  
# Împărțim setul de date în set de antrenare și set de testare  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Inițializăm clasificatorul AdaBoost  
adaboost\_clf = AdaBoostClassifier(n\_estimators=50, random\_state=42)  
  
# Antrenăm clasificatorul AdaBoost pe setul de antrenare  
adaboost\_clf.fit(X\_train, y\_train)  
  
# Facem predicții pe setul de testare  
y\_pred = adaboost\_clf.predict(X\_test)  
  
# Calculăm acuratețea predicțiilor  
accuracy = accuracy\_score(y\_test, y\_pred)  
print("Acuratețea clasificatorului AdaBoost:", accuracy)  
  
# Calculăm alte metrici de evaluare  
print("\nMatricea de Confuzie:")  
print(confusion\_matrix(y\_test, y\_pred))  
  
print("\nRaportul de Clasificare:")  
print(classification\_report(y\_test, y\_pred))

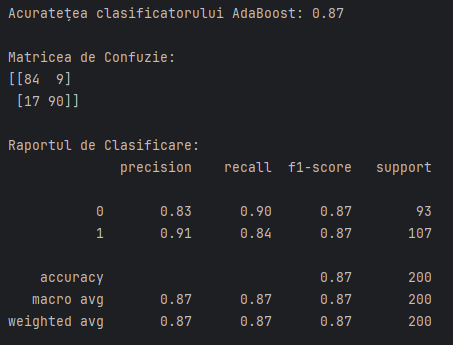


Figura 5 – Rezultatele metodei AdaBoost

**Concluzie:**

După evaluarea și implementarea algoritmilor de clasificare precum Naiv Bayes, Support Vector Machines (SVM), Arbori de Decizie și Random Forest pe date sintetice, rezultatele evidențiază caracteristicile și performanțele distinctive ale fiecărui algoritm în ceea ce privește clasificarea datelor.

În primul rând, algoritmul Naiv Bayes a prezentat o abordare simplă și rapidă pentru clasificare, presupunând independența atributelor. Acesta a obținut o acuratețe moderată, dar trebuie menționat că poate avea performanțe mai scăzute în situații în care ipoteza de independență nu este respectată.

În al doilea rând, Support Vector Machines (SVM) au demonstrat capacitatea lor de a construi hiperplane de separare complexe în spațiul de caracteristici. Cu toate acestea, parametrizarea adecvată a kernelului poate fi crucială pentru performanța optimă a SVM-ului.

Arborii de decizie, pe de altă parte, oferă interpretabilitate și ușurință în înțelegerea procesului de luare a deciziilor. Cu toate acestea, există riscul de overfitting și necesitatea de a optimiza parametrii pentru a evita acest lucru.

În fine, Random Forests, o tehnică de ansamblu bazată pe arbori de decizie, au prezentat o performanță robustă în clasificarea datelor, datorită capacității lor de a combina multiple modele slabe într-un clasificator puternic. Cu toate acestea, creșterea numărului de arbori din ansamblu poate duce la creșterea timpului de antrenare.

În concluzie, evaluarea performanței algoritmilor de clasificare a evidențiat necesitatea de a alege și de a configura algoritmii în funcție de natura specifică a datelor și de cerințele problemei. Abordarea potrivită poate varia în funcție de natura datelor, dimensiunea setului de date și cerințele de interpretare și performanță. Este important să se țină cont de aceste aspecte în dezvoltarea și implementarea modelelor de clasificare în practică.

* Precision: Rata de exemple pozitive corect identificate din totalul exemplelor identificate ca pozitive.
* Recall: Rata de exemple pozitive corect identificate din totalul exemplelor pozitive din setul de date.
* F1-score: Media armonică a preciziei și recall-ului. Oferă o măsură echilibrată între cele două.
* Suport: Numărul total de exemple din setul de date pentru fiecare clasă.