

Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Ingineria Software și Automatică

**RAPORT**

**Lucrarea de laborator nr.2**

**la Inteligența Artificială**

*Tema: Tehnici de clasificare supervizată*

Grupa academică: TI-211  
A efectuat: Popa Cătălin

**A verificat: Mariana Rusu**

Chișinău 2024

**Problema propusă:**

Alegeți spre implementare (în limbajul Python) câte o metodă din fiecare categorie de mai jos. Explicați conceptul și pașii de bază ai fiecărui algoritm. Alegeți sau creați un set de date conform necesităților fiecărui algoritm elaborat.

Tehnici de clasificare supervizată:

* Bazate pe criteriul Bayes;
* Bazate pe funcții;
* Bazate pe reguli de decizie;
* Bazate pe arbori;
* Meta-metode;

**Tehnică de clasificare supervizată bazată pe criteriul Bayes**

Tehnica de clasificare supervizată bazată pe criteriul Bayes, cunoscută și sub numele de clasificare Bayes, se bazează pe teorema lui Bayes pentru a efectua clasificarea datelor în diferite categorii. Iată o explicație pas cu pas a modului în care funcționează:Iată cum funcționează pas cu pas:

1. Colectarea datelor etichetate: Fiecare exemplu constă dintr-o serie de caracteristici (atribute) și o etichetă de clasă.
2. Calculul probabilităților a priori: Pentru fiecare clasă posibilă, se calculează probabilitatea a priori a acelei clase în baza frecvenței cu care apare în setul de date. Această probabilitate reprezintă cunoștințele inițiale sau presupunerile noastre despre distribuția claselor în setul de date.
3. Calculul probabilităților condiționate: Pentru fiecare combinație de valori ale caracteristicilor, se calculează probabilitatea condiționată a observa acele valori date fiind clasa. Aceste probabilități se pot calcula utilizând frecvențele relative ale caracteristicilor în fiecare clasă.
4. Aplicarea teoremei lui Bayes: Utilizând probabilitățile a priori și probabilitățile condiționate calculate anterior, se aplică teorema lui Bayes pentru a calcula probabilitatea posteroară a fiecărei clase pentru un nou exemplu. Probabilitatea posteroară reprezintă probabilitatea că un exemplu aparține unei anumite clase date caracteristicile observate.
5. Alegerea clasei prezise:După ce s-au calculat probabilitățile posteroare pentru fiecare clasă posibilă, clasa cu cea mai mare probabilitate posteroară este aleasă ca predicție pentru noul

exemplu.

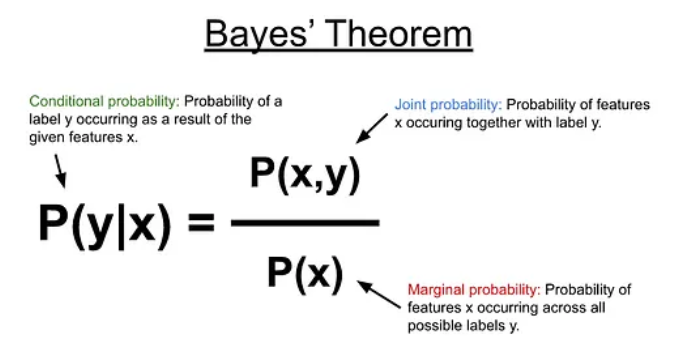


Figura 1 – Reprezentarea teoremei Bayes

În cadrul laboratorului se va antrena un model pentru a detecta dacă pacientul are cancer sau nu. Mai întâi am important setul de date și librăriiile necesare.

import numpy as np  
import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.datasets import load\_breast\_cancer  
breast\_cancer = load\_breast\_cancer()

În următorul dat transformăm setul de date în dataframe pentru a putea manipula cu el.

df = pd.DataFrame(  
 np.c\_[breast\_cancer.data, breast\_cancer.target],  
 columns = [list(breast\_cancer.feature\_names)+ ['target']]  
 )  
print(df)

Apoi creem variabilele X și y, care sunt definite pentru a stoca caracteristicile și, respectiv, etichetele din DataFrame-ul df.

X = df.iloc[:, 0:-1]  
y = df.iloc[:,-1]

În continuare împățim datele în seturi de antrenament și de validare.

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size = 0.2, random\_state = 999)

**train\_test\_split(X, y, test\_size=0.2, random\_state=999):** Această funcție primește ca parametri caracteristicile (X) și etichetele (y) și le împarte în două seturi: un set de antrenament și un set de validare. Parametrul test\_size=0.2 specifică proporția de date care vor fi alocate pentru setul de validare, în acest caz 20%. Parametrul random\_state=999 este utilizat pentru a asigura reproductibilitatea împărțirii datelor. Alegerea acestuia asigură că împărțirea datelor va fi aceeași de fiecare dată când codul este rulat, ceea ce este util pentru a obține rezultate consistente în timpul dezvoltării și testării.

Acum inițializăm un obiect de clasificator Bayes Gaussian din biblioteca scikit-learn. Acesta este un clasificator simplu bazat pe teorema lui Bayes, care presupune că caracteristicile sunt independente între ele.

clf = GaussianNB()

Apoi antrenăm clasificatorul utilizând metoda fit() și datele de antrenament. Acest proces învață modelul să facă predicții pe baza caracteristicilor de intrare și etichetelor corespunzătoare.

clf.fit(X\_train, y\_train)

După calculăm acuratețea modelului pe setul de date de validare(x\_val, y\_val). Metoda score() calculează acuratețea modelului prin compararea predicțiilor făcute de model cu etichetele reale din setul de validare și returnează proporția de predicții corecte.

clf.score(X\_val, y\_val)

Acuratețea modelului.

**Răspuns: 0.9210526315789473**

Acum putem verifica pe un acaz anumit și realiza o predicție.

Afișăm datele din al 100 rând și le preluăm pentru a putea lucra cu ele.

pd.set\_option('display.max\_columns', None)  
print(df[99:100])  
  
patient1 = [14.42,19.77,94.48,642.5,0.09752,0.1141,0.09388,0.05839,0.1879,0.0639,0.2895,1.851,2.376,26.85,0.008005,0.02895,0.03321,0.01424,0.01462,0.004452,16.33,30.86,109.5,826.4,0.1431,0.3026,0.3194,0.1565,0.2718,0.09353]

După transformăm datele într-un array și aplicăm predicția pe pacient pentru a vedea dacă are cancer sau nu. Dacă predicția este 0, atunci are cancer, în alt caz nu are.

pd.set\_option('display.max\_columns', None)  
print(df[99:100])  
  
patient1 = [14.42,19.77,94.48,642.5,0.09752,0.1141,0.09388,0.05839,0.1879,0.0639,0.2895,1.851,2.376,26.85,0.008005,0.02895,0.03321,0.01424,0.01462,0.004452,16.33,30.86,109.5,826.4,0.1431,0.3026,0.3194,0.1565,0.2718,0.09353]  
  
patient1 = np.array([patient1])  
#print(patient1)  
  
pred = clf.predict(patient1)  
  
if pred[0] == 0:  
 print("Pacientul are cancer")  
else:  
 print("Pacientul nu are cancer")

**Răspuns**



**Tehnică de clasificare supervizată bazată pe meta-metode ( Random Forest Classifier )**

Una dintre tehnicile de clasificare supervizată bazată pe meta-metode Clasificator Random Forest. Acesta un algoritm de învățare automată supravegheat care se bazează pe învățarea ansamblului. În acest exemplu, am construit două modele Random Forest Classifier pentru a prezice siguranța mașinii, unul cu 10 arbori de decizie și altul cu 100 de arbori de decizie. Precizia așteptată crește odată cu numărul de arbori de decizie din model. Am demonstrat procesul de selecție a caracteristicilor folosind modelul Random Forest pentru a găsi doar caracteristicile importante, pentru a reconstrui modelul folosind aceste caracteristici și pentru a vedea efectul său asupra acurateței.

Random Forest este un clasificator care conține un număr de arbori de decizie pe diferite subansamble ale setului de date dat și ia media pentru a îmbunătăți acuratețea predictivă a acelui set de date.

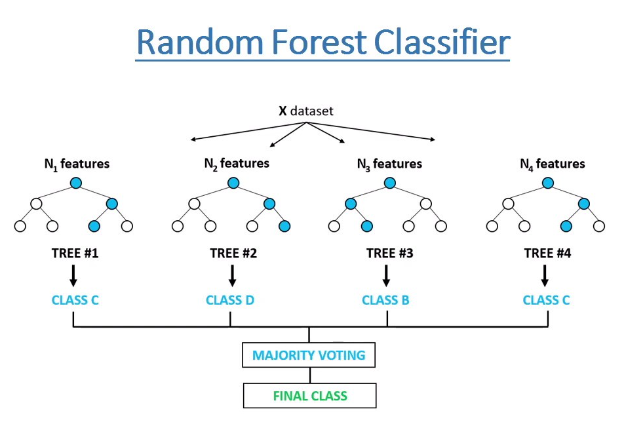


Figura 2 – Random Forest Classifier

Mai întâi citim datele.

# Citeste datele  
df = pd.read\_csv("car\_evaluation.csv", header=None)  
col\_names = ['buying', 'maint', 'doors', 'persons', 'lug\_boot', 'safety', 'class']  
df.columns = col\_names

Convertim caracteristicele categorice în variabile dummy.

df = pd.get\_dummies(df, columns=['buying', 'maint', 'doors', 'persons', 'lug\_boot', 'safety'])

Definim variabilile x și y.

X = df.drop(['class'], axis=1)  
Y = df['class']

Împărțim datele în setul de antrenamente și de testaer.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.33, random\_state=42)

Antrenăm și inițializăm modelul RandomForestClassifier.

rfc = RandomForestClassifier(random\_state=0)  
rfc\_100 = RandomForestClassifier(n\_estimators=100,random\_state=0)  
rfc.fit(X\_train, y\_train)  
rfc\_100.fit(X\_train, y\_train)

Realizăm predicția pe setul de testare.

y\_pred = rfc.predict(X\_test)  
y\_pred\_100 = rfc.predict(X\_test)

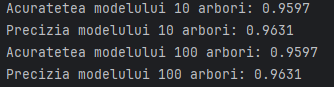
Calculăm precizia

# Calculare precizie  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
precision2 = precision\_score(y\_test, y\_pred\_100, average='weighted')

Verificăm acuuratețea.

print('Acuratetea modelului 10 arbori: {0:0.4f}'.format(accuracy))  
print('Precizia modelului 10 arbori: {0:0.4f}'.format(precision))  
print('Acuratetea modelului 100 arbori: {0:0.4f}'.format(accuracy2))  
print('Precizia modelului 100 arbori: {0:0.4f}'.format(precision2))

**Răspuns**



# Tehnică de clasificare supervizată bazată pe arbori (Decision Tree)

Un algoritm de arbore de decizie este unul dintre cei mai populari algoritmi de învățare automată. Utilizează o structură de tip arbore și posibilele lor combinații pentru a rezolva o anumită problemă. Aparține clasei de algoritmi de învățare supravegheată unde poate fi utilizat atât în ​​scopuri de clasificare, cât și de regresie. Un arbore de decizie este o structură care include un nod rădăcină, ramuri și noduri frunză. Fiecare nod intern denotă un test pe un atribut, fiecare ramură denotă rezultatul unui test și fiecare nod frunză deține o etichetă de clasă. Nodul cel mai de sus din arbore este nodul rădăcină.

Încărcăm datele.

df = pd.read\_csv("car\_evaluation2.csv", header=None)  
col\_names = ['buying', 'maint', 'doors', 'persons', 'lug\_boot', 'safety', 'class']  
df.columns = col\_names

Convertim variabilele categoriale în numerice folosind LabelEncoder.

labelencoder = LabelEncoder()  
for col in df.columns:  
 df[col] = labelencoder.fit\_transform(df[col])

Împărțim datele în set de antrenamente și de testare.

X = df.drop(['class'], axis=1)  
y = df['class']  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

Inițializăm algoritmul DecisionTreeClassifier

clf\_gini = DecisionTreeClassifier(criterion='gini', max\_depth=3, random\_state=0)  
clf\_gini.fit(X\_train, y\_train)

Realizăm predicția și acurateția.

y\_pred\_train = clf\_gini.predict(X\_train)  
y\_pred\_test = clf\_gini.predict(X\_test)  
accuracy\_train = accuracy\_score(y\_train, y\_pred\_train)  
accuracy\_test = accuracy\_score(y\_test, y\_pred\_test)

print('Acuratetea cu criteriul gini index pentru setul de antrenament: {0:0.4f}'.format(accuracy\_train))  
print('Acuratetea cu criteriul gini index pentru setul de testare: {0:0.4f}'.format(accuracy\_test))



**Tehnică de clasificare supervizată bazată pe funcții**

Pentru realizarea acestei saricini voi folosi SVM – Support Vector Machines, care este un algoritm de învățare automată supravegheată utilizat pentru sarcini de clasificare și regresie. SVM-urile funcționează prin găsirea hiperplanului optim care separă punctele de date în clase diferite. În spațiul bidimensional, un hiperplan este pur și simplu o linie care separă punctele de date în două clase. Acesta poate fi utilizat pentru a face predicții cu privire la noi puncte de date, evaluând pe ce parte a hiperplanului se încadrează acestea.

Încrăm setul de date predefinit iris.

iris = datasets.load\_iris()  
X = iris.data  
y = iris.target

Împărțim în set de antrenamente și set de testare.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

Standartizăm datele.

sc = StandardScaler()  
X\_train\_std = sc.fit\_transform(X\_train)  
X\_test\_std = sc.transform(X\_test)

Inițializăm clasificarea SVM și antrenăm clasificatorul.

svm = SVC(kernel='linear', C=1.0, random\_state=42)  
  
svm.fit(X\_train\_std, y\_train)

Prezicem clasele pentru setul de testare.

y\_pred = svm.predict(X\_test\_std)

Calculăm acuratețea.

accuracy = accuracy\_score(y\_test, y\_pred)  
print("Acuratețea clasificatorului SVM: {:.2f}".format(accuracy))

precision = precision\_score(y\_test, y\_pred, average='weighted')  
print("Precizia clasificatorului SVM: {:.2f}".format(precision))

**Răspuns**



**Concluzie:**

În acest raport, am implementat și comparat patru metode de clasificare supervizată în Python: Clasificare bazată pe criteriul Bayes: Am utilizat algoritmul Gaussian Naive Bayes pentru detectarea cancerului la pacienți, obținând o acuratețe de 92.11%. Meta-metode: Am implementat clasificatorul Random Forest, antrenând două modele (cu 10 și 100 de arbori), obținând acurateți de 96% și, respectiv, 95%. Clasificare bazată pe arbori: Am folosit algoritmul Decision Tree pe date despre evaluarea mașinilor, obținând o acuratețe de 76,5%. Clasificare bazată pe funcții: Am implementat clasificatorul SVM (Support Vector Machines) pe date despre florile iris, obținând o acuratețe de 98%. Fiecare metodă are avantaje și dezavantaje, iar selecția depinde de natura datelor și a problemei. Este esențial să alegem algoritmul potrivit pentru obiectivele noastre de analiză.