INGINERIA SISTEMELOR CU INTELIGENȚĂ ARTIFICALĂ
Student: Iulia Cristina STANCIU
Grupa: 421A

SPECT Heart

Clasificarea tomografiilor pacienților în normale și anormale cu ajutorul mașinii cu vectori suport

1. Scopul proiectului

Scopul acestui proiect este clasificarea cu acuratețe cât mai mare a analizelor cardiace de tip SPECT ale pacienților două categorii: normale și anormale.

2. Baza de date și caracteristicile acesteia

În acest proiect a fost folosită baza de date "SPECT Heart", disponibilă pe Machine Learning Repository (UCI Machine Learning Repository: SPECT Heart Data Set)

Tipul de problemă: **Clasificare**
Mărimea bazei de date:

→ Numărul de instanțe: **267**
→ Numărul de atribute: **22**

Alte caracteristici:

→ Baza de date nu are valori lipsă.

Observație! A fost tratat și cazul în care existau valori lipsă, marcate cu "?". În acest caz, întreaga instanță era eliminată din baza de date.

→ Atributele și etichetele reprezintă valori binare.
→ Setul de date este împărțit în date de testare și antrenare

3. Fișiere conținute în baza de date

Baza de date "SPECT Heart" conține trei fișiere:

- "SPECT.names" – Fișierul conține informații referitoare la setul de date.
- "SPECT.train" – Fișier .csv ce conține datele de antrenare și etichetele acestora
- "SPECT.test" – Fișier .csv ce conține datele de testare și etichetele acestora

4. Raportul instanțelor – antrenare : testare

Raportul împărțirii datelor in antrenare-testare este aproximativ 30:70.
Setul de date de antrenare conține 80 de instanțe. Setul de date de testare conține 187 de instanțe.

INGINERIA SISTEMELOR CU INTELIGENȚĂ ARTIFICALĂ
Student: Iulia Cristina STANCIU
Grupa: 421A

5. Librăriile folosite

1) Pandas

Librăria este folosită pentru deschiderea fișierelor cu date de antrenare și testare.

2) Numpy

Librărie folosită pentru crearea si separare matricelor de date în atribute și etichete pentru setul de date de antrenare și pentru cel de testare.

3) Scikit-learn

Librărie folosită pentru aplicarea algoritmului SVM și pentru calcularea acurateței.

4) Math

6. Sistemul de clasificare

Pentru rezolvarea problemei se folosește algoritmul SVM (Support Vector Machines) pentru clasificare (SVC).

7. Variația parametrilor și rezultatele obținute

A fost folosit algoritmul SVM cu nucleu liniar și parametrul "Cost" variabil. Au fost obținute următoarele rezultate:

Parametrul "Cost"	**Acuratețe**	**Acuratețe echilibrată**
2^{-5}	0.5508	0.5
2^{-3}	0.6684	0.6430
2^{-1}	0.6738	0.6534
2	0.6845	0.6664
2^{3}	0.5775	0.555
2^{5}	0.5561	0.5345
2^{7}	0.5561	0.5345

Am încercat și algoritmul SVM cu kernel polinomial. Rezultatele cele mai bune au fost obținute pentru parametrul "Cost" C=1 și un grad al polinomului degree=2:

Acuratețe	**Acuratețe echilibrată**
0.6898	0.6701

8. Metrica folosită pentru măsurarea performanței

1) Acuratețe (accuracy_score) – $Accuracy = \frac{TP+TN}{TP+FP+TN+FP}$
2) Acuratețe echilibrată (balance_accuracy_score) - $Balanced\ Accuracy = \frac{1}{2}(\frac{TP}{TP+FN} + \frac{TN}{TN+FP})$

Unde TP = true positive, TN = true negative, FP = false positive, FN = false negative

INGINERIA SISTEMELOR CU INTELIGENȚĂ ARTIFICALĂ
Student: Iulia Cristina STANCIU
Grupa: 421A

9. Cele mai bune rezultate obținute

Cea mai bună acuratețe pentru SVC cu nucleu liniar a fost 0.6845, obținută pentru parametrul “Cost” egal cu 2. (Acuratețea echilibrată maximă pentru SVC cu nucleu liniar este 0.6664 și a fost obținută pentru același “Cost”.) Cea mai bună acuratețe pentru SVC cu kernel polinomial de gradul 2 este 0.6898. (Acuratețea echilibrată este 0.6701.)

10. Adăugiri făcute codului pentru a crește performanța sistemului

1) Funcție pentru ștergerea instanțelor cu atribute lipsă, marcate cu “?”

```
#functia verifica daca exista valori lipsa marcate cu "?" si sterge instanta cu valori lipsa
data_labels = np.array([row for row in data_labels if '?' not in row])
```

2) Funcție pentru ștergerea atributelor corelate total cu alte atribute

```
""" functie verificare corelatie intre atribute """
def delete_corelated_data(d_train, d_test):
    for col1 in range(np.shape(d_train)[1]-1, 0, -1):
        for col2 in range(col1-1, -1, -1):
            check_eq = np.equal(d_train[:, col1], d_train[:, col2])
            check_not_eq = np.not_equal(d_train[:, col1], d_train[:, col2])

            if(np.all(check_eq) or np.all(check_not_eq)):
                d_train = np.delete(d_train, col2, 1)
                d_test = np.delete(d_test, col2, 1)
                break
    # functia returneaza datele de antrenare si cele de testare dupa ce se scapa de atributele nefolositoare
    return d_train, d_test
```

3) Analiza complexității datelor prin calcularea predicției prin metoda celor mai apropiați vecini

```
for i in range(1, (int)(math.sqrt(nr_attributes_train)) ):
    knn = KNeighborsClassifier(n_neighbors = i, metric='euclidean')
    knn.fit(data_train,labels_train)
    predictions = knn.predict(data_test)

    accuracy_knn.append(accuracy_score(labels_test, predictions))
    balanced_accuracy_knn.append(balanced_accuracy_score(labels_test, predictions))

print('Acuratete maxima pentru knn =', max(accuracy_knn))
print('Acuratetea echilibrata maxima pentru knn =', max(balanced_accuracy_knn))
print('Acestea se obtin pentru',  accuracy_knn.index(max(accuracy_knn))+1,'vecini.')
print()
```

Numarul k de vecini	Acuratețe	Acuratețe echilibrată
1	0.6631	0.6612
2	0.6898	0.6581
3	0.6738	0.6577