Department of Computer Science
Technical University of Cluj-Napoca

Intelligent Systems

Laboratory activity 2021-2022

Machine Learning

Project title: Assignment 2

Name: Izabella Bartalus
Group: 30231
Email: bartalusiza08@gmail.com

Asst. Prof. Eng. Roxana Ramona Szomiu
Roxana.Szomiu@cs.utcluj.ro

Contents

Chapter 1

Project description

1.1 Introduction

Proiectul meu porneste de la un set de date care estimeaza nivelurile de obezitate la indivizi din tarile Mexic, Peru si Columbia, pe baza obiceiurilor alimentare si a conditiei lor fizice. Coloanele prezentate sunt: Gender, Age, Height, Weight, family history with overweight, FAVC, FCVC, NCP, CAEC, SMOKE, CH2O, SCC, FAF, TUE, CALC, MTRANS, NObeyesdad.

Problema aleasa pentru proiect se refera la clasificarea setului de date, folosind algoritmii Decision Tree Classifier si SVM Classifier folosind SVC. Cea de-a doua parte a problemei reprezinta aplicarea asupra setului de date a urmatoarelor metode ansamblu, specifice clasificarii: Bagging Classifier si Boosting Classifier. Pentru algoritmul de Bagging am folosit doua metode diferite: Random Forest si ExtraTrees, iar pentru algoritmul de Boosting am folosit metodele: AdaBoost si Gradient Tree.

Se va incarca setul de date in Jupyter Notebook si se va prelucra astfel incat sa folosim toate metodele mentionate mai sus pe dataset-ul nostru. De asemenea, vom calcula o serie de metrici pentru clasificarea datelor pentru a putea compara performanta algoritmilor.

1.2 The main goal of the project

In cadrul acestui assignment voi prezenta cum si in ce maniera diferiti algoritmi de Machine Learning reusesc sa obtina clasificarii sau predictii valide asupra unui set de date in ceea ce priveste nivelul de obezitate al unor persoane.

Se vor urmari de asemenea si anumite metrici de clasificare precum: matricea de confuzie, acuratete, precizie, recall si f1. Metricile alese reprezinta:

- Matricea de confuzie C este astfel incat Ci,j este numarul de observatii despre care se stie ca sunt in grupul i si sunt prezise in grupul j;
- Acuratetea calculeaza fractia de sample-uri prezise corect;
- Precizia calculeaza fractia de sample-uri prezise pozitive si care de fapt sunt pozitive;
- Recall calculeaza fractia de sample-uri pozitive care sunt prezise corect;
- F1 este media armonica a recall-ului si a preciziei.

1.3 Dataset

Sursa dataset-ului: `https://archive.ics.uci.edu/ml/datasets/Estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition+#`

Initial dataset-ul continea 2112 sample-uri iar coloana NObeyesdad (care spune nivelul de obezitate) avea mai mult decat 2 valori posibile, iar pentru a putea ajunge la un dataset care sa aiba o coloana care sa reprezinte o asa-zisa variabila discreta am redus dataset-ul initial la 499 samples iar coloana NObeyesdad a ramas doar cu valorile Normal Weight si Overweight Level I.

In [5]:
```
datasets
```
Out[5]:

ıt	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NOb
ı0	64.000000	yes	no	2	3	Sometimes	no	2.000000	no	0.000000	1	no	Public_Transportation	Norma
ı0	56.000000	yes	no	3	3	Sometimes	yes	3.000000	yes	3.000000	0	Sometimes	Public_Transportation	Norma
ı0	77.000000	yes	no	2	3	Sometimes	no	2.000000	no	2.000000	1	Frequently	Public_Transportation	Norma
ı0	87.000000	no	no	3	3	Sometimes	no	2.000000	no	2.000000	0	Frequently	Walking	Overweigh
ı0	89.800000	no	no	2	1	Sometimes	no	2.000000	no	0.000000	0	Sometimes	Public_Transportation	Overweigh
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
ı0	66.000000	no	yes	3	3	Sometimes	no	2.000000	no	0.000000	0	Sometimes	Public_Transportation	Norma
ı0	60.000000	yes	yes	3	1	Always	no	1.000000	yes	0.000000	0	no	Motorbike	Norma
ı0	53.000000	yes	yes	2	3	Sometimes	no	2.000000	no	0.000000	2	Sometimes	Public_Transportation	Norma
ı0	45.000000	no	no	2	3	Sometimes	no	2.000000	no	1.000000	1	Sometimes	Public_Transportation	Norma
ı6	104.572712	yes	yes	3	3	Sometimes	no	1.152736	no	0.319156	1	Sometimes	Public_Transportation	Overweigh

In continuare, se vor prezenta operatii asupra setului de date care au fost aplicate pentru toti algoritmii mentionati mai sus.

Primele 5 valori din setul de date:

In [6]:
```
#afisez primele 5
datasets.head()
```
Out[6]:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRAN
0	Female	21.0	1.62	64.0	yes	no	2	3	Sometimes	no	2.0	no	0.0	1	no	Public_Transportati
1	Female	21.0	1.52	56.0	yes	no	3	3	Sometimes	yes	3.0	yes	3.0	0	Sometimes	Public_Transportati
2	Male	23.0	1.80	77.0	yes	no	2	3	Sometimes	no	2.0	no	2.0	1	Frequently	Public_Transportati
3	Male	27.0	1.80	87.0	no	no	3	3	Sometimes	no	2.0	no	2.0	0	Frequently	Walki
4	Male	22.0	1.78	89.8	no	no	2	1	Sometimes	no	2.0	no	0.0	0	Sometimes	Public_Transportati

Dimensiunea setului de date si tipul coloanelor:

In [4]:
```
datasets.shape
```
Out[4]:
```
(499, 17)
```

In [286]:
```
datasets.dtypes
```
Out[286]:
```
Gender                             object
Age                               float64
Height                            float64
Weight                            float64
family_history_with_overweight     object
FAVC                               object
FCVC                                int64
NCP                                 int64
CAEC                               object
SMOKE                              object
CH2O                              float64
SCC                                object
FAF                               float64
TUE                                 int64
CALC                               object
MTRANS                             object
NObeyesdad                         object
dtype: object
```

Codificarea datelor:

In [8]:
```
le_ObesityLevel = LabelEncoder()
le_ObesityLevel.fit(datasets['NObeyesdad'])
data_encoded_ObL = le_ObesityLevel.transform(datasets['NObeyesdad'])
data_encoded_ObL
```

Out[8]:
```
array([0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1,
       1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1])
```

In [288]:
```
le_ObesityLevel.classes_
```

Out[288]:
```
array(['Normal_Weight', 'Overweight_Level_I'], dtype=object)
```

In [9]:
```
datasets_1 = datasets.apply(le_ObesityLevel.fit_transform) # for all the attributes
datasets_1
```

Out[9]:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad
0	0	7	14	29	1	0	1	1	2	0	2	0	0	1	3	3	0
1	0	7	4	19	1	0	2	1	2	1	3	1	4	0	2	3	0
2	1	9	33	46	1	0	1	1	2	0	2	0	3	1	1	3	0
3	1	14	33	57	0	0	2	1	2	0	2	0	3	0	1	4	1
4	1	8	31	60	0	0	1	0	2	0	2	0	0	0	2	3	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
494	0	4	19	32	0	1	2	1	2	0	2	0	0	0	2	3	0
495	1	5	33	25	1	1	2	0	0	0	0	1	0	0	3	2	0
496	1	4	25	13	1	1	1	1	2	0	2	0	0	2	2	3	0
497	1	6	8	5	0	0	1	1	2	0	2	0	2	1	2	3	0
498	0	12	21	76	1	1	2	1	2	0	1	0	1	1	2	3	1

499 rows × 17 columns

Impartitea dataset-ului in feature names si target names:

In [290]:
```
X = datasets_1[datasets.columns.drop('NObeyesdad')]
Y = datasets_1['NObeyesdad']
```

Impartirea datelor in test si train:

In [293]:
```
#Impart dataset-ul in Training set si Test set
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.25)
```

Scalarea datelor:

In [297]:
```
#Feature Scaling
stdScaler = StandardScaler() #Standardize features by removing the mean and scaling to unit variance
X_Train = stdScaler.fit_transform(X_Train)
X_Test = stdScaler.transform(X_Test)
```

Chapter 2

Implementation details

2.1 Algorithm 1

Decision Tree este o tehnica de machine learning care poate fi utilizata atat pentru probleme de clasificare, cat si pentru probleme de regresie, dar este de preferat pentru rezolvarea problemelor de clasificare. Este un clasificator structurat in arbore, in care nodurile interne reprezinta caracteristicile unui set de date, ramurile reprezinta regulile de decizie si fiecare nod frunza reprezinta rezultatul.

Avantajele arborelui decizional:

- Este simplu de ineeles, deoarece urmeaza acelasi proces pe care il urmează un om nn timp ce ia orice decizie în viata reala.

- Poate fi foarte util pentru rezolvarea problemelor legate de decizii.

- Ajuta sa te gandesti la toate rezultatele posibile pentru o problema.

- Exista mai putine cerinte de curatare a datelor in comparatie cu alti algoritmi.

Dezavantajele arborelui decizional:

- Arborele de decizie contine o multime de straturi, ceea ce il face complex.

- Poate avea o problema de supra-adaptare, care poate fi rezolvata folosind algoritmul Random Forest.

- Pentru mai multe etichete de clasa, complexitatea de calcul a arborelui de decizie poate creste.

Decision Tree folosind criteriul entropy:

```
In [34]: dt1 = tree.DecisionTreeClassifier(criterion="entropy")
         dt1

Out[34]: DecisionTreeClassifier(criterion='entropy')

In [35]: dt1.fit(X_Train, Y_Train)

Out[35]: DecisionTreeClassifier(criterion='entropy')

In [36]: pred = dt1.predict(X_Test)
         pred

Out[36]: array([1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
                1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
                1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
                0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
                1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1])
```

```
In [37]: dt1.predict_proba(X_Test)

Out[37]: array([[0., 1.],
                [0., 1.],
                [0., 1.],
                [1., 0.],
                [0., 1.],
                [0., 1.],
                [1., 0.],
                [0., 1.],
                [0., 1.],
                [0., 1.],
                [1., 0.],
                [0., 1.],
                [1., 0.],
                [1., 0.],
                [1., 0.],
                [1., 0.],
                [1., 0.],
                [1., 0.],
                [1., 0.],
```

Metricile de clasificare:

```
In [38]: #METRICA 1: Matricea de confuzie
         #METRICA 2: Accuracy;
         #METRICA 3: Precision
         #METRICA 4: Recall
         #METRICA 5: F1
         from sklearn import metrics
         matrix = confusion_matrix(Y_Test, pred)
         print("Confusion Matrix:")
         print(matrix)
         print("Accuracy score:",metrics.accuracy_score(Y_Test, pred))
         print("Precision score:",metrics.precision_score(Y_Test, pred))
         print("Recall score:",metrics.recall_score(Y_Test, pred))
         print("F1 score:",f1_score(Y_Test, pred))

         Confusion Matrix:
         [[78  3]
          [ 4 40]]
         Accuracy score: 0.944
         Precision score: 0.9302325581395349
         Recall score: 0.9090909090909091
         F1 score: 0.9195402298850575
```

Pentru acest apel al algoritmului am folosit parametri urmatori: criterion = 'entropy', care reprezinta functia cu care se masoara calitatea divizarii nodurilor la nivelul arborelui de decizie. De asemenea, pentru acest prin apel am facut si fit asupra setului de training si predict asupra setului de test.

Decision Tree folosind criteriul gini:

```
In [40]: dt2 = tree.DecisionTreeClassifier(criterion="gini")
         dt2

Out[40]: DecisionTreeClassifier()

In [41]: dt2.fit(X_Train, Y_Train)

Out[41]: DecisionTreeClassifier()

In [42]: X_Train

Out[42]: array([[-1.09544512, -0.3555065 , -0.31931143, ...,  0.46143898,
                  1.22884788,  0.42618806],
                [-1.09544512,  0.09879679, -0.92809211, ...,  0.46143898,
                 -0.38374548, -1.90641195],
                [-1.09544512, -0.3555065 , -1.63833623, ...,  0.46143898,
                  1.22884788,  0.42618806],
                ...,
                [ 0.91287093, -0.80980979,  0.1880058 , ...,  0.46143898,
                  1.22884788,  0.42618806],
                [ 0.91287093,  0.25023122,  0.1880058 , ...,  0.46143898,
                  1.22884788,  0.42618806],
                [ 0.91287093, -0.80980979,  0.59385958, ..., -0.96482696,
                  1.22884788,  0.42618806]])

In [43]: Y_Train

Out[43]: 64     0
         353    0
         45     1
         238    0
         363    0
               ..
         374    0
```

```
In [44]: pred = dt2.predict(X_Test)
         pred

Out[44]: array([1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
                1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
                1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
                1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
                1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0,
                1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1])

In [45]: dt2.predict_proba(X_Test)
                [1., 0.],
                [1., 0.],
                [1., 0.],
                [1., 0.],
                [1., 0.],
                [0., 1.],
                [0., 1.],
                [0., 1.],
                [0., 1.],
                [1., 0.],
                [1., 0.],
                [0., 1.],
                [1., 0.],
                [0., 1.],
                [1., 0.],
                [1., 0.],
                [1., 0.],
                [1., 0.],
                [1., 0.],
                [0., 1.],
```

Metricile de clasificare:

```
In [46]: #METRICA 1: Matricea de confuzie
         #METRICA 2: Accuracy;
         #METRICA 3: Precision
         #METRICA 4: Recall
         #METRICA 5: F1
         from sklearn import metrics
         matrix = confusion_matrix(Y_Test, pred)
         print("Confusion Matrix:")
         print(matrix)
         print("Accuracy score:",metrics.accuracy_score(Y_Test, pred))
         print("Precision score:",metrics.precision_score(Y_Test, pred))
         print("Recall score:",metrics.recall_score(Y_Test, pred))
         print("F1 score:",f1_score(Y_Test, pred))

         Confusion Matrix:
         [[78  3]
          [ 1 43]]
         Accuracy score: 0.968
         Precision score: 0.9347826086956522
         Recall score: 0.9772727272727273
         F1 score: 0.9555555555555557
```

Pentru acest apel al algoritmului am folosit parametri urmatori: criterion = 'gini', care este o masura a impuritatii sau puritatii utilizata in timpul crearii unui arbore de decizie in algoritmul CART (Classification and Regression Tree). De asemenea, pentru acest prin apel am facut si fit asupra setului de training si predict asupra setului de test.

2.2 Algorithm 2

Support Vector Machine este utilizat pentru probleme de clasificare si regresie.

Avand in vedere datele de antrenament etichetate, algoritmul produce cel mai bun hiperplan care a clasificat noi exemple. In spatiul bidimensional, acest hiperplan este o linie care imparte un plan in doua parti in care fiecare clasa se afla de o parte si de alta. Intentia algoritmului de masina vector suport este de a gasi un hiperplan intr-un spatiu N-dimensional care clasifica separat punctele de date.

Avantajele algoritmului:

- Support vector machine funcționeaza in mod comparabil bine atunci cand exista o marja de disociere de inteles intre clase.

- Este mai productiv in spatii cu dimensiuni mari.

- Este eficient in cazurile in care numarul de dimensiuni este mai mare decat numarul de specimene.

- Este comparabil cu memoria sistematica.

Dezavantajele algoritmului:

- Algoritmul de support vector machine nu este acceptabil pentru seturi mari de date.

- Nu se executa foarte bine atunci cand setul de date are mai mult sunet, adica clasele tinta se suprapun.

- In cazurile in care numarul de proprietati pentru fiecare punct de date depaseste numarul de specimene de date de antrenament, masina vectorului suport va avea performante slabe.

- Deoarece clasificatorul vector suport functionează prin plasarea punctelor de date, deasupra si sub hiperplanul de clasificare nu exista o clarificare probabilistica pentru clasificare.

SVM folosind clasificatorul SVC:

```
In [108]: #SVC

In [120]: #Antrenarea clasificatorului in Training set
          svc = SVC(kernel = 'linear', probability =True, random_state = 0)
          svc.fit(X_Train, Y_Train)

Out[120]: SVC(kernel='linear', probability=True, random_state=0)

In [121]: #Prezicerea rezultatelor setului de testare
          Y_Pred = svc.predict(X_Test)
          Y_Pred

Out[121]: array([1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1,
                 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
                 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
                 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0,
                 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0])

In [122]: Y_Test

Out[122]: 125    1
          28     0
          316    0
          330    0
          128    0
                ..
          406    0
          365    0
          21     1
          333    0
          253    0
          Name: NObeyesdad, Length: 125, dtype: int64

In [125]: my_test0 = [0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3]
          svc.predict([my_test0])

Out[125]: array([1])

In [126]: #METRICA 1: Matricea de confuzie
          #METRICA 2: Accuracy
          #METRICA 3: Precision
          #METRICA 4: Recall
          #METRICA 5: F1
          from sklearn import metrics
          print("Confusion matrix:")
          matrix = confusion_matrix(Y_Test, Y_Pred)
          print(matrix)
          print("Accuracy score:",metrics.accuracy_score(Y_Test, Y_Pred))
          print("Precision score:",metrics.precision_score(Y_Test, Y_Pred))
          print("Recall score:",metrics.recall_score(Y_Test, Y_Pred))
          print("F1 score:",f1_score(Y_Test, Y_Pred))

          Confusion matrix:
          [[81  1]
           [ 8 35]]
          Accuracy score: 0.928
          Precision score: 0.9722222222222222
          Recall score: 0.813953488372093
          F1 score: 0.8860759493670887
```

Pentru a apela algoritmul folosind clasificatorul SVC, l-am aplicat pe setul de date care contine toate feature-urile initiale. Parametrii folositi sunt kernel = 'linear' care se refera la coeficientul folosit pentru kernel, probability = True care decide daca se activeaza estimarile de probabilitate si random-state = 0 care controleaza numarul de numere pseudo-aleatoare folosite. De asemenea, si pentru acest apel am facut si fit asupra setului de training si predict asupra setului de test.

SVM folosind clasificatorul NuSVC:

```
In [170]: #NuSVC

In [33]: nusvc = NuSVC (nu =0.5 , kernel ='rbf', degree =3, gamma ='auto', coef0 =0.0, shrinking =True, probability =True,
                    tol =0.001, cache_size =200, class_weight = None, verbose =False, max_iter = -1, random_state = None)
         nusvc.fit(X_Train, Y_Train)

Out[33]: NuSVC(gamma='auto', probability=True)

In [35]: #Prezicerea rezultatelor setului de testare
         Y_Pred_nusvc = nusvc.predict(X_Test)
         Y_Pred_nusvc

Out[35]: array([0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0,
                1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
                1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
                0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0])

In [36]: Y_Test

Out[36]: 378    0
         426    0
         366    1
         486    0
         338    1
               ..
         411    1
         164    1
         303    0
         319    1
         464    0
         Name: NObeyesdad, Length: 125, dtype: int64

In [37]: my_test1 = [0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3]
         nusvc.predict([my_test1])

Out[37]: array([1])

In [38]: #METRICA 1: Matricea de confuzie
         #METRICA 2: Accuracy
         #METRICA 3: Precision
         #METRICA 4: Recall
         #METRICA 5: F1
         from sklearn import metrics
         print("Confusion matrix:")
         matrix = confusion_matrix(Y_Test, Y_Pred)
         print(matrix)
         print("Accuracy score:",metrics.accuracy_score(Y_Test, Y_Pred))
         print("Precision score:",metrics.precision_score(Y_Test, Y_Pred))
         print("Recall score:",metrics.recall_score(Y_Test, Y_Pred))
         print("F1 score:",f1_score(Y_Test, Y_Pred))

         Confusion matrix:
         [[79  1]
          [ 3 42]]
         Accuracy score: 0.968
         Precision score: 0.9767441860465116
         Recall score: 0.9333333333333333
         F1 score: 0.9545454545454545
```

Pentru un alt apel al algoritmului SVM am folosit clasificatorul NuSVC, l-am aplicat din nou pe intreg setul de date care contine toate feature-urile intiale. Parametrii folositi sunt cei default, care descriu orice clasificator NuSVC. De asemenea, si pentru acest apel am facut si fit asupra setului de training si predict asupra setului de test.

2.3 Ensemble Method 1

Bagging method

Aceasta metoda de ansamblu combina doua modele de invatare automata, adica Bootstrapping si Aggregation, intr-un singur model de ansamblu. Obiectivul metodei de bagging este reducerea variatiei mari a modelului. Arborii de decizie au varianta si partinire scazuta. Setul mare de date este (sa zicem 1000 de esantioane) sub-esantionat (sa zicem 10 sub-probe fiecare poarta 100 de esantioane de date). Arborele de decizie multipli sunt construite pe fiecare sub-esantion de date de antrenament. In timp ce se exploateaza datele sub-esantionate pe diferitii arbori de decizie, preocuparea de supra-adaptare a datelor de antrenament pe fiecare arbore de decizie este redusa. Pentru eficienta modelului, fiecare dintre arborii de decizie individuale este crescut in adancime, care contine date de antrenament sub-esantionate. Rezultatele fiecarui arbore de decizie sunt agregate pentru a intelege predictia finala. Varianta datelor agregate vine sa se reduca. Precizia predictiei modelului in metoda de bagging depinde de numarul de arbore de decizie utilizat. Diferitele sub-esantion ale unui esantion de date sunt alese aleatoriu cu inlocuire. Iesirea fiecarui arbore are o corelatie ridicata.

Am aplicat atat metoda de Random Forest Clasification, cat si metoda de ExtraTrees Clasification pornind de algoritmul Decision Tree.

Pentru aceasta metoda am impartit datele in test si train, am testat clasificatorul folosind 3 sample-uri si am afisat cele 5 metrici pentru clasificare. De asemenea, am apelat de doua ori aceasta metoda pentru a folosi parametri diferiti (am schimbat num trees si max features).

```
In [81]: #Decision Tree Classification
         dt = DecisionTreeClassifier()
         scores = cross_val_score(dt, X, Y, cv=5)
         print(scores.mean())

         0.9618787878787879
```

Algoritmul de Bagging folosind modelul Random Forest Clasification pentru DT:

```
#pentru primul set de parametri
num_trees = 100
max_features = 3
rmodel = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
results2 = cross_val_score(rmodel, X, Y, cv=5)
print(results2.mean())

#1. split the dataset: train and test and then fit the classifier using X_train and Y_train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
0.8877373737373737
[0 0 1]
Y_prediction: [1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 0 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1
 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0]
Y_test: 90     1
254    0
283    0
444    0
474    0
      ..
381    0
56     0
439    0
60     0
208    0
Name: NObeyesdad, Length: 100, dtype: int64
Confusion Matrix:
[[56  4]
 [ 8 32]]
Accuracy score: 0.88
Precision score: 0.8888888888888888
Recall score: 0.8
F1 score: 0.8421052631578948
```

```
#pentru al doilea set de parametri
num_trees2 = 300
max_features2 = 5
rmodel = RandomForestClassifier(n_estimators=num_trees2, max_features=max_features2)
results2 = cross_val_score(rmodel, X, Y, cv=5)
print(results2.mean())

#1. split the dataset: train and test and then fit the classifier using X_train and Y_train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
0.8937373737373738
[0 0 1]
Y_prediction: [1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0
 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0]
Y_test: 90     1
254    0
283    0
444    0
474    0
      ..
381    0
56     0
439    0
60     0
208    0
Name: NObeyesdad, Length: 100, dtype: int64
Confusion Matrix:
[[57  3]
 [ 6 34]]
Accuracy score: 0.91
Precision score: 0.918918918918919
Recall score: 0.85
F1 score: 0.8831168831168831
```

Algoritmul de Bagging folosind modelul ExtraTrees Clasification pentru DT:

La fel si pentru aceasta metoda am impartit datele in test si train, am testat clasificatorul folosind 3 sample-uri si am afisat cele 5 metrici pentru clasificare. De asemenea, am apelat de doua ori aceasta metoda pentru a folosi parametri diferiti (n estimators).

```python
#pentru primul set de parametri
clf = ExtraTreesClassifier(n_estimators=10, max_depth=None, min_samples_split=2, random_state=0)
scores = cross_val_score(clf, X, Y, cv=5)
print(scores.mean())

#1. split the dataset: train and test and then fit the classifier using X_train and Y_train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
0.8176161616161617
[0 0 1]
Y_prediction: [1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0
 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0]
Y_test: 90     1
254    0
283    0
444    0
474    0
      ..
381    0
56     0
439    0
60     0
208    0
Name: NObeyesdad, Length: 100, dtype: int64
Confusion Matrix:
[[57  3]
 [ 6 34]]
Accuracy score: 0.91
Precision score: 0.918918918918919
Recall score: 0.85
F1 score: 0.8831168831168831
```

```python
#pentru al doilea set de parametri
clf = ExtraTreesClassifier(n_estimators=15, max_depth=None, min_samples_split=5, random_state=0)
scores = cross_val_score(clf, X, Y, cv=5)
print(scores.mean())

#1. split the dataset: train and test and then fit the classifier using X_train and Y_train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
0.8316969696969696
[0 0 1]
Y_prediction: [1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0
 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0]
Y_test: 90     1
254    0
283    0
444    0
474    0
      ..
381    0
56     0
439    0
60     0
208    0
Name: NObeyesdad, Length: 100, dtype: int64
Confusion Matrix:
[[57  3]
 [ 6 34]]
Accuracy score: 0.91
Precision score: 0.918918918918919
Recall score: 0.85
F1 score: 0.8831168831168831
```

Totodata, am aplicat metodele Random Forest Clasifier si ExtraTrees Clasifier pornind si de la algoritmul Support Vector Machine.

Algoritmul de Bagging folosind modelul Random Forest Clasification pentru SVM:

SVM cu clasificatorul SVC:

```
#BAGGING: Random Forest Clasifier
num_trees = 100
max_features = 2
rmodel = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
results2 = cross_val_score(rmodel, X, Y, cv=5)
print(results2.mean())

#1. split the dataset: train and test and then fit the classifier using X_train and Y_train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
0.8797171717171718
[0 0 1]
Y_prediction: [1 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 0 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 0 0 1
 0 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 1 0 0 0 0 0]
Y_test: 90     1
254    0
283    0
444    0
474    0
      ..
381    0
56     0
439    0
60     0
208    0
Name: NObeyesdad, Length: 100, dtype: int64
Confusion Matrix:
[[57  3]
 [ 9 31]]
Accuracy score: 0.88
Precision score: 0.9117647058823529
Recall score: 0.775
F1 score: 0.8378378378378379
```

SVM cu clasificatorul NuSVC:

```
#Random Forest Clasifier
num_trees = 100
max_features = 5
rmodel = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
results2 = cross_val_score(rmodel, X, Y, cv=5)
print(results2.mean())

#1. split the dataset: train and test and then fit the classifier using X_train and Y_train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
0.8997777777777778
[0 0 1]
Y_prediction: [1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0
 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0]
Y_test: 90     1
254    0
283    0
444    0
474    0
      ..
381    0
56     0
439    0
60     0
208    0
Name: NObeyesdad, Length: 100, dtype: int64
Confusion Matrix:
[[57  3]
 [ 6 34]]
Accuracy score: 0.91
Precision score: 0.918918918918919
Recall score: 0.85
F1 score: 0.8831168831168831
```

Algoritmul de Bagging folosind modelul ExtraTrees Clasification pentru SVM:

SVM cu clasificatorul SVC:

```
#BAGGING: EXTRATREES
clf = ExtraTreesClassifier(n_estimators=10, max_depth=None, min_samples_split=2, random_state=0)
scores = cross_val_score(clf, X, Y, cv=5)
print(scores.mean())

#1. split the dataset: train and test and then fit the classifier using X_train and Y_train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
0.8176161616161617
[0 0 1]
Y_prediction: [1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 0 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 0 0 1
 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0]
Y_test: 90     1
254    0
283    0
444    0
474    0
      ..
381    0
56     0
439    0
60     0
208    0
Name: NObeyesdad, Length: 100, dtype: int64
Confusion Matrix:
[[56  4]
 [ 7 33]]
Accuracy score: 0.89
Precision score: 0.8918918918918919
Recall score: 0.825
F1 score: 0.8571428571428571
```

SVM cu clasificatorul NuSVC:

```
#EXTRA TREE
clf = ExtraTreesClassifier(n_estimators=10, max_depth=None, min_samples_split=2, random_state=0)
scores = cross_val_score(clf, X, Y, cv=5)
print(scores.mean())

#1. split the dataset: train and test and then fit the classifier using X_train and Y_train
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
0.8176161616161617
[0 0 1]
Y_prediction: [1 0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0
 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0]
Y_test: 90     1
254    0
283    0
444    0
474    0
      ..
381    0
56     0
439    0
60     0
208    0
Name: NObeyesdad, Length: 100, dtype: int64
Confusion Matrix:
[[58  2]
 [ 6 34]]
Accuracy score: 0.92
Precision score: 0.9444444444444444
Recall score: 0.85
F1 score: 0.8947368421052632
```

2.4 Ensemble Method 2

Boosting method

Metode de boosting combină, de asemenea, acelasi tip de clasificator. Boostingul este una dintre metodele de ansamblu secvential in care fiecare model sau clasificator ruleaza pe baza caracteristicilor pe care le va utiliza urmatorul model. In acest fel, metoda de amplificare evidentiaza un model de invatare mai puternic din modelele de invatare slabe, facand o medie a ponderilor acestora. Cu alte cuvinte, un model antrenat mai puternic depinde de multiplele modele antrenate slabe. Un cursant slab sau un model instruit la uzura este unul care este foarte putin corelat cu clasificarea adevarata. Dar urmatorul cursant slab este putin mai corelat cu clasificarea adevarata. Combinatia unor astfel de elevi slabi diferiti ofera un cursant puternic, care este bine corelat cu adevarata clasificare.

Am aplicat metoda de AdaBoost Clasifier, cat si metoda Gradient Tree Boosting pornind de la algoritmul Decision Tree.

```
In [11]: #Decision Tree Classification
         dt = DecisionTreeClassifier()
         scores = cross_val_score(dt, X, Y, cv=5)
         print(scores.mean())

         0.9618787878787879
```

Algoritmul de Boosting folosind metoda AdaBoost Clasification pentru DT:

Pentru aceasta metoda am impartit datele in test si train, am prezis datele, am testat clasificatorul folosind 3 sample-uri si am afisat cele 5 metrici pentru clasificare. De asemenea, am apelat de doua ori aceasta metoda pentru a folosi parametri diferiti (am schimbat seeds si num trees).

```
#pentru primul set de parametri
seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=seed)

model = AdaBoostClassifier(base_estimator = dt, n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)

print(results)
print(results.mean())

#split the dataset: train + test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
model.fit(X_train,Y_train)

#Predict the response for test dataset
Y_prediction = model.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#test the model using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1
Y_prediction_test= model.predict(my_tests)
print("Prediction form my given test:",Y_prediction_test)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
[0.92       1.         0.96       1.         0.98       0.96
 0.94       0.98       0.94       0.93877551]
0.9618775510204081
Y_prediction: [1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0
 0 1 0 0 1 0 0 0 0 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0]
Y_test: 90     1
254    0
283    0
444    0
474    0
      ..
381    0
56     0
439    0
60     0
208    0
Name: NObeyesdad, Length: 100, dtype: int64
Prediction form my given test: [0 0 1]
Confusion Matrix:
[[59  1]
 [ 2 38]]
Accuracy score: 0.97
Precision score: 0.9743589743589743
Recall score: 0.95
F1 score: 0.9620253164556962
```

```
#pentru al doilea set de parametri
seed = 5
num_trees = 50
kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=seed)

model = AdaBoostClassifier(base_estimator = dt, n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)

print(results)
print(results.mean())

#split the dataset: train + test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
model.fit(X_train,Y_train)

#Predict the response for test dataset
Y_prediction = model.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#test the model using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1
Y_prediction_test= model.predict(my_tests)
print("Prediction form my given test:",Y_prediction_test)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
[0.96       0.92       0.94       0.96       1.         0.96
 0.92       0.98       0.96       0.93877551]
0.953877551020408
Y_prediction: [1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 0 1 0 1 1 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0
 0 1 0 0 1 0 0 0 0 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0]
Y_test: 90     1
254    0
283    0
444    0
474    0
      ..
381    0
56     0
439    0
60     0
208    0
Name: NObeyesdad, Length: 100, dtype: int64
Prediction form my given test: [0 0 1]
Confusion Matrix:
[[59  1]
 [ 3 37]]
Accuracy score: 0.96
Precision score: 0.9736842105263158
Recall score: 0.925
F1 score: 0.9487179487179489
```

Algoritmul de Boosting folosind metoda Gradient Tree Boosting pentru DT:

Pentru aceasta metoda am impartit datele in test si train, am antrenat datele, le-am prezis si am afisat cele 5 metrici pentru clasificare. De asemenea, am apelat de doua ori aceasta metoda pentru a folosi parametri diferiti (am schimbat num trees).

```
#pentru primul set de parametri
num_trees = 100
kfold = model_selection.KFold(n_splits=10)

model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

X_for_train, X_for_test, Y_for_train, Y_for_test = train_test_split(X, Y, test_size = 0.15, random_state = 0)

my_test = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,4

# Train GradientBoostingClassifier: fitting the X and Y
gboost = GradientBoostingClassifier(n_estimators=100,learning_rate=1)
gboost.fit(X_for_train, Y_for_train)

#Predict the response for test dataset
Y_prediction = gboost.predict(X_test)
# test using the sample given: test the model using 3 new samples, and print the results
Y_prediction_test= gboost.predict(my_test)

#print the predictions
print("Prediction form my given test:",Y_prediction_test)
print("Prediction:",Y_prediction)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
0.9638775510204081
Prediction form my given test: [0 0 1]
Prediction: [1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0
 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 0 0 0]
Confusion Matrix:
[[60  0]
 [ 3 37]]
Accuracy score: 0.97
Precision score: 1.0
Recall score: 0.925
F1 score: 0.961038961038961
```

```
#pentru al doilea set de parametri
num_trees = 300
kfold = model_selection.KFold(n_splits=10)

model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

X_for_train, X_for_test, Y_for_train, Y_for_test = train_test_split(X, Y, test_size = 0.15, random_state = 0)

my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,

# Train GradientBoostingClassifier: fitting the X and Y
gboost = GradientBoostingClassifier(n_estimators=100,learning_rate=1)
gboost.fit(X_for_train, Y_for_train)

#Predict the response for test dataset
Y_prediction = gboost.predict(X_test)
# test using the sample given: test the model using 3 new samples, and print the results
Y_prediction_test= gboost.predict(my_test)

#print the predictions
print("Prediction form my given test:",Y_prediction_test)
print("Prediction:",Y_prediction)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
0.9658775510204081
Prediction form my given test: [0 0 1]
Prediction: [1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0
 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 0 0 0]
Confusion Matrix:
[[60  0]
 [ 3 37]]
Accuracy score: 0.97
Precision score: 1.0
Recall score: 0.925
F1 score: 0.961038961038961
```

Totodata, am aplicat metodele de AdaBoost Clasifier si Gradient Tree Boosting pornind si de la algoritmul Support Vector Machine.

Algoritmul de Boosting folosind metoda AdaBoost Clasification pentru SVM:

SVM cu clasificatorul SVC:

```
#BOOSTING: ADABOOST
seed = 5
num_trees = 30
kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=seed)

model = AdaBoostClassifier(base_estimator = svc, n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)

print(results)
print(results.mean())

#split the dataset: train + test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
model.fit(X_train,Y_train)

#Predict the response for test dataset
Y_prediction = model.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#test the model using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,
Y_prediction_test= model.predict(my_tests)
print("Prediction form my given test:",Y_prediction_test)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
[0.92       0.98       0.88       0.9        0.94       0.96
 0.9        0.94       0.82       0.81632653]
0.9056326530612246
Y_prediction: [1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 1 0 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0
 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0]
Y_test: 90     1
254    0
283    0
444    0
474    0
      ..
381    0
56     0
439    0
60     0
208    0
Name: NObeyesdad, Length: 100, dtype: int64
Prediction form my given test: [0 0 1]
Confusion Matrix:
[[60  0]
 [ 6 34]]
Accuracy score: 0.94
Precision score: 1.0
Recall score: 0.85
F1 score: 0.9189189189189189
```

SVM cu clasificatorul NuSVC:

```python
#BOOSTING: ADABOOST
seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=seed)

model = AdaBoostClassifier(base_estimator = nusvc, n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)

print(results)
print(results.mean())

#split the dataset: train + test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 0)
model.fit(X_train,Y_train)

#Predict the response for test dataset
Y_prediction = model.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#test the model using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,
Y_prediction_test= model.predict(my_tests)
print("Prediction form my given test:",Y_prediction_test)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
[0.92       0.98       0.96       0.92       0.96       0.92
 0.96       0.98       0.94       0.97959184]
0.9519591836734692
Y_prediction: [1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1 0 1 1 1 0 1 1 0 1 1 0 1 1 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0
 0 1 0 0 1 0 0 0 0 1 1 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0]
Y_test: 90     1
254    0
283    0
444    0
474    0
      ..
381    0
56     0
439    0
60     0
208    0
Name: NObeyesdad, Length: 100, dtype: int64
Prediction form my given test: [0 0 1]
Confusion Matrix:
[[58  2]
 [ 2 38]]
Accuracy score: 0.96
Precision score: 0.95
Recall score: 0.95
F1 score: 0.9500000000000001
```

Algoritmul de Boosting folosind metoda Gradient Tree Boosting pentru SVM:
SVM cu clasificatorul SVC:

```
#GRADIENT BOOSTING
num_trees = 100
kfold = model_selection.KFold(n_splits=10)

model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

X_for_train, X_for_test, Y_for_train, Y_for_test = train_test_split(X, Y, test_size = 0.15, random_state = 0)

my_test = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,4

# Train GradientBoostingClassifier: fitting the X and Y
gboost = GradientBoostingClassifier(n_estimators=100,learning_rate=1)
gboost.fit(X_for_train, Y_for_train)

#Predict the response for test dataset
Y_prediction = gboost.predict(X_test)
# test using the sample given: test the model using 3 new samples, and print the results
Y_prediction_test= gboost.predict(my_test)

#print the predictions
print("Prediction form my given test:",Y_prediction_test)
print("Prediction:",Y_prediction)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
0.9638775510204081
Prediction form my given test: [0 0 1]
Prediction: [1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0
 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 0 0 0]
Confusion Matrix:
[[60  0]
 [ 3 37]]
Accuracy score: 0.97
Precision score: 1.0
Recall score: 0.925
F1 score: 0.961038961038961
```

SVM cu clasificatorul NuSVC:

```
#GRADIENT BOOSTING
num_trees = 50
kfold = model_selection.KFold(n_splits=10)

model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

X_for_train, X_for_test, Y_for_train, Y_for_test = train_test_split(X, Y, test_size = 0.15, random_state = 0)

my_test = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3],[1,14,33,57,0,0,2,1,2,0,2,0,3,0,1,4

# Train GradientBoostingClassifier: fitting the X and Y
gboost = GradientBoostingClassifier(n_estimators=100,learning_rate=1)
gboost.fit(X_for_train, Y_for_train)

#Predict the response for test dataset
Y_prediction = gboost.predict(X_test)
# test using the sample given: test the model using 3 new samples, and print the results
Y_prediction_test= gboost.predict(my_test)

#print the predictions
print("Prediction form my given test:",Y_prediction_test)
print("Prediction:",Y_prediction)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
0.9618775510204081
Prediction form my given test: [0 0 1]
Prediction: [1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0
 1 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0
 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 0 0 0]
Confusion Matrix:
[[60  0]
 [ 3 37]]
Accuracy score: 0.97
Precision score: 1.0
Recall score: 0.925
F1 score: 0.961038961038961
```

Chapter 3

Analysis of results - Comparisons

Deoarece setul de date nu a fost unul destul de echilibrat, pentru metricile de clasificare pe care le-am ales la fiecare metoda realizata mai sus, nu s-a obtinut o diferenta foarte mare intre metrici.

Chapter 4

Conclusion

In concluzie, in cadrul acestui proiect am facut o recapitulare a doi algoritmi de machine learning: Decision Tree si Support Vector Machine. Le-am analizat comportamentul, am observat ca pentru SVM rularea dureaza mai mult timp dar este mai eficient ca si DT. Totodata, am introdus si doua metode care tin de ensemble methods: Bagging si Boosting, pe care i-am verificat atat pe DT cat si pe SVM.

Appendix A

Your original code

This section should contain only code developed by you, without any line re-used from other sources. This section helps me to correctly evaluate your amount of work and results obtained. Including in this section any line of code taken from someone else leads to failure of IS class this year. Failing or forgetting to add your code in this appendix leads to grade 1. Don't remove the above lines.

```
###DECISION TREE

import os
import subprocess

import pandas as pd
import numpy as np
import graphviz
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn import tree

# In[13]:

#citesc setul de date
datasets = pd.read_csv('ObesityDataSet.csv')

# In[14]:

#afisez primele 5
datasets.head()

# In[15]:

datasets.shape

# In[16]:
```

```
datasets.dtypes

# In[17]:

le_ObesityLevel = LabelEncoder()
le_ObesityLevel.fit(datasets['NObeyesdad'])
data_encoded_ObL = le_ObesityLevel.transform(datasets['NObeyesdad'])
data_encoded_ObL

# In[18]:

le_ObesityLevel.classes_

# In[19]:

datasets_1 = datasets.apply(le_ObesityLevel.fit_transform) # for all the attr
datasets_1

# In[20]:

X = datasets_1[datasets.columns.drop('NObeyesdad')]
Y = datasets_1['NObeyesdad']

# In[21]:

X

# In[22]:

Y

# In[23]:

#Impart dataset-ul in Training set si Test set
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.25)

# In[24]:

X_Train

# In[25]:

Y_Train

# In[26]:

print(len(X_Train))
print(len(X_Test))
print(len(Y_Train))
```

```
print(len(Y_Test))

# In[27]:

#Feature Scaling
stdScaler = StandardScaler() #Standardize features by removing the mean and s
X_Train = stdScaler.fit_transform(X_Train)
X_Test = stdScaler.transform(X_Test)

# In[28]:

X_Train

# In[29]:

Y_Train

# In[34]:

dt1 = tree.DecisionTreeClassifier(criterion="entropy")
dt1

# In[35]:

dt1.fit(X_Train, Y_Train)

# In[36]:

pred = dt1.predict(X_Test)
pred

# In[37]:

dt1.predict_proba(X_Test)

# In[38]:

#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_Test, pred)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_Test, pred))
print("Precision score:",metrics.precision_score(Y_Test, pred))
print("Recall score:",metrics.recall_score(Y_Test, pred))
print("F1 score:",f1_score(Y_Test, pred))
```

```
# In[40]:

dt2 = tree.DecisionTreeClassifier(criterion="gini")
dt2

# In[41]:

dt2.fit(X_Train, Y_Train)

# In[42]:

X_Train

# In[43]:

Y_Train

# In[44]:

pred = dt2.predict(X_Test)
pred

# In[45]:

dt2.predict_proba(X_Test)

# In[46]:

#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_Test, pred)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_Test, pred))
print("Precision score:",metrics.precision_score(Y_Test, pred))
print("Recall score:",metrics.recall_score(Y_Test, pred))
print("F1 score:",f1_score(Y_Test, pred))


###SUPPORT VECTOR MACHINE

import os
import subprocess

import pandas as pd
```

```
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
from sklearn.svm import NuSVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import cross_val_score
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier

# In[45]:

#citesc setul de date
datasets = pd.read_csv('ObesityDataSet.csv')

# In[46]:

#afisez primele 5
datasets.head()

# In[47]:

datasets.shape

# In[48]:

datasets.dtypes

# In[49]:

le_ObesityLevel = LabelEncoder()
le_ObesityLevel.fit(datasets['NObeyesdad'])
data_encoded_ObL = le_ObesityLevel.transform(datasets['NObeyesdad'])
data_encoded_ObL

# In[50]:

le_ObesityLevel.classes_

# In[51]:

datasets_1 = datasets.apply(le_ObesityLevel.fit_transform) # for all the attr
datasets_1
```

```
# In[52]:

X = datasets_1[datasets.columns.drop('NObeyesdad')]
Y = datasets_1['NObeyesdad']

# In[53]:

X

# In[54]:

Y

# In[55]:

#Impart dataset-ul in Training set si Test set
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.25)

# In[56]:

X_Train

# In[57]:

print(len(X_Train))
print(len(X_Test))
print(len(Y_Train))
print(len(Y_Test))

# In[58]:

#Feature Scaling
stdScaler = StandardScaler() #Standardize features by removing the mean and s
X_Train = stdScaler.fit_transform(X_Train)
X_Test = stdScaler.transform(X_Test)

# In[59]:

X_Train

# In[60]:

Y_Train

# In[61]:

#SVC

# In[62]:
```

```
#Antrenarea clasificatorului in Training set
svc = SVC(kernel = 'linear', probability =True, random_state = 0)
svc.fit(X_Train, Y_Train)

# In[63]:

#Prezicerea rezultatelor setului de testare
Y_Pred = svc.predict(X_Test)
Y_Pred

# In[64]:

Y_Test

# In[65]:

my_test0 = [0,7,4,19,1,0,2,1,2,1,3,1,4,0,2,3]
svc.predict([my_test0])

# In[66]:

#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
print("Confusion matrix:")
matrix = confusion_matrix(Y_Test, Y_Pred)
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_Test, Y_Pred))
print("Precision score:",metrics.precision_score(Y_Test, Y_Pred))
print("Recall score:",metrics.recall_score(Y_Test, Y_Pred))
print("F1 score:",f1_score(Y_Test, Y_Pred))

# In[67]:

#BAGGING: Random Forest Clasifier
num_trees = 100
max_features = 2
rmodel = RandomForestClassifier(n_estimators=num_trees, max_features=max_feat
results2 = cross_val_score(rmodel, X, Y, cv=5)
print(results2.mean())

#1. split the dataset: train and test and then fit the classifier using X_tra
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, r
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
```

```
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))

# In[69]:

#BAGGING: EXTRATREES
clf = ExtraTreesClassifier(n_estimators=10, max_depth=None, min_samples_split
scores = cross_val_score(clf, X, Y, cv=5)
print(scores.mean())

#1. split the dataset: train and test and then fit the classifier using X_tra
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, r
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
```

```
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))

# In[75]:

#BOOSTING: ADABOOST
seed = 5
num_trees = 30
kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=seed)

model = AdaBoostClassifier(base_estimator = svc, n_estimators=num_trees, ranc
results = model_selection.cross_val_score(model, X, Y, cv=kfold)

print(results)
print(results.mean())

#split the dataset: train + test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, r
model.fit(X_train,Y_train)

#Predict the response for test dataset
Y_prediction = model.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#test the model using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0
Y_prediction_test= model.predict(my_tests)
print("Prediction form my given test:",Y_prediction_test)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
```

```
print("F1 score:",f1_score(Y_test, Y_prediction))

# In[76]:

#GRADIENT BOOSTING
num_trees = 100
kfold = model_selection.KFold(n_splits=10)

model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

X_for_train, X_for_test, Y_for_train, Y_for_test = train_test_split(X, Y, tes

my_test = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0

# Train GradientBoostingClassifier: fitting the X and Y
gboost = GradientBoostingClassifier(n_estimators=100,learning_rate=1)
gboost.fit(X_for_train, Y_for_train)

#Predict the response for test dataset
Y_prediction = gboost.predict(X_test)
# test using the sample given: test the model using 3 new samples, and print
Y_prediction_test= gboost.predict(my_test)

#print the predictions
print("Prediction form my given test:",Y_prediction_test)
print("Prediction:",Y_prediction)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))

# In[170]:

#NuSVC

# In[79]:
```

```
nusvc = NuSVC (nu =0.5 , kernel ='rbf', degree =3, gamma ='auto', coef0 =0.0,
                tol =0.001, cache_size =200, class_weight = None, verbose =Fal
nusvc.fit(X_Train, Y_Train)

# In[80]:

#Prezicerea rezultatelor setului de testare
Y_Pred_nusvc = nusvc.predict(X_Test)
Y_Pred_nusvc

# In[81]:

Y_Test

# In[82]:

my_test1 = [0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3]
nusvc.predict([my_test1])

# In[83]:

#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
print("Confusion matrix:")
matrix = confusion_matrix(Y_Test, Y_Pred)
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_Test, Y_Pred))
print("Precision score:",metrics.precision_score(Y_Test, Y_Pred))
print("Recall score:",metrics.recall_score(Y_Test, Y_Pred))
print("F1 score:",f1_score(Y_Test, Y_Pred))

# In[84]:

#Random Forest Clasifier
num_trees = 100
max_features = 5
rmodel = RandomForestClassifier(n_estimators=num_trees, max_features=max_feat
results2 = cross_val_score(rmodel, X, Y, cv=5)
print(results2.mean())

#1. split the dataset: train and test and then fit the classifier using X_tra
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, r
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0
```

```
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))

# In[85]:

#EXTRA TREE
clf = ExtraTreesClassifier(n_estimators=10, max_depth=None, min_samples_split
scores = cross_val_score(clf, X, Y, cv=5)
print(scores.mean())

#1. split the dataset: train and test and then fit the classifier using X_tra
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, r
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
```

```
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))

# In[86]:

#BOOSTING: ADABOOST
seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=seed)

model = AdaBoostClassifier(base_estimator = nusvc, n_estimators=num_trees, ra
results = model_selection.cross_val_score(model, X, Y, cv=kfold)

print(results)
print(results.mean())

#split the dataset: train + test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, r
model.fit(X_train,Y_train)

#Predict the response for test dataset
Y_prediction = model.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#test the model using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0
Y_prediction_test= model.predict(my_tests)
print("Prediction form my given test:",Y_prediction_test)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

```
# In[88]:

#GRADIENT BOOSTING
num_trees = 50
kfold = model_selection.KFold(n_splits=10)

model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

X_for_train, X_for_test, Y_for_train, Y_for_test = train_test_split(X, Y, tes

my_test = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0

# Train GradientBoostingClassifier: fitting the X and Y
gboost = GradientBoostingClassifier(n_estimators=100,learning_rate=1)
gboost.fit(X_for_train, Y_for_train)

#Predict the response for test dataset
Y_prediction = gboost.predict(X_test)
# test using the sample given: test the model using 3 new samples, and print
Y_prediction_test= gboost.predict(my_test)

#print the predictions
print("Prediction form my given test:",Y_prediction_test)
print("Prediction:",Y_prediction)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))


###BAGGING METHOD

import os
import subprocess
import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import *
from sklearn import metrics
from sklearn.metrics import *

# In[3]:

#citesc setul de date
datasets = pd.read_csv('ObesityDataSet.csv')

# In[4]:

#afisez primele 5
datasets.head()

# In[5]:

datasets.columns

# In[6]:

le_ObesityLevel = LabelEncoder()
le_ObesityLevel.fit(datasets['NObeyesdad'])
data_encoded_ObL = le_ObesityLevel.transform(datasets['NObeyesdad'])
data_encoded_ObL

# In[7]:

le_ObesityLevel.classes_

# In[8]:

datasets_1 = datasets.apply(le_ObesityLevel.fit_transform) # for all the attr
datasets_1

# In[9]:

X = datasets_1[datasets.columns.drop('NObeyesdad')]
Y = datasets_1['NObeyesdad']

# In[10]:

X

# In[11]:
```

```
Y

# In[12]:

#Decision Tree Classification
dt = DecisionTreeClassifier()
scores = cross_val_score(dt, X, Y, cv=5)
print(scores.mean())

# In[82]:

#RANDOM FOREST CLASIFICATION

# In[83]:

#pentru primul set de parametri
num_trees = 100
max_features = 3
rmodel = RandomForestClassifier(n_estimators=num_trees, max_features=max_feat
results2 = cross_val_score(rmodel, X, Y, cv=5)
print(results2.mean())

#1. split the dataset: train and test and then fit the classifier using X_tra
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, r
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
```

```
print("F1 score:",f1_score(Y_test, Y_prediction))

# In[13]:

#pentru al doilea set de parametri
num_trees2 = 300
max_features2 = 5
rmodel = RandomForestClassifier(n_estimators=num_trees2, max_features=max_fea
results2 = cross_val_score(rmodel, X, Y, cv=5)
print(results2.mean())

#1. split the dataset: train and test and then fit the classifier using X_tra
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, r
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))

# In[14]:

#EXTRATREES CLASIFICATION

# In[15]:

#pentru primul set de parametri
clf = ExtraTreesClassifier(n_estimators=10, max_depth=None, min_samples_split
scores = cross_val_score(clf, X, Y, cv=5)
print(scores.mean())
```

```
#1. split the dataset: train and test and then fit the classifier using X_tra
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, r
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))

# In[16]:

#pentru al doilea set de parametri
clf = ExtraTreesClassifier(n_estimators=15, max_depth=None, min_samples_split
scores = cross_val_score(clf, X, Y, cv=5)
print(scores.mean())

#1. split the dataset: train and test and then fit the classifier using X_tra
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, r
rmodel.fit(X_train,Y_train)

#2. test the classifier using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0
Y_prediction_test= rmodel.predict(my_tests)
print(Y_prediction_test)

#3. print the predictions
Y_prediction = rmodel.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)
```

```
#4. check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))


###BOOSTING METHOD

import os
import subprocess
import pandas as pd
import numpy as np
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import *
from sklearn.preprocessing import LabelEncoder
from sklearn import metrics
from sklearn.metrics import *
from sklearn import model_selection

# In[2]:

#citesc setul de date
datasets = pd.read_csv('ObesityDataSet.csv')

# In[3]:

#afisez primele 5
datasets.head()

# In[4]:

datasets.columns

# In[5]:
```

```
le_ObesityLevel = LabelEncoder()
le_ObesityLevel.fit(datasets['NObeyesdad'])
data_encoded_ObL = le_ObesityLevel.transform(datasets['NObeyesdad'])
data_encoded_ObL

# In[6]:

le_ObesityLevel.classes_

# In[7]:

datasets_1 = datasets.apply(le_ObesityLevel.fit_transform) # for all the attr
datasets_1

# In[8]:

X = datasets_1[datasets.columns.drop('NObeyesdad')]
Y = datasets_1['NObeyesdad']

# In[9]:

X

# In[10]:

Y

# In[11]:

#Decision Tree Classification
dt = DecisionTreeClassifier()
scores = cross_val_score(dt, X, Y, cv=5)
print(scores.mean())

# In[12]:

#ADABOOST CLASIFIER

# In[14]:

#pentru primul set de parametri
seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=seed)

model = AdaBoostClassifier(base_estimator = dt, n_estimators=num_trees, rando
results = model_selection.cross_val_score(model, X, Y, cv=kfold)

print(results)
print(results.mean())
```

```
#split the dataset: train + test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, r
model.fit(X_train,Y_train)

#Predict the response for test dataset
Y_prediction = model.predict(X_test)
print("Y_prediction:",Y_prediction)
print("Y_test:",Y_test)

#test the model using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0
Y_prediction_test= model.predict(my_tests)
print("Prediction form my given test:",Y_prediction_test)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))

# In[15]:

#pentru al doilea set de parametri
seed = 5
num_trees = 50
kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=seed)

model = AdaBoostClassifier(base_estimator = dt, n_estimators=num_trees, rando
results = model_selection.cross_val_score(model, X, Y, cv=kfold)

print(results)
print(results.mean())

#split the dataset: train + test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, r
model.fit(X_train,Y_train)

#Predict the response for test dataset
Y_prediction = model.predict(X_test)
print("Y_prediction:",Y_prediction)
```

```
print("Y_test:",Y_test)

#test the model using 3 samples
my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,(
Y_prediction_test= model.predict(my_tests)
print("Prediction form my given test:",Y_prediction_test)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))

# In[16]:

#GRADIENT BOOSTING

# In[18]:

#pentru primul set de parametri
num_trees = 100
kfold = model_selection.KFold(n_splits=10)

model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

X_for_train, X_for_test, Y_for_train, Y_for_test = train_test_split(X, Y, tes

my_test = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,0

# Train GradientBoostingClassifier: fitting the X and Y
gboost = GradientBoostingClassifier(n_estimators=100,learning_rate=1)
gboost.fit(X_for_train, Y_for_train)

#Predict the response for test dataset
Y_prediction = gboost.predict(X_test)
# test using the sample given: test the model using 3 new samples, and print
Y_prediction_test= gboost.predict(my_test)

#print the predictions
```

```
print("Prediction form my given test:",Y_prediction_test)
print("Prediction:",Y_prediction)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))

# In[48]:

#pentru al doilea set de parametri
num_trees = 300
kfold = model_selection.KFold(n_splits=10)

model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())

X_for_train, X_for_test, Y_for_train, Y_for_test = train_test_split(X, Y, tes

my_tests = [[0,7,14,29,1,0,1,1,2,0,2,0,0,1,3,3],[0,7,4,19,1,0,2,1,2,1,3,1,4,(

# Train GradientBoostingClassifier: fitting the X and Y
gboost = GradientBoostingClassifier(n_estimators=100,learning_rate=1)
gboost.fit(X_for_train, Y_for_train)

#Predict the response for test dataset
Y_prediction = gboost.predict(X_test)
# test using the sample given: test the model using 3 new samples, and print
Y_prediction_test= gboost.predict(my_test)

#print the predictions
print("Prediction form my given test:",Y_prediction_test)
print("Prediction:",Y_prediction)

#check/test 5 metrics for classification
#METRICA 1: Matricea de confuzie
#METRICA 2: Accuracy;
#METRICA 3: Precision
#METRICA 4: Recall
```

```
#METRICA 5: F1
from sklearn import metrics
matrix = confusion_matrix(Y_test, Y_prediction)
print("Confusion Matrix:")
print(matrix)
print("Accuracy score:",metrics.accuracy_score(Y_test, Y_prediction))
print("Precision score:",metrics.precision_score(Y_test, Y_prediction))
print("Recall score:",metrics.recall_score(Y_test, Y_prediction))
print("F1 score:",f1_score(Y_test, Y_prediction))
```

Bibliography

https://www.geeksforgeeks.org/support-vector-machine-in-machine-learning/
https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
https://www.educba.com/ensemble-methods-in-machine-learning/

Intelligent Systems Group