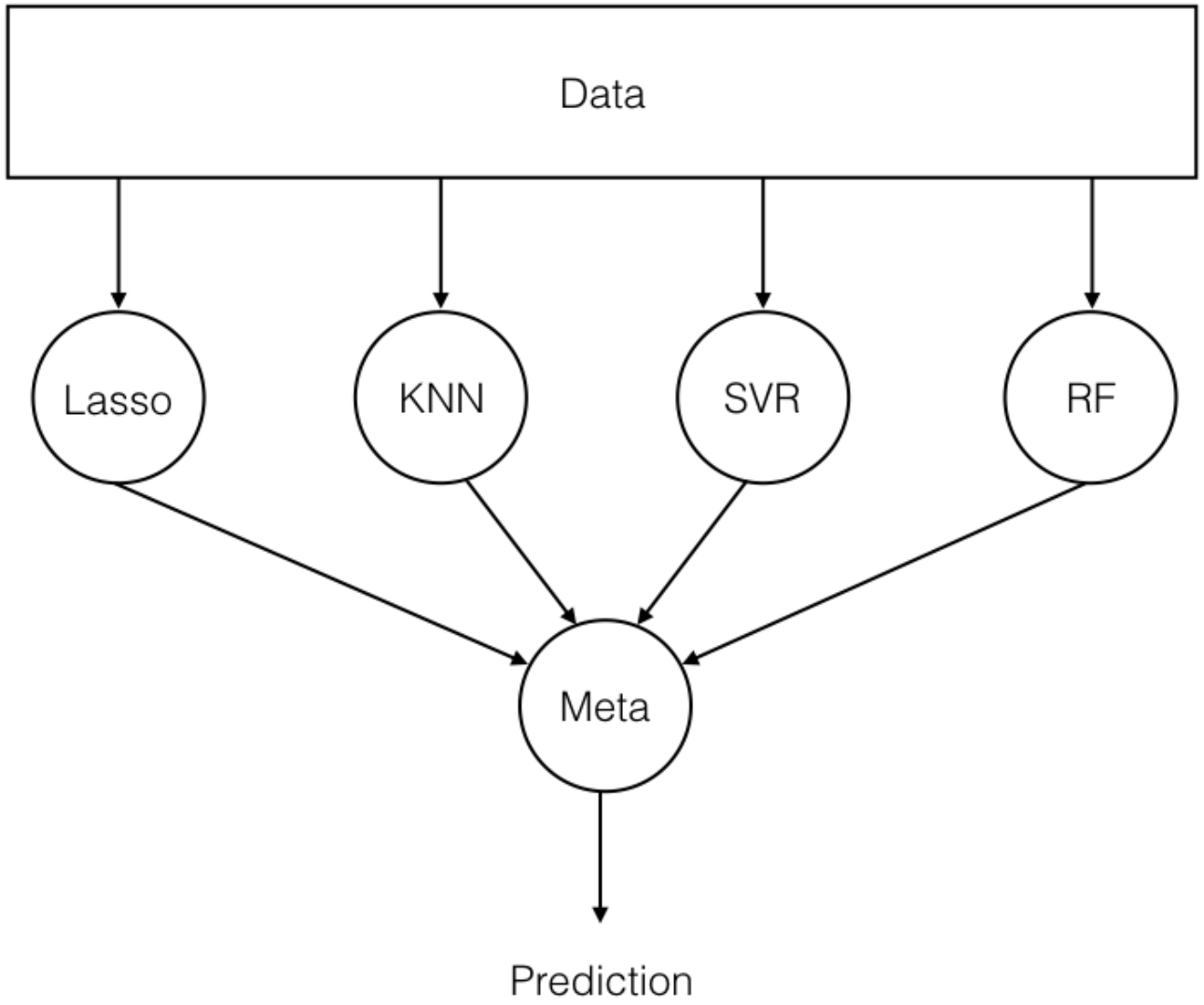
Ensemble learning

Desi in ML-ul traditional se lucreaza pentru a crea modele care, singure, sa poata sa rezolve cat mai bine un set de probleme, in practica se folosesc frecvent mai multe modele care, impreuna, au sanse mai mari de a da un rezultat mai bun decat oricare din cele care le compun.

Mai clar, un astfel ansamblu de modele de regresie poate arata precum:



Sursa imaginii (https://www.dataquest.io/blog/introduction-to-ensembles/)

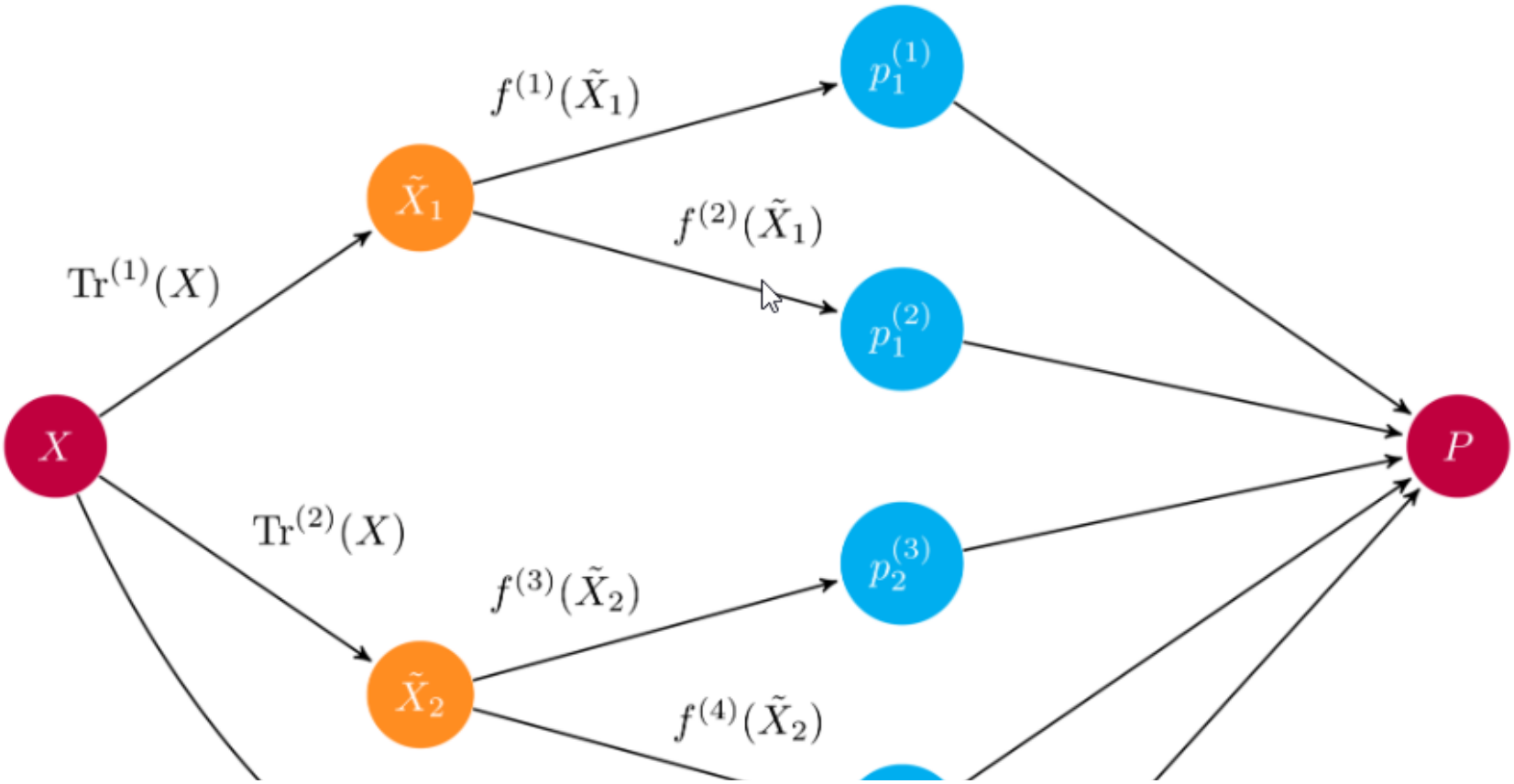
Exemplu din lumea reala in care se folosesc mai multe modele pentru a lua o decizie: pentru a determina daca merita investiti banii la bursa intr-o anumita companie, se iau in considerare:

1. Informatii despre activitatea companiei, perspective - afisate public sau din surse interne
2. Studiul evolutiei bursiere
3. Informatii despre companie provenind de la competitori
4. Cercetare de piata pe domeniul pe care compania activeaza - competitori, achizitii recente, preferintele clientilor
5. Social media - opinia populara despre companie sau diomeniul ei de activitate

Prin combinarea deciziilor de regula se ajunge la rezultate mai bune decat daca se considera o singura opinie de expert. In ML se pot combina mai multe modele de regresie sau clasificare pentru a produce raspunsul final. Majoritatea competitiilor Kagle din ultimii ani au fost castigate prin ansambluri de modele, uneori de dimensiuni foarte mari: KAGGLE ENSEMBLING GUIDE (https://mlwave.com/kaggle-ensembling-guide/).

Cateva situatii clare in care se poate folosi ensemble learning sunt:

1. Set de date prea mare: se poate intampla ca un singur model sa nu poata sa fie antrenat pe tot setul de date. Se pot obtine atunci modele diferite pe subseturi de date, ca in final deciziile lor sa fie agregate. Chiar daca se foloseste un acelasi model de baza, rezultatele sunt diferite pentru ca datele de instruire sunt diferite.
2. Set de date prea mic: se poate folosi metoda bootstrapping (https://en.wikipedia.org/wiki/Bootstrapping_(statistics)), prin care prin esantionari aleatoare se obtin subseturi de date diferite; se obtin deci modele diferite, la fel ca mai sus.
3. Set de date complex: pot exista cazuri de valori lipsa, sau tipuri de date pentru intrari ce nu pot fi manipulate de catre modele consacrate. Se pot obtine modele care lucreaza pe proiectii ale datelor, sau pe rezultatele unor fluxuri de procesare specifice, fiecare model vazand o parte din intreg.

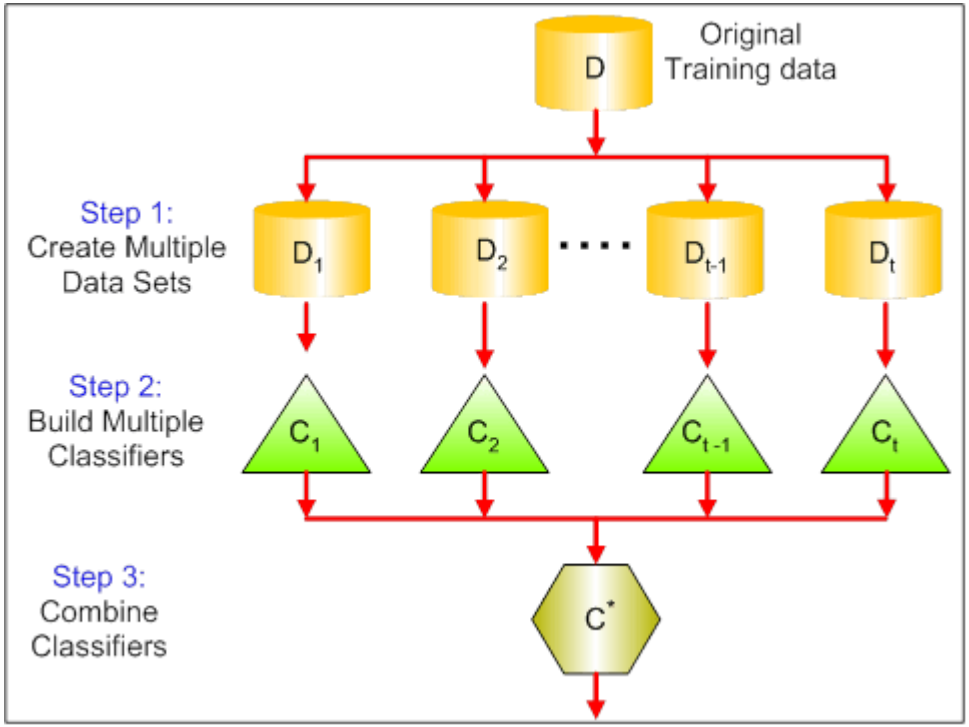


Metode de realizare de ansambluri

Bagging

Se bazeaza pe bootstrap aggregating (https://en.wikipedia.org/wiki/Bootstrapping_(statistics)) - se face o esantionare (extragere) cu intoarcere din populatia initiala; pentru un set de date, se extrag n elemente (posibil unele sa fie duplicate) din populatia initiala. Pe fiecare din cele t seturi de n elemente extrase ca mai sus se antreneaza cate un model. Cele t modele se agrega:

- pentru o problema de clasificare se poate considera clasa majoritar prezisa
- pentru o problema de regresie se ia media aritmetica a celor t modele de regresie



Sursa imaginii (https://www.datacamp.com/community/tutorials/ensemble-learning-python)

Boosting

Plecand de la modele de inferenta al caror comportament e chiar si doar un pic mai bun decat o ghicire aleatoare (weak learner), se poate obtine un ansamblu care sa obtina o acuratete arbitrar de mare. Ideea de baza este de a determina care din datele din setul de instruire sunt dificil de invatat, ca apoi sa se asigneze acestora o pondere mai mare. Modelul ajunge sa sa se concentreze mai mult pe invatarea datelor cu pondere mai mare (pondere data din cauza ca acele date sunt mai dificile). Modelul cel mai popular este AdaBoost (https://en.wikipedia.org/wiki/AdaBoost).

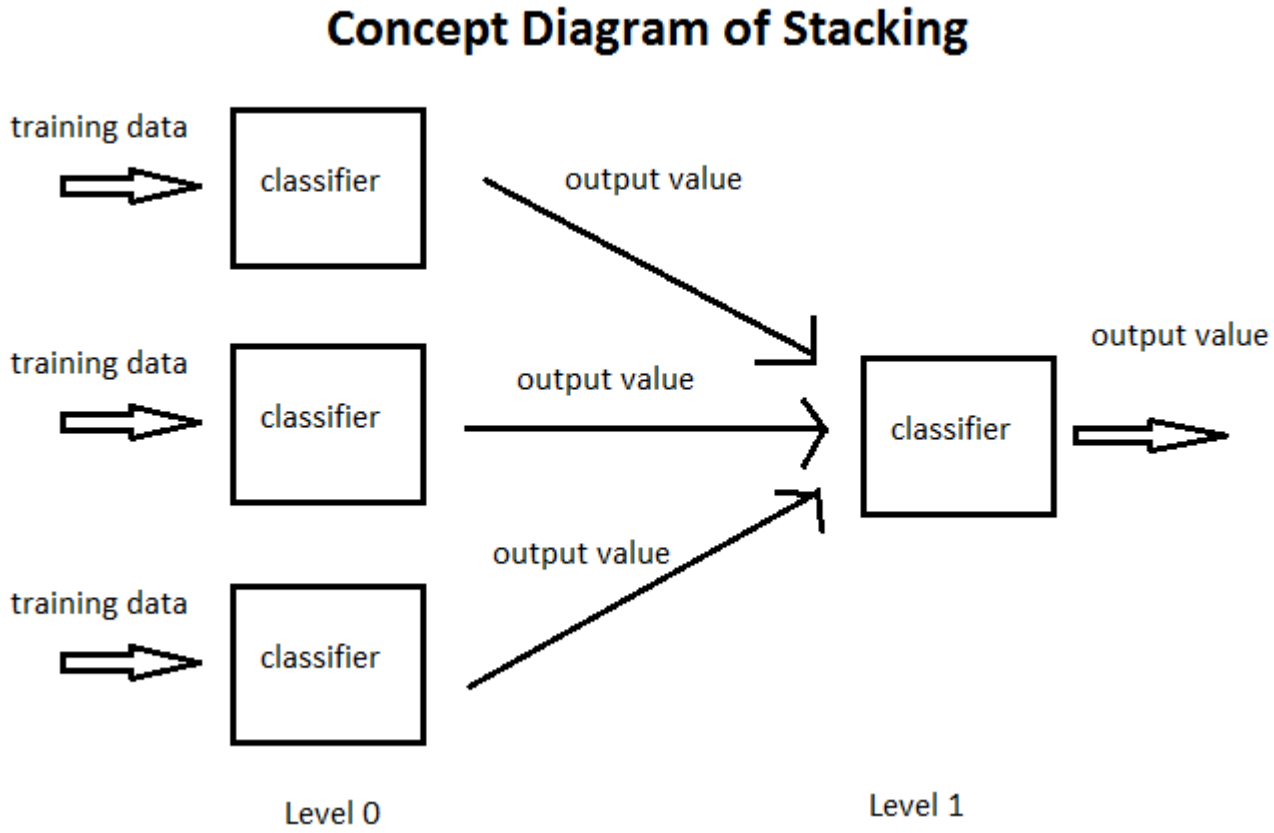
Algoritmul AdaBoost are doua componente majore: determinarea ponderilor datelor din setul de instruire si calculul coeficientilor modelelor rezultate. Algoritmul pentru o problema de clasificare in doua clase este schitat in figura de mai jos:

Input: Data set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
Base learning algorithm L;
Number of learning rounds T.

Process:

Stacking

O modalitate simpla de agregare a "opiniilor" date de catre fiecare model dintr-un ansamblu este votarea sau calcularea mediei iesirilor acestor modele. O varianta mai elaborata este ca un model suplimentar sa invete cum sa agrege "opiniile" date de catre modelele din ansamblu, inlocuind o agregare simpla cu una invatata. Modelele care compun ansamblul sunt de nivel 0, modelul care invata sa pondereze iesirile date de cele de nivel 0 este de nivel 1:



Sursa (https://medium.com/@gurucharan_33981/stacking-a-super-learning-technique-dbed06b1156d)

O schita a pasilor este data mai jos:

Algorithm Stacking

Input: training data $D = \{x_i, y_i\}_{i=1}^{m}$
Ouput: ensemble classifier H
Step 1: learn base-level classifiers
for $t = 1$ to T **do**
    learn h_t based on D
end for
Step 2: construct new data set of predictions
for $i = 1$ to m **do**
    $D_h = \{x'_i, y_i\}$, where $x'_i = \{h_1(x_i), ..., h_T(x_i)\}$
end for
Step 3: learn a meta-classifier
learn H based on D_h
return H

Sursa (https://blog.statsbot.co/ensemble-learning-d1dcd548e936)

Se pot, desigur, adauga si alte niveluri de modele de agregare peste nivelul 0.

Exemplu

Incarcarea si preprocesarea datelor

Se exemplifica folosirea unui ansamblu de clasificatori pentru problema Breast Cancer Wisconsin (Original) Data Set (https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)). Datele sunt descarcate local in directorul `./data`

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
```

In [2]:
```
path = './data/breast-cancer-wisconsin.csv'
data = pd.read_csv(path)
data.head()
```

Out[2]:

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

Coloana 'Sample code number' poate fi inlaturata, deoarece nu poarta informatie utila.:

In [3]:
```
data.drop(['Sample code number'],axis = 1, inplace = True)
data.head()
```

Out[3]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	5	1	1	1	2	1	3	1	1	2
1	5	4	4	5	7	10	3	2	1	2
2	3	1	1	1	2	2	3	1	1	2
3	6	8	8	1	3	4	3	7	1	2
4	4	1	1	3	2	1	3	1	1	2

Obtinem niste statistici despre date:

In [4]:
```
data.describe()
```

Out[4]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bland Chromatin	Normal Nucleoli	Mitoses	Class
count	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000
mean	4.417740	3.134478	3.207439	2.806867	3.216023	3.437768	2.866953	1.589413	2.689557
std	2.815741	3.051459	2.971913	2.855379	2.214300	2.438364	3.053634	1.715078	0.951273
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	2.000000
25%	2.000000	1.000000	1.000000	1.000000	2.000000	2.000000	1.000000	1.000000	2.000000
50%	4.000000	1.000000	1.000000	1.000000	2.000000	3.000000	1.000000	1.000000	2.000000
75%	6.000000	5.000000	5.000000	4.000000	4.000000	5.000000	4.000000	1.000000	4.000000
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	4.000000

Remarcam ca atributul 'Bare nuclei' lipseste din descriere, ceea ce inseamna ca nu toate valorile de pe coloana sunt numerice.

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
Clump Thickness               699 non-null int64
Uniformity of Cell Size       699 non-null int64
Uniformity of Cell Shape      699 non-null int64
Marginal Adhesion             699 non-null int64
Single Epithelial Cell Size   699 non-null int64
Bare Nuclei                   699 non-null object
Bland Chromatin               699 non-null int64
Normal Nucleoli               699 non-null int64
Mitoses                       699 non-null int64
Class                         699 non-null int64
dtypes: int64(9), object(1)
memory usage: 54.7+ KB
```

Remarcam ca toate atributele sunt numerice, mai putin 'Bare nuclei'. Acest lucru se datoreaza faptului ca pe coloana numita se gasesc valori nule:

In [6]:
```
data['Bare Nuclei'].unique()
```

Out[6]:
```
array(['1', '10', '2', '4', '3', '9', '7', '?', '5', '8', '6'],
      dtype=object)
```

Inlocuim aceste valori '?' cu 'nan', pentru ca transformarea din string in floating point sa functioneze usor:

In [7]:
```
data['Bare Nuclei'].replace('?', 'nan', inplace = True)
data['Bare Nuclei'] = data['Bare Nuclei'].astype('float64')
```

In [8]:
```
data['Bare Nuclei'].describe()
```

Out[8]:
```
count    683.000000
mean       3.544656
std        3.643857
min        1.000000
25%        1.000000
50%        1.000000
75%        6.000000
max       10.000000
Name: Bare Nuclei, dtype: float64
```

Vom face missing value imputation, inlocuind valorile lipsa cu media lor:

In [9]:
```
values = data.values # pentru a le da algoritmului de missing value imputation

imputer = SimpleImputer()
imputedData = imputer.fit_transform(values)
```

Scalele atributelor sunt diferite, lucru care dauneaza multor modele de clasificare. Vom face o scalare in intervalul $[0, 1]$:

In [10]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
normalizedData = scaler.fit_transform(imputedData)
```

Model singular

Utilizam un model de clasificare singular - arbore de decizie - si observam care suntperformantele lui:

In [11]:
```
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [12]:
```
# Separa datele de intrare de etichete
X = normalizedData[:,0:-1]
Y = normalizedData[:,-1]
```

In [13]:
```
kfold = model_selection.KFold(n_splits=10, random_state=7)
cart = DecisionTreeClassifier()
results = model_selection.cross_val_score(cart, X, Y, cv=kfold)
print(f'Acuratetea modelului singular: {results.mean()}')
```

```
Acuratetea modelului singular: 0.9313871635610764
```

Ansamblu prin bagging

In [17]:
```
num_trees = 20
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_state=7)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(f'Acuratetea ansamblului obtinut prin bagging: {results.mean()}')
```

```
Acuratetea ansamblului obtinut prin bagging: 0.9571221532091098
```

Ansamblu prin AdaBoost

In [15]:
```
from sklearn.ensemble import AdaBoostClassifier
seed = 7
num_trees = 70
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(f'Acuratetea ansamblului obtinut prin AdaBoost: {results.mean()}')
```

```
Acuratetea ansamblului obtinut prin AdaBoost: 0.9599792960662527
```

Ansamblu prin votare

Pregatim mai multe modele diferite. Rezultatele acestora sunt agregate prin votare.

In [16]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

kfold = model_selection.KFold(n_splits=10, random_state=seed)
# create the sub models
estimators = []
model1 = LogisticRegression(solver='lbfgs')
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC(gamma='auto')
estimators.append(('svm', model3))
# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(f'Acuratetea ansamblului obtinut prin votare: {results.mean()}')
```

```
Acuratetea ansamblului obtinut prin votare: 0.9628571428571429
```

De retinut

1. Nu este adevarat ca *intotdeauna* ensemble learning functioneaza mai bine. Frecvent insa, acest lucru e adevarat, dar fara a putea spune apriori ce strategie de ensemble e cea mai buna.
2. Daca modelele au varianta mare (aka au tendinta de a face overfit), atunci bagging e mai indicat. Daca modele sunt biased (fac underfitting), atunci boosting e mai indicat.
3. Nu orice strategie de asamblare se potriveste cu orice tip de model: modele biased (care fac underfitting) impreuna cu bagging nu functioneaza in practica prea bine.

In [ ]: