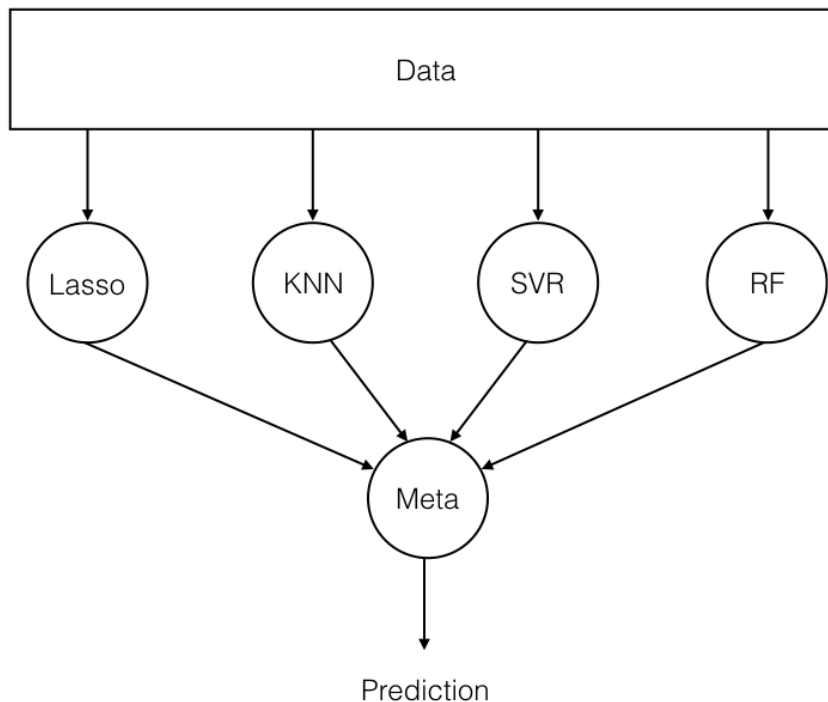


## Ensemble learning

Deși în ML-ul tradițional se lucrează pentru a crea modele care, singure, să poată să rezolve cât mai bine un set de probleme, în practică se folosesc frecvent mai multe modele care, împreună, au șanse mai mari de a da un rezultat mai bun decât oricare din cele care le compun.

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



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

Exemplu din lumea reală în care se folosesc mai multe modele pentru a lua o decizie: pentru a determina dacă merita investiți banii la bursă într-o anumită companie, se iau în considerare:

1. Informații despre activitatea companiei, perspective - afișate public sau din surse interne
2. Studiul evoluției bursiere
3. Informații despre companie provenind de la competitori
4. Cercetare de piață pe domeniul pe care compania activează - competitori, achiziții recente, preferințele clienților
5. Social media - opinia populară despre companie sau domeniul ei de activitate

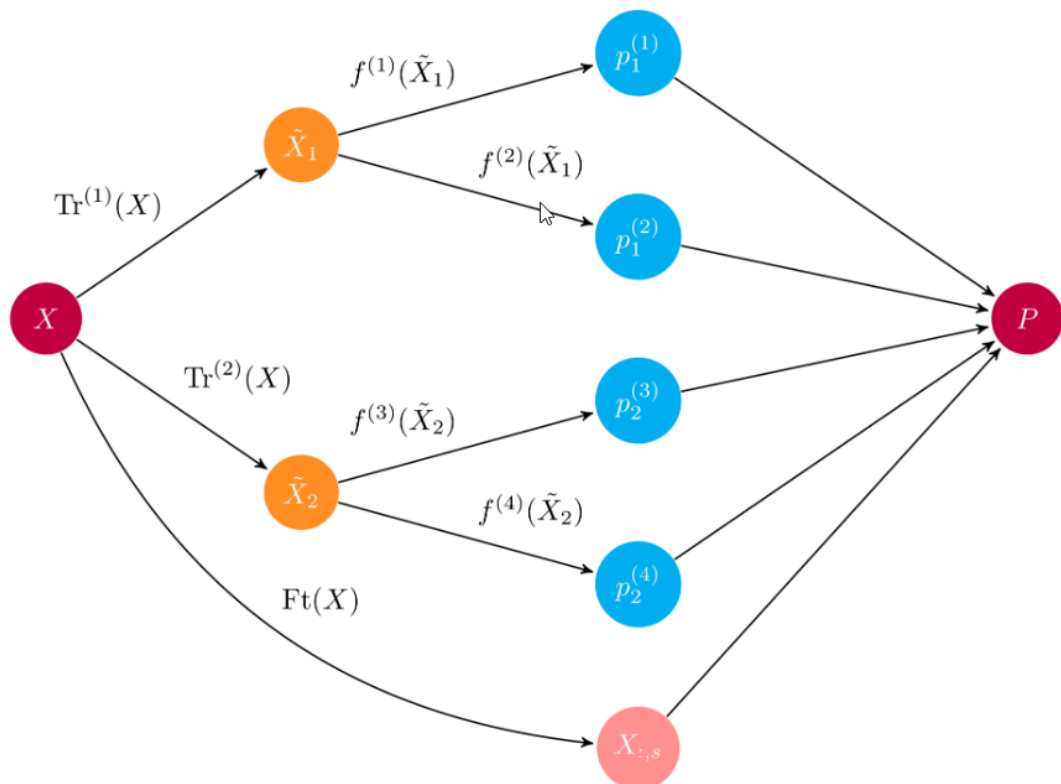
Prin combinarea deciziilor de regulă se ajunge la rezultate mai bune decât dacă se consideră o singură opinie de expert. În ML se pot combina mai multe modele de regresie sau clasificare pentru a produce răspunsul final. Majoritatea competițiilor Kaggle din ultimii ani au fost câștigate prin ansambluri de modele, uneori de dimensiuni foarte mari: [KAGGLE ENSEMBLING GUIDE \(https://mlwave.com/kaggle-ensembling-guide/\)](https://mlwave.com/kaggle-ensembling-guide/).



Efectul succesului combinatiilor se explica prin faptul ca fiecare model din ansamblu are intrinsec o presupunere asupra modului in care variabila dependenta (de iesire) este determinata de variabilele independente (de intrare). In plus, ele pot excela pe anumite de subseturi de date (exemplu: sub-populatii de clienti) dar sa aiba performante mai mici pe alte subseturi; sau, votarea majoritara ori ponderarea valorilor de iesire poate sa duca la atenuarea valorilor extreme produse de un singur model; sau, se poate urmari un melanj inttre modele cae sunt cu acuratete mare - dar computational intensive si lente - si unele mai putin precise, dar rapide.

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)) ([https://en.wikipedia.org/wiki/Bootstrapping\\_\(statistics\)](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 complexe: 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.



4. Masurarea gradului de incredere pentru predictia rezultata din ansamblu: daca din 5 modele de clasificare 4 decid ca intrarea este de o aceeaasi clasa, avem o indicatie de incredere in rezultatul clasificarii.

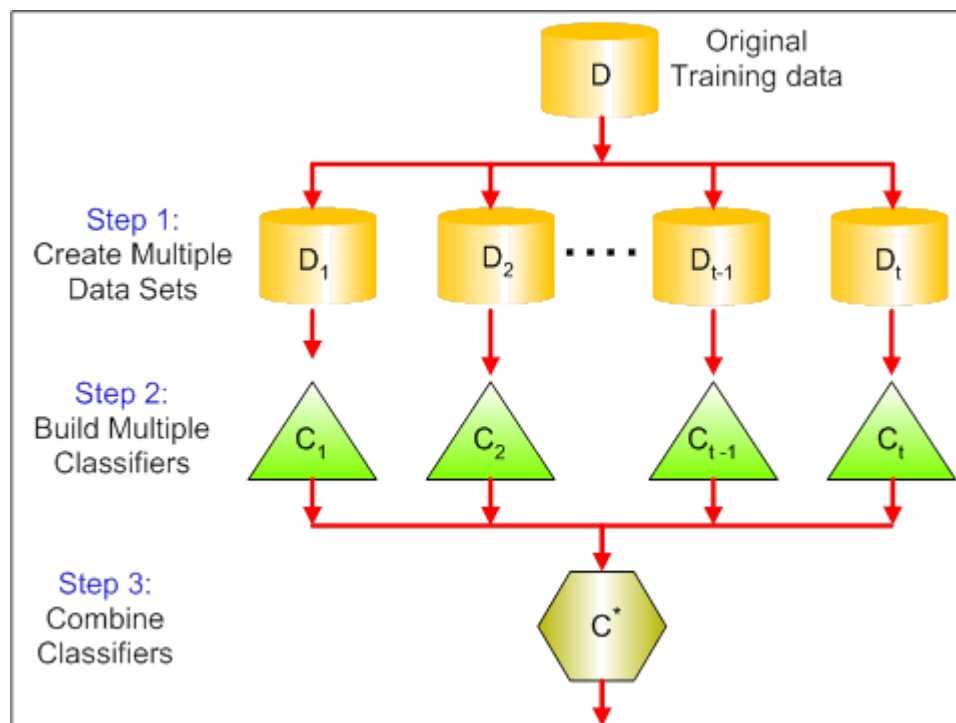


## 1 Metode de realizare de ansambluri

## 1.1 Bagging

Se bazează pe [bootstrap aggregating](https://en.wikipedia.org/wiki/Bootstrapping_(statistics)) ([https://en.wikipedia.org/wiki/Bootstrapping\\_\(statistics\)](https://en.wikipedia.org/wiki/Bootstrapping_(statistics))) - se face o esantionare (extragere) cu întoarcere din populația inițială; pentru un set de date, se extrag  $n$  elemente (posibil unele să fie duplicate) din populația inițială. Pe fiecare din cele  $t$  seturi de  $n$  elemente extrase ca mai sus se antrenează câte un model. Cele  $t$  modele se agregă:

- pentru o problemă de clasificare se poate considera clasa majoritară prezisă
- pentru o problemă de regresie se ia media aritmetică a celor  $t$  modele de regresie



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

## 1.2 Boosting

Plecând de la modele de inferență al căror comportament e chiar și doar un pic mai bun decât o ghicire aleatoare (weak learner), se poate obține un ansamblu care să obțină o acuratețe arbitrar de mare. Ideea de bază este de a determina care din datele din setul de instruire sunt dificil de învățat, ca apoi să se asigneze acestora o pondere mai mare. Modelul ajunge să se concentreze mai mult pe învățarea datelor cu pondere mai mare (pondere dată din cauza că acele date sunt mai dificile). Modelul cel mai popular este [AdaBoost](https://en.wikipedia.org/wiki/AdaBoost) (<https://en.wikipedia.org/wiki/AdaBoost>).

Algoritmul AdaBoost are două componente majore: determinarea ponderilor datelor din setul de instruire și calculul coeficienților modelelor rezultate. Algoritmul pentru o problemă de clasificare în două clase este schitat în figura de mai jos:

---

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

**Process:**

1.  $\mathcal{D}_1(i) = 1/m$ .      % Initialize the weight distribution
2. **for**  $t = 1, \dots, T$ :
3.     $h_t = L(D, \mathcal{D}_t)$ ;    % Train a learner  $h_t$  from  $D$  using distribution  $\mathcal{D}_t$
4.     $\epsilon_t = \Pr_{\mathbf{x} \sim \mathcal{D}_t, y} [h_t(\mathbf{x}) \neq y]$ ;    % Measure the error of  $h_t$
5.    **if**  $\epsilon_t > 0.5$  **then break**
6.     $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$ ;    % Determine the weight of  $h_t$
7.     $\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(\mathbf{x}_i) = y_i \\ \exp(\alpha_t) & \text{if } h_t(\mathbf{x}_i) \neq y_i \end{cases}$   
     $\frac{\mathcal{D}_t(i) \exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$     % Update the distribution, where  
    %  $Z_t$  is a normalization factor which  
    % enables  $\mathcal{D}_{t+1}$  to be distribution
8.    **end**

**Output:**  $H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(\mathbf{x}) \right)$

---

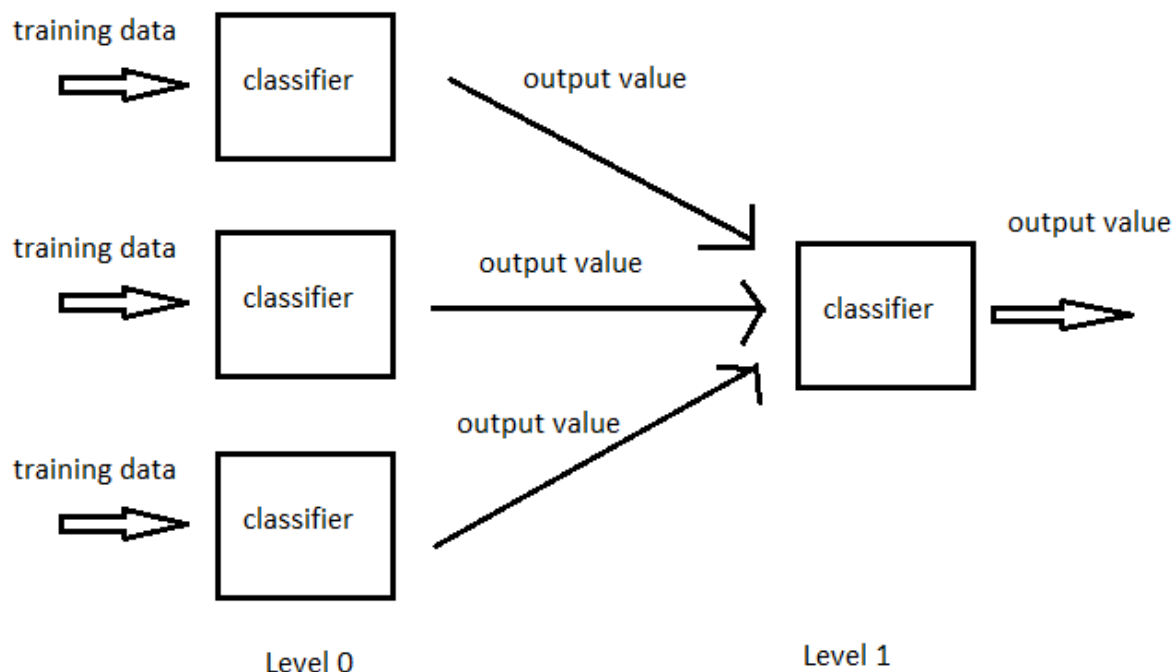
[Sursa \(https://www.amazon.com/Algorithms-Mining-Chapman-Knowledge-Discovery/dp/1420089641/ref=sr\\_1\\_1?crid=13FRIM1B488BK&keywords=the+top+ten+algorithms+in+data+mining&qid=1559746296&s=ga1\)](https://www.amazon.com/Algorithms-Mining-Chapman-Knowledge-Discovery/dp/1420089641/ref=sr_1_1?crid=13FRIM1B488BK&keywords=the+top+ten+algorithms+in+data+mining&qid=1559746296&s=ga1)



### 1.3 Stacking

O modalitate simplă de agregare a "opiniilor" date de către fiecare model dintr-un ansamblu este votarea sau calcularea mediei ieşirilor acestor modele. O variantă mai elaborată este ca un model suplimentar să înveţe cum să agreghe "opiniile" date de către modelele din ansamblu, înlocuind o agregare simplă cu una învăţată. Modelele care compun ansamblul sunt de nivel 0, modelul care învaţă să pondereze ieşirile date de cele de nivel 0 este de nivel 1:

## Concept Diagram of Stacking



[Sursa \(https://medium.com/@gurucharan\\_33981/stacking-a-super-learning-technique-dbed06b1156d\)](https://medium.com/@gurucharan_33981/stacking-a-super-learning-technique-dbed06b1156d)

O schita a pasilor este data mai jos:

Algorithm	Stacking
1:	Input: training data $D = \{x_i, y_i\}_{i=1}^m$
2:	Output: ensemble classifier $H$
3:	<i>Step 1: learn base-level classifiers</i>
4:	for $t = 1$ to $T$ do
5:	learn $h_t$ based on $D$
6:	end for
7:	<i>Step 2: construct new data set of predictions</i>
8:	for $i = 1$ to $m$ do
9:	$D_h = \{x'_i, y_i\}$ , where $x'_i = \{h_1(x_i), \dots, h_T(x_i)\}$
10:	end for
11:	<i>Step 3: learn a meta-classifier</i>
12:	learn $H$ based on $D_h$
13:	return $H$

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

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

## ▼ 2 Exemplu

### ▼ 2.1 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)) ([https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+\(original\)](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
```

executed in 11.8s, finished 22:39:40 2021-05-16

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

executed in 50ms, finished 22:39:40 2021-05-16

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	N
0	1000025	5	1	1	1	2	1	3	1	
1	1002945	5	4	4	5	7	10	3	2	
2	1015425	3	1	1	1	2	2	3	1	
3	1016277	6	8	8	1	3	4	3	7	
4	1017023	4	1	1	3	2	1	3	1	

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()`

executed in 17ms, finished 22:39:40 2021-05-16

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	C
0	5	1	1	1	2	1	3	1	1	
1	5	4	4	5	7	10	3	2	1	
2	3	1	1	1	2	2	3	1	1	
3	6	8	8	1	3	4	3	7	1	
4	4	1	1	3	2	1	3	1	1	



Obtinem niste statistici despre date:

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

executed in 55ms, finished 22:39:40 2021-05-16

Out[4]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bland Chromatin	Normal Nucleoli	Mit
<b>count</b>	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.00
<b>mean</b>	4.417740	3.134478	3.207439	2.806867	3.216023	3.437768	2.866953	1.58
<b>std</b>	2.815741	3.051459	2.971913	2.855379	2.214300	2.438364	3.053634	1.71
<b>min</b>	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00
<b>25%</b>	2.000000	1.000000	1.000000	1.000000	2.000000	2.000000	1.000000	1.00
<b>50%</b>	4.000000	1.000000	1.000000	1.000000	2.000000	3.000000	1.000000	1.00
<b>75%</b>	6.000000	5.000000	5.000000	4.000000	4.000000	5.000000	4.000000	1.00
<b>max</b>	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.00



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

In [18]:

data.info()

executed in 27ms, finished 22:53:27 2021-05-16

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Clump Thickness                       699 non-null    int64
1   Uniformity of Cell Size               699 non-null    int64
2   Uniformity of Cell Shape              699 non-null    int64
3   Marginal Adhesion                    699 non-null    int64
4   Single Epithelial Cell Size           699 non-null    int64
5   Bare Nuclei                          683 non-null    float64
6   Bland Chromatin                      699 non-null    int64
7   Normal Nucleoli                      699 non-null    int64
8   Mitoses                             699 non-null    int64
9   Class                               699 non-null    int64
dtypes: float64(1), int64(9)
memory usage: 54.7 KB
```

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

In [19]: data['Bare Nuclei'].unique()

executed in 13ms, finished 22:53:42 2021-05-16

Out[19]: array([ 1., 10., 2., 4., 3., 9., 7., nan, 5., 8., 6.])

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

executed in 12ms, finished 22:39:40 2021-05-16

```
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 [23]: from sklearn.pipeline import Pipeline

pipeline = Pipeline([('imputer', SimpleImputer()), ('scaler', MinMaxScaler())])

values = data.values # pentru a le da algoritmului de missing value imputation
```

executed in 7ms, finished 22:55:27 2021-05-16



## 2.2 Model singular

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

```
In [24]: from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
```

executed in 5ms, finished 22:55:44 2021-05-16

```
In [25]: # Separa datele de intrare de etichete
X = values[:,0:-1]
Y = values[:, -1]
```

executed in 4ms, finished 22:55:46 2021-05-16

```
In [26]: seed = 7
```

executed in 5ms, finished 22:55:50 2021-05-16

```
In [27]: kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
cart = DecisionTreeClassifier()
pipeline = Pipeline([('imputer', SimpleImputer()), ('scaler', MinMaxScaler())],
results = model_selection.cross_val_score(pipeline, X, Y, cv=kfold)
print(f'Acuratetea modelului singular: {results.mean()}')
```

executed in 70ms, finished 22:55:54 2021-05-16

Acuratetea modelului singular: 0.9499171842650103

## 2.3 Ansamblu prin bagging

```
In [28]: from sklearn.ensemble import BaggingClassifier
```

executed in 4ms, finished 22:56:03 2021-05-16

```
In [29]: num_trees = 20
model = BaggingClassifier(base_estimator=pipeline, n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(f'Acuratetea ansamblului obtinut prin bagging: {results.mean()}')
```

executed in 814ms, finished 22:56:05 2021-05-16

Acuratetea ansamblului obtinut prin bagging: 0.959896480331263

## 2.4 Ansamblu prin AdaBoost

```
In [40]: from sklearn.ensemble import AdaBoostClassifier
num_trees = 100
kfold = model_selection.KFold(n_splits=10, random_state=seed, shuffle=True)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(pipeline, X, Y, cv=kfold)
print(f'Acuratetea ansamblului obtinut prin AdaBoost: {results.mean()}')
```

executed in 60ms, finished 22:57:16 2021-05-16

Acuratetea ansamblului obtinut prin AdaBoost: 0.9484886128364389

## 2.5 Ansamblu prin votare

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

```
In [42]: 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, shuffle=True)
# creare modele de nivel 0
estimators = []
model1 = LogisticRegression(solver='lbfgs')
pipeline1 = Pipeline([('imputer', SimpleImputer()), ('scaler', MinMaxScaler())],
estimators.append(('logistic', pipeline1))

model2 = DecisionTreeClassifier()
pipeline2 = Pipeline([('imputer', SimpleImputer()), ('decision_tree', model2)])
estimators.append(('cart', pipeline2))

model3 = SVC(gamma='auto')
pipeline3 = Pipeline([('imputer', SimpleImputer()), ('scaler', MinMaxScaler())],
estimators.append(('svm', pipeline3))

# creare ansamblu de tip stack
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(f'Acuratetea ansamblului obtinut prin votare: {results.mean()}')
```

executed in 308ms, finished 22:58:03 2021-05-16

Acuratetea ansamblului obtinut prin votare: 0.964223602484472

## 3 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 [ ]: