

Gépi Tanulás Előadás 1

Felügyelt Tanulás: Bevezetés, K Legközelebbi Szomszédok, Véletlen Erdők

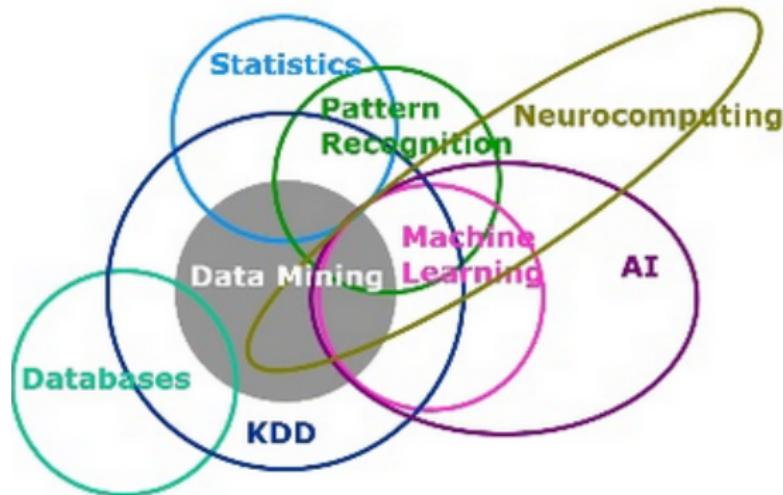
Milacski Zoltán Ádám¹

¹Eötvös Loránd Tudományegyetem
Programozáselmélet és Szoftvertechnológiai Tanszék
srph25@gmail.com

2018. május 3.



MACHINE LEARNING IS...



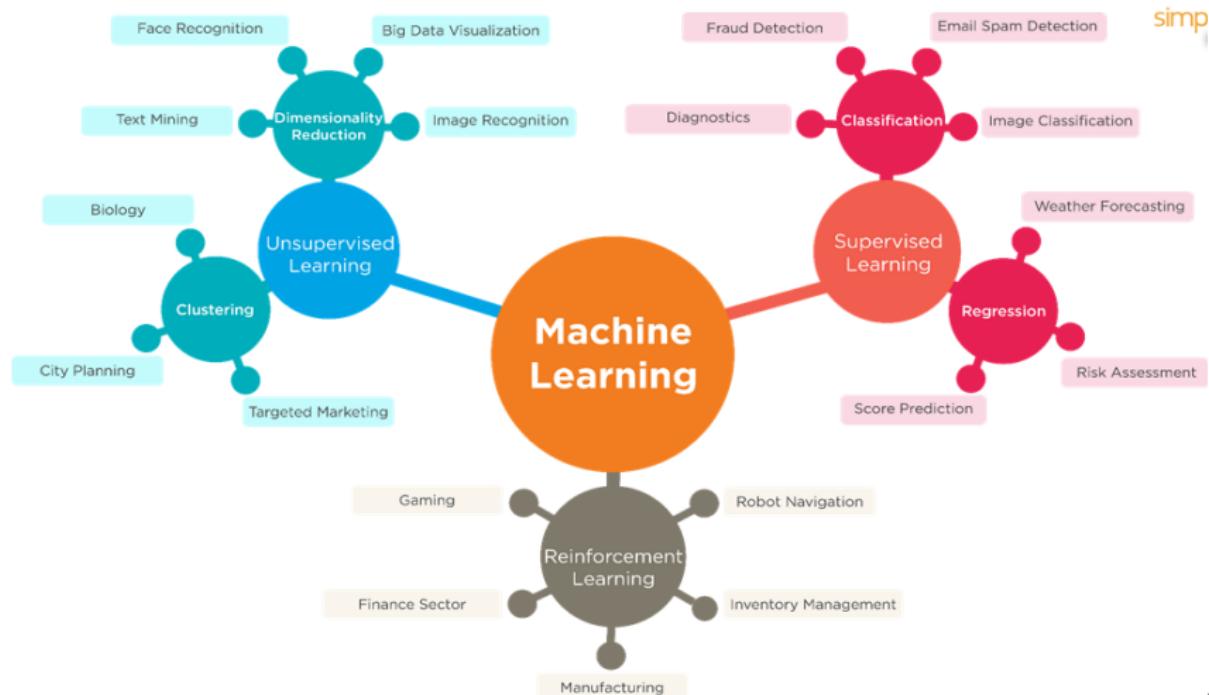
“Field of study that gives computers the ability to learn **without being explicitly programmed.**”

~ Arthur Samuel, 1959

- AI: Intelligens gépek, amik emberszerűen gondolkodnak és cselekednek.
- ML: Rendszerek amik előre programozás nélkül tanulnak.
YCombinator szerinti következő forradalmi technológia (internet, mobil után).

Gépi Tanulás

Mi ez és mi nem



ML módszerek tipikusan jól skálázódnak a feladat méretében (zavarbaejtően párhuzamos tenzor műveletek GPU-n).



Gépi Tanulás

Mi ez és mi nem

► Pure Reinforcement Learning (cherry)

- The machine predicts a scalar reward given once in a while.

► A few bits for some samples

► Supervised Learning (icing)

- The machine predicts a category or a few numbers for each input
- 10→10,000 bits per sample

► Unsupervised/Predictive Learning (génoise)

- The machine predicts any part of its input for any observed part.
- Predicts future frames in videos
- Millions of bits per sample



► Unsupervised Learning is the Dark Matter (or Dark Energy) of AI

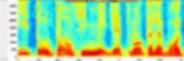
- RL: Megy, ha a jutalom (szám) adott tanuláshoz. Humán tudást reprodukál.
- SL: Megy, ha az output (vektor) adott tanuláshoz. Humán tudást reprodukál.
- UL: Megoldatlan, sok nyitott kutatási kérdés (tenzor). Ember nélkül tanul.



Felügyelt Tanulás

Szemléletesen

Felügyelt Tanulás: tanuljuk meg az input-ból output-ba menő leképezést, $x \mapsto y$

Input	Output
Pixels: 	"lion"
Audio: 	"see at tuhl res taur aun ts"
<query, doc>	P(click on doc)
"Hello, how are you?"	"Bonjour, comment allez-vous?"
Pixels: 	"A close up of a small child holding a stuffed animal"

Véges számú tanító példapárból: $(x_n, y_n), n = 1, \dots, N$.

Úgy, hogy a leképezés pontos legyen nem látott párokra is!



Felügyelt Tanulás

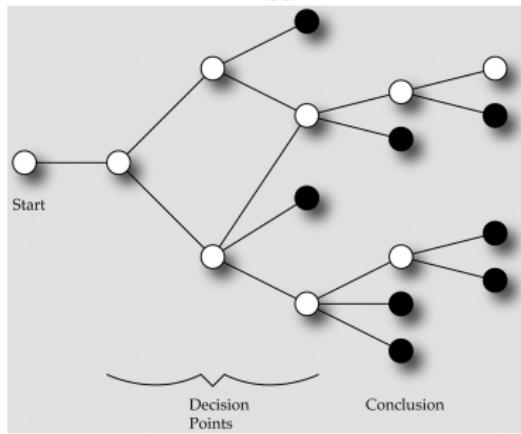
Szemléletesen

- Első gondolat: csinálunk leképezést elágazással, azaz küszöböljünk le egy változót:

if feature>threshold then .. else ..

if $\varphi(x) > t$ then return \hat{y}_1 else return \hat{y}_2

- Egymásba ágyazott elágazások több változóval: esetek száma exponenciálisan nő, nagyon nehéz kézzel megtervezni és leprogramozni.



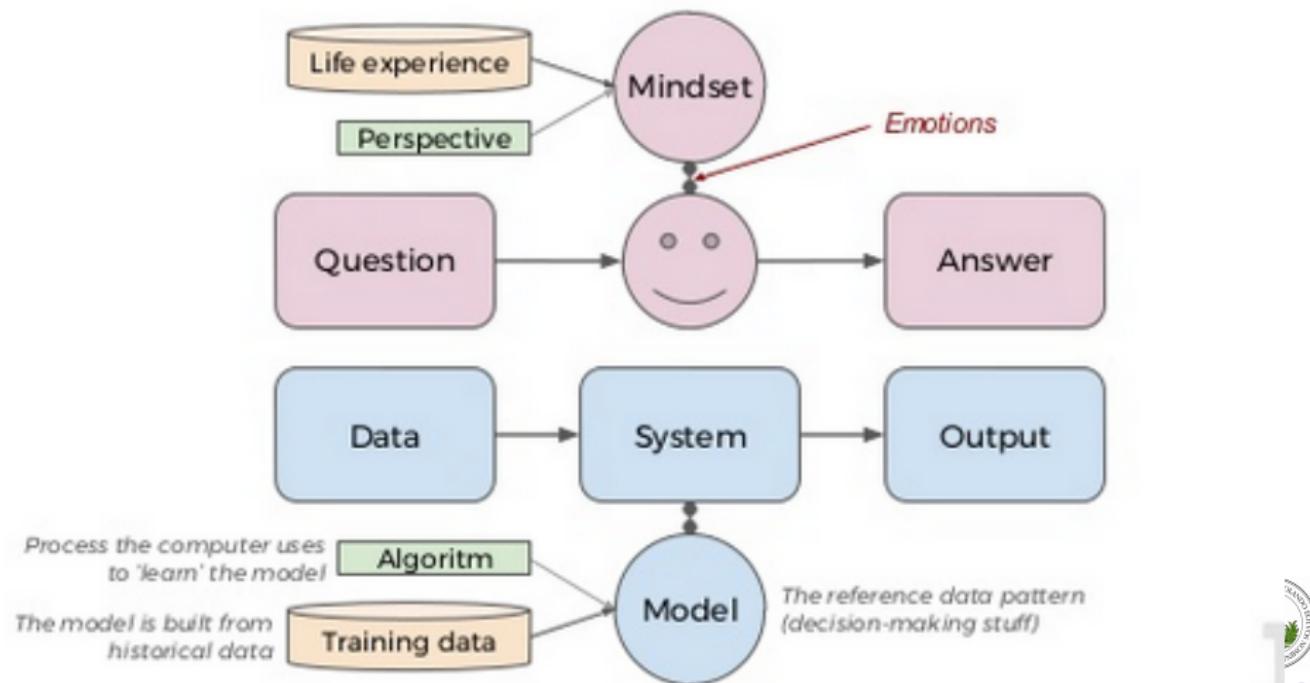
Her: I develop artificial intelligence for chatbots

Me: [trying to think of something to impress her] I write nested if blocks too



- Megoldás: tanuljuk meg a $\varphi(x)$ változókat és a t küszöböket az adatból!
 - Emlékeztetőül: ML egy szoftver, ami önmagát írja...

SIMILAR TO HOW WE LEARN



Felügyelt Tanulás

Szemléletesen

Algorithm: Learn Something

Input: Data, Mental Model

Output: Updated Mental Model

1 while *Mental Model Makes Bad Predictions* do

2 make a guess at answer

3 measure error

4 if *error is acceptably small* then

5 break

6 else

7 propose model adjustment

8 Mental Model \leftarrow Mental Model + adjustment

Algorithm: Gradient Descent

Input: Y, Θ, X, α , tolerance, max iterations

Output: Θ

1 for $i = 0; i < \text{max iterations}; i++$ do

2 current cost = $\text{Cost}(Y, X, \Theta)$

3 if *current cost < tolerance* then

4 break

5 else

6 gradient = $\text{Gradient}(Y, X, \Theta)$

7 $\theta_j \leftarrow \theta_j - \alpha \cdot \text{gradient}$



Felügyelt Tanulás

Formálisan: Optimalizációs Feladat

- Adottak az $(x_n, y_n), n = 1, \dots, N$ tanítópárok, keressük az optimális θ^* paraméterbeállítást az $f(\theta, \cdot)$ paraméteres leképezéshez úgy, hogy $f(\theta^*, x_n) \approx y_n$ (közelítsük az $x_n \mapsto y_n$ leképezést):

$$\min_{\theta} L(\theta, x_n, y_n) = \frac{1}{N} \sum_{n=1}^N \ell\left(\underbrace{f(\theta, x_n)}_{\hat{y}_n}, y_n\right),$$

ahol $\ell(\hat{y}_n, y_n)$ a hibafüggvény és θ a paramétervektor.

Becslés ($f(\theta^*, x_n)$ kiszámítása adott θ^* -ra) nagyon gyors!

Tanítás (minimalizáló θ^* megkeresése) nagyon lassú!

Általában: nemkonvex optimalizálási feladat, NP-nehéz a θ^* globális optimumot megtalálni.

- Jó hírek: működik, de csak ha...

- ... N elég nagy! Az y_n -ek összegyűjtése drága humán munka... (UL?),
- ... $\ell(\hat{y}_n, y_n)$, $f(\theta, x_n)$ és az optimalizálási módszer megfelelően vannak megválasztva! Nehéz humán munka, sok kísérlet... (UL?),
- ... θ közel van θ^* -hoz! Egyelőre nem tudjuk bizonyítani, de megy, így feltehetően igaz... (θ^* egyébként is túl mohó és túltanulásra vezet...).



Felügyelt Tanulás

Formálisan: Hibafüggvény

Hibafüggvény: $\ell: \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$

- Felügyelt:

- Regresszió ($y \in \mathbb{R}^m$):

négyzetes hiba: $\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_2^2$

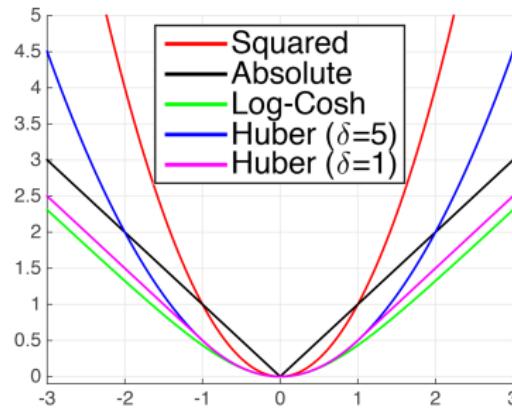
abszolút hiba: $\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_1$

- Osztályozás ($y \in \{0, 1\}^m$):

kategorikus kereszt-entrópia: $\ell(\hat{y}_n, y_n) = - \sum_{i=1}^m y_{n,i} \log \hat{y}_{n,i}$

- Felügyeletlen: amikor y_n -t nem adja meg az ember, jobb híján legyen $y_n = x_n$

- $\ell(\hat{x}_n, x_n) = \|\hat{x}_n - x_n\|_2^2$



Felügyelt Tanulás

Formálisan: Paraméteres leképezés és Optimalizálási módszer

Paraméteres leképezés: hármat fogunk látni hamarosan.

- K Legközelebbi Szomszédok: részletesen **ma**,
- Véletlen Erdők (Döntési Fák): részletesen **ma**,
- Mély Neurális Hálózatok: részletesen **jövő héten**.

Optimalizálási módszer: paraméteres leképezések köré más, annak jellegéből adódik (pl. ha diszkrét, akkor mohó; ha folytonos, akkor gradiens-módszer). De általában így néz ki a tanítás:

```
def train_and_val(hyp, train_set, val_set):  
    model = build_model(hyp)  
    for _ in range(300):  
        train_losses.append(model.train(train_set))  
        val_losses.append(model.test(val_set))  
        plot_learning_curves(train_losses, val_losses)  
        if val_losses[-1] < best_val_loss:  
            best_val_loss = val_losses[-1]  
            best_weights = weights  
    return best_val_loss, best_weights
```

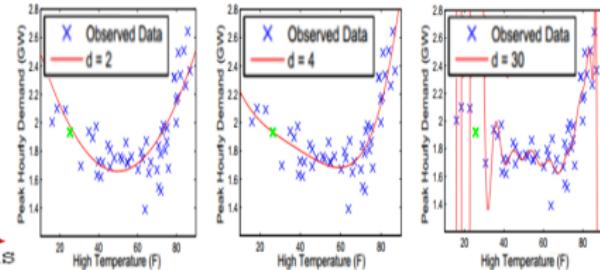
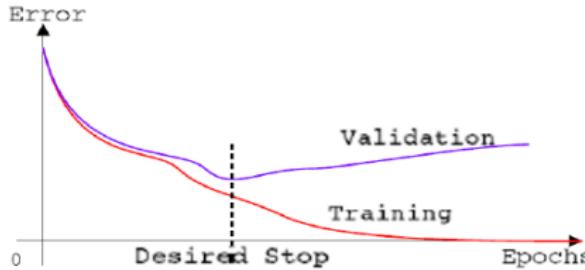


Felügyelt Tanulás

Túltanulás

A betanított leképezés nem általánosít jó! Mohó módon csak az $(x_n, y_n), n = 1, \dots, N$ párokra optimalizáltunk, megtanult olyat is, ami csak ezekre igaz, általában viszont nem! Ez a **túltanulás**.

- Korai Leállás: Monitorozzuk a hibát egy tanító halmaztól független validációs halmazon (bal: lila görbe, jobb: zöld x), álljunk le amikor nőni kezd.



- Regularizáció: Adjunk hozzá $\lambda \|\theta\|_2^2$ -t $L(\theta, x_n, y_n)$ -hoz, hogy egyszerű modellt kényszerítsünk (Occam borotvája: egyszerűbb modell jobban általánosít).
- Zajtalanítás: Zajosítsuk a mintát tanításkor (N nő), a leképezésnek meg kell tanulnia a zajt kiküszöbölni.
- Augmentáció: Új tanító példák generálása (N nő), pl. képeknél kivágás, tükrözés, forgatás, színcsere (ami nem változtatja y -t).
- Finomhangolás: Indítsuk θ -t nagy adatbázison optimalizált θ^* -ból.

Felügyelt Tanulás

Hiperparaméterek

Az ML-ben sok hiperparaméter van: leképezés, hibafüggvény, optimalizációs módszer kiválasztása, regularizáció λ -ja, zajszint, stb. Hogyan állítsuk be ezeket?

- Másoljuk le másnak a beállításait. Nem lesz a mi feladatunkon is optimális.
- Csinálunk Véletlen Keresést vagy Bayes-Optimalizációt*: tanítsunk sok hiperparaméter-beállítással a tanító halmazon, mohón válasszuk a legjobbat a validációs halmazon (sok lila görbe közül a legalsó). Sajnos ez többszörözi a számítási igényt, de egyelőre nem tudunk jobbat...

* BO is felügyelt: x = hiperparaméter, y = validációs halmazon vett hiba.

```
train_set , val_set, test_set = load_dataset()
for _ in range(100):
    hyps.append(pick_hyperparams(hyps, val_losses))
    val_losses.append(train_and_val(hyps[-1], train_set,
val_set))
    if val_losses[-1] < best_val_loss:
        best_val_loss = val_losses[-1]
        best_hyp = hyps[-1]
train_and_val(best_hyp, train_set + val_set, test_set)
```



K Legközelebbi Szomszédok

Paraméteres leképezés és Optimalizálási módszer

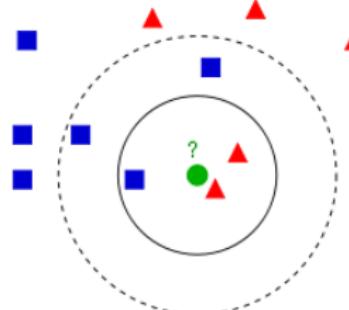
- **K Legközelebbi Szomszédok (KNN):** (Nem-)paraméteres leképezés, fix K -ra becsüljünk a K legközelebbi x_n -ekhez tartozó y -ok átlagával:

$$f(\theta, x) = \sum_{n=1}^N \underbrace{\frac{\mathbb{I}(x_n \text{ benne van } x \text{ K legközelebbi szomszédjában})}{K}}_{w_n} y_n.$$

Speciálisan $\theta = \{(x_n, y_n) \mid n = 1, \dots, N\}$, nem kell optimalizálni (de lehetne).

Le kell tárolni az összes tanítópárt, nem kompresszálja x_n -ekben rejlő információt. Legközelebbi szomszédok: $\text{argsort}_n \|x_n - x\|_2^2$.

Előny: Egyszerű leprogramozni. Hátrány: Ha N nagy, tárolni és sorbarendezni erőforrásigényes.



K Legközelebbi Szomszédok

Szoftver

Kód elérhető az sklearn csomagban (Python):
sklearn.neighbors.KNeighborsClassifier

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
>>> print(neigh.predict_proba([[0.9]]))
[[ 0.66666667  0.33333333]]
```

sklearn.neighbors.KNeighborsRegressor

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsRegressor
>>> neigh = KNeighborsRegressor(n_neighbors=2)
>>> neigh.fit(X, y)
KNeighborsRegressor(...)
>>> print(neigh.predict([[1.5]]))
[ 0.5]
```



Véletlen Erdők

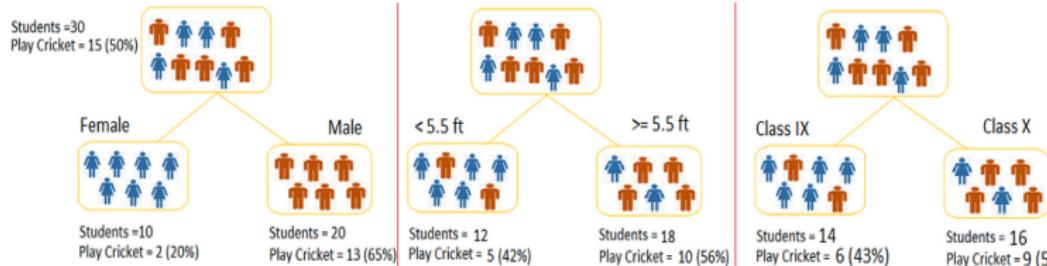
Paraméteres leképezés és Optimalizálási módszer

- Döntési Fa (DT): Paraméteres leképezés x komponenseinek (változók) sorozata, amik a tanító halmaz legjobb vágásait adják y szerint minden lépésben (Kereszt-entrópia vagy Négyzetes hiba szerint). Gyakorlatilag if-then-else fastruktúrában, paraméterek a változók és a küszöök.

$$f(\theta, x) = \sum_{n=1}^N \underbrace{\frac{\mathbb{I}(x_n \text{ az } x \text{ levelében levő } K' \text{ input egyike})}{K \cdot K'}}_{w_n} y_n.$$

Optimalizálás: faépítés mohó algoritmussal a legjobb vágásokra.

Pl. krikettezésre egy szinten 3 lehetséges vágás közül a Nő/Férfi a legjobb:



Előny: értelmezhető, olcsó használat, a párokat nem kell tárolni. Hátrány: faépítés NP-teljes, a mohó lokálisan optimális, de globálisan nem optimális.



Véletlen Erdők

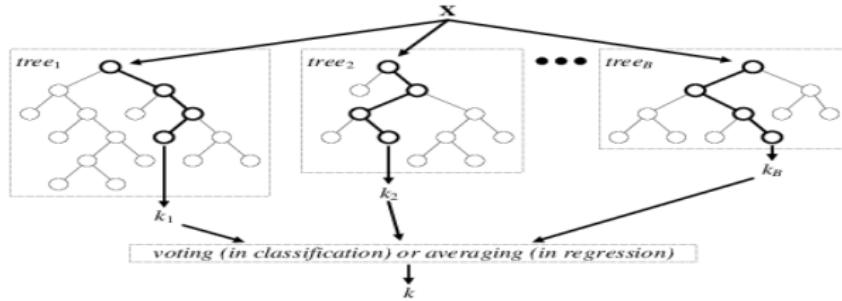
Paraméteres leképezés és Optimalizálási módszer

- **Véletlen Erdő (RF)**: Paraméteres leképezés K db DT, minden egyik véletlen változó- és példa-részhalmazzal, ezek eredményeinek átlaga a végső becslés:

$$f(\theta, x) = \sum_{n=1}^N \underbrace{\sum_{k=1}^K \frac{\mathbb{I}(x_n \text{ az } x \text{ levelében levő } K'_k \text{ input egyike } DT_k\text{-ban})}{K \cdot K'_k}}_{w_n} y_n.$$

A véletlen generálás (zaj) elkerüli a túltanulást, kevésbé mohó, viszont kevésbé értelmezhető.

Előny: **párhuzamos, olcsó** használat. Hátrány: szintén **NP-teljes**.



Véletlen Erdők

Szoftver

Kód elérhető az sklearn csomagban (Python):
sklearn.ensemble.RandomForestClassifier

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.datasets import make_classification
>>>
>>> X, y = make_classification(n_samples=1000, n_features=4,
...                             n_informative=2, n_redundant=0,
...                             random_state=0, shuffle=False)
>>> clf = RandomForestClassifier(max_depth=2, random_state=0)
>>> clf.fit(X, y)
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                      max_depth=2, max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                      oob_score=False, random_state=0, verbose=0, warm_start=False)
>>> print(clf.feature_importances_)
[ 0.17287856  0.806008704  0.01884792  0.00218648]
>>> print(clf.predict([[0, 0, 0, 0]]))
[1]
```

sklearn.ensemble.RandomForestRegressor

```
>>> from sklearn.ensemble import RandomForestRegressor
>>> from sklearn.datasets import make_regression
>>>
>>> X, y = make_regression(n_features=4, n_informative=2,
...                         random_state=0, shuffle=False)
>>> regressor = RandomForestRegressor(max_depth=2, random_state=0)
>>> regressor.fit(X, y)
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=2,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                      oob_score=False, random_state=0, verbose=0, warm_start=False)
>>> print(regressor.feature_importances_)
[ 0.17339552  0.81594114  0.          0.01066333]
>>> print(regressor.predict([[0, 0, 0, 0]]))
[-2.50699856]
```