Mesterséges Intelligencia Gyakorlat

Állapottér Reprezentáció

- Állapotok halmaza: S
- Kezdőállapot(ok): $s \in S$
- Végállapot(ok): $f \in S$
- Állapotátmenet-függvény (operátor): $Op \subset S \times S$
- Költségfüggvény: $c \subset Op \times R$

8(n)-királynő

Helyezzünk el 8 királynőt a sakktáblán, hogy semelyik kettő ne üthesse egymást!

- 8 elemű lista, amely elemei definiálják hanyadik sorban van a királynő: $\{0, ..., 8\}^8$
- egyik oszlopban sincs még királynő: [0, 0, ..., 0]
- egy megfelelő 8 elemű permutáció
- állapotátmenet:
  [a_1, a_2, ..., a_8] -> [b_1, b_2, ..., b_8]
  - valamelyik oszlop változik:
    létezik i (1, ..., 8), a_i != b_i
  - maximum egy oszlop változik:
    minden j (1, ..., 8), j != i ,a_j = b_j
  - vagy leveszünk egy királynőt, vagy olyan helyre tesszük, hogy ne legyen ütésben (nincsenek egy sorban és sorok közötti távolság nem egyenlő az oszlopok közötti távolsággal):
    b_i = 0 OR (b_i != b_j AND |b_i-b_j| != |i-j|)

Hanoi-tornyai

Adott három pálca és az elsőn három (n) különböző átmérőjű korong, felfele csökkenő méretben elhelyezve. A feladat a korongok áthelyezése az utolsó pálcára úgy, hogy mindig csak a legfelső korongot vehetjük le egy pálcáról és egy korongot csak nála nagyobb méretű korongra rakhatunk.

- 3 (n) elemű lista, amelynek i-ik eleme az i méretű korong pozícióját (1, 2, 3) jelöli: $\{1, 2, 3\}^n$
- mindegyik korong az első oszlopon van: [1, ..., 1]
- mindegyik korong az utolsó oszlopon van: [3, ..., 3]
- állapotátmenet:
  [a_1, a_2, ..., a_n] -> [b_1, b_2, ..., b_n]
  - valamelyik korong pozíciója változik:
    létezik i (1, ..., n), a_i != b_i
  - maximum egy korong pozíciója változik:
    minden j, j != i, a_j = b_j
  - valamelyik oszlopról a fölső korongot (azon az oszlopon nincs kisebb méretű) áttesszük egy másik oszlopra, ahol nincs annál kisebb korong:
    minden l < i, a_i != a_l AND b_i != b_l

Tehát egy gráfként reprezentáljuk a problémát, ahol az állapotok a gráf csúcsai, az állapotátmenet függvény definiálja az éleket, a költségfüggvény pedig az élek súlyait. A feladat pedig egy útvonal keresése a kezdő állapotból valamelyik végállapotba, többnyire feltétel még, hogy minimális költségű legyen az út.

A költségfüggvény változtatásával kihathatunk az optimális megoldás milyenségére. Pl.: a feladat legrövidebb út keresése egyik városból a másikba. Az állapotok a városok, él két város között akkor van, ha azok közvetlen össze vannak kötve közúttal. Ha a költségfüggvény a közút hossza, akkor az optimális megoldás a legrövidebb utat adja meg. Míg ha az út szakasz megtételéhez szükséges idő, akkor a leggyorsabbat.

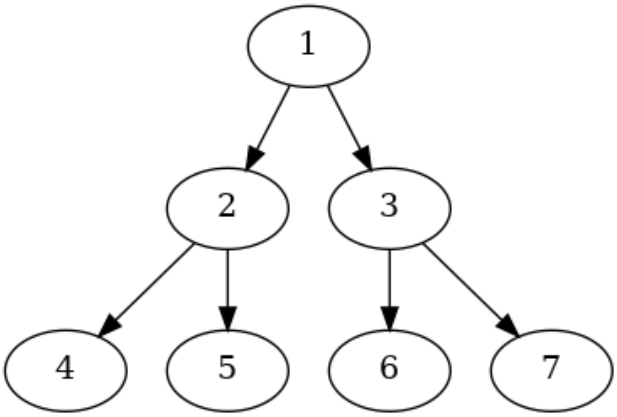
FEL

Informálatlan Keresés

Algoritmus, amely a fent definiált gráfban való útvonal keresésre szolgál.

Alap algoritmus

```
search()
  s #kezdő állapot
  N.add(s) #nyílt halmaz / perem
  while N not empty
    c = N.get() #a keresés aktuális állapota
    if végállapot(c) return c
    for n in szomszéd(c)
      N.add(n)
  return nincs MO. #nem vezet út kezdőállapotból végállapotba
```



Futtassuk le az algoritmust a fenti gráfon. Kezdőállapot a fa gyökere, végállapot pedig nincs.

Algoritmus futtatása sor adatszerkezettel (szélességi keresés):

```
| iter | N         |        c |
|------|-----------|----------|
|1     | [1]       |        1 |
|2     | [2,3]     |        2 |
|3     | [3,4,5]   |        3 |
|4     | [4,5,6,7] |        4 |
|5     | [5,6,7]   |        5 |
|6     | [6,7]     |        6 |
|7     | [7]       |        7 |
|8     | []        | nincs MO.|
Bejárás sorrendje: 1, 2, 3, 4, 5, 6, 7
```

Algoritmus futtatása verem adatszerkezettel (mélységi keresés):

```
| iter | N       |        c |
|------|---------|----------|
|1     | [1]     |        1 |
|2     | [2,3]   |        3 |
|3     | [2,6,7] |        7 |
|4     | [2,6]   |        6 |
|5     | [2]     |        2 |
|6     | [4,5]   |        5 |
|7     | [4]     |        4 |
|8     | []      | nincs MO.|
Bejárás sorrendje: 1, 3, 7, 6, 2, 5, 4
```

Azt feltehetjük, hogy a branching factor véges, tehát véges számú szomszédja van minden állapotnak. Viszont ettől még előfordulhat, hogy a keresési fa végtelen mélységű. Például, ha az algoritmust egy olyan gráfon futtatjuk, amely tartalmaz kört.

Ez a szélességi keresésnek nem okoz jelentős problémát, megtalálja a végállapothoz vezető utat (feltéve, hogy van ilyen), valamennyi memóriaigény növekedés mellet. Viszont a mélységi keresés végtelen ciklusba kerülhet.

Ennek a problémának az elkerülése végett vezessünk be mélységkorlátot a keresésbe. Előnye: biztos leáll a keresés, hátránya: nem feltétlen találja meg a célig vezető utat (túl mélyen van a megoldás).

További javítás: iteratívan mélyülő mélységkorlátos keresés. Kezdjünk `1` korláttal, és növeljük, amíg meg nem találunk egy megoldást.

Idő és tárbonyolultság

```
|     alg     |    Ido    |    Tár    | tel | opt |
|-------------|-----------|-----------|-----|-----|
| szélességi  | O(b^d+1)  | O(b^d+1)  |  +  |  +! |
| mélységi    | O(b^m)    | O(bm)     |  +? |  -  |
| iteratív    | O(b^d)    | O(bd)     |  +  |  +! |
```

```
b: branching factor
m: maximális mélység
d: minimális mélységű megoldás mélysége
tel: teljes, megtalálja a célhoz vezető utat, feltéve hogy létezik
opt: az optimális utat találja meg
+!: ha a költség a mélység nem csökkenő függvénye
+?: ha a keresési fa véges mélységű
```

Körök elkerülése (N u Z), javítóút (OR második fele), útvonal visszakeresés (p)

```
search()
  s #kezdő állapot
  N.add(s) #nyílt halmaz / perem
  p(s) = nil #szülőre mutató pointer
  g(s) = 0 #idevezető út költsége
  Z = üres #zárt halmaz / már bejárt csúcsok
  while N not empty
    c = N.get() #a keresés aktuális állapota
    if végállapot(c) return c
    Z.add(c)
    for n in szomszéd(c)
      if n not in (N u Z) OR g(c) + KTG(c,n) < g(n)
        N.add(n)
        Z.remove(n)
        p(n) = c
        g(n) = g(c) + KTG(c, n)
  return nincs MO. #nem vezet út kezdőállapotból végállapotba
```

Ha nyílt halmaznak prioritási sor adatszerkezetet használunk, amelyben az elemek az elérési idejük szerint vannak rendezve, megkapjuk az egyenletes költségű keresést.

FEL

Informált Keresés

Módosítsuk a fenti algoritmust a következők szerint.

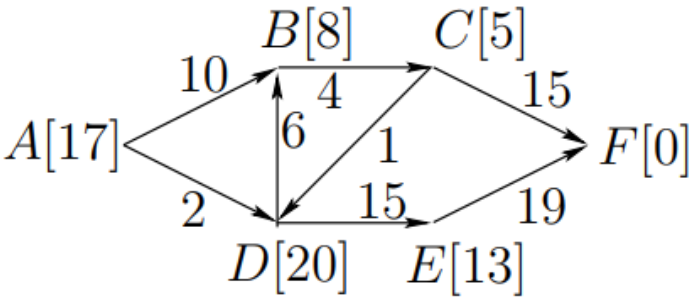
- `g(x)`: x-ig megtett út költsége
- `h(x)`: heurisztikus függvény, becslés az x-től célállapotig vezető út költségére
- `f(x) := g(x) + h(x)`
- `N` adatszerkezet legyen prioritási sor, csomópontok legyenek `f` szerint rendezve.

Heurisztika megadásának módszere:

- enyhítsünk a probléma feltételrendszerén
- oldjuk meg az `egyszerű` feladatot
- használjuk az optimális megoldás értékét heurisztikaként az eredeti feladatban.

Pl.: Városok közötti útvonal keresés esetén jó heurisztika a légvonalbeli távolság.

- enyhítés: nem kell úton közlekedni
- megoldás egyszerűen számolható mondjuk a városok koordinátáinak segítségével.



Keresd meg a legrövidebb utat a fenti gráfon az `A`-tól `F`-be az `A*` algoritmus futtatásával!

N	Z
×A(nil, 0, 17)	
×B(A, 10, 18) D(A, 2, 22)	A(nil, 0, 17)
×C(B, 14, 19) D(A, 2, 22)	A(nil, 0, 17) B(A, 10, 18)
F(C, 29, 29) ×D(A, 2, 22)	A(nil, 0, 17) B(A, 10, 18) C(B, 14, 19)
×B(D, 8, 16) E(D, 17, 30) F(C, 29, 29)	A(nil, 0, 17) C(B, 14, 19) D(A, 2, 22)
×C(B, 12, 17) E(D, 17, 30) F(C, 29, 29)	A(nil, 0, 17) D(A, 2, 22) B(D, 8, 16)
×F(C, 27, 27) E(D, 17, 30)	A(nil, 0, 17) D(A, 2, 22) B(D, 8, 16) C(B, 12, 17)

[Informált és informálatlan keresés összehasonlítás]

[Interaktív vizualizáció]

[Super Mario irányítása]

[FEL]

Kétszemélyes Játékok

- két játékos
- felváltva hajtanak végre akciókat
- akció hatására a játék egy szomszédos állapotba kerül
- determinisztikus (nincs benne véletlen)
- 0 összegű (amennyit nyer az egyik, annyit veszít a másik)
- játéktér ábrázolása fa struktúrával gráf helyett minden páros mélységben a kezdő, páratlan a másik játékos döntése alapján folytatódik

Max és Min ‘játékosok’, Max kezdi a játékot, maximalizálja a nyereséget, a lehetséges folytatások értékei közül választja a maximálisat. Folytatás kiértékelése a Min játékos ‘feladata’, aki minimalizálja a veszteséget, tehát a lehetséges folytatások közül veszi a minimális értékűt. Majd ez ismétlődik, amíg végállapothoz nem érkezünk, ennek az értéke a kezdő játékos szemszögéből a nyereség értéke.

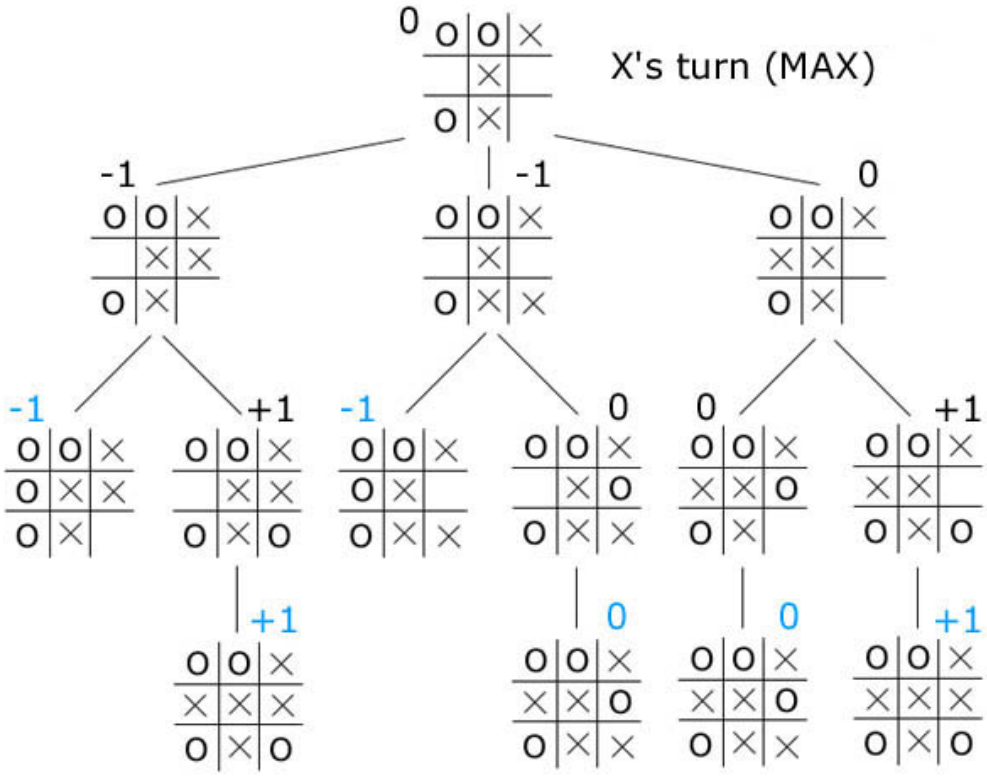
Min-Max Algoritmus

Min-max algoritmus játékfák kiértékelésére, `max(node)` hívása a játék kezdőállapotával.

```
max_func(node)
  if node is terminal
    return value(node)
  value = -inf
```

```
  for each child of node
    value = max(value, min_func(child))
  return value

min_func(node)
  if node is terminal
    return value(node)
  value = inf
  for each child of node
    value = min(value, max_func(child))
  return value
```



https://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html

Min-Max Algoritmus, alpha-beta vágással

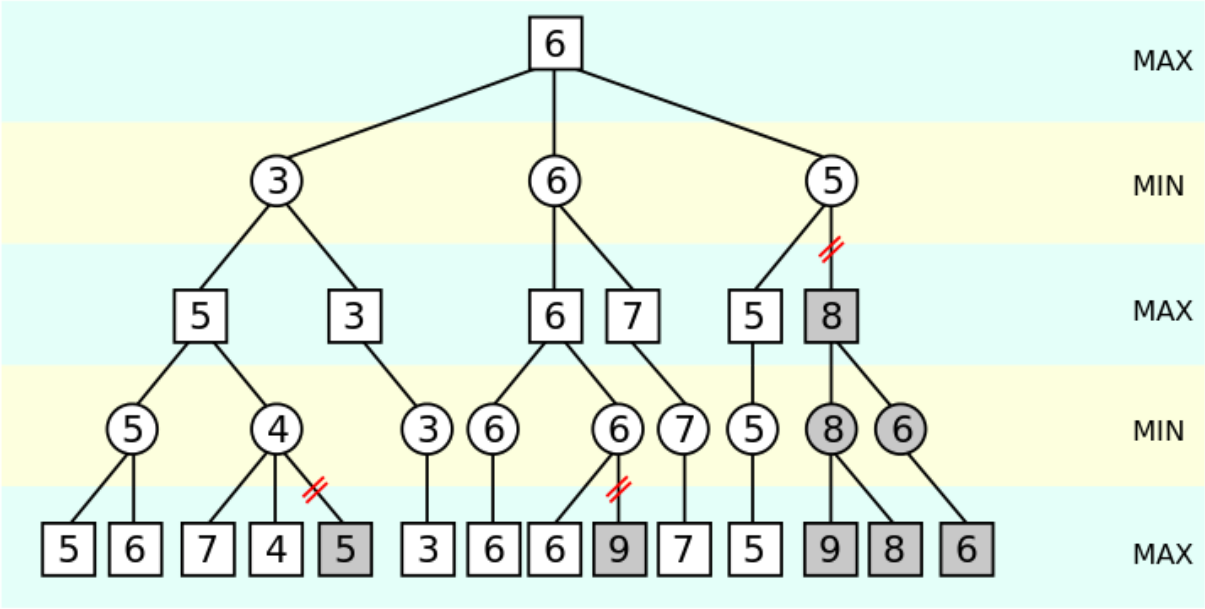
Felesleges irányok levágása, hogy a végeredmény ekvivalens maradjon a vágás nélküli esettel. Mélységkorlát bevezetése esetén nem feltétlen végállopot a fa levele, definiálnunk kell valamilyen hasznosság függvényt a nem végállapotok kiértékeléséhez. Pl.

- sakk: táblán lévő bábúk értékkülönbsége
- amőba: minták keresés, pl. mindkét oldalon nyitott 3-asok
- othello: értékes pozíciók, sarkok elfoglalásának esélye

Min-max algoritmus, alpha-beta vágással és mélységkorláttal, játékfák kiértékelésére, `max(node, 4, -inf, inf)` hívása a játék kezdőállapotával, `4` mélységkorláttal.

```
max_alpha_beta(node, depth, alpha, beta)
  if node is terminal or depth = 0
    return value(node)
  value = -inf
  for each child of node
    value = max(value, min_alpha_beta(child, depth-1, alpha, beta))
    alpha = max(value, alpha)
    if value >= beta
      break # beta cutoff
  return value

min_alpha_beta(node, depth, alpha, beta)
  if node is terminal or depth = 0
    return value(node)
  value = inf
  for each child of node
    value = min(value, max_alpha_beta(child, depth-1, alpha, beta))
    beta = min(value, beta)
    if value <= alpha
      break # alpha cutoff
  return value
```



https://upload.wikimedia.org/wikipedia/commons/thumb/9/91/AB_pruning.svg

interactive example video

FEL

Felügyelt Tanulás

Adott egy `n` elemű, példákat tartalmazó adatbázis, pl az iris adatbázis, amely elemei `(X,Y)` alakban reprezentálhatók. Ahol $X \in R^{n \times d}$ dimenziós valós jellemző vektorokból álló mátrix, ami a példák leírására szolgál. Az `iris` adatbázis esetén a példák száma `n=150` és a jellemzők száma pedig `d=4`, amelyek a virág szirmainak a méretét írják le (jellemzők). Az $Y \in C$ pedig a példákhoz tartozó osztálycímkéket jelöli, az `iris` esetén `C = {setosa,versicolor,virginica}`. A felügyelt tanulás feladata pedig egy modell vagy függvény és paraméterinek a keresése (F_Θ`, F: függvény,`Θ`: paraméterek`), amely a példák alapján jól közelíti az `X->Y` leképezést ($F_\Theta(X) \approx Y$). Továbbá elvárás az is, hogy a modell jól teljesítsen az eddig nem látott példák esetén is, jól tudjon általánosítani.

A példák leírására szolgáló változók, melyek lehetnek folytonosak vagy kategorikusak. Kategorikus jellemzők automatikus átalakítása numerikussá esetleges hibaforrás lehet. Pl a `{piros, kék, zöld} -> {0, 1, 2}` átalakítás esetén különböző távolság érték jön ki `piros<->kék` és `piros<->zöld` esetén. Ok: az átalakítás sorrendiséget vezetett be, ami nem minden esetben hibás (pl. `{fiatal, középkorú, idős}`). Bináris jellemzőkké alakítással elkerülhető.

További előnye lehet a jellemzők egy értéktartományra hozásának. Pl. egy nagy tarományon mozgó jellemző dominálhatja a távolság függvény értékét. Érdemes normalizálni a jellemzőket:

- normalizálás: `(x-min)/(max-min)`
- standardizálás: `(x-mean)/std`

K-NN

`K` legközelebbi szomszéd osztályozó algoritmus (K-NN). A kérdéses példa címkéjét többségi szavazás segítségével jósolja meg a példához valamilyen távolság függvény alapján legközelebbi `K` db tanító példa alapján.

Leginkább alkalmazott távolság függvények

- Euklideszi távolság: $d(a,b) = sqrt(\sum_i (a_i - b_i)^2)$
- Manhatten távolság: $d(a,b) = \sum_i |a_i - b_i|$
- Cosinus hasonlóság: $s(a,b) = a \times b / |a||b|$

Modellek Kiértékelése

- adatbázis szétosztása tanító és teszt adatbázisokra:
  Algoritmus teljesítményének a mérése a tanító adatbázis alapján a teszt példákra adott jóslatok a tényleges címkéktől való eltérései alapján.
- paraméterek finomhangolása: az adatbázis tanító, validációs és teszt halmazokra vágásával. Paraméter hangolás a validációs halmazon.
- Metrikák:

- átlagos négyzetes hiba: $1/n \sum_i (F_\Theta(x) - y)^2$
- [tévesztési mátrix](#) alapú metrikák:
  - TP: true positive
  - FP: false positive
  - FN: false negative
  - TN: true negative

```
F(x)/y | C1 | C2
-----------------
    C1 | TP | FP
    C2 | FN | TN
```

FEL

Naive-Bayes Osztályozó

A tanító adatbázis alapján egy `(F1, ..., Fd)` jellemzőkkel leírt példához rendeljük hozzá a legvalószínűbb osztálycímkét:

```
C* = argmax_C P(C|F1, ..., Fd)
```

Azaz, keressük azt az osztálycímkét, amelynek a legnagyobb a valószínűsége, feltéve az adott példához tartozó jellemző értékeket.

Ha feltesszük, hogy a jellemzők páronként függetlenek egymástól és alkalmazzuk a láncszabályt és a bayes szabályt, akkor a fenti valószínűség a következő valószínűségek szorzatára alakítható át:

```
P(C|F1, ..., Fd) = P(C) P(F1|C) ... P(Fd|C) 1/P(F1, ..., Fd)
```

A lehetséges osztálycímkékre nézve az `1/P(F1, ..., Fd)` érték konstans szorzóként viselkedik, gyakorlatban nem számolják ki az értékét.

Példa

Az alábbi megfigyelések (adatbázis) alapján melyik osztályba tartozik az a példa, amely a következő jellemzőkkel írható le

```
Color=Red, Type=SUV, Origin=Domestic
```

```
Example No. | Color  | Type   | Origin   | Stolen?
------------|--------|--------|----------|--------
1           | Red    | Sports | Domestic | Yes
2           | Red    | Sports | Domestic | No
3           | Red    | Sports | Domestic | Yes
4           | Yellow | Sports | Domestic | No
5           | Yellow | Sports | Imported | Yes
6           | Yellow | SUV    | Imported | No
7           | Yellow | SUV    | Imported | Yes
8           | Yellow | SUV    | Domestic | No
9           | Red    | SUV    | Imported | No
10          | Red    | Sports | Imported | Yes
```

```
P(Y|R,Su,D) ~= P(Y) P(R|Y) P(Su|Y) P(D|Y)
             = 5/10    3/5     1/5    2/5 = 3/125

P(N|R,Su,D) ~= P(N) P(R|N) P(Su|N) P(D|N)
             = 5/10    2/5     3/5    3/5 = 9/125
```

Valószínűségek becslése megfigyelések alapján: jó esetek száma osztva az összes esetek számával. `P(Y) = 5/10`

Feltételes valószínűség becslése hasonlóan, de előbb akalmazni kell a feltételes valószínűség [definícióját](#). `P(R|Y) = P(R,Y) / P(Y) = 3/10 / 5/10 = 3/10 * 10/5 = 3/5`

Tehát a válasz `N`, mert a hozzá tartozó valószínűség becslés a legnagyobb.

Együttes előfordulások számolása ritka értékek esetén lehet `0` az együttes előfordulások száma, ennek elkerülése érdekében alkalmazható az [m-estimate](#) módszer. Továbbá folytonos értékek kategorizálása, vagy valamilyen valószínűségi eloszlással való becslése.

FEL

Döntési Fa Osztályozó

A korábbiakhoz hasonlóan adott egy tanító adatbázis. A döntési fa osztályozó egy gyökeres fát épít az adatok alapján, amelynek a belső csomópontjai egy-egy jellemzővel vannak címkézve, míg a levelek pedig osztálycímkékkel. Egy belső csomópontnak pedig

annyi gyereke van, ahány lehetséges értéket vehet fel a címkében szereplő jellemző.

Osztályozás

Egy példa osztályozása a felépített fa segítségével a következőképpen oldható meg: a fa gyökerétől kezdve haladjunk a levelek felé annak megfelelően, hogy az adott példában az egyes jellemzők milyen értéket vesznek fel. Ha elérünk egy levélig, akkor a példa címkéje legyen a levélben szereplő osztálycímke.

```
predict(root, X)
  if root.label != nil
    return root.label
  return predict(root.child[X.value[root.attribute]], X)
```

ID3 Algoritmus

Döntési fa építésére használható az `ID3 (Iterative Dichotomiser 3)` algoritmus, amely bemenete egy adatbázis és egy jellemző lista (ami alapján fel lehet építeni a fát), a kimenete pedig az elkészült fa gyökere.

Ez egy mohó algoritmus, amely a fa gyökerénél kezdi az építést a legjobbnak tűnő jellemző kiválasztásával. A megtalált jellemző lehetséges értékei alapján szétválogatja a tanító adatbázist és rekurzívan meghívja a faépítő algoritmust a részadatbázissal, de a megtalált attribútum nélkül.

A legjobbnak tűnő jellemző pedig az entrópia és az Information Gain mértéke alapján kerül kiválasztásra.

Entrópia: `E(S) = -` $\sum_i p_i \times log(p_i)$, ahol `p_i` az `i`-ik osztály előfordulási valószínűsége.

Information Gain: `Gain(S,F) = E(S) -` $\sum_{v \in F} |S_v|$ `/ |S| ×` $E(S_v)$

Ahhoz a jellemzőhöz tartozik a maximális `Gain` érték, amely legkevésbé csökkenti az adatbázis entrópiáját. Azaz, amelyik jellemző különböző értékeihez tartozó példák különböző osztályba is tartoznak, tehát a jellemző értékei jól szeparálják az osztályokat egymástól.

```
ID3(S, F)
  root = new node
  if all label in S is c
    root.label = c
    return root
  if F is empty
    root.label = most frequent label in S
    return root
  F_best = argmax_F Gain(S, F)
  root.attribute = F_best
  for all value v of F_best
    S_v: samples of S, where value of F_best is v
    child = new node
    if S_v is empty
      node.label = most frequent label in S
    else
      node = ID3(S_v, F-F_best)
    root.child[v] = node
  return root
```

Példa

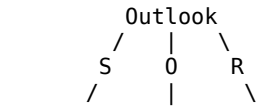
Az alábbi megfigyelések (adatbázis) alapján melyik osztályba tartozik az a példa, amely a következő jellemzőkkel írható le

```
Outlook=Rain, Temp=Hot, Humidity=High, Wind=Strong
```

Day No.	Outlook	Temp	Humidity	Wind	Play?
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Építsük fel a fát, az entrópia számításához használjuk a Gini-indexet (E(S)=4×p(1-p)):

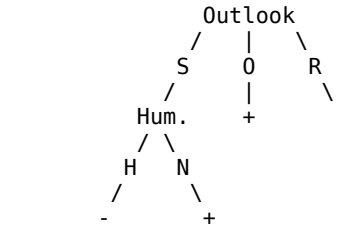
- Az adatbázis entrópiája: `E(S) = 4×9/14×5/14 = 90/98`
- Válasszuk ki a legjobbnak tűnő jellemzőt a fa gyökeréhez:
  - `Gain(S,O)=E(S) - (5/14×4×2/5×3/5 + 4/14×4×4/4×0/4 + 5/14×4×3/5×2/5) = 90/98 - 24/35 = 0.23`
  - `Gain(S,T)=E(S) - (4/14×4×2/4×2/4 + 6/14×4×4/6×2/6 + 4/14×4×3/4×1/4) = 90/98 - 37/42 = 0.03`
  - `Gain(S,H)=E(S) - (7/14×4×3/7×4/7 + 7/14×4×6/7×1/7) = 90/98 - 36/49 = 0.18`
  - `Gain(S,W)=E(S) - (8/14×4×6/8×2/8 + 6/14×4×3/6×3/6) = 90/98 - 6/7 = 0.06`
  - A fentiek alapján az `Outlook` jellemző lesz a fa gyökerében.



- Bontsuk részekre az adatbázist a kiválasztott jellemző alapján és rekurzívan építsük tovább a fát.
- `Outlook=Sunny` esetén az adatbázis:

Day No.	Outlook	Temp	Humidity	Wind	Play?
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes

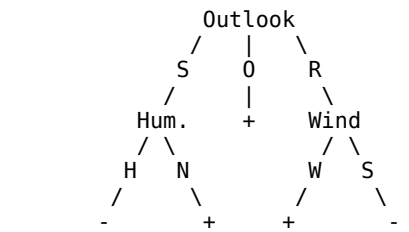
- Innen már látszik is, hogy a `Humidity` a legjobb jellemző, mert tökéletesen szeparálja a példákat, de a számolás:
  - `E(`$S_{O=S}$`)=4×2/5×3/5=24/25`
  - `Gain(`$S_{O=S}$`,T)=24/25 - (2/5×4×0/2×2/2 + 2/5×4×1/2×1/2 + 1/5×4×1/1×0/1) = 24/25 - 4/10`
  - `Gain(`$S_{O=S}$`,H)=24/25 - (3/5×4×0/3×3/3 + 2/5×4×2/2×0/2) = 24/25 - 0`
  - `Gain(`$S_{O=S}$`,W)=24/25 - (3/5×4×1/3×2/3 + 2/5×4×1/2×1/2) = 24/25 - 14/15`
- `Outlook=Overcast` esetén minden példa pozitív, tehát a fa:



- `Outlook=Rain` esetén az adatbázis:

Day No.	Outlook	Temp	Humidity	Wind	Play?
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D10	Rain	Mild	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

- Itt pedig az látszik, hogy a `Wind` jellemző szeparál.
- A felépített fa pedig:

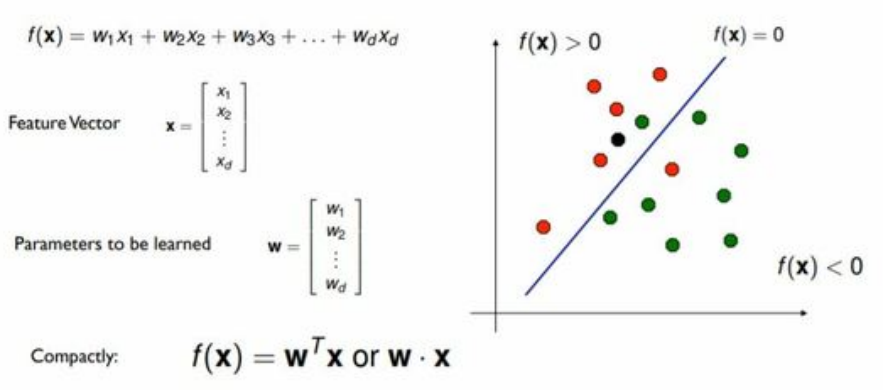


Ugyanez a fa lenne az eredmény, ha a `Gini-index` helyett a `Shannon` féle entrópiát használjuk, csak a számolás változik.

FEL

Gradiens Módszer

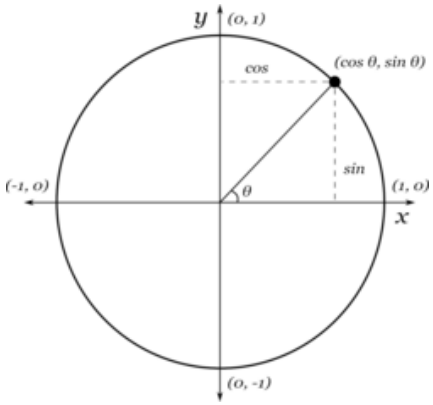
Adott egy példahalmaz a korábban definiált `(X,Y)` párok alakjában. A gradiens módszer alkalmazása során egy hipotézis függvény azon paramétereit keressük, amelyek minimalizálják a veszteségfüggvényt (költségfüggvényt).



https://mylearningsinaiml.wordpress.com/linear-classification/

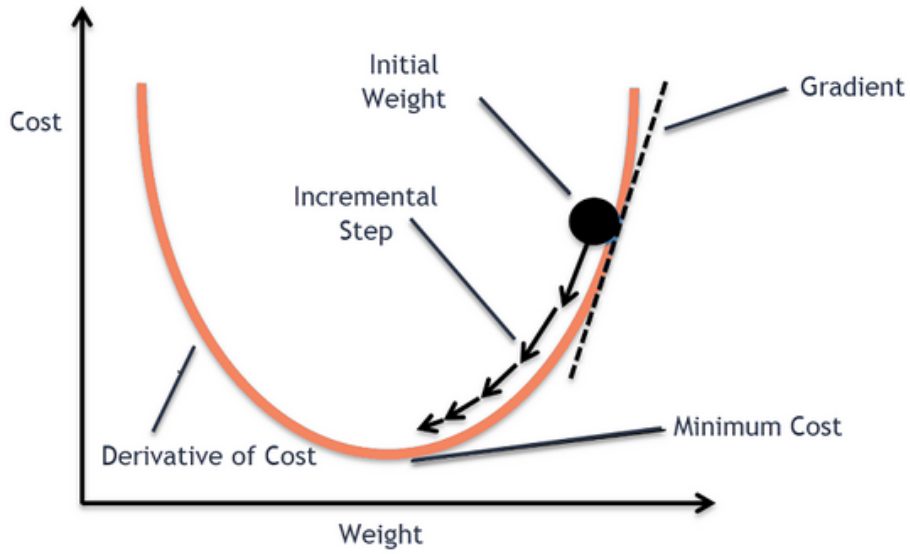
A fenti kép a lineáris szeparálást reprezentálja, ahol az elválasztó egyenes egyenlete: $w^Tx + b = 0$. Ahol a `w` az egyenes normálvektora `b` pedig az eltolás. Az egyenes bal oldalára eső pontok koordinátáit behelyettesítve az egyenletbe pozitív előjelű eredményt kapunk, így ezeket `pozitívként` osztályozza a módszer, a jobbra esőeket pedig megatívként. Azok a pontok amik az egyenesre esnek, `0` eredményt adnak.

A fentiek oka a normálvektor és az osztályozandó pont által bezárt szög cosinuszának értékéből következik. Másképp, az egyenes normálvektorának és egy pontnak a szorzata a pont egyenesre eső előjeles vetületével egyezik meg:



https://www.inchcalculator.com/unit-circle-calculator/

Az optimális súlyok keresését így képzelhetjük el:



https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/

- Weight: a keresendő paraméterek (súlyok) tere
- Cost: hiba (költségfüggvény) értékek tere
- Initial Weight: kezdeti súlyok (random) és a hozzá tarozó kölség fgv érték
- Derivative of Cost: hiba értéke a súlyok függvényében
- Minimum Cost: az ehhez a ponthoz tartozó súlyokat keressük
- Gradient: a költség fgv érintője az aktuális súlyok mellett (a költségfüggvény paraméterek szerinti parciális deriváltja)
- Incremental Step: súlyok változtatásának mértéke

Tehát a hipotézis függvény paramétereit léptetjük η lépésenként a veszteségfüggvény gradiensével ellentétes irányban. Azaz paraméterenként deriváljuk a veszteségfüggvényt, ebbe behelyettesítjük az aktuális paraméter étékeket és ennek az értéknek a lépték-szeresét levonjuk abból a paraméterből, amely szerint deriváltunk. Ezt ismételjük minden paraméterre, és az egészet iteráljuk, amíg csökken a veszteségfüggvény értéke.

Feladat

Adott egy $D \subset R \times R$ adatbázis, amely `n` példát tartalmaz. A következőképpen szeretnénk erre modellt illeszteni optimalizálás segítségével:

1. $H = h_{a,b,c}(x) = ax^2 + bx + c : a, b, c \in R$
2. $\ell(x, y, h_{a,b,c}) = (h_{a,b,c}(x) - y)^2$
3. Az optimalizáló algoritmus a gradiens módszer

Add meg a frissítési szabályt az `a` paraméter tanulásához!

Megoldás

A veszteségfüggvénybe behelyettesítve a hipotézist, egy `(x, y)` példa vesztesége

$\ell(x, y, h_{a,b,c}) = (h_{a,b,c} - y)^2 = (ax^2 + bx + c - y)^2$

Tehát a fenti példában a hipotézisen a másodfokú függvényeket értjük, a veszteség függvény pedig a négyzetes hiba. És keressük a másodfokú függvények azt a paraméterezését, amely minimalizálja a tanító hallmaz hibáját.

A teljes `n` elemű adatbázis vesztesége lesz a minimalizálandó függvény:

$J(a, b, c) = \sum_i (ax_i^2 + bx_i + c - y_i)^2$

A frissítési szabály ennek az `(a, b, c)` szerinti deriváltjából adódik:
$a_{t+1} = a_t - \eta \nabla_a = a_t - \eta \sum_i 2(a_t x_i^2 + b_t x_i + c_t - y_i) x_i^2$
$b_{t+1} = b_t - \eta \nabla_b = b_t - \eta \sum_i 2(a_t x_i^2 + b_t x_i + c_t - y_i) x_i$
$c_{t+1} = c_t - \eta \nabla_c = c_t - \eta \sum_i 2(a_t x_i^2 + b_t x_i + c_t - y_i)$,
ahol η a gradienssel való ‘léptetés’ mértékei, ∇_a pedig az `a` paraméter szerinti derivált.

Megjegyzés:

Lehet variálni az adatok típusát, pl. lehet $D \subset R^2 \times R$, lehet variálni a `h` hipotézist, pl. lehet akár trigonometrikus, pl. $h_{a,b}(x) = sin(ax + b)$, illetve lehet variálni a veszteségfüggvényt is, pl. lehet $\ell(x, y, h_{a,b}) = |h_{a,b}(x) - y|$ (ekkor persze a derivált két alesetre bomlik, ezért kissé nehezebb). Rá lehet kérdezni bármelyik paraméterre, ha több is van.

FEL

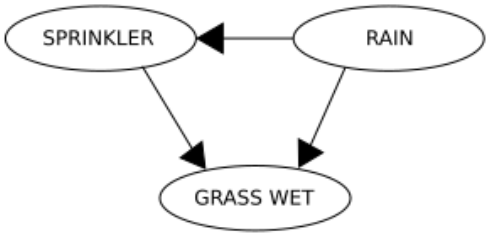
Bayes Hálók

Valószínűségi események modellezésére használatos gráf alapú modell. A gráf irányított és körmentes (DAG), melynek csomópontjai reprezentálják a valószínűségi változókat, az élek pedig a feltételes függőséget mutatják a változók között.

Következtetés Bayes Hálókban

Adott az alábbi háló struktúra és a hozzá tartozó valószínűségi táblázatok. Adjuk meg a $P(R = i | G = i)$ feltételes valószínűség értékét a háló alapján.

SPRINKLER		
RAIN	T	F
F	0.4	0.6
T	0.01	0.99



RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Tehát megfigyelések alapján összeáll egy valószínűségi modell a gyep vizességére, amely attól függ, hogy esett-e az eső vagy ment-e a locsoló, valamint a locsoló működése is függ az esőtől. Az eseményekhez tartozó táblázatokból látjuk tehát, hogy az eső valószínűsége 0.2, valamint ha esik az eső, akkor a locsoló csak 0.01 eséllyel kapcsol be, míg egyébkélnt 0.4-el.

Alakítsuk át a feltételes valószínűséget:

`P(R=i|G=i) = P(R=i,G=i)/P(G=i)`

Alakítsuk át a számlálót és a nevezők olyan alakra, hogy a táblázatokból kiolvashatók legyenek a szükséges értékek.

Számláló: Mivel nem tudjuk a locsoló állapotát, de feltétele a fű változónak, be kell vezetkünk egy x változót.

$P(R=i,G=i) = \sum_{x=\{i,h\}} P(R=i)*P(G=i|R=i,S=x)*P(S=x|R=i) =$

```
P(R=i)*P(G=i|R=i,S=i)*P(S=i|R=i) + P(R=i)*P(G=i|R=i,S=h)*P(S=h|R=i) =
0.2   *0.99          *0.01       + 0.2   *0.8           *0.99       = 0.00198 + 0.1584 = 0.16038
```

Nevező: Itt sem az eső sem a locsoló értéke nem ismert, be kell vezetni az x és y változókat.

$P(G=i) = \sum_{x,y=\{i,h\}} P(R=x)*P(G=i|R=x,S=y)*P(S=y|R=x) =$

```
P(R=i)*P(G=i|R=i,S=i)*P(S=i|R=i) + P(R=i)*P(G=i|R=i,S=h)*P(S=h|R=i) +
P(R=h)*P(G=i|R=h,S=i)*P(S=i|R=h) + P(R=h)*P(G=i|R=h,S=h)*P(S=h|R=h) =
0.2   *0.99          *0.01       + 0.2   *0.8           *0.99       +
0.8   *0.9           *0.4        + 0.8   *0.0           *0.6        = 0.00198 + 0.1584 + 0.288 + 0 = 0.44838
```

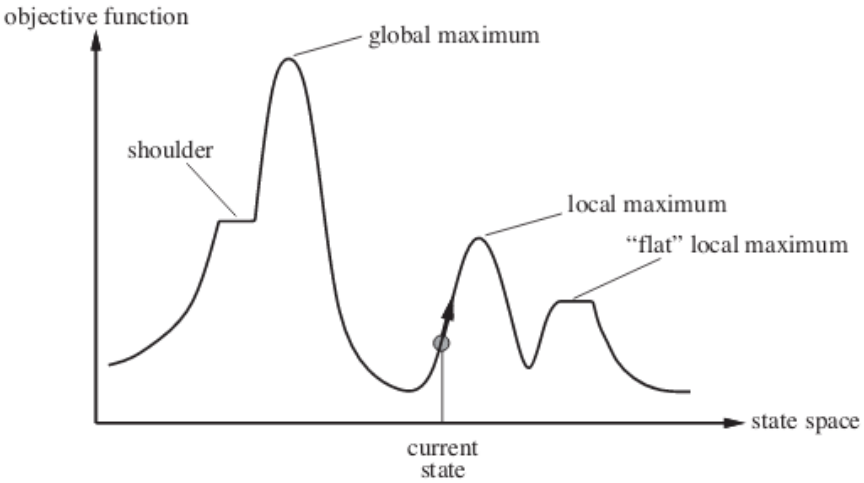
Tehát `P(R=i,G=i) = 0.16038 / 0.44838 ~= 0.3577`

FEL

Optimalizáció

Adott egy függvény, amelynek keressük azt a paraméterezését, amely mellett maximális (vagy minimális) a függvény értéke. Ha drága a függvény kiértékelése , a függvény nem differenciálható vagy a kimerítő keresés nem kivitelezhető, akkor a következő algoritmusok segítségünkre lehetnek.

Tipikus feladat: utazó ügynök probléma.



Hegymászó algoritmus

Ha találunk egy jobb paraméterezést, akkor azt mentsük el.

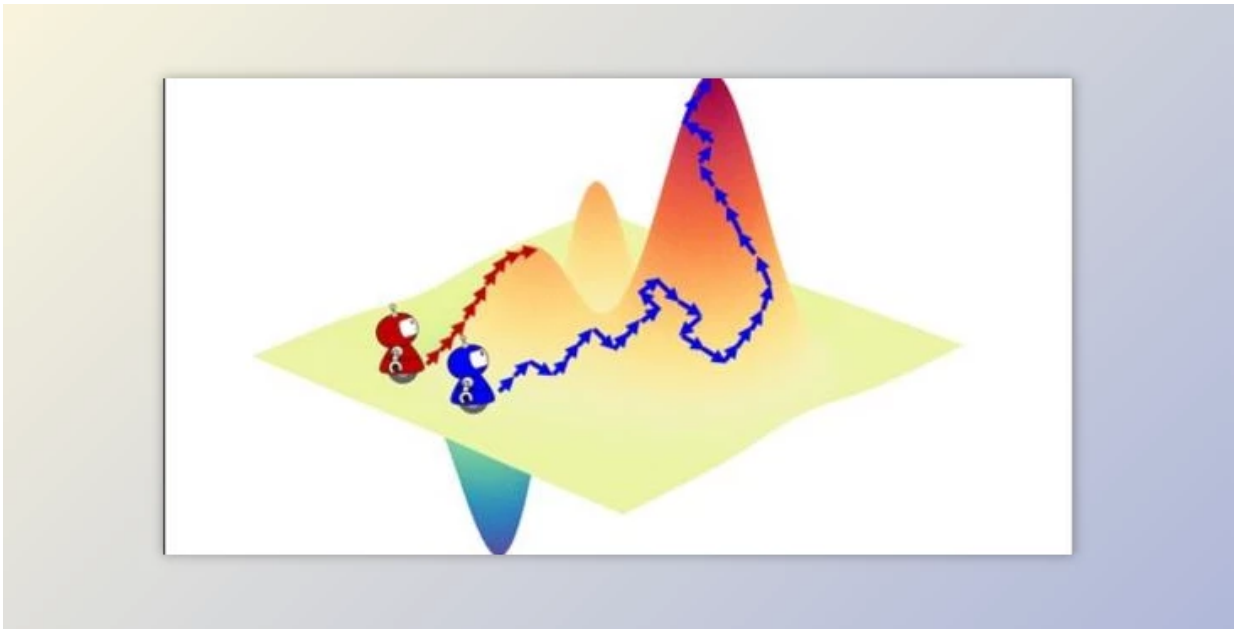
```
s: kezdő állapot
k: 0 #iterátor
while not exit
  n = szomszéd(s)
  k = k + 1
  if f(s) < f(n)
    s = n
return s
```

Több változat létezik: * legjobbat a szomszédok közül (mohó) * egyenletesen véletlenszerűen választunk a szomszédok közül (stochastikus) * függvény értékkel súlyozott alapján véletlenszerű szomszéd választás * újraindított helymászó, valahányszor újraindítjuk, hátha jobb eredményt kapunk

Szimulált hűtés

Nem csak a jobb értékű paramétert fogadjuk el, hanem valamekkora valószínűséggel a rosszabbat is (Wikipedia).

```
s: kezdő állapot
k: 0 #iterátor
T_k: kezdeti hőmérséklet
  while not exit
  n = szomszéd(s)
  if f(s) < f(n) or exp(f(n)-f(s)/T_k) < rand(0,1)
    s = n
  k = k + 1
  T_k = T_0/k
```

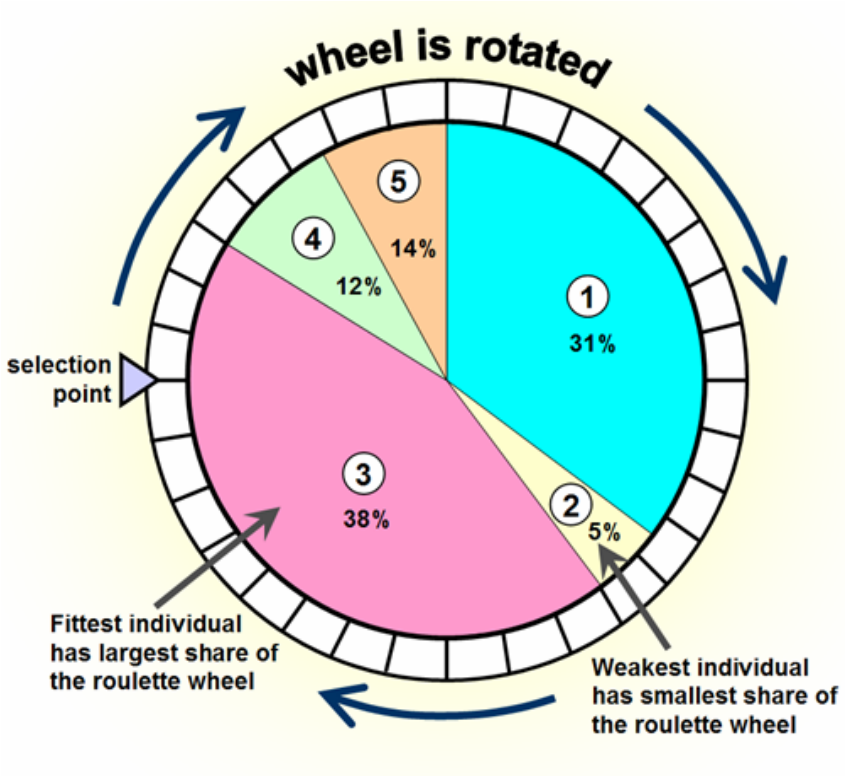


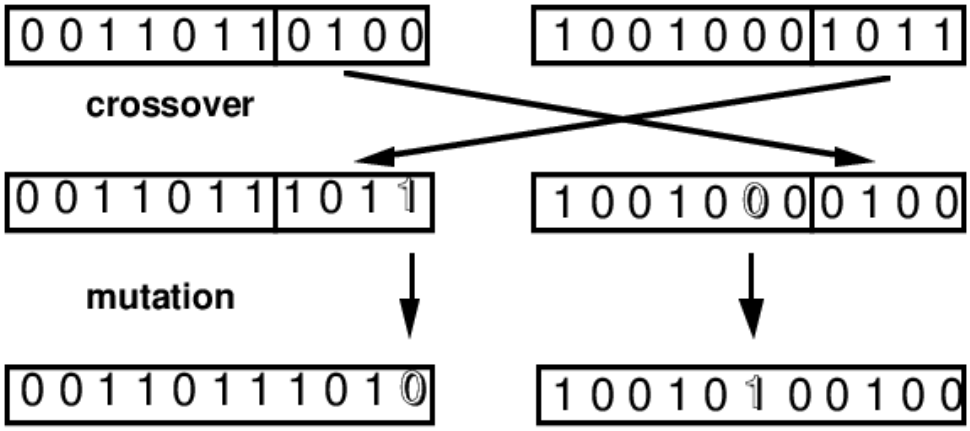
Genetikus algoritmus

Több lehetséges megoldás egyidejű karbantartása, majd azok alapján újak generálása (Autóverseny).

```
k: 0 #iterátor
P_k: kezdeti populáció #iniciális állapotok
while not exit
  P_s: szelekció(P_k)
  P_n: rekombináció(P_s)
  P_k+1: mutáció(P_n)
  k = k + 1
```

- Szelekció: fitness érték alapján egyedek kiválasztása
- Rekombináció: kiválasztott egyedek alapján új populáció készítése
- mutáció: kis valószínűséggel véletlen zaj alkalmazása egyes egyedeken





FEL