Schopnosť strojov učiť sa z dát

Základné kroky spracovania dát:

- **Čistenie dát:** odstránenie nekvalitných a nekonzistentných dát
- **Integrácia dát:** napr. Kombinácia rôznych dátových zdrojov
- **Výber dát:** výber relevantných dát z databázy
- **Transformácia dát:** preprocessing, konsolidácia a transformácia dát do podoby vhodnej pre dolovanie znalostí
- **Dolovanie znalostí:** proces aplikácie učiacich sa algoritmov za účelom extrakcie vzorov a pravidiel
- **Ohodnotenie naučeného modelu:** posúdenie kvality na základe objektívneho kritéria
- **Reprezentácia znalostí:** naučená znalosť je používateľovi vizualizovaná či prezentovaná v inej podobe

Formy dát:
-matica dát
-dokumenty - frekvencia výskytu slov v dokumente
-grafy
Vlastnosti dát:
-veľké množstvo dát, vysoká dimenzionalita dát: náročné na pamäť, spracovanie
-riedke (sparse) dáta - zaznamenaná je napr. Iba prítomnosť v danej kategórii, veľa nulových hodnôt
-pomer šumu
-body ležiace mimo prevažnu časť dát - outliers
Typy dát:
a-nominálne (nominal, categorical): binominálne, polynominálne
b-ordinálne (ordinal): nasledujúce za sebou
1-numeric: reálne, celočiselné, interval, percento
2-špeciálne: obraz, zvuk, časová postupnosť, video

Nominálne dáta: Hodnoty môžu byť názvy, možné je len rozlíšiť, či sa rovnajú. Príklad: muž/žena, psč, farba očí. Operácie: entropia, korelácia, chí kvadrát test
Ordinálne dáta: Dáta je možné porovnávať, pr. Tvrdosť materiálu (mäkký, stredný, tvrdý). Operácie: medián, percentil, korelácia
Interval: V tomto prípade je zmysluplné porovnávať rozdiely medzi hodnotami. Napr. dátumy v kalendári. Operácie: stredná hodnota, odchylka, Pearsonova korelácia, t a F test
Percento: V tomto prípade je zmysluplné porvnávať aj s inými atribútami. Pr.: teplota v kelvinoch, vek, dĺžka, elektrické napätie. Operácie: geometrická a harmonická stredná hodnota, percentuálna odchylka

Základné koncepty:

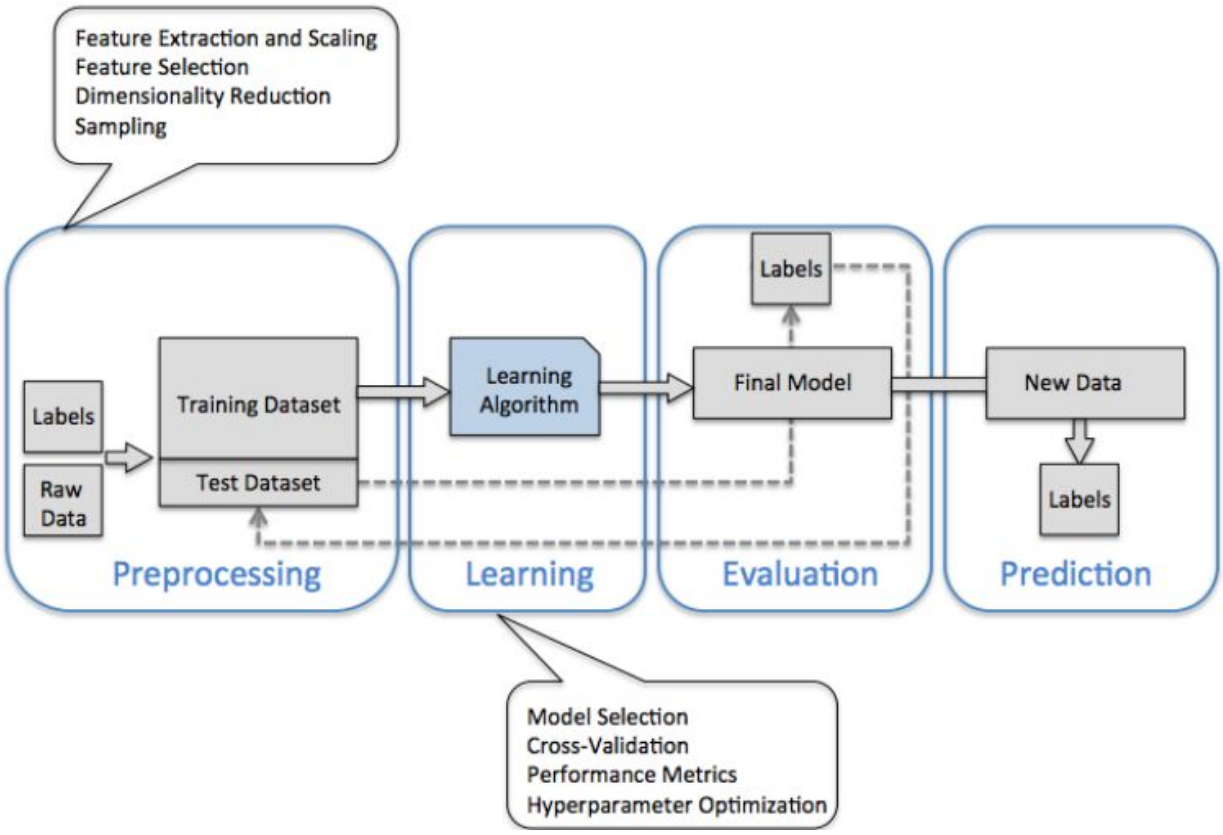
- **Učenie s učiteľom** (Supervised learning):
    - Klasifikácia, regresná analýza
    - Spam filter, rozpoznávanie čísel
- **Učenie bez učiteľa** (Unsupervised learning):
    - Klastrovanie (clustering)
    - Redukcia dimenzionality (dimensionality reduction)
- **Učenie posilňovaním** (Reinforcement learning):
    - **Agent** -- (State) --> **Environment** -- (Reward) -- (Action) --> **Agent**

Terminológia a označenie:

-vzorky (pozorovania) “uskutočnené merania, x

-príznaky (atribúty) “namerané hodnoty”, determinujú dimenzionalitu, y

-označenie triedy, cieľová premenná



Lineárna regresia: modelue lineárny vzťah medzi atribútom a cieľovou premennou

Príklad: lineárna regresia sa na základe dát naučí parametre (A, B) modelu:

$y = A + Bx$

Klasifikácia:

Príklad: predikcia druhu kvetu

1. Načítanie databázy a výber trénovacích dát (75%) vzoriek
2. Zobrazenie dát
3. Trénovanie

4. Predikcia
5. Vyhodnotenie výsledkov

Metriky pre hodnotenie kvality predikce:

Prediction: Positive

a) Target cass: Positive -> True positive (TP)
b) Negative -> False positive (FP)

Prediction: Negative

a) Target cass: Positive -> False Negative (FN)
b) Negative -> True Negative (TN)

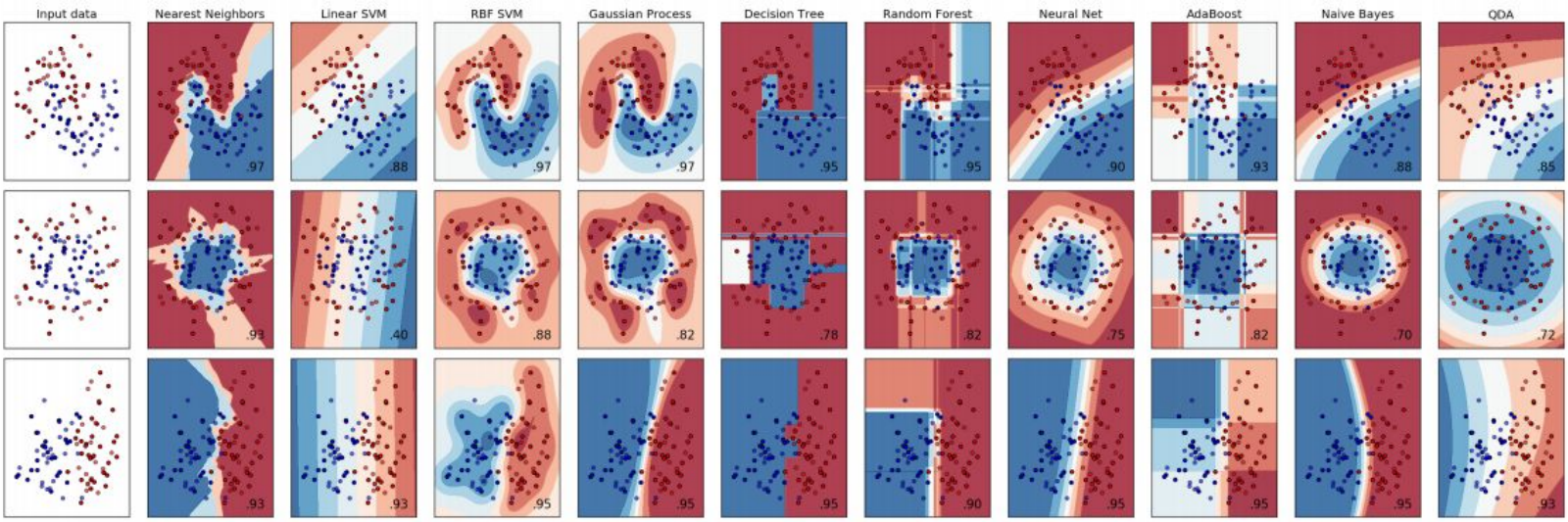
Accuracy = (TP + TN) / m
Precision = TP / (TP + FP)
Recall = TP / (TP + FN)
F1-score = 2 * Precision * Recall / (Precision + Recall)
m - počet vzoriek m = TP + FP + FN + TN

Štandardizácia dát:

-atribúty s výrazne odlišným rozsahom (0.1 - 1.5; 2000 - 8500)
-dáta nadobudnú normálne rozdelenie
- mean = 0, std = 1 per feature

$$\mathbf{x_j'} = \frac{\mathbf{x}_j - \mu_j}{\sigma_j}$$

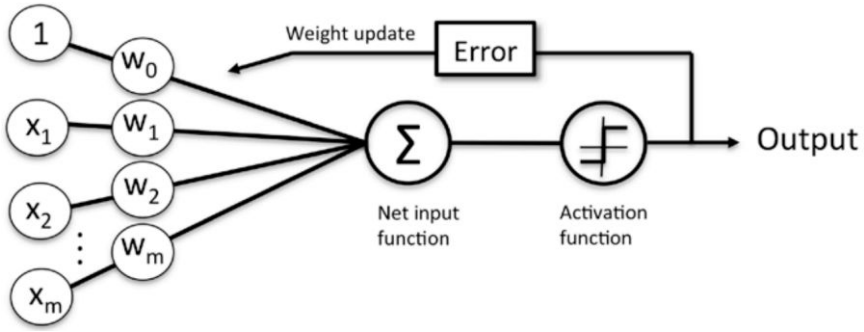
-škálovať trénovacie a testovacie dáta osobitne



Klasifikátory v scikit-learn

- **Perceptron:**

-najjednoduchšia neurónová sieť.
-je vyjadrená váhovaná kombinácia vstupov a výsledok je porovnaný s rozhodovacou úrovňou



-problém binárnej (v zmysle dvojskupinovej) klasifikácie
-aktivačná funkcia $\phi(z)$
-výstup je lineárna funkcia vstupných hodnôt

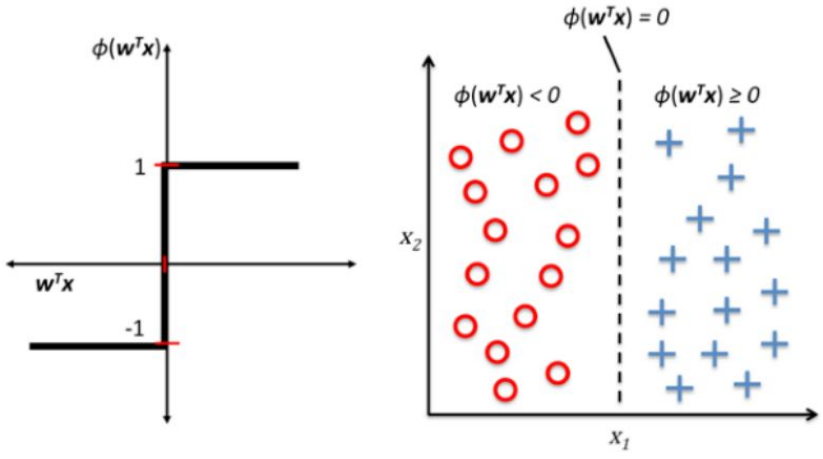
$$z = w_1 x_1 + \ldots + w_m x_m$$

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

-na základe hranice je rozhodnuté do ktorej triedy vzorka patrí

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

$$z = \mathbf{w}^T \mathbf{x}$$

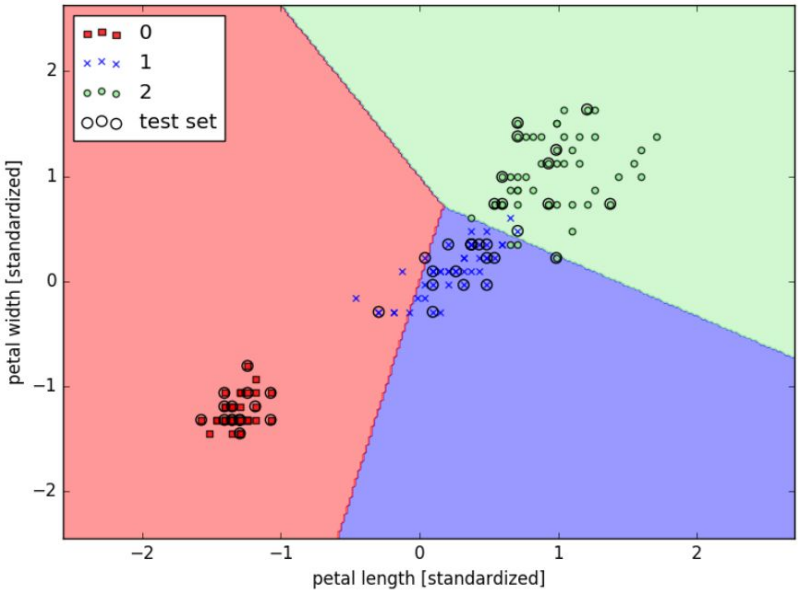


-ako získať **w** $z = \mathbf{w}^T \mathbf{x}$

1. Inicializuj váhy na nulu alebo malé náhodné číslo

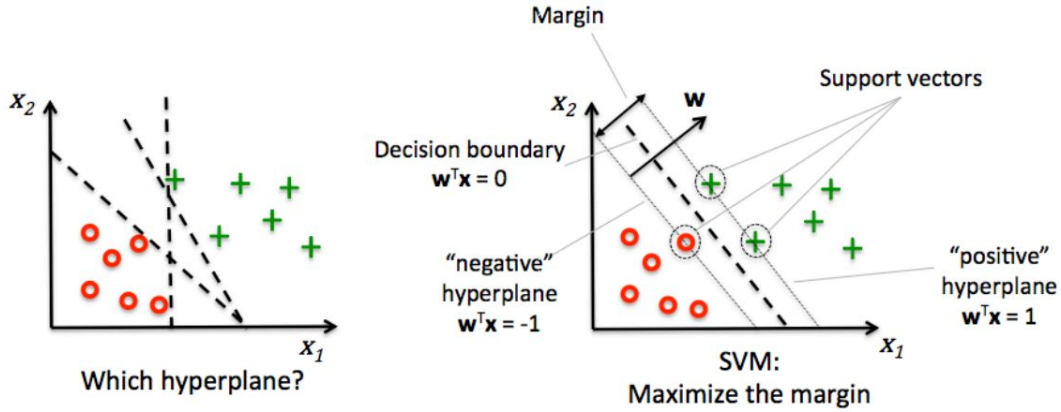
2. Pre každé trénovacie dáta $x^{(i)}$ urobte nasledujúce kroky:
   a. Vypočítajte výslednú hodnotu $\hat{y}$
   b. Aktualizujte váhy $w_j := w_j + \Delta w_j$

$$\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)}$$



- **SVM: Support vector machines, metóda podporných vektorov**

-Perception nedokáže aproximovať nelineárne funkcie, pr.: Xor funkcia
-je potrebné použiť metódy schopné modelovať lineárne neseparovateľné dáta:
-kernelizácia (projekcia do viacrozmerného priestoru
-umelé neurónové siete (univerzálny aproximátor funkcie)
-rožšírenie Perceptionu, cieľ: minimalizovať chybu
-cieľ: maximalizovať hraničné pásmo (margin) -> vzdialenosť medzi oddeľujúcimi hranicami/rovinami



$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{pos} = 1 \qquad (1)$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}_{neg} = -1 \qquad (2)$$

$$\Rightarrow \boldsymbol{w}^T \left(\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg} \right) = 2$$

margin $\dfrac{\boldsymbol{w}^T \left(\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg} \right)}{\|\boldsymbol{w}\|} = \dfrac{2}{\|\boldsymbol{w}\|}$

$$\|\boldsymbol{w}\| = \sqrt{\sum_{j=1}^{m} w_j^2}$$

cieľom je maximalizovať margin

Predpokladajme správne zaradenie vzoriek:

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 \; if \; y^{(i)} = 1$$

$$w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} < -1 \; if \; y^{(i)} = -1$$

$$y^{(i)} \left(w_0 + \boldsymbol{w}^T \boldsymbol{x}^{(i)} \right) \geq 1 \; \forall_i$$

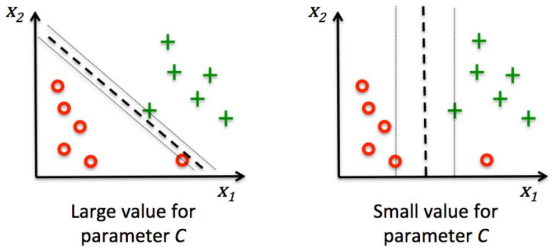
Minimalizovať: $\frac{1}{2}\|\boldsymbol{w}\|^2$.

Q: Čo v prípade ak dáta nemôžu byť oddelené rovinou/rovinami?

$$\boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 1 \; if \; y^{(i)} = 1 - \xi^{(i)}$$

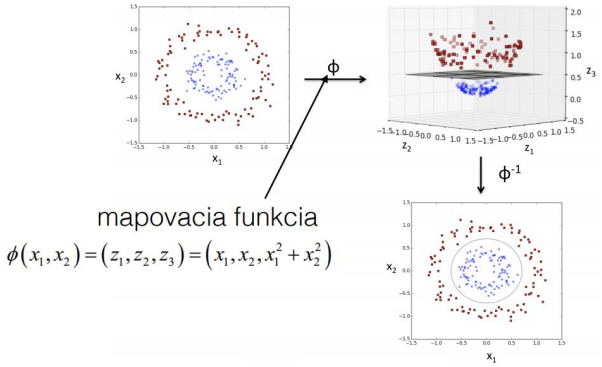
minimalizovať': $\frac{1}{2}\|\boldsymbol{w}\|^2 + C\left(\sum_i \xi^{(i)} \right)$

A: Slack "voľný" parameter $\boldsymbol{w}^T \boldsymbol{x}^{(i)} < -1 \; if \; y^{(i)} = 1 + \xi^{(i)}$



-kernelizácia SVM

-kernelizácia umožňuje riešiť problémy nelineárnej klasifikácie



-nové príznaky - výpočtovo náročné -> kernel trick

Parametre:

slack parameter

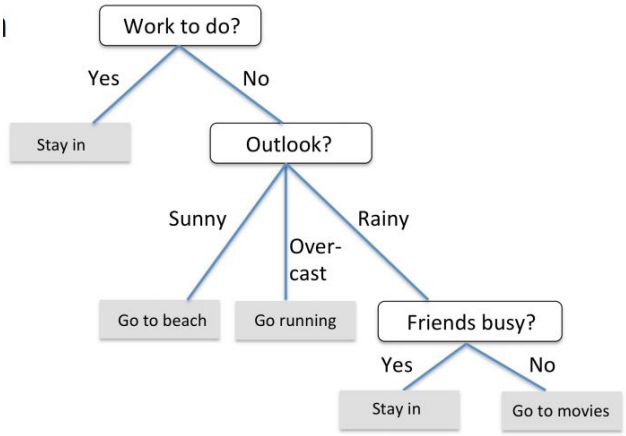
parameter jadier (kernels):

-radial basis function kernel: gamma

- **Rozhodovacie stromy (decision trees):**

-výhoda: interpretovateľnosť

-dobre zvládajú zmiešané numerické a kategorické atribúty



-optimálny vs. “dostatočne dobrý” rozhodovací strom

Návrh r. stromu:

-začiatok - koreň (root)

-vetvy: správne otázky - veľa nových informácií

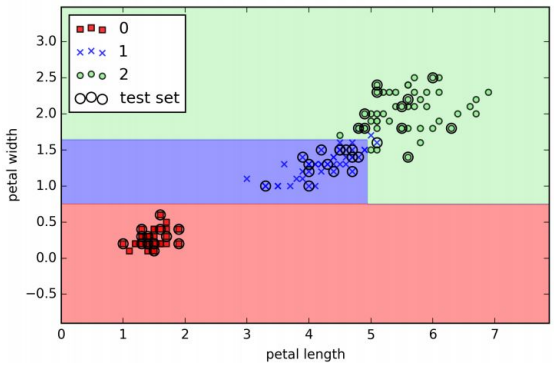
-entropia $H(S) = -p_1 \log_2 p_1 - \ldots - p_n \log_2 p_n$

-vysoká entropia - dáta distribuované vo viacerých triedach

Rozhodovacie regióny

-škálovanie

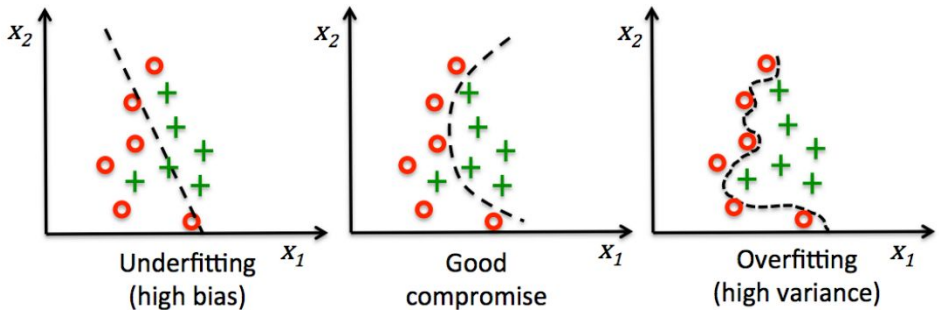
-zlozitosť rozh. hraníc



- **Náhodné lesy:**

-ensemble (skupinový klasifikátor) - kombinuje výsledky viacerých “slabších” klasifikátorov

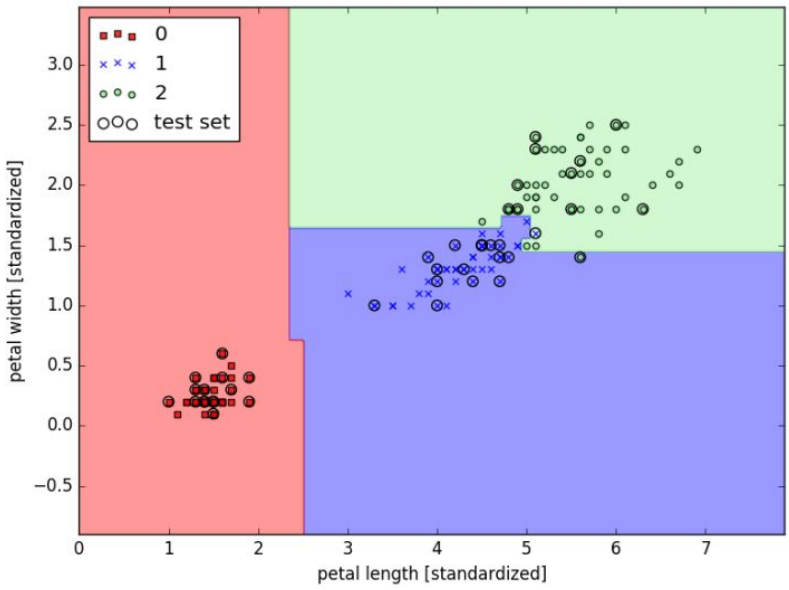
-obmedzuje slabinu DT - pretrénovanie (overfitting)



Algoritmus:

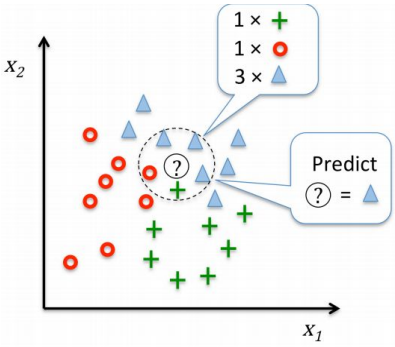
1. Výber podmnožinu dát zo skupiny trénovacích dát (bootstrap)
2. Aplikovať rozhodovací strom na vybranú podmnožinu (bootstrap sample)
3. Opakovať krok 1 a 2 N krát
4. Agregovať výsledky jednotlivých stromov (alg.: pre algregáciu: majority voting)

- nevyžaduje až také náročné nastavovanie parametrov ako DT
- menšia možnosť pretrénovania, vyššia výpočtová náročnosť



- **K najbližších susedov:**

-lenivý (lazy) prístup: nesnaží sa naučiť funkciu rozdeľujúcu dáta, ale dáta samotné
-nenapovedá prečo



Algoritmus:

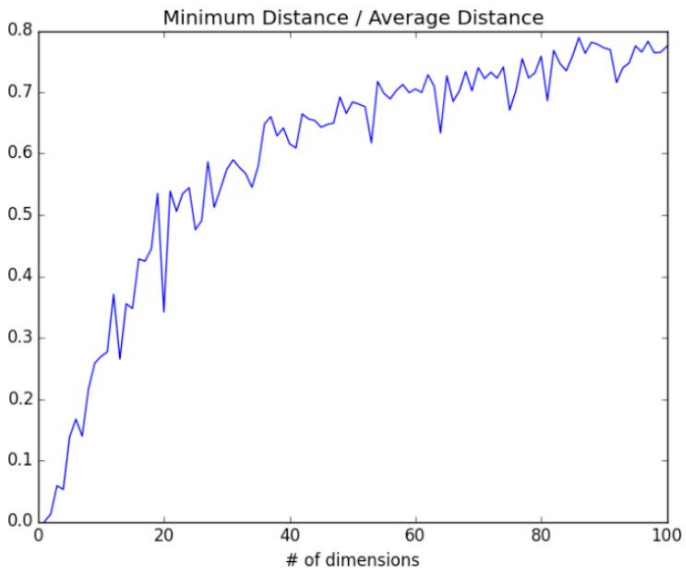
1. Vyber hodnotu *k* a mieru vzdialenosti (euklidovská, manhattan, minkowski a i.)
2. Nájsť *k* najbližších susedov k vzorke, ktorú spracúvavame
3. Priradiť označenie triedy na základe majoritného rozhodnutia

-*k* ↑ výpočtová náročnosť ↑
- rovnováha medzi pretrénovaním a podtrénovaním

-citlivý na “kliatbu veľkých rozmerov” (curse of dimensionality)

Curse of dimensionality

-dáta sú vysokej dimenzionality, i.e.: vysoký počet príznakov.
Zvyčajne *N_features >> N_samples*



Dôsledky:
-zvýšené nároky na uložný priestor
-zvýšené nároky na výpočtovú náročnosť (klasifikátor)
-výrazne horšia interpretovateľnosť
-pretrénovanie klasifikátora (KNN)
-narastá vzdialenosť medzi jednotlivými dátovými bodmi

No Free Lunch Theorem

Model je založený na zjednodučení reality → zjednodušenie je založené na predpokladoch → predpoklady neplatia vždy a všade

Predspracovanie dát

-príprava dát pred aplikáciou predikčného algoritmu
-oboznámiť sa s dátami: formát, chýbajúce hodnoty?, diskrétne vs spojité hodnoty, vizualizácia dát

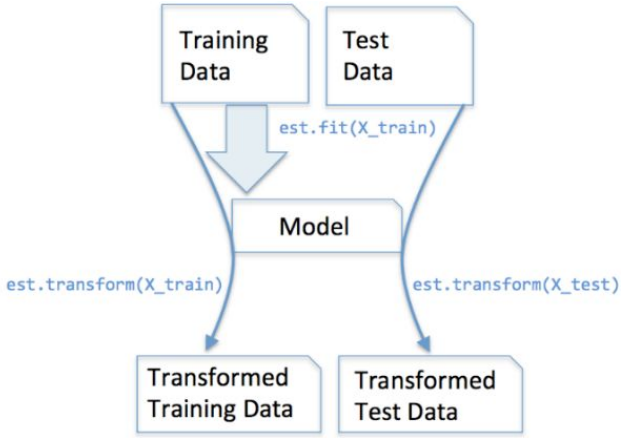
Chýbajúce dáta
-Not a Number (NaN)

- **Eliminácia vzoriek alebo atribútov s chýbajúcimi dátami**

-pandas - metódy dataFrame pre operácie s chýbajúcimi hodnotami
-pri eliminácii dochádza k nenávratnej strate dát/informácií

- **Imputácia chýbajúcich hodnôt**

-dopočítanie chýbajúcej hodnoty
-trieda *Imputer* v *scikir-learn*
Imputácia na základe ostatných hodnôt daného atribútu (stĺpca) - mean, median
-metódy fit a transform aplikované na dáta rovnakých rozmerov



Zaobchádzanie s kategorickými dátami
-numerické
-kategorické
-ordinálne (S, M, L, XL, XXL)
-nominálne (červený zelený, modrý, čierny)
-mapovanie ordinálnych atribútov na numerické (S: 1, M: 2, L:3, XL: 4, XXL: 5)
-pandas map metóda
-mapovanie nominálneho označenia tried
“chorá” 0, “zdravá” 1
“pop” 0, “rock” 1, “classical” 2 -- > irelevantné poradie

-mapovanie nominálnych atribútov
červený, zelený, modrý -> problém je zelený > červený?
-> riešenie: one hot encoding

f_size	f2_price	f_new_black	f_new_green	f_new_red
L	10	0	1	0
S	7	1	0	0
XL	15	0	0	1

Detekcia odľahlých hodnôt (outlier)

-cieľom môže byť:

-nájsť takéto hodnoty (fraud detecion)

-vyčistiť dáta od týchto hodnôt -> strata informácie

Diskretizácia

Rozdelenie dát na trénovacei a testovacie

-trénovacie: natrénovanie modelu, naladenie parametrov

-testovacie: finálne overenie

-vo fáze trénovania nemôžu byť testovacie dáta nijak použité = nikdy sa nedotýkaj testovacích dát!

-pomer rozdelenia: 80:20, 90:10, 60:40

Škálovanie

-výrazne zlepšuje správanie učiacich sa a optimalizačných algoritmov (výnimkou sú napr. rozhodovacie stromy, naopak KNN citlivé)

-normalizácia (normalisation)

-štandardizácia (standardisation)

Normalizácia

-[0,1]

- $x_{norm}^{(i)} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}$ min-max škálovanie (MinMaxScaler)

Štandardizácia

$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$ (StandarScaler)

Fit -> trénovacie

Transform -> trénovacie

Transform -> testovacie

Nikdy nie Fit -> testovacie

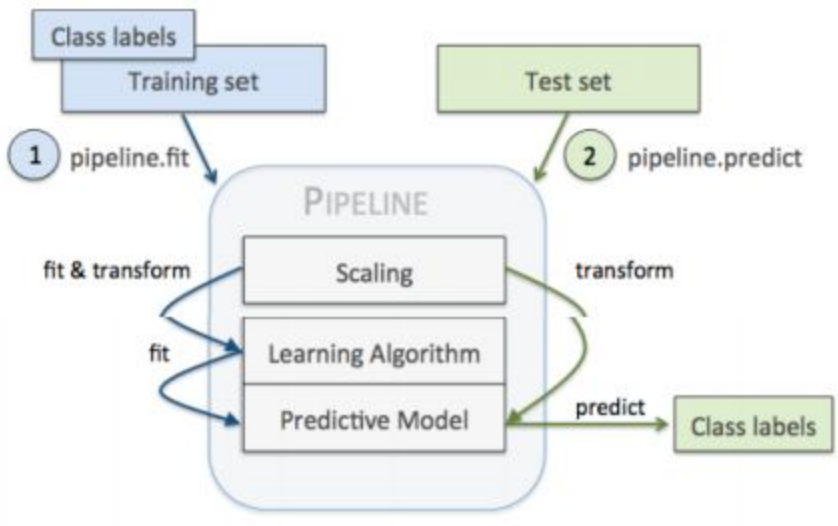
input	standardized	normalized
0.0	-1.336306	0.0
1.0	-0.801784	0.2
2.0	-0.267261	0.4
3.0	0.267261	0.6
4.0	0.801784	0.8
5.0	1.336306	1.0

Metodológia evaluácie modelov a ladenia parametrov

Pipeline

-špecifické pre scikit-learn

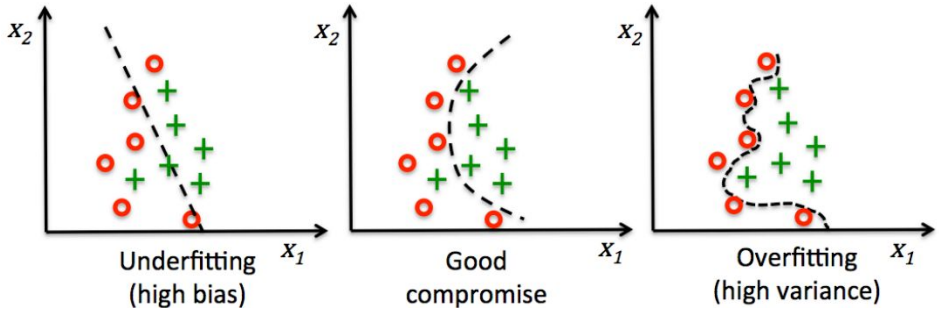
-"zgrupenie" niekoľkých krokov



Validácia modelu

-podtrénovanie (vysoká odchýlka) → výsledok príliš jednoduchého modelu

-pretrénovanie (vysoký rozptyl) → dôsledok modelu príliš naviazaného na trénovacie dáta



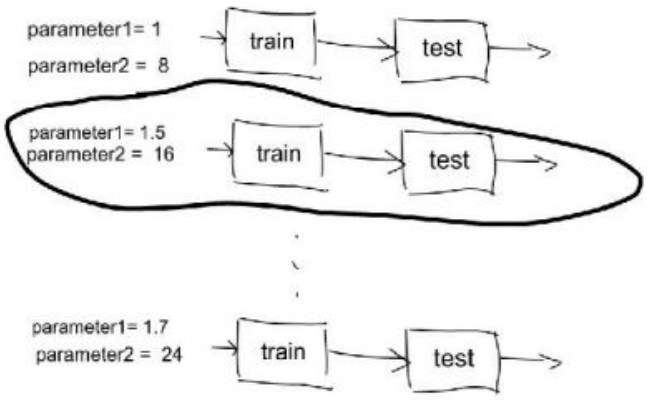
- **Hold-out metóda**

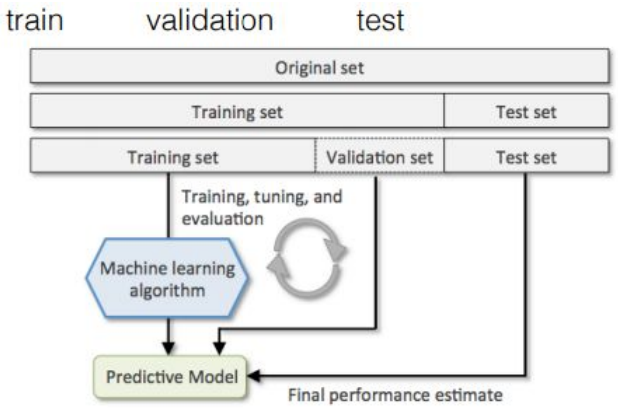
Zaužívaný (metodologický nie úplne správny prístup): train test

-výber modelu (model selection) - ladenie parametrov

-testovacie dáta sú nepriamo súčasťou natrénovania modelu → hrozba pretrénovania

-pridáme validačné dáta





-citlivé na rozdelenie dát na train/validation/test

- **Metóda krížovej validácie (cross-validation)**

-cross-validation without replacement

k-fold cross-validation

-tradičná hodnota k = 10

-v prípade veľkých dátových sád k = 5

-k↑ viac dát na trénovanie (menšie dtb), dlhšie trvanie

-špeciálny prípad leave-one-out cross-validation k=N

Stratified *k*-fold cross-validation

-vylepšenie, jednotlivé podskupiny sú vzorkované tak aby zastúpenie vzoriek v podskupinách zodpovedalo ich zastúpeniu v celej dátovej sade

Diagnostika pomocou kriviek učenia sa a validácie

-krivka učenia sa (learning curve)

-krivka validácie (validation curve)

Krivka učenia sa

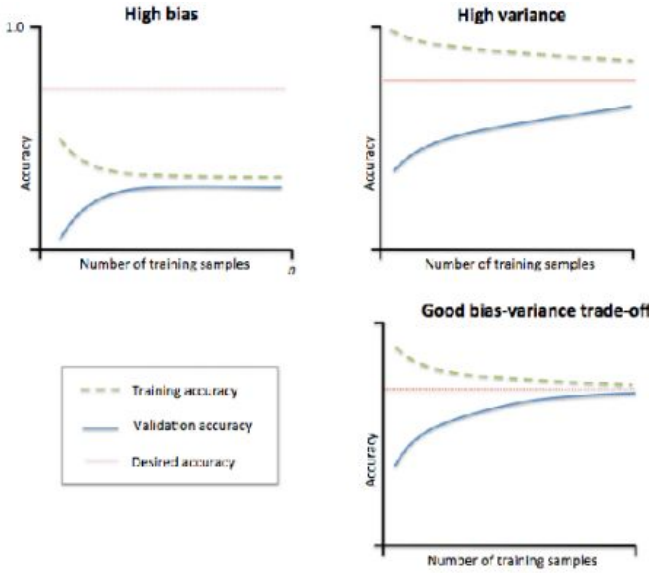
high bias (odchýlka), variance (variancia/rozptyl)

-získať viac dát

-drahé, nie vždy pomôže

Underfitting → pridať atribúty (high bias)

Overfitting → znížiť komplexitu modelu, viac dát, odobrať atribúty



-krivka učenia sa v scikit-learn:

-presnosť predikcie ako funkcia veľkosti vzorky dát

-na krivke mean a std, ilustrovať

Krivka validácie

-presnosť predikcie ako funkcia parametrov modelu

Ladenie parametrov modelu

-rozlišujeme dva typy parametrov

-parametre modelu naučené z dát

-parametre modelu nastaviteľné - optimalizované separátne

-validačná krivka

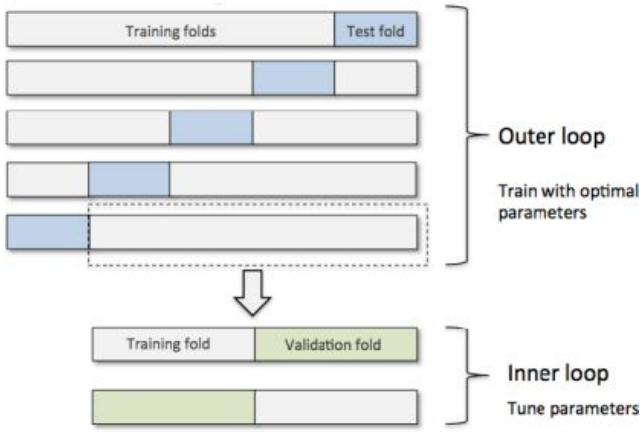
-grid search

-spôsob hľadania optimálnych parametrov

-brute-force prístup

Cross-validácia + ladenie parametrov prehľadávaním gridu = vnorená cross-validácia

- pre väčšie dátové sady



Metriky pre evaluáciu výkonnosti:
-Matica zámien (confusion matrix)

		Predicted class: P	Predicted class: N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

$$ERR = \frac{FP + FN}{FP + FN + TP + TN}$$ **chyba**

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR$$ **presnosť**

Metriky vhodné pre nevyvážené triedy v dátach

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$ **pomer nesprávnych pozitívnych** - false positive rate

$$TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$ **pomer správnych pozitívnych** - true positive rate

Precision, recall

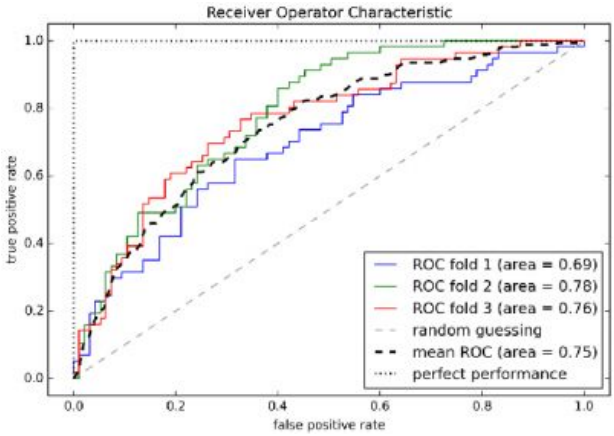
$$PRE = \frac{TP}{TP + FP}$$

$$REC = TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$

F1-skóre

$$F1 = 2\frac{PRE \times REC}{PRE + REC}$$

Operačná charakteristika prijímača / receiver operator characteristic (ROC)



Učenie bez učiteľa

Klastering

Klastering (analýza klastrov) je metodológia, ktorá umožňuje nájsť skupiny podobných objektov; objektov, ktoré sú si podobné viac ako objekty v iných skupinách

Aplikácie:

-vytváranie skupín, hudby, filmov rovnakého žánru

-hľadanie skupín zákazníkov s podobným správaním (diaper story)

- **K-means**

K-means (nájdenie centier podobnosti)

-klastrovanie založené na prototypoch - klastre sú reprezentované prototypmi

centroid - priemer podobných objektov (spojité hodnoty)

medoid - najreprezentatívnejši z najčastejšie sa vyskytujúcich objektov (kategorické hodnoty)

-je potrebné špecifikovať počet klastrov

Algoritmus:

1. Náhodne vybrať *k* centroidov
2. Prideliť každú vzorku k najbližšiemu centroidu
3. Presunúť centroid do centra vzoriek, ktoré mu boli priradené
4. Opakovať kroky 2 a 3, kým sa neprestane meniť rozmiestnenie klastrov, alebo kým nie je dosiahnutá špecifikovaná podmienka, prípadne počet iterácií algoritmu

Podobnosť medzi objektami je meraná vzdialenosťou (napr. Euklidovskou, existujú aj ďalšie vzdialenosti - minkowski, manhattan, ...)

K-means - optimalizačný problém, minimalizácie sumy štvorcov chýb v rámci klastra

$$SSE = \sum_{i=1}^{n}\sum_{j=1}^{k} w^{(i,j)} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{\mu}^{(j)} \right\|_2^2$$

Elbow method - lakťová metóda

-v porovnaní s učením s učiteľom, nepoznáme skutočnú triedu vzoriek

-určenie počtu klastrov (niektoré metódy to dokážu samé)

-grafické využitie klastrovej SSE

- **DBSCAN**

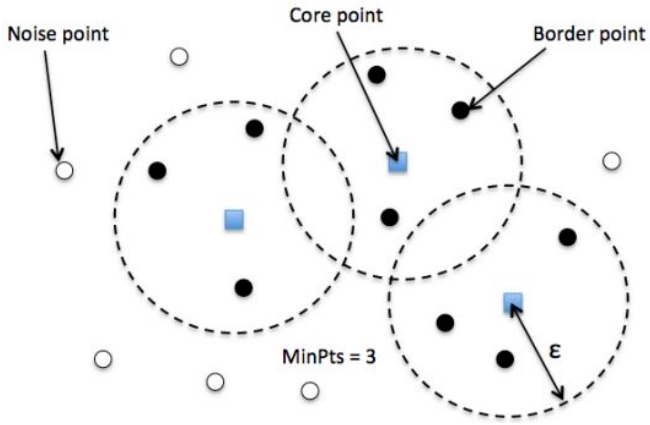
Density-based Spatial Clustering of Applications with Noise

-pojem hustoty (density) je v tomto prípade definovaný ako počet bodov vo vzdialenosti s polomerom E

Algoritmus:

1. Označenie jednotlivých bodov:
   a. Hlavný (core) bod - minimálne MinPts bodov je vo vzdialenosti danej polomerom E
   b. Hraničný (border) bod - má menej ako MinPts susedných bodov, avšak je vo vzdialenosti <E od hlavného bodu
   c. Ostatné body - šum
2. Vytvoriť klaster pre každý hlavný bod, prípade pre skupinu hlavných bodov ak ich vzdialenosť <E

3. Priradiť hraničné body k zodpovedajúcim klastrom, podľa ich hlavného bodu



Výhody: nepredpokladá guľatý tvar klastrov ako K-means
Nevýhody:
-trpí kliatbou dimenzionality (ako všetky alg. využívajúce euklidovskú vzdialenosť)
-je potrebné správne nastaviť parametre MinPts a E

Unsupervised dimensionality reduction

Pretrénovanie: ako vyriešiť?
-získať viac dát
-zvoliť jednoduchší model
-redukovať dimenzionalitu dát

Redukcia dimenzionality vytvorením nového priestoru atribútov
Kompresia dát pri zachovaní informácie
Principal component analysis (PCA)
-metóda učenia bez učiteľa pre lineárnu transformáciu dát
-hľadá smery maximálneho rozptylu vo vysokodimenzionálnych dátach s následnou projekciou do priestoru nižších rozmerov

Konštrukcia **W** (rozmeru *dxk*)
-mapuje vektor príznakov **x** do nového *k* rozmerného priestoru
k < *d, d*-rozmer pôvodného priestoru príznakov

$$\boldsymbol{x} = [x_1, x_2, \ldots, x_d], \quad \boldsymbol{x} \in \mathbb{R}^d$$

$$\downarrow \boldsymbol{x}\boldsymbol{W}, \quad \boldsymbol{W} \in \mathbb{R}^{d \times k}$$

$$\mathbf{z} = [z_1, z_2, \ldots, z_k], \quad \mathbf{z} \in \mathbb{R}^k$$

Základný postup pri PCA:
1. Štandardizácia dátovej sady
2. Konštrukcia kovariančnej matice
3. Dekompozícia kovariančnej matice na vlastné vektory a vlastné hodnoty matice
4. Výber *k* vlastných vektorov, zodpovedajúcich *k* najväčším vlastným číslam
5. Konštrukcia **W** využitím vybraných vlastných vektorov
6. Transformácia *d* rozmernej dátovej sady do *k* rozmerného priestoru

Dôsledok: nižší počet príznakov, nové príznaky

Úvod do AI + vyhľadávanie bez informácie

AI história:

1982: R1 - AI navrhuje komponenty počítačov na základe požiadavko zákazníkov
1981: “Fifth generation” - inteligentné počítače, umožňujúce hľadať riešenie bez zásahu programátora. Programátor nemusí hľadať optimálny algoritmus, iba špecifikuje podmienky - Prolog
1986: neurónové siete so spätnou väzbou (neural networks with back-propagation) : súčasnosť - validácia dát, operačný výskum, burza, risk manažment a iné
1995: inteligentní agenti: opozícia voči klasickým optimalizačným prístupom. Prístupy ako rozpoznávanie reči, detekcia a predpovede sú nedokonalé a istá miera inteligencie je vyhnutá. Plánovanie a rohodovanie agentov má potenciál zlepšiť túto situáciu (prevádzka letísk, modelovanie dopravy, finančná kríza 2008)
2001: mnohorozmerné databázy
1996 Kasparov vs Deep blue → Kasparov vyhráva, 1997 Kasparov prehráva

Agent
Agent je entita, ktorá vníma a reaguje na zmeny v stavovom priestore pomocou senzorov (vnemov) a ovplyvňuje prostredie prostredníctvom akcií

Agent based vs multi-agent?

Príklady:
Človek: oči, uši, nos → ruky, nohy, ústa
Robot: kamera, IR, detektor → ?
Software: klávesnica, pakety zo siete → obrazovka, pakety do siete

Stavový priestor (agentový priestor)
-Single-agent vs. multi-agent
    Kooperácia vs kompetetívnosť
-Deterministický vs. stochastický priestor
    Ďalší stav prostredia je plne určený súčasným stavom agenta a jeho akciou
-Statický vs dynamický priestor
    Statické prostredie sa nemení, kým agent rozmýšľa čo urobiť
    Semi-dynamické: prostredie sa s časom nemení, ale miera výkonu áno
-Diskrétny vs. spojitý priestor
-Úplný vs. čiastočný (Fully observable vs. partially observable)
    Senzory poskytujú úplný obraz o agentovom prostredí

Agent

Agent vníma prostredie a na základe internej logiky (tabuľky, prípadne sady inštrukcií) na neho reaguje

Formálne vieme každého agenta popísať agentovou funkciou (tabuľkou):

V* → A, kde V: množina vnemov, A: je množina akcií

Tabuľka: abstraktný matematický popis agenta

Príklad:

Agent vysávač

Vnemy: miesto (A, B), obsah (čisto, špina)

Akcie: vysávaj, doľava, doprava, nič?

Agentova tabuľka:

Zoznam vnemov	Akcia
(A, Čisto)	doprava
(A, špina)	vysávaj
(B, čisto)	doľava
(B, špina)	vysávaj

Reakcie agenta: pamäťový, resp. bezpamäťový (myopic agent)

- **Reflexný agent**, cieľový agent, úžitkový agent, učenlivý agent
- **Racionálny agent** - je nutné zadefinovať požiadavky a tzv. Úžitok, resp. Výkon
  Príklad predpoklady:
  - Agentový priestor je známy. Časová premenlivosť špiny nie
  - 3 akcie: vľavo, vpravo a vysávaj
  - Agent identifikuje miesto a špinu → je racionálny?
  - Pridáme penalizáciu za pohyb → je racionálny?

Racionalita agentov

Agent je **racionálny** ak:

- Ciele sú reprezentované formou úžitku
- Svoje akcie volí tak, že maximalizuje pravdepodobnosť dosiahnutia špecifického cieľa
- Byť racionálnym znamená, že agent maximalizuje svoj **očakávaný úžitok** (platí v **stochastických prostrediach**) prípadne svoj **úžitok** (platí v **deterministických prostrediach**)

Príklad: autonómny vodič taxislužby

Agent	**úžitok**	**prostredie**	**akcie**	**senzory**	
Taxi služba	bezpčnosť	cesta		brzda	kamera
	Rýchlosť	doprava		volant	sonar
	Legálnosť	Chodci	Plyn	Rýchlomer	
	Pohodlie	Zákazník		Smerovka	GPS
	Zisk			Húkačka	

Agentový priestor

Príklad

Priestor	Pozorovateľnosť	Determinizmus	Statickosť	Diskrétnosť	Agenti
Križovka	Úplná	Deterministické	Statické	Diskrétne	Jeden
Šach	Úplná	Deterministické	Semi	Diskrétne	Multi
Taxi	Čiastočná	Stochastické	Dynamické	Spojité	Multi
Atď	…	…	…	…	…

Vyhľadávanie (search problems)

-Typy agentov

-Proces vyhľadávania (search)

-Vyhľadávanie bez informácie

-Depth-First Search

-Breadth-First Search

-Iterative deepening

-Uniform-Cost Search

Reflexný agent

-volí akciu na základe súčasného stavu sveta

-nepozerá na dôsledok akcie a interakciu s prostredím

Cieľový (úžitkový) agent

-plánuje v budúcnosti pre potreby dosiahnutia cieľa (offline plánovanie)

-začiatočný stav (Start state) a cieľový test (Goal test): nerozlišuje medzi ostatnými stavmi (**cieľový agent**)

-Aký je úžitok z danej akcie (**evaluačná funkcia**)

-Sekvencia rozhodnutí a korešpondujúci úžitok (**úžitkový agent**)

-Kompletné riešenie: alogirtmus nájde riešenia, ak existuje. Nemusí byť optimálne (napr. v počte krokov)

-**Optimálne riešenie:** algoritmus nájde optimálne riešenie

-Časová zložitosť: Ako dlho trvá hľadanie riešenia

-**Priestorová zložitosť:** Koľko pamäti potrebujeme na dosiahnutie cieľa

Vyhľadávanie agenta v priestore

-Vyhľadávanie (**search problem**) pozostáva zo:

- Stavový priestor
- Akcie agenta v danom stave (a jej váhy) (**successor functions**) a ich dopad
- Začiatočný a konečný stav agentat (**goal and end state**)

-**Riešenie:** postupnosť akcií vedúce agenta smerom zo začiatočného ku konečnému stavu v agentovom priestore

Príklad Cestovanie v Rumunsku

-Agentový priestor: Mestá (v ktorých sa nachádza agent). Prípadne GPS

-Akcia: Kam smeruje agent z daného mesta (stavu) + vzdialenosť (váha) → benzín, komfort jazdy

-Začiatočný stav: mesto Arad

-Konečný stav: Je mesto Buchurest?

-Riešenie: závisí od algoritmu (kompletnosť, optimalita)

Priestor (world state)
→ zahŕňa všetky detaily “agentového sveta”
Stavový priestor (state space) zahŕňa len atribúty nevyhnutné pre dané riešenie

Problém: Potrava (Eat-all-dots) Pacman
Stavy: {(x, y), boolean potrava}
Akcie: North, South, East, West
Successor: nová súradnica + boolean potrava (true → false)
Goal test: všetky potravy false (veľké guľky na istý čas vystrašia duchov)
Nutné špecifikovať: pozície agentov (pacman, duchovia), pozície potravy + veľkých guličiek, zostávajúci čas vystrašenia duchov

→ **Stavový priestor vo forme grafu**: matematická reprezentácia vyhľadávacieho problému:
Vrcholy grafu sú stavy stavového priestoru
Hrany reprezentujú akciu (s váhami)
Cieľ je tiež vrchol grafu (1, či viaceré)

-V grafe sa každý stav vyskytuje len raz
→ **Vyhľadávanie v strome:**
Strom:
-súslednosť stavov a akcií agenta
-začiatočný stav agenta - koreň stromu (je to konečný stav?)
-Ak nie, expandujeme daný stav, čím generujeme novú množinu stavov
-Vyberáme ďalší stav pre pokračovanie vo vyhľadávaní pomocou vyhľadávacej stratégie
-**Uzol stromu-** dátová štruktúra, charakterizuje plán hľadania cesty zo začiatočného stavu do stavu pred expanziou
-**Pre väčšinu problémom je nereálne postaviť celý strom**

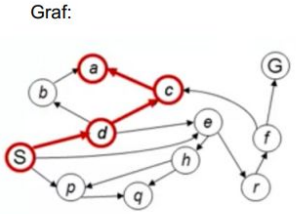
Stavový priestor reprezentovaný grafom nie je rovnaký ako reprezentovaný stromom. Stav v grafe nie je uzol stromu.
Uzol stromu obsahuje:

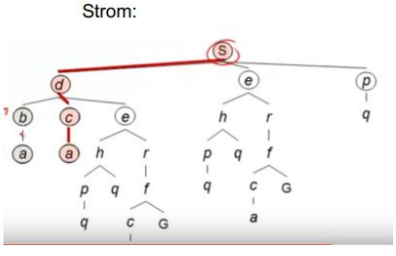
- Aktuálny stav
- Odkaz na rodičov
- Akciu, ktorá viedla od rodičov do aktuálneho stavu
- Váhu cesty
- Hĺbku

Stavový graf vs. strom
Uzol v strome- plán akým sa agent dostal do daného stavu zo začiatočného stavu
Snaha - minimalizácia expandovaných uzlov
Vyhľadávanie v strome
-Uzly stromu: plánovanie agenta (začiatočný stav + akcie = stav)
-**Zásobník:** jednotlivé uzly (**leaf nodes**) určené na expanziu
-**Cieľ:** minimalizovať čas a pamäťové nároky

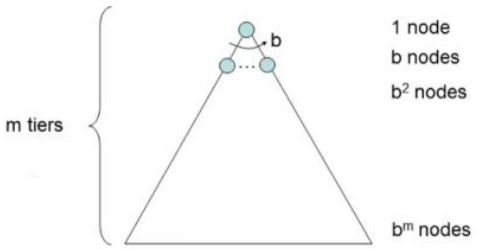
- **Depth-First Search**

Stratégia: expanduje najhlbší uzol
Zásobník: LIFO
Je kompletný



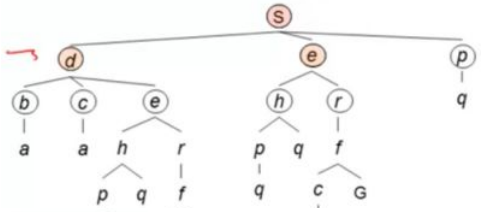


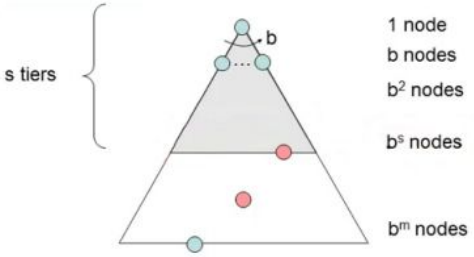
Nie je optimálny
Časová zložitosť: $O(b^m)$
Priestorová zložitosť: O(bm)
→ stačí si pamätať jednu cestu
z koreňa spoločne so susediacimi uzlami
Charakteristiky:
-**b** - vetviaci faktor
-**m** - maximálna hĺbka stromu
-**Goal state** sa môžu nachádzať na rozličných úrovniach
-Počet stavov v strome: $1 + b + b^2 + \dots + b^m = O(b^m)$



- **Breadth-First Search**

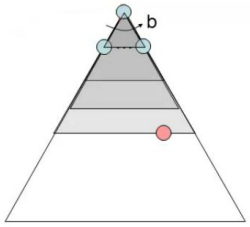
Stratégia: zo zásobníka vyberaj **najplytšiu možnosť**
Zásobník: FIFO
Expanduje všetky uzly nad konečným stavom
Nech *s* je hĺbka konečného stavu (najplytšieho)
Priestorová náročnosť: $O(b^s)$
Časová náročnosť: $O(b^s)$
Kompletnosť: Ak *s* je konečné, tak áno
Optimálnosť: Ak jednotlivé akcie majú rovnakú váhu





- **Iterative deepening**

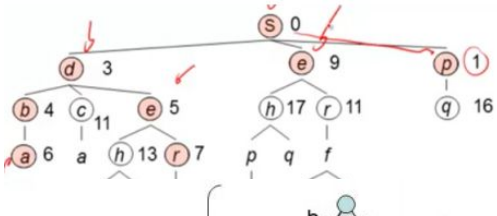
Nápad: skĺbiť DFS a BFS
DFS s hĺbkou 1
DFS s hĺbkou 2. Riešenie?
DFS s hĺbkou 3. Riešenie?

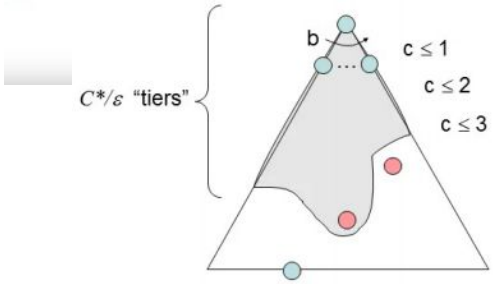


DFS a BFS sú slepé algoritmy. Neberú do úvahy váhy jednotlivých akcií

- **Uniform-Cost Search**

Stratégia: expanduj najlacnejší nód
Zásobník: najnižsie váhy. Ak kumulatívna váha cesty k riešeniu je C* a váha jednej hrany je aspoň ε, potom platí, že:
Časová náročnosť $O(b^{C^*/\varepsilon})$
Pamäťová náročnosť $O(b^{C^*/\varepsilon})$
Je kompletný pokiaľ náklady nie sú záporné
Je optimálny





Kompletný a optimálny:
-Všetky smery v strome sú prehľadávané
-Žiadna informácia o cieli

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Vyhľadávanie s informáciou

- Metrika s heuristikou
- Greedy search (best search)
- A* search

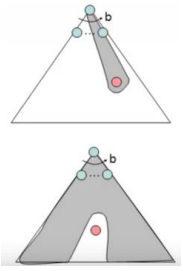
Vyhľadávanie (search problem)

-Vyhľadávanie:
    -Stavy (states)
    -Akcie a ich váhy
    -Začiatočný stav a konečný stav
-Strom (search tree)
    -Uzly stormu - plán pre dosiahnutie cieľa
    -Plány majú akumulované váhy (cost)
-Vyhľadávací algoritmus:
    -Buduje strom (nie celý)
    -Zásobník (DFS, BFS)

Heuristická informácia:

-Metrika, ktorá odhaduje vzdialenosť k cieľu (ku goal state)
-Špecifická pre vyhľadávací problém
-Euklidovská vzdialenosť, Manhattan metrika
    Príklad Rumunsko: Vzdušná čiara mesta do Bukurešte
    Príklad Palacinky: Najväčšia palacinka, ktorá nie je umiestnená dobre

- **Greedy Search**



Stratégia: expanduj nód o ktorom si **myslíš,** že je najbližší: urob to na základe heuristickej informácie
Poznámka: Ak je heuristika zle zvolená, môžeme sa dostať až k zle expandovanému DFS

- **A* Search**

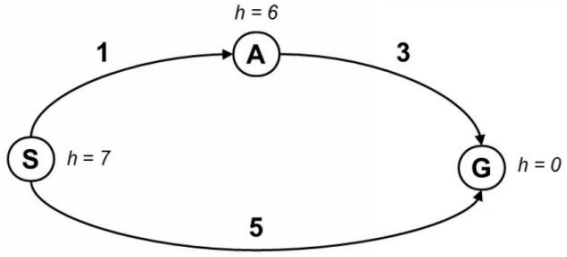
UCS expanduje stavy na základe váh akcií (g(n)) v pláne
Greedy search expanduje stavy na základe heuristickej funkcie (h(n))
A* Search f(n) = g(n) + h(n) … suma predchádzajúcich

Optimalita A*

Odhad (heuristická funkcia) musí byť vždy menší ako váha

Prípustnosť heuristiky (Admissibility)

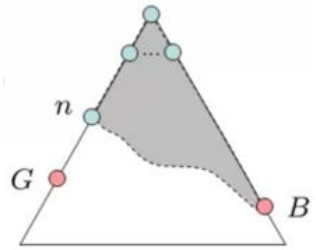
Pesimistická heuristika: (t.j. > váhy) nevyberá zo zásobníka dobré plány



Optimistická heuristika (t.j. > váhy) nikdy nepresahuje reálne náklady cesty

Heuristika *h(n)* je prípustná (optimistická), ak:
0 <= *h(n)* << *h*(n)*, *kde h*(n)* sú reálne náklady (váhy) do cieľa (goal state)
-Návrh dobrej heuristiky je kritickým prvkom pri A* search

Optimalita A* search
Uvažujme:
-G je optimálny goal state
-B je suboptimálny
-h je prípustná (optimistická)
Tvrdenie: G bude vybrané zo zásobníka skôr ako B

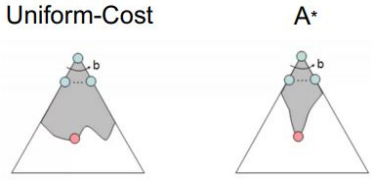


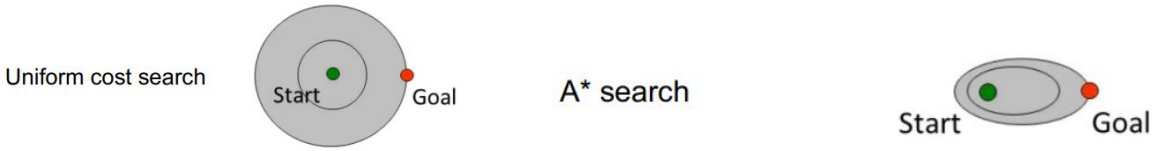
Dôkaz:
Predpokladajme, že B je v zásobníku
V zásobníku je aj predchodca G (stav n)
Tvrdenie: *n* bude expandované pred B

1. *f(n)* <= *f(G),* pretože h je prípustné
2. *f(G)* <= *f(B),* pretože B je suboptimálne
3. Všetci predchodcovia G sú expandovaní pred B
4. G je expandované pred B

f(n) <= *f(G)* <= *f(B)*

UCS vs A*
Uniform-Cost: expanduje všetky nódy: *g(n)* <= *C**
A*: expanduje všetky nódy: *f(n)* = *g(n)* + *h(n)* <= *C**
Tvrdenie: Ak h je prípustná, A* expanduje menej nódov na dosiahnutie cieľa





Využitie A*
Starcraft, Smerovacie protokoly, Pohyby robota, Spracovanie reči

Vytváranie prípustných heuristík
-Návrh nových heuristických metrík
-Jedna z možností - návrh z heuristických metrík zo stavového priestoru s nižším počtom ohraničení (zjednodušený problém - relaxed problem)

Príklad: 8 puzzle
-Heuristika: počet dlaždíc, ktoré sú out of position
-Je táto heuristika prípustná? h(start) = 8

	Average nodes expanded when the optimal path has...		
	...4 steps	...8 steps	...12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

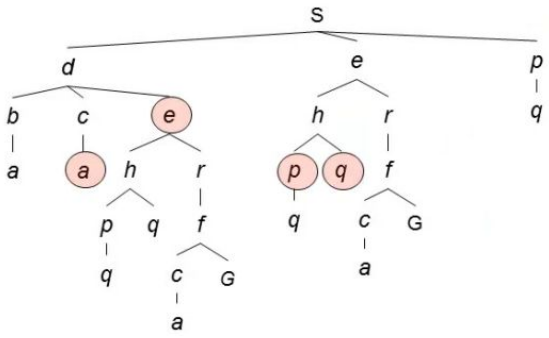
-Optimálne riešenie zjednodušeného problému
-Zjednodušený problém: dlaždice sa môžu prekrývať (Manhattan metrika)
-*h(start)* = 3 + 1 + 2 +.. = 18

	Average nodes expanded when the optimal path has...		
	...4 steps	...8 steps	...12 steps
TILES	13	39	227
MANHATTAN	12	25	73

Pôvodný problém:
Dlaždica sa môže pohnúť z miesta A do miesta B ak A je horizontálne, resp. vertikálne susediace s B a súčasne platí, že B je prázdne
Relaxovaný problém:

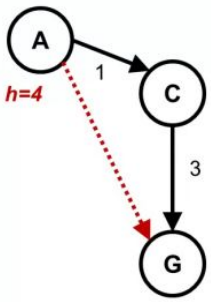
1. Dlaždica sa môže pohnúť z miesta A do miesta B, ak A susedí s B
2. Dlaždica sa môže pohnúť z miesta A do miesta B

Vyhľadávanie v grafe
-e, a, p, q už boli expandované, nie je nutné to robiť znova
-**Základná myšlienka**: neexpanduj dvakrát ten istý stav
-Implementácia:



- Strom + množina expandovaných stavov (“closed set”)
- Tradičné DFS, resp. BFS, ale pred expandovaním nódu zištuj, či už nód nebol expandovaní (nachádza sa v (“closed set”)
- Ak sa nenachádza, expanduj nód a pridaj do (“closed set”)
- Nemusí platiť, pre všetky problémy (Pacman), keď sú “loopy” nevyhnutné

Konzistencia heuristiky
Myšlienka: heuristika <= reálne váhy
-Prípustnosť (v strome): *heuristic cost* <= *actual cost* do cieľa
h(a) <= reálne váhy z A do cieľa
-Konzistencia (v grafe) (trojuholníková nerovnosť)
h(a) - h(C) <= cost(A to C)



A* zhrnutie:
-A*: spätné a dopredné váhy
-A*: je optimálna ak je heuristika prípustná/konzistentná
-Ako získať správnu heuristickú metriku?

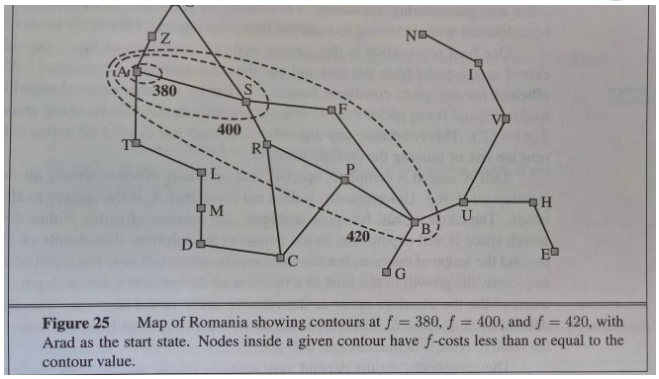


Figure 25 Map of Romania showing contours at $f = 380$, $f = 400$, and $f = 420$, with Arad as the start state. Nodes inside a given contour have f-costs less than or equal to the contour value.

Identifikačné problémy s ohraničeniami

Vyhľadávanie: zhrnutie
Stavový priestor: single-agent, deterministický, diskrétny, úplný (fully observable)
Plánovanie: vektor akcií
-Dôležitá je cesta (plán) k cieľu
-Akcie (cesty) majú rozličnú váhu
-Správne navrhnutá heurisitka urýchľuje vyhľadávanie

Identifikačný problém
-Dôležitý je cieľ, nie cesta
-Na goal state sa kladú rôzne ohraničenia → analógia s lineárnym a kvadratickým dynamickým programovaním

Vyhľadávací problém:
Goal test - hľadanie štruktúry akcií, ktoré spĺňajú goal test

Identifikačný problém s ohraničeniami (CSP)
-Podmnožina vyhľadávacích problémov
-Stav je definovaný **premennými X_i**, ktoré nadobúdajú hodnoty z domény **D**
-Goal test - súbor ohraničení, ktoré jednotlivé stave a ich hodnoty musia splniť

Príklady:

- **Farbenie Grafu (mapy)**
  - Premenné: štáty austrálie
  - Obor hodnôt: farby D = {red, green, blue}
  - Ohraničenia: susedné štáty rôzne farby
- **N-Queens**
  - Premenné: kráľovné
  - Obor hodnôt {1, 2, 3, .. N}
  - Ohraničenia: kráľovné sa nemôžu ohrozovať
- **Sudoku, Kryptografia, Rozvrh, Hardvérový návrh, Doprava (letiská, železničné uzly), Detekcia porúch**

Grafy s ohraničeniami
-Binárny CSP: ohraničenie na 2 premenných
-Binárny graf s ohraničeniami: uzly sú premenné a hrany charakterizujú ohraničenie
-Vyhľadávanie v grafe (pre zrýchlenie algoritmu)

Diskrétne premenné:
-Obor hodnôt s konečným počtom prvkov
-*n* premenných a *d* je rozsah hodnôt domén
-$O(d^n)$ → počet možných stavov CSP
-Boolean CSPs

-Nekonečný obor hodnôt (integers, strings, atď.)
Spojité premenné:
-lineárne programovanie

Ohraničenia
-Druhy ohraničení:
-Unárne - viažu sa na samotnú premennú (SA != green)
-Binárne - dvojica premenných (SA != WA)
-Ohraničenia vyššieho rádu: 3 a viac premenných e.g.: cryptarithmetic
-**Preferencie:** - napr. červená je lepšia ako modrá

Základné pojmy:
Stav - odpovedá priradeniu hodnôt do niektorých premenných
Konzistentný stav - stav neporušujúci žiadne ohraničenie
Úplný stav - Všetky premenné majú priradenú hodnotu
Cieľ - Nájsť úplný konzistentný stav

Zadanie problému:
-Začiatočný stav: prázdny obor hodnôt stavov {}
-Akcia: priraď **hodnotu premennej** z oboru hodnôt, tak že sa neporuší podmienka (neplatí pre DFS)
-Goal test: jednotlivé premenné majú priradené hodnoty, ktoré spĺňajú ohraničenia, úplné konzistentné priradenie

- **Backtracking (spätné prehľadávanie)**

-základný algoritmus pre riešenie CSP

1. Vyhľadávanie po jednom
    - Nie všetky permutácie hodnôt premenných
    - Premenné testuje po jedno. Nie na konci a naraz ako pri DSP
2. Overuj ohraničenia počas chodu algoritmu
    - Uvažuj len tie hodnoty z oboru hodnôt, ktoré neporušujú ohraničenia
    - Overujeme ohraničenia počas chodu algoritmu (nevýhoda)

    DFS + 1. a 2. → spätné prehľadávanie

- **Forward checking (dopredná filtrácia)**

Filtrácia: redukcia oboru hodnôt dosiaľ nepriradených premenných
-Po priradení hodnoty danej premennej, verifikuj obor hodnôt ostatných dosiaľ nepriradených premenných a prípadne ho redukuj
-Filtrácia: len medzi priradenou premennou a jej susedmi
-NT a SA nemôžu byť modré. Filtrácia s tým nič nerobí, neskôr budeme musieť urobiť backtracking

- **Arc consistency**

(konzistencia hrany)

Hrana X → Y je **konzistentná** ak pre každú hodnotu *x* z oboru hodnôt X existuje aspoň jedna hodnota y z Y, ktorá vyhovuje ohraničeniu

Filtrácia zabezpečuje konzistenicu hranu, ale len so susedmi a to nemusí stačiť

-Všetky hrany CSP musia byť navzájom konzistentné (CSP je konzistentné)

-Ak X stratí hodnotu z oboru hodnôt, susedia X musia byť znova "ocheckovaný" na konzistenciu svojej hrany

-Deteguje chybu skôr ako dopredná filtrácia

-Nevýhoda: komplexita

Môže nájsť 1 riešenie, viacero riešení, žiadne riešenie

-Spúšťa sa po každom spätnom vyhľadávaní

-Algoritmus AC3 beží neustále v algoritme spätného vyhľadávania

Radenie premenných

- ❖ **Minimum remaining values**
  - ➢ Vyberaj tie premenné, ktoré majú najmenej možných hodnôt v doméne
- ❖ **Least Constraining value**
  - ➢ Vyberaj tie hodnoty z domény pre danú premennú, tak že vybratá hodnota premennej narúša minimálne obor hodnôt ostatných premenných

- **K-consistency**

-**Úroveň konzistencie:**

1-Consistency (Node Consistency): každý uzol spĺňa unárne podmienky

2-Consistency (Arc Consistency): Pre každý pár uzlov platí, že z jedného uzla sa dá prejsť do druhého v rámci riešenia CSP

k-Consistency: Pre každé k-1 uzly platí, že existuje konzistentné zadanie rozšírené pre k-ty uzol

-čím vyššie *k,* tým väčšia komplexita

-k-konzistencia: k-1, k-2 … k-n sú konzistentné

Tvrdenie: n uzlov - n-konzistencia nám zabezpečí to, že nie je nutné spúšťať spätné vyhľadávanie

Prečo:

- Vyber ľubovoľnú hodnotu pre premennú (unary constraint)
- Vyber nový premennú
- Konzistencia hrany nám zabezpečí to, že existuje dovolená hodnota
- Vyber novú premennú
- 3-konzistencia nám zabezpečí to, že existuje dovolená hodnota tejto premennej, atď.

Príklad: k=3 - path consistency

Riešenie CSP využitím špecifickej štruktúry

-Krajný prípad: nezávislé subproblémy (Príklad: Tasmania a Austrália)

-Planárny graf ohraničení determinuje závislosť

-Uvažujme, že graf s *n* premennými vieme rozložiť na sub-grafy s *c* premennými:

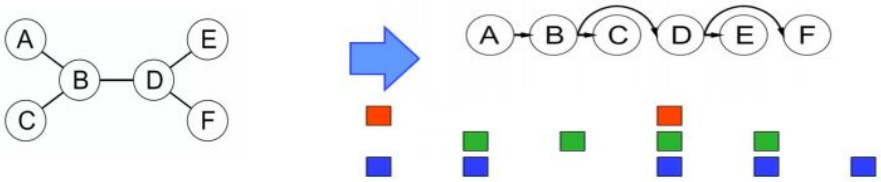
- Zložitosť - $O((n/c)(d^c))$, lineárne s n!
- Príklad: n = 80, d = 2, c = 20
- 2^{80} = 4 milióny rokov
- $(4)(2^{20})$ = 0.4 sekundy

CSP v štruktúre stromu

-Poznámka: ak v CSP nie sú slučky (loops), tak komplexita riešenia CSP je $O(n\ d^2)$, všeobecne je $O(d^n)$

-**Algoritmus pre stromové CSP**

-Poradie: Urči koreňovú premennú a zotrieď zvyšné premenné



-Konzistencia hrany, ale zozadu. For i = n : 2, applyRemoveInconsistent(Parent(X_i), X_i)

-Priraď hodnoty v poprednom smere. For i = 1 : n, assign X_i consistently with Parent(X_i))

-$O(n\ d^2)$

Tvrdenie 1: Po prvej časti algoritmu (spätný chod), všetky hrany v strome sú konzistentné (v hrane, 2-konzistencia)

Dôkaz: Pre každú hranu X → Y, ktorá je konzistentná platí, že jej konzistentnosť nemôže byť narušená (je to strom, potomkovia Y sa už nebudú meniť)

Tvrdenie 2: Ak sú všetky hrany v strome konzistentné, dopredná filtrácia nebude **nikdy** vykonávať spätné vyhľadávanie

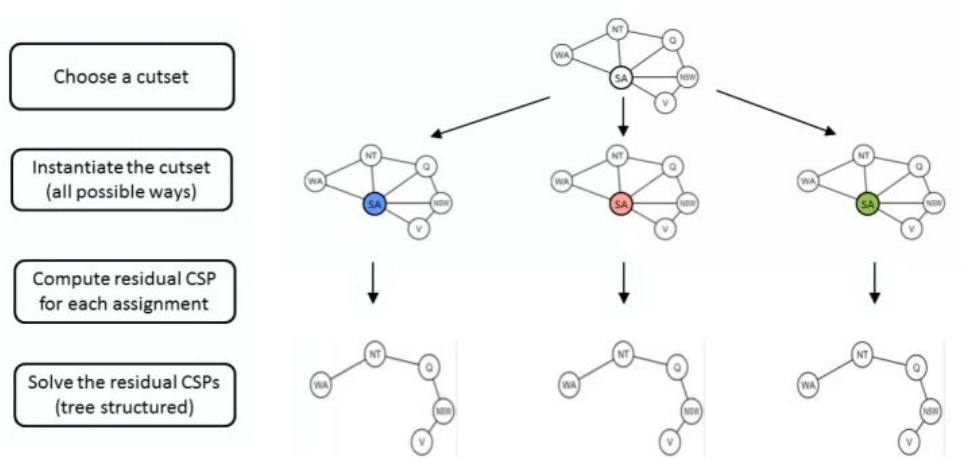
Dôkaz: indukcia

-Funguje daný algoritmus v grafe bez stromovej štruktúry?

Podmienky (Conditioning): zadajme hodnoty z domény premennje SA a filtrujme domýn zostávajúcich premenných

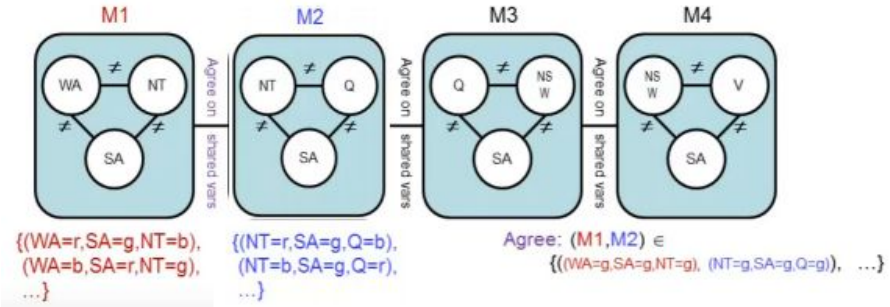
Cutset conditioning: zadajme všetky hodnoty z domény pre premenný SA (ale tých premenných môže byť viacej a filtrujme domény zostávajúcich premenných

Zložitosť: $O((d^c)\ (n\ c)\ d^2)$, pre malé c



Dekompozícia stromu

-Myšlienka: vytvoriť strom pozostávajúciz mega-premenných
-Každá mega-premenná obsahuje časť z originálneho CSP (podproblém)
-Podproblémy sa prekrývajú pre zabezpečenie riešenia



Lokálne hľadanie (local search)

-Min-konflikt
-Hill Climbing (horolezecký algoritmus)
-Simulated annealing (simulované žíhanie)

Lokálne hľadanie: algoritmy pracujú s konečným stavom, t.j. Všetkým premenným sú priradené (náhodné) hodnoty
Pracuje väčšinou s jednou premennou - nízke nároky na pamäť

Ako je to v CSPs:

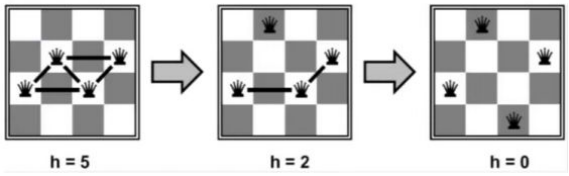
-Náhodne vyber assigment
-Zmeň jedno, či niekoľko hodnôt premenných
-Žiadny zásobník

Algoritmus

-Výber premennej - náhodný, resp. premenná s najväčším počtom konfliktov
-Výber hodnoty premennej: tá ktorá minimalizuje počet konfliktov

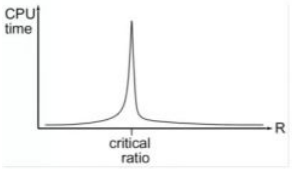
-Evaluačná funkcia (fitness, score) h(n) = celkový počet konfliktov

- **Min-konflikt**



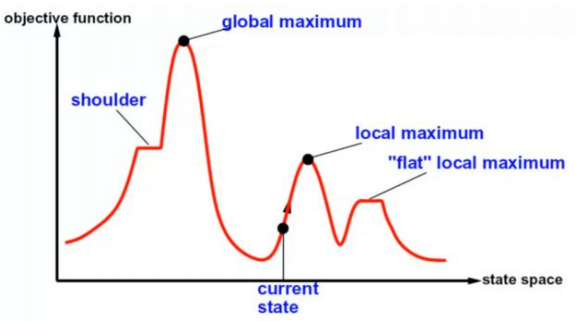
Stavy: 4 kráľovné v 4 stĺpcoch (4^4 = 256 stavov)
Akcia: posuň kráľovnú v stĺpci
Cieľový stav: kráľovné sa neohrozujú
Evaluačná funkcia: c(n) = počet útokov
Lokálne vyhľadávanie je schopné riešiť N-Queens problém v konštantnom čase
(e.g., n= 10 000 000)
-Možnosti v praktických realizáciách

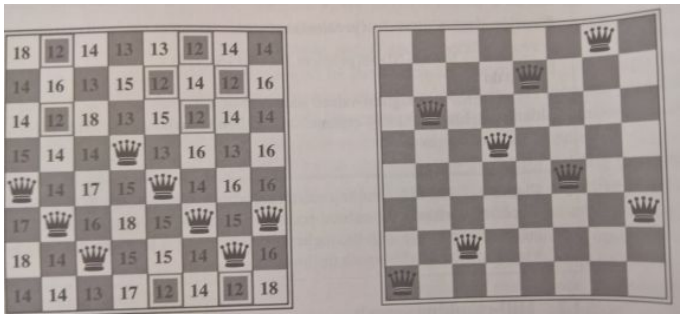
$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



- **Hill Climbing**

Základná myšlienka:
-ľubovoľny stav CSP (assigment)
-Loop: ak nájdem lepšieho suseda
-Ak nie, koniec





Lokálne optimum: h(n) = 1… Hill Climbing sa z tohto stavu nevie dostať
-Greedy (vyberá najlepší stav a korešpondujúcu akciu, bez uvažovania čo ďalej)
-Nekompletný (lokálne optimá)
-Steepetest-ascent, First-choice hill climbing, Random-restart hill climbing

- **Simulated annealing**

Myšlienka: dovoliť aj tie akcie, ktoré znižujú úžitok, aby sme sa dostali z lokálnych miním, resp. maxím
-ale znižuj pravdepodobnosť výberu týchto akcií počas trvania simulácie
-Namiesto akcií vedúcich k zlepšeniu evaluačnej funkcie, volíme náhodne akcie
Vlastnosť:
-Ak T klesá k nule dostatočne pomaly, simulované žíhanie nájde optimálne riešenie!

$$T(k) = T_{init}\frac{T_{final}}{T_{init}}^{\frac{k}{k_{max}}}$$

n: predchádzajúci stav, n': súčasný stav, T: aktuálna teplota,
P: pravdepodobnosť výberu daného stavu

$$P(n, n', T) = \exp(\frac{h(n)' - h(n)}{T})$$

-Musíme zadefinovať počiatočnú a konečnú teplotu, ako aj maximálny počet iterácií
-Teplota musí klesať (monotónne klesať), inak algoritmus môže neponúkať suboptimálne výsledky
-Nevyberajte konečnú teplotu 0, ale niečo blízke nule