UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH

PRÍRODOVEDECKÁ FAKULTA

ROZHODOVACIE DŽUNGLE A INÉ ACYKLICKÉ KLASIFIKAČNÉ ALGORITMY STROJOVÉHO UČENIA

2019 **Bc. Šimon Horvát**

UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH

PRÍRODOVEDECKÁ FAKULTA

ROZHODOVACIE DŽUNGLE A INÉ ACYKLICKÉ KLASIFIKAČNÉ ALGORITMY STROJOVÉHO UČENIA

DIPLOMOVÁ PRÁCA

Študijný program:	Informatika
Pracovisko (katedra/ústav):	Ústav informatiky
Vedúci diplomovej práce:	RNDr. Ľubomír Antoni, PhD.

Košice 2019 **Bc. Šimon Horvát**

Univerzita P. J. Šafárika v Košiciach
Prírodovedecká fakulta

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Šimon Horvát
Študijný program:	Informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	9.2.1. informatika
Typ záverečnej práce:	Diplomová práca
Jazyk záverečnej práce:	slovenský
Sekundárny jazyk:	anglický

Názov:	Rozhodovacie džungle a iné acyklické klasifikačné algoritmy strojového učenia
Názov EN:	Decision jungles and other acyclic classification tasks in machine learning.
Cieľ:	1. Popísať klasické metódy dolovania údajov zamerané na klasifikáciu. 2. Popísať nový model rozhodovacích pralesov s použitím acyklických grafov. 3. Navrhnúť spôsob voľby vhodných parametrov v metóde rozhodovacie pralesy. 4. Implementovať model rozhodovacích pralesov s voľbou optimálnych parametrov.
Literatúra:	[1] J. Han, M. Kamber: Data mining, Concepts and Techniques, Second edition, Elsevier, 2006 [2J J. Paralič: Objavovanie znalostí v databázach. Elfa, Košice 2003, ISBN 80-89066-60-7, 80 s. [3J Charu C. Aggarwal: Data Mining: The Textbook. Springer, 2015 [4J G. James, D. Witten, T. Hastie and R. Tibshirani: An Introduction to Statistical Learning with Applications in R. Springer, 2015.

Vedúci:	RNDr. Ľubomír Antoni, PhD.
Oponent:	doc. RNDr. Gabriela Andrejková, CSc.
Ústav :	ÚINF - Ústav informatiky
Riaditeľ ústavu:	prof. RNDr. Viliam Geffert, DrSc.

Dátum schválenia: 29.04.2019

Poďakovanie

Ďakujem Bohu a Panne Márii za všetky milosti počas celého štúdia, rodičom sa podporu a za financie potrebné k doštudovaniu. Ďakujem tiež mojej priateľke Márii Vaňovej za podporu, milé slova a za nespočetné chvíle radosti. Tiež by som chcel poďakovať vedúcemu práce, RNDr. Ľubomírovi Antonimu, PhD.**,** za možnosť pracovať na veľmi zaujímavej téme a za cenne rady už počas bakalárskeho štúdia. Rovnako tak, ďakujem doc. RNDr. Gabriele Andrejkovej CSc., za všetky rady v rámci predmetu diplomový seminár a za pochopenie mnohých súvislosti z témy a doc. RNDr. Jozefovi Jiráskovi, PhD., za čas a pomoc pri zodpovedaní odborných otázok spojených s diplomovou prácou.

Abstrakt v štátnom jazyku

V tejto práci rozoberáme rozhodovacie džungle, ktoré boli navrhnuté Shottonom (*Microsoft Research* tím) na konferencii NIPS 2013. Rozhodovacia džungľa je skupinovým modelom rozhodovacích DAG-ov (*directed acyclic graphs*). Koncept džungle veľmi úzko súvisí s rozhodovacím lesom, rovnako DAG je analógiou k rozhodovaciemu stromu, preto v tejto práci analyzujeme aj tieto klasifikačné modely. Počet uzlov v rozhodovacích stromoch rastie exponenciálne s hĺbkou stromu. Pre niektoré aplikácie, napríklad na mobilných alebo zabudovaných procesoroch, je pamäť obmedzeným zdrojom, a tak exponenciálny rast stromov obmedzuje ich hĺbku, teda aj ich potenciálnu presnosť. Na rozdiel od rozhodovacích stromov, ktoré umožňujú iba jednu cestu ku každému uzlu, DAG v rozhodovacej džungli umožňuje viacero ciest od koreňa ku každému listu. V práci okrem popisu Shottonovej džungle, navrhneme aj vlastný variant rozhodovacích džunglí, ktorý zovšeobecňuje DAG z binárneho grafu na všeobecný. Myšlienka rozhodovacích džunglí teda prichádza s úsporou pamäťových nárokov. Navyše, zvyšujú generalizáciu, čo sa často odzrkadľuje v presnosti klasifikácie.

Abstrakt v cudzom jazyku

In this thesis, we discuss decision jungles as proposed by Shotton (*Microsoft Research team*) at NIPS 2013. Decision jungles are ensembles of directed acyclic graphs (DAGs). The concept of jungle is closely related to decision forest, by the same way DAG is an analogy to the decision tree, so we will also discuss these classification models. The number of nodes in decision trees grows exponentially with the depth of the tree. For some applications, such as mobile or embedded processors, memory is a limited resource, so exponential tree growth limits their depth and hence also their potential accuracy. Unlike decision trees that allow only one path to each node, DAG in the decision jungle allows multiple paths from the root to each leaf. In addition to the introduction of the Shotton Jungle, we will also present our own decision-making jungle, which generalizes the DAG from a binary graph to a general one. Thus, the idea of decision jungles comes with savings of memory requirements. In addition, they increase generalization, which often reflects the accuracy of classification.

Obsah

Úvod

Rozhodovacie stromy majú dlhú históriu v strojovom učení a boli jedným z prvých modelov predstavených v induktívnom učení [5]. Ich použitie v klasifikácií a regresii sa stálo veľmi populárnym v mnohých oblastiach strojového učenia, a to aj vďaka schopnosti poradiť si s veľkým množstvom dát a zhotoviť kvalitnú predikciu, pričom si model dokáže zachovať jednoduchú interpretovateľnosť (v každom rozhodovacom uzle je zrejmé, na základe čoho sa rozhodnutie uskutočňuje). Podrobnosti o rozhodovacích stromoch, o algoritme ich učenia aj o jednotlivých typoch stromov uvádzame v kapitole 2.

Hoci rozhodovacie stromy umožňujú efektívne predikcie, učenie sa optimálneho rozhodovacieho stromu je NP-úplný problém [6]. Niektorí vedci však tvrdia, že pokúšanie sa o konštrukciu optimálneho rozhodovacieho stromu, môže byť škodlivé, lebo zvyšuje mieru preučenia [7].

Preučenie je hlavnou nevýhodou rozhodovacieho stromu. Riešením je skupinový model (*ensemble learning*) rozhodovacích lesov. Vplyv preučenia sa dá do istej miery znížiť, stromy však limituje ešte jeden aspekt: exponenciálna veľkosť vzhľadom k ich hĺbke. Skupinový model lesov pamäťovú veľkosť ešte znásobí (počtom stromov v lese).

Obe tieto problémy rieši skupinový model (kapitola 3) rozhodovacích DAG-ov – rozhodovacia džungľa, ktorú popisujeme v kapitole 5. Skupinové modely vo všeobecnosti odstraňujú preučenie a myšlienka DAG-u, ktorá je založená na spájaní navzájom podobných uzlov, zmenšuje vplyv hĺbky na celkovú pamäťovú náročnosť algoritmu.

Ako rozhodovací strom, tak aj DAG, je v Shottovej publikácii binárny graf. Binárne rozhodovacie stromy, veľmi dávno nahradili pôvodný nebinárny (všeobecný) rozhodovací strom (generovaný algoritmom ID3), ktorý už v roku 1986 vynašiel J. Ross Quinlan [4]. Novšie, binárne stromy sú schopné pracovať ako s kategorickými, tak aj s numerickými hodnotami (so spojitou premennou). Keďže DAG vychádza z binárneho rozhodovacieho stromu, analogicky, aj DAG v Shottonovej práci je binárnym grafom. Avšak, aby binárne grafy vedeli pracovať s kategorickou premennou, kategorické údaje sa musia kódovať do binárnych vektorov (kapitola 5.1.1),

ktoré zvyšujú dimenziu celého dátového priestoru. Prirodzeným následkom je nárast hĺbky grafu, čo zvyšuje pamäťové nároky.

Po predstavení konceptu rozhodovacích džunglí, ako ich opísal Shotton, navrhneme zovšeobecnenie DAG-u ako grafovej štruktúry vychádzajúcej z algoritmu ID3, čo nám pomôže pracovať s kategorickými hodnotami priamo, bez potreby prekódovania. Prezentujeme tiež algoritmus, ktorým zakódujeme numerické dáta tak, aby s nimi všeobecný rozhodovací graf vedel pracovať, a zároveň, aby sme nezvyšovali dimenziu rozhodovacieho priestoru.

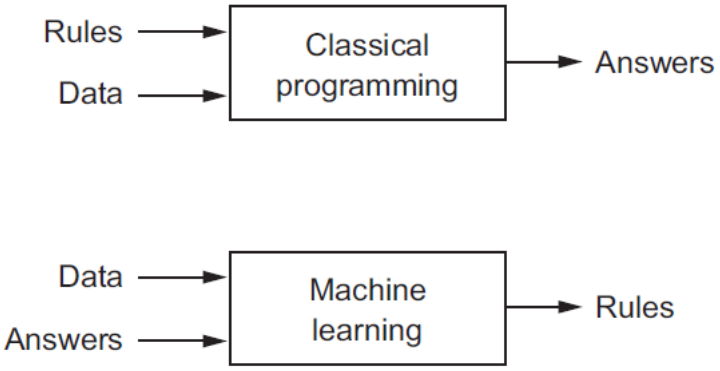
V poslednej časti práce sa venujeme prezentácii výsledkov a porovnaniu s Pohlenovým testom Shottonovej rozhodovacej džungle. Hoci hlavné využitie nášho modelu vidíme pri práci s dátami prevažne kategorického charakteru, ukazuje sa, že naša zovšeobecnená džungľa je schopná pracovať aj numerickými dátami. Výsledky v poslednej kapitole ukazujú, že zovšeobecnená rozhodovacia džungľa dramaticky zmenšuje pamäťové nároky, nie len v porovnaní s rozhodovacím lesom, ale aj v porovnaní s binárnou rozhodovacou džungľou, prezentovanou *Microsoft Research* tímom na konferencii NIPS 2013.

1 Strojové učenie

Ak chceme vyriešiť problém pomocou počítača, potrebujeme nejaký algoritmus. **Algoritmus** je postupnosť inštrukcií, ktoré by sa mali vykonať pre transformáciu vstupu na výstup. Napríklad sa dá vytvoriť algoritmus na triedenia čísel. Vstup je množina čísel a výstup je ich usporiadaný zoznam. Pre tú istú úlohu môžu existovať rôzne algoritmy a my môžeme mať záujem nájsť ten najefektívnejší, vyžadujúci najmenší počet inštrukcií alebo najmenšiu pamäť.

Pre niektoré úlohy však nemáme algoritmus – napríklad rozoznať spamové e-maily od legitímnych e-mailov. Vieme, aký je vstup: e-mailový dokument, ktorý v najjednoduchšom prípade predstavuje súbor znakov. Taktiež vieme, ako by mal vyzerať výstup: áno / nie – označujúci, či správa je spam alebo nie. Nevieme však, ako zmeniť vstup na výstup.

To, v čom nám chýbajú **poznatky**, si vykompenzujeme **množstvom dát**. Môžeme ľahko zhromaždiť tisíce príkladov správ, o ktorých vieme, či sú spamom alebo nie. A to, čo chceme, je "naučiť sa" z týchto príkladov, čo robí spamovú správu spamom. Inými slovami, chceme, aby počítač (stroj) automaticky **extrahoval** algoritmus pre túto úlohu. Nie je potrebné naučiť sa triediť čísla, na to už algoritmy máme. Existuje však veľa aplikácií, pre ktoré nemáme algoritmus, ale máme množstvo dát.



Obr. 1 Klasický algoritmus a strojové učenie

S pokrokom v oblasti výpočtovej techniky sme v súčasnosti schopní uchovávať a spracovávať veľké množstvo dát. Väčšina zariadení na zhromažďovanie údajov je digitálna a zaznamenáva spoľahlivé údaje.

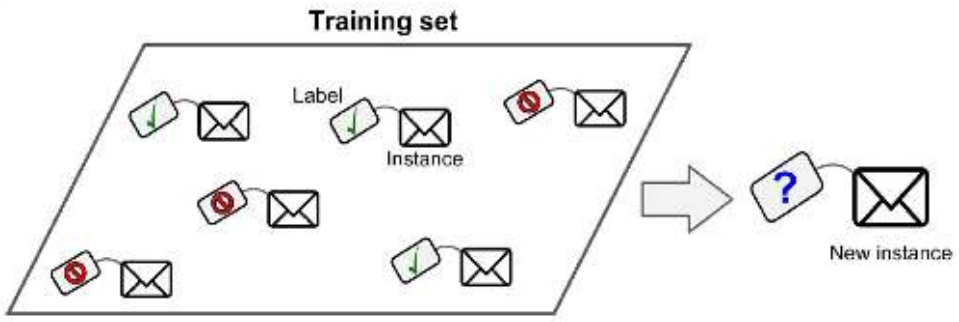
1.1 Učenie s dozorom/bez dozoru

Strojovo učiace sa systémy môžu byť klasifikované podľa množstva a druhu dohľadu, ktorý dostávajú počas tréningu. Existujú dve hlavné kategórie: **učenie s dozorom** a **učenie bez dozoru.**

1.1.1 Učenie s dozorom

Pri učení s dozorom dáta, z ktorých sa algoritmus učí, zahŕňajú požadované riešenia – dáta majú svoje „štítky“ (*labels*).

Typickou úlohou učenia s dozorom je **klasifikácia**. Filter spamu je dobrým príkladom. Výsledný algoritmus je vyškolený s mnohými príkladmi e-mailov, spolu s ich triedou (spam alebo nespamová správa) a musí sa naučiť klasifikovať nové, „neoznačené“ e-maily.



Obr. 2 Učenie s dozorom

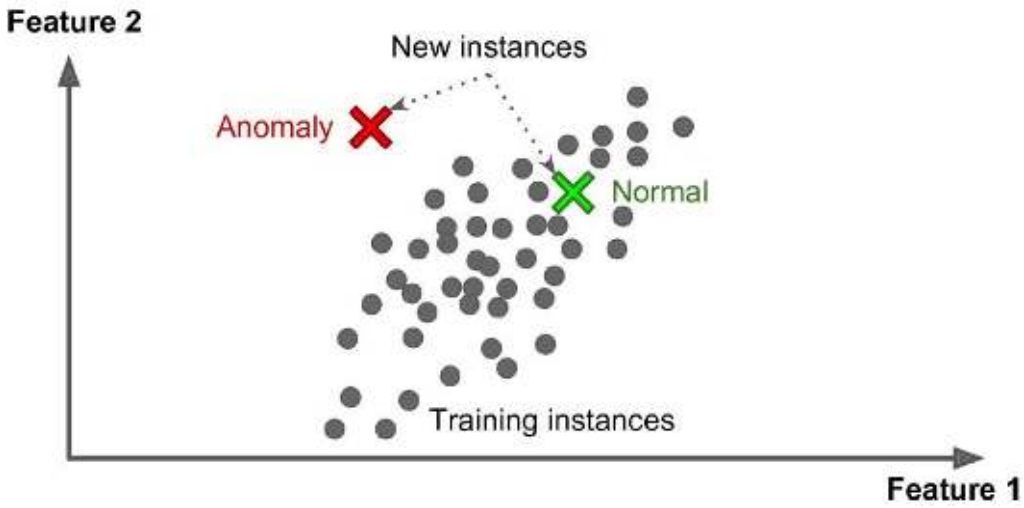
Ďalšou typickou úlohou je predpovedať číselnú (spojitú) **cieľovú premennú**, napríklad cenu auta, vzhľadom na súbor **atribútov** (počet kilometrov, vek, značka atď.), nazývaných predikátory. Tento druh úlohy sa nazýva **regresia**. Učiaci sa systém k tréningu potrebuje veľa príkladov automobilov, vrátane ich predikátorov a ich štítkov (t.j. ich ceny).

1.1.2 Učenie bez dozoru

V učení bez dozoru máme dočinenia s neoznačenými dátami (dáta bez štítkov) alebo s dátami neznámej štruktúry. Použitím tejto techniky, sme schopní preskúmať štruktúru našich dát tak, aby sme získali zmysluplné informácie.

Typickou úlohou učenia bez dozoru je **klastrovanie.** Jedná sa o techniku analýzy údajov, ktorá nám umožňuje organizovať hromadu informácií do zmysluplných podskupín (klastrov) bez akýchkoľvek predchádzajúcich vedomostí o členstve v skupine. Každý **klaster**, ktorý vzniká počas analýzy definuje skupinu objektov, ktoré majú určitý stupeň podobnosti, ale sú viac odlišné od objektov v iných klastroch.

Ďalšou dôležitou úlohou tohto typu je **detekcia anomálií** (*anomaly detection*) - napríklad zistenie nezvyčajnej transakcie medzi kreditnými kartami, aby sa zabránilo podvodom, detekcia výrobných chýb alebo automatické odstraňovanie nezvyčajných dát (*outliers*) z datasetu pred použitím nejakého iného učiaceho algoritmu.



Obr. 3 Detekcia anomálií

1.1.3 Reinforcement learning

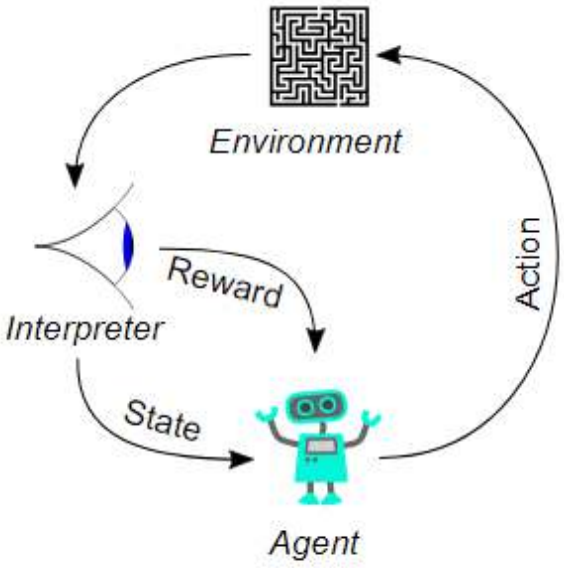
Reinforcement learning sa vyskytuje, keď analyzujeme dáta s príkladmi, ktoré neobsahujú riešenia (závislú premennú), ako v prípade učenia bez dozoru.

Príklad je však ohodnotený pozitívnou alebo negatívnou spätnou väzbou podľa riešenia, ktoré algoritmus navrhuje.

Tento typ učenia sa vyskytuje v aplikáciách, v ktorých algoritmus musí prijímať rozhodnutia a rozhodnutia majú následky. V ľudskom svete sa to dá pripodobniť učeniu na štýl pokus – omyl.

V reinforcement learning agent dostane informácie o svojom prostredí a naučí sa vyberať akcie, ktoré maximalizujú nejakú odmenu.

V tomto prípade aplikácia prezentuje algoritmus s príkladmi konkrétnych situácií, ako je napríklad to, že hráč bol uviaznutý v bludisku, pričom sa vyhýbal nepriateľovi. Aplikácia umožňuje, aby algoritmus vedel o výsledkoch akcií, ktoré prijal, a učenie sa vyskytuje pri snahe vyhnúť sa tomu, čo zistí, že je nebezpečné, a snažiť sa o prežitie.



Obr. 4 Schéma učenia

Zaujímavý príklad využitia reinforcement learning-u je, keď sa počítače učia hrať videohry sami. V tomto prípade sú vstupne dáta pre algoritmus príkladmi konkrétnych situácií, ako je napríklad to, že hráč je uviaznutý v bludisku, pričom sa vyhýbal nepriateľovi, teda algoritmus vie o výsledkoch akcií, ktoré vykonal.

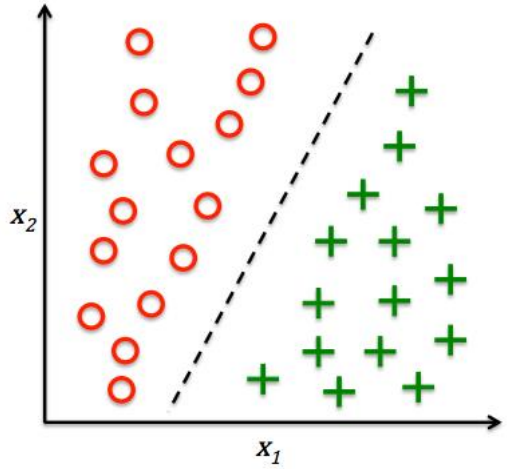
Učenie teda spočíva v snahe vyhnúť sa tomu, o čom zistí, že je nebezpečné, a snažiť sa o prežitie.

1.2 Klasifikácia v strojovom učení

Klasifikácia je typ úlohy učenia s dohľadom, ktorého cieľom je predpovedať klasické označenia triedy nových inštancií na základe predchádzajúcich pozorovaní.

Tieto označenia triedy sú diskrétne, neusporiadané hodnoty, ktoré možno chápať ako skupinové členstvo v inštanciách.

Uvedený príklad – detekcia spamu prostredníctvom e-mailu – predstavuje typický príklad **binárnej klasifikácie** – algoritmus strojového učenia sa učí súbor pravidiel na rozlíšenie medzi dvoma možnými **triedami**: spam a nespamový email.



Obr. 5 Binárna klasifikácia

Súbor tried však nemusí mať binárny charakter. Prediktívny model môže priradiť akémukoľvek triedu, ktorá bol prezentovaná v súbore dát počas tréningu, na novú, neoznačenú inštanciu. Typickým príkladom všeobecnej klasifikačnej úlohy (**multi-class classification**) je rozpoznávanie ručne písaných znakov. Ak používateľ poskytne nový ručne písaný znak prostredníctvom vstupného zariadenia, náš prediktívny model bude môcť predpovedať správne písmeno v abecede s určitou presnosťou.

Definícia 1.2a – Všeobecný klasifikačný problém

Majme tréningovú množinu $X = \{(\mathbf{x_1}, y_1), \dots, (\mathbf{x_N}, y_N)\} \subset \mathbb{R}^n \times \{1, \dots, C\}$

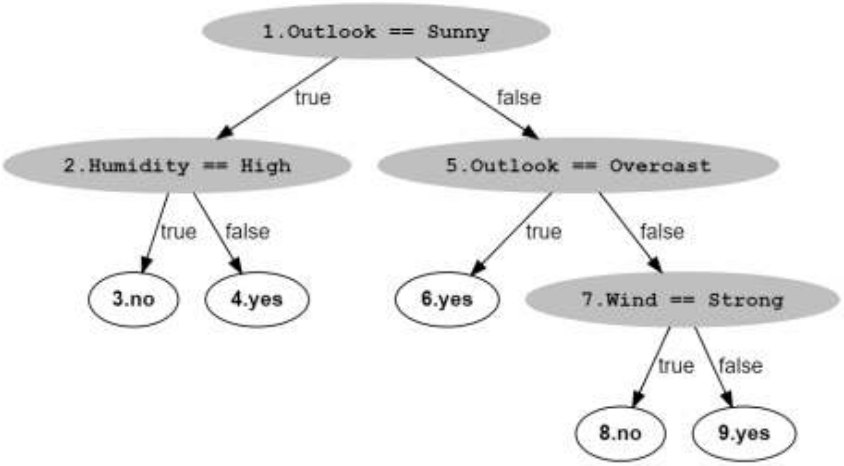
- *tréningové príklady* $\boldsymbol{x_i} \in \mathbb{R}^n$
- *triedy* $y_i \in \{1, \dots, C\}$

Klasifikácia je priradenie nevyriešeného príkladu **x** do jednej z tried *1, … , C*

2 Rozhodovacie stromy

Rozhodovací strom tvorí sada hierarchicky usporiadaných rozhodovacích pravidiel. Príkladom jednoduchého rozhodovacieho stromu môže byť rozdelenie živočíchov podľa rôznych kritérií. So stromovou štruktúrou sa stretávame pomerne často, lebo je prehľadná a ľahko interpretovateľná. Môže ísť napríklad o rodokmene alebo zobrazenie adresára a jeho položiek v počítači.

U rozhodovacích stromov je zrejmá analógia s reálnymi stromami v prírode, preto bola jednoducho prevzatá terminológia, ktorá je pre stromy bežná a dobre vystihuje podstatu algoritmu. Podobne ako u reálneho stromu teda hovoríme o tom, že rozhodovací strom **rastie**, **vetví** sa alebo ho **prerezávame**.



Obr. 6 Rozhodovací strom

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
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

Tab. 1 Dáta k modelu z Obr. 6

Rozhodovací strom sa skladá z koreňa, ktorý predstavuje celý súbor dát a postupne prebieha vetvenie do ďalších uzlov – strom **rastie**. Uzly, ktoré sa už ďalej nedelia, sa označujú ako **listy**. Stromy sú **binárne** alebo **nebinárne**, podľa toho, či sa vetvia na dve alebo viac vetiev.

Rozhodovacie stromy môžeme rozdeliť podľa typu **závislej premennej** na **klasifikačné** a **regresné**.

Majme strom T s uzlami $t = (t_1, \dots, t_N)$.

Ak je závislá premenná spojitá $Y = (y_1, \dots, y_n)$, pozorovaniam je priradená hodnota aritmetického priemeru z hodnôt objektov reprezentujúcich listový uzol a výsledný strom bude **regresný**.

U **klasifikačného stromu** sú pozorovania kategorickej závislej premennej Y s J kategóriami zaradené do niektorej z kategórií $c = (c_1, \dots, c_j)$, kde $J \geq 2$.

Pozorovanie premennej Y sú rozdelené do uzlov s hodnotami vysvetľujúcich premenných (*prediktorov*) $X_1, \dots, X_M$. Rozdelenie je znázornené graficky pomocou vetiev stromu.

Ak sú **prediktory** kategoriálne, ako v prípade stromu na **Obr. 6**, hodnoty x_i sú rozdelené podľa kategórií predikátor X . Napríklad prvé vetvenie predikátorom "$\boldsymbol{Outlook = sunny}$" s dvoma kategóriami "$\boldsymbol{true}$" a "$\boldsymbol{false}$" rozdelí premennú Y do dvoch dcérskych uzlov (podľa toho, či podmienka asociovaná s uzlom, predikátor, je splnená alebo nie je). Prvý uzol pre hodnotu predikátora "$\boldsymbol{true}$" obsahuje 5 inštancií – 3 krát hodnotu "$\boldsymbol{No}$" a 2 krát "$\boldsymbol{Yes}$" a druhý uzol pre kategóriu predikátora "$\boldsymbol{false}$" obsahuje zvyšných 7 inštancií. Vetvenie takto pokračuje, kým v uzle nie sú len inštancie s rovnakou hodnotou závislej premennej Y (v našom prípade ide o hodnotu "$\boldsymbol{No}$" alebo "$\boldsymbol{Yes}$"). Vtedy sa vetvenie zastaví a z uzla sa stáva **list,** ktorý je asociovaný s nejakou výslednou triedou – hodnotou závislej premennej Y.

Všeobecný algoritmus konštrukcie rozhodovacieho stromu vyzerá teda takto:

Algoritmus 1 Konštrukcia rozhodovacieho stromu

```
1: function KonštruujRozhodovacíStrom(TrénovaciaMnožina T, Float k):
2:     if aspoň k objektov z T patrí do triedy C
3:         then vytvor listový uzol asociovaný s triedou C;
4:         return;
5:     else
6:         for each atribút a ∈ A do
7:             for each možné rozdelenie hodnôt atribútu a do
8:                 ohodnoť kvalitu rozdelenia, ktoré by takýmto spôsobom vzniklo
9:         vykonaj najlepšie zo všetkých možných rozdelení;
10:        nech T_1, T_2, ..., T_m sú množiny, ktoré vzniknú týmto rozdelením;
11:        for i = 1 to m do:
12:            KonštruujRozhodovacíStrom (T_i, min);
13:        return
```

Ako však nájsť správne rozdelenie (správny predikát)? V podstate sa snažíme o také rozdelenie závislej premennej Y prediktorom X, aby hodnoty premennej Y boli vnútri uzla čo najhomogénnejšie a zároveň medzi uzlami čo najrozdielnejšie. Ktorý prediktor (a jeho hodnota) nám zaistí najlepšie rozdelenie, zistíme pomocou tzv. kriteriálnej štatistiky (*spliting Criterium*), ktorá určuje homogenitu uzla.

2.1 Kriteriálne štatistiky

2.1.1 Kvadratické chyby

Ako už bolo spomenuté, kriteriálna štatistika meria homogenitu v uzloch (*node impurity*). Predpokladajme, že máme regresný strom rozdelený do určitého počtu listových uzlov a predikovanú hodnotu závislej spojitej premennej Y chceme vyjadriť ako konštantu pre každý dcérsky uzol. Ak použijeme kritérium, ktoré minimalizuje strednú kvadratickú chybu, najlepším odhadom tejto konštanty bude priemer.

Snažíme sa teda nájsť také rozdelenie závislej premennej Y, ktoré bude mať najmenšiu priemernú kvadratickú odchýlku hodnôt y_i v potenciálnom uzle t od priemeru týchto hodnôt.

Kritérium minima kvadratickej chyby (*Least Square Deviation*) $Q(T)$:

$$\bar{y_t} = \frac{1}{N_t} \sum y_{i(t)}$$

$$Q_t(T) = \frac{1}{N_t} \sum_{i=1}^{N_t} (y_i - \bar{y_t})^2$$

kde N_t je počet pozorovaní v uzle t a $y_{i(t)}$ sú hodnoty závislej premennej v uzle t.

2.1.2 Gini index

Gini index (GI) je najčastejšie používaná kriteriálna štatistika pre klasifikačné stromy typu CART. Hodnota Gini indexu sa rovná nule, ak je v konečnom uzle iba jediná kategória premennej Y a dosahuje maximum, ak je v konečnom uzle v každej kategórii premennej Y rovnaký počet pozorovaní. Vo chvíli, keď dôjde k rozdeleniu uzla na dva dcérske uzly, je GI spočítaný pre každý dcérsky uzol.

Celková hodnota GI pre dané rozdelenie je rovná váženému súčtu všetkých GI jednotlivých dcérskych uzlov. Váha GI dcérskych uzlov je daná ako veľkosť dcérskeho uzla. GI teda jednoducho spočítame ako súčet $GI(i)$ dcérskych uzlov, ktoré sú vynásobené príslušným podielom pozorovaní v danom dcérskom uzle z celkového počtu pozorovaní v pôvodnom materskom uzle.

$$GI = \sum_{i=1}^{K} \frac{N_i}{N_t} GI(i) \,,$$

kde K je počet dcérskych uzlov (v prípade binárneho stromu je $K = 2$), N_t je počet pozorovaní v materskom uzle t a N_i sú počty v dcérskych uzloch.

$$GI = \sum_{c=1}^{J} p_{tc}(1 - p_{tc}) = 1 - \sum_{c=1}^{J} {p_{tc}}^2,$$

kde p_{tc} je podiel pozorovaní y_i s kategóriou c v uzle t z celkového počtu všetkých pozorovaní y_i v tomto uzle alebo pravdepodobnosť kategórie c v uzle t.

2.1.3 Entropia

Entropia (H) dáva veľmi podobné výsledky ako Gini index. Dosahuje maximum, ak sú jednotlivé kategórie premennej Y rovnomerne zastúpené v uzloch a minimum, ak pozorovania v uzle patria iba do jednej kategórie. Entropia je často používaná v algoritme C4.5 (neskôr si o tomto konkrétnom algoritme povieme viac).

Výpočet entropie v konkrétnom stromovom uzle:

$$H = -\sum_{c=1}^{J} p_{tc} \log_2 p_{tc} ,$$

kde p_{tc} je podiel pozorovaní y_i s kategóriou c v uzle t z celkového počtu všetkých pozorovaní y_i v tomto uzle, čiže pravdepodobnosť kategórie c v uzle t.

2.1.4 Klasifikačná chyba

Poslednou z vyššie uvedených kriteriálnych štatistík je klasifikačná chyba (ME). ME je podiel chybne klasifikovaných pozorovaní, čiže $1 - ME$ je celková presnosť stromu (podiel správne klasifikovaných pozorovaní). Klasifikačná chyba je obvykle používaná práve k finálnemu meraniu presnosti klasifikačného stromu, preto je logické jej použitie ako kriteriálnej štatistiky.

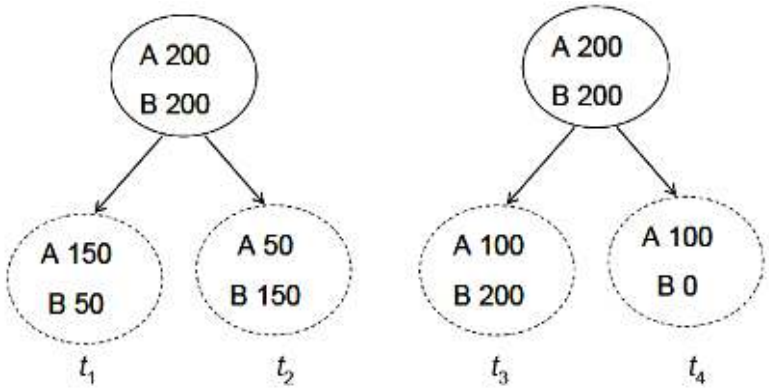
Na príklade si však ukážeme, prečo sú preferované iné indexy. Celková klasifikačná chyba pre dané delenie je opäť váženým súčtom ME v dcérskych uzloch.

$$ME = 1 - max\{p_{tc}\}$$

kde p_{tc} je podiel pozorovaní y_i s kategóriou c v uzle t z celkového počtu všetkých pozorovaní y_i v tomto uzle, čiže pravdepodobnosť kategórie c v uzle t.

2.1.5 Príklad – porovnanie kriteriálnych štatistík

Majme dve kategórie závislej premennej A, B. Počet vzoriek v uzle, ktorý chceme rozdeliť, je v skupine A i B rovnaký, $n_A = n_B = 200$. Preskúmajme teraz pomocou klasifikačných kriteriálnych štatistík dve možné rozdelenia stromu (**Obr. 7**). Ktoré bude vybrané ako vhodnejšie?



Obr. 7 Ukážka možného rozdelenia stromu do dvoch dcérskych uzlov – vľavo D1, vpravo D2

Spočítame hodnotu GI a ME pre každý dcérsky uzol $(t_1 - t_4)$ možného rozdelenia a následne určíme celkové indexy GI a ME pre obe rozdelenia $D1$ a $D2$.

	Uzol	n_A	n_B	p_A	p_B	p_t	$Gini = 1 - p^2_A - p^2_B$	$p_t * Gini$	
D1	t_1	150	50	3/4	1/4	1/2	$1-(3/4)^2-(1/4)^2 = 3/8$	1/2 * 3/8	0,1875
	t_2	50	150	1/4	3/4	1/2	$1-(1/4)^2-(3/4)^2 = 3/8$	1/2 * 3/8	0,1875
								celkový	**0,375**
D2	t_3	100	200	1/3	2/3	3/4	$1-(1/3)^2-(2/3)^2 = 4/9$	3/4 * 4/9	0,3333
	t_4	100	0	1	0	1/4	1-1-0 = 0	1/4 * 0	0
								celkový	**0,3333**

	Uzol	n_A	n_B	p_A	p_B	p_t	$ME = 1 - \max(p_A, p_B)$	$p_t * ME$	
D1	t_1	150	50	3/4	1/4	1/2	1- 3/4 = 1/4	1/2 * 1/4	0,125
	t_2	50	150	1/4	3/4	1/2	1- 3/4 = 1/4	1/2 * 1/4	0,125
								celkový	**0,25**
D2	t_3	100	200	1/3	2/3	3/4	1- 2/3= 1/3	3/4 * 1/3	0,25
	t_4	100	0	1	0	1/4	1-1 = 0	1/4 * 0	0
								celkový	**0,25**

Tab. 2 Priebeh výpočtu

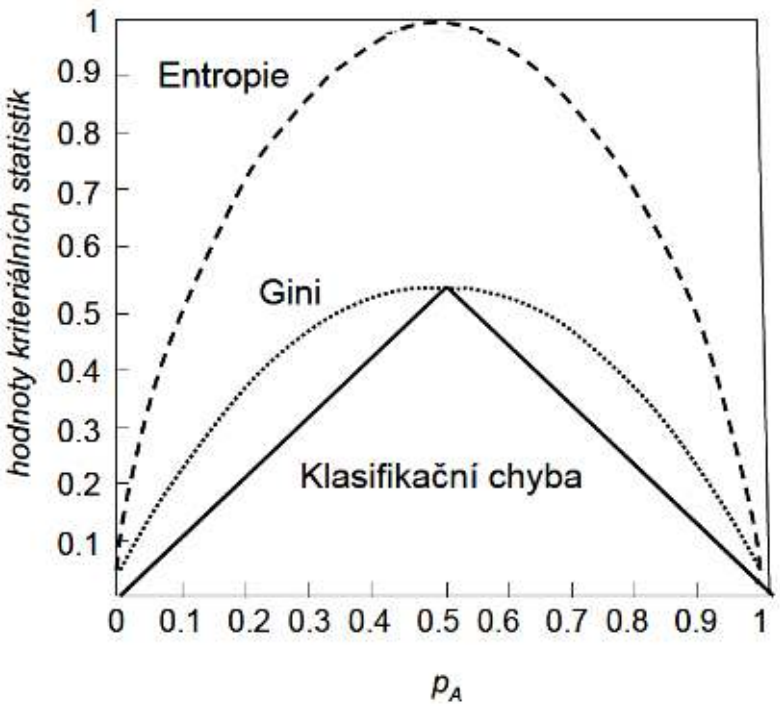
V **Tab. 2** sú vyjadrené počty pozorovaní jednotlivých kategórií v dcérskych uzloch (n_A, n_B), následne je spočítané zastúpenie kategórií v danom uzle z celkového počtu pozorovaní v tomto uzle, čiže ich pravdepodobnosť (p_A, p_B). Poslednou premennou, ktorú budeme potrebovať pri výpočte, je podiel pozorovaní v dcérskom uzle s celkovým počtom pozorovaní v materskom uzle (p_t).

Hodnota Gini indexu v prvom delení je pre oba uzly rovnaká $(GI = 0{,}1875)$. Podobne dopadla aj klasifikačná chyba $(ME = 0{,}125)$. Tento výsledok samozrejme nie

je prekvapením, vzhľadom na rozdelenie uzlov, ktorých homogenita je rovnaká. Pre prvé rozdelenie tak dostávame celkové hodnoty indexov $GI = 0{,}375$ a $ME = 0{,}25$.

Druhé rozdelenie *D2* obsahuje úplne homogénny uzol t_4, v ktorom je zastúpená len jedna kategória. Toto rozdelenie sa teda javí ako výhodnejšie. Celková hodnota Gini indexu je pre *D2* nižšia ($GI = 0{,}333$) než pre rozdelenie *D1* ($GI = 0{,}375$). Podľa klasifikačnej chyby ME sú však obe rozdelenia rovnako vhodné ($ME = 0{,}25$). Na základe hodnôt GI by bolo v našom príklade vybrané rozdelenie *D2*. Dôvodom, prečo ME nedokázala rozlíšiť medzi rozdeľovaním, je citlivosť na zmeny v pravdepodobnostiach uzlov. GI a Entropia sú totiž oveľa viac citlivé na zmeny v pravdepodobnostiach kategórií v uzloch ako ME, a preto sú ako kriteriálne štatistiky vhodnejšie.

Na **Obr. 8** je zobrazený všeobecný priebeh jednotlivých indexov v závislosti na pravdepodobnosti kategórie. Môžeme si všimnúť, že hodnoty ME majú strmší priebeh.



Obr. 8 Priebeh kriteriálnych štatistík

2.2 Algoritmy založené na rozhodovacích stromoch

Existuje cela rada algoritmov založená na rozhodovacích stromoch, ktoré sa od seba líšia hlavne výberom kriteriálnej štatistiky a tým, či výsledný strom je alebo nie je binárny.

2.2.1 CART

Stromy typu CART (*Classification and Regression Trees*) sú vhodné pre kategoriálne i regresné úlohy a rastú na základe rekurzívneho **binárneho delenia**. Na začiatku tvorby stromu patria všetky pozorovania súboru do jedného uzla, čiže koreňa. Následne sú tieto pozorovania rozdelené do dvoch dcérskych uzlov, na základe hodnoty predikátora X, ktoré sú ďalej delené opäť binárne na ďalšie uzly.

Definícia 2.2.1a – Binárny rozhodovací strom

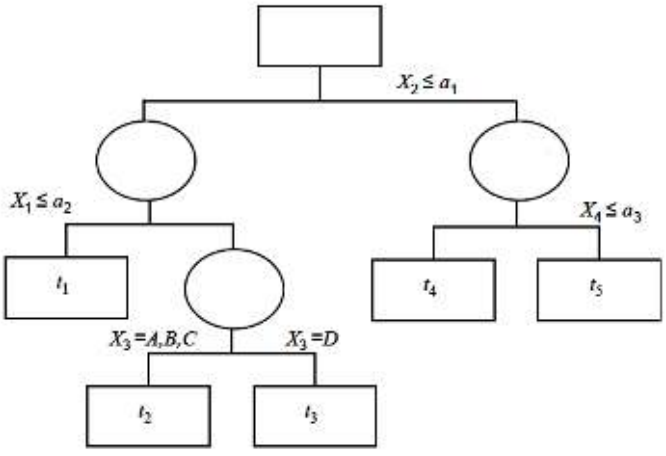
Binárny rozhodovací strom je binárny strom $G = (V, E)$ s nasledujúcimi vlastnosťami:

Vnútorný uzol v je asociovaný s:

- *atribútom* $d_v \in \{1, \dots, n\}$
- *prahom* $\theta_v \in \mathbb{R}$

Listový uzol je asociovaný s:

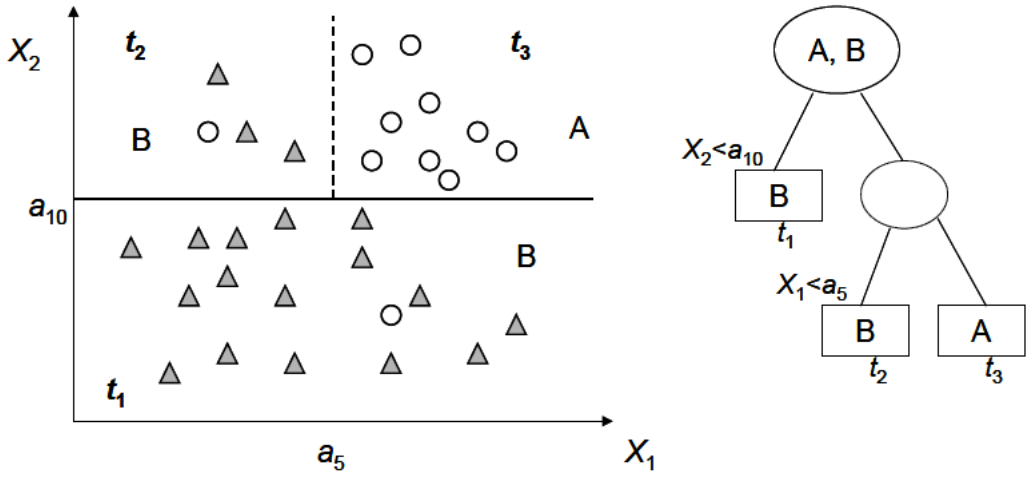
- *triedou* $c_v \in \{1, \dots, C\}$



Obr. 9 Grafická štruktúra rozhodovacieho stromu CART

Indexy u terminálnych uzlov udávajú, v akom poradí došlo k oddeleniu jednotlivých listových uzlov. Prediktory X_1, X_2 a X_4 sú spojité, prediktor X_4 je kategoriálny s kategóriami A, B, C, D.

Hodnoty vysvetľujúcich premenných, použité pri vetvení, rozdeľujú daný priestor na sadu pravouholníkov (**Obr. 10**), ktoré sa označujú aj ako regióny R_t.



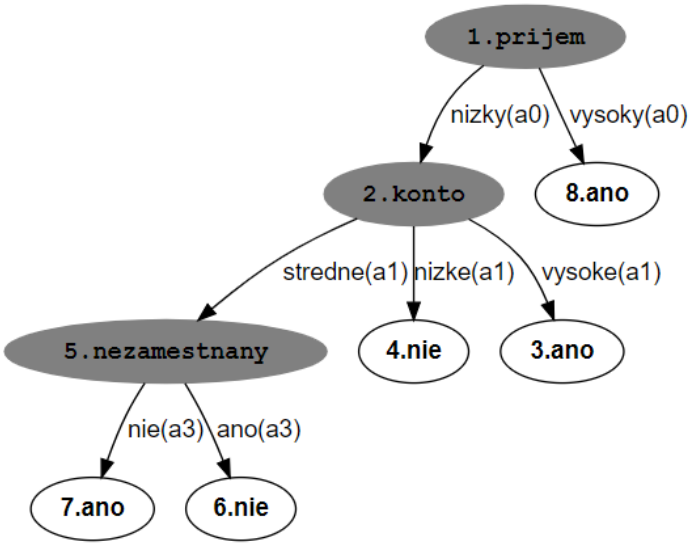
Obr. 10 Grafické znázornenie rozdelenia pozorovaní do kategórií A a B závislej premennej Y s použitím dvoch spojitých prediktorov X_1, X_2

Zložitosť konštrukcie takéhoto binárneho stromu je $\boldsymbol{O}(m * n\, log(n))$, kde m je dimenzia rozhodovacieho priestoru a n označuje počet vzoriek.

2.2.2 ID3

ID3 (*Iterative Dichotomiser 3*) je algoritmus používaný na vytvorenie rozhodovacieho stromu, ktorý vynašiel Ross Quinlan. Tento algoritmus vytvára rozhodovací strom z pevného súboru príkladov. Vzorky daného datasetu majú niekoľko atribútov a každý príklad patrí do triedy (napr. "yes" alebo "no"). Listy rozhodovacieho stromu obsahujú názov triedy, zatiaľ čo nelistový uzol je rozhodovací uzol. Rozhodovací uzol sa vetví, podľa počtu možných hodnôt atribútov. Napríklad, ak má atribút "$konto$" hodnoty {"stredné", "nízke", "$vysoké$"}, uzol sa vetví na tri podstromy. Každý podstrom pracuje iba s tými dátami, ktoré majú v atribúte "$konto$" rovnakú hodnotu ako je hodnota asociovaná s hranou spájajúcou rodiča a daný podstrom. Výsledný strom na rozdiel od algoritmu CART nie je binárny.

Zložitosť konštrukcie takéhoto ID3 stromu je $\boldsymbol{O}(m * n)$, kde m je dimenzia rozhodovacieho priestoru a n označuje počet vzoriek (nie je potrebné usporiadanie čísel pri výbere prahu, ako v predchádzajúcom algoritme).



Obr. 11 Všeobecný rozhodovací strom

2.2.3 C4.5

Jedná sa o vylepšenie pôvodného algoritmu ID3. Nové funkcie (oproti ID3) sú:

- akceptujú ako spojité, tak aj diskrétne premenné,
- spracováva aj neúplné dátové vzorky,
- podporuje **prerezávanie** stromu, ktoré zmenšuje preučenosť modelu,
- rôzne hmotnosti môžu byť použité funkcie, ktoré obsahujú tréningové dáta.

Nevýhodou C4.5 algoritmu je, že konštruuje **prázdne vetvy,** v ktorých nie sú reálne dáta (algoritmus, vytvorí vždy všetky vetvy, podľa toho koľko má daný atribút možných hodnôt).

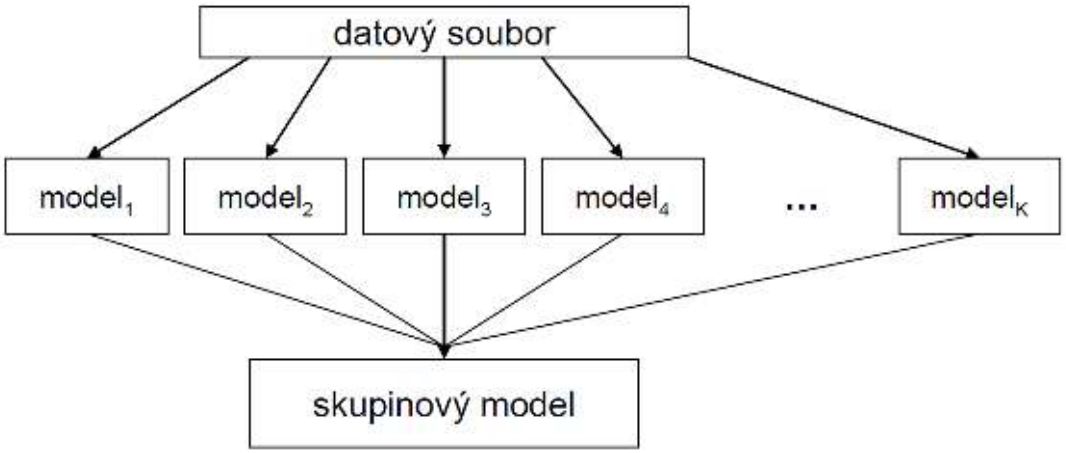
Zložitosť konštrukcie takéhoto stromu je $\boldsymbol{O}(m * n\, log(n))$, kde m je dimenzia rozhodovacieho priestoru a n označuje počet vzoriek.

3 Skupinové modely – ensemble learning

3.1 Všeobecný princíp

Skôr než sa dostaneme k jednotlivým typom rozhodovacích lesov a džunglí, pozrieme sa na problém trochu všeobecnejšie. Oba algoritmy totiž patria do skupiny tzv. **skupinových** (*ensamble*) **metód**. Princíp skupinových modelov je jednoduchý. Skupine modelov (napr. rozhodovacích stromov) zadáme rovnaký problém, na ktorom sa učia. Výstupy naučených modelov sa kombinujú (**Obr. 12**). Zaujíma nás, či kombináciou výsledkov modelov dosiahneme zlepšenie výsledkov predikcie či klasifikácie.

Výsledkom skupinového modelu je spriemerovanie všetkých výsledkov jednotlivých modelov v prípade regresie, alebo väčšinové hlasovanie jednotlivých modelov v prípade klasifikácie (aj u klasifikácie sa však môže použiť priemerovanie).



Obr. 12 Princíp skupinových modelov

Otázkou je, či kombináciou modelov môžeme získať presnejší model. Predpokladom pre použitie kombinácie modelov je podmienka, že jednotlivé modely musia byť rôzne. Ak by sme vytvorili sadu modelov na rovnakom dátovom súbore, pridaná hodnota bude nulová, lebo výsledný skupinový model bude zhodný s výsledkom jednotlivých modelov. Riešením je použiť rôzne súbory pre učenie modelu, ktoré získame náhodným výberom z tréningovej množiny dát. Modely tak budú vykazovať "odlišné" chyby. Presnosť a stabilita týchto modelov sa následne overuje na testovacích súboroch.

3.2 Rozklad na systematickú chybu a varianciu (*Bias-Variance Decomposition*)

Poďme sa detailnejšie pozrieť, čím všetkým je vlastne chybovosť modelu spôsobená. Začnime jednoduchým prípadom, kedy budeme merať náhodnú veličinu Y v populácii (napr. váha človeka) a chceme vyjadriť jej reprezentatívnu hodnotu pre celú populáciu. Hľadáme takýto odhad $\hat{Y}$, ktorý minimalizuje strednú hodnotu chyby $E_y(y-Y)^2$ cez celú populáciu.

V ideálnom prípade by sme zmerali všetky vzorky v populácii (odvážili všetkých ľudí) a zistili ich strednú hodnotu $E_y(y)$ (napr. priemer, medián), ktorú by sme vyhlásili za optimálny odhad. V praxi však tento prístup nie je možný a pomôžeme si výberom iba určitej skupiny pozorovaní z populácie, ktorý však musí mať rovnaké vlastnosti ako celá populácia. Takýto výber vytvoríme **náhodným výberom**.

Rovnaký prípad nastáva pri modeloch, kedy vyberáme pozorovanie pre tréningový súbor z množiny všetkých pozorovaní. Odchýlky pozorovaných od predikovaných hodnôt (presnosť klasifikácie či regresie) nebudú spôsobené iba "prírodnou" variabilitou, ktorú sme modelom nevysvetlili, ale aj rozdielom vo výsledkoch pre rôzne náhodné výbery.

Majme súbor trénovacích dát:

$$L = (y_i, x_i), i = 1, \dots, n$$

Chceme nájsť takú funkciu v priestore všetkých prediktorov a hodnôt závislej premennej, aby predikčná chyba bola malá. Ak majú (Y, X) rovnaké rozdelenie a daná funkcia R udáva rozdiel medzi pozorovanou hodnotou $\tilde{y}_i$ a predikovanou hodnotou $\tilde{y}_i$ závislej premennej Y, potom môžeme predikčnú chybu (*prediction error*) všeobecne vyjadriť ako:

$$PE(f, L) = E_{Y,X} R(Y, f(X, L))^2$$

kde $f(X, L)$ sú predikované hodnoty $\tilde{y}_i$ pre trénovací súbor L. Zvyčajne je závislá premenná Y jednorozmerná. Rozdelenie chyby na jednotlivé zložky modelu si predvedieme na regresii. Majme jednoduchú závislosť náhodnej veličiny Y na náhodnom vektora X:

$$Y = f(X) + \varepsilon ,$$

kde $f(X) = E(X|Y)$, $\mathrm{E}(\varepsilon|X) = 0$.

Y môžeme rozložiť na jej štrukturálnu časť $f(X)$, ktorá je predikovaná pomocou X a šum ε, ktorý nie sme schopní vysvetliť pomocou modelu. Priemerná všeobecná chyba (*mean-squared generalization error*) na trénovacom súbore L je rovná:

$$PE(f, L) = E_{Y,X}(Y - f(X, L))^2$$

Optimálny model by mal mať minimálnu priemernú chybu pre rôzne výbery L, čo v podstate znamená, že výsledky modelu pre jednotlivé výbery trénovacích súborov by sa nemali príliš líšiť.

Vyjadríme priemer trénovacích súborov rovnakej veľkosti s rovnakým rozdelením:

$$\bar{f}(x) = E_L f(x, L),$$

kde $E_L f(x, L)$ je priemer cez všetky trénovacie súbory L, predikované hodnoty $\widetilde{y_i}$ v hodnote x_i.

Celkovú chybu modelu môžeme rozdeliť na tri zložky, a to šum, skreslenie (bias) a varianciu:

$$PE = \underbrace{E\varepsilon^2}_{\check{s}um} + \underbrace{E_{Y,X}(f(X) - E_L f(X, L))^2}_{skreslenie^2} + \underbrace{E_{X,L}(f(X, L) - E_L f(X, L))^2}_{variancia}$$

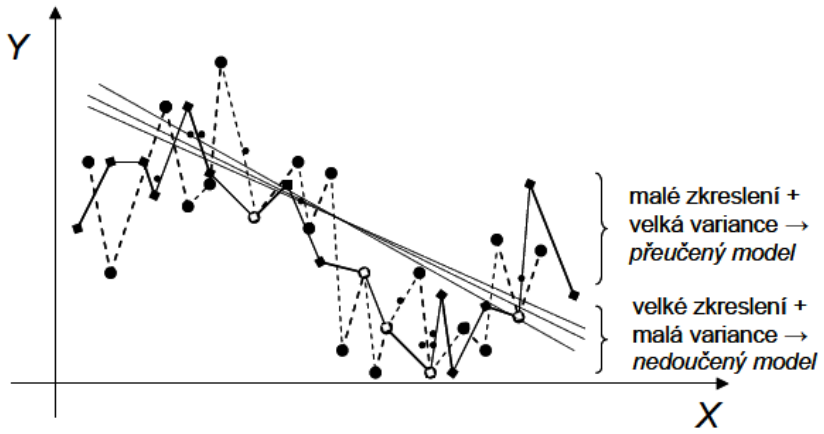
• $\check{S}um$ – je reziduálna chyba, čiže minimálna dosiahnuteľná chyba modelu, ktorú nie sme schopní modelom vysvetliť.

• $Skreslenie^2$ – určuje systematickú chybu modelu. Je to rozdiel optimálneho modelu od priemerného modelu.

• $Variancia$ – je variabilita výsledkov jednotlivých výberov, inými slovami, ako veľmi sa predikované hodnoty Y_i líšia v rámci trénovacích podvýberov L. Vysoká variancia značí preučený model.

Modely, ktoré sa používajú v skupinových modeloch, sa označujú ako slabé modely alebo *weak learners* (slabý žiak, u klasifikácie tiež slabý klasifikátor). Slabý model je definovaný všeobecne ako model, ktorý má malé skreslenie, ale vysokú varianciu. Slabé modely teda majú veľmi vysokú presnosť, ale iba pre pozorovanie z trénovacieho súboru. Takýto model je väčšinou preučený a nemá všeobecnú

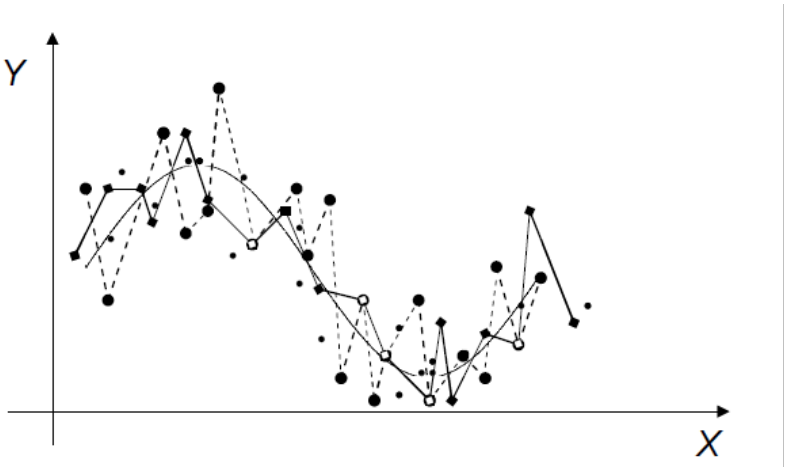
platnosť[17]. Príkladom slabých modelov s malým skreslením, ale vysokou varianciou môže byť interpolácia bodov pomocou lineárnych splajnov (**Obr. 13**). Čiarkovanou a plnou čiarou sú znázornené interpolácie bodov pre dva trénovacie súbory. Vidíme, že splajny spájajú všetky body, ale takéto rozloženie bodov nereflektuje variabilitu súboru, kedy jednotlivé pozorovania nemusia byť presne zmerané. Modely sú preučené. Majú vysokú presnosť, ale variancia pre výsledky na rôznych trénovacích súboroch je veľká. Naopak, ak je skreslenie veľké a variancia nízka, dostávame model, ktorý má nízku presnosť a nevysvetľuje dobre závislosť v dátach. Takýto model je nedoučený a na **Obr. 13** je znázornený troma priamkami, ktoré vznikli pre tri náhodné výbery z množiny trénovacích dát.



Obr. 13 Ukážka modelu s malým skreslením a vysokou varianciou modelu s veľkým skreslením a nízkou varianciou.

Hľadáme teda model, ktorý by mal nízku variáciu aj skreslenie. Kombinovaním niekoľkých slabých modelov môžeme znížiť obe tieto zložky.

Ako na to? Vráťme sa k príkladu na **Obr. 13**. Vytvorme náhodným výberom trénovacie súbory, ktoré obsahujú $2/3$ všetkých pozorovaní. Všetky body z trénovacieho súboru prekryjeme pomocou lineárnych splajnov. Na **Obr. 14** sú opäť znázornené výsledky pre dva trénovacie súbory. Tento postup opakujeme až do vopred stanoveného počtu trénovacích súborov (napr. 1000). Získali by sme tisíc pokrytí pomocou splajnov. Následne dôjde ku spriemerovaniu výsledkov predikcie zo všetkých slabých modelov, čím získame celkový výsledok (na **Obr. 14** ho predstavuje sínusoida), ktorý má malé skreslenie aj variáciu.



Obr. 14 Spriemerovaním slabých modelov získame výsledný model

Všeobecnejšie: Majme *k*-tý slabý model v tvare:

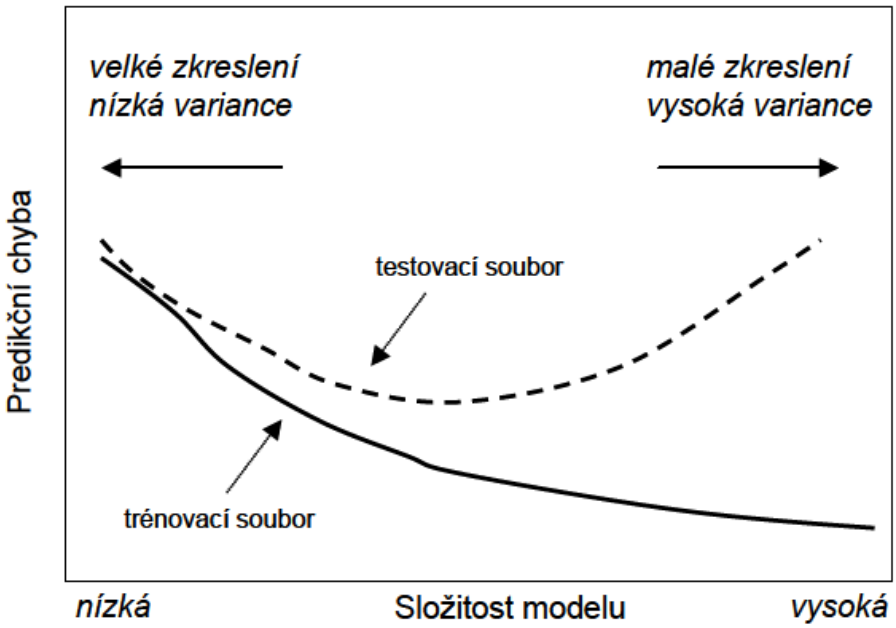
$$f_k(x, L) = f(x, L, \Theta_k),$$

kde Θ_k je náhodný vektor, na základe ktorého sa vyberú pozorovania pre *k*-tý slabý model. Vektory Θ_k sú nezávislé, s rovnakým rozdelením. Ak N je celkový počet pozorovaní trénovacieho súboru, každý Θ_k vyberie náhodne $2N/3$ pozorovaní. Hodnoty $y(n), x(n)$ pre vybrané n , sú odstránené z trénovacieho súboru. Skupinový model je rovný:

$$F(x, L) = \frac{1}{K}\sum_k f(x, L, \Theta_k) ,$$

kde K je celkový počet slabých modelov. Rozhodovacie stromy sú dobrými kandidátmi pre použitie v skupinových modeloch. Neprerezané stromy majú totiž vysokú presnosť pre trénovací súbor (teda nízky bias), ale vysokú variabilitu (výsledky medzi testovacím a trénovacím súborom sa líšia). Rozhodovacie stromy, na ktoré nie sú aplikované metódy pre hľadanie optimálnej veľkosti stromu, sú teda podľa vyššie uvedenej definície slabými modelmi.

Pripomeňme ešte, že u rozhodovacích stromov sme pre určenie ich optimálnej veľkosti museli takisto nájsť kompromis medzi varianciou a skreslením (**Obr. 15**).



Obr. 15 Výber optimálneho stromu pomocou testovacieho a trénovacieho súboru

Označenie *skupinové modely* sa občas používa aj pre kombináciu výsledkov z rôznych modelov (napr. Neurónových sietí, rozhodovacích stromov a regresie) na rovnakom súbore dát.

4 Náhodné lesy pre klasifikáciu a regresiu

"Looking inside the black box is necessary" Leo Breiman

V predchádzajúcej kapitole sme si ukázali, že kombináciou viacerých modelov môžeme získať model, ktorý bude všeobecnejší a presnejší ako jednotlivé modely. S myšlienkou použitia viacerých stromov pre spresnenie klasifikácie a predikcie prišiel prvýkrát Breiman [9] a vytvoril tak les. Lesy sú teda nadstavbou nad rozhodovacími stromami. Môžu byť použité pre klasifikáciu i regresiu a odstraňujú niektoré problémy, ktoré nastávajú pri použití stromov, najmä ich nestabilitu. Možno si však ľahko domyslieť, že les je zložitejší a menej prehľadný ako jeden strom a informácie jednotlivých stromov sa stratia v rámci celého lesa. Niekedy je táto metóda vďaka svojej "nejasnosti" označovaná ako čierna skrinka (*black box*). Strata jednoduchosti však nie je dôvodom na nepochopenie pozadia metódy.

Existuje niekoľko typov lesov. Najviac využívanou metódou je **Random Forest** [10]. Je to spôsobené najmä množstvom ďalších informácií mimo klasifikáciu a predikciu, ktoré možno z tohto lesa získať.

Táto technika bola vyvinutá pre súbory obsahujúce veľké množstvo prediktorov a veľmi dobre funguje aj na malých dátových súboroch. Random Forest je určený pre klasifikáciu i regresiu, ale aj meranie významnosti premenných, či detekciu odľahlých hodnôt.

Náhodný les sa skladá zo súboru stromov $T_{1,\dots,}T_N$, ktorých klasifikačné alebo regresné funkcie možno vyjadriť ako $h(X,\Theta_1),\dots,h(X,\Theta_N)$, kde h je funkcia, X je prediktor a $\Theta_1,\dots,\Theta_N$ sú nezávislé rovnako rozdelené náhodné vektory. Pre metódu Random Forests sa používajú binárne stromy typu CART. Podobne ako pri tvorbe jednotlivých stromov sa aj tu používa rozdelenie na testovací a trénovací súbor. Trénovacie súbory pre jednotlivé stromy T_i sú tzv. Bootstrapové výbery z dátového súboru L. Bootstrapové výbery sú náhodnými výbermi s opakovaním o veľkosti n. Tvorbu výberu s opakovaním výberov si možno jednoducho predstaviť ako žrebovanie čísel. Množinou všetkých pozorovaní by boli všetky čísla, z ktorých sa bude losovať. Pri žrebovaní sa náhodne vytiahne určitý počet čísel. Každé číslo je po jeho vylosovaní vrátené späť, môže teda byť opäť vybrané. Vopred stanovený počet vyžrebovaných

čísel tvorí bootstrapový výber. Po žrebovaní sú všetky čísla vrátené späť a nový ťah (nový výber) začína opäť so všetkými číslami. Takto je možné pri každom ťahu vybrať rovnaký počet čísel bez toho, aby sa nám znižovala veľkosť množiny pôvodných pozorovaní. Týmto spôsobom možno rozdeliť aj veľmi malé súbory na veľký počet trénovacích a testovacích súborov.

Bootstrapové výbery ale majú jednu nevýhodu, jednotlivé výbery nie sú nezávislé ako napríklad u krosvalidacie, lebo do bootstrapového výberu sú niektoré pozorovania vybrané opakovane a niektoré naopak vôbec. Počet pozorovaní, ktoré sa do bootstrapového výberu nedostanú je približne 37% [1] [9].

Pozorovanie, ktoré sú v *i*-tom bootstrapovom výbere L_i sa použijú pri tvorbe stromu T_i (trénovací súbor), naopak pozorovania, ktoré sa do toho výberu nedostali (testovací súbor) sú použité k odhadu jeho chyby. Odhady chyby na testovacom súboru sa nazývajú **OOB** (*out-of-bag, out of bootstrap sample*) odhady. Celkový počet **OOB** pozorovaní tvorí $1/3$ dátového súboru.

Pri použití lesov pre klasifikáciu získame z každého stromu informáciu o zaradení každého pozorovanie do výslednej kategórie. Výsledok klasifikácia lesa je daný väčšinovým hlasovaním všetkých stromov.

$$\hat{C}_{rf} = v\text{äčš}inove\_hlasovanie\{\hat{C}_i(x)\}_1^N ,$$

kde $\hat{C}_i(x)$ je výsledok klasifikácie *i*-teho stromu.

Ak je les použitý pri regresii, predikcie každého pozorovania je jednoduchým priemerom zo všetkých stromov.

$$\hat{f}_{rf}(x) = \frac{1}{N}\left(\sum_{i=1}^{N} T_i(x)\right)$$

[1] Počet opakovaní má pre jednotlivé pozorovania z $\boldsymbol{L}$ (pre $\boldsymbol{n} \to \infty$) Poissonovo rozdelenie so strednou hodnotou 1. Pravdepodobnosť, že pozorovanie nebude vybrané, je približne $\boldsymbol{e - 1 \approx 0.37}$.

Náhodný les zvyšuje presnosť (znižuje skreslenie) tým, že necháva narásť stromy dosť veľké, zároveň udržuje znesiteľnú variánciu kombinovaním výsledkov jednotlivých stromov (väčšinové hlasovanie / priemerovanie). Oproti ostatným lesom je tu však snaha zaistiť tiež nízku koreláciu medzi jednotlivými stromami. Ak totiž tvoríme výbery s opakovaním, ktoré nie sú navzájom nezávislé, budú výsledné stromy korelované. Táto "podobnosť" jednotlivých stromov môže viesť k nadhodnoteným výsledkom klasifikácie či predikcie. Zníženie korelácie medzi stromami sa dosiahne náhodným výberom iba určitého počtu prediktorov. Pre každý strom sa tak najlepšie vetvenie pre daný uzol hľadá iba z m prediktorov $X_1, \dots, X_m$. Náhodný les teda používa ako náhodný výber pozorovaní, tak náhodný výber prediktorov.

Algoritmus tvorby lesa môžeme popísať nasledovne:

1. Vytvor bootstrapovu podskupinu L_i s veľkosťou N – trénovací súbor.
2. Vyber náhodne m prediktorov.
3. Vytvor strom T_i na bootstrapovom súbore L_i iba s použitím m náhodne vybraných prediktorov. Rast stromu sa zastaví, až keď strom dosiahne minimálne hodnoty veľkosti uzla.
4. Zaraď OOB pozorovaní (testovací súbor) vytvoreným stromom a urči výslednú klasifikačnú triedu (kategóriu) alebo predikciu všetkých OOB pozorovaní.

Krok 1 – 4 sa opakuje do konečného počtu stromov v lese.

5. Spočítaj celkový výsledok klasifikácie / predikcie celého lesa väčšinovým hlasovaním / spriemerovaním.

Pre algoritmus tvorby lesa je potrebné vybrať správny počet premenných (m) pre náhodný výber a počet stromov (*ntree*) v lese. Určenie týchto parametrov je do istej miery experimentálne a vyžaduje skúsenosti. Klasickou cestou je vykonanie radu experimentov s rôznym nastavením týchto parametrov na získanie lesa, ktorý má najmenšiu celkovú chybovosť. Vzhľadom k časovo náročnému testovaniu (najmä ak ide o súbory obsahujúce tisíce záznamov) je vhodné vybrať taký počet stromov, ktorý bude dostačujúci pre optimálnu klasifikáciu. Na začiatku teda nastavíme počet stromov v lese na vyššiu hodnotu (napr. 20-násobok počtu prediktorov). Po určitom čase začínajú

stromy konvergovať k správnej hodnote OOB odhadu. Minimálnu veľkosť lesa možno určiť ako počet stromov, kedy sa chyba OOB odhadu s pribúdajúcimi stromami už nemení. Ďalším parametrom je počet náhodne vybraných prediktorov p. Pre náhodné lesy je doporučené nasledujúce nastavenie:

• pre klasifikáciu je hodnota $m = p$ a minimálna veľkosť uzla je jedna;

• pre regresiu je hodnota $m = p/3$ a minimálna veľkosť koncového uzla je päť.

Vyššie uvedené hodnoty slúžia ako defaultné nastavenie vo väčšine softvérov. V praxi však určenie počtu prediktorov závisí od riešeného problému a parameter m je vhodné zvoliť podľa výsledkov testovania modelov s rôznym nastavením. Vyberieme také m, pri ktorom má výsledný les najmenšiu chybovosť. Vzhľadom k tomu, že stromy nemožno pretrénovať, je počet prediktorov najdôležitejšou hodnotou, ktorú musíme zvoliť, pretože počet stromov nás obmedzuje len časovo.

Definícia 5a – Rozhodovací les

Rozhodovací les $J = \{G_1, \dots, G_m\}$ je skupinovým modelom tvoreným rozhodovacími stromami G_i.

Klasifikácia – Dátový bod $\boldsymbol{x}_i \in \mathbb{R}^n$ je priradený triede, ktorá dostane najviac hlasov.

5 Rozhodovacie džungle

V tejto kapitole rozdiskutujeme rozhodovacie džungle (skupinový model rozhodovacích DAG–*directly acyclic graph*), ako boli navrhnuté v Shottonovom článku [1]. Začneme s formálnym úvodom, v ktorom porovnáme koncept DAG-u s rozhodovacím stromov. Potom prediskutujeme rozhodovacie džungle, kde sa zameriavame na optimalizačný algoritmus.

Neskôr predstavíme aj vlastný model všeobecnej (nebinárnej) rozhodovacej džungle, s hlavným zameraním na kategorické dáta, ktorý sa pri porovnaní s Pohlenovými výsledkami pôvodného modelu ukazuje ako efektívnejší.

Naše príspevky sú:

1) *zovšeobecníme pojem DAG-u v oblasti strojového učenia z binárneho acyklického grafu na všeobecný acyklický graf,*
2) *navrhneme nový optimalizačný algoritmus pre všeobecný DAG,*
3) *overíme tvrdenia Shottona, že rozhodovacie džungle prinášajú lepšiu generalizáciu (vyššia presnosť) ako náhodné lesy pri súčasnom znižovaní spotreby pamäte,*
4) *zhotoviť vizualizáciu acyklických grafových štruktúr v rozhodovacom procese,*
5) *zverejníme implementáciu pre všeobecné rozhodovacie džungle.*

Definícia 5a – Rozhodovacia džungľa

Rozhodovacia džungľa $J = \{G_1, \dots, G_m\}$ je skupinovým modelom tvoreným rozhodovacími DAG-mi G_i.

Klasifikácia – Dátový bod $\boldsymbol{x}_i \in \mathbb{R}^n$ je priradený triede, ktorá dostane najviac hlasov.

5.1
5.1 Binárny rozhodovací DAG - directed acyclic graph

V tejto kapitole rozoberieme DAG ako ho predstavil Shotton [1]. Začneme definíciou, potom poukážeme na výhody DAG-u oproti rozhodovacím stromom, ukážeme kódovanie kategorických údajov aby boli vhodným vstupom pre DAG a nakoniec opíšeme učiaci proces DAG-u, zachytený v LSEARCH algoritme.

Definícia 5.1a – Rozhodovací DAG - Shotton

Rozhodovací DAG (*directed acyclic graph*) je orientovaný acyklický graf $G = (V, E)$, v ktorom platí, že všetky usmernené cesty od koreňa r k uzlu v sú rovnakej dĺžky a pre tieto uzly $v \in V$ platí, že:

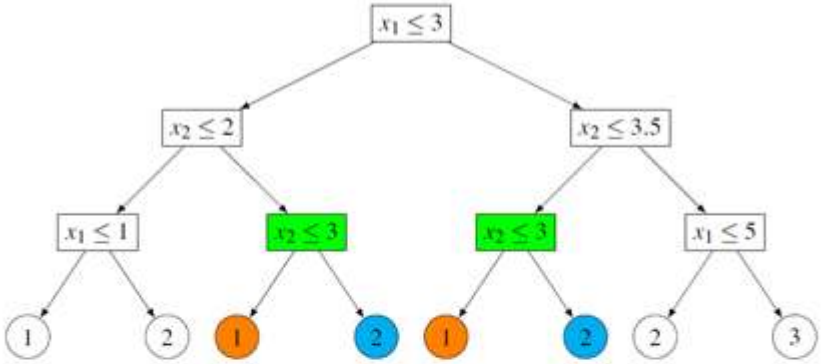
Vnútorný uzol je asociovaný s:

- *atribútom* $d_v \in \{1, \dots, n\}$
- *prahom* $\theta_v \in \mathbb{R}$
- *ľavým dcérskym uzlom* $l_v \in V$
- *pravým dcérskym uzlom* $r_v \in V$

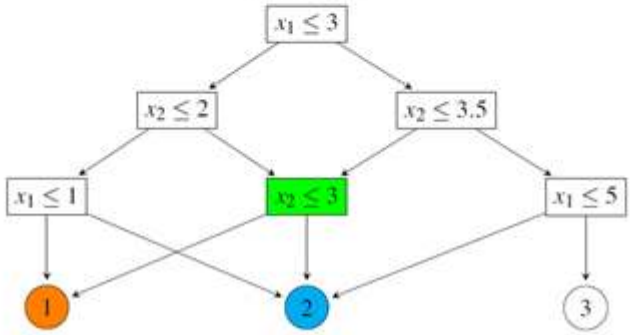
List je asociovaný s označením triedy $c_v \in \{1, \dots, C\}$

Učenie alebo tréning rozhodovacieho DAG-u spočíva v nájdení optimálnych parametrov $(d_v; \theta_v; l_v; r_v)$ pre každý uzol v. Učenie rozhodovacieho DAG-u je podstatne náročnejšie než učenie binárneho rozhodovacieho stromu, pretože parametre l_v a r_v (t.j. štruktúra grafu) sú fixné pre stromy. Z tohto dôvodu je potrebné optimalizovať len atribút d_v a prah θ_v pre stromy.

Shotton predstavil koncept rozhodovacieho DAG-u ako pamäťovo efektívnejšiu alternatívu k binárnym rozhodovacím stromom. **Obr. 16a** a **Obr. 16b** ukazujú rovnaký kvalifikátor. Na **Obr. 16a** je klasifikátor vyjadrený cez binárny rozhodovací strom a na **Obr. 16b** je vyjadrený ako rozhodovací DAG. To prináša nejakú intuíciu toho, že rozhodovací DAG skutočne zaberá menej pamäti ako binárny rozhodovací strom pri zachovaní zhruba rovnakej klasifikačnej sily.



Obr. 16 a



Obr. 16 b

Obrázky ukazujú rovnaký kvalifikátor pre dvojrozmerný problém klasifikácie do troch tried. Hlavným rozdielom medzi týmito dvoma modelmi je, že uzly DAG-u môžu mať viac ako jedného rodiča.

5.1.1 Kategorické dáta

Doteraz sme považovali naše dátové vzorky $x \in X$ za reálne vektory. V praxi sa však často stretávame s kategorickými údajmi. Kategorické vzorky x predstavujú súbor diskrétnych údajov, ktoré nevykazujú akúkoľvek algebrickú štruktúru. Riešenie, ktoré navrhuje *Shotton* vyzerá nasledovne:

Definícia 5.1.1a – Kódovacia funkcia

Nech $\times_{i=1}^{n} \Omega_i$ je vektorový priestor, v ktorom je definovaný klasifikačný problém kategorických údajov. Kódovacia funkcia ϕ je definovaná nasledovane:

$$\phi\colon \times_{i=1}^{n} \Omega_i \mapsto \times_{i=1}^{n} \mathbb{R}^{|\Omega_i|}, (x_1, \dots, x_n) \mapsto (\mathbb{I}(x_1, \omega_{1,1}), \dots, \mathbb{I}(x_1, \omega_{1,|\Omega_1|}), \dots, \mathbb{I}(x_n, \omega_{n,1}), \dots, \mathbb{I}(x_n, \omega_{n,|\Omega_n|}))$$

kde $\Omega_i = \{\omega_{i,1}, \dots, \omega_{i,k}\}$ a pre funkciu $\mathbb{I}$ platí:

$$\mathbb{I}(\mathrm{x}, \mathrm{y}) = \begin{cases} 1 & ak\ x = y \\ 0 & inak \end{cases}$$

Túto kódovaciu funkciu použil T. Pohlen na transformáciu kategorického datasetu CONNECT 4 vo svojom porovnávacom teste, čím trojnásobne zvýšil dimenziu pôvodných dát (zo 42 na 126).

Príklad 1

Nech $\Omega_1 = \{hot, cold\}$ a $\Omega_2 = \{winter, spring, summer, fall\}$ sú možne hodnoty atribútov 1 a 2. Vektor $x = (hot, spring) \in \Omega_1 \times \Omega_2$ má nasledovné kódovanie:

$$f(hot, spring) = (1, 0, 0, 1, 0, 0)$$

5.1.2 Shottonov optimalizačný algoritmus

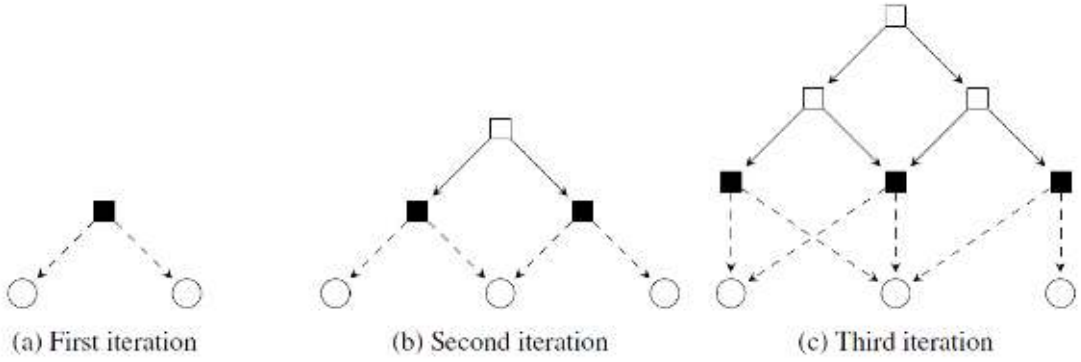
V tejto časti prezentujeme tréningový algoritmus navrhnutý Shottonom na skonštruovanie rozhodovacích DAG-ov.

Tradične sa rozhodovacie stromy učia rozdelením uzlov prahovaním jedného z atribútov. Aby sa učenie rozhodovacieho DAG-u generalizovali, Shotton formuloval problém ako minimalizáciu chybovej funkcie (*objective function*). Chybová funkcia poskytuje skóre (číselné ohodnotenie) pre nastavenia všetkých parametrov. Preto minimalizácia chybovej funkcie znamená, že sa musia určiť parametre vedúce k najmenšiemu skóre. Ak cieľová funkcia nie je konvexná[2], nájdenie týchto optimálnych parametrov môže byť ťažké. V takýchto prípadoch sa veľa tréningových algoritmov pokúša nájsť len lokálne minimá.

Rovnako ako bežné stromové tréningové algoritmy, Shottonova metóda pre učenie rozhodovacieho DAG-u funguje následným rozdelením uzlov. Pre rozhodnutie DAG-u je však potrebné získať viac parametrov ako pri rozhodovacích stromoch. Konkrétnejšie, pre každý uzol je potrebné navyše vybrať ľavý a pravý detský uzol. Pre stromy sú tieto parametre dané štruktúrou grafu.

Algoritmus učenia rozhodovacieho DAG-u sa učí po úrovniach(*level-by-level*). Pri každej iterácii pridáva ďalšiu úroveň uzlov, optimálne atribúty bývalých listových uzlov, ktoré sú teraz vnútornými uzlami. **Obr. 17** znázorňuje koncept *level-by-level* učenia.

[2] Konvexná funkcia má jediné globálne minimum



Obr. 17 Koncept *level-by-level* učenia

Pri každej iterácii algoritmu sa do grafu a atribútov pridá nová úroveň uzlov z predchádzajúcich listových uzlov. Obrázok zobrazuje prvé tri iterácie. Štvorcové uzly predstavujú vnútorné uzly a kruhové zas uzly listov. Plné uzly sú tie, ktorých parametre aktuálne optimalizujeme.

Aby sme mohli formálne diskutovať o algoritme, musíme najprv uviesť niekoľko notácií. Nech $G = (V; E)$ je rozhodovací DAG a nech $v \in V$. S_v budeme označovať množinu tréningových vzoriek na úrovni v (tzn. tréningové príklady z pôvodného tréningového súboru, ktoré boli odovzdané od rodičov v).

Pri každej iterácii, nazývame predchádzajúcu úroveň listových uzlov rodičovská úroveň a novo pridanú úroveň, úroveň deti. Analogicky, nazývame jednotlivé uzly **rodičovské uzly** a **detské uzly**. Chybová funkcia je definovaná ako **entropia** (kapitola 2. 1. 3) na úrovni dieťaťa.

Nech $p_1, \dots, p_k \in V$ sú rodičovské uzly a nech $c_1, \dots, c_l \in V$ sú deti. Chybová funkcia E, ktorú chceme minimalizovať, je definovaná ako

$$E(\Theta_{p_1}, \dots, \Theta_{p_k}) = \sum_{i=1}^{l} |S_{c_i}| H(S_{c_i})$$

kde $\boldsymbol{\theta_{p_i}} = (d_{p_i}; \theta_{p_i}; l_{p_i}; r_{p_i})$ sú parametre rodičovských uzlov ako v **definícii 5.1a**.

Z tohto vzťahu nie je úplne zjavné ako chybová funkcia závisí od parametrov rodičovských uzlov, ktoré sa snažíme optimalizovať.

Pozrime sa teda na to, ako je definované S_{c_i} pre detské uzly c_i.

S_{c_i} je množina tréningových vzoriek, ktoré prechádzajú od rodičov k c_i:

$$S_{c_i} = \bigcup_{j=1,\dots,k:l_{p_j}=c_i} \left\{(\mathbf{x}, y) \in S_{p_j} : x_{d_{p_j}} \leq \theta_{p_j}\right\} \cup \bigcup_{j=1,\dots,k:r_{p_j}=c_i} \left\{(\mathbf{x}, y) \in S_{p_j} : x_{d_{p_j}} > \theta_{p_j}\right\}$$

Z tejto formulácie jasne vidieť, že každé S_{c_i} – a preto aj E – priamo závisí od nastavenia parametrov rodičovských uzlov.

Teraz budeme prezentovať optimalizačný algoritmus **LSEARCH**, ktorý navrhol Shotton a kol. Algoritmus je určený na nájdenie lokálneho minima chybovej funkcie. Optimalizuje každú úroveň DAG jednotlivo. Okrem algoritmu LSEARCH navrhli aj alternatívny algoritmus **CLUSTERSEARCH**.

Tvrdia však, že funguje permanentne horšie ako algoritmus LSEARCH. Z tohto dôvodu sa v práci zaoberáme len algoritmom LSEARCH.

Algoritmus 2 LSEARCH

1: **function** LSEARCH($\Theta_{p_1}, \dots, \Theta_{p_k}$)
2: **while** nejaká zmena **do**
3: **for** i = 1, ... , k **do**
4: $\mathcal{F} \leftarrow$ výber atribútu (random feature selection)
5: $(d_{p_i}, \theta_{p_i}) \leftarrow$arg min $_{d\in\mathcal{F},\theta\in R} E(\Theta_{p_1}, \dots, \Theta_{p_{i-1}}, (d, \theta, l_{p_i}, r_{p_i}), \Theta_{p_{i+1}}, \dots, \Theta_{p_k})$
6: **for** i = 1, ... , k **do**
7: $l_{p_i} \leftarrow$ arg min $_{l=c_1,\dots,c_l} E(\Theta_{p_1}, \dots, \Theta_{p_{i-1}}, (d_{p_i}, \theta_{p_i}, l, r_{p_i}), \Theta_{p_{i+1}}, \dots, \Theta_{p_k})$
8: $r_{p_i} \leftarrow$ arg min $_{l=c_1,\dots,c_l} E(\Theta_{p_1}, \dots, \Theta_{p_{i-1}}, (d_{p_i}, \theta_{p_i}, l_{p_i}, r), \Theta_{p_{i+1}}, \dots, \Theta_{p_k})$
9: **return** $\boldsymbol{\Theta_{p_1}, \dots, \Theta_{p_k}}$

Vzhľadom na inicializáciu všetkých parametrov $\Theta_{p_1}, \dots, \Theta_{p_k}$ **Algoritmus 2** optimalizuje jeden parameter a ponechá všetky ostatné parametre pevné. To považujeme za *greedy* krok s vysokým aproximačným pomerom. V **kapitole 5.2.2** navrhujeme vlastný optimalizačný algoritmus (pre všeobecný DAG), ktorý optimalizačný problém rieši „viac globálne“.

5.2 Zovšeobecnený DAG

Hlavným dôvodom, prečo si myslíme, že binárny DAG je potrebné zovšeobecniť, je vysoký nárast dimenzionality dát po použití kódovacej funkcie opísanej v **kapitole 5.1.1**.

Uvažujme dáta popísané v **príklade 1.** Atribút $\Omega_1 = \{hot, cold\}$ má 2 možné hodnoty a $\Omega_2 = \{winter, spring, summer, fall\}$ má 4 možné hodnoty. Pre ľubovoľné pozorovanie z pôvodných dát x_i platí $dim(x_i) = 2$, avšak $dim(\mathrm{f}(x_i)) = 6$. Kódované dáta sa teda trojnásobne zväčšili.

Vo všeobecnosti sa dimenzia po kódovaní mení takto:

$$\dim(x_i) \rightarrow \sum_{i=1}^{\dim(x_i)} |\Omega_i|$$

Túto funkciu je však nutné použiť ak chceme v binárnom acyklickom grafe pracovať s kategorickou premennou. Zväčšenie dimenzie dát má však za následok nárast hĺbky stromu, čo priamo súvisí so zväčšením pamäťových nárokov (exponenciálny nárast počtu uzlov).

Binárny DAG, ako ho navrhol Shotton vychádza z myšlienky algoritmu **CART (kapitola 2.2.1)**, ktorý buduje binárne rozhodovacie stromy. My sa budeme snažiť opísať koncept všeobecného DAG-u, hoci vychádza, z viac zastaralého, pôvodného algoritmu rozhodovacích stromov – **ID3**. To nám umožní pracovať s kategorickými premennými priamo, bez kódovania. Navyše o takom strome platí, že v ľubovoľnej ceste od koreňa po list sa už vybraný rozhodovací atribút neopakuje. **Teda hĺbka stromu je ohraničená dimenziou dát**, zatiaľ čo hĺbka binárnych rozhodovacích stromov nemá žiadne ohraničenie.

Definícia 5.2a – Rozhodovací DAG – všeobecný graf

Rozhodovací DAG (*directed acyclic graph*) je orientovaný acyklický graf $G = (V, E)$, pre ktorý platí, že:

Vnútorný uzol v je asociovaný s:

- *atribútom* $d_v \in \{1, \dots, n\}$
- *s množinou* $deti_v = \{child_i \in V \mid i \in \Omega_{d_v}\}$

List je asociovaný s *označením triedy* $c_v \in \{1, \dots, C\}$

5.2.1 Numerické dáta

Hlavnou nevýhodou rozhodovacích stromov generovaných algoritmom **ID3** je, že nevedia pracovať s numerickými dátami. Ak by sme náš zovšeobecnený model postavili na neupravenej idei algoritmu ID3, tiež by nedokázal pracovať s numerickými

dátami. Spôsob, ako budeme pracovať s numerickými dátami bude založený na tom, že najprv spojitú premennú rozdelíme na nespojité – diskrétne body.

Naivným algoritmom ako to spraviť by bolo rozdelenie spojitej premennej na intervaly s rovnako veľkým rozsahom.

Povedzme, že by sme mali spojitú premennú v rozsahu od 1 – 100. Hodnoty, ktoré táto premenná nadobúda by sme rozložili na povedzme na 10 intervalov s rovnako veľkým rozsahom ($I_1 = <1, \dots ,10>$, $I_2 = <11, \dots ,20>$, … $I_{10} = <91, \dots ,100>$). Naša nová diskrétna premenná by v tomto prípade mala 10 kategórií[3], s ktorými už vieme pracovať.

Tento prístup by mohol fungovať, keby sme mali zaručenú rovnomernú distribúciu bodov spojitej premennej. Predstavme si situáciu, že by naša spojitá premenná vyzerala takto[4]: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4 181, 6 765, 10 946, 17 711, 28 657, 46 368, 75 025, 121 393.

Rozsah premennej je od 1 – 121 393. Skúsme opäť túto premennú rozložiť na 10 intervalov: I_1 = {1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987}, I_2 = {1597, 2584, 4 181, 6 765, 10 946, 17 711}, I_3 = {28 657}, I_4 = { }, I_5 = { }, I_6 = { }, I_7 = {75 025}, I_8 = { }, I_9 = { }, I_{10} = {121 393}.

Môžeme vidieť, že kým prvý a druhý interval obsiahol väčšinu čísel, ostatných 7 intervalov obsahuje jedno číslo alebo sú prázdne, čo teda nie je dobrá diskreditácia (uzly by mali prázdne vetvy, čo je zbytočné a pamäťovo náročné).

Použijeme teda sofistikovanejší postup a to klastrovací algoritmus, ktorý pridelí kategóriu každému bodu zo spojitej premennej, pričom rozdelenie bude rovnomerné bez ohľadu na distribúciu. Za klastrovací algoritmus pre náš 1-dimenzionálny problém sme zvolili *k-means.*

Algoritmus *k-means* patrí do rodiny klastrovacích algoritmov, ktoré sme spomenuli v kapitole 1.1.2, kde sme sa zaoberali učením bez dozoru. Tento algoritmus je najpoužívanejší algoritmus nehierarchickej zhlukovej analýzy. Rieši problém

---

3 Touto úpravou zo spojitej na diskrétnu premennú, nemeníme počet bodov v premennej, len každý bod zo spojitej premennej nahradíme jednou z desiatich kategórii (1, 2,.., 10). Napr. body zo spojitej premennej 1, 5, 46, 53, 93, 97, 100 vieme zobraziť cez príslušnosť do intervalov I_1, I_2 ,..., I_{10} na body z diskrétnej premennej 1, 1, 4, 5, 9, 9, 10.

4 Prvých 25 členov Fibonacciho postupnosti.

klastrovania (zhlukovania), ktorý vytvára zhluky prvkov s podobnými vlastnosťami. Zhlukovanie sa podobá klasifikácii, ale na rozdiel od nej zaraďuje prvky do tried bez označenia (bez *štítkov*).

Priebeh algoritmu k-means:

Algoritmus k-means funguje v dvoch krokoch.

Prvým je **priraďovanie**. Najprv sa do priestoru so zhlukmi pridá niekoľko centroidov, čo sú body definujúce jednotlivé zhluky (neskôr by sa mali nachádzať v strede zhluku). Centroidy sú pridané na náhodné pozície a mal by ich byť rovnaký počet, ako počet zhlukov. Centroidy sa pri krokoch algoritmu môžu pohybovať, ale dátové body musia zostať nehybné. Body sa zaradia do zhluku, ktorého centroid je bodu najbližšie.

Druhým krokom je **optimalizácia**. V tomto kroku sa centroidy posunú do stredu oblasti, v ktorej sú umiestnené body ich kategórie. Následne sa im znova priradia najbližšie body. Poloha centroidu sa znova prepočíta tak, aby tvoril ťažisko svojho zhluku. Tento proces sa opakuje, pokiaľ sa poloha centroidov neustáli. Po tomto procese sa centroidy z priestoru odstránia a vzniknú označené zhluky, ktorých body majú podobné vlastnosti.

Tieto zhluky budú reprezentovať naše kategórie – možné hodnoty novej – diskrétnej premennej.

Voľba počtu centroidov

Algoritmus *k-means* je jednoduchý algoritmus, ktorý sám nevie určiť optimálny počet centroidov. Pre nás je však počet zhlukov veľmi dôležitý údaj, ktorý nám po diskreditácii bude hovoriť, koľko rôznych hodnôt bude môcť nadobúdať rozhodovací atribút (prediktor).

Hoci je *k-means* algoritmom, ktorý sa učí bez dozoru (bez *y* – predikovanej premennej), my túto informáciu (keďže riešime problém klasifikácie) máme.

Navrhujeme teda túto predikovanú premennú využiť pri optimalizácií počtu centroidov. Idea postupu na výber počtu centroidov je jednoduchá.

Na každý bod v jednom klastri, sa môžeme pozerať ako na zodpovedajúcu hodnotu predikovanej premennej *y*. Čím je entropia týchto hodnôt menšia, tým istejšie vieme povedať, že klaster obsahuje navzájom podobné body.

Stačí teda zvyšovať počet klastrov, kým súčet entropií jednotlivých zhlukov klesá. Benefitom takéhoto postupu je, že neklastrujeme len na základe hodnôt, ktoré chceme diskretizovať, ale naše kastrovanie zachytáva aj možnú koreláciu (aj keď len štatistickú) medzi klastrovanými dátami a závislou premennou *y*.

5.2.2 Algoritmus učenia pre všeobecný DAG

Ako sme už spomínali, učenie binárneho rozhodovacieho DAG-u spočíva v nájdení optimálnych parametrov $\Theta_v = (d_v; \theta_v; l_v; r_v)$ pre každý uzol v.

Pre všeobecný DAG ide o nastavenie týchto parametrov:

- $\boldsymbol{d_v}$ – atribút, ktorého výberom získame najlepšie rozdelenie dát v ďalšej úrovni, k čomu nám takisto poslúži entropia.
- $\boldsymbol{out(v)}$ – množina všetkých hrán vychádzajúcich z vrcholu v. V prípade binárneho DAG-u ide len o dve hrany l_v a r_v, ktoré rozdeľujú dáta na dve skupiny podľa pravdivosti predikátu v rodičovskom uzle. V tomto prípade budeme optimalizovať nastavenia všetkých hrán, ktoré z rodičovského uzla vychádzajú.

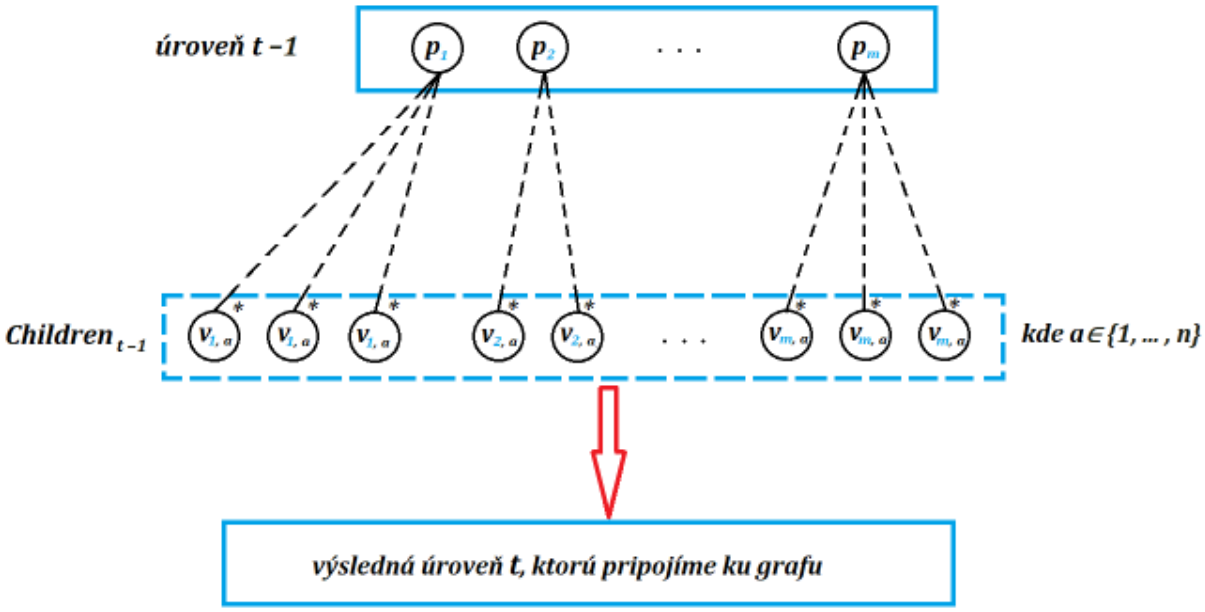
Zjavné je, že na rozdiel od binárneho DAG-u, vo všeobecnom DAG-u nám odpadá povinnosť hľadať nastavenia pre prah θ_v, ktorým sme delili dátový priestor vždy na dve skupiny. Avšak pribúda nám povinnosť namiesto parametrov l_v a r_v zaoberať sa celou množinou hrán vychádzajúcou z vrcholu $v - out(v)$.

Namiesto **LSEARCH** algoritmu na optimalizáciu parametrov navrhujeme použiť vlastný optimalizačný algoritmus, ktorý bude využívať tzv. **optimalizačnú tabuľku**.

Nech $\boldsymbol{x}_i \in \mathbb{R}^n$ a úroveň $t-1$ nech obsahuje m uzlov. Nech $Children_{t-1}$ je množina všetkých detí uzlov z úrovne $t-1$.

O každom $v_{i,j} \in Children_{t-1}$ uchovávame tieto informácie:

- $\boldsymbol{X_t, y_t}$ – *dáta, ktoré* $t-1$ *krát boli rozdeľované nejakým rozhodovacím atribútom,*
- $\boldsymbol{p_i}$ – *rodič uzla* $v_{i,j}$ *z úrovne* $t-1$ *, kde* $i \in \{1, \dots, m\}$
- $\boldsymbol{a_j}$ – *atribút, ktorým budeme dátový priestor deliť na úrovni* t*, kde* $j \in \{1, \dots, n\}$.



Obr. 18

Úroveň t rozhodovacieho DAG-u vybudujeme pomocou **optimalizačnej tabuľky**. Optimalizačná tabuľka je tabuľka o rozmeroch $m \times n$, ktorá obsahuje $|Children_{t-1}|$ hodnôt[5]. Každý uzol $v_{i,j} \in Children_{t-1}$ sa zapíše do optimalizačnej tabuľky hodnotou $maxOccurrance(y_t)$[6] na pozíciu (i, j).

Ak už máme takúto optimalizačnú tabuľku, úroveň t získame tak, že do nej pridáme uzly z množiny $Children_{t-1}$ s tým, že spojíme do jedného uzla všetky tie uzly, ktoré sú v rovnakom stĺpci tabuľky a zároveň majú rovnaké hodnoty.

To, že spájame uzly len v rámci jedného stĺpca, značí, že spojíme uzly, ktoré si vybrali rovnaký rozhodovací atribút. Dôvodom, prečo však musíme uvažovať aj rovnakú zapísanú hodnotu, je to, že rozhodovací atribút sa nemusí snažiť odseparovať – očistiť od „iných“ rovnakú hodnotu z y(závislá premenná).

Napríklad, ak by $y^1 = \{b, b, b, c\}$ a $y^2 = \{d, d, d, e\}$ mali nejaký spoločný rozhodovací parameter a, tak y^1a y^2 by boli v rovnakom stĺpci tabuľky. Parameter a (keďže je optimálny) by mohol rozdeľovať $y_1$na $\{b, b, b\}$ a $\{c\}$ a $y_2$na $\{d, d, d\}$ a $\{e\}$, čím by sme získali čisté (dokonale odseparované) dáta. Po spojení do spoločného uzla

[5] V tabuľke je zapísané každé dieťa rodičov z úrovne $t-1$.

[6] $maxOccurrance(y_t)$ vracia najčastejšie vyskytujúcu sa hodnotu $c \in \{1, \dots, C\}$ v množine y_t.

by sme mali $y_{1+2} = \{b, b, b, c, d, d, d, e\}$ a delením atribútom a by sme získali $\{b, b, b, d, d, d\}$ a $\{c, e\}$, čím získavame nečisté dáta a proces klasifikácie musí pokračovať ďalej.

Príklad 3

Uvažujme takúto optimalizačnú tabuľku z ktorej chceme vygenerovať úroveň t.

uzly/atribúty	a_1	a_2	a_3	a_4	a_5
p_1	c_1			c_3	c_2
p_2	c_3	c_2		c_1	c_2
p_3	c_1	c_2			c_2
p_4	c_3	c_2		c_2	c_2

Tab. 3 Optimalizačná tabuľka

Z počtu stĺpcov vieme, že $x \in R^5$ (lebo každý stĺpec reprezentuje jeden konkrétny rozhodovací atribút).

Tiež vieme povedať (podľa počtu riadkov), že úroveň $t-1$ obsahuje 4 uzly – $t-1 = \{p_1, \dots, p_4\}$

Tabuľka 3 obsahuje 14 hodnôt (keby sme budovali rozhodovací strom, úroveň t by obsahovala 14 uzlov), avšak, ak pospájame uzly, ktoré majú rovnaké hodnoty v rámci jedného stĺpca (tie hodnoty, ktoré majú rovnakú farbu[7]), úroveň t bude mať len 6 uzlov.

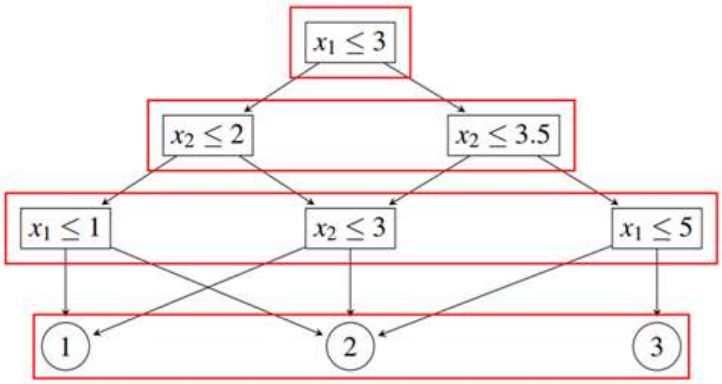
Z tabuľky sa dajú vyčítať aj prepojenia (hrany) medzi úrovňou $t-1$ a t.

Z prvého stĺpca získavame 2 uzly: $\{(\boldsymbol{p_1}, \boldsymbol{a_1}), (\boldsymbol{p_3}, \boldsymbol{a_1})\}$ a $\{(\boldsymbol{p_2}, \boldsymbol{a_1}), (\boldsymbol{p_4}, \boldsymbol{a_1})\}$**.** Prvý uzol bude dieťaťom rodičov p_1 a p_3 a rodičia druhého uzla sú p_2 a p_4. Obe tieto uzly sa budú rozhodovať na základe rozhodovacieho atribútu a_1 (keďže pracujeme s prvým stĺpcom).

[7] Rovnaká farba znamená v tomto prípade rovnaký reprezentant z množiny $\{1, \dots, C\}$

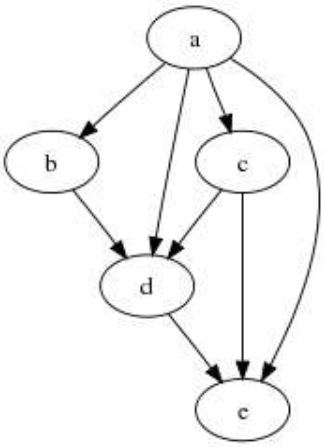
5.2.3 Redukcia listových uzlov

Shottonov návrh grafu predpokladá *level-wise* štruktúru. To znamená, že prepojenia smú byť optimalizované len v rámci dvoch za sebou idúcich úrovní, ako vidíme na **Obr. 18**.



Obr. 19 *level-wise* optimalizácia

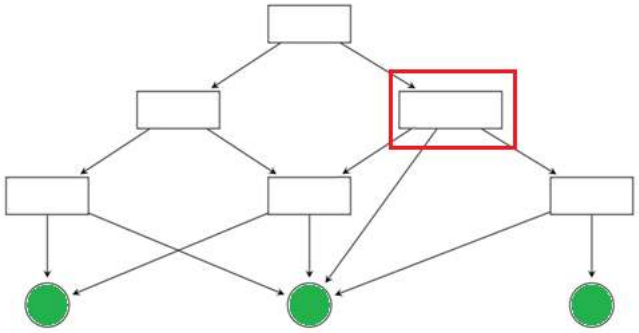
Level-wise topológia nie je jedinou možnou topológiou, ktorá spĺňa definíciu DAG-u. Príkladom je tento graf:



Obr. 20 Iný variant DAG-u

V našom modeli nebudeme implementovať myšlienku *Level-wise* tak dôsledne. **Vnútornému uzlu povolíme mať priameho potomka aj z iných úrovní, za podmienky, že potomok bude list.** To zabezpečí, že v rozhodovacom DAG budeme mať práve $|\{1, \ldots, C\}|$ listov (**Obr. 20**). Binárny DAG, ako ho navrhol Shotton, obsahuje zanedbateľný pomer listov oproti vnútorným uzlom (**Tab. 7**), avšak v našej všeobecnej

štruktúre DAG-u, ide približne o rovnaký pomer vnútorných a listových uzlov, a teda touto redukciou zmenšíme počet uzlov približne na polovicu.



Obr. 21 Schéma topológie všeobecného DAG-u s unikátnymi listami

6 Implementácia a výsledky

V poslednej časti popíšeme implementačné detaily nášho výstupu – programu, tiež popíšeme parametre jednotlivých klasifikačných metód (Rozhodovacie stromy, lesy a DAG a džungľa) a porovnáme naše výsledky s výsledkami Pohlenovho testu.

6.1 Implementačné detaily

Ako väčšina algoritmov strojového učenia, aj prezentované algoritmy prichádzajú s možnosťami „doladenia“ takzvanými hyperparametrami.

Pre stromy a DAG-y je možné nastaviť nasledujúce hyperparametre:

- *stop_value* – ak je v uzle počet vzoriek patriacej do jednej triedy (v percentách) $\geq$ ako hodnota *stop_value*, z uzla sa stáva list, strom sa ďalej nevetví.
- *max_deep* – maximálna hĺbka acyklického grafu.
- *metric* – hodnota $\epsilon \{entropia,\ gain\}$, ide o výber kriteriálnej štatistiky.

Skupinové metódy (les a džungľa) doplňujú tieto parametre:

- *n_weak_learners* – počet rozhodovacích stromov/DAG-ov v lese/džugli
- *predictors* – číslo v percentách, ktoré hovorí koľko náhodne vybraných prediktorov sa má použiť v procese učenia slabých žiakov.

Výstupom je teda implementácia spomenutých metód v jazyku Python. Ako vývojové prostredie bolo použité známe interaktívne prostredie *Jupyter Notebook.*

Súčasťou implementácie je aj vizualizačný grafový nástroj. Ukážky je možne vidieť v Prílohe B,C a D.

6.2 Výsledky

V poslednej kapitole popíšeme Pohlenov test Shottonovej rozhodovacej džungle, ukážeme jeho výsledky a následne budeme prezentovať výsledky nášho modelu všeobecnej džungle. Nakoniec, na výslednom grafe ukážeme porovnanie výkonu binárnej a všeobecnej džungle.

Pohlen vo svojej práci porovnal pamäťové nároky rozhodovacích lesov a džungli, ktoré obsahovali 15 slabých žiakov (stromov/DAG-ov). Cieľom bolo ukázať, že rozhodovacie džungle zaberajú signifikantne menej pamäte ako rozhodovacie lesy, pri súčasnom udržaní si rovnakej klasifikačnej sily (často lepšej, no často ide

o zanedbateľné rozdiely v presnosti). Pohlen deklaruje 50%-nú redukciu pamäte na testovacích dátach, s výnimkou datasetu SHUTTLE, v ktorom rozhodovacie lesy dosiahli nižšiu hodnotu veľkosti zaberanej pamäte.

Aby Pohlen dokázal určiť pamäťovú veľkosť klasifikačného modelu, musel určiť (odhadnúť) veľkosť jedného grafového uzla:

Typ uzla	**Veľkosť** (v bajtoch)	**Atribúty** (v bajtoch)
Vnútorný (strom)	16	*atribút* – 4, *prah* – 8 (double precision), *ľavé dieťa* – 4
Vnútorný (DAG)	20	*atribút* – 4, *prah* – 8, *ľavé dieťa* – 4, *pravé dieťa* – 4
List (strom/DAG)	*C**4	*označenie triedy*

Tab. 4

Pomocou tejto tabuľky teda Pohlen prepočítal počet grafových uzlov na veľkosť pamäte, ktorú model zaberal (v bajtoch).

V ďalšej tabuľke Pohlen poskytuje stručnú (no dostačujúcu) charakteristiku dát [8]. Uvádzame informácie len o tých dátach, ktoré sa objavujú aj v našich testoch[9].

Data set	**Veľkosť**	**Dimenzia**	**Atribúty**
CONNECT 4	67 557/-	42(126)	kategorické
LETTER RECOGNITION	20 000/-	16	numerické
SHUTTLE	43 500/14 500	9	numerické
USPS	3 823/1 797	64	numerické

Tab. 5

[8] Všetky spomenuté datasety je možné nájsť na https://archive.ics.uci.edu/ aj s bližšou špecifikáciou.

[9] V Pohlenovom teste bol navyše testovaný dataset **MNIST** s dimenziou 784 – pre otestovanie sme však nemali dostatočnú výpočtovú silu.

A nakoniec, v posledných tabuľkách uvádza Pohlen výsledky – priemerný[10] počet uzlov a výslednú spotrebu pamäte + doplňujúcu informáciu o presnosti:

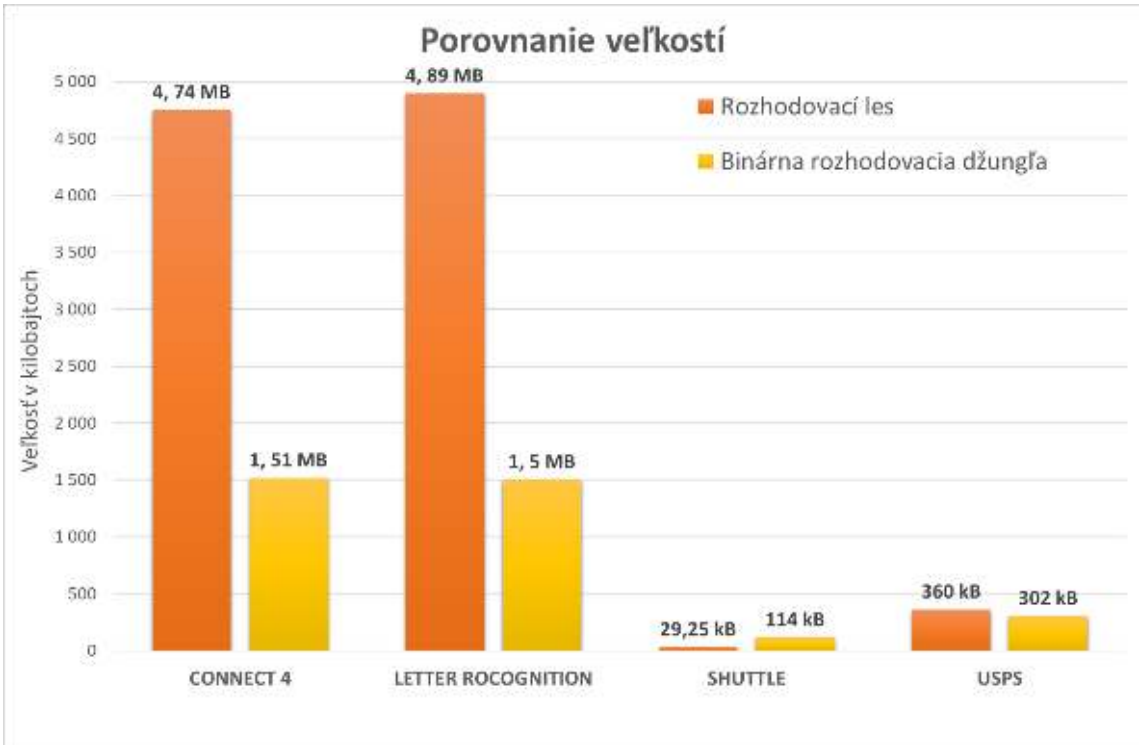
Data set	#Vnútorné uzly	#Listy	Veľkosť (v bajtoch)	Presnosť
CONNECT 4	169 604	169 619	4 749 103,20	81,50%
LETTER RECOGNITION	40 776	40 791	4 894 704	95,6%
SHUTTLE	804	819	29 251,20	99,9%
USPS	6 433	6 433	360,859	95%

Tab. 6 Rozhodovací les – výsledky

Data set	#Vnútorné uzly	#Listy	Veľkosť (v bajtoch)	Presnosť
CONNECT 4	74 633	1 920	1 515 700	81,2%
LETTER RECOGNITION	69 485	1 075	1 501 532	95,7%
SHUTTLE	4 896	802	113 980	99,9%
USPS	13 113	1 012	302 748	95%

Tab. 7 Rozhodovacia džungľa – výsledky

Výsledný graf porovnávania vyzerá takto:



Obr. 22

[10] Všetky uvedené výsledky sú spriemerované, algoritmy sa na dátach spúšťali viac krát.

V poslednej časti skúsime nasimulovať Pohlenov test, s tým, že namiesto Shottonových binárnych rozhodovacích džunglí použijeme náš zovšeobecnený model.

V nasledujúcej tabuľke odhadneme veľkosť jedného grafového uzla, aby sme boli schopný určiť pamäťovú veľkosť nášho modelu.

Typ uzla	**Veľkosť** (v bajtoch)	**Atribúty**
Vnútorný (všeobecný DAG)	16	*atribút* - 4 bajty, ~~*prah* - 8 bajtov~~, *prvé dieťa* - 4 bajty, *sused* - 4 bajty, *hrana (hodnota atribútu)* - 4 bajty
List (všeobecný DAG)	*C**4	*označenie triedy*

Tab. 8

Ako si môžeme všimnúť, oproti binárnemu DAG–uzlu, nám odpadá povinnosť započítať veľkosť prahu. Vo výslednom počítaní veľkosti modelu teda nebudeme počet uzlov násobiť 20 bajtami, ale len 16, čo má celkom signifikantný vplyv na výslednú pamäťovú hodnotu.

Charakteristika dát je rovnaká ako v **Tab. 5**, keďže náš model chceme otestovať na rovnakých dátach.

V nasledujúcej tabuľke máme teda konečné výsledky našej zovšeobecnenej rozhodovacej džungle, ktorú sme otestovali na spomenutých datasetoch:

Data set	**#Vnútorné uzly**	**#Listy**	**Veľkosť (v bajtoch)**	**Presnosť**
CONNECT 4	14 058	3	224 940	82,1%
LETTER RECOGNITION	13 622	26	218 056	96,3%
SHUTTLE	281	7	4 524	99,9%
USPS	2 245	10	35 960	90%

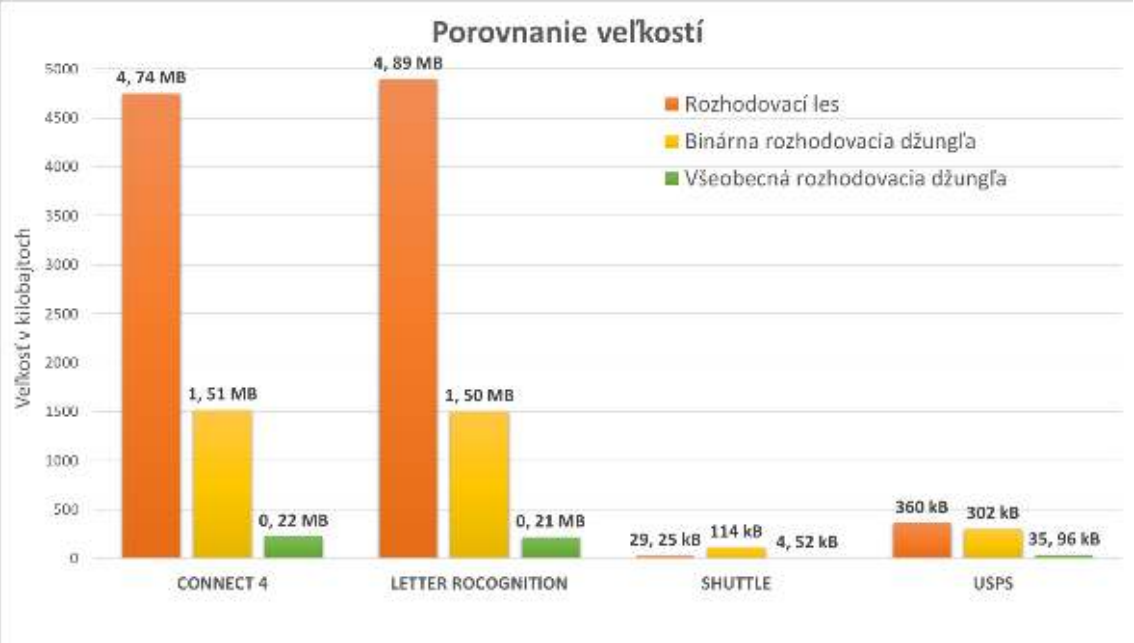
Tab. 9

Pre lepšiu prehľadnosť uvádzame tabuľku, ktorá popisuje pamäťové skóre všetkých troch testovaných modelov:

Data set	**Rozhodovací les**	**Binárna rozhodovacia džungľa**	**Všeobecná rozhodovacia džungľa**
CONNECT 4	4, 75 MB	1, 51 MB	0, 22 MB
LETTER RECOGNITION	4, 89 MB	1, 50 MB	0, 21 MB
SHUTTLE	0, 03 MB	0, 11 MB	0, 005 MB
USPS	0, 36 MB	0, 3 MB	0, 035 MB

Tab. 10

A nakoniec pre lepšiu predstavu uvádzame celkové grafické porovnanie:



Obr. 23

V nasledujúcej tabuľke udávame, v akom pomere došlo k redukcii pamäte pri použití jednotlivých typov džunglí namiesto rozhodovacích lesov (v zátvorkách uvádzame percentuálne zlepšenie, t.j. koľko percent pamäte sme ušetrili pri použití džungle). V poslednom stĺpci tabuľky uvádzame pomer zlepšenia pri použití nami navrhnutej všeobecnej džungle v porovnaní s binárnou džungľou. Posledný riadok tabuľky zobrazuje priemerné zlepšenie v rámci všetkých datasetov.

Data set	$\frac{Rozhodovací\ les}{Binárna\ džungľa}$	$\frac{Rozhodovací\ les}{Všeobecná\ džungľa}$	$\frac{Binárna\ džungľa}{Všeobecná\ džungľa}$
CONNECT 4	3,14 (68%)	21,60 (95%)	6,86 (85%)
LETTER ROCOGNITION	3,26 (69%)	22,23 (95%)	6,82 (85%)
SHUTTLE	~~0,27 (-266%)~~	6 (83%)	22 (95%)
USPS	1,2 (16%)	10,28 (90%)	8,57 (88%)
Priemerné zlepšenie	**51%**[11]	**91%**	**88%**

Tab. 11

[11] Pri počítaní priemerného zlepšenia sme do úvahy nebrali dataset SHUTTLE, ktorý pri výpočtoch prejavoval atypické správanie a jeho výsledky by negatívne vplývali na výpočet priemeru, ktorý by tým stratil svoju výpovednú hodnotu

Záver

Jedným z cieľov tejto práce bolo poskytnúť prehľad o grafových acyklických metódach v úlohe klasifikácie v strojovom učení. Po úvodnom vymedzení základných pojmov z oblasti strojového učenia a zadefinovaní problému klasifikácie, sme opísali obľúbený klasifikačný algoritmus – rozhodovací strom. Kompletnú implementáciu algoritmu môžeme nájsť v prílohe (v jazyku Python). Tento silný klasifikátor má mnoho kladných vlastností, ktoré sme v práci spomenuli, avšak objavujú sa aj slabiny algoritmu. Model má tendenciu modelovať tréningové dáta tak podrobne, že dospeje do takého štádia, kedy z dát neabstrahuje, ale kopíruje špecifické črty dát (model je preučený). Presnosť takého to klasifikátora na testovacích dát je veľmi slabá.

Túto slabinu rieši princíp *ensemble learning*-u, ktorý je v práci vysvetlený v kapitole 3. Keďže skupinové modely – v našom prípade rozhodovacie lesy, na zvýšenie generalizácie (teda aj presnosti) využívajú natrénovanie viacerých klasifikátorov, pamäťové nároky sa prudko zvyšujú. V poslednej kapitole sme testovali dáta algoritmom rozhodovacích lesov o veľkosti 15 stromov.

To samozrejme znásobilo pamäťové nároky. Rozhodovací les, ktorý modeloval dataset CONNECT 4 zaberal 4,75 MB, teda jeden rozhodovací strom mohol zaberať približne len 4, 75 MB / 15 = 316,7 kB.

Túto cenu za presnosť nie sú schopné zaplatiť niektoré hardvérové zariadenia, ako bolo spomenuté v úvode práce. Pohlen prichádza s testom rozhodovacích džunglí, ktoré sú schopné zredukovať v priemere 50 % veľkosti rozhodovacích lesov (**Tab. 6** a **Tab. 7**). Okrem predstavenia pôvodného Shottonovho algoritmu rozhodovacích džunglí sme v práci navrhli aj vlastný zovšeobecnený model džunglí s hlavným zameraním na zväčša kategorické dáta. Popísali sme aj vlastný optimalizačný algoritmus pomocou optimalizačnej tabuľky. Vo výsledných testoch sa však ukázalo, že všeobecný model džungle je možné použiť aj na numerické dáta. Celkovo, použitím nášho modelu namiesto rozhodovacích lesov ušetrili 91% pamäte (88% zlepšenie oproti binárnej džungle) ako sme uviedli v **Tab. 11**.

Za hlavný dôvod úspechu nášho modelu pokladáme, to, že sa nám podarilo obmedziť výšku acyklického grafu. Práve výška grafu má za následok exponenciálny nárast pamäťových nárokov acyklických klasifikačných modelov [1], [2]. Kým koncept Shottonovej džungle, zmenšuje šírku grafu, náš model súčasne obmedzuje výšku zhruba

na dimenziu dát (často vnútorné uzly skonvergujú na listové, ešte pred dosiahnutím spomenutej hĺbky).

Za ďalší dôvod zlepšenia pokladáme použitie vlastného optimalizačného algoritmu namiesto algoritmu LSEARCH. LSEARCH algoritmus optimalizuje prepojenia pre jeden uzol, ostatné uzly zafixuje. V ďalšej iterácií sa pre ďalší uzol vyberá už len zo zvyšných prepojení z rodičovskej úrovne. Náš algoritmus rieši optimalizačný problém hľadania prepojení hrán medzi dvoma úrovňami grafu pozerá viac globálne. Každému uzlu z rodičovskej úrovne dovolíme, aby sa podľa vlastných špecifikácií zapísal do optimalizačnej tabuľky a až potom na základe pozícií v tabuľke sú uzly spájané.

Za úspech pokladáme aj zlepšenie časovej zložitosti z $\boldsymbol{O}(m * n\, log(n))$ na $\boldsymbol{O}(m * n)$, kde m je dimenzia rozhodovacieho priestoru a n označuje počet vzoriek (v našej všeobecnej džungli nepracujeme s prahom a teda vzorky nemusíme preusporiadávať).

Možným pokračovaním by bolo predostrieť myšlienky všeobecnej rozhodovacej džungle aj na algoritmus, ktorý rieši regresný problém. Ďalej, vidíme potenciál vo využití genetických algoritmov pri hľadaní optimálnych hyperparametrov, a tak zlepšiť proces učenia a potenciálne aj výsledky.

Pohlen na konci svojej práce o rozhodovacích džungliach napísal, že rozhodovacia džungľa je alternatíva k rozhodovacím lesom, ktorá je prínosná, aktuálna a užitočná v oblasti strojového učenia. My s jeho tvrdením súhlasíme a dodávame, že naša všeobecná rozhodovacia džungľa sa stáva vhodnou alternatívou k Shottonovej džungli, hlavne (no nielen) ak sa jedná o zväčša kategorické dáta.

Zoznam použitej literatúry

[1] Jamie Shotton, Toby Sharp, Pushmeet Kohli, Sebastian Nowozin, John Winn, and Antonio Criminisi. Decision jungles: Compact and rich models for classification. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems 26, pages 234–242. Curran Associates, Inc., 2013.

[2] T. Pohlen, *Decision Jungles,* Seminar Report, Fakultät für Mathematik, Informatik und Naturwissenschaften Lehr – und Forschungsgebiet Informatik, 2014.

[3] K. Komprdová, *Rozhodovací stromy a lesy,* 2012, Brno, ISBN 978-80-7204-785-7.

[4] T. Hastie, *The Elements of Statistical Learning,* Stanford, Springer, 2008.

[5] E. B. Hunt, J. Marin, and P. T. Stone. *Experiments in Induction*. Academic Press, New York, 1966.

[6] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[7] K. V. S. Murthy and S. L. Salzberg. *On growing better decision trees from data*. PhD thesis, John Hopkins University, 1995.

[8] J.R. QUINLAN, *Induction of Decision Trees*, 1986, Machine Learning, 1:81-106

[9] Breiman, L.: Bagging Predictors. Machine Learning 24, 123--140 (1996)

[10] Breiman, L.: Random Forests. Machine Learning 45, 5--32 (2001)

Prílohy

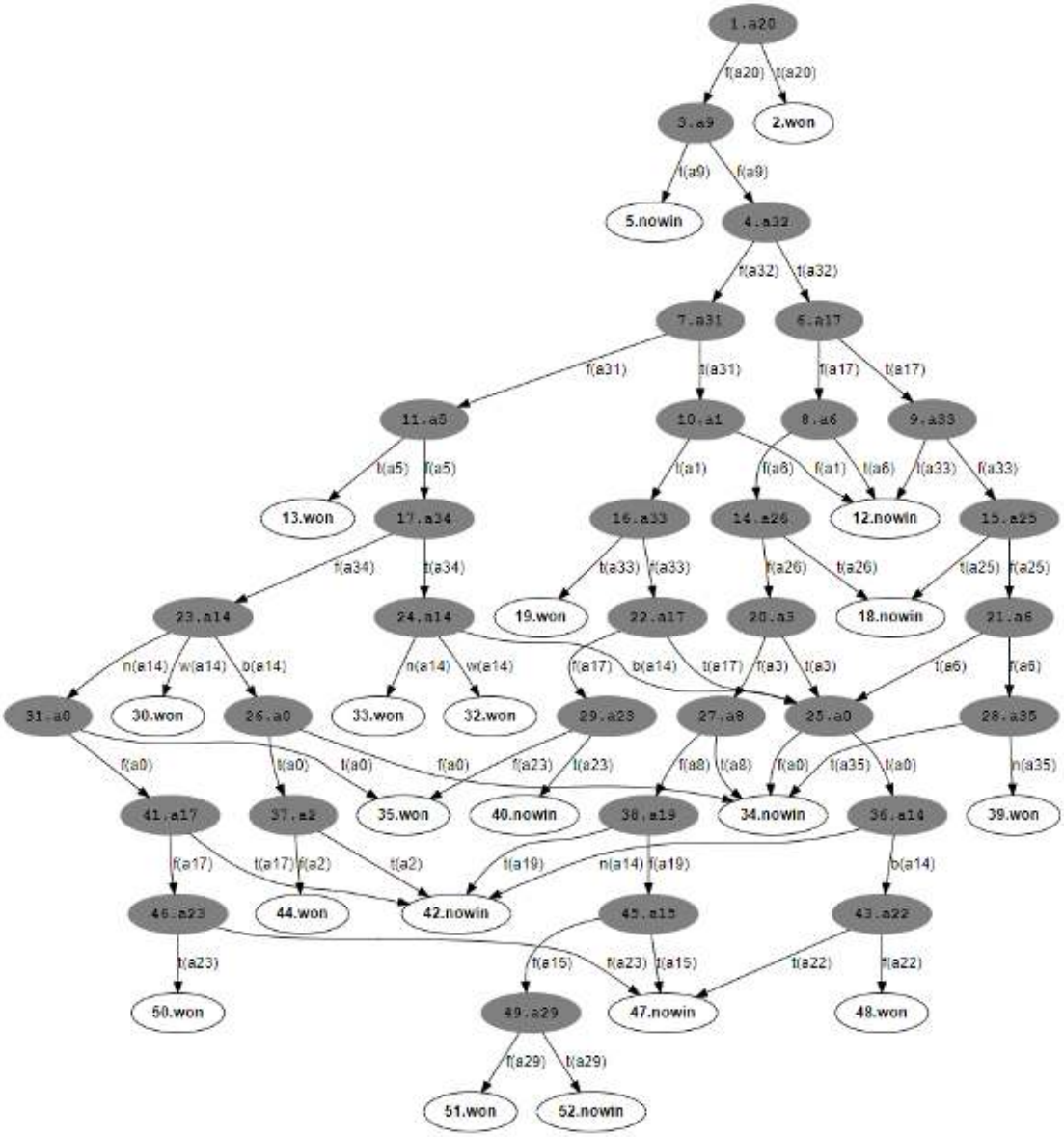
Príloha A: CD so súborom typu *Jupyter notebook,* obsahujúce kompletnú implementáciu všetkých spomenutých klasifikačných algoritmov

Príloha B: Vizualizácia DAG, dataset: Chess (King-Rook vs. King-Pawn), presnosť: 98, 75 %

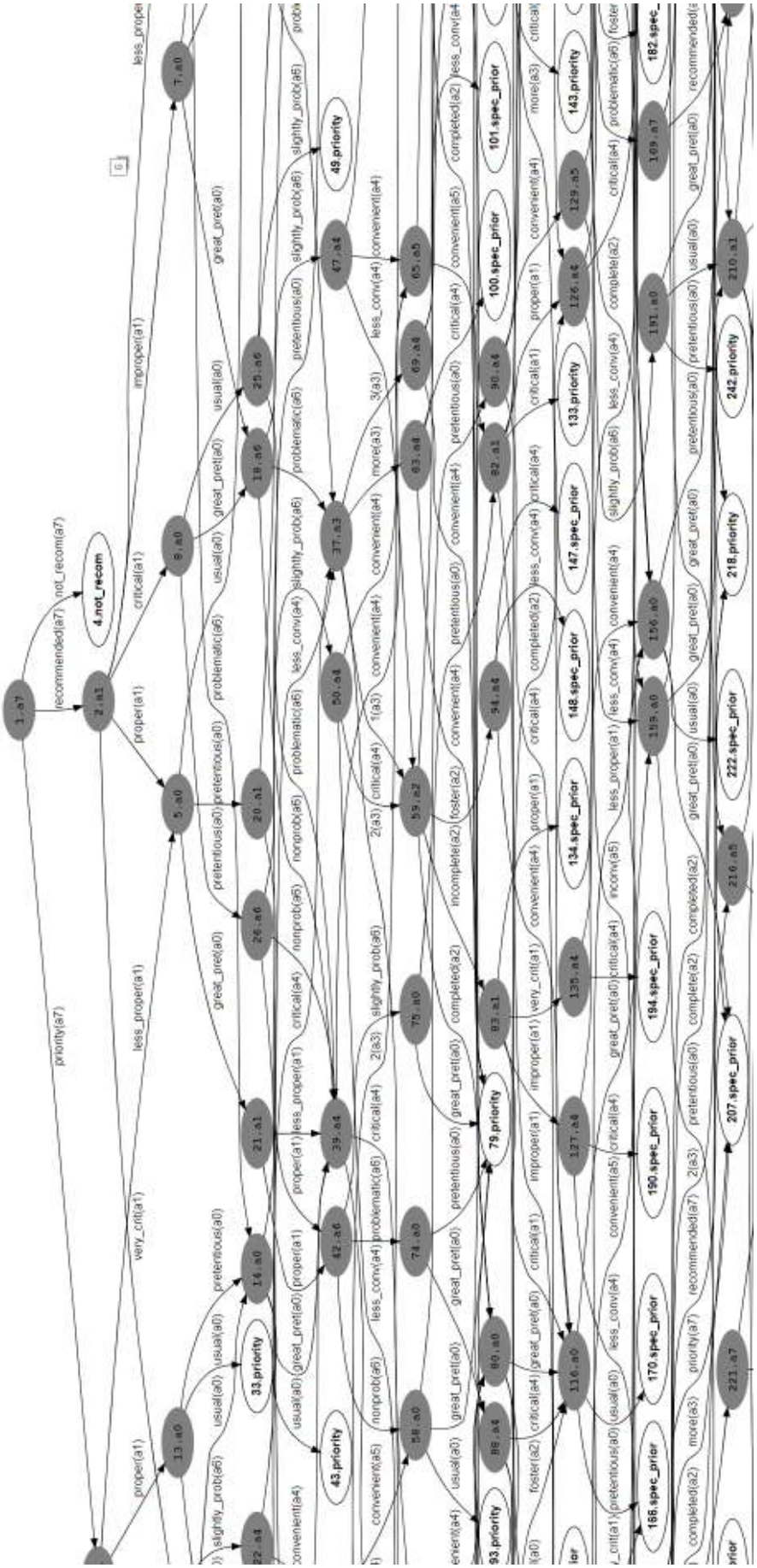
Príloha C: Vizualizácia DAG - výsek z grafu (asi 1/5 z celého grafu), dataset: Nursery, presnosť: 97, 5 %

Príloha D: Vizualizácia TREE, dataset: Heart Disease, presnosť: 80 %

Príloha B:


1.a20
f(a20)
t(a20)
3.a9
2.won
f(a9)
t(a9)
5.nowin
4.a32
f(a32)
t(a32)
7.a31
6.a17
f(a31)
t(a31)
f(a17)
t(a17)
11.a5
10.a1
8.a6
9.a33
t(a5)
f(a5)
t(a1)
f(a6)
f(a1)
t(a6)
f(a33)
t(a33)
13.won
17.a34
16.a33
14.a26
12.nowin
15.a25
f(a34)
t(a34)
t(a33)
f(a33)
f(a26)
t(a26)
t(a25)
f(a25)
23.a14
24.a14
19.won
22.a17
20.a3
18.nowin
21.a6
n(a14)
w(a14)
b(a14)
n(a14)
w(a14)
f(a17)
b(a14)
t(a17)
f(a3)
t(a3)
f(a6)
t(a6)
31.a0
30.won
26.a0
33.won
32.won
29.a23
27.a8
25.a0
28.a35
f(a0)
t(a0)
t(a0)
f(a0)
f(a23)
t(a23)
f(a8)
t(a8)
f(a0)
t(a35)
t(a0)
n(a35)
41.a17
37.a2
35.won
40.nowin
38.a19
34.nowin
36.a14
39.won
f(a17)
t(a17)
f(a2)
t(a2)
t(a19)
n(a14)
f(a19)
b(a14)
46.a23
44.won
42.nowin
45.a15
43.a22
f(a23)
f(a15)
f(a23)
t(a15)
f(a22)
t(a22)
50.won
49.a29
47.nowin
48.won
f(a29)
t(a29)
51.won
52.nowin

Príloha C:



Príloha D: