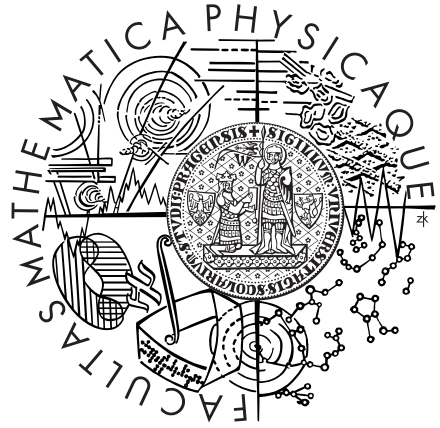
UNIVERZITA KARLOVA V PRAZE
MATEMATICKO-FYZIKÁLNÍ FAKULTA

DIPLOMOVÁ PRÁCE



Jakub Čechmánek

Rozhodovací stromy pro klasifikaci dat

Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Jana Štanclová, Ph.D.

Studijní program: Informatika

Studijní obor: Softwarové systémy

Praha 2008

Na tomto místě bych chtěl poděkovat všem, kteří mi pomáhali při vzniku této práce. Děkuji vedoucí mé diplomové práce RNDr. Janě Štanclové, Ph.D. za její cenné poznámky, připomínky a velmi obětavý přístup. Dále bych chtěl poděkovat celé mé rodině a své přítelkyni Elišce. Bez jejich podpory by tato práce nemohla vzniknout.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 18. dubna 2008 Jakub Čechmánek

Obsah

Název práce:	Rozhodovací stromy pro klasifikaci dat
Autor:	Jakub Čechmánek
Katedra (ústav):	Katedra softwarového inženýrství
Vedoucí diplomové práce:	RNDr. Jana Štanclová, Ph.D.
E-mail vedoucího:	stanclova@ksi.ms.mff.cuni.cz

Abstrakt:

K problémům týkajících se klasifikace dat je možné přistupovat různými způsoby. Mezi ty nejvýznamnější patří neuronové sítě, Bayesovské sítě, klastrování, lineární modely, asociační pravidla apod. Tato práce se zabývá rozhodovacími stromy, které si rovněž zaslouží pozornost mezi odbornou veřejností. Postupně budou popsány metody C4.5, CART a SDT stromy, které využívají teorii fuzzy množin. Podstatná část je také věnována ořezávacím algoritmům. Jednotlivé modely budou experimentálně ověřeny a vzájemně srovnány na volně dostupných datových množinách příznakových vektorů s ohledem na ukončovací kritéria, kritéria na dělení uzlu a velikost vzniklých stromů. Součástí experimentů je i zhodnocení vlastních výsledků.

Klíčová slova: rozhodovací stromy, ID3, C4.5, klasifikační a regresní stromy (CART), SDT stromy, ořezávání

Title:	Decision Trees for Data Classification
Author:	Jakub Čechmánek
Department:	Department of software engineering
Supervisor:	RNDr. Jana Štanclová, Ph.D.
Supervisor's e-mail address:	stanclova@ksi.ms.mff.cuni.cz

Abstract:

There is a lot of approaches for data classification problems resolving. The most significant data classification methods are neural networks, Bayes nets, clustering, linear models, associative rules, etc. This thesis deals with decision trees which deserves attention of experts as well. Step by step are discussed C4.5, CART and SDT trees, a variant of classical decision tree inductive learning using fuzzy sets theory. Substantial part of work is devoted to pruning algorithms as well. Particular methods are examined and compared over freely available data sets of feature vectors with respect to stopping criteria, splitting criteria of a node and size of constructed trees. A summary of our own results is included.

Keywords: decision trees, ID3, C4.5, classification and regression trees (CART), SDT trees, pruning

Kapitola 1

Úvod

Klasifikace dat (též rozpoznávání dat) spadá do oblasti tzv. strojového učení. Hlavním úkolem strojového učení je vytvořit autonomní systém, který není závislý na lidském faktoru. Pojem klasifikace dat zahrnuje široké spektrum problémů týkajících se zpracování informací. Používá se v řadě oborů jako je rozpoznávání obrazu nebo rozpoznávání řeči. S rozpoznáváním dat je možné se setkat také v medicíně např. při stanovování diagnózy, při detekci chyb ve strojích apod.

Cílem klasifikace dat je identifikovat objekty, které jsou popsané příznaky, do tříd. Rozhodovací stromy jsou jednou z nejoblíbenějších technik, které se uplatňují v oblasti klasifikace dat. Důvodů pro to je několik.

Hlavní důvod spočívá v jejich přehlednosti a snadné interpretovatelnosti, která umožňuje uživatelům rychle a snadno vyhodnocovat získané výsledky, identifikovat klíčové položky a vyhledávat zajímavé segmenty případů. Rozhodovací stromy, na rozdíl např. od neuronových sítí, reprezentují sekvence pravidel. Sekvence pravidel jsou pro člověka snadno pochopitelné a je možné je přímo převést např. do databázového jazyka SQL. Tímto způsobem je možné efektivně a rychle určit kategorii, do které daný záznam patří.

Dalším důvodem pro použití rozhodovacích stromů je jejich adaptabilita. Díky adaptabilitě nejsou rozhodovací stromy závislé na konkrétním problému. Pokud je získána nová informace, lze rozhodovací strom poměrně snadno modifikovat. Rozhodovací strom je struktura, která se formuje na základě tzv. trénovacích dat. Jedná se tedy o učení s učitelem.

1.1 Cíle práce

Cílem práce je nastudování a experimentální ověření vybraných modelů rozhodovacích stromů. V práci se budeme zabývat různými metodami budování stromu a jeho ořezáváním.

Součástí práce bude experimentální ověření modelů a také porovnání jednotlivých modelů s ohledem na různé míry nečistoty uzlů, ukončovací kritéria

ria a s ohledem na daný typ ořezávání. Jednotlivé modely budou otestovány na některých volně dostupných databázích vzorů.

Rozhodovací stromy mají za sebou poměrně dlouhý vývoj a bylo navrženo mnoho různých modelů rozhodovacích stromů. V této práci se nejprve zaměříme na rozhodovací strom CART, jehož autory jsou Breiman a kol. [2], a na Quinlanův rozhodovací strom C4.5 [31]. S jednotlivými modely úzce souvisí také metody ořezávání. V případě stromů CART popíšeme Error Cost-Complexity Pruning, které je součástí Breimanova výzkumu a také ořezávání Minimal Error Pruning [5] používající k odhadu chyby pro ořezávání tzv. m-odhad. U stromů C4.5 se zaměříme na ořezávací metody Reduced Error Pruning a Pessimistic Error Pruning [30]. Kromě základních stromů C4.5 a CART se práce zabývá tzv. SDT stromy, které spadají do oblasti fuzzy rozhodovacích stromů [28]. V SDT stromech je zavedena tzv. *přechodová oblast*. Díky ní mohou být vzory propagovány více větvemi najednou.

1.2 Obsah práce

Pro lepší orientaci v práci jsme zavedli jednotné konvence pro psaní textu. Důležité pojmy, které se v práci objeví poprvé a nejsou součásti definice, jsou psané *kurzívou*. Definice, věty a tvrzení jsou číslovány podle kapitol a podkapitol. Text uvnitř definic, vět a tvrzení je psán *kurzívou*. Obrázky, algoritmy, tabulky a důležité vzorce jsou číslovány s ohledem na kapitoly. Citace jsou vkládány přímo do textu formou čísla v hranaté závorce. Na konci dokumentu je potom uveden číslovaný seznam použité literatury setříděný podle příjmení prvního z autorů.

Nyní popíšeme strukturu práce pro snažší orientaci. Text je členěn na pět částí.

Obecné pojmy

Kapitola 2 obsahuje uvedení do problematiky. Budou zde zavedeny obecné pojmy a značení, které budeme v práci používat.

Základní modely rozhodovacích stromů

Třetí kapitola je věnována základním modelům stromů a jejich budování. V kapitole 3.1 nastíníme proces růstu stromu a uvedeme nejjednodušší algoritmus budování rozhodovacího stromu. Růst stromu je založen na dělení uzlu a výpočtu nečistoty uzlu. Dělení uzlu, výpočet nečistoty a výběr nejlepšího dělení je popsán v kapitole 3.2. Velikost stromu ve fázi růstu je závislá na ukončovací podmínce (viz kapitola 3.3). Podstatná část třetí kapitoly je věnována také ořezávacím algoritmům. Ořezávací algoritmy a jejich vlast-

nosti jsou tématem kapitoly 3.4. Nejrozšířenějšími rozhodovacími stromy jsou CART a C4.5 a jim budou věnovány kapitoly 3.5 a 3.6).

SDT stromy

Kapitola 4 podrobně popisuje SDT stromy, které využívají teorii tzv. fuzzy množin. Budování a ořezávání jsou fáze, ve kterých je určena struktura stromu (viz kapitoly 4.1 - 4.4). Pro zpřesnění klasifikace SDT stromu se ladí hodnoty parametrů stromu. O ladění parametrů SDT stromu pojednává kapitola 4.5.

Experimenty

Pátá, experimentální kapitola, zkoumá a porovnává algoritmy popsané v předchozích kapitolách na různých datových množinách. Součástí kapitoly jsou vlastní závěry a pozorování, která jsou získána experimentálně. V kapitole jsou uvedena také pozorování, vlastní tvrzení a jejich důkazy.

Závěr

V závěrečné kapitole budou shrnuty získané výsledky a nastíněny možnosti dalšího vývoje.

Kapitola 2

Obecné pojmy

V této kapitole uvedeme pojmy, které se budou objevovat v dalších částech práce. Zadefinujeme základní model rozhodovacího stromu, k jehož popisu využijeme teorii grafů [22, 34].

2.1 Základní pojmy

Problematika rozhodovacích stromů spadá do oblasti rozpoznávání dat. Typicky máme k dispozici tzv. *příznakové vektory*, které popisují určitý objekt. Objektem může být cokoliv, co lze změřit. Jedna složka příznakového vektoru je konkrétní naměřená hodnota. Konkrétní příznak nazýváme též *atributem*. Hodnota atributu může být *numerického* charakteru nebo také *nominálního* charakteru. Numerický atribut je spojitá hodnota z oboru reálných čísel $\mathbb{R}$. Nominální atribut nabývá diskrétních hodnot z nějakého výčtu. Všechny hodnoty nominálního atributu jsou předem známé, jsou neuspořádané a je jich vždy konečný počet. Často je také označován jako kategorický atribut. Na základě příznakového vektoru je pak daný objekt klasifikován. Je mu přiřazena konkrétní třída z diskrétního prostoru tříd nebo nějaká reálná hodnota ze spojitého prostoru reálných čísel $\mathbb{R}$. Třída je opět atribut.

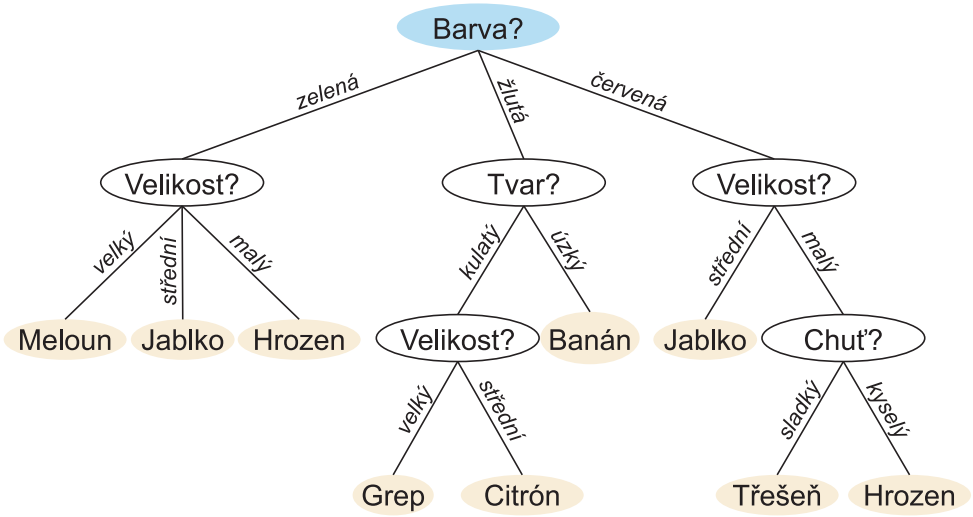
Předpokládejme například příznakové vektory, které popisují ovoce [7]. Uvažujme atributy barva, tvar, chuť a velikost. Příznakový vektor pro konkrétní kus ovoce může vypadat takto: {červená, oválný, sladký, střední}. Tomuto vektoru je přiřazena třída jablko.

V dalším textu zavedeme pojem příznakového vektoru formálně.

Definice 2.1.1 *Příznakovým vektorem se rozumí vektor $o \in O$, kde O je příznakový prostor.*

Příznakový vektor budeme také někdy nazývat *vzor* nebo *instance*.

Definice 2.1.2 *Třídou c budeme označovat prvek z množiny tříd C, kde C je podmnožinou $\mathbb{R}$ nebo C je konečná neuspořádaná množina.*



Obrázek 2.1: Rozhodovací strom. Kořenový uzel je označen modrou barvou a listové uzly oranžovou.

Definice 2.1.3 *Nechť O je příznakový prostor a C je množina tříd. Trénovací množina vzorů T je množina uspořádaných dvojic $T \subset O \times C$. X bude symbolizovat množinu všech vzorů $O \times C$.*

Definice 2.1.4 *Mějme vzor $x = (o, c) \in X$, kde $o \in O$ a $c \in C$. Pak $\pi_2(x) = c$ je projekce vzoru x do třídy c.*

Intuitivně $\pi_2(X) \subseteq C$ vrátí vektor různých tříd obsažených ve všech vzorech množiny X. Na trénovací množinu vzorů je možné nahlížet jako na matici příznaků, jejíž řádky jsou tvořeny vzory a konkrétní sloupec matice je tvořen vektorem hodnot daného příznaku všech vzorů.

Definice 2.1.5 *Nechť máme matici příznaků T, kde T je rovněž trénovací množina vzorů. Atributem budeme označovat index sloupce této matice. Poslední sloupec odpovídá třídám jednotlivých vzorů, jehož index nazveme třídovým atributem a_c. Hodnotou atributu trénovacího vzoru $t = (x_1, x_2, \ldots, x_n, c) = (t_1, t_2, \ldots, t_{n+1})$, kde $x = (x_1, x_2, ..., x_n) \in X$ a $c \in C$, budeme označovat zobrazení $a_i(t) = t_i$.*

2.2 Rozhodovací strom

Rozhodovací strom (viz obr. 2.1) je mnohoúrovňová struktura. Každá úroveň obsahuje uzly, které jsou spojeny hranou právě s jedním uzlem - *předchůdcem* v předchozí úrovni a jsou spojeny hranou aspoň se dvěma uzly - *potomky* v následující úrovni. První úroveň obsahuje právě jeden uzel tzv. *kořen*. Kořen je spojen hranou se svými potomky, které jsou opět kořeny *podstromů*.

Podstromy mohou mít různý počet úrovní. Poslední úroveň podstromů obsahuje uzly - *listy*, které nemají žádného potomka. Uzly, které nejsou list ani kořen, jsou *vnitřní* nebo také *testovací* uzly. Pomocí dalších definic zavedeme pojem rozhodovací strom.

Definice 2.2.1 *Nechť $G = (V, E)$ je graf. V označuje vrcholy nebo též uzly a E hrany. Nechť v je uzel z V, pak **vějíř** $F_{in}(v)$ je definován:*

$$F_{in}(v) = \{u \in V \mid (u, v) \in E\} \tag{2.1}$$

*Dále definujme **rozvětvení** $F_{out}(v)$ uzlu v:*

$$F_{out}(v) = \{u \in V \mid (v, u) \in E\} \tag{2.2}$$

F_{in} označuje všechny vrcholy vcházející do vrcholu v, které jsou s ním spojeny hranou, a $F_{out}(v)$ je tvořen vrcholy, které jsou spojeny hranami vycházejícími z uzlu v.

Definice 2.2.2 *Nechť $G = (V, E)$ je graf. Graf G je strom pokud:*

1. *Do každého uzlu vchází nejvýše jedna hrana, tj. $|F_{in}(v)| \leq 1$ pro $\forall v \in V$.*
2. *Existuje právě jeden uzel r, do kterého nevchází žádná hrana, tj. $|F_{in}(r)| = 0$ $\exists! r \in V$. Tento uzel se nazývá **kořenový uzel**.*
3. *G je souvislý graf.*

Více o souvislosti grafu lze nalézt např. v [22]. Můžeme si všimnout, že strom na Obr. 2.1 odpovídá zavedené definici. Kořenový uzel je označen modrou barvou, nevchází do něj žádná hrana a ve stromě již není žádný jiný uzel s touto vlastností. Do všech ostatních vchází právě jedna hrana z jeho předchůdce. Z libovolného uzlu se do všech ostatních uzlů dostaneme cestou, která je tvořena posloupností hran a uzlů a tedy je splněna i třetí podmínka definice.

Definice 2.2.3 *Nechť $G = (V, E)$ je strom a r je jeho kořenový uzel. Množina všech listů (listových uzlů) stromu G je definována takto:*

$$L(G) \stackrel{def}{=} \{v \in V \mid |F_{out}(v)| = 0\} \tag{2.3}$$

Uzel $v \in V$ je vnitřní nebo taktéž testovací uzel, pokud $v \neq r$ a $v \notin L(G)$.

Definice 2.2.4 *Nechť $G = (V, E)$ je strom. Podstromem $G|_v$ uzlu v rozumíme podgraf stromu G, pro který platí:*

1. *Obsahuje v.*

2. *Každý uzel v $G|_v$ obsahuje také celé rozvětvení, které bylo v původním G:*

$$F_{out}(u) \subseteq G|_v, \quad \forall u \in V(G|_v). \tag{2.4}$$

$G|^v$ označuje strom G bez podstromu $G|_v$ s uzlem v. Nechť $V|_v$ označuje všechny vrcholy podstromu $G|_v$. Potom $G|^v$ obsahuje uzly:

$$V \setminus V|_v \cup \{v\}. \tag{2.5}$$

$G|^v$ tedy obsahuje všechny hrany a uzly původního stromu G, ale neobsahuje již hrany a uzly podstromu $G|_v$ kromě vrcholu v. Strom je hierarchická struktura, ve které můžeme zavést „příbuznost“ jednotlivých vrcholů:

Definice 2.2.5 *Mějme strom $G = (V, E)$ a dva vrcholy $u, v \in V(G)$.*

- *u je potomkem vrcholu v, právě když $u \in V(G|_v)$.*
- *u je následníkem vrcholu v, právě když $(v, u) \in E(G)$.*
- *u je předchůdce vrcholu v, právě když u leží na cestě mezi kořenem a vrcholem v.*
- *u je rodičem vrcholu v, právě když $(u, v) \in E(G)$.*

Následník je taktéž potomek a předchůdce je také rodič. Dále zadefinujeme rozhodovací strom:

Definice 2.2.6 *Rozhodovací strom pro příznakový prostor O a klasifikační třídy C je trojice (G, δ, κ), kde*

- *$G = (V, E)$ je strom.*
- *Zobrazení $\delta : V \setminus L(G) \times O \to V$ každému nelistovému uzlu $v \in V$ a vzoru $o \in O$ přiřadí uzel $u \in V$ takový, že $(v, u) \in E$. Zobrazení $\delta(v, o)$ nazveme též rozhodnutím pro uzel v a vzor o.*
- *Zobrazení $\kappa : L(G) \to C$ přiřazuje každému listovému uzlu třídu.*

Mějme k dispozici příznakový vektor. *Klasifikace* je proces, který na základě daných příznaků přiřadí vektoru třídu. Klasifikaci provádí klasifikátor, v tomto případě rozhodovací strom. V každém testovacím uzlu (viz definice 2.2.3) se ohodnotí podmínka. Nejčastěji se porovnává konkrétní hodnota atributu vzoru s nějakou konstantou. Na základě tohoto ohodnocení je vybrána hrana vedoucí k některému z potomků uzlu.

Proces klasifikace začíná v kořenovém uzlu. Vzor je postupně propagován až do listového uzlu. Výsledkem je odhadnutá hodnota třídového atributu, která je tomuto listu přiřazena.

Definice 2.2.7 *Nechť máme rozhodovací strom* (G, δ, κ) *a* $o \in O$. *Dále mějme uzly* $v_1, v_2, \ldots, v_n$ *takové, že* v_1 *je kořen* G, $v_2, \ldots, v_{n-1}$ *jsou vnitřní uzly stromu* G *a* $v_n \in L(G)$. *Navíc pro uzly platí* $\delta(v_i, o) = v_{i+1}$, $i = 1, \ldots, n-1$. *Potom* $c = \kappa(v_n)$ *je klasifikací příznakového vektoru* o *nebo také odhad rozhodovacího stromu* (G, δ, κ) *pro vektor* o.

V předchozím textu jsme uvažovali uzly, které mohly mít dva a více potomků. V dalším textu budeme někdy používat i jednodušší případ, kdy uzly mají nejvýše dva potomky.

Definice 2.2.8 *Binární rozhodovací strom je takový rozhodovací strom* (G, δ, κ), *pro který:* $|F_{out}(v)| = 2$ *nebo* $|F_{out}(v)| = 0$ *pro libovolný uzel* $v \in V(G)$.

Každý obecný rozhodovací strom je možné převést na ekvivalentní rozhodovací binární strom [7].

Kapitola 3

Základní modely rozhodovacích stromů

V této kapitole se budeme zabývat fází budování základních modelů rozhodovacích stromů. V první části uvedeme nejjednodušší algoritmus ID3 [36] a na něm zdůrazníme problémy, které budeme dále rozebírat v dalších částech. Se stromy souvisí také problematika ořezávání, kterou podrobně popíšeme v části 3.4. Na závěr kapitoly popíšeme dva nejznámější algoritmy pro budování stromů. Prvním z nich je CART [2] a druhý je Quinlanův algoritmus C4.5 [31].

3.1 Vytvoření stromu

Každý algoritmus pro vytvoření rozhodovacího stromu dostane jako vstup trénovací množinu vzorů $T \subseteq X$ (viz definice 2.1.3). Na základě této množiny je vybudován rozhodovací strom [7]. Na začátku je kořen stromu, kterému odpovídá celá trénovací množina T. Trénovací množina se postupně dělí na menší v jednodušším případě disjunktní podmnožiny, které odpovídají následníkům kořenu (uzlům, které jsou spojeny s kořenovým uzlem hranou). Nově vzniklé podmnožiny se dále dělí a tím vznikají nové uzly odpovídající těmto podmnožinám. Proces dělení probíhá rekurzivně do té doby, až jsou nově vzniklé podmnožiny dostatečně „čisté“. Čím více vzorů množiny odpovídá jedné třídě, tím je množina čistší. Pojem čistoty, resp. nečistoty zavedeme formálně později v kapitole 3.2.1.

Definice 3.1.1 *Nechť $h_1, h_2, \ldots, h_n$ jsou všechny možné hodnoty atributu a_i a nechť $T \subseteq X$ je trénovací množina vzorů. Pak $T(a_i, h_j)$ označuje podmnožinu trénovacích vzorů, pro které platí:*

$$T(a_i, h_j) = \{t \mid t \in T, a_i(t) = h_j\}.$$

$T(a_i, h_j)$ jsou vzory, jejichž hodnota atributu a_i je rovna hodnotě h_j. Speciálním případem jsou trénovací vzory $T(a_c, c_j)$, jejichž hodnota třídového atributu a_c je rovna třídě c_j.

Definice 3.1.2 *Nechť $c_1, c_2, \ldots, c_n \in C$ jsou třídy a $t_1, t_2, \ldots, t_m \in T_v$, kde T_v je množina trénovacích vzorů, které během tvoření stromu odpovídají uzlu v. Nechť a_c označuje třídový atribut. Pak $c_{\text{maj}}(v) = \underset{c_i \in C}{\text{argmax}} \{|T_v(a_c, c_i)|\}$ je majoritní nebo také nejpočetnější třída vzorů T_v.*

Zjednodušeně platí, čím větší počet trénovacích vzorů $t \in T_v$ má hodnotu $a_c(t)$ rovno majoritní třídě, tím je uzel v čistší[1].

Algoritmus ID3 (Iterative dichotomizer version 3) [36] je základním algoritmus pro vytváření rozhodovacích stromů. Je popsán formou rekurzivní funkce (viz Algoritmus 3.1). Funkce dostane na vstupu „nepoužité“ nominální atributy A, které neobsahují třídový atribut a_c, a množinu trénovacích vzorů T. Je nutno poznamenat, že algoritmus ID3 předpokládá, že všechny atributy jsou nominální. Na začátku je na základě atributů A a trénovací množiny T rozhodnuto, zda bude strom dál růst nebo ne. Pokud není splněno kritérium pro ukončení růstu stromu, algoritmus hledá optimální atribut z množiny nepoužitých atributů A takový, podle kterého je množina vzorů T rozdělena na co nejčistší podmnožiny. Dělení T podle atributu $a_i \in A$ probíhá tak, že se množina vzorů T rozdělí na podmnožiny $T_j = T(a_i, h_j)$, kde h_j jsou všechny možné hodnoty atributu a_i. Potom, co je nalezen optimální atribut a, funkce vytvoří nový podstrom $G|_v$ rekurzivním voláním sebe sama. Funkce je postupně volána na všech podmnožinách $T_j = T(a, h_j)$, kde h_j jsou všechny možné hodnoty atributu a a na množině atributů $A \setminus a$. Funkce vrací odkaz na kořenový uzel nově vybudovaného stromu.

ID3 [36] používá tzv. *Greedyho prohledávání*, které spočívá v tom, že se hledá vždy mezi množinou atributů, které se ještě neúčastnily dělení v některém z předků uzlu. Dělit opakovaně podle stejného nominální atributu a nemá smysl, protože všechny vzory v jednotlivých podmnožinách odpovídajících dělení podle atributu a mají hodnotu tohoto atributu stejnou. Pokud již neexistuje atribut, podle kterého se ještě nedělilo, algoritmus končí. Můžeme si všimnout, že hloubka stromu je díky tomu limitována počtem všech atributů bez třídového atributu a_c.

[1] Čistotou uzlu v rozumíme čistotu podmnožiny trénovacích vzorů T_v.

Inicializace:
$V(G) \leftarrow \emptyset, E(G) \leftarrow \emptyset, L(G) \leftarrow \emptyset$

function ID3(atributy A, trénovací množina vzorů T)
  if $T = \emptyset$ **then return**
  end if
  $V(G) \leftarrow V(G) \cup \{v\}$, kde v je nově vytvořený uzel.
  if $\pi_2(t_1) = \pi_2(t_2) = \ldots = \pi_2(t_m)$ **then**
    $L(G) \leftarrow L(G) \cup \{v\}$
    $\kappa(v) \leftarrow \pi_2(t_1)$
    return v
  end if
  if $A = \emptyset$ **then**
    $L(G) \leftarrow L(G) \cup \{v\}$
    $\kappa(v) \leftarrow c_{\text{maj}}(v)$
    return v
  end if
  Nechť $a \in A$ nejlépe rozdělí T a $h_1, h_2, \ldots, h_n$ jsou hodnoty atributu a.
  for $i \leftarrow 1, \ldots, n$ **do**
    $T_i \leftarrow \{t \mid t \in T, a(t) = h_i\}$
    $u \leftarrow$ ID3$(A \setminus \{a\}, T_i)$ ▷ Vytvoření podstromu u.
    $E(G) = E(G) \cup (v, u)$ ▷ Napojení u na rodiče v.
    $\delta(v, o) \stackrel{def}{=} u, a_i(o) = h_i, \forall o \in O$
  end for
  return v
end function

Algoritmus 3.1: Algoritmus ID3 [36]. ID3 vyžaduje nominální příznaky. ID3 vytvoří rozhodovací strom (G, δ, κ) a vrátí odkaz na jeho kořenový uzel.

Jedním z cílů algoritmů pro vytvoření rozhodovacích stromů je vybudovat co nejmenší strom, který rozdělí trénovací množinu vzorů na co nejčistší podmnožiny. Velikost rozhodovacího stromu nazýváme *složitost* a definujeme ji takto:

Definice 3.1.3 *Složitostí rozhodovacího stromu (G, δ, κ) rozumíme počet vrcholů stromu G.*

Pro složitost rozhodovacího stromu budeme používat označení $|V(G)|$. Někdy se složitostí rozumí počet listů $|L(G)|$ stromu G. V dalším textu budeme používat oba významy složitosti tak, aby bylo patrné, o kterou složitost se jedná. V případě ID3 stromů je složitost omezena počtem atributů vzorů.

V jiných typech algoritmů je možně dělit vícekrát podle stejného atributu. Tím značně roste složitost stromu a z toho důvodu je také potřeba více času k vytvoření stromu. Čas potřebný k vybudování rozhodovacího stromu často bývá omezujícím faktorem. Výhodou rozhodovacích stromů je, že není třeba vybudovat kompletní strom, aby byl schopen klasifikovat. Složitost stromu a čas nutný k vytvoření stromu je možné snížit pomocí *ukončovací podmínky*, která předčasně zastaví další růst stromu. Více o ukončovací podmínce pojednává kapitola 3.3.

Někdy nezáleží na době vytváření stromu, ale pouze na tom, aby byl strom co nejmenší a dostatečně přesný. V tomto případě se strom se nechá vyrůst do maximální velikosti a pak je *ořezán*. Ořezávání má oproti předčasném ukončení růstu stromu pomocí ukončovací podmínky výhody, které zmíníme v kapitole 3.4.

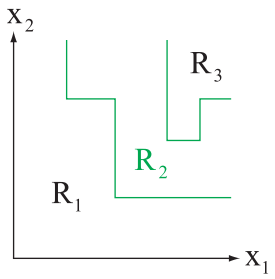
3.2 Dělení uzlu

Při vytváření stromů se snažíme najít takové zobrazení δ, které vede k co nejjednoduššímu stromu G s malým počtem co nejčistších uzlů. Zobrazení δ je nejčastěji založeno na ohodnocení podmínek, podle kterých je danému uzlu a vzoru přiřazen jeho následník. Uvažujme atributy $A = (a_1, a_2, \ldots, a_n)$ bez třídového atributu a vzor $o \in O$. Příkladem jedné takové podmínky může být splnění rovnosti $a_i(o) =$ 'červená'. V případě, že rovnost platí, $\delta(v, o)$ vrátí jednoho z následníků uzlu v, který odpovídá splnění rovnosti $a_i(o) =$ 'červená'. V případě, že tato podmínka není splněna, testuje se další podmínka. Zkouší se tak dlouho, až je některá z podmínek odpovídajících danému uzlu splněna. Přitom musí platit, že je splněna právě jedna podmínka. Situace je znázorněna na obr. 2.1. V kořenu stromu se ptáme na všechny možné hodnoty atributu 'barva'. Barva může být 'zelená', 'žlutá', 'červená'. Je-li barva daného vzoru červená, funkce δ přiřadí tomuto vzoru v kořenu třetí uzel vpravo s testem atributu 'velikost'.

V případě binárních stromů se testuje právě jedna podmínka. Pokud je podmínka splněna, pak $\delta(v, o)$ přiřadí pravého následníka; v opačném případě je přiřazen levý následník. Binární rozhodovací stromy rozdělí na základě podmínek příznakový prostor na oblasti, jejichž hranice jsou rovnoběžné se souřadnicovými osami příznakového prostoru (viz Obr. 3.1).

3.2.1 Nečistota uzlu

V předchozím textu jsme intuitivně pracovali s pojmem *čistota uzlu*. Uzel byl čistší, čím větší počet trénovacích vzorů odpovídající uzlu byl označen jednou konkrétní třídou. Spíše než čistota uzlu se používá pojem *nečistota uzlu*. V této kapitole jej přesně definujeme a probereme různé způsoby, jak



Obrázek 3.1: Binární rozhodovací strom rozdělí příznakový prostor na oblasti, jejíž hranice jsou rovnoběžné se souřadnicovými osami prostoru.

lze nečistotu spočítat.

Definice 3.2.1 *Nechť X_v je množina všech vzorů příznakového prostoru X, které se pomocí rozhodovacích podmínek dostanou od kořene stromu do uzlu v, a $C_v = \pi_2(X_v)$. Potom* $\mathrm{P}(c_i|v)$ *značí pravděpodobnost, že pro $x \in X_v$ je $\pi_2(x) = c_i, c_i \in C_v$.*

Tedy $\mathrm{P}(c_i|v)$ je pravděpodobnost, s jakou libovolný vzor z množiny všech vzorů X označený třídou c_i projde rozhodovacím stromem a přitom navštíví vrchol v.

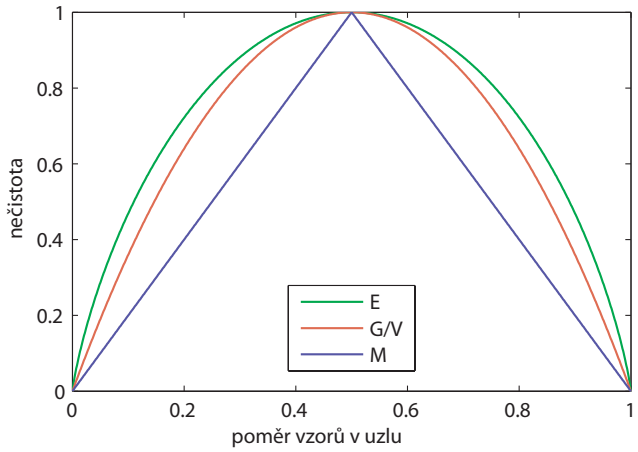
Definice 3.2.2 *Mějme libovolný uzel v a dále X_v nechť jsou vzory odpovídající uzlu v a $C_v = \pi_2(X_v)$. Potom nečistota $i(v)$ uzlu v splňuje tyto podmínky:*

- $i(v) = 0 \Leftrightarrow \exists c \in C_v$ *takové, že* $\pi_2(x) = c$ *pro* $\forall x \in X_v$. *Nebo-li* $i(v) = 0 \Leftrightarrow \exists c \in C_v$ *takové, že* $\mathrm{P}(c|v) = 1$.
- $i(v) = 1 \Leftrightarrow \mathrm{P}(c|v) = \frac{1}{|C_v|}$
- $i(v) \in \langle 0, 1 \rangle$.

Nechť $T_{\mathrm{maj}} = \{t \mid t \in T_v, \pi_2(t) = c_{\mathrm{maj}}(v)\}$. Intuitivně $i(v)$ bude menší, čím větší bude $\mathrm{P}\left(c_{\mathrm{maj}}(v)|v\right)$. V praxi ovšem nemáme k dispozici celou množinu vzorů X_v a tak se omezujeme pouze na trénovací množinu vzorů $T_v \subset X_v$ odpovídající uzlu v. Potom $\mathrm{P}(c|v)$ se odhaduje jako poměr $\frac{|T_v(c)|}{|T_v|}$, kde $T_v(c) = \{t \in T_v \mid \pi_2(t) = c\}$. Nebo-li $T_v(c)$ jsou takové vzory z T_v, které jsou označeny třídou c. Označme tento poměr $\hat{P}_v(c)$.

V dalším nechť $C_v = \{c_1, c_2, \ldots, c_n\}$. Nejčastěji se nečistota $i(v)$ vyjadřuje pomocí *entropie*:

$$i(v) = -\sum_{j=1}^{n} \hat{P}_v(c_j) \log_2 \hat{P}_v(c_j). \tag{3.1}$$



Obrázek 3.2: Srovnání nečistot uzlu pro dvě třídy. E - entropie, G/V - Gini/variance, M - nečistota chybné klasifikace.

Entropie uzlu obsahující vzory[2] odpovídající jedné třídě je nulová. Naopak entropie uzlu, kde jsou vzory rovnoměrně zastoupeny ve všech třídách $\pi_2(T_v)$, je 1.

Předpokládejme pro jednoduchost $C_v = \{c_1, c_2\}$. Entropie vyjadřuje počet bitů nutných k zakódování libovolného vzoru z T_v [18]. Je-li $\hat{P}_v(c_1) = 1$, pak je jisté, že $\pi_2(t) = c_1$ pro libovolné t a entropie je 0. Podobně je tomu pro $\hat{P}_v(c_2) = 1$. V případě $\hat{P}_v(c_1) = 0{,}5$ je nutný jeden bit k tomu, aby se rozhodlo, zda-li je $\pi_2(t) = c_1$ nebo $\pi_2(t) = c_2$. Je-li např. $\hat{P}_v(c_1) = 0{,}8$, pak každý vzor je možné zakódovat v průměru méně jak jedním bitem. Vzorům $\{t \mid \pi_2(t) = c_1\}$ se proto přiřadí kratší kód a méně pravděpodobným vzorům $\{t \mid \pi_2(t) = c_2\}$ se přiřadí kód delší.

Jiným vyjádřením nečistoty pro případ, kdy jsou k dispozici pouze dvě kategorie, je pomocí tzv. *variance*:

$$i(v) = \hat{P}_v(c_1)\hat{P}_v(c_2). \tag{3.2}$$

Zobecněním variance pro $|C_v| \geq 2$ je tzv. *Giniho nečistota*:

$$i(v) = \sum_{j \neq k \in C_v} \hat{P}_v(c_j)\hat{P}_v(c_k) = 1 - \sum_{l=1}^{n} \hat{P}_v^2(c_l). \tag{3.3}$$

Další možností je *nečistota chybné klasifikace* (dále MI), která značí minimální pravděpodobnost, se kterou bude trénovací vzor v uzlu v chybně klasifikován. Nečistota chybné klasifikace se definuje jako:

[2]Nepřesný zápis, který však budeme používat pro zkrácení zápisu. Uzel sám o sobě nemůže obsahovat vzory. Přesněji by mělo být zapsáno: „ Uzel, kterému odpovídá množina vzorů.“

$$i(v) = 1 - \max_{j=1,\dots,n} \hat{P}_v(c_j). \tag{3.4}$$

Lze ukázat, že MI má nespojitou první derivaci [7]. Tento fakt způsobí problém, pokud budeme hledat optimální rozhodnutí pomocí derivování.

Na obr. 3.2 je znázorněno srovnání výše uvedených nečistot uzlu pro dvě třídy. Giniho nečistota, resp. variance a MI byly vynásobeny konstantou 2 pro lepší srovnání a všechny funkce byly posunuty, abychom se vyhnuli problémům v nule. Tyto úpravy nemají vliv na budování stromu ani na klasifikaci. Všechny funkce mají vrchol v bodě 0,5, kdy četnosti vzorů z první a druhé třídy jsou stejné. Z obrázku si můžeme všimnout, že MI má v bodě 0,5 ostrý zlom, ve kterém není první derivace spojitá.

3.2.2 Nejlepší dělení

V případě binárních rozhodovacích stromů se na základě nečistoty pro uzel v vybírá atribut a a prahová hodnota s, které určují rozhodnutí δ. Budeme se snažit vybrat takovou rozhodovací podmínku, která nejvíce sníží nečistotu uzlu v.

Definice 3.2.3 *Mějme binární rozhodovací strom (G, δ, κ), uzel $v \in V(G)$ a uzly $v_L, v_R \in V(G)$ jsou následníci uzlu v, libovolný vzor $x \in X$, $a \in A \setminus \{a_c\}$, kde A je množina všech atributů a a_c je třídový atribut. Potom:*

- *$\delta(v, x) = v_L$, pokud je $a(x)$ numerická hodnota a je splněna podmínka $a(x) < s$. Jinak $\delta(v, x) = v_R$.*
- *$\delta(v, x) = v_L$, pokud je $a(x)$ nominální hodnota a je splněna podmínka $a(x) = s$. Jinak $\delta(v, x) = v_R$.*

Symbol s značí dělení.

Dělení s je prahová hodnota, se kterou porovnáváme hodnoty atributu vzorů a na základě porovnání vzor půjde do levého či pravého následníka uzlu v. Hodnota s je reálné číslo, pokud atribut a je numerický nebo s je nominální hodnota atributu a, pokud a je nominální.

Pro binární stromy je úbytek nečistoty pro rozhodnutí $\delta(v, x), x \in X$ definován jako

$$\Delta i(v) = i(v) - \mathrm{P}_L \cdot i(v_L) - (1 - \mathrm{P}_L) \cdot i(v_R), \tag{3.5}$$

kde v_L, resp. v_R je levý, resp. pravý potomek uzlu v. Symbolem P_L označíme poměr:

$$\mathrm{P}_L = \frac{|T_{v_L}|}{|T_v|}. \tag{3.6}$$

Během dělení uzlu hledáme takový atribut $a \in A \setminus \{a_c\}$ a dělení s, které maximalizují $\Delta i(v)$. Algoritmy pro budování stromů preferují funkce nečistoty, které se snadno a rychle vyčíslí, aby bylo vytváření stromu co nejrychlejší.

Uvažujme binární strom. Často se stává, že optimálních dělení s je více. Pro numerické atributy to může být dokonce nekonečně mnoho různých dělení s, protože mezi dvěma sousedními hodnotami numerických atributů dvou vzorů t_{i_1} a t_{i_2} leží nekonečně mnoho hodnot, podle kterých můžeme vzory rozdělit. V tom případě se vybere pouze jedno konkrétní dělení. Pokud není předem známo jakým způsobem jsou vzory rozmístěny v prostoru vzorů, vybírá se nejčastěji hodnota ležící uprostřed tohoto intervalu: $s = (a(t_{i_1}) + a(t_{i_2}))/2$ nebo vážený průměr: $s = (1 - \mathrm{P})a(t_{i_1}) - \mathrm{P}\, a(t_{i_2})$, kde P je poměr vzorů, které se dělením dostanou do jednoho z potomků.

Je nutno poznamenat, že úbytek nečistoty uzlu v (viz vztah 3.5) má pouze lokální charakter, protože jej odhadujeme jen na základě části trénovacích vzorů T_v, které odpovídají uzlu v. Nalezené dělení je tedy optimální pouze pro konkrétní uzel. Není také zajištěno, že po skončení trénovací fáze budeme mít vytvořený nejmenší strom.

Úbytek nečistoty podle vztahu 3.5 je možné aplikovat pouze v případě binárních stromů, kde $|F_{out}(v)| \leq 2$ [7]. Stromy obecně mohou mít více potomků. Pro stromy, které mají více potomků, je úbytek nečistoty vyjádřen takto:

$$\Delta i(v) = i(v) - \sum_{k=1}^{|F_{out}(v)|} \mathrm{P}_k\, i(v_k), \tag{3.7}$$

kde P_k označuje poměr trénovacích vzorů poslaných do potomka v_k. Přitom platí $\sum_{k=1}^{|F_{out}(v)|} \mathrm{P}_k = 1$. Bohužel takto definovaný úbytek nečistoty bude upřednostňovat uzly s velkým $|F_{out}(v)|$. Dělení s velkým rozvětvením $|F_{out}(v)|$ budou mít větší úbytek nečistoty než dělení s malým rozvětvením $|F_{out}(v)|$. Tomuto nepříznivému jevu lze zabránit tak, že úbytek nečistoty ve vztahu 3.7 normalizujeme:

$$\Delta i_N(v) = \frac{\Delta i(v)}{-\sum_{k=1}^{|F_{out}(v)|} \mathrm{P}_k \log_2 \mathrm{P}_k}. \tag{3.8}$$

Pro množiny vzorů, jejichž třídový atribut a_c nabývá více možných hodnot, se často používá tzv. *twoing kritérium* (dále TC). Základní myšlenkou TC je postupné rozdělování množiny vzorů do podskupin tříd, které mají společné charakteristiky [1, 2, 7]. Obecné charakteristiky se řeší v úrovních blízko kořene a charakteristiky typické pro menší skupinu vzorů se zpracovávají v uzlech ve větší hloubce. Předpokládejme, že hledáme optimální dělení pro uzel v. Nechť $C_v = \{c_1, c_2, \ldots, c_n\} = \pi_2(T_v)$ jsou všechny třídy, kterými jsou označeny vzory v uzlu v. Nechť $C_1 = \{c_{i_1}, c_{i_2}, \ldots, c_{i_m}\}$ a $C_2 = C_v \setminus C_1$ je rozdělení tříd C na dvě disjunktní podmnožiny C_1 a C_2. Označme třídy

vzorů

$$\pi_2(t) \stackrel{def}{=} c'_1, \forall t \in T_v, \pi_2(t) \in C_1$$

a podobně

$$\pi_2(t) \stackrel{def}{=} c'_2, \forall t \in T_v, \pi_2(t) \in C_2$$

Tímto způsobem přeznačíme třídy vzorů, jejichž třídy spadají do množiny C_1, třídou c'_1 a podobně přeznačíme třídy vzorů, jejichž třídy spadají do množiny C_2, třídou c'_2. Následuje hledání optimálního dělení s uzlu v, které maximalizuje $\Delta i(s, v, C_1)$. $\Delta i(s, v, C_1)$ symbolizuje úbytek nečistoty pro dělení s, uzel v, který je spočten vzhledem k třídě c'_1. Celý proces je iterativně prováděn pro všechna možná rozdělení tříd C_1 a C_2. Nakonec je vybráno takové dělení s, pro které je úbytek Δi maximální.

Na závěr je nutno poznamenat, že výběr funkce nečistoty zřídkakdy ovlivní přesnost klasifikátoru [7]. Vybírá se metoda, která má co nejmenší výpočetní nároky, nejčastěji entropie nebo Giniho nečistota. Klasifikátory se zpřesňují a zobecňují pomocí ořezávání a kritérií pro ukončení dělení uzlů.

3.3 Ukončovací podmínka

Během budování stromu je třeba rozhodnout, zda-li má dojít k dělení uzlu nebo dělení ukončit a uzel prohlásit za list. Není žádoucí vytvořit strom, jehož listy mají nejmenší nečistotu. Takový strom je obvykle přetrénovaný. Dává dobré výsledky na trénovacích vzorech, naopak neznámé (tj. vzory, které nebyly v trénovací množině) klasifikuje často chybně. V extrémním případě může každý list odpovídat jednomu trénovacímu vzoru. Naopak volbou příliš omezující ukončovací podmínky vybudujeme malý strom, který není ještě dostatečně přesný. Přesnost je míra správné klasifikace na neznámých vzorech. Čím je míra správné klasifikace vyšší, tím je vyšší přesnost klasifikátoru.

Jinou metodou pro ukončení dělení uzlu je volba prahové hodnoty pro pokles nečistoty uzlu. Dělení uzlu je ukončeno, pokud maximální pokles nečistoty uzlu s použitím nejlepšího dělení s^* je menší než daný práh β. Tedy dělení je ukončeno, pokud $\Delta i(s^*) \leq \beta$. Výhodou oproti předchozí metodě je to, že se pro trénování využije celá trénovací množina. Kritickým faktorem této metody je volba prahové hodnoty.

Dalším přístupem, jak předčasně ukončit růst stromu, je volba prahové hodnoty pro minimální počet trénovacích vzorů v listu. Pokud během budování stromu je počet vzorů v listu menší než tato prahová hodnota, dělení uzlu je ukončeno.

Cílem algoritmů pro vytvoření stromu je snaha vytvořit co nejčistší a nejmenší stromy. Čistotu stromu lze měřit tak, že jsou sečteny všechny nečistoty v listech. K nečistotě v listech se navíc přičte také složitost stromu:

$$\alpha \cdot |V(G)| + \sum_{v \in L(G)} i(v), \tag{3.9}$$

kde α je kladná konstanta, která určuje míru vlivu složitosti stromu na ukončení dělení. Čím větší zvolíme hodnotu α, tím bude mít složitost stromu větší vliv na výpočet hodnoty pro ukončení dělení. Např. zvolíme-li hodnotu α blízkou nule, bude ukončovací podmínka závislá pouze na nečistotě uzlů v listech.

3.4 Ořezávání

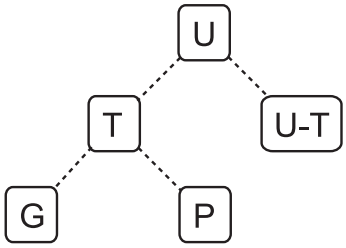
Doposud byla velikost stromu určena pomocí volby ukončovací podmínky. Avšak dobrá volba této podmínky je obtížná. Pokud zvolíme příliš omezující ukončovací podmínku, strom bude mít sice malou velikost, ale nebude dostatečně přesný. Naopak, bude-li zvolena podmínka, která nebude omezovat další dělení uzlů, dostaneme příliš velký strom. Takový strom je obvykle přetrénovaný.Pokud však chceme snížit časové nároky pro trénování, je předčasné ukončení růstu stromu žádoucí. Jiným řešením je nehledat správnou ukončovací podmínku, ale nechat vytvořit velký strom a ten pak ořezat.

Ořezávání stromu má několik výhod. Např., existují situace, kdy dělení uzlu podle jednotlivých atributů nepřinesou velká zlepšení. Pokud jsou však dělení uzlu zkombinována dohromady, dojde k významnému poklesu nečistoty. Nejprve se tedy vybuduje maximální strom, tj. strom o maximální možné velikosti, a pak se směrem od listu ke kořeni stromu ořezává. Maximální strom obsahuje i podstromy, které přispívají ke zlepšení klasifikátoru. Metody ořezávání se tyto podstromy snaží zachovat, naopak redundantní podstromy (podstromy, které nemají velký vliv na přesnost klasifikátoru) budou odstraněny.

Některé ořezávací algoritmy vyžadují pro svůj chod tzv. *ořezávací množinu vzorů*. Jsou to trénovací vzory, které jsou vybrány nezávisle na množině vzorů určených pro růst. Obecně máme k dispozici množinu vzorů, kde část je použita na trénování a doplněk na testování. V kapitole 3.3 jsme se zmínili o validační množině vzorů. Tyto pojmy mají mírně odlišný význam. Validační množina se používá ve fázi budování pro nalezení optimálního stromu, kdežto testovací množina slouží k odhadu přesnosti již vybudovaného stromu. *Trénovací vzory*[3] se dále dělí na *vzory určené pro růst* a na již zmíněné *ořezávací vzory*. V této kapitole budeme používat toto značení pro množiny vzorů:

- U je množina všech dostupných vzorů pro trénování i testování.

[3] V předchozích kapitolách jsme používali označení „trénovací vzory“ pro vzory určené pro růst stromu. V této kapitole budeme tyto dva pojmy rozlišovat.



Obrázek 3.3: Rozdělení množiny vzorů. U jsou všechny dostupné vzory, T je trénovací množina vzorů, $U \setminus T$ testovací vzory, G vzory určené pro růst a P ořezávací vzory.

- $T (T \subset U)$ je trénovací množina vzorů a $U \setminus T$ je testovací množina vzorů.
- $G (G \subseteq T)$ je množina vzorů určených pro růst a $P = T \setminus G$ jsou ořezávací vzory.

Zavedené značení je znázorněno na obr. 3.3.

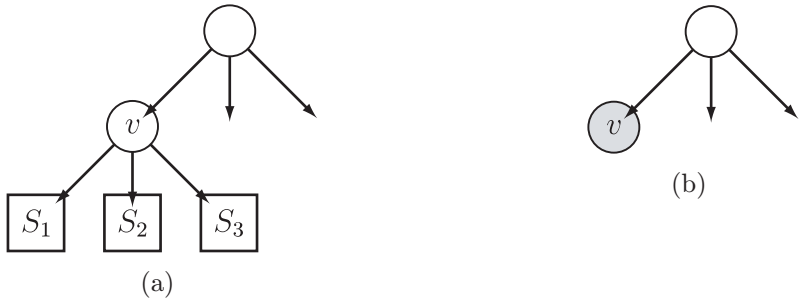
Ořezávání je možné chápat jako proces, který sníží velikost vytvořeného stromu. Nejdříve se najde část stromu, kterou je nutno odstranit, a pak je provedena jedna z následujících ořezávacích operací [37]:

Definice 3.4.1 *Mějme rozhodovací strom* (G, δ, κ).

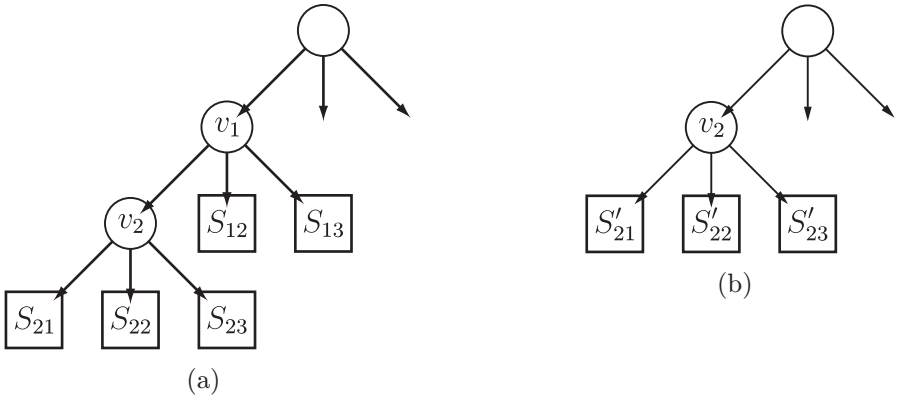
- *Operace nahrazení podstromu listem: Mějme uzel* $v \in V(G)$ *a nechť* $G|_v$ *je podstrom* G *s kořenovým uzlem* v. *Nahrazení podstromu listem proběhne tak, že ze stromu* G *odebereme všechny vrcholy* $u \in V(G|_v) \setminus \{v\}$ *a hrany* $e \in E(G|_v)$. *Dále rozšíříme množinu listů* $L(G) \leftarrow L(G) \cup \{v\}$ *a definujeme zobrazení* $\kappa(v) \leftarrow c_{\text{maj}}(P_v)$. *Kde* $c_{\text{maj}}(P_v)$ *značí majoritní třídu ořezávacích vzorů, které se klasifikací dostanou do uzlu* v.

- *Operace budování podstromu: Nechť* $v_2 \in V(G)$ *je jeden z potomků uzlu* $v_1 \in V(G)$, *nebo-li* $v_2 \in F_{out}(v)$ *a* $S_{11}, S_{12}, \ldots, S_{1m}$ *jsou podstromy, jejichž kořeny jsou potomci uzlu* v_1 $(\in F_{out}(v_1))$. *Označme podobně* $S_{21}, S_{22}, \ldots, S_{2m}$ *podstromy, jejichž kořeny jsou potomci uzlu* v_2 $(\in F_{out}(v_2))$. *Nechť podstromy* $S'_{21}, S'_{22}, \ldots, S'_{2m}$ *jsou nově vybudované podstromy, které nahradí* $S_{21}, S_{22}, \ldots, S_{2m}$. *Tyto podstromy byly vytvořeny na základě jim odpovídajících ořezávacích vzorů z množin* $P_1, P_2, \ldots, P_m$, *které vznikly z* P_{v_1} *rozdělením podle podmínky v uzlu* v_2. *Nový podstrom s kořenem* v_2 *a podstromy* $S'_{21}, S'_{22}, \ldots, S'_{2m}$ *nahradí celý podstrom s kořenem* v_1.

Budování podstromu vytváří nové podstromy $S'_{21}, S'_{22}, \ldots, S'_{2m}$, jak bylo popsáno v definici 3.4.1. Tyto podstromy jsou vybudovány z ořezávací vzorů,



Obrázek 3.4: Nahrazení podstromu listem. Podstrom s kořenem v bude nahrazen listem v.



Obrázek 3.5: Budování podstromu. Podstrom s kořenem v_2 je znova vybudován z ořezávacích vzorů, které obsahoval uzel v_1.

které se dostanou při klasifikaci do uzlu v_1 původního neořezaného stromu. V praxi se častěji setkáme s operací nahrazení podstromu listem (viz Obr. 3.4). Operace budování podstromu (3.5) je podstatně složitější a nemusí být jasné, jestli se vyplatí ji provádět. Zjevně bude náročnější na časovou složitost.

Dalším úkolem ořezávání je rozhodnutí, jestli jej nahradit listem (pro operaci nahrazení podstromu listem) nebo nahradit vnitřní uzel jedním z jeho potomků (pro operaci budování podstromu). K takovému rozhodnutí je nutné odhadnout očekávaný poměr chyby v daném uzlu na nezávisle vybrané množině ořezávacích vzorů. Pokud budeme mít takový odhad, bude pak snadné rozhodnout zda daný podstrom ponechat nebo na něm provést některou z ořezávacích operací. Toto rozhodnutí učiníme na základě porovnání poměru chyby původního podstromu s poměrem chyby jeho náhrady.

3.4.1 Reduced Error Pruning

Metoda REP se používá k ořezání rozhodovacího stromu C4.5 (viz kapitola 3.6). Jejím autorem je J. R. Quinlan [30]. K ořezávání používá nezávislou množinu ořezávacích vzorů na vzorech pro budování rozhodovacího stromu. Platí pravidlo, že čím je množina ořezávacích vzorů větší, tím lépe je strom ořezán. Při trénování vzniknou tzv. *nejisté části stromu* [16], které jsou vytvořeny z velmi malého počtu trénovacích vzorů. K tomu, aby tyto nejisté části stromu byly odhaleny, je nutné mít dostatečný počet ořezávacích vzorů, které se do uzlů v nejistých částech dostanou během klasifikace. Nevýhodou je, že toho se obvykle docílí tím, že se zvýší počet ořezávacích vzorů na úkor trénovacích vzorů.

Quinlan nikdy nepopsal REP algoritmicky. Díky tomu vznikla spousta různých interpretací REP. Z Quinlanova popisu totiž není jasné, zda algoritmus REP postupuje odspodu nahoru nebo iterativním způsobem. Po ořezávání vzniknou v rozhodovacím stromě nové listy, pro které musíme definovat zobrazení κ (označení listů) (viz definice 2.2.6. Z popisu REP není jasné, zda použít k definici nového zobrazení κ ořezávací nebo trénovací množinu dat.

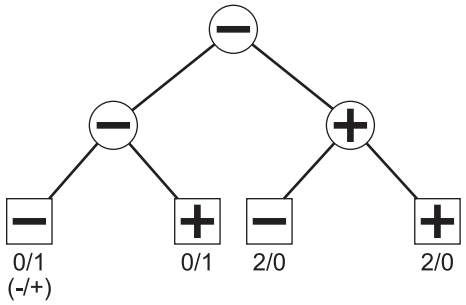
Díky víceznačnostem definice REP vznikly chybné interpretace Quinlanova algoritmu [23, 24]. Uzly se ořezávají iterativně tak, že se vybere uzel, jehož odstranění nejvýše zvýší přesnost rozhodovacího stromu na ořezávací množině. Pokud by další ořezání bylo pro strom z hlediska přesnosti škodlivé, proces je ukončen.

Nesprávnost tohoto algoritmu ukázali v [9, 10]. Strom vytvořený tímto algoritmem nesplňuje požadavek Quinlana, že strom ořezaný algoritmem REP je nejpřesnější vzhledem k ořezávací množině. Navíc tento algoritmus neprovádí explicitní kontrolu podstromu na snížení klasifikační chyby. Existují i jiné iterativní algoritmy, které počítají i s touto kontrolou, avšak jsou mnohem složitější než použití algoritmu odspodu nahoru.

V dalším textu budeme uvažovat průchod stromu odspodu nahoru. Uzel, který je ořezán, se dalšího ořezávání již neúčastní. Uzly jsou zpracovávány v postorder pořadí. Tím bude zajištěna jedna důležitá vlastnost: Odstraňovaný podstrom nemůže obsahovat jiný vnořený podstrom, který měl být odstraněn.

Označování listů

Označováním listů se rozumí určení zobrazení κ (viz definice 2.2.6). Dalším zdrojem chybných interpretací v Quinlanově popisu REP je nejasná specifikace označování listů, které vzniknou při ořezávání. V publikaci [26] se rozhodli označovat listy podle převládající třídy trénovacích vzorů. Avšak jimi analyzovaná verze algoritmu označovala tyto listy majoritní třídou ořezávacích vzorů. Autoři článku to vysvětlovali tím, že v praxi je velmi malý rozdíl, podle které množiny vzorů se budou výsledné listy označovat. Ve skutečnosti



Obrázek 3.6: Rozdíl v použití trénovacích a ořezávacích vzorů k označení listů během ořezávání metodou REP. Značky $(+,-)$ uvnitř uzlů symbolizují převládající třídu trénovacích vzorů, které se dostanou do těchto uzlů. x/y vyjadřuje, že x negativních a y pozitivních ořezávacích vzorů dosáhne listu.

označení ořezaných listů podle převládající třídy ořezávacích vzorů je jiné než označení podle původní trénovací množiny.

Rozdíl v použití trénovacích a ořezávacích vzorů k označení listů během ořezávání demonstrujeme na obrázku 3.6. V této kapitole budeme používat pojmy negativní a pozitivní vzory ve smyslu: Vzory $\{t \in P \mid \pi_2(t) = c_1\}$ jsou pozitivní vzory a vzory $\{t \in P \mid \pi_2(t) = c_2\}$ jsou negativní vzory.

(i): Nechť k označení listů po ořezání je použita trénovací množina. Vybudovaný strom klasifikoval 3 vzory špatně. Algoritmus REP nejdříve testuje situaci v kořenu levého podstromu. Levý podstrom klasifikoval 1 vzor špatně. Nahrazením kořene podstromu listem by se chyba zvýšila (2 vzory by byly klasifikovány chybně) a tedy nedojde k nahrazení podstromu listem. V pravém podstromě je situace podobná. Chyba (=počet chybně klasifikovaných vzorů) by se nahrazením podstromu listem opět zvýšila. Naposledy se testuje kořen celého stromu. Nahradíme-li celý strom listem, chyba se z původních 3 špatně klasifikovaných vzorů sníží na 2 špatně klasifikované vzory. Kořen je pomocí trénovacích vzorů označen třídou '−' a tedy 4 vzory budou klasifikovány správně a 2 špatně. Celý strom bude tedy nahrazen jediným listem označeným třídou '−'.

(ii): Nechť k označení listů po ořezání je použita ořezávací množina. Kořen levého podstromu bude nahrazen listem, protože chyba původního stromu je 1 a chyba budoucího listu je 0. Uzel totiž obsahuje dva vzory odpovídající třídě '+' a tedy budoucí list bude podle ořezávacích vzorů označen třídou '+'. Podobně bude nahrazen listem i pravý podstrom. Nový list bude taktéž klasifikovat všechny ořezávací vzory správně. Potom, co jsou provedeny ořezávací operace, má strom původní kořen a dva nové listy. Chyba nového stromu je 0. Nakonec je zkoumána si-

tuace v kořeni. Strom však nebude nahrazen listem, protože by se tím chyba zvětšila.

Z ukázkového příkladu je vidět, že ořezání metodou REP je jiné v případě použití trénovací množiny a jiné v případě použití ořezávací množiny k označení nových listů.

Později bude vysloveno tvrzení, že pokud uzel obsahuje netriviální podstromy, nemůže být ořezán [8]. Z toho plyne, že kandidáty na ořezání jsou pouze uzly, jejichž potomci jsou listy.

Další strategie používá k označení listů jak trénovací tak i ořezávací množinu vzorů. Obvykle počet trénovacích vzorů bývá mnohem vyšší než počet ořezávacích vzorů a díky tomu je těžké (ne-li nemožné) z ořezaného stromu rozeznat, jaká strategie byla použita pro označování listů. Je těžké rozeznat, zda-li k označování byla použita jen trénovací množina nebo trénovací a ořezávací množina dohromady.

Prázdné podstromy

REP používá různé množiny vzorů k vytváření a ořezávání rozhodovacích stromů. Díky tomu se pak může stát, že se do některých částí stromu ořezávací vzory nedostanou. Takové části rozhodovacího stromu mohou být (a skutečně budou) nahrazeny listem aniž by se změnil počet klasifikačních chyb, které vzniknou na ořezávacích vzorech. Jinak řečeno, stromy, které neobdrží žádný ořezávací vzor, budou vždy ořezány. Quinlan [30] poznamenal, že takové části původního stromu je možné odstranit. Je otázkou, zda by měly být podstromy, do kterých se ořezávací vzory nedostanou, ořezány nebo ponechány. Na tuto otázku není snadné odpovědět. Na jednu stranu tyto podstromy mají smysl pro trénovací množinu, která je obvykle početnější než ořezávací množina. Na druhou stranu fakt, že žádný vzor z ořezávací množiny neodpovídá těmto částem stromu, by mohl rozhodnout o tom, že tyto části stromů byly vytvořeny náhodnými příznaky trénovacích dat. Problém prázdných stromů je spojen s problémem *malých disjunkcí* v algoritmech strojového učení [13].

Algoritmus REP

Algoritmus REP probíhá ve dvou fázích [30]. V první fázi se propagují ořezávací vzory do celého stromu. Propagace probíhá tak, že každý vzor, podobně jako během klasifikace, projde od kořene směrem k listu. Přechodová funkce δ určuje, které uzly vzor na své cestě od kořene k listu „navštíví“. U každého uzlu jsou udržovány čítače, které odpovídají třídám vzorů. V případě, že je uzel navštíven ořezávacím vzorem, je čítač odpovídající třídě vzoru zvýšen o 1.

Pro jednoduchost předpokládejme, že máme binární strom a množina tříd C obsahuje právě dvě třídy c_1, c_2. Přitom se pro každý uzel v udržují dva čítače. První z nich $|P_v|$ udržuje počet všech ořezávacích vzorů P_v, které se dostanou do uzlu v. Nechť $P_v(c_1) = \{t \mid t \in P_v, \pi_2(t) = c_1\}$. Druhý čítač $|P_v(c_1)|$ označuje počet ořezávacích vzorů uzlu v, které náleží do třídy c_1. Na základě čítačů budou porovnávány chyby uzlů a podstromů. Toto porovnání je třeba k provedení operace nahrazení podstromu listem. Dále jsou čítače používány k označení nově vzniklých listů.

V druhé fázi algoritmu dojde k samotnému ořezávání stromu. Postupuje se od listů směrem ke kořenu stromu. Na základě hodnot čítačů se rozhoduje, zda provést ořezávací operace, či nikoliv. Ořezávací operace se provedou jen na takových uzlech v, ve kterých dojde ke snížení chyby. Platí přitom, že každý uzel je navštíven právě jednou. Nově vzniklé uzly jsou označeny majoritní třídou ořezávacích vzorů, které se do tohoto uzlu dostanou. Formální popis druhé fáze algoritmu REP je uveden v Algoritmu 3.2.

Vlastnosti REP

Pro jednoduchost opět uvažujme binární strom a $C = \{c_1, c_2\}$. Následující tvrzení je pravdivé pro libovolnou strategii označování listů po ořezávání.

Věta 3.4.2 *Použití algoritmu* REP *s množinou ořezávacích vzorů P na rozhodovací strom $\tau = (G, \delta, \kappa)$ vytvoří ořezaný strom $\tau' = (G', \delta, \kappa')$ stromu τ takový, že τ' je nejmenším ořezaným stromem τ, který má nejmenší chybu pro ořezávací množinu vzorů P [30].*

Věta je dokázána v [8].

Uvažujme situaci, kdy se algoritmus REP dostane do uzlu a provádí test, zda provést ořezávací operaci či nikoliv.

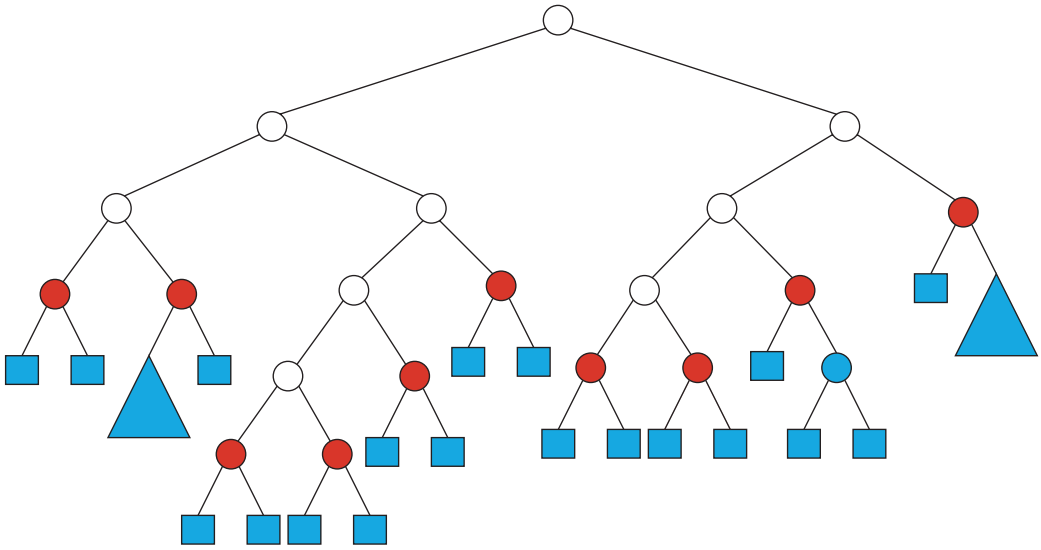
Věta 3.4.3 *Předpokládejme, že se algoritmus* REP *dostane do vnitřního uzlu v. Nechť žádný z následníků uzlu v není list. Uzel v nebude algoritmem* REP *ořezán, pokud obsahuje netriviální podstrom.*

Věta je dokázána v [8]. Díky této větě je možné ořezávání optimalizovat. Nechť levý podstrom τ_0 stromu τ obsahuje netriviální podstrom τ_0', který nebyl nahrazen listem během ořezávání. Uzly na cestě od kořene stromu τ ke kořeni podstromu τ_0' nebudou díky větě 3.4.3 nahrazeny listem.

Je-li následníkem uzlu v původní list (list vytvořený ve fázi růstu stromu), pak může být ořezán dokonce i když následník uzlu v je netriviální podstrom. Pokud následníci uzlu v jsou oba dva triviální podstromy, může být ořezán. Ořezání záleží na rozložení ořezávacích vzorů, které se dostanou do uzlu v a jeho potomků.

```
1: function REP(uzel v, strom (G, δ, κ))
2:     if v ∈ L(G) then
3:         if κ(v) = c_1 then
4:             return |P_v| − |P_v(c_1)|
5:         else
6:             return |P_v(c_1)|
7:         end if
8:     else
9:         Nechť v_L a v_R značí levého a pravého následníka uzlu v.
10:        chyba_v = REP(v_L, (G|_{v_L}, δ, κ)) + REP(v_R, (G|_{v_R}, δ, κ))
11:        if chyba_v < min{|P_v(c_1)|, |P_v| − |P_v(c_1)|} then
12:            return chyba_v
13:        else
14:            L(G) ← L(G) ∪ {v}
15:            Pro ∀u ∈ {V(G|_v) \ {v}} : V(G) ← V(G) \ {u}
16:            Pro ∀e ∈ E(G|_v) : E(G) ← E(G) \ {e}
17:            if |P_v(c_1)| > |P_v| − |P_v(c_1)| then
18:                κ(v) ← c_1
19:                return |P_v| − |P_v(c_1)|
20:            else
21:                κ(v) ← c_2
22:                return |P_v(c_1)|
23:            end if
24:        end if
25:    end if
26: end function
```

Algoritmus 3.2: Druhá fáze algoritmu REP (Reduced Error Pruning). Funkce REP vrací chybu stromu (počet chybně klasifikovaných ořezávacích vzorů).



Obrázek 3.7: Okraj, vnitřek a bezpečné uzly rozhodovacího stromu. Okraj stromu je označen červenou a modrou barvou, vnitřek stromu jsou bílé uzly a bezpečné uzly stromu jsou červené. Trojúhelníky reprezentují netriviální podstromy.

V analýze [26] bylo ukázáno, že nezbytným předpokladem pro ořezání uzlu v je, aby všichni jeho potomci v hloubce d byli ořezáni. d je hloubka po první list původního neořezaného podstromu kořenícího ve v. Uzel v nebude ořezán, pokud jeden nebo více jeho potomků v hloubce d budou zachovány. Pomocí rekurzivního použití věty 3.4.3 je toto tvrzení dokázáno.

Důsledek 3.4.4 *Strom τ kořenící v uzlu v bude zachován, pokud jeden nebo více potomků uzlu v v hloubce d nejsou ořezány.*

Abychom se nemuseli zabývat řešením hledání nejbližšího kořene podstromu daného listu, uvažujeme množinu listů patřících kořeni na všech větvích stromu.

Definice 3.4.5 *Mějme neořezaný rozhodovací strom $\tau = (G, \delta, \kappa)$.*

- *Okrajem $F_f(G)$ (angl. fringe) jsou všechny uzly $A = \{v \in V(G) \mid \exists u \in L(G) : (v, u) \in E(G)\}$ a také $\{v \in V(G) \mid v \in V(G|_u), u \in A\}$.*
- *Vnitřek (angl. interior) $F_i(G)$ jsou všechny uzly, které nejsou v okraji, tedy $F_i(G) = \{v \in V(G) \mid v \notin F_f(G)\}$.*
- *Bezpečné uzly (angl. safe nodes) jsou uzly $F_s(G) = \{v \in F_f(G) \mid \exists u \in F_i(G) : (u, v) \in E(G)\}$.*

Na obr. 3.7 jsou názorně zobrazeny jednotlivé typy uzlů v rozhodovacím stromě popsaných v definici. Na cestě od kořene k bezpečnému uzlu nejsou žádné listy, protože všechny uzly mezi kořenem a bezpečnými uzly leží ve vnitřku a samotné bezpečné uzly nemohou být listy. Pokud tedy ořezávací proces dospěje k bezpečnému uzlu, můžeme na odpovídající větev aplikovat větu 3.4.3.

Předpokládejme, že celý rozhodovací strom bude ořezán do jediného uzlu. Z bezpečných uzlů se v průběhu ořezávání taktéž stanou listy i když ne nutně ve stejnou chvíli. Nechť proces ořezávání se dostane do bezpečného uzlu. U bezpečných uzlů se provádí rozhodnutí o ořezání těchto uzlů pouze v případě, že všechny uzly na cestě mezi bezpečným uzlem a kořenem mají stejnou majoritní třídu. Ořezávání stromu je charakterizováno následovně.

Nechť τ je strom, který má být ořezán, a P množina ořezávacích vzorů, kde $|P| = n$. Bez úhony na obecnosti předpokládejme, že nejméně polovina ořezávacích vzorů je odpovídá třídě c_1. Nechť $p = \frac{|\{t \in P | \pi_2(t) = c_1\}|}{|P|}$ a platí $p \geq 0{,}5$. V případě, že τ by byl nahrazen majoritním listem, byl by označen třídou c_1. Na základě těchto předpokladů platí následující věta:

Věta 3.4.6 ([8]) *Strom τ bude ořezán do jediného listu právě když*

- *všechny stromy kořenící v bezpečných uzlech stromu τ budou ořezány a*
- *nejméně tolik pozitivních jako negativních ořezávacích vzorů se dostane do každého bezpečného uzlu v τ.*

Věta je dokázána v [8].

3.4.2 Minimal Error Pruning

Autory algoritmu Minimal Error Pruning (MEP) jsou Niblett a Bratko [25]. Nejdříve byl MEP založen na *Laplaceově odhadu pravděpodobnosti*. Později byl Laplaceův odhad pravděpodobnosti nahrazen *m-odhadem*, jehož autorem je Cestnik [5], který odstranil některé nevýhody Laplaceova odhadu. Výhodou MEP proti algoritmu REP je, že chybu odhaduje přímo ze vzorů určených pro růst a tudíž nevyžaduje speciální ořezávací množinu vzorů.

Hlavním principem algoritmu MEP je opět ořezávání od listů ke kořeni. Ořezává se do té doby, dokud není klasifikační chyba minimální. V daném uzlu je třeba porovnat chybu uzlu v s chybou podstromu $\tau|_v = (G|_v, \delta, \kappa)$.

Definice 3.4.7 *Statická chyba E_s v uzlu v je pravděpodobnost, že vzor $x \in X_v$, který se dostane do uzlu v, nebude patřit do majoritní třídy $c_{\text{maj}}(v)$:*

$$E_s(v) = \mathrm{P}(c \neq c_{\text{maj}}(v) \mid \pi_2(x) = c, \forall x \in X_v). \tag{3.10}$$

Definice 3.4.8 *Mějme rozhodovací strom $\tau = (G, \delta, \kappa)$. A nechť $\tau|_v$ je jeho podstrom. Zpětnou chybu*[4] *E_b podstromu $\tau|_v$ definujeme takto:*

- *Pokud $\tau|_v$ je tvořen pouze jedním uzlem v, který vznikne buď ořezáním τ nebo v je list, pak*
$$E_b(\tau|_v) = E_s(v), \tag{3.11}$$
kde $E_s(v)$ je statická chyba.

- *Pokud je τ_v netriviální, pak*
$$E_b(\tau|_v) = p_1 E_b(\tau_1) + p_2 E_b(\tau_2) + \ldots, \tag{3.12}$$
kde τ_i jsou podstromy s kořeny potomků uzlu v a p_i jsou pravděpodobnosti, se kterou vzor spadne do dané větve.

Pravděpodobnosti p_i $(i > 0)$ v definici 3.4.8 mohou být odhadnuty pomocí poměru trénovacích vzorů, které se dostanou do daného podstromu τ_i. Uzel v bude ořezán a prohlášen za list, právě když

$$E_b(\tau|_v) > E_S(v). \tag{3.13}$$

Přitom chyba nově vzniklého podstromu $\tau|_v$ je:

$$E_s(\tau|_v) = \min\{E_b(v), E_b(\tau|_v)\} \tag{3.14}$$

Chyba E_s se odhaduje pomocí Bayesovy metody odhadu pravděpodobnosti. Konkrétně se používá *Laplaceův odhad pravděpodobnosti*[5] [25] nebo sofistikovanější metoda *m-odhad pravděpodobnosti* [5].

Mějme třídy C a $k = |C|$ je počet tříd. Mějme N vzorů, které se dostanou do uzlu v. Z nich n je počet vzorů takových, že $\{t \in T_v \mid \pi_2(t) = c\}$ nebo-li n je počet trénovacích vzorů, které spadají do třídy c.

Definice 3.4.9 *Laplaceův odhad pravděpodobnosti, že vzor bude patřit do třídy c, je*

$$p = \frac{n+1}{N+k}, \tag{3.15}$$

kde n je počet vzorů uzlu v odpovídajících třídě c, N je celkový počet vzorů v uzlu v a k je počet všech různých tříd.

Tento odhad předpokládá, že apriorní distribuce pravděpodobnosti je stejná pro všechny třídy. A navíc míra ořezávání je ovlivněna celkovým počtem tříd k.

[4] Angl. překlad zpětné chyby je *backed-up error*.

[5] Angl. překlad Laplaceova odhadu pravděpodobnosti je *Laplace's law of succession*.

Definice 3.4.10 *Mějme k různých tříd, n vzorů třídy c z celkových N vzorů v uzlu v. m-odhad pravděpodobnosti, že vzor bude spadat do třídy c je podle Bayesovy metody dán vztahem:*

$$p = \frac{n + p_{ac}m}{N + m}, \tag{3.16}$$

kde p_{ac} je apriorní pravděpodobnost, že vzor bude patřit do třídy c, a m je parametr odhadu.

Všimněme si, že pokud bude m rovno počtu tříd a $p_{ac} = 1/m$, pak výraz 3.16 bude ekvivalentní s Laplaceovým odhadem 3.15. m-odhad přináší tato zlepšení [5]:

(i): m-odhad zavádí do výpočtu také apriorní pravděpodobnost. V případě Laplaceova odhadu byla apriorní pravděpodobnost pro všechny třídy stejná. Taková situace se v praxi stává jen zřídka.

(ii): Změnou parametru m můžeme získat různě ořezané stromy.

(iii): Počet tříd neovlivňuje stupeň ořezání. To, jak bude strom ořezán, závisí na volbě parametru m. m je možné nastavit na základě vlastností z dané oblasti učení, např. na základě úrovně zašumění ve vstupních datech.

K tomu, abychom mohli určit m-odhad, potřebujeme ručně zvolit apriorní pravděpodobnost p_{ac} a hodnotu parametru m. Volba těchto parametrů se jeví jako základní omezení Bayesova odhadu. Hodnota p_{ac} se nejčastěji volí na základě trénovací množiny dat. A sice jako poměr počtu vzorů, které jsou v dané třídě c a celkového počtu vzorů v trénovací množině. Dále je třeba volit parametr m. V nejjednodušším případě se m volí stejné pro všechny atributy a jejich kombinace v dané doméně. Avšak v praxi taková volba parametru m nemusí být postačující. Často m závisí na konkrétním atributu nebo dokonce na konkrétní hodnotě atributu.

Během budování stromových klasifikátorů se uplatňuje pravděpodobnostní odhad v několika fázích:

(i): Při výběru nejlepšího atributu pro kořenový uzel podstromu podle nějakého kritéria (např. podle zisku informace pomocí entropie) ve fázi budování klasifikátoru.

(ii): Ve fázi ořezávání, kdy větve s malou statistickou hodnotou jsou identifikovány a odstraněny.

(iii): V klasifikační fázi, kde na základě trénovacích vzorů v listu je neznámému vzoru přiřazena nejpravděpodobnější třída.

Pro každou fázi se volí odlišné m [5]. Typicky se volí $m = 0$ pro fázi budování stromu. Úplný vybudovaný strom je pouze nová forma reprezentace datové množiny pro učení. Tedy není potřeba přidávat „apriorní informaci“ a zbytečně komplikovat strom. Na druhou stranu ve fázi ořezávání je cílem strom upravit takovým způsobem, aby byl co nejlepší z hlediska rozpoznávání všech dat z dané domény (např. aby dával dobré výsledky na neznámých vzorech - na nezávislé datové množině). V této fázi je nezbytné uvažovat apriorní pravděpodobnost a tedy volit parametr $m > 0$. Konkrétní volba hodnoty m závisí také na úrovni zašumění dat (čím větší zašumění, tím větší hodnota m). m je voleno ručně nebo podle nějaké heurestiky. Např. m může být nastaveno tak, aby maximalizovalo přesnost klasifikátoru na nezávislé množině dat.

m-odhad můžeme použít i v případě výpočtu zpětné chyby E_b, kde je potřeba určit, s jakou pravděpodobností se vzor dostane do daného podstromu. Pro binární rozhodovací podstrom se typicky volí hodnota parametru $m = 2$. Bylo zjištěno, že volba parametru m v tomto případě výrazně neovlivní výsledky [5].

Statická chyba založená na m-odhadu je:

$$E_s = 1 - \frac{n_c + p_{ac} \cdot m}{N + m} = \frac{N - n_c + (1 - p_{ac})m}{N + m}, \tag{3.17}$$

kde $N = |T_v|$, $n_c = |\{t \in T_v \mid \pi_2(t) = c\}|$ a třída c je taková třída, která minimalizuje E_s pro dané m, p_{ac} je apriorní pravděpodobnost třídy c a m je parametr metody odhadu.

V ořezávací fázi se snažíme minimalizovat očekávanou klasifikační chybu pro danou hodnotu m. Stejná hodnota parametru m musí pak být také použita při klasifikaci vzoru novým ořezaným stromem. Neplatí totiž, že chybová funkce nenabývá svého minima ve stejném bodě pro různé hodnoty m [5].

Nyní se zaměříme na to, jakým způsobem ovlivňuje m kombinaci odhadu pravděpodobnosti z dat a apriorní pravděpodobnosti. Výraz 3.16 může být přepsán do tvaru:

$$p = \frac{N}{N + m}\hat{p} + \frac{m}{N + m}p_{ac}, \tag{3.18}$$

kde N je počet vzorů v uzlu, $\hat{p} = n/N$ je poměr počtu vzorů dané třídy a počtu vzorů v uzlu, p_{ac} je apriorní pravděpodobnost, že vzor bude patřit do třídy c a m je parametr metody odhadu.

Z výrazu 3.18 je patrné, že pokud $m = 0$, je p rovno poměru $\hat{p} = n/N$. Pokud $m = N$ (m je rovno počtu vzorů v uzlu), potom p je průměrem hodnot poměru $\hat{p}$ a apriorní pravděpodobnosti p_{ac}. Budeme-li uvažovat limitu pro m jdoucí do nekonečna, bude p rovno p_{ac} [5].

3.4.3 Minimal Cost-Complexity Pruning

Tento způsob ořezávání se používá k ořezání stromů CART [2]. Základní myšlenkou Minimal Cost-Complexity Pruning (MCCP)[6] je nalezení posloupnosti stromů, které minimalizují funkci - *cenu stromu založenou na jeho složitosti* (dále budeme označovat CCR[7]). CCR lineárně kombinuje klasifikační chyby a počet listů stromu, který je ovlivněn parametrem α. Pro různé hodnoty α jsou generovány různé posloupnosti stromů. K výběru nejlepšího stromu z nalezené posloupnosti se používá standardní chyba SE. SE zadefinujeme později.

Mějme rozhodovací strom (G, δ, κ), který je vybudován a připraven k ořezání.

Definice 3.4.11 *Cenu stromu založenou na jeho složitosti definujeme jako:*

$$R_\alpha(G) = R(G) + \alpha \cdot |L(G)|, \tag{3.19}$$

kde α je reálné číslo a $|L(G)|$ označuje počet listů stromu G a $R(G)$ je míra chybné klasifikace stromu G.

Cílem ořezávání MCCP je najít optimálně ořezaný podstrom. Jeho definici ukážeme dále. Jak již bylo poznamenáno, nejdříve vytvoříme posloupnost podstromů původního vybudovaného stromu. Tyto podstromy jsou navzájem vnořené a všechny obsahují kořenový uzel původního stromu. Každý strom, který je v této posloupnosti, seřazené sestupně podle složitosti, obsahuje všechny uzly následujícího podstromu.

Definice 3.4.12 *Nechť máme strom G s kořenem r. Potom G' je ořezaným podstromem stromu G, pokud $\exists G|_r : G' = G|_r$. Pro tuto skutečnost zavedeme relaci $G' \preceq G$.*

Definice 3.4.13 *Nechť α je nezáporné reálné číslo. Ořezaný podstrom G' stromu G se nazývá optimálně ořezaný podstrom stromu T vzhledem k α, pokud $R_\alpha(G') = \min_{G'' \preceq G} R_\alpha(G'')$.*

Optimálně ořezaný podstrom nemusí být jediný podstrom s touto vlastností.

Definice 3.4.14 *Nechť G_0 je vybudovaný neořezaný strom. Pokud platí $G' \preceq G''$ pro libovolný optimálně ořezaný podstrom $G'' \preceq G_0$ takový, že $R_\alpha(G') = R_\alpha(G'')$, pak G' je nejmenší optimálně ořezaný podstrom stromu G_0 vzhledem k α a označuje se $G_0(\alpha)$.*

[6] V literatuře se můžeme také setkat s názvem *Error-Complexity Pruning*.
[7] Angl. cost-complexity risk.

Proces ořezávání MCCP se provádí ve dvou krocích. Předpokládejme, že máme vybudovaný strom G_0. V prvním kroku je na základě množiny vzorů pro učení vygenerována posloupnost ořezaných podstromů $\{G_k\}_{k=0}^{K}$ stromu G_0 takových, že $G_0 \succ G_1 \succ G_2 \succ \ldots \succ G_K$, kde G_K obsahuje pouze kořenový uzel stromu G_0. V druhém kroku je pro každý ořezaný podstrom této posloupnosti odhadnuta chyba $\hat{R}(G_k)$ (bude zadefinováno později). Na základě této chyby je z posloupnosti vybrán podstrom s vhodnou velikostí. Vybraný podstrom je nejlepším ořezaným podstromem z hlediska velikosti a přesnosti.

Generování posloupnosti podstromů

Opět podobně jako ve většině ořezávacích algoritmů, definujeme měřítko, které určuje, do jaké míry je uzel stromu vhodným kandidátem k ořezání [2]. Nejčastěji se provádí rozdíl míry chybné klasifikace v uzlu a míry chybné klasifikace v celém podstromu, jehož je tento uzel kořenem.

Definice 3.4.15 *Mějme rozhodovací strom (G, δ, κ), uzel $v \in V(G)$, S_v vzory určené pro růst stromu, které se dostanou při budování stromu do uzlu v a S jsou všechny trénovací vzory. Poměr chybné klasifikace v uzlu v je dán vztahem*

$$R(v) = \left(1 - \frac{n_{\text{maj}}}{|S_v|}\right) \cdot \frac{|S_v|}{|S|} = \frac{|S_v| - n_{\text{maj}}}{|S|}, \tag{3.20}$$

kde $n_{\text{maj}} = |\{t \in S_v \mid \pi_2(t) = c_{\text{maj}}\}|$. Podobně pro podstrom $G|_v$ definujeme míru chybné klasifikace jako:

$$R(G|_v) = \sum_{u \in L(G|_v)} R(u), \tag{3.21}$$

kde $L(G|_v)$ jsou listy podstromu $G|_v$.

Další definice popisují, jakým způsobem je vytvářena posloupnost stromů, které uzly se odřezávají nejdříve a které naposledy. Nakonec uvedeme algoritmus pro generování posloupnosti stromů.

Definice 3.4.16 *Mějme podstrom $G_k|_v$ a s kořenem v. Střední zhoršení míry chybné klasifikace pro uzel v podstromu $G_k|_v$ je dáno:*

$$g_k(v) = \begin{cases} \frac{R(v) - R(G_k|_v)}{|L(G_k|_v)| - 1} & \text{je-li } v \in V(G_k|_v) \setminus L(G_k|_v), \\ +\infty & \text{je-li } v \in L(G_k|_v) \end{cases} \tag{3.22}$$

kde $R(v)$ je poměr chybné klasifikace v uzlu v, $R(G_k|_v)$ je míra chybné klasifikace podstromu $G_k|_v$, $L(G_k|_v)$ je počet listů podstromu $G_k|_v$ a G_k je k-tý strom posloupnosti.

K vybudování posloupnosti podstromů se používá pouze množina vzorů určených pro učení. Na začátku je dána hodnota $\alpha_{\min}$ (nejčastěji $\alpha_{\min} = 0$) a dále máme k dispozici vybudovaný rozhodovací strom (G_0, δ, κ).

Definice 3.4.17 *Mějme rozhodovací strom* (G_0, δ, κ) *a* $\alpha_0 = \alpha_{\min}$. *Posloupnost ořezaných podstromů* $\{G_k\}_{k=0}^{K}$, *kde* K *je počet podstromů posloupnosti, definujme:*

$$G_{k+1} = \{v \in G_k \mid g_k(u) > \alpha_{k+1}, \forall u \in \mathrm{anc}(v)\}, \tag{3.23}$$

kde $\mathrm{anc}(v)$ *jsou předci uzlu* v *a* g_k *je střední zhoršení míry chybné klasifikace. Posloupnost* $\{\alpha_k\}_{k=0}^{K}$ *je definována:*

$$\alpha_{k+1} = \min_{v \in G_k} g_k(v). \tag{3.24}$$

Definice 3.4.17 popisuje posloupnost postupně ořezaných podstromů. Ořeže se takový uzel, v jehož podstromu dojde k největšímu zhoršení klasifikace. Pokud strom takový uzel už neobsahuje, je ořezán uzel, ve kterém dojde k nejmenšímu zlepšení klasifikace. Pokud existují ve stromě ještě další takové uzly, jsou taktéž ořezány.

Mějme posloupnost reálných hodnot $-\infty < \alpha_1 = \alpha_{\min} < \alpha_2 < \ldots < \alpha_K < +\infty$ a posloupnost ořezaných stromů $G_0 \succ G_1 \succ \ldots \succ G_K$ takových, že nejmenší optimálně ořezaný podstrom stromu G_0 pro dané α je

- $G_0(\alpha) = G_0$ pro $\alpha < \alpha_1$
- $G_0(\alpha) = G_0(\alpha_k) = G_k$ pro $\alpha_k \leq \alpha < \alpha_{k+1}$ a $1 \leq k < K$
- $G_0(\alpha) = G_0(\alpha_K) = G_K$ pro $\alpha_K \leq \alpha$

Pro uzel v je v_L levý a v_R pravý potomek. v_P bude označovat rodiče uzlu v. Během ořezávání podstromu budeme u jeho každého uzlu udržovat tyto parametry:

- $N(v) = |L(G_k|_v)|$ je počet listů podstromu $G_k|_v$ stromu G_k posloupnosti.
- $E(v) = R(v)$ v případě, že v je list, jinak $E(v) = R(G_k|_v)$, nebo-li $E(v)$ je míra chybné klasifikace podstromu $G_k|_v$ stromu G_k posloupnosti.
- $\omega(v) = \min_{u \in V(G_k|_v)} g_k(u)$ je minimální zhoršení míry chybné klasifikace v podstromu $G_k|_v$ pro strom G_k posloupnosti.

Algoritmus pro hledání posloupnosti podstromů probíhá tak, že se nejdříve pomocí zpětné propagace od listů ke kořeni spočtou u každého uzlu jeho parametry viz. algoritmus 3.3.

1: **for all** $v \in V(G_0)$ **do**
2: **if** $v \in L(G_0)$ **then**
3: $N(v) \leftarrow 1, E(v) \leftarrow R(v), g(v) \leftarrow +\infty, \omega(v) \leftarrow +\infty$
4: **else**
5: $N(v) \leftarrow N(v_L) + N(v_R)$
6: $E(v) \leftarrow E(v_L) + E(v_R)$
7: $g(v) \leftarrow \frac{R(v)-E(v)}{N(v)-1}$
8: $\omega(v) \leftarrow \min\{g(v), \omega(v_L), \omega(v_R)\}$
9: **end if**
10: **end for**

Algoritmus 3.3: Inicializace vybudovaného stromu G_0.

Potom, co je strom inicializovaný, proběhne samotný algoritmus 3.4 pro generování posloupnosti ořezaných podstromů. Ten spočívá v nalezení „nejhoršího“ uzlu daného podstromu, ve kterém se provedou ořezávací operace a aktualizují se všichni jeho předchůdci až ke kořenu. Algoritmus 3.4 je konečný. V každém kroku jsou odstraněny minimálně dva listové uzly [2].

Výběr optimálně ořezaného podstromu

K tomu, abychom vybrali podstrom z posloupnosti $\{G_k\}_{k=0}^{K}$ původního podstromu G_0, je potřeba určit chybu $\hat{R}(G_k)$ [2]. Pokud nemáme k dispozici testovací množinu, použije se míra chybné klasifikace z definice 3.4.15. V případě, že máme nezávislou testovací množinu dat, použijeme opět tento odhad chyby, avšak chyba je odhadnutá na testovací množině. Na základě tohoto odhadu je vybrán takový podstrom $G_{k'}$, pro který platí:

$$\hat{R}(G_{k'}) = \min_k \hat{R}(G_k), \text{ pro } k \in \{0, 1, 2, \ldots, K\}. \tag{3.25}$$

Výsledný ořezaný podstrom je vybrán na základě standardní chyby SE pomocí *pravidla* b-SE.

Definice 3.4.18 *Nechť $G_{k'}$ je strom posloupnosti s minimální chybou $\hat{R}(G_{k'})$. Standardní chyba* SE *stromu $G_{k'}$ na základě odhadu chyby $\hat{R}(G_{k'})$ je:*

$$\text{SE}(\hat{R}(G_{k'})) = \sqrt{\frac{\hat{R}(G_{k'}) \cdot (1 - \hat{R}(G_{k'}))}{N}}, \tag{3.26}$$

kde N je počet vzorů množiny, na které byla odhadována chyba $\hat{R}(G_{k'})$.

Definice 3.4.19 *Nechť $G_{k'}$ je strom posloupnosti s minimální chybou $\hat{R}(G_{k'})$, b je nezáporná reálná konstanta. Pravidlo b-*SE *vybere největší podstrom $G_{k''}$ takový, že:*

$$\hat{R}(G_{k''}) \leq \hat{R}(G_{k'}) + b \cdot \text{SE}(\hat{R}(G_{k'})). \tag{3.27}$$

1: $k \leftarrow 1, \alpha \leftarrow \alpha_{\min}$
2: **if** $\omega(r) > \alpha$ **then** ▷ r je kořenem podstromu G_{k-1}
3: $\alpha_k \leftarrow \alpha$, $G_k \leftarrow \{v \in G_{k-1} \mid \forall u \in \text{anc}(v) : g(s) > \alpha_k\}$
4: $\alpha \leftarrow \omega(r), k \leftarrow k + 1$
5: **else**
6: **if** $N(r) = 1$ **then**
7: Proces generování je ukončen.
8: **end if**
9: **end if**
10: $v \leftarrow r$ ▷ r je kořenem podstromu G_k
11: **while** $\omega(v) < g(v)$ **do**
12: **if** $\omega(v) = \omega(v_L)$ **then**
13: $v \leftarrow v_L$
14: **end if**
15: **if** $\omega(v) = \omega(v_R)$ **then**
16: $v \leftarrow v_R$
17: **end if**
18: **end while**
19: $L(G_k) \leftarrow L(G_k) \cup \{v\}$ ▷ Vytvoří se nový list v
20: $N(v) \leftarrow 1, E(v) \leftarrow R(v), g(v) \leftarrow +\infty, \omega(v) \leftarrow +\infty$
21: **while** $v \neq r$ **do** ▷ Aktualizují se informace předků uzlu v
22: $v \leftarrow v_P$
23: $N(v) \leftarrow N(v_L) + N(v_R)$
24: $E(v) \leftarrow E(v_L) + E(v_R)$
25: $g(v) \leftarrow \frac{R(v)-E(v)}{N(v)-1}$
26: $\omega(v) \leftarrow \min\{g(v), \omega(v_L), \omega(v_R)\}$
27: **end while**
28: Algoritmus se opakuje, dokud není splněna podmínka v kroku 6.

Algoritmus 3.4: Algoritmus pro generování posloupnosti ořezaných podstromů ve stromu G_0.

Konstanta b v pravidle b-SE je uživatelsky nastavená hodnota. Nejčastěji se používá $b = 1$ [2].

3.4.4 Pessimistic Error Pruning

Pesimistické ořezávání[8] PEP nevyžaduje ořezávací množinu dat [31, 37]. Používá se jako součást Quinlanova C4.5. Algoritmus PEP opět prochází strom od listů směrem ke kořenu. V každém uzlu se rozhoduje o tom, zda provést ořezávací operaci na základě pesimistické aproximace chyby podle tzv. *faktoru věrohodnosti* (též faktor spolehlivosti) pro normální rozdělení.

Předpokládejme, že měříme chybu klasifikátoru na testovací množině vzorů a obdržíme poměr správné klasifikace např. 75% (poměr chybné klasifikace je 25%). Tato hodnota je však pouze odhadnutá. O skutečném poměru správné klasifikace můžeme pouze říci, že se pravděpodobně pohybuje blízko 75%. To, jak blízko, závisí na velikosti množiny vzorů, na které jsme tento poměr měřili. Pokud bude tento poměr naměřen na testovací množině s deseti tisíci vzory, bude odhad věrohodnější než na množině, která bude obsahovat sto vzorů.

Mějme uzel v rozhodovacího stromu. Označme $E_v = |\{t \in T_v \mid \pi_2(t) \neq c_{\text{maj}}(v)\}|$ a nechť $N_v = |T_v|$, kde T_v označuje trénovací množinu dat, která se dostane do uzlu v. Celá trénovací množina dat T je použita pro růst stromu a na ní se odhaduje poměr chyby. Aby bylo možné použít věrohodnost, musí být množina testovacích dat nezávislá. PEP ovšem žádnou nezávislou množinu testovacích dat k dispozici nemá. Kdyby byla použita chyba na množině vzorů určených pro růst, nic by ořezáno nebylo, protože strom se buduje tak, aby došlo k maximálnímu poklesu chyby a v okamžiku, kdy chyba neklesá nebo se dokonce zvětší, růst je stromu je ukončen. PEP však k odhadu chyby používá horní pesimistickou hranici věrohodnosti.

Před procesem ořezávání je stanovena věrohodnost c pro každý uzel ve stromě (v algoritmu C4.5 je typicky nastavena na $c = 25\%$). Na základě parametru c se určí hranice věrohodnosti:

$$\mathrm{P}\left(\frac{f_v - q_v}{\sqrt{q_v(1-q_v)/N_v}}\right) = c, \tag{3.28}$$

kde $f_v = E_v/N_v$ je pozorovaný poměr chyby v uzlu v a q_v vyjadřuje skutečný poměr chyby v uzlu v. Spočteme horní hranici věrohodnosti pro q_v a použijeme jej jako pesimistický odhad poměru chyby v uzlu v [37]:

$$e_v = \left(f_v + \frac{z^2}{2N_v} + z\sqrt{\frac{f_v}{N_v} - \frac{f_v^2}{N_v} + \frac{z^2}{4N_v^2}}\right) \Bigg/ \left(1 + \frac{z^2}{N_v}\right), \tag{3.29}$$

[8] Pessimistic Error Pruning (PEP)

kde z je počet směrodatných odchylek odpovídajících věrohodnosti c (pro $c = 25\%$ je $z = 0{,}69$). Hodnoty z pro příslušné věrohodnosti c lze najít ve statistických tabulkách (např. v [19]).

Formální popis algoritmu PEP je v Algoritmu 3.5. Do algoritmu PEP 3.5 byly zaintegrovány obě dvě metody ořezávání - nahrazení podstromu listem a budování podstromu (viz. definice 3.4.1).

```
1: procedure PEP(uzel v ∈ V(G))
2:     if v ∉ L(G) then
3:         for all u ∈ F_out(v) do
4:             PEP(u)
5:         end for
6:         w ← argmax_{u∈F_out(v)} |T_u|
7:         if strom je ořezáván metodou budování podstromů then
8:             e_S ← e(G|_w) na vzorech T_v
9:         else
10:            e_S ← +∞
11:        end if
12:        e_L ← e(v)          ▷ Chyba pro vrchol v, jakoby v představoval list
13:        e_T ← e(G|_v)
14:        if e_L ≤ e_T and e_L ≤ e_S then
15:            F_out(v) ← {}
16:            L(G) ← L(G) ∪ {v}
17:            return
18:        end if
19:        if e_S ≤ e_T then
20:            G|_v se nahradí podstromem G|_w
21:            T_v se rozdistribuuje do nového podstromu G|_w
22:            PEP(w)
23:        end if
24:    end if
25: end procedure
```

Algoritmus 3.5: Pessimistic error pruning. Všechny chyby $e(.)$ jsou počítány pomocí pesimistického odhadu viz výraz 3.29. Chyba podstromu $e(G|_v)$, resp. $e(G|_w)$ je získána postupnou propagací chyby od listů až ke kořeni daného podstromu v příslušných poměrech.

Praktickou ukázku ořezávání PEP demonstrujeme na rozhodovacím stromě, který je na obr. 3.8. Vstupním parametrem je věrohodnost c. Pro $c = 25\%$ je odpovídající hodnota $z = 0{,}69$ [37]. Proměnné f_v, N_v a E_v budeme v dalším textu uvádět bez spodního indexu v. Pro nejspodnější nejlevější list

je $E = 2$, $N = 6$ a tedy $f = 0{,}33$. Dosadíme-li tyto hodnoty do výrazu 3.29, dostaneme horní hranici věrohodnosti $e = 0{,}47$. To znamená, že místo použití poměru chyby na trénovací množině vzorů (který je 0,33) použijeme pesimistický odhad 0,47. Pro problém se dvěma třídami je tento odhad opravdu pesimistický, protože dosahuje skoro nejhorší možné chyby 0,5. Pro sousední listový uzel je situace ještě horší. Pro tento uzel je $E = 1$ a $N = 2$ (se stejnou věrohodností $c = 25\%$) a tedy horní hranice věrohodnosti $e = 0{,}72$. Třetí list má stejný odhad jako první list.

Dalším krokem algoritmu PEP je kombinace odhadů chyb v těchto třech listech v poměru počtu vzorů, které obsahují a ten je 6 : 2 : 6. A tedy odhad chyby podstromu $G|_{v_3}$ je 0,51. Do kořene v_3 podstromu $G|_{v_3}$ se dostane 14 vzorů, do levého listu se dostane 6, do prostředního 2 a do pravého 6 vzorů. Výsledný odhad podstromu $G|_{v_3}$ se spočte takto:

$$e = \frac{6}{14} \cdot 0{,}47 + \frac{2}{14} \cdot 0{,}72 + \frac{6}{14} \cdot 0{,}47 \doteq 0{,}51. \tag{3.30}$$

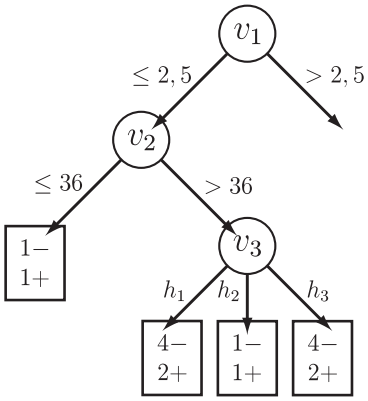
Nyní spočteme chybu uzlu v_3, jako by byl považován za list. Pro tento uzel patří 9 vzorů do třídy '−' a 5 vzorů do třídy '+'. To znamená, že budoucí list by byl označen třídou '−'. Poměr chyby na trénovacích vzorech je $f = \frac{5}{14}$. Pesimistický odhad pro tyto hodnoty je $e = 0{,}46$. Protože tato chyba je menší než chyba $e = 0{,}51$ podstromu $G|_{v_3}$, budou jeho listy ořezány. Posuneme se do uzlu v_2, který má po předchozím ořezání oba dva potomky listy. Odhad chyby v levém listu, pro který $E = 1$ a $N = 2$, je $e = 0{,}72$ a podobně se spočte odhad pro pravý list $e = 0{,}46$. Poměr vzorů uzlu v_2, které se dělí do těchto listů je 2 : 14, což vede k chybě $e = 0{,}49$. Chyba v uzlu v_2 je $e = 0{,}46$, což je méně než v podstromě $G|_{v_2}$ a proto opět dojde k ořezání.

Odhad chyby je založen na mnoha slabých předpokladech. Používají se horní hranice věrohodnosti, předpokládá se normální rozložení a také všechno je odhadováno na trénovací množině dat. I když tento odhad není úplně správný, v praxi dává PEP poměrně slušné výsledky [37]. Míru ořezávání můžeme podle potřeby řídit parametrem věrohodnosti c.

3.5 CART

V této části práce popíšeme budování CART (Classification and Regression Trees) podle [2]. Strom CART je binární rozhodovací strom, jehož tvar je docílen opakovaným dělením uzlu do dvou potomků. Na začátku algoritmu CART je kořen, který obsahuje celou množinu vzorů určenou pro učení. Některé vlastnosti stromů CART jsme zmínili již v předchozích kapitolách. V této kapitole přesně popíšeme především fázi vytvoření stromu a nakonec zmíníme, které metody ořezávání se používají k redukci velikosti stromu.

Základní myšlenkou růstu stromu je výběr nejlepšího dělení mezi všemi možnými děleními v každém uzlu tak, aby nově vytvořené uzly byly co nej-



Obrázek 3.8: Ořezávání stromu metodou PEP. Čísla v obdélnících znázorňují počty vzorů v listech odpovídající třídě +, resp. třídě −. Popis u spojnic uzlů znázorňuje rozhodnutí podle numerického, resp. nominálního atributu.

čistší. Při odhadování nečistoty uzlu se uplatňují pravděpodobnosti, které jsou odhadovány z množiny trénovacích vzorů.

Definice 3.5.1 *Mějme třídy* $C = \{c_1, c_2, ..., c_J\}$ *a uzel* v *rozhodovacího stromu* (G, δ, κ). *Dále mějme množinu všech vzorů* X *a nechť* X_v *je množina vzorů uzlu* v. *Definujme následující pravděpodobnosti:*

- $\pi(j), j = 1, \ldots, J$ *je apriorní pravděpodobnost výskytu třídy* c_j *v prostoru* X.
- $\mathrm{p}(j, v), j = 1, \ldots, J$ *je pravděpodobnost, že vzor z* X *označený třídou* c_j *spadne do uzlu* v.
- $\mathrm{p}(v)$ *je pravděpodobnost, že vzor z* X *spadne do uzlu* v.
- $\mathrm{P}(j|v), j = 1, \ldots, n$ *je pravděpodobnost, že vzor z množiny* X_v *uzlu* v *bude označen třídou* c_j.

Jak bylo poznamenáno výše, tyto pravděpodobnosti se odhadují na základě vzorů z trénovací množiny T. Konkrétní odhad je závislý na tom, jestli je třídový atribut a_c nominální nebo numerický. Pro všechny trénovací vzory $t_1, \ldots, t_n$ definujeme váhy $w_1, \ldots, w_n$ $(0 \leq w_i \leq 1,\ i = 1, \ldots, n)$.

Uvažujme situaci, že třídový atribut je nominální. Definujme pro $\forall t \in T$ funkci $I(\pi_2(t), c_j) = 1$ v případě, že $\pi_2(t) = c_j$, jinak je $I(\pi_2(t), c_j) = 0$.

Definice 3.5.2 *Nechť* T *je trénovací množina vzorů a* $c_j \in \pi_2(T)$. *Vážený součet* N *všech trénovacích vzorů je*

$$N = \sum_{t \in T} w_t, \tag{3.31}$$

vážený součet $N(j)$ trénovacích vzorů, které jsou označeny třídou c_j, je

$$N(j) = \sum_{t \in T} w_t \cdot I(\pi_2(t), c_j), \tag{3.32}$$

vážený součet $N_v(j)$ trénovacích vzorů ve vrcholu v, které jsou označeny třídou c_j, je

$$N_v(j) = \sum_{t \in T_v} w_t \cdot I(\pi_2(t), c_j) \tag{3.33}$$

a vážený součet N_v vzorů ve vrcholu v je dán vztahem

$$N_v = \sum_{t \in T_v} w_t. \tag{3.34}$$

Symbol w_t představuje váhu vzoru t.

Z definice je zřejmé, že $N_v = \sum_{j=1}^{J} N_v(j)$. Pro nominální třídový atribut a_c lze pravděpodobnosti z definice 3.5.1 odhadnout takto [2]:

$$\begin{aligned} \mathrm{p}(j, v) &= \frac{\pi(j) N_v(j)}{N(j)}, \\ \mathrm{p}(v) &= \sum_{j=1}^{J} \mathrm{p}(j, v), \\ \mathrm{P}(j|v) &= \frac{\mathrm{p}(j, v)}{\mathrm{p}(v)} = \frac{\mathrm{p}(j, v)}{\sum_{j=1}^{J} \mathrm{p}(j, v)}. \end{aligned} \tag{3.35}$$

Apriorní pravděpodobnost $\pi(j)$ lze odhadnout z trénovací množiny jako $\pi(j) = N(j)/N$. Pravděpodobnost $\mathrm{p}(j, v)$ se zjednoduší na $\mathrm{p}(j, v) = N_v(j)/N$ a $\mathrm{p}(v) = N_v/N$. K výpočtu nečistoty uzlu je důležitá pravděpodobnost $\mathrm{P}(j|v)$, která je pro $\pi(j) = N(j)/N$ a po dosazení $\mathrm{p}(j, v)$ a $p(v)$ dána vztahem:

$$\mathrm{P}(j|v) = \frac{N_v(j)}{N_v}. \tag{3.36}$$

Tedy pravděpodobnost, že vzor označený třídou c_j spadne do vrcholu v, je dána poměrem počtu trénovacích vzorů $t \in T_v : \pi_2(t) = c_j$ a počtu trénovacích vzorů, které se dostanou do uzlu v. Pro výpočet nečistoty se zavádí pokuta za chybnou klasifikaci. Pokuta za chybnou klasifikaci určuje závažnost chyby. Všechny chyby nemusí být vždy rovnocenné.

Definice 3.5.3 *Nechť máme třídy $c_j, c_k \in C$. Pokuta $C(j|k)$ za to, že vzor označen třídou c_k bude klasifikací označen třídou c_j, je definována takto:*

$$C(j|k) \begin{cases} \geq 0 & \textit{je-li } j \neq k, \\ = 0 & \textit{jinak.} \end{cases} \tag{3.37}$$

Nečistota $i(v)$ uzlu v je pak definována podobně jako v kapitole 3.3:

$$i(v) = \sum_{j,k} C(j|k)\,\mathrm{p}(j|v)\,\mathrm{p}(k|v). \tag{3.38}$$

Pokud nejsou zadány pokuty $C(j|k)$, je implicitně nastaveno $C(j|k) = 1$ pro $j \neq k$ a $C(j|k) = 0$ pro $j = k$. Pokles nečistoty pro konkrétní dělení s, které rozdělí trénovací množinu T_v na T_{v_L} náležící levému potomkovi v_L a na T_{v_R} náležící pravému potomkovi v_R, je dán vztahem

$$\Delta i(s,t) = i(t) - \mathrm{p}_L\, i(v_L) - \mathrm{p}_R\, i(v_R), \tag{3.39}$$

kde p_L a p_R jsou pravděpodobnosti, že daný vzor bude poslán do levého potomka v_L, resp. do pravého potomka v_R. Pravděpodobnosti p_L a p_R lze spočítat jako $\mathrm{p}_L = \mathrm{p}(v_L)/\,\mathrm{p}(v)$ a $\mathrm{p}_R = \mathrm{p}(v_R)/\,\mathrm{p}(v)$ [2]. V případě, že apriorní pravděpodobnost $\pi(j) = N(j)/N$, pak $\mathrm{p}_L = N_{v_L}/N_v$ a $\mathrm{p}_R = N_{v_R}/N_v$.

Pokles nečistoty podle vztahu 3.39 se používá v základní verzi algoritmu pro vytvoření CART stromů. K výpočtu poklesu nečistoty je možné také použít twoing kritérium - viz kapitola 3.2.2.

Nyní uvažujme situaci, kdy třídový atribut a_c má numerický charakter. Nečistotu pro numerický atribut a_c není možné počítat stejně jako v případě nominálního atributu, protože numerický atribut obsahuje nekonečně mnoho hodnot. Výhodou ovšem je, že v případě numerického atributu máme definovanou metriku, tedy můžeme měřit vzdálenost dvou prvků. Pro numerický atribut platí, že čím větší je rozdíl odhadnuté hodnoty a skutečné hodnoty třídového atributu, tím je hodnota nečistoty větší. Nyní nečistotu numerického atributu popíšeme formálněji.

Definice 3.5.4 *Mějme vrchol v stromu* CART*, a_c numerický třídový atribut a T_v jsou trénovací vzory vrcholu v. Odhad vrcholu v pro numerický třídový atribut a_c je*

$$\bar{\kappa}(v) = \frac{\sum_{t \in T_v} w_t \cdot \pi_2(t)}{\sum_{t \in T_v} w_t} \tag{3.40}$$

kde w_t je váha trénovacího vzoru $t \in T_v$ a $\pi_2(t)$ je třída odpovídající vzoru t.

Definice 3.5.5 *Pokud je třídový atribut numerický, nečistota $i(v)$ vrcholu v je definována pomocí čtvercové odchylky:*

$$i(v) = \frac{\sum_{t \in T_v} w_t(\pi_2(t) - \bar{\kappa}(v))^2}{\sum_{t \in T_v} w_t}, \tag{3.41}$$

kde $\bar{\kappa}(v)$ je odhad vrcholu v, w_t je váha vzoru t, $\pi_2(t)$ je třída odpovídající vzoru t a T_v je trénovací množina vrcholu v.

Úbytek nečistoty pro numerický třídový atribut a dělení s je téměř totožný s úbytkem nečistoty pro nominální třídový atribut. Nečistotu pro nominální třídový atribut nahradí nečistota pro numerický třídový atribut:

$$\Delta i(s, v) = i(v) - \mathrm{p}_L \, i(v_L) - \mathrm{p}_R \, i(v_R), \tag{3.42}$$

kde $\mathrm{p}_L = N_{v_L}/N_v$ a $\mathrm{p}_R = N_{v_R}/N_v$. N_{v_L}, N_{v_R} a N_v jsou opět součty vah vzorů v daných vrcholech.

Odhad poklesu nečistoty se někdy zpřesňuje tím, že se vynásobí pravděpodobností uzlu v:

$$\Delta I(s, v) = \mathrm{p}(v) \cdot \Delta i(s, v). \tag{3.43}$$

Strom CART se vytváří algoritmem typu rozděl a panuj. Postupně se hierarchicky dělí trénovací množina na dvě co nejčistší disjunktní podmnožiny do doby, kdy je splněno ukončovací kritérium. Každé této podmnožině odpovídá jeden vrchol stromu. V každém nově vytvořeném vrcholu se hledá nejlepší dělení pro každý atribut z hlediska úbytku nečistoty. Z této množiny jednotlivých dělení se vybere takový atribut a jemu odpovídající nejlepší dělení s největším úbytkem nečistoty.

Hledání atributu a dělení

Nyní popíšeme situaci, kdy algoritmus pro vytvoření CART je v uzlu v a hledá pro následníky v_L a v_R takový netřídový atribut a_i ($i = 1, \ldots, A$) a jemu odpovídající dělení s, které rozdělí množinu trénovacích vzorů T_v na co nejčistší T_{v_L} a T_{v_R}. Hledání takového dělení a atributu probíhá ve dvou krocích. V prvním kroku se pro každý netřídový atribut a_i spočte nejlepší dělení s_i. V druhém kroku se mezi všemi nalezenými děleními s_i vybere nejlepší dělení s. Nyní se zaměříme na první krok detailněji.

Uvažujme situaci, že a_i je numerický atribut. Algoritmus setřídí vzestupně jeho hodnoty $a_i(t), t \in T_v$. Výsledkem je setříděná posloupnost hodnot atributů vzorů $a_i(t_1) \leq a_i(t_2) \leq \ldots \leq a_i(t_{|T_v|})$. Z posloupnosti $\{a_i(t_j)\}_{j=1}^{|T_v|}$ vynecháme opakující se hodnoty. Vznikne tím posloupnost $\{a_i(t_{j_k})\}_{k=1}^{K}$, kde $a_i(t_{j_1}) < a_i(t_{j_2}) < \ldots < a_i(t_{j_K})$. Z této rostoucí posloupnosti odebereme poslední člen $a_i(t_{j_K})$. V zbylých bodech $a_i(t_{j_k}), k = 1, \ldots, K-1$ počítáme úbytek nečistoty Δi. Body $a_i(t_{j_k}), k = 1, \ldots, K-1$ budou představovat všechna možná dělení atributu a_i. Proto byl také odebrán poslední atribut, protože dělením podle poslední hodnoty atributu by k žádnému dělení nedošlo. Tedy atribut s K různými hodnotami má $K-1$ různých dělení. Vzory $t \in T_v$, které mají hodnotu atributu $a_i(t) \leq a_i(t_{j_k})$, budou přiřazeny levému vrcholu v_L, v opačném případě budou přiřazeny pravému vrcholu v_R. Pro nejlepší dělení s_i' platí:

$$s'_i = \underset{a_i(t_{j_k})}{\text{argmax}} \Delta i(a_i(t_{j_k}), v). \tag{3.44}$$

Dělení s'_i atributu a_i v uzlu v je takové dělení $a_i(t_{j_k})$, které způsobí největší pokles nečistoty.

Nyní budeme uvažovat případ, že a_i je nominální atribut. Nominální atribut nemá žádné uspořádání a tedy není možné použít stejný postup jako v případě numerického atributu. Předpokládejme, že $A_i = \{a_i(t) \mid t \in T_v\}$ jsou hodnoty atributu a_i trénovacích vzorů T_v v uzlu v. Označme $S_i \subset A_i$. Vzory $\{t \in T_v \mid a_i(t) \in S_i\}$ budou patřit levému následníkovi v_L uzlu v a ostatní vzory budou patřit pravému následníkovi v_R. Nechť $n = |A_i|$. Dělení nominálního atributu a_i je tvořeno všemi podmnožinami $S_i \subset A_i$. S_i nazvěme dělením nominálního atributu a_i. Počet všech možných dělení nominálního atributu a_i je tedy $2^{n-1}-1$. Předpokládejme všechna možná dělení S_i atributu a_i. Pro nejlepší dělení S'_i pro nominální atribut a_i platí:

$$S'_i = \underset{S_i}{\text{argmax}} \Delta i(S_i, v), \tag{3.45}$$

kde Δi má mírně pozměněný význam, avšak intuitivně vyjadřuje to samé jako v případě numerického atributu. Množina trénovacích vzorů T_v je podle S'_i rozdělena na levou trénovací množinu T_{v_L} a pravou trénovací množinu T_{v_R} a na základě těchto množin vzorů jsou určeny nečistoty pro uzel a jeho potomky.

Uvažujme druhý krok výběru nejlepšího dělení a atributu uzlu v. Nechť s_i označuje nejlepší dělení pro atribut a_i. Pokud je atribut numerický je $s_i = s'_i$. V případě, že atribut je nominální, nechť platí $s_i = S'_i$. Pro tato dělení máme odpovídající úbytky nečistot $\Delta i(s_i, v)$. Pro nejlepší dělení s uzlu v platí:

$$s = \underset{s_i}{\text{argmax}} \Delta i(s_i, v). \tag{3.46}$$

Nejlepšímu dělení s odpovídá nejlepší atribut a. Na základě tohoto dělení a atributu a rozdělíme trénovací množinu vzorů T_v na levou podmnožinu T_{v_L} odpovídající levému následníkovi v_L uzlu v a pravou podmnožinu T_{v_R} odpovídající pravému následníkovi v_R. Celý proces hledání optimálního dělení a atributu se pak rekurzivně aplikuje na nové poduzly v_L a v_R. Dělí se do doby, kdy je splněno ukončovací kritérium.

K dělení nominálního atributu je dobré doplnit, že v praxi se většinou nehledají všechny možné podmnožiny hodnot nominálního atributu a_i. Vybere se pouze jedna konkrétní hodnota h atributu a_i daného vzoru v uzlu v. T_{v_L} je pak tvořena všemi vzory $\{t \in T_v \mid a_i(t) = h\}$ a T_{v_R} všemi ostatními vzory $t \in T_v \setminus T_{v_L}$. Oba dva přístupy jsou ekvivalentní [2]. Vychází se z toho, že binární strom, sestávající se pouze z jednodušších dělení, pokryje libovolnou podmnožinu hodnot atributu a_i. Nevýhodou jednoduššího dělení

nominálního atributu je větší složitost stromu. Naopak výhodou je jednodušší hledání dělení a také menší výpočetní složitost pro rozhodování v konkrétním uzlu při klasifikaci.

Inicializace:

Vytvoří se nový kořen r stromu G.
$V(G) \leftarrow \{r\}, E(G) \leftarrow \emptyset, L(G) \leftarrow \emptyset$

Procedura CART je zavolána se vstupními parametry r a T, kde T je trénovací množina vzorů.

1: **procedure** CART(vrchol v, trénovací množina T_v)
2: **if** je splněna ukončovací podmínka **then**
3: $L(G) \leftarrow L(G) \cup \{v\}$
4: $\kappa(v) \leftarrow c_{\text{maj}}(v)$
5: **return**
6: **end if**
7: Nechť $\{a_1, \ldots, a_n\}$ jsou atributy T_v bez třídového atributu a_c.
8: Pro každý atribut a_i se najde nejlepší dělení s_i.
9: $s \leftarrow \underset{s_i}{\text{argmax}} \Delta i(s_i, v)$
10: T_v se rozdělí podle s na T_{v_L} a T_{v_R}.
11: $\delta(v, x) \stackrel{def}{=} v_L, \forall x \in X$, které patří podle s levému následníkovi v_L.
12: $\delta(v, x) \stackrel{def}{=} v_R, \forall x \in X$, které patří podle s pravému následníkovi v_R.
13: $E(G) \leftarrow E(G) \cup \{(v, v_L)\} \cup \{(v, v_R)\}$
14: CART(v_L, T_{v_L})
15: CART(v_R, T_{v_R})
16: **end procedure**

Algoritmus 3.6: Schéma budování rozhodovacího stromu CART.

Ukončovací podmínky

Ukončovací podmínky kontrolují proces růstu stromu a ve vhodnou chvíli růst ukončí. Růst rozhodovacího stromu je ukončen, pokud je splněna aspoň jedna z následujících podmínek:

- Vrchol v je čistý, tedy $\exists(c \in C)\forall(t \in T_v)\pi_2(t) = c$.
- Pro každý netřídový atribut a platí: $a(t_1) = a(t_2) = \ldots = a(t_n)$, kde $t_1, t_2, \ldots, t_n \in T_v$ a T_v jsou trénovací vzory uzlu v.
- Hloubka vybudovaného stromu dosáhne uživatelem definovanou limitní hloubku d.

- Počet vzorů $|T_v|$ ve vrcholu v je menší než uživatelem definovaný minimální počet vzorů ve vrcholu.
- Nejlepší dělení má alespoň jednoho potomka takového, že počet vzorů odpovídající tomuto potomkovi je menší než uživatelem definovaná hodnota.
- Pro nejlepší dělení s vrcholu v je $\Delta i(s, v) < \epsilon$, kde ϵ je minimální zlepšení.

Formální popis pro vybudování CART stromu je zachycen v Algoritmu 3.6.

Strom se nechává obyčejně vyrůst do maximální velikosti tak, že se volí mírnější ukončovací podmínky [2]. Takto vybudovaný strom je ovšem často přetrénovaný a má velkou složitost. Je tedy potřeba zlepšit přesnost klasifikace stromu na testovacích datech a zmenšit složitost pomocí ořezávání. CART stromy je možné ořezat metodou minimal cost-complexity pruning (MCCP), která byla popsána v části 3.4.3. Metoda MCCP najde takový ořezaný strom, který je kompromisem mezi složitostí a přesností na testovací množině.

3.6 C4.5

Rozhodovací stromy C4.5 byly vyvinuty během řady let J. Ross Quinlanem v Austrálii na Univerzitě v Sydney [31]. Budování stromu C4.5 je založeno na zisku informace podobně jako u algoritmu ID3. Algoritmus pro budování stromu C4.5 však přináší v tomto ohledu zlepšení. Zavádí tzv. *poměr zisku*. Poměr zisku odstraňuje problém s nominálními atributy, které mají mnoho různých hodnot. Později popíšeme poměr zisku podrobněji. Algoritmus pro budování C4.5 integruje do algoritmu numerické atributy, chybějící hodnoty, zacházení se zašuměnými daty a také generování pravidel ze stromu. Generování pravidel je však nad rámec této práce. Více o generování pravidel ze stromu je možné najít např. v [31]. V této části se budeme hlavně zabývat budováním rozhodovacího stromu metodou C4.5.

Podobně jako ve všech stromových algoritmech je strom budován rekurzivním způsobem. Na začátku se vybere nejlepší atribut pro kořenový vrchol a na základě něj se vytvoří větve. Trénovací množina vzorů odpovídající kořenovému uzlu je podle atributu rozdělena na podmnožiny, které odpovídají jednotlivým větvím. Vznikají nové uzly a jim odpovídají podmnožiny vzorů. Proces se rekurzivně opakuje pro všechny nově vytvořené potomky. Dělení je zastaveno, pokud je splněna ukončovací podmínka.

Algoritmus pro vytvoření stromu C4.5 umí pracovat s nominálními i numerickými atributy. Ovšem třídový atribut musí být nominální. Nyní popíšeme hlavní část algoritmu a tou je výběr nejlepšího atributu, podle kterého se bude dělit trénovací množina do podmnožin. Opět se dělí tak, aby nově

vzniklé podmnožiny vzorů byly co nejčistší. Dělení podle nominálního atributu se mírně liší od dělení podle numerického atributu.

Jako měřítko nečistoty se zde uplatňuje entropie. Entropie $i(v)$ pro uzel v reprezentuje očekávané množství informace nezbytné pro klasifikaci předloženého vzoru za podmínky, že tento vzor se dostane do uzlu v:

$$i(v) = -\sum_{j=1}^{|C_v|} \mathrm{P}(j|v) \log_2(\mathrm{P}(j|v)). \tag{3.47}$$

$|C_v|$ značí počet tříd odpovídající vzorům, které se dostanou do uzlu v, $\mathrm{P}(j|v)$ je pravděpodobnost, že vzor označený třídou c_j spadne do vrcholu v. Pravděpodobnost $\mathrm{P}(j|v)$ se odhaduje jako poměr váženého součtu trénovacích vzorů označených třídou c_j a váženého součtu trénovacích vzorů v uzlu v, tedy $N_v(j)/N_v$ (viz vzorec 3.35). Počet počítání jednotlivých podílů $N_v(j)/N_v$ lze díky vlastnostem logaritmu zredukovat:

$$i(v) = -\sum_{j=1}^{|C_v|} \frac{N_v(j)}{N_v} \log_2 \left(\frac{N_v(j)}{N_v} \right) = \log_2(N_v) - \frac{1}{N_v} \sum_{j=1}^{|C_v|} N_v(j) \log_2(N_v(j)) \tag{3.48}$$

Úbytek nečistoty nebo-li zisk nové informace pro atribut se spočte jako rozdíl entropie v uzlu v a součtu poměrů entropií v jeho potomcích $v_i, i = 1, \ldots, n$:

$$\Delta i(v) = i(v) - \sum_{j=1}^{n} \mathrm{p}_j \, i(v_j), \tag{3.49}$$

kde p_j je pravděpodobnost, že vzor bude poslán do j-tého potomka v_j. Tuto pravděpodobnost lze odhadnout jako N_{v_j}/N_v, kde N_{v_j} je vážený součet trénovacích vzorů v uzlu v_j a podobně N_v je vážený součet trénovacích vzorů v uzlu v (viz definice 3.5.2).

Pro každý atribut se spočte $\Delta i(v)$. Vybere se opět takový atribut a_i, pro který je $\Delta i(v)$ maximální. V případě nominálního atributu je počet poduzlů $v_j \in F_{out}(v)$ roven počtu jeho možných hodnot v uzlu v. Díky tomu se nebude podle tohoto atributu v podstromě již dělit, protože každé možné hodnotě odpovídá právě jedna větev. Tudíž v odpovídajících podmnožinách vzorů jsou hodnoty tohoto atributu stejné a dělení podle tohoto atributu na cestě od uzlu směrem k listům by nemělo smysl. U numerického atributu se dělí množina trénovacích vzorů právě na dvě podmnožiny. Pro numerický atribut se najde takové dělení, pro které je zisk $\Delta i(v)$ maximální. Dělení podle stejného numerického atributu je na cestě od daného uzlu k listům možné zopakovat, protože konkrétním dělením se dosud nevyčerpaly všechny možnosti, jak lze rozdělit množinu vzorů podle tohoto atributu.

Pro nominální atribut se někdy povolují jenom binární dělení. Trénovací množina je rozdělena právě na dvě podmnožiny. Vzory, které mají danou hodnotu atributu jsou poslány doleva a vzory s jinou hodnotou jdou doprava. V tomto případě je možné atribut testovat na cestě od uzlu k listům taktéž vícekrát.

Hledání atributu a dělení

Předpokládejme, že hledáme optimální nominální atribut a je přípustné vícenásobné rozdělení uzlu. Mějme libovolný nominální atribut a_i a vrchol v. Atributu a_i odpovídají hodnoty $h_1, h_2, \ldots, h_n$. V případě, že tento atribut a_i je optimální, jsou vytvořeny nové poduzly $v_1, v_2, \ldots, v_n$ uzlu v (a tedy také odpovídající hrany $(v, v_1), (v, v_2), \ldots, (v, v_n)$). Uzlu v náleží množina trénovacích vzorů T_v a poduzlům v_i náleží množiny $T_{v_1}, T_{v_2}, \ldots, T_{v_n}$ takové, že $T_{v_i} \subset T_v, i = 1, \ldots, n$ a $T_{v_i} \cap T_{v_j} = \emptyset, \forall i \neq j$. Zároveň platí, že $\bigcup_{i=1}^{n} T_{v_i} = T_v$. Množina trénovacích vzorů T_{v_j} obsahuje takové vzory $t \in T_v$, pro které je hodnota atributu $a_i(t) = h_j$, tedy $T_{v_j} = \{t \in T_v \mid a_i(t) = h_j\}$.

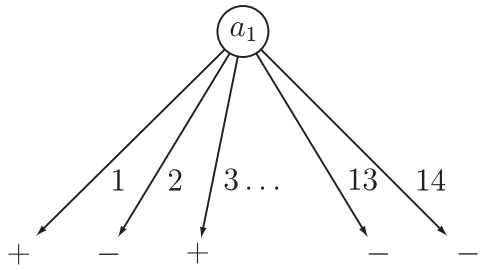
Uvažujme situaci, kdy nominální atribut má mnoho možných hodnot. Odpovídající vrchol v pak bude mít velké rozvětvení $F_{out}(v)$. Čím víc se bude počet možných hodnot přibližovat počtu trénovacích vzorů v uzlu, tím čistší pak budou podvrcholy $v_i \in F_{out}(v)$ a tím bude také růst zisk informace [37]. Tento jev není často žádoucí. Zmíněný problém demonstrujeme na extrémním příkladě. Uvažujme trénovací množinu T_v odpovídající uzlu v, která obsahuje celkem 14 vzorů, 2 netřídové atributy a_1 a a_2 a jeden binární třídový atribut a_c (viz tabulka 3.1).

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}
a_1	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a_2	a	a	a	a	a	b	b	b	b	c	c	c	c	c
a_c	+	−	+	+	+	+	+	+	+	−	−	+	−	−

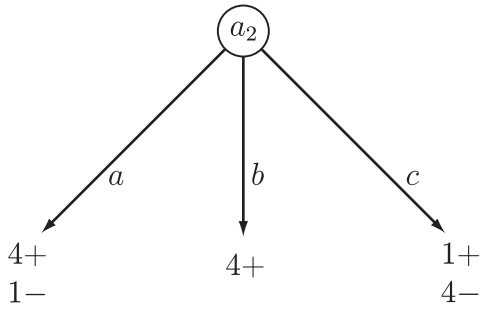
Tabulka 3.1: Trénovací množina vzorů odpovídající uzlu v. Extrémní případ, kdy atribut a_1 má počet různých hodnot roven počtu všech vzorů. Všechny atributy jsou nominální. Symboly '+' a '−' značí třídy.

Pro první atribut a_1 je zisk informace $\Delta i(v)$ (viz vzorec 3.49) v uzlu v roven 0,9403, protože entropie v uzlu v je 0,9403 a součet poměrů entropií v uzlech $v_i \in F_{out}(v)$ je 0, neboť každý uzel v_i obsahuje právě jeden trénovací vzor. Dělení uzlu v podle atributu a_1 je znázorněno na obr. 3.9.

Pro atribut a_2 je zisk informace $\Delta i(v)$ v uzlu v roven 0,4246, protože entropie v uzlu v je pro 9 kladných vzorů a 5 záporných vzorů opět 0,9403. Součet poměrů entropií v poduzlech $v_i \in F_{out}(v)$ je 0,5157. Dělení podle atributu a_2 je znázorněno na obr. 3.10.



Obrázek 3.9: Uzel v s velkým větvením. Podle atributu a_1 vznikne velké množství potomků. Počet potomků je roven počtu všech trénovacích vzorů v uzlu v. Čísla na spojnicí označují hodnotu atributu a_1, symboly '+' a '−' pod spojnicemi značí třídu třídu odpovídajícího vzoru.



Obrázek 3.10: Dělení podle atributu a_2.

Pro atribut a_1 je zisk informace vyšší a tedy dělení v uzlu v proběhne podle tohoto atributu. Ovšem větvení podle atributu a_1 není příliš dobré pro správnou klasifikaci neznámého vzoru a neříká nic o struktuře dat. Obecně zisk informace preferuje atributy s velkým počtem možných hodnot [37].

Pro odstranění této nepříjemnosti zavedeme *poměr zisku*. Poměr zisku dostaneme tak, že do výpočtu zisku uzlu v vzhledem k nominálnímu atributu a_i zahrneme také počet a velikost podmnožin vzorů T_{v_j} odpovídajících poduzlům $v_j \in F_{out}(v)$. Tato úprava nemůže být závislá na třídách trénovacích vzorů.

Definice 3.6.1 *Mějme uzel v rozhodovacího stromu. Poměr zisku informace $\Delta i_r(v)$ je definován vztahem:*

$$\Delta i_r(v) = \frac{\Delta i(v)}{e(v)}, \tag{3.50}$$

kde $e(v)$ je penalizační funkce a $\Delta i(v)$ je zisk informace uzlu v.

Penalizační funkce $e(v)$ by měla být závislá na velikostech množin $|T_{v_j}|, v_j \in F_{out}(v)$ a také na počtu poduzlů $|F_{out}(v)|$ uzlu v. Pro tento účel lze použít entropii, která je v tomto tvaru [37]:

$$e(v) = -\sum_{i=1}^{|F_{out}(v)|} \frac{|T_{v_i}|}{|T_v|} \log_2 \left(\frac{|T_{v_i}|}{|T_v|} \right), \tag{3.51}$$

kde $|T_{v_i}|$ je počet trénovacích vzorů v potomkovi v_i a $|T_v|$ počet trénovacích vzorů v uzlu v. Takto definovaná funkce odpovídá našim požadavkům. Čím je větší faktor větvení $|F_{out}(v)|$, tím je $e(v)$ menší, a podobně, čím větší jsou T_{v_i}, tím je $e(v)$ větší.

Nyní se opět vrátíme k extrémnímu příkladu. Pro první atribut a_1 je poměr zisku $\Delta i_r(v) = 0{,}247$ a pro druhý atribut a_2 je poměr zisku $\Delta i_r(v) = 0{,}269$ a tedy v tomto případě bude vybrán lepší atribut a_2. Dělení podle atributu a_1 by rozdělilo trénovací množinu vzorů T_v na úplně čisté podmnožiny $T_{v_1}, T_{v_2}, \ldots, T_{v_{14}}$, ale faktor větvení $F_{out}(v)$ je však mnohem větší než pro dělení podle atributu a_2. Dělení podle atributu a_2 nerozdělí T_v na úplně čisté podmnožiny, ale faktor větvení bude mnohem menší. Díky poměru zisku bude preferováno dělení podle atributu a_2.

V některých situacích poměr zisku dává přednost atributům s velmi nízkou informací (nižší než v ostatních atributech) [37]. Tomuto se dá vyhnout tak, že se vybere atribut, který maximalizuje poměr zisku za podmínky, že zisk informace pro tento atribut je aspoň tak velký, jako je průměrný zisk informace pro všechny testované atributy.

Nyní popíšeme proces hledání optimálního atributu v případě, že atribut je numerický. V základní verzi algoritmu pro budování stromu C4.5 se používají pouze binární dělení podle numerického atributu, která rozdělí trénovací množinu vzorů T_v na levou podmnožinu T_{v_L} a pravou podmnožinu T_{v_R}. Předpokládejme, že máme numerický atribut a_i. Stejně jako v případě algoritmu pro budování stromu CART (viz Algoritmus 3.5) vzestupně setřídíme hodnoty atributu a_i pro vzory v uzlu v. Opět vynecháme opakující se hodnoty této posloupnosti. Tím vznikne rostoucí posloupnost $\{a_i(t_{j_k})\}_{k=1}^{K}$ hodnot atributů ve vrcholu v. V dalším kroku vybereme všechna možná dělení numerického atributu a_i vzorů v uzlu v. Pro dělení s_l platí:

$$s_l = \frac{a_i(t_{j_l}) + a_i(t_{j_{l+1}})}{2}, l = 1, \ldots, K-1. \tag{3.52}$$

Tedy dělení s_l leží uprostřed mezi dvěma sousedními hodnotami atributu a_i. Hodnota dělení s_l může být vybraná nějakou sofistikovanější metodou v případě, že víme více o rozložení vzorů v příznakovém prostoru. Často se také používá výpočet hodnoty dělení s_l na základě nejbližších hodnot atributu vzorů. Nejlepší dělení s_l je takové, které maximalizuje zisk informace $\Delta i(v)$ [31].

V každém uzlu dojde k dělení trénovací množiny vzorů na podmnožiny. To se děje rekurzivním způsobem ve všech poduzlech. Pro každý vrchol je nutno mít k dispozici uspořádanou množinu vzorů podle hodnot každého

numerického atributu. Není však nutné vzory znova uspořádat v každém poduzlu podle již použitého atributu [37]. Nalezené pořadí v rodičovském uzlu může být použito k odvození pořadí každého potomka. Tímto se docílí větší rychlosti v procesu hledání optimálního dělení numerického atributu.

V dalším příkladě demonstrujeme jak využít setříděnou posloupnost vzorů podle hodnoty atributu v uzlu v pro jeho potomky. Uvažujme první dva atributy trénovacích vzorů v tabulce 3.2, které jsou vzestupně setříděny podle numerického atributu a_2.

	t_7	t_6	t_5	t_9	t_4	t_{14}	t_8	t_{12}	t_{10}	t_{11}	t_2	t_{13}	t_3	t_1
a_1	o	r	r	s	r	r	s	o	r	s	s	o	o	s
a_2	64	65	68	69	70	71	72	72	75	75	80	81	83	85

Tabulka 3.2: První dva atributy trénovacích vzorů. První atribut a_1 je nominální, druhý atribut a_2 je numerický. Vzory jsou vzestupně uspořádané podle numerického atributu a_2.

Předpokládejme, že se nejdříve v uzlu v rozhodneme dělit podle nominálního atributu a_1. V tomto případě vzniknou tři poduzly $v_i \in F_{out}(v), i = 1, 2, 3$. Poduzlu v_1 budou odpovídat vzory $T_{v_1} = \{t \in T_v \mid a_1(t) = \text{'}o\text{'}\}$, uzlu v_2 budou odpovídat vzory $T_{v_2} = \{t \in T_v \mid a_1(t) = \text{'}r\text{'}\}$ a uzlu v_3 budou odpovídat vzory $T_{v_3} = \{t \in T_v \mid a_1(t) = \text{'}s\text{'}\}$. Uvažujme uzel v_3, pro jehož vzory $t \in T_{v_3}$ je $a_1(t) = \text{'}s\text{'}$. Množina T_{v_3} je tvořena vzory $T_{v_3} = \{t_1, t_2, t_8, t_9, t_{11}\}$. Společně s množinou vzorů si udržujeme uspořádání vzorů podle hodnot numerického atributu a_2 tak, že vzor t_7 obsahuje ukazatel na vzor t_6, t_6 ukazuje na t_5, t_5 ukazuje na t_9 atd. V případě, že hledáme uspořádání vzorů T_{v_3} podle numerického atributu v uzlu v_3, stačí projít toto uspořádání vzorů podle hodnot atributu a_2 a přečíst v příslušném pořadí vzory, pro které je hodnota atributu $a_1(t) = \text{'}s\text{'}$. Tím dostaneme uspořádanou posloupnost vzorů $\{t_9, t_8, t_{11}, t_2, t_1\}$ podle hodnot atributu a_2. Opakovanému třídění podle hodnot vzorů atributu se můžeme vyhnout tak, že pro každou podmnožinu odpovídající danému uzlu budeme udržovat pořadí vzorů podle hodnot všech numerických atributů. V případě, že setřídíme na začátku vzory pro každý numerický atribut, není již další třídění třeba.

Další optimalizací výpočtu nejlepšího dělení numerického atributu spočívá v redukci počtu testovaných dělení.

Tvrzení 3.6.2 *Mějme uspořádané trénovací vzory podle hodnot numerického atributu a_i. Mějme dělení s a nechť vzory t_1 a t_2 jsou vzory z T_v takové, že $a_i(t_1) < s$, $a_i(t_2) > s$ a $\nexists t \in T_v : a_i(t) \in (a_i(t_1), a_i(t_2))$. Zisk informace stačí počítat pouze pro taková dělení s, pro která platí:*

$$\pi_2(t_1) \neq \pi_2(t_2). \tag{3.53}$$

1: **function** C4.5(trénovací vzory T_v, netřídové atributy A, uzel v)
2: **if** $T_v = \emptyset$ **then**
3: **return** v
4: **end if**
5: $V(G) = V(G) \cup \{v\}$
6: **if** splněna jedna z ukončovacích podmínek **then**
7: $L(G) \leftarrow L(G) \cup \{v\}$
8: $\kappa(v) \leftarrow c_{\text{maj}}(v)$
9: **return** v
10: **end if**
11: Nechť $a \in A$ je atribut, pro který je $\Delta i(v)$ (resp. $\Delta i_r(v)$) maximální.
12: **if** a je nominální **then**
13: $A \leftarrow A \setminus \{a\}$
14: **for all** hodnoty h_i atributu a **do**
15: $V(G) \leftarrow V(G) \cup \{v_i\}$ ▷ Vytvoří se nový uzel v_i.
16: $T_{v_i} \leftarrow \{t \in T_v \mid a(t) = h_i\}$
17: $v_i \leftarrow$ C4.5(T_{v_i}, A, v_i)
18: $E(G) = E(G) \cup \{(v, v_i)\}$ ▷ v_i je následník v.
19: $\delta(v, o) \stackrel{def}{=} v_i, \forall o \in O, a(o) = h_i$
20: **end for**
21: **end if**
22: **if** a je numerický **then**
23: Nechť s je nejlepší dělení atributu a vzhledem k T_v
24: $T_{v_L} \leftarrow \{t \in T_v \mid a(t) < s\}$, $T_{v_R} \leftarrow \{t \in T_v \mid a(t) > s\}$
25: $V(G) \leftarrow V(G) \cup \{v_L, v_R\}$ ▷ Vytvoří se nové uzly v_L a v_R.
26: $v_L \leftarrow$ C4.5(T_{v_L}, A, v_L)
27: $v_R \leftarrow$ C4.5(T_{v_R}, A, v_R)
28: $E(G) \leftarrow E(G) \cup \{(v, v_L)\} \cup \{(v, v_R)\}$ ▷ v_L a v_R jsou následníci v.
29: $\delta(v, o) \stackrel{def}{=} v_L, \forall o \in O, a(o) < s$
30: $\delta(v, o) \stackrel{def}{=} v_R, \forall o \in O, a(o) > s$
31: **end if**
32: **return** v
33: **end function**

Algoritmus 3.7: Algoritmus C4.5. A představuje všechny atributy bez třídového atributu a_c. Funkce vrací odkaz na uzel podstromu, který právě vytvořila. Výsledkem je odkaz na kořen rozhodovacího stromu (G, δ, κ).

a_2	64	65	68	69	70	71	72	72	75	75	80	81	83	85
a_c	+	−	+	+	+	−	−	+	+	+	−	+	+	−

Obrázek 3.11: Redukce počtu dělení. Zelenou barvou jsou označena dělení, ve kterých má smysl počítat zisk informace. Červenou barvou je označeno dělení, pro které je zbytečné počítat zisk informace.

To znamená, že dělení mezi hodnotami atributu vzorů se stejnou třídou nemohou být optimální (viz obr. 3.11). Tvrzení dokázali Fayad a Irani v [11].

Algoritmus pro budování stromu C4.5 je formálně popsán v Algoritmu 3.7. Algoritmus dostane na vstup množinu trénovacích vzorů, které odpovídá kořenový vrchol v. Najde se nominální atribut nebo numerický atribut s dělením maximalizující zisk informace $\Delta i(v)$, resp. poměr zisku $\Delta i_r(v)$. Podle toho, jestli je atribut numerický nebo nominální, jsou vzory T_v rozděleny na dvě, resp. alespoň dvě podmnožiny. Tyto nové podmnožiny odpovídají novým uzlům. Celý proces se znova aplikuje na všechny vzniklé poduzly. To se provádí rekurzivně do té doby, až je splněna jedna z ukončovacích podmínek:

- Dělení je ukončeno, právě když je vrchol v čistý, tzn. že entropie $i(v) = 0$ nebo-li $\exists c \in C, \forall t \in T_v : \pi_2(t) = c$.
- Dělení uzlu v je ukončeno, právě když $|T_v| \leq m$, kde m je předem definovaná konstanta.

Stejně jako v případě CART stromů se volí co nejmírnější ukončovací podmínky a strom se nechá vyrůst do maximální velikosti. Poté je strom ořezán. K ořezání lze použít algoritmus REP, který byl popsán v kapitole 3.4.1, nebo algoritmus PEP popsaný v kapitole 3.4.4.

Kapitola 4

SDT stromy

Měkké rozhodovací stromy (Soft Decision Trees) [28], dále SDT stromy, spadají do oblasti fuzzy rozhodovacích stromů. SDT metoda popisuje fázi růstu a ořezávání, které určují strukturu SDT stromu. Součástí SDT je také ladění stromu. Ladění stromu je možné provést pomocí tzv. *refittingu* a *backfittingu*. Refitting a backfitting zlepšují generalizační schopnosti stromu, tzn. zlepšují přesnost klasifikace na trénovacích a testovacích datech. Cílem SDT metody bylo vytvořit klasifikační strukturu, která vykazuje lepší přesnost klasifikace než standardní rozhodovací stromy. SDT přidávají ke klasické teorii vytváření rozhodovacích stromů teorii fuzzy množin [38]. Bylo ukázáno [10], že klasické rozhodovací stromy jsou v řadě případů použitelné, efektivní, nejsou závislé na konkrétním problému a dávají uspokojivé výsledky v řadě aplikací. Nevýhodou je, že jsou velmi nestabilní vzhledem k malým odchylkám v trénovacích datech. Teorie fuzzy množin byla zavedena, aby tuto nestabilitu rozhodovacích stromů odstranila.

Na začátku kapitoly představíme SDT stromy na příkladě, který srovnává CART a SDT. V ostatních částech pak vysvětlíme jednotlivé kroky konstrukce SDT stromů. Struktura stromu je určena ve fázi růstu a ořezávání, které jsou probrány v částech 4.3 a 4.4. Metody pro ladění parametrů stromu jsou představeny v části 4.5. První ladící metoda *refitting* zlepšuje hodnoty parametrů pouze v listech stromu. *Backfitting*, který je časově náročnější, zpřesňuje parametry ve všech uzlech stromu.

4.1 Struktura SDT stromu

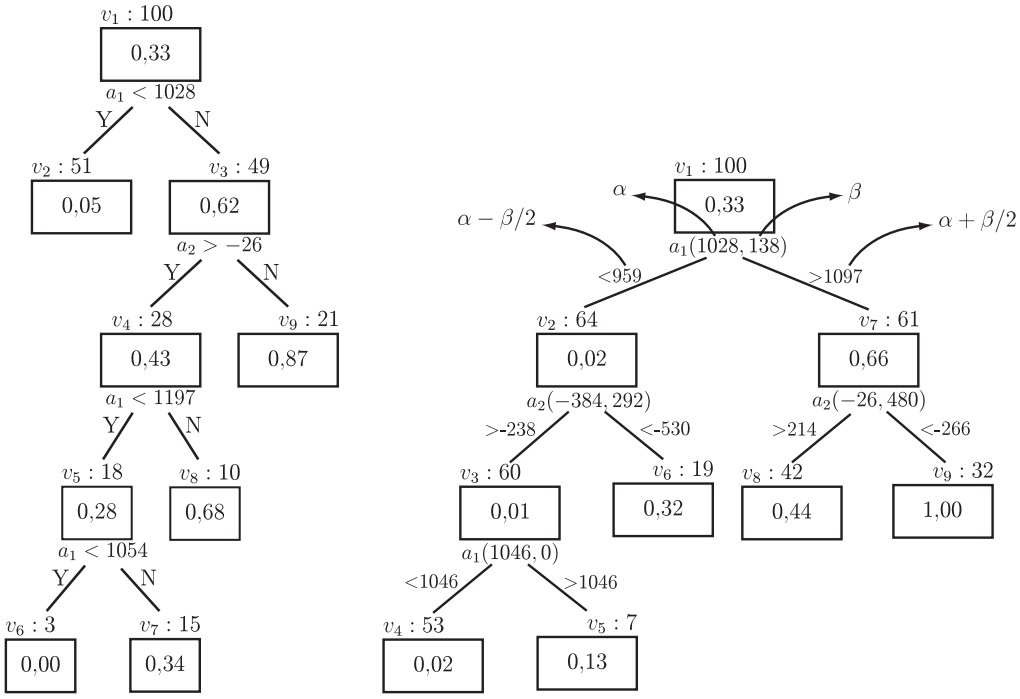
SDT stromy se liší od standardních rozhodovacích stromů zavedením fuzzy množiny.

Definice 4.1.1 *Nechť X je prostor všech vzorů. Fuzzy množina vzorů $S \subset X$ je charakterizována členskou funkcí:*

$$\mu_S : X \to \langle 0, 1 \rangle, \tag{4.1}$$

která každému vzoru $x \in X$ přiřazuje číslo $\mu_S(x)$ z intervalu $\langle 0, 1\rangle$ a představuje stupeň náležitosti vzoru x do množiny S.

SDT strom prezentujeme na ukázkovém příkladě. Předpokládejme, že máme k dispozici CART a SDT strom, jejichž struktura je znázorněna na obr. 4.1.



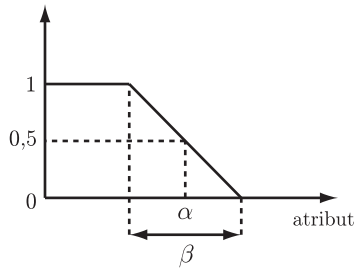
(a) CART. (b) SDT.

Obrázek 4.1: Ukázka SDT a CART stromu vybudovaných ze stejné množiny vzorů.

Oba dva stromy byly vybudovány z trénovací množiny, která je tvořena třemi numerickými atributy a_1, a_2, a_c, kde a_c je třídový atribut. Každý uzel ve stromě CART je označen hodnotou, která je (váženým) průměrem hodnot třídového atributu a_c trénovacích vzorů, které se do uzlu ν dostanou. Oba stromy predikují stupeň náležitosti k třídě C.

CART

CART strom na obr. 4.1(a) má čtyři vnitřní uzly a pět listů. V obdélnících jsou znázorněny lokální odhady tzv. *označení uzlu*. Označení uzlu v je stupeň náležitosti k třídě C v případě, že by uzel v byl listem. V každém vnitřním uzlu se provádí test, který na základě prahové hodnoty a hodnoty daného atributu vzoru pošle vzor do levé nebo pravé větve. Do levé větve je poslán vzor v případě splnění podmínky a do pravé v případě nesplnění podmínky.



Obrázek 4.2: Diskriminační funkce.

Pomocí dělení v uzlu je vstupní prostor rozdělen do dvou nepřekrývajících se podprostorů. Našim cílem je rozdělit vstupní prostor pomocí rozhodovacího stromu do takových podprostorů, aby vzory, které se v nich nacházejí, měly stejnou výstupní hodnotu. Klasifikace vzoru může být chápána jako hledání odpovídajícího podprostoru, jehož vzory mají přiřazen stejný stupeň náležitosti k třídě C. Vzor, který klasifikujeme, projde CART stromem od kořene směrem k listu právě jednou cestou. Tato cesta je určena hodnotami atributů vzoru a podmínkami v uzlech stromu.

Mějme vzor x s atributy $a_1(x) = 1100$ a $a_2(x) = -40$. Pomocí CART stromu na obr. 4.1(a) je odhadnut stupeň náležitosti ke třídě C. Vzor dorazí do listu v_9 a je mu přiřazena hodnota 0,87.

SDT

SDT strom na obr. 4.1(b) taktéž obsahuje čtyři vnitřní uzly a pět listů. Každý vrchol je označen odhadem výsledku, jakoby vrchol představoval list. Vnitřním uzlům opět odpovídají testy, které na základě hodnoty daného atributu vzoru pošlou vzor do pravé nebo levé větve nebo do obou dvou větví stromu. Test je určen dvěma parametry, na obr. 4.1(b) to jsou hodnoty v závorkách. Tyto dva parametry charakterizují funkci, která se nazývá *diskriminační funkce*, na základě které jsou vzory rozdělovány oběma poduzlům. Nejvíce používanou diskriminační funkcí je po částech lineární funkce (viz obr. 4.2). Parametry, které definují diskriminační funkci mají tento význam:

- α odpovídá hodnotě dělení ve vnitřním uzlu rozhodovacích stromů.
- β je šířka a definuje přechodovou oblast vybraného atributu v daném uzlu.

Pomocí takovéto po částech lineární diskriminační funkce je lokální prostor netřídových atributů v daném uzlu rozdělen do dvou překrývajících se podprostorů vzorů. Některé vzory putují pouze do levého následníka, některé pouze do pravého následníka a vzory, které se nachází v oblastech, které se

překrývají, jdou do obou následníků. Čím větší je přechodová oblast ve vnitřním vrcholu, tím větší je překrytí podprostorů atributů. Vzor, jehož hodnota atributu leží v přechodové oblasti je propagován skrz strom zároveň více cestami. V nejjednodušším případě půjde vzor právě jednou cestou k listu a v nejsložitějším půjde všemi cestami od kořene ke všem listům. Každý vzor nemusí patřit pouze jednomu listu, může patřit více listům s daným stupněm náležitosti. Na daný vrchol může být pohlíženo jako na fuzzy množinu. Daný vzor nakonec bude náležet více listům a výsledný odhad je dán seskupením odhadů v těchto listech.

Situaci demonstrujme na příkladě. Uvažujme vzor x, jehož hodnoty atributů a_1 a a_2 jsou stejné jako v předchozím příkladě ($a_1(x) = 1100$ a $a_2(x) = -40$). Vzor projde dvěma cestami. První cesta je tvořena vrcholy v_1, v_7, v_8 a druhá vrcholy v_1, v_7, v_9. Stupeň náležitosti tohoto vzoru k fuzzy množině vnitřního uzlu v_7 je roven 1, protože všechny vzory, které mají hodnotu atributu a_1 rovnu 1100 půjdou pouze doprava ($1100 > 1097$) viz obr. 4.1(b). Ovšem v uzlu v_7 hodnota atributu $a_2(x) = -40$ leží v přechodové oblasti - uvnitř intervalu $(-266, 214)$ a tedy vzor sestoupí do obou následníků v_8 i v_9. Do listu v_8 dorazí se stupněm náležitosti 0,53 a do listu v_9 se stupněm náležitosti 0,47. Nakonec na základě výše uvedených hodnot je spočten výsledný odhad stromu:

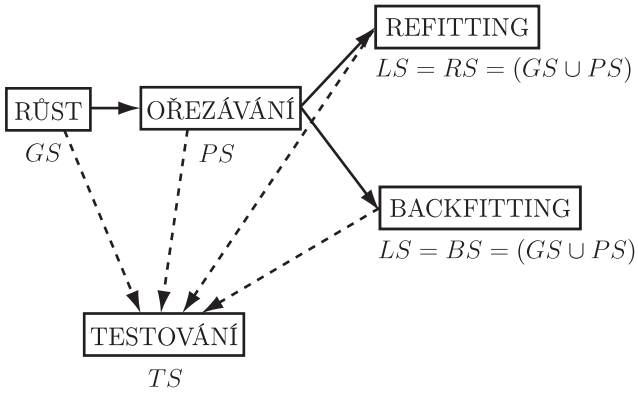
$$\begin{aligned} 1{,}0 \cdot 0{,}53 \cdot \text{ označení } v_8 + 1{,}0 \cdot 0{,}47 \cdot \text{ označení } v_9 &= \\ = 1{,}0 \cdot 0{,}53 \cdot 0{,}44 + 1{,}0 \cdot 0{,}47 \cdot 1{,}0 &= 0{,}70. \end{aligned} \tag{4.2}$$

Tedy stupeň náležitosti k třídě C je 0,70.

Na ukázkovém příkladě jsme demonstrovali základní rozdíly mezi stromy CART a stromy SDT. SDT se liší od CART tím, že zavádějí přechodovou oblast. V CART stromech je vzor propagován právě jednou cestou, v případě SDT stromů může být vzor propagován více cestami najednou. U každého vzoru v uzlu je udržován stupeň členství vzoru k fuzzy množině vzorů náležící uzlu. Pokud dojde k „rozdělení“ vzoru do dvou následníků, je stupeň členství upraven na základě diskriminační funkce. Odhad CART stromu pro daný vzor je dán označením listu, do kterého vzor náleží. Odhad SDT stromu pro daný vzor je spočten na základě stupně členství k fuzzy množinám vzorů listů a na základě označení těchto listů.

4.2 Budování SDT

Kompletní procedura budování SDT stromu je znázorněna na obrázku 4.3. Celý proces začíná *růstem* dostatečně velkého stromu ze vzorů *GS* určených pro růst. Růst stromu probíhá podobně jako v rozhodovacích stromech, které



Obrázek 4.3: Budování SDT. Množiny vzorů GS pro růst, PS pro ořezávání a TS pro testování jsou disjunktní. LS je množina vzorů pro růst a pro ořezávání, RS je množina vzorů pro refitting a BS je množina vzorů pro backfitting.

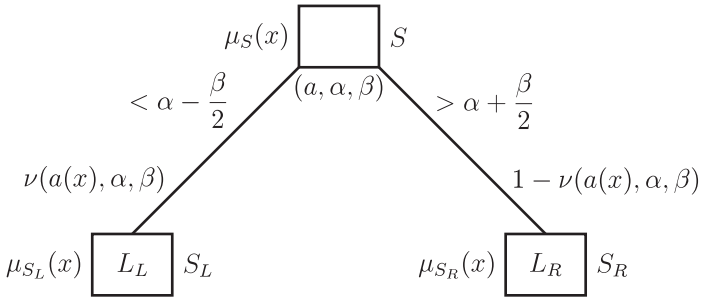
jsme popisovali v kapitolách 3.5 a 3.6. Vrcholy jsou postupně přidávány odshora dolů. Na začátku je kořen, který se rozdělí do dvou následníků a ty se dále dělí. Růst SDT stromu probíhá rekurzivně do doby, kdy je splněna ukončovací podmínka. V druhé fázi je SDT strom *ořezán*. Ořezává se odspodu nahoru pomocí ořezávací množiny vzorů *PS*. Po růstu a ořezávání je určena struktura stromu (tj. množina uzlů a příslušných hran stromu). Poslední fází je tzv. ladění parametrů stromu. Existují dvě metody - *refitting* a *backfitting*. Metody změní parametry stromu, aby se poměr chybné klasifikace na vzorech ještě více minimalizoval. Pro ladění parametrů se používá celá množina vzorů pro učení $LS = GS \cup PS$. Po každé fázi je možné strom testovat na množině testovacích vzorů TS, která je nezávislá na množině vzorů LS určené pro učení ($LS \cap TS = \emptyset$).

Původní trénovací množina je tedy rozdělena Předložené trénovací vzory jsou rozděleny do množiny vzorů pro učení LS a do testovací množiny TS, které jsou disjunktní. Množina vzorů pro učení LS je dále rozdělena na množinu vzorů GS určených pro růst a ořezávací množinu vzorů PS. Množiny GS a PS jsou složeny ze vzájemně nezávislých vzorů.

4.3 Růst SDT

Nejpodstatnější část růstu stromu SDT spočívá na metodě pro výběr dělení v každém novém uzlu, na volbě ukončovací podmínky a pravidle pro označení listu.

Na strom SDT lze pohlížet jako na funkci, která aproximuje stupeň náležitosti vzorů k třídě [28]. Vstupem této funkce jsou hodnoty atributů vzorů a výstupem je stupeň náležitosti k třídové fuzzy množině. Nechť C je třídová



Obrázek 4.4: Dělení uzlu v SDT stromu.

fuzzy množina vzorů. Třídový atribut a_c vzorů nabývá hodnot z intervalu $\langle 0, 1\rangle$. Pokud má třídový atribut vzoru hodnotu 0, pak vzor do fuzzy množiny C nepatří. Pokud je hodnota třídového atributu vzoru 1, pak vzor do fuzzy množiny C patří. Popsanou vlastnost popíšeme formálně. Pro každý vzor $x \in X$, kde X je prostor všech vzorů, platí, že $x \in C$ a vzoru x odpovídá stupeň náležitosti $\mu_C(x)$, který je buď ostrý (tj. nabývá hodnot 0 nebo 1) anebo nabývá hodnot z intervalu $\langle 0, 1\rangle$. Dále předpokládejme, že všechny hodnoty atributů jsou numerické a normalizované v intervalu $\langle 0, 1\rangle$. $\mu_C(x)$ označuje skutečnou hodnotu třídového atributu vzoru $a_c(x)$. Skutečná hodnota je používána k srovnání s odhadnutou hodnotou SDT stromem. V dalším textu zavedeme odhad stromu $\hat{\mu_C}(x)$ pro vzor x. K tomu je nutné vědět, jak probíhá dělení uzlu SDT stromu.

Na obrázku 4.4 je znázorněno dělení uzlu v SDT stromu, kterému odpovídá fuzzy množina vzorů S, do dvou fuzzy podmnožin vzorů. Podmnožina S_L odpovídá levému vrcholu a podmnožina S_R pravému vrcholu. Obě podmnožiny byly vytvořeny na základě hodnot atributu a vzorů z S a parametrů α a β. Funkce μ_S, μ_{S_L} a μ_{S_R} vyjadřují stupně náležitosti k fuzzy množinám S, S_L a S_R.

Definice 4.3.1 *Mějme vzor $x \in S$, kde S je množina vzorů odpovídajících uzlu v v* SDT *stromě. Nechť uzlu v odpovídá atribut a a parametry α, β a diskriminační funkce ν. Diskriminační funkce ν je zobrazení přiřazující hodnotě atributu $a(x)$ a parametrům α, β hodnotu z intervalu $\langle 0, 1\rangle$:*

$$\nu(a(x), \alpha, \beta) \rightarrow \langle 0, 1\rangle \tag{4.3}$$

Diskriminační funkce určuje fuzzy dělení vzorů uzlu na základě hodnoty atributu vzorů a podle hodnot parametrů α, β. V této práci budeme uvažovat po částech lineární diskriminační funkci, jejíž tvar je určen dělením α a šířkou β viz obr. 4.2. Stupeň náležitosti pro levého následníka uzlu je odvozen z původního stupně náležitosti $\mu_S(x)$ pomocí diskriminační funkce ν tímto způsobem [28]:

$$\mu_{S_L}(x) = \mu_S(x) \cdot \nu(a(x), \alpha, \beta). \tag{4.4}$$

Po částech lineární diskriminační funkce $\nu(a(x), \alpha, \beta)$ je nulová pro všechny vzory, pro které je $a(x) > \alpha+\beta/2$. Vzory $x \in S$, pro které $a(x) \leq \alpha+\beta/2$, budou přiděleny levé podmnožině S_L se stupněm náležitosti μ_{S_L}. Podobně vzory $x \in S$, pro které platí $a(x) \geq \alpha - \beta/2$, budou přiděleny pravé podmnožině S_R odpovídající pravému uzlu. Stupeň náležitosti vzoru x k fuzzy množině S_R je odvozen z $\mu_S(x)$, avšak je vážen doplňkem diskriminační funkce ν [28]:

$$\mu_{S_R}(x) = \mu_S(x) \cdot (1 - \nu(a(x), \alpha, \beta)). \tag{4.5}$$

Všechny vrcholy SDT kromě kořene jsou následníky právě jednoho rodičovského vrcholu.

Stupeň náležitosti vzoru k fuzzy podmnožině S odpovídající vrcholu u SDT je dán rekurzivně. Je funkcí hodnot atributů vzorů, parametrů definujících jednotlivé diskriminační funkce použité v předchůdcích vrcholu u a stupněm členství k fuzzy množině R kořenu. Stupeň náležitosti $\mu_R(x)$ může mít uživatelsky definované hodnoty. Nejčastěji však bývá $\mu_R(x) = 1{,}0$ pro každý vzor x, protože ve většině trénovacích množin si jsou vzory rovnocenné a mají stejnou důležitost.

Nyní zadefinujeme odhad stromu, ke kterému jsme chtěli dospět:

Definice 4.3.2 *Nechť $u_j \in L(G)$ je list* SDT *stromu G a L_j je označení listu u_j a S_{L_j} je fuzzy množina odpovídající tomuto listu. Stupeň náležitosti $\hat{\mu}_C(x)$ k třídové fuzzy množině C odhadnutý* SDT *stromem G je*

$$\hat{\mu}_C(x) = \frac{\sum_{u_j \in L(G)} \mu_{S_{L_j}}(x) L_j}{\sum_{u_j \in L(G)} \mu_{S_{L_j}}(x)}, \tag{4.6}$$

kde x je vzor z prostoru vzorů X.

Suma $\sum_{u_j \in L(G)} \mu_{S_{L_j}}(x)$ je rovna stupni náležitosti $\mu_R(x)$, kde R je fuzzy množina odpovídající kořeni SDT stromu [28]. Jak již bylo poznamenáno výše, stupeň náležitosti $\mu_R(x)$ je často roven jedné a tudíž i suma $\sum_{u_j \in L(G)} \mu_{S_{L_j}}(x)$ je jedna. V dalším budeme pro zjednodušení zápisu předpokládat tuto skutečnost.

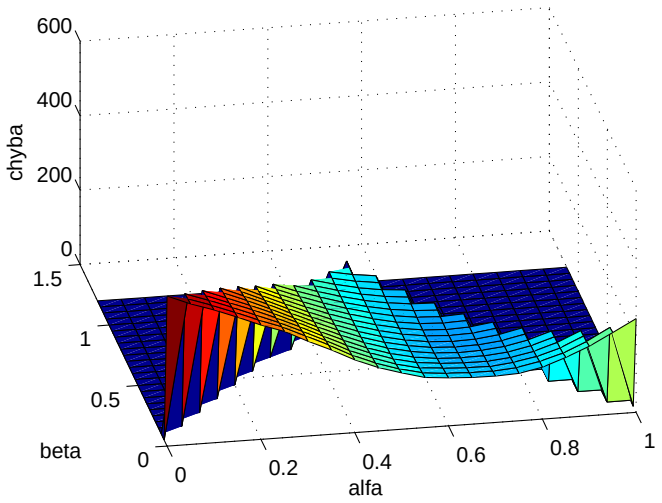
Podobně jako v jiných metodách pro vytváření rozhodovacích stromů zavedeme nečistotu uzlu u.

Definice 4.3.3 *Mějme fuzzy množinu vzorů S odpovídající vrcholu u* SDT *stromu a $x \in S$. Čtvercová chyba E_S fuzzy množiny S je definována jako:*

$$E_S = \sum_{x \in S} \mu_S(x) \cdot [\mu_C(x) - \hat{\mu}'_C(x)]^2, \tag{4.7}$$

kde

$$\hat{\mu}'_C(x) = \nu(a(x), \alpha, \beta) L_L + (1 - \nu(a(x), \alpha, \beta)) L_R. \tag{4.8}$$



Obrázek 4.5: Čtvercová chyba E_S pro $L_L = 0{,}9, L_R = 0{,}259$ a atribut a_1. Dělení α je z intervalu $\langle 0, 1\rangle$, šířka β je z intervalu $\langle 0, \min\{2\alpha, 2(1-\alpha)\}\rangle$. Uzlu SDT stromu odpovídá množina vzorů, která obsahuje tisíc vzorů.

Cílem je najít takový atribut a, dělení α a šířku β (parametry definující diskriminační funkci ν), dále pak označení L_L levého a označení L_R pravého následníka uzlu u tak, aby chyba E_S byla minimální. Zavedená chybová funkce neupřednostňuje ostrá dělení ($\beta = 0$) jako to dělají jiné chybové funkce viz [3, 21, 32, 33]. Kdyby byla preferována ostrá dělení, strom SDT by měl stejný tvar jako strom CART a dával by podobné výsledky. Tím by zavedení přechodové oblasti postrádalo smysl.

Proces hledání optimálního atributu a fuzzy dělení je rozděleno do dvou kroků. V první kroku se hledá atribut a a dělení α. Přitom položíme $\beta = 0$ (ostré dělení). Hledání probíhá tak, že se projdou všechny hodnoty všech atributů vzorů v uzlu a všechna možná dělení α těchto atributů. Najde se takový atribut a a dělení α, které minimalizují chybovou funkci E_S. Výsledkem je optimální atribut a^*, dělení α^* a provizorní označení levého následníka L_L a pravého následníka L_R, které jsou odhadnuty podobně jako u CART stromů. Detaily prvního kroku jsou popsány v kapitole 4.3.1.

V druhém kroku se hledá optimální šířka β a označení levého následníka L_L a pravého následníka L_R. β se hledá již pro optimální atribut a^* a dělení α^*, které bylo nalezeno v předchozím kroku. Pro hledání šířky β lze použít *Fibonacciho prohledávání*. Fibonacciho prohledávání je iterativní algoritmus, který v každé iteraci zpřesní hodnotu β. Pro každou novou hodnotu β jsou aktualizována označení L_L a L_R. Hledání šířky a označení je popsáno v části 4.3.2.

Fakt, že proces hledání je možné rozdělit na tyto dva kroky, vychází

z tvaru chybové funkce E_S (viz definice 4.3.3). Jak najít efektivně minimum čtvercové chyby, se můžeme dočíst v [27]. Na obr. 4.5 je chybová funkce E_S vykreslena pro parametry diskriminační funkce $\alpha \in \langle 0, 1\rangle$ a $\beta \in \langle 0, \min\{2\alpha, 2(1-\alpha)\}\rangle$. Chybová funkce je dále závislá na označení levého následníka L_L a označení pravého následníka L_R. Tyto dva parametry jsou na chybové funkci lineárně závislé a je možné pro ně zvolit konkrétní hodnoty a zafixovat je. Na obr. 5.10 je možné si všimnout, že pro libovolnou šířku β je minimum chybové funkce dosaženo pro stejnou hodnotu dělení α. To znamená, že hodnota α odpovídající minimu chybové funkce E_S pro libovolné β je obecně blízko hodnotě α odpovídající globálnímu minimu funkce [28]. Tvrzení je pravdivé i pro $\beta = 0$ [28]. Tedy optimální minimum v prvním kroku lze hledat jen pro parametr α a ostatní parametry β, L_L a L_R jsou pevné (konstantní). A podobně v druhém kroku se hledá lokální minimum pro β s tím, že α, L_L a L_R jsou pevné. Tím se efektivně najde optimum pro všechny parametry, aniž bychom použili nějakou nelineární optimalizační techniku přes všechny čtyři parametry.

Výše popsané dělení uzlů stromu SDT probíhá do té doby, až je splněna některá z ukončovacích podmínek. Růst SDT stromu by měl být ukončen v okamžiku, kdy další dělení vrcholu už nemá smysl. Vzhledem k tomu, že vytvořený strom se bude ořezávat, není volba ukončovacího kritéria kritická. Strom se nechá rozrůst i za cenu nároků na čas. Volí se proto mírnější podmínky, aby se vygeneroval dostatečně velký strom. Dělení stromu je ukončeno, právě když je splněna aspoň jedna z následujících podmínek [28]:

1. Součet náležitostí vzorů S v uzlu $\sum_{x\in S} \mu_S(x)$ je menší než zvolený práh pro minimální součet náležitostí vzorů v uzlu.

2. Chyba $\sum_{x\in S} \mu_S(x)[\mu_C(x) - L_u]^2$ ve vrcholu u je menší než zvolený práh pro chybu. L_u je označení vrcholu u.

3. Úbytek chyby

$$\frac{E_S}{\sum_{x\in S} \mu_S(x)} - \frac{E_{S_L}}{\sum_{x\in S_L} \mu_{S_L}(x)} - \frac{E_{S_R}}{\sum_{x\in S_R} \mu_{S_R}(x)} \tag{4.9}$$

   je menší než stanovená hodnota prahu pro úbytek chyby.

SDT jsou navrženy pro jednu výstupní třídu odpovídající fuzzy množině C. Výsledek pro doplňkovou třídu vzoru x se získá z odhadu stupně náležitosti $\hat{\mu}_C(x)$ tak, že se odečte od jedničky. Tedy odhad stupně náležitosti pro doplňkovou fuzzy množinu $\bar{C}$ lze spočítat jako:

$$\hat{\mu}_{\bar{C}}(x) = 1 - \hat{\mu}_C(x) \tag{4.10}$$

Na začátku kapitoly jsme v motivačním příkladě odhadli stupeň náležitosti 0,70 vzhledem ke třídě C a tedy stupeň náležitosti vzhledem k doplňkové třídě

$\bar{C}$ je 0,30. Pokud řešíme problém s více třídami, je nutné vybudovat les SDT stromů. Každý SDT strom se snaží rozpoznat jednu třídu od sjednocení všech ostatních tříd [20]. Mějme fuzzy množiny C_i, které odpovídají jednotlivým třídám. Pro každý vzor x jim odpovídají stupně náležitosti $\hat{\mu}_{C_i}(x)$ odhadnuté jednotlivými rozhodovacími stromy. Dále předpokládejme, že pokuty za chybnou klasifikaci nejsou závislé na druhu chyby. Potom fuzzy množina C^*, která odpovídá klasifikační třídě, je množina s největším stupněm náležitosti $\hat{\mu}_{C_i}(x)$ odhadnutá i-tým stromem SDT:

$$C^* = \underset{C_i}{\operatorname{argmax}}\, \hat{\mu}_{C_i}(x) \tag{4.11}$$

V obecnějším případě, ve kterém pokuty za chybnou klasifikaci nejsou rovnocenné, je nezbytné upravit odhady stupně náležitosti na odhady podmíněných pravděpodobností náležitosti vzorů k ostatním třídám. Vybrána bude taková třída, která minimalizuje očekávanou pokutu chybné klasifikace odhadnutou pro daný vzor.

4.3.1 Hledání atributu a prahu

V této kapitole bude podrobněji vysvětlen proces hledání optimálního atributu a prahu pro uzel SDT stromu. Během hledání optimálního atributu a a dělení α uvažujeme pevnou hodnotu parametru $\beta = 0$. Nejlepší dělení α se hledá přes všechny možné atributy a tak, aby byla minimalizována chyba E_S. Protože šířka $\beta = 0$, je proces totožný s hledáním optimálního atributu a dělení v CART stromech. Pro každou hodnotu dělení α se spočtou také „provizorní“ hodnoty označení následníků L_L a L_R.

Díky tomu, že parametr $\beta = 0$, jsou množiny vzorů S_L a S_R vzájemně disjunktní (viz obr. 4.4). Diskriminační funkce je binární a výstupem jsou tedy hodnoty 0 nebo 1. Chybová funkce z definice 4.3.3 může být rozepsána pomocí výrazu 4.8 takto:

$$E_S = \sum_{x \in S_L} \mu_S(x)[\mu_C(x) - L_L]^2 + \sum_{x \in S_R} \mu_S(x)[\mu_C(x) - L_R]^2 = E_{S_L} + E_{S_R} \tag{4.12}$$

Cílem hledání optimálního atributu a prahu je minimalizovat chybovou funkci E_S. Je třeba zvolit takové parametry L_L a L_R, aby chybová funkce E_S a tedy také chyby potomků E_{S_L} a E_{S_R} byly minimální. Extrémy chybové funkce se spočtou tak, že parciální derivace se položí rovny nule (tj. $\frac{\partial E_{S_L}}{\partial L_L} = 0$ a $\frac{\partial E_{S_R}}{\partial L_R} = 0$). Derivace lze rozepsat na tvar:

$$-2 \sum_{x \in S_L} \mu_S(x)[\mu_C(x) - L_L] = 0, \quad -2 \sum_{x \in S_R} \mu_S(o)[\mu_C(o) - L_R] = 0. \tag{4.13}$$

Po úpravách získáme výraz pro levé označení L_L a pravé označení L_R:

$$L_L = \frac{\sum_{x \in S_L} \mu_S(x)\mu_C(x)}{\sum_{x \in S_L} \mu_S(x)}, \quad L_R = \frac{\sum_{x \in S_R} \mu_S(x)\mu_C(x)}{\sum_{x \in S_R} \mu_S(x)}. \tag{4.14}$$

Pokud se dosadí označení následníků L_L a L_R ze vztahu 4.14 do výrazů chybových funkcí E_{S_L} a do E_{S_R}, vyjde formule pro výpočet chyb:

$$E_A = \sum_{x \in A} \mu_S(x)\mu_C^2(x) - \frac{(\sum_{x \in A} \mu_S(x)\mu_C(x))^2}{\sum_{x \in A} \mu_S(x)}, \tag{4.15}$$

kde A zastupuje S_L či S_R.

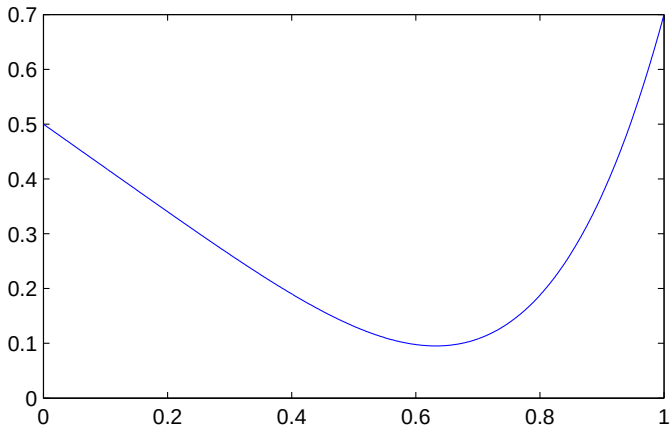
Inkrementální formule 4.15 není třeba počítat celé odznova pro každé dělení atributu. E_{S_L} a E_{S_R} se pro daný atribut spočtou pouze jednou pro první dělení. U dalších dělení se bude pomocí operací přičítání a odčítání členů sum E_{S_L} a E_{S_R} „přesunovat“ odpovídající vzory z pravé množiny S_R do levé množiny S_L.

Proces hledání optimálního atributu a^* a optimálního prahu α^* spočívá v projití všech atributů a všech jejich možných dělení. Optimum odpovídá nejmenší chybě $E_S = E_{S_L} + E_{S_R}$. Další kapitola popisuje druhý krok hledání optimálních parametrů dělení uzlu. Pro nalezený atribut a^* a práh α^* se hledá optimální šířka β^* a aktualizují se hodnoty označení L_L a L_R.

4.3.2 Hledání šířky a označení

Po prvním kroku je nalezen optimální atribut a^* a práh α^*. Následuje druhý krok a to hledání šířky β. Během hledání šířky β zůstávají parametry a^* a α^* nezměněné díky vlastnostem chybové funkce E_S. Nejlepší šířka β se získá pomocí *Fibonacciho prohledávání*.

Fibonacciho prohledávání hledá minimum funkce na daném intervalu tak, že ohodnocuje funkci v bodech Fibonacciho posloupnosti. Tím se lze vyhnout netriviálnímu derivování chybové funkce podle β. Fibonacciho prohledávání končí ve chvíli, kdy je dosažen předem definovaný počet ohodnocení. Funkce, na které se hledá minimum, musí být unimodální. Unimodální funkce je klesající mezi počátečním bodem intervalu a bodem, ve kterém funkce nabývá minimum, a rostoucí mezi minimem a koncovým bodem intervalu. Příklad unimodální funkce je na obr. 4.6. Autor článku [28] předpokládá, že pro chybovou funkci E_S je tento předpoklad splněn. Fibonacciho prohledávání se spustí na interval $\langle 0, \min\{2\alpha, 2(1-\alpha)\}\rangle$. Po první iteraci Fibonacciho algoritmu se interval zúží a za β se dosadí jeden z možných koncových bodů nového intervalu příští iterace. Pro tuto hodnotu se aktualizují hodnoty označení L_L a L_R. Toto se provádí do té doby, než Fibonacciho algoritmus dosáhne maximálního počtu iterací zadaného uživatelem. V dalším textu popíšeme, jakým způsobem se aktualizují označení levého a pravého následníka L_L a L_R.



Obrázek 4.6: Příklad unimodální funkce $f(x) = \frac{1}{2} + x^5 - \frac{4}{5}x$ na intervalu $\langle 0, 1 \rangle$.

Pro každou novou hodnotu β získanou v iteraci Fibonacciho algoritmu jsou aktualizovány hodnoty následníků L_L a L_R pomocí lineárních regresních formulí. Cílem je minimalizace chybové funkce E_S. K nalezení minima chybové funkce E_S se využijí parciální derivace:

$$\frac{\partial E_S}{\partial L_L} = 0 \quad \text{a} \quad \frac{\partial E_S}{\partial L_R} = 0.$$

V souladu se vzorcem 4.7 jsou parciální derivace rozepsány takto:

$$-2\sum_{x \in S} \mu_S(x)\nu(a(x))\{\mu_C(x) - [\nu(a(x))L_L + (1 - \nu(a(x)))L_R]\} = 0,$$
$$-2\sum_{x \in S} \mu_S(x)(1 - \nu(a(x)))\{\mu_C(x) - [\nu(a(x))L_L + (1 - \nu(a(x)))L_R]\} = 0.$$

Z výše uvedených výrazů se vyjádří hodnoty pro levého a pravého následníka L_L a L_R:

$$L_L = \frac{cd - eb}{b^2 - ac}, \quad L_R = \frac{ae - bd}{b^2 - ac}, \tag{4.16}$$

kde členy $a, \ldots, e$ jsou dány vztahy:

$$\begin{aligned} a &= \sum_{x \in S} \mu_S(x)\nu(a(x))^2, \quad b = \sum_{x \in S} \mu_S(x)\nu(a(x))(1 - \nu(a(x))), \\ c &= \sum_{x \in S} \mu_S(x)(1 - \nu(a(x)))^2, \quad d = -\sum_{x \in S} \mu_S(x)\mu_C(x)\nu(a(x)), \\ e &= -\sum_{x \in S} \mu_S(x)\mu_C(x)(1 - \nu(a(x))). \end{aligned}$$

Hodnoty následníků L_L a L_R bylo možné také určit přímo z výrazů 4.14. Ovšem předem definované výrazy pro následníky vedou k suboptimálnímu výsledku [28].

4.4 Ořezávání

Ořezávání je standardní technikou při budování rozhodovacích stromů. Cílem je poskytnout kompromis mezi jednoduchostí a přesností predikce rozhodovacího stromu. Během ořezávání se odstraní jeho nadbytečné části. Ořezávání je upřednostňováno před volbou přísné ukončovací podmínky. Ukončovací podmínka je citlivá na odchylky v řešeném problému, což vede k vytvoření stromu náchylného na malou změnu v datech nebo k přetrénování stromu.

Dříve než bude popsán proces ořezávání, zavedeme pojem obecný podstrom:

Definice 4.4.1 *Obecný podstrom G_S stromu G je takový strom, který vznikne ze stromu G tak, že jeden nebo více jeho vnitřních vrcholů $u_1, u_2, \ldots$ se prohlásí za listy a podstromy $G|_{u_i}$ se odstraní kromě jejich kořenových vrcholů u_i. Relaci G_S je podstromem G označíme $G_S \preceq G$.*

Uvažujme úplný strom SDT a ořezávací množinu vzorů PS. Cílem ořezávání je najít takový obecný podstrom G_S SDT stromu G s nejmenší střední absolutní chybou MAE na ořezávací množině dat PS. Tedy hledá se takový podstrom G_S

$$\underset{G_S \preceq G}{\operatorname{argmin}}\{\mathrm{MAE}_{PS}\}, \tag{4.17}$$

kde

$$\mathrm{MAE}_{PS} = \frac{1}{|PS|} \cdot \sum_{x \in PS} |\mu_C(x) - \hat{\mu}_C(x)|. \tag{4.18}$$

Celkový počet obecných podstromů SDT roste exponenciálně se složitostí stromu [28]. Nalezení nejlepšího podstromu znamená projít všechny takové podstromy. V praktických aplikacích je však takovýto průchod nemožný. Uvažují se proto jen sekvence vnořených podstromů $G|^u$ (viz definice 2.2.4). Počet vnořených podstromů je omezen počtem vnitřních uzlů neořezaného SDT stromu a roste lineárně se složitostí SDT stromu [28]. Procedura ořezávání SDT se provádí ve třech krocích.

Krok 1 - Uspořádání testovacích uzlů

Na začátku se vnitřní uzly neořezaného stromu SDT vzestupně uspořádají podle chyby E_S každého uzlu u, kterému odpovídá fuzzy množina vzorů S. Nechť vrchol u má označení L. Chyba vrcholu je dána:

$$E_S = \sum_{x \in S} \mu_S(x)[\mu_C(x) - L]^2. \tag{4.19}$$

Chyby E_S jsou spočteny již ve fázi růstu stromu z množiny vzorů GS určených pro růst.

Z posloupnosti se odstraní uzly, které mají v tomto uspořádání před sebou svého předchůdce (uzel na cestě od vrcholu ke kořeni). Tím se odstraní všechny vrcholy, které mají předchůdce s menší chybou. V případě, že dojde k odstranění nějakého vnitřního uzlu u, je nutné z tohoto uspořádání také odstranit všechny vrcholy podstromu $G|_u$. Platí, že uspořádání bude vždy obsahovat kořenový uzel původního neořezaného SDT stromu, protože kořen SDT stromu nemá žádného předchůdce.

Krok 2 - Generování posloupnosti podstromů

Potom, co je vytvořeno uspořádání vrcholů, přichází na řadu generování posloupnosti podstromů. V generované sekvenci podstromů bude tolik podstromů, kolik je uzlů v uspořádané posloupnosti uzlů.

Postupně se odstraňují ze stromu vrcholy podle uspořádané posloupnosti uzlů. Uzel je ze stromu odstraněn a vzniklý strom je uložen do posloupnosti stromů. Na konci bude posloupnost obsahovat stromy se snižující se složitostí. Na začátku posloupnosti stromů bude původní vybudovaný strom SDT a poslední strom posloupnosti bude obsahovat pouze kořenový uzel původního SDT stromu. Pro každý nový strom posloupnost se spočte jeho střední absolutní chyba MAE na ořezávací množině vzorů PS.

Chyba MAE se spočte pouze jednou pro neořezaný SDT strom. Pro další podstromy posloupnost není nutné znova propagovat stromem všechny ořezávací vzory. Hodnotu MAE spočtenou pro předchozí strom posloupnosti stačí aktualizovat pomocí tzv. *inkrementálních formulí*, které ukážeme později. Díky tomu je výpočetní složitost ořezávacího algoritmu v podstatě lineární vzhledem ke složitosti počátečního stromu posloupnosti [28].

Na začátku generování vnořených podstromů jsou některé věci už předpočítány. Vzory ořezávací množiny PS se propagují přes původní strom a pro každý vzor $x \in PS$ a každý uzel u, kterému odpovídá fuzzy množina S, je spočítán stupeň náležitosti $\mu_S(x)$ a uloží se hodnota součinu $\mu_S(x)L_S$. Dále se udržuje odhad stromu $\hat{\mu}_C(x)$ všech vzorů $x \in PS$, který je součtem všech takovýchto součinů v listových uzlech (viz vzorec 4.6). V dalším odstavci vysvětlíme výpočet MAE pro další strom v posloupnosti pomocí inkrementálních formulí.

Uvažujme, že uzel u je odstraňovaný uzel z posloupnosti uzlů (uzel u bude v novém stromě listem). Nechť S_{L_j} je fuzzy množina vzorů odpovídající listům, kteří jsou následníci uzlu u. L_j jsou označení těchto listů. Odhad nového stromu $\hat{\mu}_C(x)$ je aktualizován pro všechna označení L_j a všechny

vzory $x \in S_{L_j}$ tak, že od původního odhadu $\hat{\mu}_C(x)$ se odečte $\mu_{S_{L_j}}(x)L_j$. $\mu_{S_{L_j}}(x)L_j$ je část odhadu stromu odpovídající listům, které jsou následníky vrcholu u. Tyto listy v novém stromě již nebudou. K odhadu se dále přičte $\mu_S(x)L$, kde S je fuzzy množina vzorů odpovídající u a L je jeho označení:

1. Pro $\forall L_j$ a pro $\forall x \in S_{L_j}$ $\hat{\mu}_C(x) = \hat{\mu}_C(x) - \mu_{S_{L_j}}(x)L_j$
2. Pro $\forall x \in S$ $\hat{\mu}_C(x) = \hat{\mu}_C(x) + \mu_S(x)L$.

Díky tomuto triku je druhý krok algoritmu ořezávání, který je v procesu ořezávání časově nejnáročnější, lineární vzhledem k počtu vnitřních uzlů. Druhý krok končí po odstranění všech vrcholů v posloupnosti.

Krok 3 - Výběr nejlepšího podstromu

Pro výběr stromu se používá pravidlo b-SE (viz definice 3.4.19) [2, 35]. Z předchozího kroku ořezávání je již pro každý podstrom posloupnosti spočtena chyba MAE. Nechť $G_1, G_2, \ldots, G_n$ jsou stromy v této posloupnosti a $\mathrm{MAE}(G_i)$ označuje chybu stromu G_i $(1 \leq i \leq n)$. Nechť strom $G_{i'}$ je strom s nejmenší MAE. Standardní chyba SE je podle definice 3.4.18 dána vztahem:

$$\mathrm{SE}(\mathrm{MAE}(G_{i'})) = \sqrt{\frac{\mathrm{MAE}(G_{i'}) \cdot (1 - \mathrm{MAE}(G_{i'}))}{|PS|}}. \tag{4.20}$$

Podle pravidla b-SE se jako optimální strom G^* vybere strom G_i s největší možnou MAE, pro kterou platí:

$$\mathrm{MAE}(G_i) \leq \mathrm{MAE}(G_{i'}) + b \cdot \mathrm{SE}(\mathrm{MAE}(G_{i'})), \tag{4.21}$$

kde b je reálné číslo, nejčastěji je $b = 1$ (1-SE pravidlo). Čím je množina ořezávacích vzorů PS větší, tím je člen $b \cdot \mathrm{SE}(\mathrm{MAE}(G_{i'}))$ blíž k nule a tedy se vybírá strom s nižší chybou MAE [2]. Pokud je PS malá, je výsledkem pravidla b-SE více ořezaný strom [2].

4.5 Ladění parametrů stromu

Po ořezávací fázi je určena struktura stromu. Ladění parametrů má za úkol zlepšit obecnost finálního stromu. Během vytváření struktury stromu jsou parametry určeny pouze lokálně a jsou založeny na informaci získané ze vzorů určených pro růst. V dalším textu popíšeme dvě metody pro globální optimalizaci SDT stromu. *Refitting* optimalizuje parametry pouze v listech stromu [28]. Backfitting je založen na *Levenberg-Marquardtově nelineární optimalizační technice* [29]. Backfitting optimalizuje všechny parametry SDT stromu. Jeho nevýhodou je velká časová náročnost.

4.5.1 Refitting

Refitting optimalizuje pouze parametry L_j v listech SDT stromu. Refitting hledá optimální L_j pomocí metody nejmenších čtverců pro soustavu lineárních rovnic. Úkolem metody nejmenších čtverců je najít vektor q takový, aby velikost vektoru

$$e = Mq - y \tag{4.22}$$

byla minimální. V následujícím textu zavedeme matici M, vektor q a vektor y.

Nechť $q = (q_1, q_2, \ldots, q_{K+1})$, jehož složky $q_j = L_j$ $(1 \leq j \leq K+1)$ jsou označení listů, kde K představuje počet vnitřních uzlů stromu. Platí, že počet listových uzlů v binárním stromu je dán počtem vnitřních uzlů plus jedna, protože počet vnitřních uzlů je $1 + 2^1 + 2^2 + \ldots + 2^{n-1} = \frac{1-2^n}{1-2} = 2^n - 1$ (kde $n+1$ označuje hloubku stromu) a počet listů je 2^n. Odhad stromu pro všechny vzory tvoří vektor $\hat{y}$, jehož složky jsou $\hat{y}_i = \hat{\mu}_C(x_i)$, a vektor $y = (y_1, y_2, \ldots, y_{K+1})$, jehož složky označují stupeň náležitosti objektu x_i k fuzzy množině C, tedy $y_i = \mu_C(x_i)$. Mějme matici M, jejíž členy $M_{ij} = \mu_{S_{L_j}}(x_i)$ označují stupeň náležitosti vzoru x_i k fuzzy množině j-tého listu S_{L_j}, kde $i = 1 \ldots |RS|$ a $j = 1 \ldots K+1$. RS představuje množinu vzorů určených pro refitting.

K dispozici máme vybudovaný ořezaný strom SDT a množinu vzorů RS pro refitting. Cílem je najít optimální vektor parametrů q^* takový, aby hodnota chybové funkce $\|y - \hat{y}\|^2$ byla minimální. Formálně zapsáno:

$$q^* = \underset{q}{\operatorname{argmin}} \|y - Mq\|, \tag{4.23}$$

protože vektor $\hat{y}$ lze podle vztahu 4.6 zapsat jako Mq. Pomocí maticové inverze dostaneme výraz pro optimální q^* [17]:

$$q^* = (M^T M)^{-1} M^T y. \tag{4.24}$$

Refitting upravuje pouze označení listových uzlů. I když jsou označení listů stromu spočítána globálně, není strom stále optimální, protože nejsou vyladěny všechny parametry stromu. Výpočet je rychlý, protože v této fázi se používají vzory RS, které vzniknou sjednocením vzorů použitých pro růst a ořezávání ($RS = GS \cup PS$). Pro ně jsou již stupně náležitosti ve vnitřních uzlech spočítány ve fázi růstu a ořezávání. Tedy matici M není třeba počítat od začátku. $M^T M$ je $(K+1) \times (K+1)$ rozměrná matice. Celkově je složitost výpočtu kubická vzhledem ke složitosti ořezaného stromu a lineární vzhledem k počtu vzorů $|RS|$ [28]. Nejvíce výpočetního času zabere součin matic $M^T \cdot M$, jehož složitost je kvadratická vzhledem ke složitosti stromu [28].

4.5.2 Backfitting

Backfitting hledá optimální parametry celého SDT stromu pomocí Levenberg-Marquardtovy nelineární optimalizace (dále LM). LM je iterační algoritmus, který se v každém kroku iterace posune v prostoru parametrů tak, aby byla snížena chyba stromu na vzorech BS. LM vyžaduje parciální derivace podle parametrů stromu ve všech vzorech $x \in BS$ pro výpočet Hessiánské matice, kterou používá LM. V této kapitole popíšeme výpočet parciálních derivací podle parametrů stromu a Levenberg-Marquardtův algoritmus.

Mějme vybudovaný strom SDT a množinu vzorů $BS = GS \cup PS$ určenou pro backfitting. Cílem metody [28] je nalezení optimálních parametrů q^* tak, aby byla minimalizována chyba:

$$E(q) = \sum_{x \in BS} [\mu_C(x) - \hat{\mu}_C(x, q)]^2, \tag{4.25}$$

kde q označuje vektor všech $3K + 1$ volných parametrů stromu SDT:

1. parametry definující fuzzy dělení v každém testovacím uzlu dělení α_i a šířky β_i, $i = 1, \ldots, K$.

2. označení listů L_j , $j = 1, \ldots, K + 1$.

K označuje počet vnitřních uzlů stromu SDT.

Podle rovnice 4.6 je $\hat{\mu}_C$ lineární vzhledem k označení L_j a nelineární v parametrech vnitřních vrcholů α_i a β_i. Funkce $\hat{\mu}_C$ je spojitá a diferencovatelná podle všech těchto parametrů skoro všude díky tvaru diskriminačních funkcí (po částech lineární funkce). Metoda backfitting využívá Levenberg-Marquardtovu nelineární optimalizační techniku, která je považována za standardní minimalizační techniku. LM minimalizuje nelineární funkci pomocí metody nejmenších čtverců [29]. LM vyžaduje výpočet gradientů podle parametrů minimalizované funkce. Požadované gradienty se získají pomocí algoritmu zpětného šíření.

Předpokládejme, že se algoritmus LM dostal do bodu q_0 v prostoru parametrů a následně se posune do bodu $q_0 + \delta q$, kde δq představuje krok iterace, kterým se algoritmus pohybuje v prostoru parametrů. Jestliže je tento krok posunutí dostatečně malý, je možné rozvinout gradient chybové funkce do Taylorova rozvoje obsahujícího pouze první dva členy [28]:

$$\nabla_q E(q_0 + \delta q) = \nabla_q E(q_0) + \nabla_q^2 E(q_0)\delta q. \tag{4.26}$$

$\nabla_q E(q_0 + \delta q) = 0$, jestliže $q_0 + \delta q$ je bod, který odpovídá minimu chybové funkce E. Rovnici 4.26 upravíme takto:

$$\mathbf{A}\delta q = \mathbf{B}, \tag{4.27}$$

kde

$$\mathbf{A} = \frac{1}{2}\nabla_q^2 E(q_0) \quad \text{a} \quad \mathbf{B} = -\frac{1}{2}\nabla_q E(q_0).$$

K výpočtu optimální velikosti kroku δq se používá metoda *největšího spádu* (Steepest Descent), která umožňuje adaptaci kroku iterace δq podle gradientu:

$$\delta q = konstanta * \mathbf{B}. \tag{4.28}$$

Levenberg-Marquardtův algoritmus [29] přidává faktor λ, který určuje velikost kroku a mění pouze prvky na diagonále matice $\mathbf{A}$.

$$A_{pp} = A_{pp}(1 + \lambda). \tag{4.29}$$

Matice $\mathbf{A}$ a $\mathbf{B}$ mají tvar:

$$A_{pl} = \sum_{x \in BS} \frac{\partial \hat{\mu}_C(x)}{\partial q_p} \frac{\partial \hat{\mu}_C(x)}{\partial q_l}, \quad p, l = 1, \ldots, 3K + 1, \tag{4.30}$$

$$B_p = \sum_{x \in BS} [\mu_C(x) - \hat{\mu}_C(x)] \frac{\partial \hat{\mu}_C(x)}{\partial q_p}, \quad p = 1, \ldots, 3K + 1. \tag{4.31}$$

Je třeba tedy spočítat pouze první derivace. Druhé derivace v Levenberg-Marquardtově metodě se „vyruší“ díky součtu přes všechny vzory BS. Konkrétně je třeba spočítat derivace:

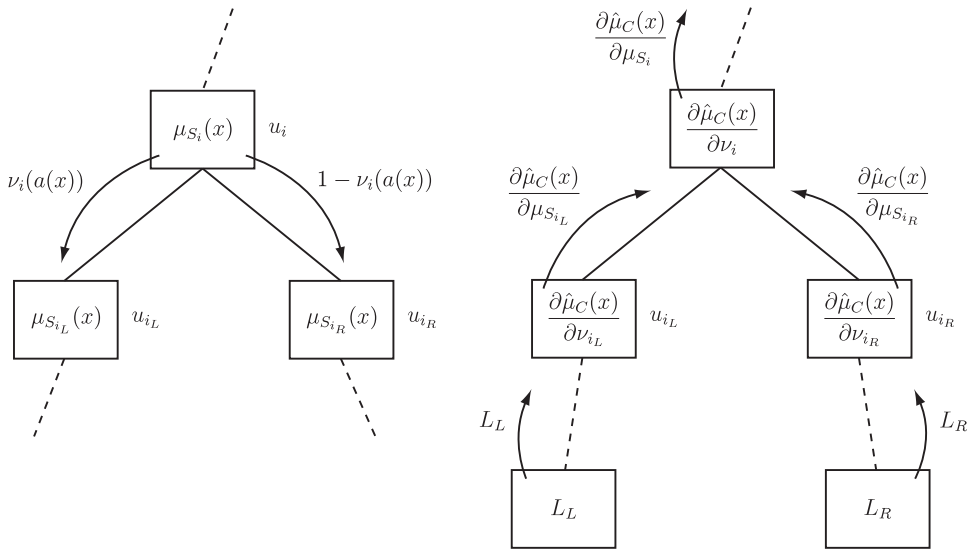
$$\frac{\partial \hat{\mu}_C(x)}{\partial L_j} = \mu_{S_{L_j}}(x), \quad j = 1, \ldots, K + 1, \tag{4.32}$$

$$\frac{\partial \hat{\mu}_C(x)}{\partial \alpha_i} = f_1\left(\frac{\partial \hat{\mu}_C(x)}{\partial \nu_i}\right), \quad i = 1, \ldots, K, \tag{4.33}$$

$$\frac{\partial \hat{\mu}_C(x)}{\partial \beta_i} = f_2\left(\frac{\partial \hat{\mu}_C(x)}{\partial \nu_i}\right), \quad i = 1, \ldots, K. \tag{4.34}$$

Odvození funkcí f_1 a f_2, nebo-li výpočet derivací 4.33 a 4.34 jsme odvodili v kapitole 5.7.1.

Algoritmus zpětného šíření pro výpočet parciálních derivací $\partial \hat{\mu}_C(x)/\partial \nu_i, i = 1 \ldots K$ probíhá ve dvou fázích (viz obr. 4.7). V dopředné fázi se postupuje směrem od kořene k listům. Pro každý uzel u_i se počítá diskriminační funkce $\nu_i(a(x))$ na základě aktuálních parametrů q. Hodnota diskriminační funkce je poslána oběma následníkům u_{i_L} a u_{i_R} a spočte se stupeň náležitosti $\mu_{S_{i_L}} = \nu_i(a(x))\mu_{S_i}(x)$ v levém následníkovi u_{i_L} a stupeň náležitosti $\mu_{S_{i_R}} = [1 - \nu_i(a(x))]\mu_{S_i}(x)$ v pravém následníkovi u_{i_R}. V druhé - zpětné fázi začne výpočet od listů a jsou posílána jejich označení



(a) Dopředná fáze. (b) Zpětná fáze.

Obrázek 4.7: Propagace parciálních derivací do vrcholů SDT stromu.

L_L a L_R. Každý vnitřní vrchol u_i obdrží derivaci $\partial\hat{\mu}_C(x)/\partial\mu_{S_{i_L}}$ z jeho levého následníka u_{i_L}. Derivace $\partial\hat{\mu}_C(x)/\partial\mu_{S_{i_L}}$ je rovna označení L_L v případě, že levý následník je list. Podobně obdrží derivaci $\partial\hat{\mu}_C(x)/\partial\mu_{S_{i_R}}$ z jeho pravého následníka u_{i_R}. Derivace $\partial\hat{\mu}_C(x)/\partial\mu_{S_{i_R}}$ je rovna označení L_R v případě, že pravý následník je list. U každého vrcholu u_i jsou udržovány hodnoty $\mu_{S_i}(x)$ a $\nu_i(a(x))$ spočtené v dopředné fázi. Na základě těchto informací lze spočíst parciální derivace pro uzel u_i [28]:

$$\frac{\partial\hat{\mu}_C(x)}{\partial\nu_i} = \mu_{S_i}(x)\left[L_{i_L} - L_{i_R}\right] = \mu_{S_i}(x)\left[\frac{\partial\hat{\mu}_C(x)}{\partial\mu_{S_{i_L}}} - \frac{\partial\hat{\mu}_C(x)}{\partial\mu_{S_{i_R}}}\right], \tag{4.35}$$

$$\begin{aligned}\frac{\partial\hat{\mu}_C(x)}{\partial\mu_{S_i}} &= \nu_i(a(x))L_{i_L} + [1-\nu_i(a(x))]L_{i_R} = \\ &= \nu_i(a(x))\frac{\partial\hat{\mu}_C(x)}{\partial\mu_{S_{i_L}}} + [1-\nu_i(a(x))]\frac{\partial\hat{\mu}_C(x)}{\partial\mu_{S_{i_R}}}.\end{aligned} \tag{4.36}$$

Touto cestou obdržíme požadované parciální derivace v časové složitosti $O(|BS|\cdot K)$, kde K je velikost stromu [28]. Představené formule lze použít i na stromy, které nejsou binární a nejsou závislé na tvaru diskriminační funkce [28].

Backfitting je iterativní algoritmus. Počáteční hodnoty parametrů stromu jsou k dispozici po fázi růstu a ořezávání. V každé iteraci jsou zlepšeny.

Algoritmus končí ve chvíli, kdy chybová funkce již výrazně neklesá. Pro tuto skutečnost zavedeme tzv. *úbytek chyby* pro sousední iterace:

$$\Delta E = E(q) - E(q + \delta q) \tag{4.37}$$

V případě, že $\Delta E < \Delta E_{\min}$, algoritmus končí. $\Delta E_{\min}$ je uživatelsky definovaná konstanta. Může se stát, že ΔE nezkonverguje k danému prahu $\Delta E_{\min}$. V tomto případě je algoritmus ukončen po dosažení maximálního počtu iterací. Pokud je hodnota $\Delta E_{\min}$ záporná, algoritmus nesmí být ukončen. Tento jev totiž signalizuje, že λ ještě nebylo optimálně nastaveno. Algoritmus backfitting je formálně popsán v algoritmu 4.1.

Optimalizační algoritmus 4.1 tedy hledá nejlepší hodnoty volných parametrů q^*, pro které je $E(q^*) = \min E(q)$, na množině vzorů BS. Počáteční parametry q jsou získány z vybudovaného a ořezaného stromu SDT. Na začátku algoritmu je nutné zvolit prahový úbytek chyby $\Delta E_{\min}$, počáteční hodnotu faktoru λ_{init} a krok pro úpravu faktoru λ_{step}.

1: **procedure** Backfitting(λ_{init}, λ_{step}, $\Delta E_{\min}$)
2: $\lambda \leftarrow \lambda_{\text{init}}$.
3: **Dopředná fáze:**
4: - Spočtení $E(q)$ pomocí vztahu 4.25.
5: **Zpětná fáze:**
6: - Spočtení prvních derivací $\partial\hat{\mu}_C(x,q)/\partial q_p$ pomocí výrazů 4.32,
7: 4.33 a 4.34.
8: - Nastavení matic **A** a **B** podle výrazů 4.30 a 4.31.
9: Úprava prvků matice **A** podle 4.29.
10: Vyřešení soustavy $\mathbf{A} \cdot \delta q = \mathbf{B}$ pomocí Gaussovy eliminace.
11: **Dopředná fáze:**
12: - Spočtení chyby $E(q + \delta q)$ pomocí výrazu 4.25.
13: Spočtení úbytku chyby $\Delta E(q) = E(q) - E(q + \delta q)$
14: **if** $0 \leq \Delta E(q)$ **and** $\Delta E(q) \leq \Delta E_{\min}$ **then**
15: $q^* \leftarrow q + \delta q$
16: Pokračovat krokem **27**.
17: **end if**
18: **if** $\Delta E(q) < 0$ **then**
19: $\lambda \leftarrow \lambda \cdot \lambda_{\text{step}}$
20: Pokračovat krokem **9**.
21: **end if**
22: **if** $\Delta E(q) > \Delta E_{\min}$ **then**
23: $\lambda \leftarrow \lambda / \lambda_{\text{step}}$
24: $q \leftarrow q + \delta q$
25: Pokračovat krokem **5**.
26: **end if**
27: Aktualizovat parametry stromu SDT.
28: **end procedure**

Algoritmus 4.1: Backfitting. Pro jednoduchost bylo vynecháno počítání iterací a podmínky pro ukončení algoritmu po dosaženém počtu iterací.

Kapitola 5

Experimenty

V této kapitole se budeme zabývat zkoumáním jednotlivých modelů stromů a jejich srovnáním na různých datových množinách vzorů. Teoretický základ testovaných metod je možné najít v kapitolách 2 až 4. Část našich pozorování a výsledků získáme experimentálně. Součástí práce jsou také naše tvrzení, která dokážeme. Při testování jednotlivých modelů se zaměříme na přesnost a velikost vybudovaných stromů. Na závěr kapitoly jsou srovnáme všechny metody podle velikosti vybudovaných stromů, podle přesnosti klasifikace a také času nutného k vybudování jednotlivých stromů.

Kapitola 5.1 popisuje námi použité množiny vzorů a jejich předzpracování. Je možné se zde dovědět i o metodách, které použijeme k testování jednotlivých klasifikátorů.

CART stromy jsou zkoumány v kapitole 5.2. Cílem našeho prvního testu je srovnat Giniho nečistotu a twoing kritérium. V teoretické části jsme se zaměřili na dva typy ořezávání CART stromů. Ořezávání MCCP vygeneruje posloupnost ořezaných stromů a z nich vybere ten nejlepší na základě velikosti a poměru chybné klasifikace na neznámých datech. Cílem našich zkoumání je zjistit, které stromy z posloupnosti ořezaných stromů algoritmus MCCP vybere. Druhým z testovaných ořezávacích algoritmů stromu CART je MEP. Ořezávání metodou MEP je ovlivněno parametrem m pro m-odhad. My se v našich testech pokusíme zjistit, jaký vliv má parametr m na ořezání stromu a také na jeho přesnost (poměr počtu chybně klasifikovaných neznámých vzorů a počtu všech neznámých vzorů).

Výsledky našich experimentů na stromech C4.5 jsou diskutovány v kapitole 5.3. Základním algoritmem pro ořezávání stromu C4.5 je jednoduché ořezávání REP. Naším záměrem v prvním testu je srovnat přesnost neořezaného stromu C4.5 a ořezaného stromu metodou REP v závislosti na velikosti množiny vzorů GS určené pro růst stromu. Další metodou pro ořezávání stromu C4.5 je ořezávání PEP. Míru ořezávání je možné řídit pomocí parametru věrohodnosti c. My se pokusíme vybudovat stromy pro různé hodnoty parametru c a různé velikosti množin vzorů a na základě výsledků zjistit,

jaký vliv má parametr c na velikost a přesnost stromu. Na závěr kapitoly 5.3 porovnáme obě ořezávací metody REP a PEP. Budeme se snažit zjistit, pro které situace je daná ořezávací metoda vhodná.

V kapitolách 5.4 až 5.7 podrobně analyzujeme SDT stromy. Autoři SDT stromů předpokládají unimodalitu chybové funkce E_S uzlu v závislosti na parametru β pro pevné hodnoty parametrů α, L_L a L_R. My se pokusíme na podobných datech ověřit, jestli předpoklady autorů byly správné. Opět jako v předchozích pokusech bude našim cílem zjistit, jaký má ořezávání vliv na SDT stromy. Refitting by měl zlepšit přesnost SDT stromu. Pokusíme se zjistit, o kolik se zmenší míra chybné klasifikace. Backfitting je druhá ladící metoda, která optimalizuje všechny parametry stromu. Backfitting ovšem nelze obecně použít na všechny SDT stromy, což se nám podařilo dokázat. Analýza metody backfitting je hlavní náplní kapitoly 5.7.

V závěrečné kapitole našich experimentů srovnáme jednotlivé metody pro budování stromu. Vybudované stromy opět srovnáme podle různých kritérií. Součástí výsledků jednotlivých testů jsou také naše zdůvodnění a závěry.

5.1 Testovací data

K testování jsme použili databázi vzorů, které vyjadřují problém bezpečnosti elektrické napájecí sítě (OMIB database), přímo od jednoho z autorů SDT stromů. Datová množina je volně ke stažení na adrese:

- `http://www.montefiore.ulg.ac.be/~lwh/Gtdidt/#dbs`.

Na podobné množině vzorů autoři článku [28] prováděli experimenty s SDT stromy. Část dat je znázorněna na obr. 5.1. OMIB databáze obsahuje 1000 vzorů. První dva numerické atributy charakterizuji stav napájecí sítě. Atributy jsou pojmenovány 'pu' a 'qu'. Třetí atribut je třídový a nabývá hodnot '0' a '1'. '0' znamená, že napájecí síť je nestálá. Označení '1' symbolizuje, že napájecí síť je stálá. 302 vzorů je označených třídou '0' a 698 vzorům odpovídá třída '1'. Na obr. 5.2 jsou vzory databáze OMIB zobrazeny ve vstupním příznakovém prostoru. Vodorovná osa znázorňuje hodnoty atributu 'pu' a svislá osa hodnoty parametru 'qu'. Vzory označené křížkem spadají do třídy '1' a vzory označené kroužkem do třídy '0'. Z obrázku 5.2 je patrné, že jednotlivé shluky vzorů jsou dobře odděleny. Je možné je rozdělit lomenou čárou, která je složená z úseček rovnoběžných se souřadnicovými osami 'pu' a 'qu'. Na takových datech je poměrně snadné vybudovat dostatečně přesný klasifikátor.

Další množina vzorů, kterou jsme využili v experimentech, popisovala písmena anglické abecedy A - Z (26 písmen). Je volně dostupná na ftp serveru:

- `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/`

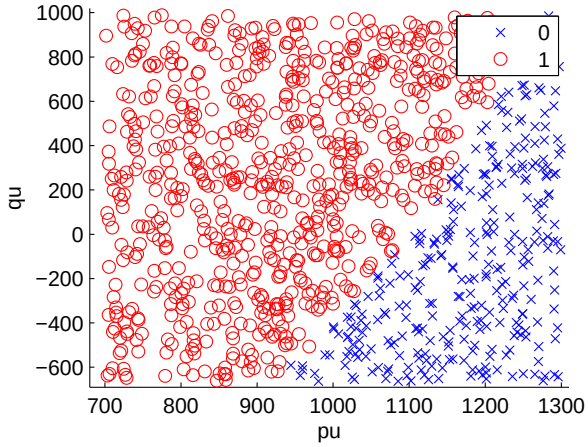
```
@relation omib

@attribute pu numeric
@attribute qu numeric
@attribute security {0,1}

@data

876.029,-193.66,1
1110.8800000000001,-423.19,0
980.13199999999995,79.722300000000004,1
974.13900000000001,217.07300000000001,1
1127.1900000000001,-312.41000000000003,0
858.62699999999995,288.54599999999999,1
740.85199999999998,465.29500000000002,1
1127.71,455.84899999999999,1
1208.8299999999999,397.26900000000001,0
[...]
```

Obrázek 5.1: Množina vzorů OMIB ve formátu arff.



Obrázek 5.2: Shluky vzorů OMIB databáze. Křížkem je označena třída ’0’ a kroužkem třída ’1’.

```
@relation letter-image-recognition

@attribute x-box numeric
@attribute y-box numeric
@attribute width numeric
[...]
@attribute lettr {A,B,C,[...],Z}

@data
2,8,3,5,1,8,13,0,6,6,10,8,0,8,0,8,T
4,9,6,7,7,7,7,5,6,7,6,8,3,8,3,8,H
4,4,6,5,4,8,9,4,5,7,6,9,3,7,8,8,J
2,10,3,8,1,11,3,10,3,13,7,13,1,6,0,8,J
4,10,6,7,5,6,8,3,4,8,7,7,6,7,4,7,J
[...]
```

Obrázek 5.3: Ukázka souboru ve formátu arff s množinou vzorů pro písmena anglické abecedy. Řetězec `[...]` označuje vynechaný prostor.

Ukázka částí textového souboru obsahující množinu vzorů s písmeny anglické abecedy je na obr. 5.3.

Příznakové vektory byly získány z rastrových obrázků velkých písmen anglické abecedy. Každý znak na obrázcích byl pořízen z 20 různých znakových sad. Jednotlivé obrázky byly binarizovány (převeden na černé a bíle pixely). Každé písmeno bylo náhodně deformováno pomocí warpingu, aby každé písmeno bylo jedinečné. Ukázka části modifikovaných znaků je k nahlédnutí na obr. 5.4. Z upravených znaků byly vytvořeny příznakové vektory, které obsahovaly 16 netřídových numerických atributů (viz obr. 5.3). Poslední třídový atribut nabývá 26 různých hodnot, které označují příslušný znak abecedy. Hodnoty numerických atributů byly převedeny do celých čísel v intervalu od 0 do 15. Jednotlivé atributy mají tento význam:

1. Horizontální pozice, nebo-li počet pixelů od okraje obrázku ke středu nejmenšího možného obdélníku, který obsahuje pouze černé pixely.

2. Vertikální pozice, nebo-li počet pixelů od spodního okraje obrázku k spodní hraně obdélníku ohraničujícího černé pixely (tzv. bounding box).

3. Šířka bounding boxu v pixelech.

4. Výška bounding boxu v pixelech.

Obrázek 5.4: Ukázka znaků po deformaci warppingem.

5. Počet černých pixelů v obrázku znaku.

6. Průměr horizontálních pozic všech černých pixelů vzhledem ke středu bounding boxu dělených jeho šířkou. Hodnota tohoto atributu vzoru je záporná, pokud více černých pixelů bounding boxu je umístěno blíž k jeho levému okraji.

7. Průměr vertikálních pozic všech černých pixelů od středu bounding boxu dělených výškou bounding boxu.

8. Střední čtvercová hodnota horizontálních vzdáleností pixelů od středu bounding boxu dělených jeho šířkou (podobně jako u 6. atributu). Hodnoty atributu budou vyšší pro obrázky znaků, které mají pixely víc vzdálené od středu bounding boxu (např. písmena W a M).

9. Střední čtvercová hodnota vertikálních vzdáleností pixelů získaných podobně jako pro 7. atribut.

10. Průměr součinů horizontálních a vertikálních vzdáleností černých pixelů získaných stejně jako pro 6. a 7. atribut. Hodnota atributu je kladná pro znaky, které mají černé pixely v blízkosti vedlejší diagonály bounding boxu a zápornou hodnotu pro černé pixely v blízkost hlavní diagonály bounding boxu.

11. Střední hodnota horizontální vzdálenosti vynásobené vertikální vzdáleností černých pixelů od středu bounding boxu na druhou, nebo-li korelace horizontálního rozptylu s vertikální pozicí.

12. Střední hodnota vertikální vzdálenosti vynásobené horizontální vzdáleností černých pixelů od středu bounding boxu na druhou, nebo-li korelace vertikálního rozptylu s horizontální pozicí.

13. Průměrný počet hran (černý pixel nacházející se bezprostředně vpravo od bílého pixelu nebo vpravo od okraje obrázku), které se objeví při systematickém průchodu obrázku po řádcích pixelů zleva doprava.

14. Součet vertikálních pozic hran, které se objeví při systematickém průchodu po řádcích (viz 13. atribut).

15. Střední počet hran (černý pixel, který se nachází bezprostředně nad bílým pixelem nebo nad okrajem obrázku), které se objeví při systematickém průchodu obrázku po sloupcích pixelů odspodu nahoru.

16. Součet horizontálních pozic hran, které se objeví při systematickém průchodu po sloupcích (viz 15. atribut).

A	B	C	D	E	F	G	H	I	J	K	L	M
394	383	368	402	384	388	387	367	378	374	369	380	396
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
392	377	401	391	379	374	398	407	382	376	393	393	367

Tabulka 5.1: Počty vzorů označených stejnou třídou.

Původní databáze obsahovala 20 000 vzorů. My jsme použili polovinu (10 000 vzorů). Počty vzorů označených stejnou třídou jsou znázorněny v tabulce 5.1.

Všechny atributy jsme normalizovali a poslední atribut jsme „zjednodušili“ tak, že některá písmena jsme nahradili jedničkou a ostatní nulou. Výsledek naší binarizace je zobrazen v tabulce 5.2. Tedy predikovaná třída nabývala binárních hodnot. Hodnoty atributů vzorů jsme dále normalizovali, tzn. $a(t) \in \langle 0, 1\rangle$ pro všechny vzory t, které jsme měli k dispozici. Tuto úpravu jsme udělali především kvůli algoritmu pro budování SDT stromů. Základní verze algoritmu totiž vyžaduje, aby hodnoty atributů byly normalizovány (viz kapitola 4.2).

A	B	C	D	E	F	G	H	I	J	K	L	M
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0	1	0	1	1	1	0	1	1	1	1	1	1
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
1	0	1	0	1	0	1	1	0	0	0	0	0

Tabulka 5.2: Binarizace tříd vzorů písmen.

V našich experimentech jsme často srovnávali výsledky algoritmů pro různé velikosti množin vzorů. Z původních množin jsme generovali množiny o menším počtu vzorů. Menší množiny byly generovány pomocí tzv. *stratifikace*. Stratifikace je proces, který zajistí, že poměr počtu vzorů označených třídou c vzhledem k počtu všech vzorů v původní množině je stejný jako odpovídající poměr v nové menší množině vzorů [37]. K testování jsme použili cross validaci [37] (přesněji desetidílnou stratifikovanou cross validaci). Desetidílná stratifikovaná cross validace rozdělí množinu vzorů na 10 přibližně stejně velkých podmnožin. Vzory jsou opět rozdělovány stratifikovaně, tzn. že v každé podmnožině jsou zachovány stejné poměry počtu vzorů s danou třídou jako v původní množině. V prvním kroku jsou vzory devíti podmnožin použity k budování klasifikátoru a na zbývající podmnožině vzorů je klasifikátor testován. V druhém kroku se vezme jiná podmnožina k testování a na zbylých devíti podmnožinách je vybudován klasifikátor. Proces budování a testování klasifikátoru se provede pro všechny podmnožiny. To znamená, že každý vzor z původní množiny bude použit právě jednou v testovací fázi. Výsledný odhad je pak průměrem výsledků dílčích testů cross validace.

5.2 CART

V této části jsme zkoumali vlastnosti stromu CART. Testy jsme prováděli na upravené databázi znaků abecedy. Nejprve jsme srovnávali závislost přesnosti a složitosti stromů na zvoleném kritériu pro dělení. Dále jsme se zabývali ořezáváním stromu.

Ve fázi růstu stromu CART je možné použít Giniho nečistotu jako měřítko, podle kterého rozdělujeme množinu trénovacích vzorů do jednotlivých potomků uzlu, nebo twoing kritérium, které bylo popsáno v kapitole 3.2.2. Výsledky obou metod jsou shrnuty v tabulce 5.3

Velikost GS	Gini			Twoing		
	$\|L(G)\|$	$E(U)$	$E(P)$	$\|L(G)\|$	$E(U)$	$E(P)$
100	9	18,75%	18,75%	9	19,64%	21,43%
200	9	26,91%	27,35%	5	26,91%	29,60%
500	19	21,22%	22,84%	19	19,60%	21,22%
1000	25	17,18%	18,08%	32	15,56%	15,92%
2000	48	13,77%	14,39%	77	13,41%	14,08%
3000	103	11,82%	12,03%	79	12,03%	12,33%
4500	109	9,84%	10,26%	109	10,84%	11,20%

Tabulka 5.3: Srovnání metod dělení vrcholu v závislosti na velikosti množiny vzorů GS. $|L(G)|$ je počet listů ořezaného stromu, $E(U)$ je poměr chyby neořezaného stromu a $E(P)$ je poměr chyby ořezaného stromu (v procentech).

Twoing kritérium nepřineslo zlepšení a výsledky jsou v obou případech téměř srovnatelné. Twoing kritérium totiž nemělo žádný vliv na výpočet minimálního úbytku nečistoty v uzlu. Rozdělení na množiny vzorů C_1 a C_2 (viz kapitola 3.2.2) nemělo žádný efekt. Upravená množina znaků abecedy totiž obsahovala pouze vzory, které byly označeny dvěma třídami $c_1 =$ '0' a $c_2 =$ '1'. Množina vzorů C_1 tedy obsahovala pouze vzory označené třídou $c_1 =$ '0' a v C_2 byly pouze vzory označené třídou $c_2 =$ '1' nebo naopak. V každém případě množiny C_1 a C_2 vždy obsahovaly vzory označené jednou třídou. Výpočet úbytku nečistoty s použitím pouhé Giniho nečistoty se pak v tomto případě téměř neliší od výpočtu úbytku nečistoty pomocí twoing kritéria.

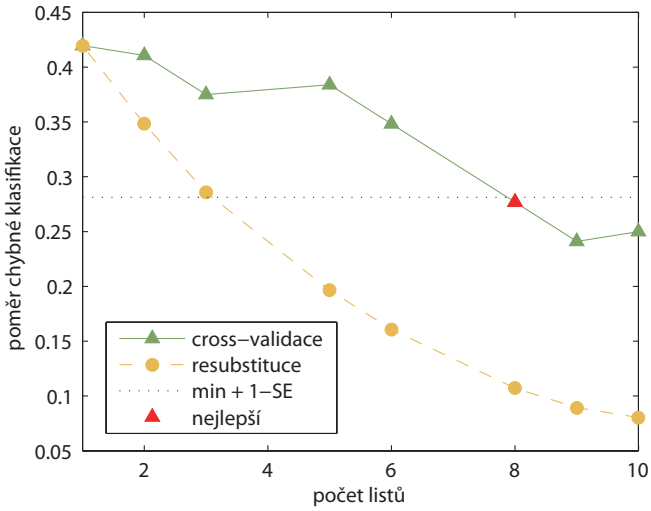
Dále jsme se pokusili srovnat Giniho nečistotu a twoing kritérium na „nezjednodušené“ množině písmen. Hodnoty netřídových atributů vzorů nabývaly hodnot celých čísel $\{0, \ldots, 15\}$. Třídový atribut vzorů byl tvořen 26 hodnotami 'A' až 'Z'. Výpočet úbytku nečistoty podle twoing kritéria se pro více tříd bude chovat jinak, než když bude pro výpočet úbytku nečistoty použita pouze Giniho nečistota. Experimenty jsme opět prováděli na různě velkých množinách GS viz tabulka 5.4. Z tabulky je patrné, že pro menší množiny vzorů byly výsledky velmi špatné. Např. pro $|GS| = 100$ byla chyba

	Gini		Twoing	
$\|GS\|$	$E(U)$ $\|L(G)\|$	$E(P)$ $\|L(G)\|$	$E(U)$ $\|L(G)\|$	$E(P)$ $\|L(G)\|$
100	84% (15)	84% (15)	84% (18)	86% (13)
200	63% (37)	63% (32)	59% (33)	61% (20)
500	44,66% (81)	45,2% (35)	47,6% (77)	48,4% (39)
1000	39,6% (134)	40,5% (98)	41,3% (124)	41,5% (76)
2000	32,9% (223)	33,7% (165)	32,5% (226)	33,5% (114)
4000	25,1% (374)	25,68% (268)	24,55% (364)	24,75% (351)
10 000	18,28% (770)	18,58% (536)	17,71% (736)	17,89% (527)

Tabulka 5.4: Srovnání metod dělení vrcholu v závislosti na velikosti množiny vzorů GS „nezjednodušené“ množiny písmen. $E(U)$ je poměr chyby neořezaného stromu a $E(P)$ je poměr chyby ořezaného stromu (v procentech). Složitost $|L(G)|$ označuje počet listových uzlů stromu $L(G)$.

CART stromu 84%. Počet vzorů množiny GS označené jednou třídou byl příliš malý na to, aby strom dával rozumné výsledky. V průměru 4 vzory označené stejnou třídou pro 26 různých tříd je pro budování stromu nedostačující počet. Pro menší množiny vzorů byly výsledky experimentů nestálé. Někdy se ukázalo přesnější použití Giniho nečistoty a někdy zase použití twoing kritéria. Tyto výchylky jsou způsobeny malou velikostí množin GS a také náhodným rozdělováním vzorů do podmnožin při cross validaci. Stromy nemají totiž dostatečnou velikost a přesnost stromu je citlivá na změnu struktury stromu. Výsledky se ustálily až pro množiny vzorů GS, které obsahovaly více jak 2000 vzorů. Průměrný počet vzorů označených jednou třídou byl pro $|GS| = 2000$ přibližně 77. Stromy vybudované s použitím twoing kritéria byly mírně přesnější a měly před ořezáním menší počet listů. S použitím twoing kritéria jsou obecné charakteristiky řešeny v úrovních stromu blízko kořene a charakteristiky typické pro menší množinu vzorů se zpracovávají až ve větší hloubce stromu. Takto vybudované stromy jsou pak přesnější na neznámých vzorech a mají menší velikost.

K ořezávání CART stromů se používá metoda MCCP, která vygeneruje posloupnost podstromů a z nich vybere nejlepší strom z hlediska složitosti a přesnosti. Metoda MCCP byla podrobněji popsána v části 3.4.3. K budování



Obrázek 5.5: Chyby podstromů, které vygeneruje algoritmus MCCP pro OMIB databázi ($|LS| = 100$). Zvýrazněné hodnoty chybové funkce odpovídají bodům (podstromům), pro které algoritmus MCCP spočetl chybu viz část 3.4.3.

stromu jsme použili množinu vzorů popisujících znaky abecedy. Na obr. 5.5 jsou znázorněny chyby CART podstromů v závislosti na složitosti stromu. Tzv. *resubstituční chyba* je chyba spočtená na datech, která byla použita pro učení klasifikátoru, tedy v našem případě pro růst stromu CART. Chyba $\hat{R}(G_k)$ každého podstromu G_k posloupnosti podstromů, kterou vygeneruje MCCP, je odhadnuta na vzorech určených pro růst stromu CART. Graf na obr. 5.5 ukazuje, že resubstituční chyba je příliš optimistická, nebo-li chyba stromu klesá s přibývající velikostí stromu.

Druhý graf na obr. 5.5 ukazuje chyby podstromů vygenerované posloupnosti podstromů spočtenou pomocí (desetidílné stratifikované) cross validace. Strom se nejdříve vybuduje na celé množině určené pro růst a na ní se určí posloupnost parametrů α_k, které určují posloupnost ořezaných podstromů G_k. Poté je strom vybudován desetkrát po sobě za sebou na 9/10 původní množiny vzorů (viz kapitola 5.1). Pro každý vybudovaný strom je podle posloupnosti α_k vygenerována posloupnost podstromů, na kterých je odhadnuta chyba na zbývající 1/10 vzorů. Pro k-tý podstrom posloupnosti je pak chyba $\hat{R}(G_k)$ získána průměrem z dílčích výsledků cross validace [2].

Od daného bodu s rostoucí složitostí stromu roste i chyba. Z obr. 5.5 vidíme, že chyba je minimální pro strom, který má 9 listů (hodnota chyby je 24,1%). MCCP se však snaží najít kompromis mezi složitostí a přesností stromu. K tomu se použije 1-SE pravidlo popsané v části 3.5, které vybere podstrom, jehož chyba je první menší nebo rovna minimální chybě podstromů plus SE. Tato hranice je na obr. 5.5 označena tečkovanou čárou. Nejbližší

chyba pod touto hranicí odpovídá stromu s osmi listovými uzly. Jeho poměr chybné klasifikace je 27,7%.

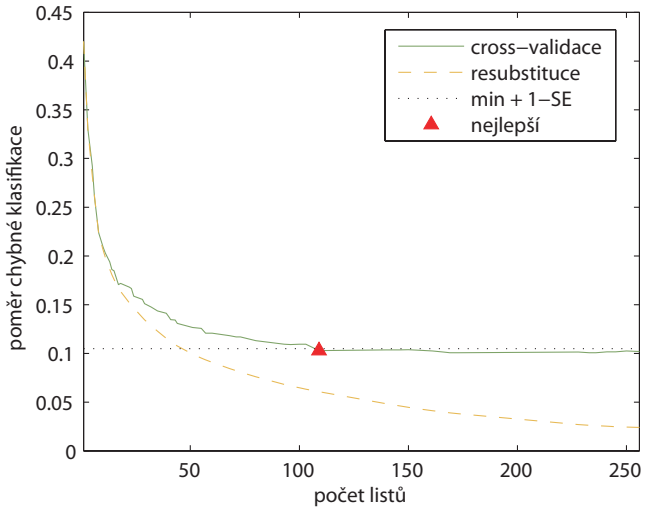
Srovnání chyb podstromů získané resubstitucí a cross validací dopadlo podle našich předpokladů. Resubstituční chyba je menší a optimisticky ubývá s rostoucí velikostí stromu. Naopak chyba odhadnutá cross validací je větší a její pokles již není tak výrazný. Chyba odhadnutá cross validací je však důvěryhodnější než chyba odhadnutá resubstitucí, protože je odhadnuta na datech, která nejsou použita pro růst stromu.

V tabulce 5.5 jsme zkoumali závislost ořezání stromu na velikosti množiny vzorů pro budování stromu GS. Chyby byly odhadovány pomocí cross validace. Z výsledku je patrné, že čím byla GS větší, tím byl poměr chybné klasifikace menší. Např. rozdíl mezi chybou pro 200 vzorů a 500 vzorů byl asi 6% a rozdíl mezi chybou pro 2000 vzorů a 3000 vzorů byl už jen kolem 2%. Z toho jsme usoudili, že od určité velikosti množiny vzorů pro budování se přesnost klasifikátoru již výrazně nezlepšuje a je zbytečné trénovat klasifikátor na větší množině vzorů. Dále čím větší byla množina GS, tím chyba SE konvergovala k nule a díky tomu pravidlo 1-SE vybralo podstrom s chybou, která se více přibližovala minimální chybě podstromu v posloupnosti.

Velikost GS	Složitost ($\|L(G)\|$)		Poměr chyby	
	Neořezaný	Ořezaný	Neořezaný	Ořezaný
100	10	8	27,68%	30,36%
200	23	8	26,91%	26,91%
500	51	31	20,86%	20,86%
1000	88	38	17,18%	18,26%
2000	156	48	13,95%	14,57%
3000	186	79	12,15%	12,66%
4500	256	109	10,04%	10,36%

Tabulka 5.5: Složitost a poměr chyby CART pro různé velikosti množin vzorů určených pro budování stromu. Použita byla upravená množina znaků abecedy.

Z tabulky 5.5 je možné vypozorovat, že čím větší byl strom, tím docházelo k většímu ořezání. Na obr. 5.6 je znázorněna závislost poměru chybné klasifikace na velikosti podstromů vygenerovaných algoritmem MCCP pro množinu obsahující 5000 vzorů. Na obrázku je vidět, že chyba se od daného bodu již výrazně nezmenšovala a byla téměř konstantní. Tento bod odpovídá stromu, který vybere pravidlo 1-SE a není snadné jej odhadnout volbou ukončovací podmínky a díky tomu je dávána přednost ořezávání kompletně vybudovaného stromu. Velikost vybudovaného stromu se ukázala částečně závislá na množství trénovacích dat. Tedy čím více vzorů je použito k růstu stromu, tím je velikost stromu větší. Z tabulky 5.5 je vidět, že míra ořezání rostla s velikostí stromu. To je způsobeno tím, že strom se do určité fáze zpřesňuje, tím



Obrázek 5.6: Chyby podstromů, které vygeneruje algoritmus MCCP pro velkou množinu vzorů GS. Použili jsme databázi znaků abecedy, která obsahovala 5000 vzorů.

se zmenšuje chyba, a od určité fáze strom dosáhne své maximální přesnosti a pokles chyby je pak minimální.

Na větší množině vzorů byl vybudován větší strom. Strom však od určitého bodu nezlepšuje svou přesnost a tak je možné pro větší množiny vzorů strom více ořezat s tím, že bude zachována jeho přesnost.

V dalším testu jsme zkoumali vlastnosti ořezávacího algoritmu MEP, který jsme nastínili v kapitole 3.4.2. Ořezávání MEP je založeno na m-odhadu, jehož vlastnosti můžeme měnit pomocí parametru m. Pomocí m-odhadu můžeme „řídit“ statickou chybu, podle které se rozhodujeme, zda daný podstrom máme při ořezávání odstranit nebo ne. Tím, jak měníme hodnoty parametru m, určujeme míru ořezání. V tabulce 5.6 můžeme vidět, jakým způsobem parametr m ovlivní přesnost a složitost stromu. V tabulce jsou výsledky pro různé hodnoty parametru m a různě velké množiny vzorů GS. Čím byla hodnota parametru m větší, tím se statická chyba uzlu spíše blížila nule, protože m-odhad konvergoval k jedničce. Tím byla chyba malá i ve vyšších úrovních stromu a tak docházelo k většímu ořezání. Např. pro $|GS| = 4500$ a hodnotu parametru $m = 20$ obsahuje strom 239 vrcholů a pro $m = 500$ ořezaný strom obsahuje už jen 107 vrcholů. Pro velké hodnoty m a menší velikosti GS je statická chyba E_S natolik ovlivněna parametrem m, že není již dále závislá na počtech vzorů dané třídy, které se dostanou do daného uzlu. m-odhad se pak bude přibližovat apriorní pravděpodobnosti dané třídy. To znamená, že od určitého bodu se již nebudou statické chyby příliš měnit a ořezávání se zastaví. Tento jev je nejvíce patrný pro nejmenší testovanou množinu GS, která obsahovala 100 vzorů. Pro hodnoty $m \geq 200$ již složitost

	Velikost GS			
	4500	2000	500	100
m	Poměr chybné klasifikace Složitost ($\|V(G)\|$)			
0	9,76% (717)	14,22% (431)	21,95% (141)	22,2% (33)
2	9,62% (573)	14,35% (317)	21,6% (109)	21,29% (29)
5	9,44% (459)	14,4% (259)	21,42% (93)	21,29% (29)
10	9,62% (295)	14,8% (207)	21,06% (87)	21,29% (25)
20	9,96% (239)	15,52% (135)	22,69% (73)	26,59% (25)
50	12,06% (151)	17,81% (79)	22,87% (39)	25,83% (21)
100	12,86% (115)	20,33% (67)	25,91% (45)	27,65% (25)
200	15,66% (161)	21,01% (49)	29,48% (65)	27,65% (21)
500	18,52% (107)	18,85% (133)	30,21% (37)	27,65% (21)
1000	17,04% (107)	18,67% (151)	30,21% (37)	27,65% (21)

Tabulka 5.6: Test závislosti přesnosti a složitosti na volbě parametru m pro ořezávání MEP, který používal pro výpočet chyby m-odhad. Pro dělení uzlu byla použita Giniho nečistota, práh pro minimální počet vzorů v listech byl nastaven na hodnotu 2.

stromu zůstala nezměněna.

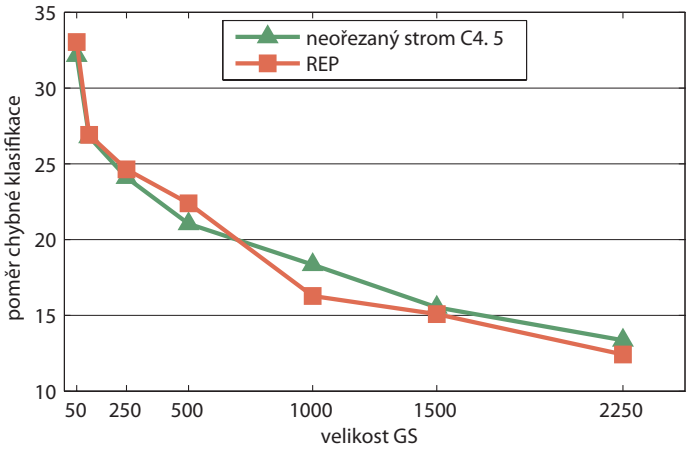
Přesnost stromu se do daného bodu m zvětšovala a pak se pozvolna zmenšovala. Neořezaný strom odpovídá hodnotě parametru $m = 0$ a je mírně přetrénovaný, protože do větších hloubek stromu se dostane málo trénovacích vzorů, dělení jsou závislá na malém počtu vzorů a tím jsou uzly v nižších vrstvách náchylnější k chybám. Jakmile dosáhneme optimální velikosti stromu, strom začne pozvolna ztrácet na své přesnosti. Např. pro $|GS| = 4500$ a $m = 5$ byl poměr chybné klasifikace 9,44% a pro $m = 20$ se chyba zhoršila pouze o 0,52%, přitom složitost klesla téměř o polovinu.

V experimentech provedených s CART stromy jsme srovnáním základní metody pro výpočet úbytku nečistoty a twoing kritéria zjistili, že pro množiny vzorů s dvěma třídami se výsledné stromy od sebe příliš neliší. K mírnému zlepšení dojde pro množiny vzorů s více třídami. Čím je však počet možných tříd vzorů vyšší, tím je potřeba pro vybudování stromu většího počtu vzorů. Experimenty také ukázaly, že neexistuje ořezávací algoritmus, který by výrazně dominoval nad ostatními. Ořezáváním metodou MCCP a MEP dostaneme přibližně stejně přesné stromy. MEP však ořezává mírněji než algoritmus MCCP. Výsledkem algoritmu MEP jsou tedy větší stromy. Lepší přesnosti algoritmu MEP je možné docílit vhodnou volbou parametru m. Nejlepší dosažená přesnost stromů se pohybovala okolo 9%, což nemusí být vždy dostačující. Lepší přesnost je možné docílit výběrem lepších příznaků pro trénovací množiny vzorů, popřípadě větší množinou trénovacích vzorů.

5.3 C4.5

Budování stromů C4.5 bylo teoreticky popsáno v kapitole 3.6. Strom C4.5 je možné ořezat metodou REP nebo metodou MEP. Experimentální srovnání obou metod je hlavním tématem této kapitoly.

Budování stromu metodou C4.5 jsme opět testovali na databázi obsahující znaky abecedy. Na obr. 5.7 jsou grafy závislosti poměru chybné klasifikace na velikosti množiny určené pro růst. První graf znázorňuje neořezaný strom C4.5 a druhý graf strom ořezaný metodou REP. Pro růst byly použity stejně velké množiny vzorů. Pro ořezávání metodou REP je nutné mít k dispozici nezávislou množinu ořezávacích vzorů. Protože jsme měli k dispozici dostatečně velkou množinu vzorů, použili jsme k ořezávání množinu vzorů, jejíž velikost byla stejná jako velikost množiny vzorů určených pro růst stromu. Kdybychom pracovali s omezenou množinou vzorů, museli bychom množinu ořezávacích vzorů volit menší. Z obrázku si můžeme všimnout, že chyba ořezaného stromu se výrazněji nelišila od neořezaného stromu. Pro menší velikosti množin vzorů byla chyba ořezaného stromu mírně větší, než tomu bylo v případě neořezané verze. Od určité velikosti množiny vzorů však byla chyba ořezaného stromu mírně menší. Pro větší množiny docházelo k mírnému pře-



Obrázek 5.7: Závislost přesnosti neořezaného stromu C4.5 a ořezaného stromu metodou REP na velikosti množiny GS určenou pro růst stromu. Množina ořezávacích vzorů pro REP měla stejnou velikost jako GS.

trénování neořezaného stromu a v tomto případě došlo ořezáním k zpřesnění stromu. Naopak pro menší množiny vzorů není strom ještě dostatečně přesný a zmenšení stromu má za následek mírné zhoršení přesnosti na nezávislých datech.

Dále jsme se zaměřili na ořezávací algoritmy stromu C4.5. V tabulce 5.7 jsou shrnuty výsledky našeho zkoumání ořezávacího algoritmu PEP. Postupně jsme měnili parametr věrohodnosti c, který určuje míru ořezání stromu. Čím byla hodnota tohoto parametru větší, tím se zvětšovala také míra ořezání. To je patrné z malých čísel tabulky v závorce, která určují složitost stromu. Na základě volby parametru věrohodnosti byl maximální rozdíl mezi nejmenší a největší chybou pro danou velikost množiny určené pro růst přibližně 3,5%. Průměrná hodnota věrohodnosti, pro kterou jsme dostali nejpřesnější stromy, je 0,25. Pro větší množiny vzorů je rozdíl mezi hodnotou chyby pro $c = 0{,}25$ a minimální chybou malý a pro množiny, které měly více než 2000 vzorů, byl tento rozdíl dokonce $< 0{,}1\%$.

Volba parametru c není pro přesnost stromu kritická. Nejlepší výsledky byly získány pro $c = 0{,}25$. Malé hodnoty c dají maximálně ořezaný strom s dostatečnou přesností. Experimenty provedené na PEP nám potvrdily, že PEP ořezává stromy mírně.

Cílem našeho dalšího experimentu bylo porovnat ořezávací algoritmy REP a PEP. REP vyžaduje ke svému chodu navíc speciální nezávislou množinu vzorů, která nebyla použita pro budování stromu. Naopak PEP odhaduje přesnost přímo na vzorech určených pro růst a tedy nepotřebuje žádnou speciální nezávislou množinu vzorů. Zde jsme narazili na problém, na jak velké množině trénovacích vzorů tyto dva algoritmy porovnat. První možnost bylo

	Velikost GS						
	100	200	500	1000	2000	3000	4500
c	Poměr chybné klasifikace (Složitost stromu ($\|V(G)\|$))						
0,01	26,79% (21)	22,87% (21)	23,92% (49)	19,15% (67)	15,61% (121)	14,28% (171)	11,04% (241)
0,02	26,79% (21)	21,52% (25)	22,84% (49)	19,33% (73)	15,11% (137)	13,32% (201)	10,16% (249)
0,05	26,79% (21)	20,18% (25)	21,76% (65)	19,06% (99)	14,53% (223)	12,24% (269)	9,16% (287)
0,1	25,89% (21)	20,18% (25)	21,40% (79)	18,17% (117)	13,77% (261)	12,00% (315)	8,80% (329)
0,15	25,89% (21)	20,18% (25)	20,50% (83)	17,99% (141)	13,50% (265)	11,94% (329)	8,82% (383)
0,2	27,86% (27)	20,18% (41)	21,22% (103)	18,08% (173)	13,72% (267)	11,73% (351)	8,58% (407)
0,25	27,68% (27)	19,73% (45)	20,86% (109)	17,54% (183)	13,59% (287)	11,73% (373)	8,54% (477)
0,3	27,68% (29)	20,18% (47)	21,04% (115)	17,72% (201)	13,54% (315)	11,70% (367)	8,52% (497)
0,35	26,79% (29)	20,18% (47)	21,40% (115)	17,27% (213)	13,50% (315)	11,76% (399)	8,48% (525)
0,4	28,57% (29)	20,18% (47)	21,04% (115)	17,36% (213)	13,54% (315)	11,76% (411)	8,48% (525)
0,45	28,57% (29)	20,18% (47)	21,04% (115)	17,36% (221)	13,54% (325)	11,67% (419)	8,46% (525)
0,5	28,57% (29)	20,18% (47)	21,04% (115)	17,53% (221)	13,59% (331)	11,73% (425)	8,56% (545)

Tabulka 5.7: Závislost složitosti a přesnosti stromu na faktoru věrohodnosti c pro PEP. Výsledky byly získány cross validací na databázi znaků abecedy.

	REP	PEP		PEP
$\|GS\|$	$E(G)$ ($\|V(G)\|$)	$E(G)$ ($\|V(G)\|$)	$\|GS\|$	$E(G)$ ($\|V(G)\|$)
50	33,04% (9)	32,14% (13)	100	27,68% (27)
100	26,91% (23)	27,68% (27)	200	19,73% (45)
250	24,64% (27)	24,1% (57)	500	20,86% (109)
500	22,39% (69)	20,86% (109)	1000	17,54% (183)
1000	16,28% (145)	17,54% (183)	2000	13,59% (287)
1500	15,09% (207)	15,9% (237)	3000	11,73% (373)
2250	12,42% (253)	13,24% (309)	4500	8,54% (477)

Tabulka 5.8: Srovnání REP a PEP. Pro algoritmus REP byla použita ořezávací množina PS, jejíž velikost byla stejná jako velikost GS ($|GS| = |PS|$). Ve třetím sloupci byla pro PEP použita poloviční GS než pro PEP v pátém sloupci tabulky.

porovnat výsledky algoritmů pro stejně velké množiny určené pro růst s tím, že pro algoritmus REP budeme mít speciální ořezávací množinu. Druhou možností bylo použití celé trénovací množiny pro REP i PEP s tím, že v případě algoritmu REP ji rozdělíme na množinu vzorů pro růst a na množinu vzorů určenou pro ořezávání. Obě situace jsme zkoumali v tabulce 5.8. REP měl k dispozici stejně velké množiny vzorů pro růst i pro ořezávání. Ve třetím sloupci tabulky jsou výsledky experimentů PEP pro stejně velké množiny určené pro růst. Výsledky v pátém sloupci jsou získány z experimentů PEP pro všechny trénovací vzory. Tedy součástí množiny vzorů pro budování stromu a ořezávání metodou PEP byly i ořezávací vzory pro metodu REP.

Pro stejně velké množiny pro růst GS se přesnost obou algoritmů nijak výrazně nelišila. Pokud ovšem vezmeme v úvahu složitost ořezaných stromů, dával algoritmus REP na výstup menší stromy. Čím byla množina vzorů větší, tím se zvětšoval také rozdíl mezi složitostí stromu ořezaného metodou REP a stromu ořezaného metodou PEP (viz tabulka 5.8). Tento jev však není překvapující, protože PEP je obecně mírnější vzhledem k ořezávání.

V předchozím experimentu jsme znevýhodnili algoritmus PEP, protože k vybudování a ořezávání stromu metodou PEP jsme použili poloviční počet vzorů než v tomu bylo v případě algoritmu REP. Nyní zopakujeme experiment s metodou PEP, ale k vybudování a ořezávání stromu použijeme stejně velké

množiny vzorů. Situace je znázorněna v posledním sloupci tabulky 5.8. Pro algoritmus REP jsme použili polovinu vzorů pro růst stromu a polovinu pro ořezávání. Výsledky algoritmu REP jsou v prvním sloupci. V tomto případě algoritmus PEP vykazoval větší přesnost a byl přibližně o 5% přesnější než REP. Tedy se stejně velkou trénovací množinou jsme získali přesnější klasifikátor. Díky tomu, že jsme vybudovali strom z dvakrát větší množiny vzorů než v předchozím experimentu a ořezali jsme jej algoritmem PEP, je jeho velikost podstatně větší než je tomu v případě REP.

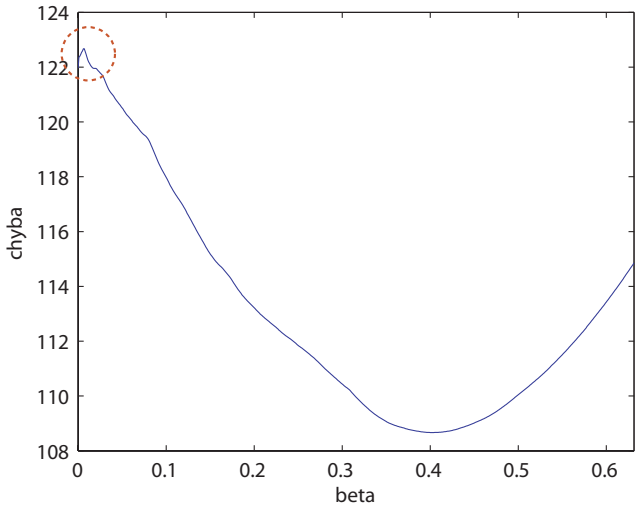
Z výsledků experimentů je patrné, ve kterých případech použít algoritmus REP a ve kterých algoritmus PEP. Pokud máme k dispozici omezenou množinu vzorů, je výhodnější použít algoritmus PEP, protože celá trénovací množina je použita pro růst stromu. Pokud máme k dispozici dostatečně velkou trénovací množinu, je lepší použít algoritmus REP, protože dostaneme stejně přesný, avšak menší strom než v případě PEP.

V této kapitole jsme zkoumali především ořezávací algoritmy stromu C4.5. Dospěli jsme k závěru, že ořezávání stromu C4.5 nemá výrazný vliv na zhoršení přesnosti stromu. Pro dostatečně velké množiny naopak dochází ke zlepšení přesnosti ořezaných stromů. Pro velké množiny vzorů jsou totiž stromy přetrénované a ořezání má na přesnost a na velikost stromu pozitivní účinek. Volba parametru c pro pesimistické ořezávání (PEP) nemá příliš zásadní vliv na přesnost stromu. Algoritmus PEP produkuje méně ořezané stromy než algoritmus REP, ale nepotřebuje navíc ořezávací množinu vzorů. Z tohoto faktu také plynou možnosti použití zmíněných algoritmů v praxi.

5.4 SDT - růst stromu

Předmětem této kapitoly bude experimentální analýza chybové funkce E_S v uzlu stromu SDT. Budeme také zkoumat chování přesnosti a složitosti SDT stromu v závislosti na zvolené ukončovací podmínce. Pro jednotlivé experimenty jsme zvolili normalizovanou množinu vzorů OMIB.

Růst strom SDT popsaný v kapitole 4.3 je založen na hledání optimálního dělení, které rozdělí vzory tak, aby chyba podle vztahu 4.3.3 byla minimální. Autoři v článku [28] popisují, že hledání optimálních parametrů α a β je možné rozdělit do dvou fází. V první fázi se položí pevně $\beta = 0$ a pro tuto hodnotu se najde dělení α, které minimalizuje chybu viz vztah 4.3.3. V druhé fázi se hledá šířka β pro již nalezené optimální dělení α. To, že je možné proces hledání optimálních parametrů rozdělit na dvě fáze, vychází z tvaru čtvercové chybové funkce [27]. V druhé fázi, kdy se hledá optimální šířka β, autoři článku [28] využívají Fibonacciho prohledávání. Aby bylo možné použít Fibonacciho prohledávání, musí být chybová funkce na prohledávaném intervalu unimodální (viz kapitola 4.3.2). Podle autorů článku [28] je tento předpoklad splněn na všech množinách vzorů, na kterých prováděli experi-



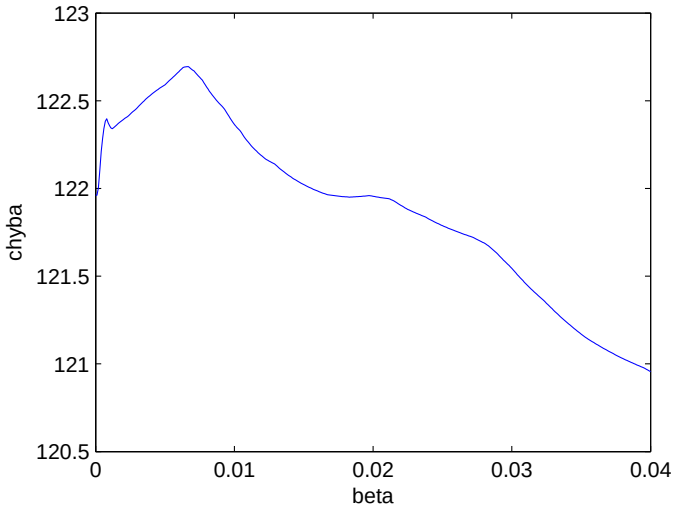
Obrázek 5.8: Chyba uzlu pro 1000 vzorů OMIB databáze.

menty. V našich experimentech se pokusíme zobrazit chybovou funkci pro optimální nalezené dělení α, abychom ověřili, jestli je chybová funkce unimodální na námi použité množině vzorů. Cílem našich testů bylo najít optimální dělení α a zobrazení chybové funkce přes parametr β.

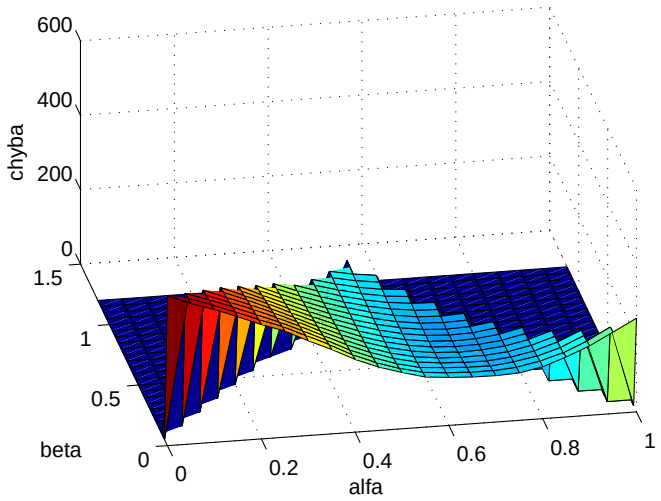
Náš první test byl spuštěn na celou množinu vzorů databáze OMIB, která obsahovala 1000 vzorů. V první fázi jsme získali optimální dělení $\alpha = 0{,}684$, $L_L = 0{,}9$ a $L_R = 0{,}259$. Chybu podle vztahu 4.3.3 jsme spočítali pro všechny možné hodnoty $\beta \in \langle 0, \min\{2\alpha, 2(1-\alpha)\}\rangle$. Výsledek je znázorněn na obr. 5.8. Funkce byla unimodální téměř všude. Ovšem blízko nuly chybová funkce unimodální nebyla, což je nutným předpokladem pro Fibonacciho prohledávání. Na obr. 5.9 je zobrazen detail chybové funkce nesplňující unimodalitu. Na obr. 5.10 je pak graf chybové funkce pro $\alpha \in \langle 0, 1\rangle$ a $\beta \in \langle 0, \min\{2\alpha, 2(1-\alpha)\}\rangle$. Je možné si všimnout, že minimum chybové funkce je dosaženo pro stejnou hodnotu parametru α, což nás skutečně opravňuje k tomu, abychom hledali parametry α a β odděleně.

Náš další test zkoumal tvar chybové funkce na malé množině, která byla tvořena deseti vzory. Vzory byly získány stratifikací [37], aby počty vzorů s danými třídami byly přibližně ve stejném poměru jako v původní kompletní množině. Chybová funkce E_S opět nebyla unimodální na celém intervalu. Interval, který porušoval unimodalitu byl teď mnohem širší (viz obr. 5.11). A opět pro větší názornost jsme zobrazili chybovou funkci pro $\alpha \in \langle 0, 1\rangle$ a $\beta \in \langle 0, \min\{2\alpha, 2(1-\alpha)\}\rangle$.

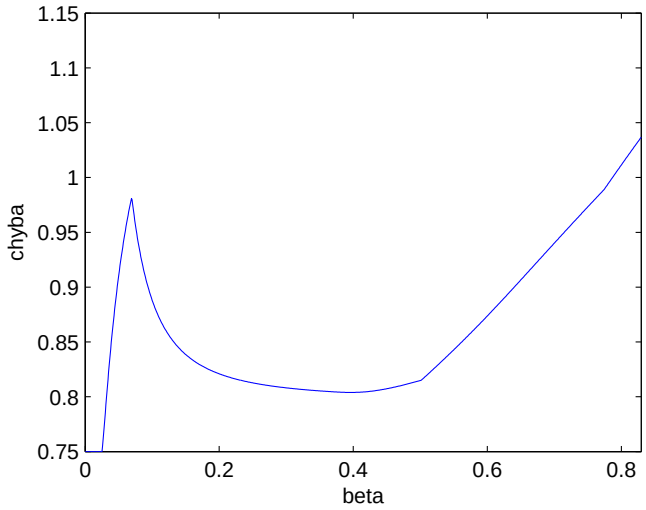
Z našich experimentů vyšlo, že unimodalita chybové funkce byla porušena vždy v pravém okolí nuly. Fibonacciho algoritmus pro menší množiny vzorů uvízne v prvním lokálním minimu (obr. 5.11) v bodě $\beta = 0$. Díky tomu jsme



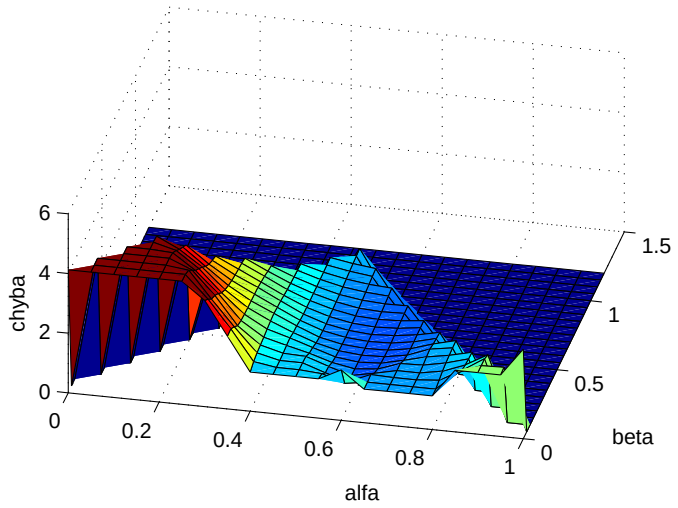
Obrázek 5.9: Detail části chybové funkce uzlu na obr. 5.8, která nesplňuje unimodalitu.



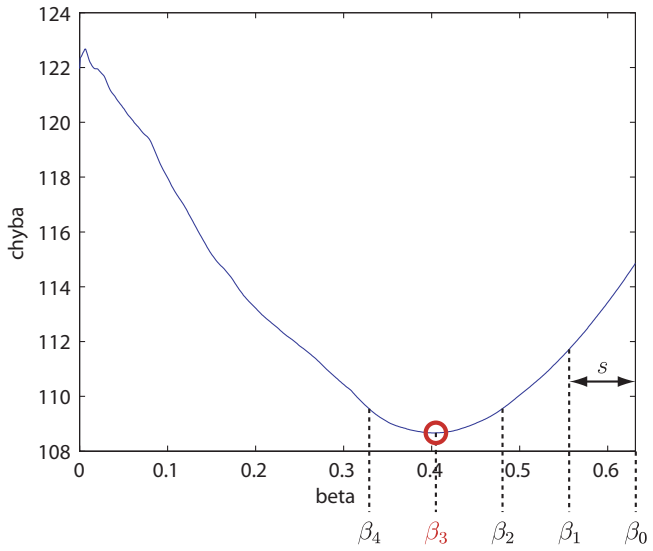
Obrázek 5.10: Chyba uzlu pro $\alpha \in \langle 0,1\rangle$, $\beta \in \langle 0, \min\{2\alpha, 2(1-\alpha)\}\rangle$, $L_L = 0{,}9$ a $L_R = 0{,}259$.



Obrázek 5.11: Chyba uzlu pro 10 vzorů databáze OMIB. $\alpha = 0{,}585$, $L_L = 1{,}0$, $L_R = 0{,}25$, $\beta \in \langle 0, 0{,}83 \rangle$.



Obrázek 5.12: Chyba uzlu pro $\alpha \in \langle 0, 1 \rangle$ a $\beta \in \langle 0, \min\{2\alpha, 2(1-\alpha)\} \rangle$.



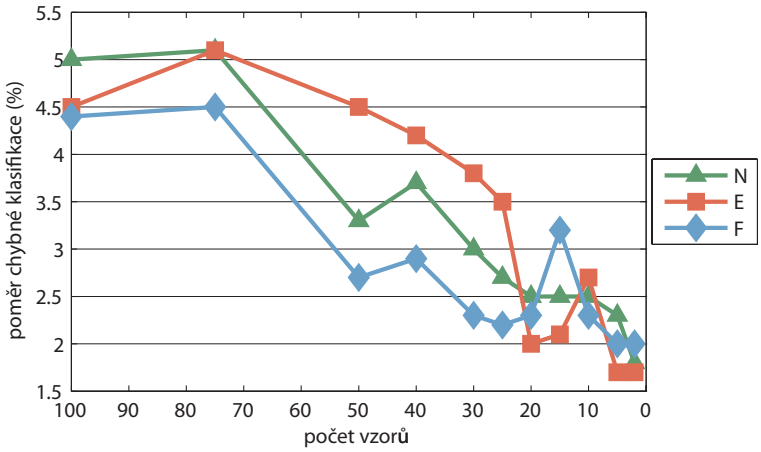
Obrázek 5.13: Schéma našeho algoritmu *první od konce* (POK). Optimální nalezený bod, pro který funkce nabývá minima je β_3. s je krok iterace.

se snažili zkoumat první minimum zprava, které by preferovalo větší hodnoty β a tím i širší oblast přechodu. Tuto jednoduchou námi navrženou metodu jsme nazvali *první od konce* (dále POK).

Mějme interval $\langle a, b\rangle$. Algoritmus POK znázorněný na obr. 5.13 spočte nejprve hodnotu chybové funkce $E_S(\beta_0)$ v pravém bodě intervalu $\beta_0 = b$. V druhém kroku spočte chybu $E_S(\beta_1)$, kde $\beta_1 = b - s$ a s je krok, kterým se posouváme do dalšího bodu. POK se posune do bodu β_1 a spočte chybu $E_S(\beta_2)$, kde $\beta_2 = b - 2s$. Pro β_i tedy platí, že $\beta_i = b - 2 \cdot i$. Algoritmus končí, pokud je $E_S(\beta_i) \leq E_S(\beta_{i+1})$ nebo pokud je $\beta_{i+1} > a$. Algoritmus nám nezaručuje, že najdeme první minimum, ovšem pro hrubý odhad prvního minima zprava nám postačí. První minimum bude tím přesnější, čím bude krok s menší. A podobně čím menší bude krok s, tím větší bude pravděpodobnost, že nalezneme první minimum funkce.

Výsledky naší metody jsme srovnávali s naivním algoritmem, který rozdělí interval a v bodech dělení spočte hodnotu chybové funkce E_S a vybere takový bod dělení, pro který je hodnota chybové funkce minimální. Naivní algoritmus opět nemusí najít globální minimum. Čím je dělení intervalu hustší tím je výsledek přesnější.

U jednotlivých metod jsme se zaměřili na přesnost klasifikace a také složitost vybudovaného neořezaného stromu v závislosti na prahu pro minimální počet vzorů v listových vrcholech. Na obr. 5.14 je zachycena závislost poměru chyby na prahu pro minimální počet vzorů v listech. Pro větší hodnoty prahu nejlépe vychází Fibonacciho prohledávání. Pro zbývající dva algoritmy není



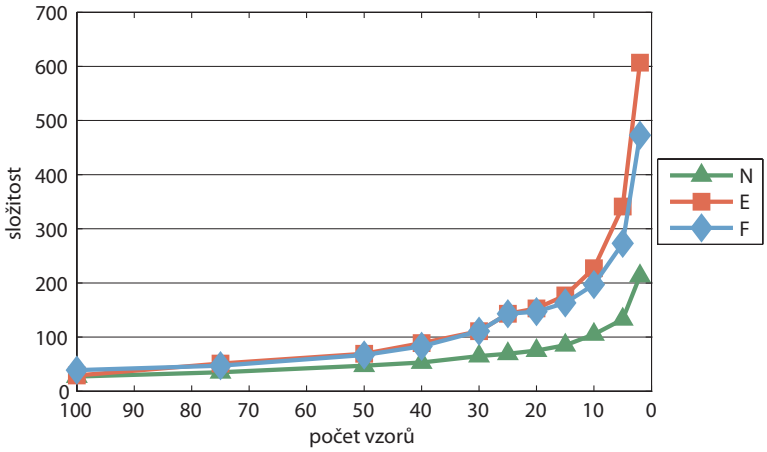
Obrázek 5.14: Poměr chyby stromu SDT pro různé metody hledání optimálního parametru β. N - naivní algoritmus, E - náš algoritmus první od konce, F - Fibonacciho prohledávání.

chyba výrazně horší. Vzhledem k přesnosti je víceméně jedno, který algoritmus použijeme. Pro malou hodnotu prahu pro počet vzorů v listech byly získány nejpřesnější stromy použitím našeho algoritmu POK. POK preferuje šířku $\beta > 0$. Ovšem zlepšení oproti zbylým dvěma algoritmům nebylo výrazné. Na obr. 5.15 je znázorněna závislost složitosti SDT stromu na prahu pro minimální počet vzorů v listech. Podle tohoto kritéria byl nejlepší naivní algoritmus. Tento algoritmus preferuje ostré dělení, protože pro menší počet vzorů v uzlu je chyba v uzlu menší pro $\beta = 0$ (obr. 5.11). Tím dojde k disjunktnímu rozdělení vzorů a do vrcholu v nižší úrovní stromu „propadne“ menší počet vzorů. Tím se počty vzorů ve vrcholech v nižších úrovní snižují rychleji, než je tomu u vrcholů s větší přechodovou oblastí, jejichž $\beta > 0$.

Ze všech testů však vychází z hlediska časové složitosti nejlépe Fibonacciho prohledávání, které konverguje k minimu mnohem rychleji než zbývající dva algoritmy a je tedy třeba k výpočtu menší počet kroků. Co se týká přesnosti klasifikace, Fibonacciho algoritmus mírně předčí ve většině případů zbylé dva. Volba algoritmu pro hledání optimálního parametru β však není kritická. SDT strom dává dobré výsledky pro všechny popsané metody. Pokud by nalezené minimum nebylo dostačující, je možné použít některou z nelineárních optimalizačních metod [29], které jsou přesnější, avšak jsou časově náročnější.

5.5 SDT - ořezávání

Ořezávání SDT stromu bylo teoreticky popsáno v kapitole 4.4. V této kapitole provedeme srovnání výsledků SDT stromů před a po ořezání. Budeme opět



Obrázek 5.15: Složitost stromu SDT pro různé metody hledání optimálního parametru β. N - naivní algoritmus, E - náš algoritmus první od konce, F - Fibonacciho prohledávání.

zkoumat vliv ořezání na přesnost a složitost stromu. K jednotlivým experimentům jsme použili vzory databáze OMIB.

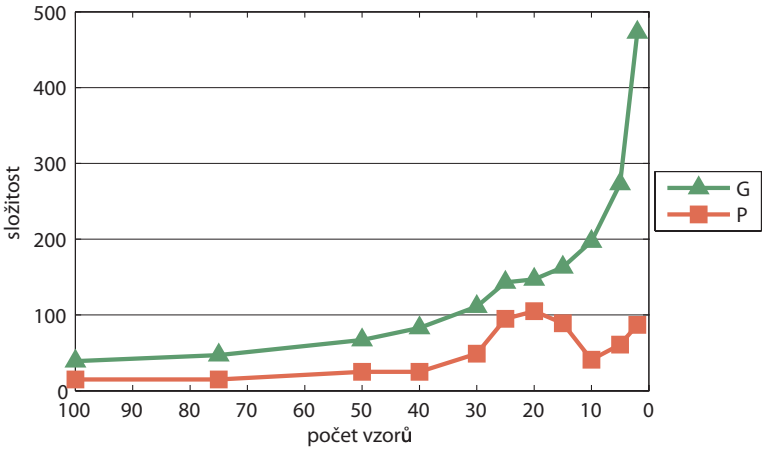
Polovina vzorů byla použita na růst stromu a polovina na odhadnutí chyby MAE na posloupnosti podstromů (viz kapitola 4.4). Na obr. 5.16 je srovnání složitostí SDT stromu před a po ořezání. Na vodorovné ose jsou hodnoty prahu pro minimální počet vzorů v listech. Na svislé ose je složitost stromu. Můžeme si všimnout, že došlo k drastickému snížení složitosti stromu. Čím větší byla složitost původního stromu, tím větší byla míra ořezání.

V případě přesnosti stromu byly výsledky o něco horší než v případě stromu před ořezáním viz obr. 5.17. Čím však víc hodnota prahu klesala, tím byl rozdíl v přesnosti stromu a před a po ořezání menší. Pro velké hodnoty prahu strom není ještě dostatečné velký a ořezávání má pak negativní vliv na přesnost stromu. Čím je však strom větší, tím se také budují nadbytečné části stromu, které nezlepší a ani nezhorší přesnost. Tyto části ořezávací algoritmus detekuje a odstraní.

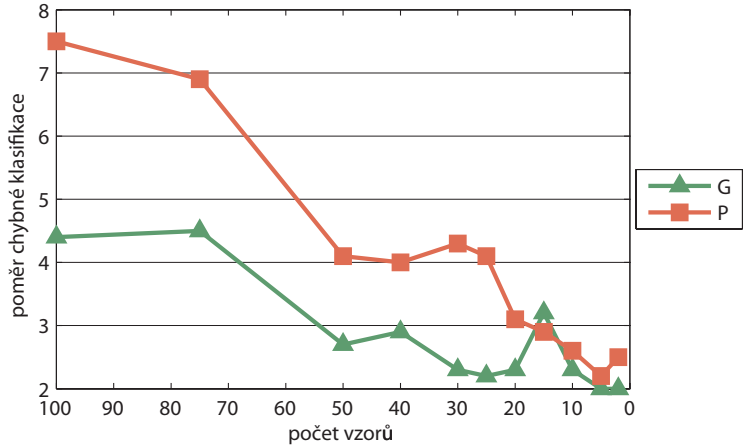
Ořezávání SDT stromů výrazně redukovalo složitost stromu, jak je patrné z výsledků experimentů. Ořezávání je dobré používat na maximálně velkých stromech, jejichž růst není předčasně zastaven ukončovací podmínkou. Ořezání takovýchto maximálně velkých stromů se projeví velkou redukcí složitosti stromu a zachováním jeho přesnosti.

5.6 SDT - refitting

Refitting je jedna z ladících procedur parametrů stromu SDT. Cílem experimentů této kapitoly je pozorovat chování metody refitting pro různé hodnoty



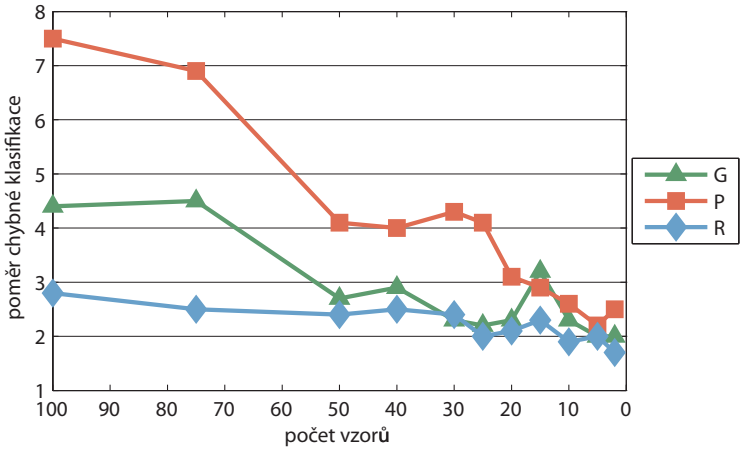
Obrázek 5.16: Srovnání složitostí stromů SDT před a po ořezání v závislosti na prahu pro minimální počet vzorů v listech. G je neořezaný strom a P je ořezaný strom.



Obrázek 5.17: Srovnání poměrů chybné klasifikace stromů SDT před a po ořezání v závislosti na prahu pro minimální počet vzorů v listech. G je neořezaný strom a P je ořezaný strom.

prahu pro minimální počet vzorů v listech.

Refitting již nemění strukturu SDT stromu, pouze provede změnu parametrů listů stromu. K tomu využívá metody nejmenších čtverců. Detaily jsou popsány v teoretické části (viz kapitola 4.5.1). V našich experimentech jsme testovali přesnost SDT stromu po provedení procedury refitting v závislosti na prahu pro minimální počet vzorů v listech. Jako testovací množinu jsme použili OMIB databázi vzorů. Výsledek je znázorněn na obr. 5.18, ve kterém je možné vidět tři grafy. Grafy znázorňují přesnost stromu před ořezáním stromu, přesnost po jeho ořezání a přesnost po provedení procedury refitting.



Obrázek 5.18: Srovnání poměrů chybné klasifikace stromu SDT před, po ořezání a po provedení refittingu na ořezaný strom v závislosti na prahu pro minimální počet vzorů v listech. *G* je neořezaný strom, *P* je ořezaný strom a *R* je ořezaný strom po refittingu.

Výsledek vyšel podle očekávání. Refitting zlepšil přesnost klasifikace. Např. pro práh 75 byla přesnost lepší o 2% než pro neořezaný strom a lepší dokonce o 4,4% než pro ořezaný strom. Přesnost ořezaného stromu bez použití refittingu byla výrazně horší pro vyšší hodnoty prahu. Pro práh 5 byla chyba ořezaného stromu jen 2,2%, zatímco pro hodnotu prahu 100 byla chyba až 7,5%. Pro vyšší hodnoty prahu je růst stromu předčasně ukončen a ořezání takového stromu má potom špatný vliv na jeho přesnost. Čím víc se hodnota prahu zmenšovala, tím se chyba ořezaného stromu bez použití refittingu blížila chybě neořezaného stromu. Potom, co jsme na ořezaný strom aplikovali refitting, situace se výrazně zlepšila zvláště pro vysoké hodnoty prahu. Pro 100 vzorů byl rozdíl mezi přesností vybudovaného neořezaného stromu a stromu po ořezání a refittingu 1,6%. Pro menší hodnoty prahu se rozdíly mezi chybami zmenšovaly.

Zavedením procedury refitting došlo k výrazné redukci špatně klasifikovaných vzorů zvláště pro vyšší hodnoty prahu pro minimální počet vzorů

v listech. Rozdíly mezi přesností stromů s malou složitostí a stromů s velkou složitostí se zmenšily. Nevýhodou metody refitting jsou jeho časové nároky na výpočet. Dobu provádění procedury refitting a ostatních metod pro budování rozhodovacích stromů jsme zkoumali v kapitole 5.8.

5.7 SDT - backfitting

Backfitting je metoda, která optimalizuje všechny parametry stromu pomocí Levenberg-Marquardtovy nelineární metody. K tomu je třeba spočíst derivace funkce podle jednotlivých parametrů. V tomto případě je funkcí odhad stromu $\hat{\mu}_C(x)$. V této části práce zpřesníme výpočet parciálních derivací. Během experimentů jsme zjistili, že tuto optimalizační techniku není možné použít obecně. Podařilo se nám dokonce dokázat, že Levenberg-Marquardtovu optimalizační techniku nelze obecně použít na všechny SDT stromy.

5.7.1 Derivace diskriminační funkce

Autoři v článku [28] v části *Backfitting* popisují neúplně výpočet derivací funkce odhadu $\hat{\mu}_C(x)$ podle parametrů stromu α_i a β_i ve vnitřních vrcholech. V článku [28] popisují pouze výpočet derivací $\partial\hat{\mu}_C(x)/\partial\nu_k$ (viz kapitola 4.5.2). V dalším textu se budeme zabývat výpočtem chybějících derivací detailněji. U zpětné fáze se pro výpočet derivace $\partial\hat{\mu}_C(x)/\partial\nu_k$ vrcholu u_k propagují derivace $\partial\hat{\mu}_C(x)/\partial\mu_{S_i}$ a $\partial\hat{\mu}_C(x)/\partial\mu_{S_j}$ z levého potomka u_i a pravého potomka u_j (viz vztah 4.35):

$$\frac{\partial\hat{\mu}_C(x)}{\partial\nu_k} = \mu_{S_k}(x)\left(\frac{\partial\hat{\mu}_C(x)}{\partial\mu_{S_i}(x)} - \frac{\partial\hat{\mu}_C(x)}{\partial\mu_{S_j}(x)}\right). \tag{5.1}$$

Výpočet derivací $\partial\hat{\mu}_C(x)/\partial\nu_k$ však není postačující, je nutno spočíst $\partial\hat{\mu}_C(x)/\partial\alpha_k$ a $\partial\hat{\mu}_C(x)/\partial\beta_k$ (viz vztahy 4.33 a 4.34). Podle pravidla o derivování složené funkce platí:

$$\begin{aligned}\frac{\partial\hat{\mu}_C(x)}{\partial\alpha_k} &= \frac{\partial\hat{\mu}_C(x)}{\partial\nu_k}\cdot\frac{\partial\nu_k}{\partial\alpha_k},\\ \frac{\partial\hat{\mu}_C(x)}{\partial\beta_k} &= \frac{\partial\hat{\mu}_C(x)}{\partial\nu_k}\cdot\frac{\partial\nu_k}{\partial\beta_k}.\end{aligned} \tag{5.2}$$

Tedy je třeba zjistit derivace $\partial\nu_k/\partial\alpha_k$ a $\partial\nu_k/\partial\beta_k$. Tyto derivace se nyní pokusíme odvodit. BÚNO přeznačíme hledané derivace $\partial\nu(\alpha,\beta)/\partial\alpha$ a $\partial\nu(\alpha,\beta)/\partial\beta$. Diskriminační funkce je obecně po částech lineární funkce. Každá po částech lineární funkce se dá zapsat jako:

$$f(x) = a_0 + \sum_{i=1}^{n} a_i|x - b_i|, \tag{5.3}$$

kde b_i jsou body nespojitosti a platí $b_1 < b_2 < \cdots < b_n$. Pro úplnost definujme $b_0 = -\infty$ a $b_{n+1} = +\infty$. Funkce f je lineární (a tedy spojitá) na intervalech $\langle b_i, b_{i+1}\rangle, i = 0, 1 \ldots, n$.

Tvar diskriminační funkce SDT stromů je dán parametry α a β. Hodnota diskriminační funkce je 1 pro $x < \alpha - \beta/2$ a 0 pro $x > \alpha + \beta/2$. Pro $x \in \langle \alpha - \beta/2, \alpha + \beta/2\rangle$ je hodnota diskriminační funkce rovna $(\alpha - x)/\beta + 0{,}5$. Diskriminační funkce je nespojitá v bodech $b_1 = \alpha - \frac{\beta}{2}$ a $b_2 = \alpha + \frac{\beta}{2}$. Diskriminační funkce má tvar:

$$\nu(\alpha, \beta, x) = \frac{1}{2} - \frac{1}{2\beta}\left|x - \alpha + \frac{\beta}{2}\right| + \frac{1}{2\beta}\left|x - \alpha - \frac{\beta}{2}\right|, \tag{5.4}$$

kde α je bod dělení, β šířka a x označuje hodnotu atributu pro daný vzor. Pro šířku β musí být splněn předpoklad $\beta \neq 0$. Algoritmus pro budování SDT stromu nám zajistí, že $\beta \geq 0$. Pro $\beta = 0$ definujeme diskriminační funkci jako:

$$\nu(\alpha, \beta, x) = \frac{1}{2}\left[1 - \operatorname{sgn}(x - \alpha)\right], \tag{5.5}$$

která je nespojitá v bodech $x - \alpha = 0$. Derivace diskriminační funkce ze vztahu 5.5 podle β je 0. Derivace podle α je v bodech spojitosti 0 a v bodě $x - \alpha = 0$ má jednostranné derivace rovny 0. V tomto bodě se dodefinujeme derivace nulou.

Derivace diskriminační funkce s nenulovou hodnotou šířky $\beta > 0$ odvodíme v následujících dvou kapitolách.

5.7.2 Derivace diskriminační funkce podle α

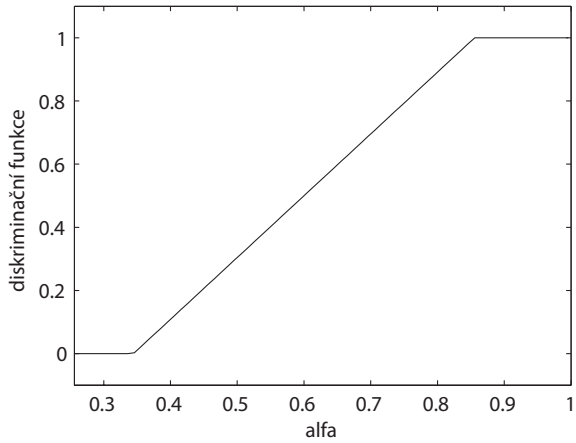
Nyní popíšeme derivaci diskriminační funkce $\partial\nu/\partial\alpha$. Na obr. 5.19 je znázorněn průběh diskriminační funkce (vztah 5.4) pro různé hodnoty dělení α ($\alpha \in \langle 0{,}256; 1\rangle$), pevnou hodnotu šířky β ($\beta = 0{,}511$) a pevnou hodnotu atributu x ($x = 0{,}6$). Funkce je po částech lineární, která má nenulovou derivaci v intervalu $\langle 0{,}3445, 0, 8555\rangle$. V dalším textu zobecníme výpočet hranic tohoto intervalu a také spočítáme konkrétní tvar derivace diskriminační funkce podle parametru α.

Pro přehlednost zápisu označme:

$$c_1 = \frac{1}{2}, c_2 = \frac{1}{2\beta}, c_3 = x + \frac{\beta}{2}, c_4 = x - \frac{\beta}{2} \tag{5.6}$$

Z uvedeného je patrné, že $c_4 < c_3$, protože $\beta > 0$. Diskriminační funkce pak po dosazení vypadá takto:

$$\nu(\alpha) = c_1 - c_2|c_3 - \alpha| + c_2|c_4 - \alpha|$$



Obrázek 5.19: Průběh diskriminační funkce ν pro $\alpha \in \langle 0{,}256, 1\rangle$, pevné $\beta = 0{,}511$ a $a(x) = 0{,}6$.

Nyní ukážeme, jak se chová derivace diskriminační funkce ν podle α na intervalech $(-\infty, c_4)$, (c_4, c_3) a (c_3, ∞).

(i): $\alpha \in (-\infty, c_4)$

$$\begin{aligned} \nu(\alpha) &= c_1 - c_2(c_3 - \alpha) + c_2(c_4 - \alpha) \\ \frac{\partial \nu(\alpha)}{\partial \alpha} &= c_2 - c_2 = 0 \end{aligned} \tag{5.7}$$

(ii): $\alpha \in (c_4, c_3)$

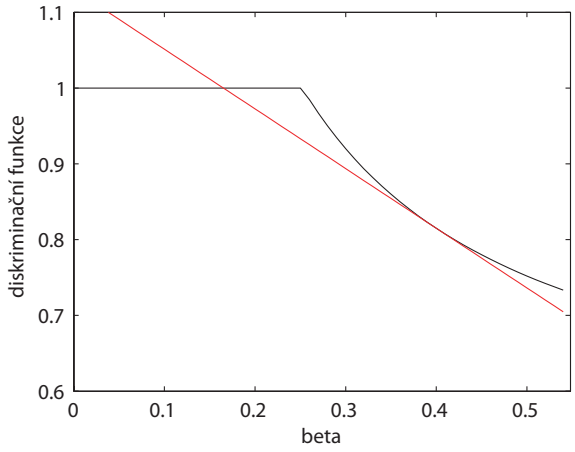
$$\begin{aligned} \nu(\alpha) &= c_1 - c_2(c_3 - \alpha) + c_2(\alpha - c_4) \\ \frac{\partial \nu(\alpha)}{\partial \alpha} &= c_2 + c_2 = 2c_2 = \frac{1}{\beta} \end{aligned} \tag{5.8}$$

(iii): $\alpha \in (c_3, \infty)$

$$\begin{aligned} \nu(\alpha) &= c_1 + c_2(c_3 - \alpha) + c_2(\alpha - c_4) \\ \frac{\partial \nu(\alpha)}{\partial \alpha} &= -c_2 + c_2 = 0 \end{aligned} \tag{5.9}$$

Pro $\alpha = c_4$ a $\alpha = c_3$ dodefinujeme derivace nulou, protože jednostranné derivace diskriminační funkce ν podle α jsou zleva (pro $\alpha = c_4$) a zprava (pro $\alpha = c_3$) nulové.

Tímto jsme spočítali derivace diskriminační funkce ν podle α. Nenulovou hodnotu derivace dostaneme pro $\alpha \in (x - \beta/2, x + \beta/2)$.



Obrázek 5.20: Průběh diskriminační funkce pro $\beta \in \langle 0, 0{,}548\rangle$ pro pevné $\alpha = 0{,}726$, $x = 0{,}6$. Červená přímka je tečna diskriminační funkce v bodě $\beta = 0{,}4$.

5.7.3 Derivace diskriminační funkce podle β

Nyní se zaměříme na výpočet derivace diskriminační funkce ν podle β. Na obr. 5.20 je znázorněn průběh diskriminační funkce pro parametr $\beta \in \langle 0; 0{,}548\rangle$ pro pevné dělení α a hodnotu atributu vzoru $x = 0{,}6$. Na obrázku si můžeme všimnout, že pro $\beta \in \langle 0; 0{,}252\rangle$ je přímka rovnoběžná s vodorovnou osou a v tomto intervalu bude derivace podle β nulová. Pro $\beta \in \langle 0{,}252; 0{,}548\rangle$ je diskriminační funkce hladká křivka, jejíž tečna není rovnoběžná s vodorovnou osou a tedy derivace diskriminační funkce podle β je různá od 0 a konkrétně menší než 0. Dále derivaci diskriminační funkce podle β zobecníme pro libovolné pevné α a libovolnou pevnou hodnotu atributu x.

Rovnici diskriminační funkce (vztah 5.4) upravíme na tvar:

$$\nu(\beta) = \frac{1}{2} - \frac{1}{4\beta}\left|2(x-\alpha) + \beta\right| + \frac{1}{4\beta}\left|2(x-\alpha) - \beta\right|.$$

Opět pro zjednodušení zápisu zavedeme značení

$$d_1 = \frac{1}{2}, d_2 = \frac{1}{4}, d_3 = 2(x-\alpha). \tag{5.10}$$

Diskriminační funkce ve zjednodušeném zápisu vypadá takto

$$\nu(\beta) = d_1 - d_2\frac{1}{\beta}\left|d_3 + \beta\right| + d_2\frac{1}{\beta}\left|d_3 - \beta\right|. \tag{5.11}$$

Z výše uvedeného je třeba brát v úvahu to, že d_3 může být kladné i záporné číslo. Nyní budeme diskutovat intervaly, na kterých má diskriminační funkce ν derivace podle β. Předpokládejme, že d_3 je kladné:

(i): $\beta \in (0, d_3)$

$$\begin{aligned} \nu(\beta) &= d_1 - \frac{d_2}{\beta}(d_3 + \beta) + \frac{d_2}{\beta}(d_3 - \beta) = d_1 - 2d_2 = 0 \\ \frac{\partial \nu(\beta)}{\partial \beta} &= 0 \end{aligned} \tag{5.12}$$

(ii): $\beta \in (d_3, \infty)$

$$\begin{aligned} \nu(\beta) &= d_1 - \frac{d_2}{\beta}(d_3 + \beta) - \frac{d_2}{\beta}(d_3 - \beta) \\ \frac{\partial \nu(\beta)}{\partial \beta} &= \frac{d_2 d_3}{\beta^2} + \frac{d_2 d_3}{\beta^2} = \frac{2 d_2 d_3}{\beta^2} = \frac{x - \alpha}{\beta^2} \end{aligned} \tag{5.13}$$

V bodě $\beta = d_3$ je derivace zleva nulová. Proto ji pro úplnost dodefinujeme

$$\frac{\partial \nu(\beta)}{\partial \beta} = 0, \quad \text{pro } \beta = d_3. \tag{5.14}$$

Pro záporné d_3 je diskriminační funkce ν derivovatelná pro $\beta \in (0, -d_3)$ a pro $\beta \in (-d_3, \infty)$. Derivace diskriminační funkce jsou totožné jako v předchozím případě:

(i): $\beta \in (0, -d_3)$

$$\begin{aligned} \nu(\beta) &= d_1 + \frac{d_2}{\beta}(d_3 + \beta) - \frac{d_2}{\beta}(d_3 - \beta) = d_1 + 2d_2 = 0 \\ \frac{\partial \nu(\beta)}{\partial \beta} &= 0 \end{aligned} \tag{5.15}$$

(ii): $\beta \in (-d_3, \infty)$

$$\begin{aligned} \nu(\beta) &= d_1 - \frac{d_2}{\beta}(d_3 + \beta) - \frac{d_2}{\beta}(d_3 - \beta) \\ \frac{\partial \nu(\beta)}{\partial \beta} &= \frac{d_2 d_3}{\beta^2} + \frac{d_2 d_3}{\beta^2} = \frac{2 d_2 d_3}{\beta^2} = \frac{x - \alpha}{\beta^2} \end{aligned} \tag{5.16}$$

V bodě $\beta = -d_3$ je derivace zleva nulová. Proto se pro úplnost dodefinuje

$$\frac{\partial \nu(\beta)}{\partial \beta} = 0, \quad \text{pro } \beta = -d_3. \tag{5.17}$$

Nechť d_3 je nula.

$$\begin{aligned} \nu(\beta) &= d_1 - d_2 - d_2 \\ \frac{\partial \nu(\beta)}{\partial \beta} &= 0 \end{aligned} \tag{5.18}$$

Spočítali jsme, že nenulovou derivaci získáme pro $\beta \in (2(x-\alpha), \infty)$, což je také patrné z obr. 5.20. „Bod zlomu“ odpovídá hodnotě $\beta = |d_3| = |2(x-\alpha)|$. Do bodu $|d_3|$ je diskriminační funkce ν konstantní a derivace je tedy nulová a pro hodnoty $\beta > |d_3|$ je derivace diskriminační funkce ν nenulová.

5.7.4 Vlastnosti diskriminační funkce

Derivace diskriminační funkce podle parametrů α a β zásadně ovlivňují derivace $\partial\hat{\mu}_C(x)/\partial\alpha$ a $\partial\hat{\mu}_C(x)/\partial\beta$, které se používají v Levenberg-Marquardtově algoritmu. Podařilo se nám dokázat, že základní verzi Levenberg-Marquardtova nelineárního optimalizačního algoritmu [29] aplikovanou na parametry SDT stromu, popsanou v [28], není možné obecně použít na všechny stromy. Tento náš důkaz prezentujeme v této kapitole.

Tvrzení 5.7.1 *Mějme* SDT *strom* $G = (V, E)$ *a vrchol* $u \in V(G)$ *stromu* SDT*, o vzor, který se dostane do vrcholu* u*. Dále nechť dělení* α *a šířka* β *jsou parametry odpovídající vrcholu* u *a* a *je nejlepší atribut vrcholu* u*. Označme* $x = a(o)$*. Pokud po částech lineární diskriminační funkce* $\nu(x, \alpha, \beta) = 0$ *nebo* $\nu(x, \alpha, \beta) = 1$*, pak:*

$$\frac{\partial\nu}{\partial\alpha}(x, \alpha, \beta) = 0 \text{ a zároveň } \frac{\partial\nu}{\partial\beta}(x, \alpha, \beta) = 0.$$

Důkaz. Parametry α i β jsou nezáporné díky procesu růstu stromu SDT. Tato vlastnost zůstává nezměněna i ve fázi ořezávání, která pouze odstraní některé podstromy a na hodnoty těchto parametrů nemá vliv.

Důkaz rozdělíme na dvě části. Nejprve dokážeme implikaci:

$$\nu = 0 \vee \nu = 1 \Rightarrow \frac{\partial\nu}{\partial\alpha} = 0,$$

následně pak dokážeme implikaci:

$$\nu = 0 \vee \nu = 1 \Rightarrow \frac{\partial\nu}{\partial\beta} = 0.$$

(i): Nyní dokážeme první implikaci. Označme $c_3 = x + \beta/2$ a $c_4 = x - \beta/2$ stejně jako ve vztahu 5.6. V popisu derivace diskriminační funkce podle dělení α (viz kapitola 5.7.2) jsme odvodili, že $\partial v/\partial\alpha = 0$, právě když $\alpha \leq c_4$ nebo $\alpha \geq c_3$.

Diskriminační funkce je po částech lineární funkce. Tedy pokud $\nu = 0$, pak musí platit nerovnost:

$$x \geq \alpha + \frac{\beta}{2}.$$

Po úpravách dostaneme nerovnost $\alpha \leq x - \beta/2$, která odpovídá nerovnosti $\alpha \leq c_4$. Tedy víme, že derivace $\partial\nu/\partial\alpha$ je nulová.

Pokud je diskriminační funkce ν rovna jedné ($\nu = 1$), pak podle tvaru po částech lineární diskriminační funkce dostaneme nerovnost:

$$x \leq \alpha - \frac{\beta}{2}.$$

Po úpravách získáme nerovnost $\alpha \geq x+\beta/2$, která odpovídá nerovnosti $\alpha \geq c_3$. Tedy i pro tento případ je derivace $\partial\nu/\partial\alpha$ nulová, což jsme chtěli dokázat.

(ii): Nyní dokážeme druhou implikaci. Označme $d_3 = 2(x - \alpha)$ stejně jako ve výrazu 5.10. Derivace diskriminační funkce podle β je nulová pro $0 \leq \beta \leq |d_3|$ (viz kapitola 5.7.3).

Pro $\nu = 0$ leží x vpravo od $\alpha + \beta/2$ a tedy platí nerovnost:

$$x \geq \alpha + \frac{\beta}{2} \tag{5.19}$$

Úpravou uvedené nerovnosti dostaneme $2(x - \alpha) \geq \beta$. A protože víme, že β je nezáporné ($\beta \geq 0$), pak musí být $2(x - \alpha) \geq 0$. Díky této nerovnosti můžeme položit $|d_3|$ rovnu $2(x-\alpha)$. Po úpravách nerovnosti 5.19 dostáváme $\beta \leq 2(x-\alpha)$ a tedy nerovnost $\beta \leq |d_3|$. Podle kapitoly 5.7.3 je derivace nulová.

Pokud $\nu = 1$, postupujeme obdobným způsobem. Pro x platí tato nerovnost:

$$x \leq \alpha - \frac{\beta}{2}. \tag{5.20}$$

Úpravou výše uvedené nerovnosti dostaneme tvar $\beta \leq 2(\alpha-x)$. Protože platí $\beta \geq 0$, pak $2(x - \alpha) \leq 0$. Po úpravách nerovnosti 5.20 je $\beta \leq -2(x - \alpha)$ a tedy $\beta \leq |d_3|$. Tedy i v tomto případě je derivace $\partial\nu/\partial\beta$ nulová.

Rozborem jednotlivých případů jsme dokázali, že pro $\nu = 0$ nebo $\nu = 1$ jsou derivace $\partial\nu/\partial\alpha$ a $\partial\nu/\partial\beta$ nulové.

□

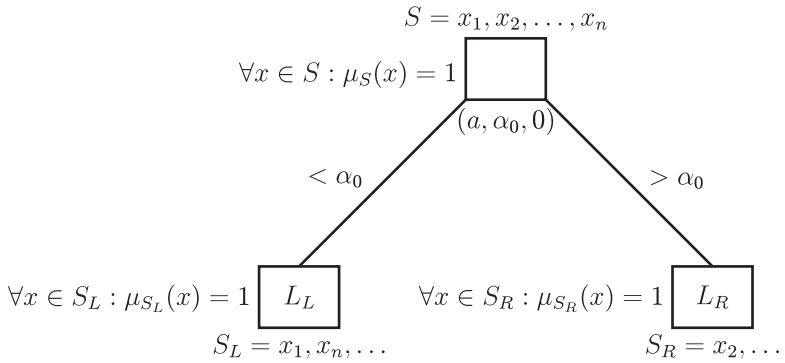
Výše uvedené tvrzení nám pomůže dokázat, že Hessiánská matice **A** (viz vztah 4.30) je někdy singulární a není možné ji vždy použít k výpočtu optimálních parametrů stromu algoritmu 4.1.

Tvrzení 5.7.2 *Mějme $q_1, q_2, \ldots, q_m$ parametry* SDT *stromu. Existuje strom* SDT*, jehož Hessiánská matice:*

$$A_{pl} = \sum_{x \in BS} \frac{\partial \hat{\mu_C}(x)}{\partial q_p} \cdot \frac{\partial \hat{\mu_C}(x)}{\partial q_l}, \quad p, l = 1 \ldots m \tag{5.21}$$

je singulární. BS označuje množinu vzorů určenou pro backfitting.

Důkaz. Uvažujme triviální ostrý SDT strom se třemi uzly na obr. 5.21. S je fuzzy množina odpovídající kořenovému uzlu stromu a je tvořena vzory



Obrázek 5.21: Triviální ostrý strom SDT se třemi uzly.

$x_1, \ldots, x_n$. Pro všechny vzory $x \in S$ v tomto uzlu je stupeň členství $\mu_S(x) = 1$. Díky tomu, že parametr $\alpha = \alpha_0, \alpha_0 \in (0, 1)$ a parametr $\beta = 0$ je diskriminační funkce ν rovna jedničce nebo nule (viz zavedení po částech lineární diskriminační funkce v kapitole 4.3). Pokud $\nu(a(x_i), \alpha, \beta) = 0$, pak vzor x_i je poslán do pravého uzlu. V případě, že $\nu(a(x_i), \alpha, \beta) = 1$, vzor x_i je poslán do levého uzlu. Abychom získali Hessiánskou matici $\mathbf{A}$, musíme určit derivace $\partial\hat{\mu}_C(x)/\partial\alpha$, $\partial\hat{\mu}_C(x)/\partial\beta$, $\partial\hat{\mu}_C(x)/\partial L_L$ a $\partial\hat{\mu}_C(x)/\partial L_R$. První dvě derivace budou nulové pro všechny vzory $x \in S$. Diskriminační funkce $\nu(a(x), \alpha_0, 0)$ je rovno 0 nebo 1 pro všechna $x \in S$. Podle tvrzení 5.7.1 je $\partial\nu/\partial\alpha(a(x), \alpha_0, 0) = 0$ a také $\partial\nu/\partial\beta(a(x), \alpha_0, 0) = 0$. Platí tedy:

$$\frac{\partial\hat{\mu}_C}{\partial\alpha} = \frac{\partial\hat{\mu}_C}{\partial\nu} \cdot \frac{\partial\nu}{\partial\alpha} = 0, \quad \frac{\partial\hat{\mu}_C}{\partial\beta} = \frac{\partial\hat{\mu}_C}{\partial\nu} \cdot \frac{\partial\nu}{\partial\beta} = 0 \tag{5.22}$$

Matice derivací parametrů triviálního SDT stromu vypadají takto:

	$\frac{\partial\hat{\mu}_C(x)}{\partial\alpha}$	$\frac{\partial\hat{\mu}_C(x)}{\partial\beta}$	$\frac{\partial\hat{\mu}_C(x)}{\partial L_L}$	$\frac{\partial\hat{\mu}_C(x)}{\partial L_R}$
x_1	0	0	1	0
x_2	0	0	0	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_n	0	0	1	0

Poslední dva sloupce budou obsahovat hodnotu 1 nebo 0. Pokud je derivace $\partial\hat{\mu}_C(x)/\partial L_L = 0$, pak derivace $\partial\hat{\mu}_C(x)/\partial L_R$ v posledním sloupci je 1 (vzor se dostal do pravého uzlu). A podobně pokud je derivace $\partial\hat{\mu}_C(x)/\partial L_L = 1$, pak derivace $\partial\hat{\mu}_C(x)/\partial L_R$ v posledním sloupci je 0 (vzor se dostal do levého uzlu). Tedy v řádku matice derivací se objeví pouze jedna hodnota 1. Pokud dosadíme hodnoty derivací do vztahu 4.30, dostaneme Hessiánskou matici:

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Tedy výsledná matice **A** je triviálně singulární. Dokázali jsme, že existuje strom, pro který je Hessiánská matice **A** singulární. □

Algoritmus 4.1 tedy pro každý ostrý strom SDT ztroskotá v okamžiku, kdy hledáme řešení lineární soustavy rovnic. Ostrý strom totiž obsahuje uzly, ve kterých diskriminační funkce nabývá hodnot 0 a 1 pro všechny vzory BS. Podle tvrzení 5.7.1 a kapitoly 5.7.1 jsou odpovídající sloupce matice derivací $\partial\hat{\mu_C}(x)/\partial\alpha$ a $\partial\hat{\mu_C}(x)/\partial\beta$ nulové pro všechny vzory $x \in BS$. Důsledkem toho je odpovídající sloupec a řádek Hessiánské matice **A** nulový. A díky tomu je matice **A** singulární. Nepomůže ani úprava prvků na diagonále $A_{pp} = A_{pp} \cdot (1 + \lambda)$ v LM algoritmu 4.1. Navíc strom nemusí mít nutně všechna dělení ostrá (šířka β je pro ostré dělení nulová). Postačující podmínkou pro to, aby Hessiánská matice **A** byla singulární, je aspoň jeden výskyt ostrého vnitřního uzlu (jehož parametr dělení $\beta = 0$) ve stromě SDT.

V kapitole jsme dokázali, že algoritmus LM není použitelný obecně na všechny stromy SDT. Pokud strom SDT obsahuje aspoň jeden ostrý vnitřní uzel, algoritmus selže. Při našich experimentálních pokusech byl výskyt takového ostrého vnitřního uzlů častým jevem a metodu backfitting tedy nebylo možné otestovat.

5.8 Srovnání metod

Na závěr jsme srovnávali všechny metody budování stromů. Stromy CART, C4.5 a také stromy SDT jsme srovnávali podle přesnosti klasifikace na neznámých vzorech, podle složitosti vybudovaných stromů a také podle času potřebného pro vybudování klasifikátorů.

Výsledky byly získány desetidílnou cross validací. K testování jsme použili obě množiny vzorů, OMIB i znaky abecedy. Každá množina byla stratifikovaně rozdělena cross validací na 10 stejně velkých dílů. V každém kroku cross validace bylo 9 dílů použito k učení stromu a 1 díl k testování stromu. Množina vzorů LS pro učení byla dále stratifikovaně rozdělena na 2 stejně velké množiny. Pomocí první množiny byl vybudován kompletní strom a druhá množina sloužila k ořezávání. Tedy $|GS| = |PS|$, kde GS označuje množinu vzorů, pomocí které jsme budovali strom, a PS je množina ořezávacích vzorů. Tímto způsobem jsme testovali stromy C4.5 a SDT. U stromu CART je situace trochu jiná, protože se nejprve vytvoří sada stromů a na nich se pomocí cross validace odhaduje chyba. Množinu vzorů LS k učení jsme volili tak velkou, aby při desetidílné cross validaci byla množina vzorů GS k budování stromu stejně velká jako v případě stromu C4.5 a SDT. Tedy pokud $|LS| = 100$, pak množina všech vzorů pro C4.5 a SDT obsahovala 112 vzorů, protože 9/10 ze 112 vzorů odpovídá přibližně 100 vzorů. Množina vzorů LS byla dále stratifikovaně rozdělena na poloviny. První polovina vzorů byla použita pro růst (množina GS) a druhá polovina vzorů byla použita na oře-

závání (množina PS). Tedy pokud LS obsahuje 100 vzorů, pak množiny GS a PS obsahují 50 vzorů ($|GS| = |PS| = 50$). Odpovídající test CART byl spuštěn na množině vzorů, která obsahovala 56 vzorů a tedy $|GS| = 50$.

V tabulce 5.9 jsme shrnuli získané složitosti a přesnosti jednotlivých metod. Složitostí rozumíme počet všech vrcholů jednoho ořezaného stromu vybudovaného v jednom kroku cross validace. Přesnost stromu je poměr počtu chybně klasifikovaných neznámých vzorů vzhledem k počtu všech neznámých vzorů. U všech metod jsme volili mírnou ukončovací podmínku a stromy jsme nechali vyrůst do maximální velikosti. V případě CART a C4.5 byl nastaven práh pro minimální počet vzorů v listech na hodnotu 2. U SDT jsme zvolili Fibonacciho prohledávání pro hledání optimální šířky β, které hledalo β ve 30 krocích. Minimální práh pro počet vzorů v listu byl nastaven na 2 vzory, práh pro minimální čtvercovou chybu byl nastaven na $E_S = 1 \cdot 10^{-4}$ a práh pro minimální redukci chyby byl nastaven na $\Delta E_S = 1 \cdot 10^{-2}$. SDT byl ořezán a parametry v listech doladěny ve fázi refittingu.

Složitost stromů se ukázala nejhorší pro SDT stromy, naopak nejlépe vycházela pro CART. CART stromy totiž měly k dispozici menší množinu vzorů pro ořezávání a proto byly stromy CART více ořezány než stromy C4.5 a SDT. Rozdíl mezi složitostí stromů jednotlivých metod se zvyšoval s rostoucí velikostí množiny GS, jak je patrné z tabulky 5.9. Velký rozdíl mezi složitostí SDT a zbylými dvěma metodami byl způsoben přechodovou oblastí SDT stromů. Díky ní může vzor „propadnout“ do každého poduzlu a tím nedojde k takové redukci velikosti množiny vzorů uzlu jako u stromů CART a C4.5. Důsledkem toho dochází u SDT k dalšímu dělení uzlů a tím k dalšímu rozrůstání stromu. Dále si můžeme všimnout, že složitost stromů pro stejně velké množiny vzorů nebyla u databáze OMIB a u databáze písmen stejná. Např. strom SDT pro 250 vzorů databáze písmen obsahoval 83 vrcholů, v případě OMIB databáze obsahoval pouze 23 vrcholů. Lepší složitost ve prospěch OMIB databáze nám vycházela i pro ostatní metody. Shluky vzorů OMIB databáze jsou snáze separabilní (viz obr. 5.2) a obsahují pouze dva atributy. Stromy obecně potřebují méně rozhodnutí na to, aby správně oddělily jednotlivé shluky vzorů označených jednou třídou. To samé nelze říci o vzorech databáze písmen. Shluky vzorů písmen se překrývají. Nalezení optimálního rozdělení pro shluky vzorů písmen je poměrně náročnější a díky tomu je třeba vytvořit větší strom.

Výsledky experimentů ukázaly nejlepší přesnost klasifikace u SDT stromů. Např. pro databází písmen obsahující 1000 vzorů je SDT lepší o 3,28% než C4.5 a dokonce o 5,71% lepší než CART. U OMIB databáze jsou výsledky přesnosti ještě optimističtější. Pro $|GS| = 450$ jsme dosáhli hodnoty 1,6%. Pro malé velikosti GS byly stromy nestálé. Např. chyba SDT stromů na databázi písmen pro $|GS| = 50$ byla jen 26,79 % a chyba pro $|GS| = 100$ se zvýšila na 30,94%. Tyto výkyvy přesnosti se však s přibývajícím počtem vzorů již neobjevovaly. Čím byla množina GS větší, tím byly klasifikátory

	$\|GS\|$	Složitost			Přesnost		
		CART	C4.5	SDT	CART	C4.5	SDT
Písmena	50	7	9	11	32,14%	33,04%	26,79%
	100	21	23	37	25,00%	26,91%	30,94%
	250	11	27	83	24,46%	24,64%	22,30%
	500	29	69	191	23,02%	22,39%	18,80%
	1000	65	145	453	18,71%	16,28%	13,00%
	1500	69	207	615	16,14%	15,09%	10,53%
	2250	109	253	1227	12,92%	12,42%	8,9%
OMIB	50	7	9	11	7,14%	10,71%	6,25%
	100	7	9	29	14,29%	5,86%	4,5%
	150	11	11	37	6,59%	4,2%	2,7%
	200	11	11	53	6,31%	2,93%	2,93%
	250	11	11	23	5,76%	5,2%	3%
	450	19	25	93	3,0%	3,2%	1,6%

Tabulka 5.9: Srovnání metod.

přesnější.

SDT stromy dávají lepší výsledky než ostré stromy (CART a C4.5) díky přidání přechodové oblasti. V ostrých stromech byly získány horší výsledky hlavně pro vzory s hodnotami atributu blízkými hodnotě dělení, což bylo zjištěno v [4, 6, 12]. Dva vzory, které jsou navzájem blízko v prostoru atributů, mohou být rozděleny do různých větví a tím umístěny „daleko“ od sebe v prostoru výstupů (tříd). Naopak pokud jsou blízko sebe vzory v prostoru atributů, jdou v SDT stromu oba dva stejnými větvemi. SDT tímto způsobem snižuje odchylku v okolí ohraničení oblastí a tím ovlivňuje chyby stromu. SDT stromy jsou z toho důvodu také méně citlivé na zašuměná data než je tomu v případě ostrých stromů. Díky přechodové oblasti má šum menší vliv na obecnost klasifikátoru.

Rekurzivní dělení v ostrých stromech rychle snižuje počty množin vzorů vrcholů tím, jak se postupuje směrem k listům. Lokální rozhodnutí mohou být příliš specifická pro konkrétní množinu vzorů, což vede k přetrénování. V SDT nedochází k redukci a lokální množiny vzorů obsahují i takové vzory, které nenáleží striktně do těchto množin. Tím jsou lokální rozhodnutí stromu SDT více stabilnější, protože byly vytvořeny na základě více vzorů. Čím je větší přechodová oblast ve vrcholu, tím více vzorů lokální množiny pro růst se dostane do jeho následníků. Na obr. 4.1(b) můžeme vidět, že do vrcholu v_7 SDT stromu „propadne“ 61 vzorů, zatímco odpovídajícímu vrcholu v_3 CART stromu na obr. 4.1(a) náleží pouze 49 vzorů.

V posledním experimentu jsme srovnávali dobu běhu jednotlivých algoritmů. Naměřené výsledky jsou shrnuty v tabulce 5.10. Všechny testy zkoumaných metod jsme prováděli na počítači s procesorem AMD Athlon XP

	Velikost $\|GS\|$						
	50	100	250	500	1000	1500	2000
SDT(G)	62 ms	94 ms	188 ms	266 ms	1,72 s	1,03 s	2,91 s
SDT(P)	16 ms	47 ms	78 ms	94 ms	890 ms	687 ms	1,63 s
SDT(R)	31 ms	46 ms	78 ms	359 ms	15,64 s	38,06 s	1 m 42 s
SDT	109 ms	187 ms	344 ms	719 ms	18,25 s	39,78 s	1 m 46 s
C4.5	60 ms	80 ms	110 ms	160 ms	330 ms	240 ms	690 ms
CART(G)	46,9 ms	46,9 ms	125 ms	266 ms	500 ms	656 ms	906 ms
CART(P)	437,5 ms	562,5 ms	1,27 s	2,45 s	4,61 s	6,33 s	8,34 s
CART	484,4 ms	609,4 ms	1,39 s	2,72 s	5,11 s	6,98 s	9,25 s

Tabulka 5.10: Srovnání časů, po který běžely jednotlivé algoritmy. Písmena v závorce mají tento význam: G - růstová fáze, P - ořezávací fáze, R - refitting.

(1,67GHz), pamětí 768MB RAM a operačním systémem Windows XP Professional. V testu nejlépe dopadl algoritmus C4.5, který i pro množinu obsahující 2000 vzorů skončil i s ořezáním za méně než 1 s. Naopak nejhůře dopadl SDT algoritmus, který je ovšem přesnější než CART a C4.5. Pro strom CART je časově náročná fáze ořezávání, protože pro každý strom je vygenerována posloupnost ořezaných stromů, na kterých se odhaduje chyba pomocí cross validace, která celý proces ořezávání výrazně zpomalí. Pro trénování SDT stromu je časově kritický refitting, který hledá optimální parametry listů metodou nejmenších čtverců, aby minimalizoval čtvercovou chybu. Tato operace je pro velké matice časově náročná. Pro $|GS| = 2000$ potřeboval refitting k nalezení optimálních parametrů až 1 min. a 42 s.

Kapitola 6

Závěr

V této kapitole zrekapitulujeme hlavní výsledky, které jsme dosáhli v diplomové práci. Naznačíme také možnosti dalšího vývoje a zlepšení.

6.1 Hlavní výsledky

Cílem práce bylo nastudování různých modelů rozhodovacích stromů a jejich rešerše a experimentální ověření. V práci jsme zkoumali rozhodovací stromy C4.5, CART a SDT. Jednotlivé stromy jsme porovnávali z hlediska přesnosti, velikosti a také z hlediska časových nároků potřebných pro jejich vybudování. U modelů jsme se zaměřili zejména na různé metody budování stromů a ořezávací metody.

U CART stromů jsme srovnávali použití poklesu nečistoty a twoing kritéria. Zjistili jsme, že velikost ani přesnost stromu se na použitých datových množinách výrazně nelišila. Ořezání metodou MCCP mělo největší vliv na stromy, které byly vybudovány na velkých množinách vzorů. Chyba optimálního ořezaného stromu se přibližovala stromu z posloupnosti s nejmenší odhadnutou chybou. Pravidlo b-SE totiž pro velké množiny preferuje stromy posloupnosti s minimální chybou. Od určité velikosti stromů v posloupnosti se chyba již výrazně neměnila, protože strom již byl dostatečně přesný a další rozrůstání již nemělo vliv na změnu přesnosti stromu. Ořezávání metodou MEP je závislé na zvolené hodnotě m (pro m-odhad). Pro velké hodnoty parametru m se přestala měnit velikost i přesnost ořezaného stromu. Tento jev je způsoben vlastnostmi m-odhadu, který pro velké hodnoty m konverguje k apriorní pravděpodobnosti dané třídy. Přesnost ořezaných stromů metodou MEP se příliš nezhoršila i pro stromy s velkou mírou ořezání.

U stromu C4.5 jsme se zaměřili na ořezávací metody REP a PEP. Ořezání metodou REP příliš nezlepšilo přesnost stromu. Pro menší stromy je chyba ořezaných stromů o něco horší, pro větší stromy je chyba ořezaných stromů mírně lepší. Pro větší množiny se totiž projevuje přetrénování stromů. Ořezávání metodou PEP nám dávalo pro stejně velké množiny použité k bu-

dování a ořezávání přesnější stromy. Velikost stromů ořezaných metodou PEP je však mnohem větší než složitost stromů ořezaných metodou REP. Pokud však srovnáme PEP a REP na stejně velkých množinách určených pro růst, je na tom REP z hlediska složitosti stromu lépe. V případě přesnosti stromů byly výsledky obou metod srovnatelné. Pokud tedy máme k dispozici dostatečně velké množiny vzorů, je lepší metoda REP. Naopak pro malé množiny je výhodnější použít PEP.

V diplomové práci byly také zkoumány stromy SDT. V experimentech jsme ověřovali nepodložená tvrzení autorů. Přišli jsme na to, že chybová funkce nemusí být pro pevné dělení α, označení L_L a L_R unimodální. Tato vlastnost je však nutným předpokladem pro hledání optimální šířky β. Navrhli jsme další dva jednoduché algoritmy (naivní algoritmus a algoritmus první od konce), které však nevedly k výraznému zlepšení přesnosti. SDT stromy se neukázaly být příliš citlivé na volbu β a dávaly dobré výsledky i v případě nepřesné aproximace β. Backfitting využívá k optimalizaci parametrů Levenberg-Marquardtovu nelineární optimalizační techniku (LM). Základní metoda LM předpokládá, že Hessiánská matice nutná k optimalizaci není singulární. Nám se podařilo dokázat, že existuje strom SDT, pro který je Hessiánská matice singulární (dokonce nulová) a díky tomu nelze vždy základní tvar LM použít.

Ořezávání SDT stromu vedlo k drastickému snížení velikosti. Chyba ořezaného stromu však byla o něco horší než chyba vybudovaného stromu. Refitting, podle očekávání, zlepšil přesnost stromu.

Na závěr jednotlivých pokusů jsme provedli srovnání všech testovaných rozhodovacích stromů. Nejhorší složitost měly SDT stromy. Velká složitost SDT stromů byla způsobena přechodovou oblastí, protože v následnících uzlů nedochází k velké redukci počtu vzorů jako v případě stromů CART a C4.5. Pro menší velikosti množin vzorů se objevovaly výkyvy ve výsledcích. Pro větší velikosti množin se výsledky ustálily. Čím byla množina trénovacích vzorů větší, tím se postupně zlepšovala přesnost stromů. Nejlepší přesnost vykazovaly téměř ve všech testech SDT stromy, protože vzory blízko prahové hodnoty dělení α jsou poslány do obou větví a tím nedojde k takovému oddálení vzorů v prostoru tříd. SDT stromy jsou také méně citlivé na mírné odchylky v datech. Citlivost na mírné odchylky v datech byla opět snížena díky přechodové oblasti. SDT stromy se ukázaly jako stabilnější, protože následníci uzlu obsahují větší počet vzorů než stromy CART a C4.5 a nová dělení jsou vytvořena na základě větších množin vzorů. Časové nároky jednotlivých metod jsou nejlepší pro stromy C4.5. U CART stromů je nejkritičtější fáze ořezávání metodou MCCP, která používá cross validaci k odhadování chyb pro posloupnost podstromů. Nejvíce času pro trénování potřebovaly SDT stromy. Konkrétně největší časové nároky měl refitting, který používá k hledání optimálních parametrů v listech metody nejmenších čtverců.

6.2 Další vývoj

Na tuto práci lze navázat a dále ji rozšiřovat. V této kapitole se pokusíme nastínit možné směry dalšího vývoje.

V práci jsme ukázali, že stromy C4.5 jsou efektivní, ale citlivé na malé odchylky dat [28]. Do budoucna by model C4.5 mohl být rozšířen na fuzzy C4.5, jako se o to pokoušeli v [14]. Pomocí teorie fuzzy množin by byla zmírněna chyba, která vznikne při klasifikaci vzorů s hodnotami atributů blízkých dělení uzlu.

U SDT stromů jsme předpokládali, že všechny hodnoty atributů jsou numerické. SDT stromy by mohly zpracovávat také nominální atributy. Diskriminační funkce by mohla mít také tvar sigmoidy, Gaussovský nebo triangulární [28].

Pomocí experimentů jsme zjistili, že samotné ořezávání SDT nezlepší přesnost stromů. Setřídění uzlů do posloupnosti totiž nemusí garantovat správné pořadí ořezávaných uzlů a tím správné podstromy s minimální chybou [28]. Relevantní podstromy je možné vybírat podle jiného kritéria - např. podle úbytku chyby dané vztahem $E_S - E_{S_L} - E_{S_R}$, kde S je fuzzy množina vzorů, které se dostanou do daného uzlu a S_L a S_R jsou fuzzy množiny odpovídající jeho následníkům. Jiným kritériem pro měření by mohla být chyba v celém podstromě $G|_v$ s kořenem v [28]. Dalším zlepšením by mohl být provedení refittingu na každém stromě v posloupnosti během ořezávání a následně vybrat nejlepší z nich.

Fibonacciho prohledávání není možné použít pro nalezení β, ve kterém nabývá chybová funkce minima. Pro optimální β by bylo zajímavé zkusit použít některé ze stochastických metod, např. simulované žíhání [15].

Levenberg-Marquardtovu optimalizaci nelze vždy použít pro fázi backfitting díky singularitě Hessiánské matice. Cílem dalšího bádání by mohlo být navržení jiné metody pro optimalizaci všech parametrů SDT stromu, která by využívala parciálních derivací parametrů stromu.

Literatura

[1] Andriyashin, A.: *Financial Applications of Classification and Regression Trees.* CASE - Center of Applied Statistics and Economics Humboldt University, Berlin, 2005

[2] Breiman, L., Friedman, J. H., Olshen, R. A., Stone, C. J.: *Classification and Regression Trees.* Chapman & Hall, New York, 1984

[3] Boyen, X., Wehenkel, L.: *On the Unfairness of Convex Discriminator Quality Measures for Fuzzy Partitioning in Machine Learning.* Technical Report, University of Liege, 1995

[4] Carter, C., Catlett, J.: *Assessing Credit Card Applications Using Machine Learning.* IEEE Expert Fall, 1987, pp. 71-79

[5] Cestnik, B., Bratko, I.: *On estimating probabilities in tree pruning.* Proc. European Working Session On Learning, (Porto), Y. Kodratoff (ed.), Springer Verlag, March 1991, pp. 138-150

[6] Dietterich, T. G., Kong, E. B.: *Machine Learning Bias, Statistical Bias, and Statistical Variance of Decision Tree Algorithms.* Technical Report, Department of Computer Science, Oregon State University, 1995

[7] Duda, R. O., Hart, P. E., Stork, D. G.: *Pattern Classification.* Wiley, 2003

[8] Eloma, T., Kääriäinen, M.: *An Analysis of Reduced Error Pruning.* Journal of Artificial Intelligence Research 15, AI Access Foundation and Morgan Kaufmann, 2001, pp. 163-187

[9] Esposito, F., Malerba, D., Semeraro, G.: *Decision tree pruning as a search in the state space.* In Brazdil, P. B. (Ed.), Machine Learning: ECML-93, Proceedings of the Sixth European Conference, Vol. 667 of LNAI, Berlin, Heidelberg, New York. Springer-Verlag, 1993, pp. 165-184

[10] Esposito, F., Malerba, D., Semeraro, G.: *A comparative analysis of methods for pruning decision trees.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(5), pp. 476-491

[11] Fayad, U. M., Irani, K. B.: *On the handling of continuous-valued attributes in decision tree generation.* Springer Netherlands, Vol. 8, No. 1, 1992

[12] Friedman, J. H.: *Local Learning Based On Recursive Covering.* Technical Report, Dept. of Statistics, Standford University, August 1996

[13] Holte, R. C., Acker, L., Porter, B.: *Concept learning and the problem of small disjuncts.* In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, San Mateo, CA. Morgan Kaufmann, 1989, pp. 813-818

[14] Kazunori, H., Motohide, U., Hiroshi, S., Yuushi, U.: *Fuzzy C4.5 for Generating Fuzzy Decision Trees and Its Improvement.* Faji Shisutemu Shinpojiumu Koen Ronbunshu, Vol. 15, 1999, pp. 515–518

[15] Kirkpatrick, S., Gelatt C. D., Vecchi M. P.: *Optimization by Simulated Annealing.* Science, New Series, Vol. 220, No. 4598., May 13, 1983, pp. 671–680

[16] Kononenko, I.: *The Minimum Description Length Based Decision Tree Pruning.* The Pacific Rim International Conferences on Artificial Intelligence, 1998

[17] Lawson, C. L., Hanson, R. J.: *Solving Least Squares Problems.* Prentice-Hall, 1974

[18] LIS - Rudjer Boskovic Institute: *DMS Tutorial - Decision trees.* `http://dms.irb.hr/tutorial/tut_dtrees.php`, 2001

[19] Likeš, J., Laga, J.: *Základní statistické tabulky.* SNTL, Praha, 1978

[20] Marsala, C., Bouchon-Meunier, B.: *Forests of fuzzy decision trees.* Proceedings of the International Fuzzy Systems Association World Congress, Vol. 2, Prague, 1997, pp. 369-374

[21] Marsala, C.: *Apprentissage inductif en présence de données imprécises: construction et utilisation d'arbres de décision flous.* Thèse de doctorat, Universitée Paris 6, 1998

[22] Matoušek, J., Nešetřil, J.: *Kapitoly z diskrétní matematiky.* Karolinum, 2000

[23] Mingers, J.: *An empirical comparision of pruning methods for decision tree induction.* Machine Learning, 4(2), 1989, pp. 227-243

[24] Mitchell, T. M.: *Machine Learning.* McGraw-Hill, New York, 1997

[25] Niblett, T., Bratko, I.: *Learning decision rules in noisy domains.* Proc. Expert Systems 86, Brighton, UK, December 1986

[26] Oates, T., Jensen, D.: *Toward a theoretical understanding of why and when decision tree pruning algorithms fail.* In Proceedings of the Sixteenth National Conference on Artificial Intelligence, Menlo Park, CA/Cambridge, MA. AAAI Press/MIT Press, 1999, pp. 372-378

[27] Olaru, C.: *Fuzzy Decision Tree Induction using Square Error Type of Criterion.* Internal Report, University of Liege, Department of Electrical and Computer Engineering, Belgium, October 1998

[28] Olaru, C., Wehenkel, L.: *A Complete Fuzzy Decision Tree Technique.* Fuzzy Sets and Systems, 138, 2003, pp. 221-254

[29] Press, W. H., Teukolsky, S. A., Vetterling W. T., Flannery B. P.: *Numerical Recipes in C.* The Art of Scientific Computing, 2nd Edition, Cambridge University Press, Cambridge, 1994

[30] Quinlan, J. R.: *Simplifying decision trees.* International Journal of Manmachine Studies, 27(3), 1987, pp. 221-248

[31] Quinlan, J. R.: *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, 1993

[32] Ramdani M.: *Système d'induction formelle à base de connaissances imprécises.* Thèse de doctorat, Universitée Paris VI, Paris, France, February 1994

[33] Suarez, A., Lutsko, F.: *Globally optimal fuzzy decision trees for classification and regression.* IEEE Trans. Pattern Anal. Machine Intelligence 21 (12),1999, pp. 1297-1311

[34] Thieme, L. S.: *Entscheidungsbaumverfahren.* `http://marketing.wiwi.uni-karlsruhe.de/institut/viror/kaiman/kaiman/eb/index.xml.html`

[35] Wehenkel, L.: *Discretization of continuous attributes for supervised learning. Variance evaluation and variance reduction.* Proceedings of the Seventh IFSA World Congress (invited paper), Vol. 2, Prague, June 25-29, 1997, pp. 381-388

[36] Winston, P. H.: *Artificial Intelligence, Third Edition.* Adison-Wesley, 1992

[37] Witten, I. H., Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition.* Morgan Kaufmann (Elsevier Inc.), 2005

[38] Zadeh, L. A.: *Fuzzy Sets.* Department of Eletrical Engineering and Electronics Research Laboratory, University California, Berkley, California, Information and Control 8, 1965, pp. 338-353

A Přílohy

A.1 Obsah CD

Na přiloženém CD je následující struktura adresářů:

- adresář `bin`: Ukázkový program pro budování SDT stromů a příklady jejich spuštění (viz dále).
- adresář `data`: Databáze testovaných množin vzorů.
- adresář `src`: Zdrojový kód ukázkových programů.
- soubor `dt.pdf`: Tento text diplomové práce v elektronické podobě.

A.2 Testovací data

Adresář `\data` obsahuje dvě sady množin vzorů ve formátu arff pro data miningovou knihovnu Weka (viz kapitola A.4). Soubory s množinami vzorů OMIB jsou umístěny v adresáři `\data\omib`. Soubory s množinami vzorů znaků anglické abecedy jsou umístěny v adresáři `\data\letters`. Pro názvy jednotlivých souborů jsme zvolili následující konvence pojmenování:

<*název množiny vzorů*>–<*transformace*><*počet vzorů*>.arff

Příklad:

`letters-bn100.arff`

Soubor obsahuje 100 vzorů znaků anglické abecedy. Vzory byly normalizovány a třídový atribut byl binarizován.

A.3 Ukázkový program

Program pro budování SDT stromů `sdt.jar` je možné spustit pouze na normalizovaných vzorech s binární třídou. Program je umístěn v adresáři `\bin`. Program jsme testovali pod operačním systémem Windows XP Professional v prostředí Java(TM) SE Runtime Environment (sestavení 1.6.0_05-b13). Předpokládejme, že máme v systémové proměnné `PATH` správně nastavenu cestu k spustitelným souborům Javy a nacházíme se v adresáři `\bin`. Budování stromů SDT je možné spustit příkazem:

```
java -jar sdt.jar
```

Pokud vše proběhne správně, na standardní výstup bude vypsána nápověda programu SDT. Nápověda programu je rozdělena do dvou částí. V první části nápovědy jsou popsána obecná nastavení data miningové knihovny Weka (viz kapitola A.4), kterými se nebudeme zabývat. V druhé části nápovědy jsou popsána nastavení týkající se pouze SDT stromů:

-C *<část růstové množiny>*
: Omezení minimálního počtu trénovacích vzorů v uzlu stromu SDT. Číslo *<část růstové množiny>* určuje část počtu všech vzorů pro růst. Pokud je počet vzorů v uzlu menší než daný počet vzorů, je další dělení uzlu ukončeno (Implicitní hodnota je 0.01).

-E *<chyba>*
: Práh pro chybu E_S v uzlu SDT stromu. Pokud je chyba v uzlu menší než *<chyba>*, je dělení uzlu ukončeno (Implicitní hodnota je 1E-4).

-D *<redukce chyby>*
: Práh pro minimální redukci chyby v uzlu SDT stromu. Pokud je redukce chyby menší než *<redukce chyby>*, dělení uzlu je ukončeno (Implicitní hodnota je 1E-2).

-W *<prohledávací algoritmus><parametr>*
: Volba algoritmu pro hledání šířky β ve fázi růstu SDT stromu. Za parametr *<prohledávací algoritmus>* je možné dosadit jeden z těchto algoritmů:

  f – Fibonacciho prohledávání. Nastavení *<parametr>* označuje počet iterací Fibonacciho algoritmu (Implicitní hodnota je 5).

  n – Naivní algoritmus. Nastavení *<parametr>* označuje počet dělení prohledávaného intervalu (Implicitní hodnota je 20).

  e – Algoritmus „První od konce“ (POK). Nastavení *<parametr>* označuje počet dělení prohledávaného intervalu (Implicitní hodnota je 20).

Příklad:

`-W n15`

Znamená, že pro hledání optimální šířky β je použit naivní algoritmus s 15 děleními.

-P *<počet dílů>*
Budou provedeny ořezávací operace. Argument *<počet dílů>* označuje na kolik dílů bude rozdělena trénovací množina vzorů. Jeden díl vzorů bude použit k ořezávání a zbytek k vytvoření stromu.

-R
Bude proveden refitting.

Následující příklad demonstruje použití programu `sdt.jar`:

```
java -jar sdt.jar -C 0.0001 -W f30 -P 2 -R
    -t ..\data\letters\letters-bn100.arff
```

Testování SDT stromu bude spuštěno na množině vzorů:

```
..\data\letters\letters-bn100.arff
```

Bude použita desetidílná cross validace. Pro budování stromu bude tedy použito 90 vzorů a pro testování přesnosti stromu 10 vzorů. Práh pro minimální počet vzorů (parametr -C 0.0001) v uzlu je nastaven na hodnotu 2, protože 1/10 000 z 90 vzorů je 0,009 ($0{,}0001 \cdot 90 = 0{,}009$). To znamená, že 0,009 vzorů bude práh pro minimální počet vzorů v uzlu. Ovšem minimální možný práh je 2, a proto bude prahová hodnota nastavena na 2. Pro hledání šířky β bude použito Fibonacciho prohledávání s 30 iteracemi. Trénovací množina vzorů bude stratifikovaně rozdělena na 2 díly. Jeden díl (45 vzorů) bude použit pro růst a druhý (45 vzorů) pro ořezávání SDT stromu. Po ořezání bude spuštěn refitting. Výstupem bude shrnutí celého experimentu:

```
Options: -C 0.0001 -W f30 -P 2 -R

SDT Tree - nodes split parameters:

L0.42

Number of Leaves: 1
Size of tree: 1
```

```
Time taken to build model: 0.11 seconds
Time taken to test model on training data: 0 seconds

=== Error on training data ===

Correctly Classified Instances          58               58  %
Incorrectly Classified Instances        42               42  %
Kappa statistic                          0
Mean absolute error                      0.42
Root mean squared error                  0.6481
Relative absolute error                 86.1625 %
Root relative squared error            131.3058 %
Total Number of Instances              100


=== Confusion Matrix ===

  a  b   <-- classified as
 58  0 |  a = 0
 42  0 |  b = 1



=== Stratified cross-validation ===

Correctly Classified Instances          66               66  %
Incorrectly Classified Instances        34               34  %
Kappa statistic                          0.2833
Mean absolute error                      0.34
Root mean squared error                  0.5831
Relative absolute error                 69.697  %
Root relative squared error            118.0515 %
Total Number of Instances              100


=== Confusion Matrix ===

  a  b   <-- classified as
 45 13 |  a = 0
 21 21 |  b = 1
```

Popis a význam výstupu je možné nalézt v dokumentaci data miningové knihovny Weka (viz kapitola A.4).

A.4 Použité nástroje

K vytvoření diplomové práce byly použity tyto nástroje a knihovny:

- Eclipse Version: 3.2.2. – vývojové prostředí použité pro implementaci SDT stromů.
- JAMA: A Java Matrix Package – maticové operace ladící fáze refitting SDT stromů.
  `http://math.nist.gov/javanumerics/jama/`
- Java 2 Platform Standard Edition 5.0 Development Kit (JDK 5.0)
- MATLAB 7.0 (R14) - Statistics Toolbox – experimenty s CART stromy, generování obrázků grafů v této práci.
- Orange – data miningová knihovna autorů ořezávacího algoritmu MEP. Použita pro všechny testy ořezávacího algoritmu MEP.
  `http://www.ailab.si/orange/`
- Weka 3: Data Mining Software in Java – data miningová knihovna použitá pro implementaci SDT stromů. Součástí knihovny je také implementace stromů C4.5. Knihovna disponuje mnoha nástroji pro testování klasifikátorů a také nástroji pro předzpracování množin vzorů.
  `http://www.cs.waikato.ac.nz/ml/weka/`

Celá práce byla kompletně vysázena v systému LaTeX s českou lokalizací.