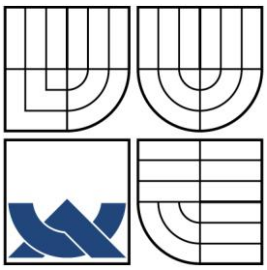
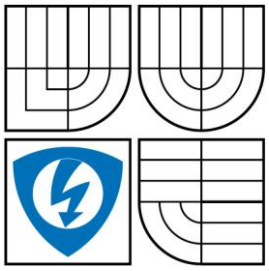


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A
KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘÍCÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

META-LEARNING

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE — Bc. MARTIN HOVORKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR HONZÍK , PH.D.
SUPERVISOR

BRNO 2008

P r o h l á š e n í

„Prohlašuji, že svou diplomovou práci na téma Meta-Learning jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne : Podpis:

P o d ě k o v á n í

Děkuji tímto vedoucímu mé diplomové práce Ing. Petru Honzíkovi, Ph.D. za cenné připomínky a rady při vypracování této práce. V neposlední řadě bych rád poděkoval rodině, přátelům a dalším, kteří mě po dobu studia podporovali

V Brně dne : Podpis:

Abstrakt:

Účelem této práce je rešerše metod Meta-Learningu, jejich naprogramování a porovnání s metodami strojového učení, které Meta-Learning nevyužívají.

První část práce se zabývá především strojovým učením a jeho základními metodami, jako Rozhodovací stromy, Naivní Bayesův klasifikátor a další, které právě Meta-Learning často využívá. Dále se samozřejmě věnuje základním a nejpoužívanějším metodám Meta-Learningu jako Bagging, Bumping, Boosting a Stacking.

Ve druhé části se práce zabývá právě algoritmem pro metodu adaBoost a dalšími metodami vytvořenými pomocí programů Matlab a RapidMiner. Výsledky jsou vzájemně porovnány. Přílohy obsahují algoritmus adaBoost a Cross Validation pro program Matlab a dále metody vytvořené pomocí programu RapidMiner.

Abstract:

The thesis goal is to background research of Meta-Learning methods, to create alorithm and compare them with machine-learning methods that the Meta-Learning don´t use.

First part deals with machine-learning basic methods like decision trees, Naive Bayes etc. which are used in Meta-Learning. Of course next are basic and most used Meta-Learning methods like Bumping, Bagging, Boosting and Stacking.

In following part work deals with adaBoost algorithm and other methods created with use of Matlab and RapidMiner programs. Results are compared and placed in table. Apendixes contains whole adaBoost and Cross Validation algorithm made in Matlab and other methods created in RapidMiner.

Klíčová slova:

Strojové učení, meta-learning, bagging, boosting, adaboost, adaboostM1, stacking, ID3, C4.5, kNN, IBL,CART, Decision Stump, Cross Validation

Key words:

Machine learning, meta-learning, bagging, boosting, adaboost, adaboostM1, stacking, ID3, C4.5, kNN, IBL,CART, Decision Stump, Cross Validation

Bibliografická citace:

HOVORKA, Martin. *Meta-Learning*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. Stran:53, příloh: 4. Ing.Pert Honzík, Ph.D.

1. Obsah:

2. Seznam obrázků a tabulek

Obrázky:

Tabulky:

3. Úvod:

Obsahem této práce by mělo být bližší seznámení a naprogramování některých metod Meta-Learningu. Dále zde budou uvedeny základy strojového učení. Pro celkovou orientaci budou ze strojového učení probrány základní pojmy, klasifikátory jako Lineární, Bayerův nebo Rozhodovací stromy. Dále a to je prvním těžištěm práce bude provedena rešerše metod Meta-Learningu, především nejpoužívanějších metod jako Bagging, Stacking a především Boosting. V další části bude předveden algoritmus pro nejpoužívanější metodu Meta-Learningu adaBoost vytvořený v programu Matlab, pro ověření funkce algoritmu bude uveden algoritmus metody Cross Validation, která určí chybu a přesnost modelu. Pro porovnání budou uvedeny další metody Meta-Learningu vytvořené v programu RapidMiner. Jedná se o metody Bagging, Stacking a pro zajímavost opět adaBoost. Pro úplnost budeou dále předvedeny výsledky základních metod strojového učení, které byli použity při tvorbě Meta-Learning modelů.

4. Umělá inteligence:

Marvin Lee Minsky (nar. 9. srpna 1927), spoluzakladatel laboratoře umělé inteligence na MIT (Massachusetts Institute of Technology) definoval umělou inteligenci jako vědu, jejímž ukolem je naučit stroje dělat věci, které vyžadují inteligenci, jsou-li prováděny člověkem

Umělá inteligence se dělí na inteligenci tzv.slabou a silnou. Slabá umělá inteligence pouze vykazuje inteligentní chování, aniž by ve skutečnosti došlo k pochopení podstaty a vyřešení problému. Jako příklad můžeme uvést Turingův test, kdy i slabá UI dokáže tento test splnit. Silná umělá inteligence disponuje chápáním jakým je obdařena lidská mysl

Podoblasti UI:

1. Neuronové sítě
2. Expertní systémy
3. Rozpoznávání řeči
4. Genetické programování
5. Strojové učení

A další

Oblastmi jako Neuronové sítě, expertní systémy, rozpoznávání řeči apod. se tu nebudeme zabývat. Pro nás je důležitá pouze část nazvaná strojové učení a především její část zabývající se tvz. Meta-Learningem *viz skripta [11]*

5. Strojové učení:

Strojové učení je podoblastí umělé inteligence, zabývající se algoritmy a techniky, které umožňují počítačovému systému 'učit se'. Učením v daném kontextu rozumíme takovou změnu vnitřního stavu systému, která zefektivní schopnost přizpůsobení se změnám okolního prostředí.

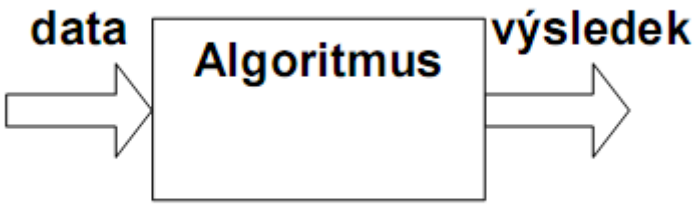
Strojové učení se značně prolíná s oblastmi statistiky a dobývaní znalostí a má široké uplatnění. Jeho techniky se využívají např. v biomedicínské informatice (tzv. systémy pro podporu rozhodování), rozlišení nelegálního užití kreditních karet, rozpoznávání řeči a psaného textu, či mnohé další. *[1]*

Úkoly strojového učení je zvolit:

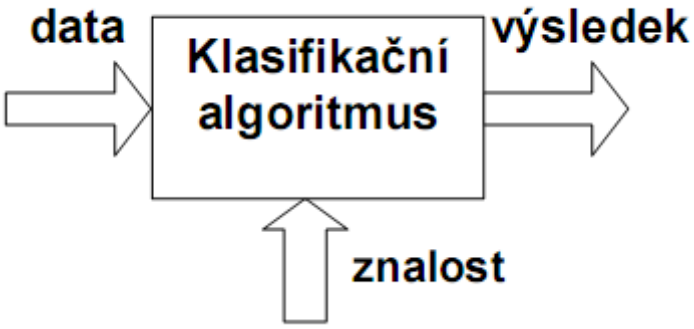
- Vhodné příznaky
- Klasifikační metodu
- Metodu učení, vyhodnocení chyb klasifikace
- Interpretace výsledku učení
- Implementace klasifikátoru do cílové aplikace

Základní pojmy:

Strojová klasifikace „pattern recognition“, nauka o získávání a zpracování znalostí



Obr 1. Klasické zpracování dat



Obr 2. Zpracování dat při pattern recognition

Znalost: Informace o dané problematice

- Mělká znalost – vychází z porovnávání skutečnosti, popis jevů je „povrchní“
- Hloubková znalost – vyjadřuje vnitřní zákonitosti jevů
- Deklarativní znalost – pravidla
- Procedurální znalost – znalost získaná opakovaným prováděním cvičení. Pro strojové učení obtížně využitelné

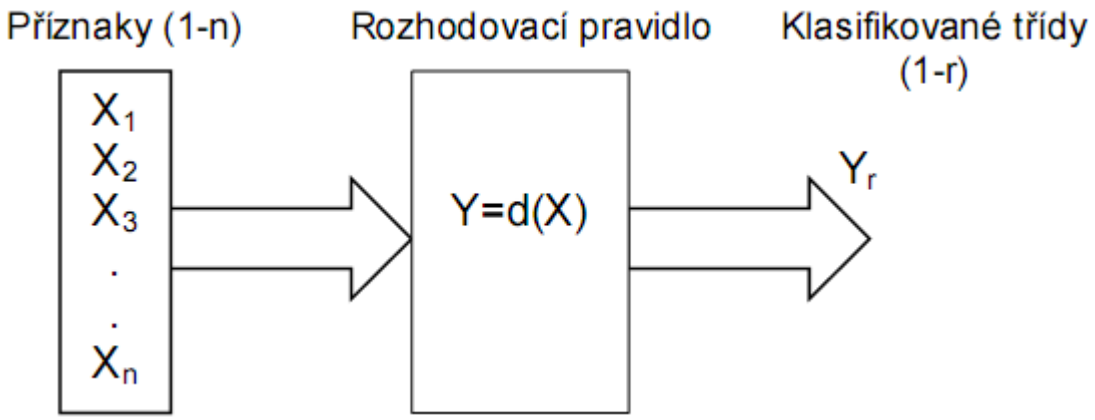
Inference: Postup k dosažení výsledku

$$F \xrightarrow{T} C$$

- Dedukce - pokud známe F , T a určujeme C. Vždy jistý správný výsledek
- Abdukce - pokud známe C , T a určujeme F Není zaručen správný výsledek
- Indekce - pokud známe F , C a hledáme T Není zaručen správný výsledek

Strojové učení tedy představuje proces hledání znalosti (inference) pomocí indukce

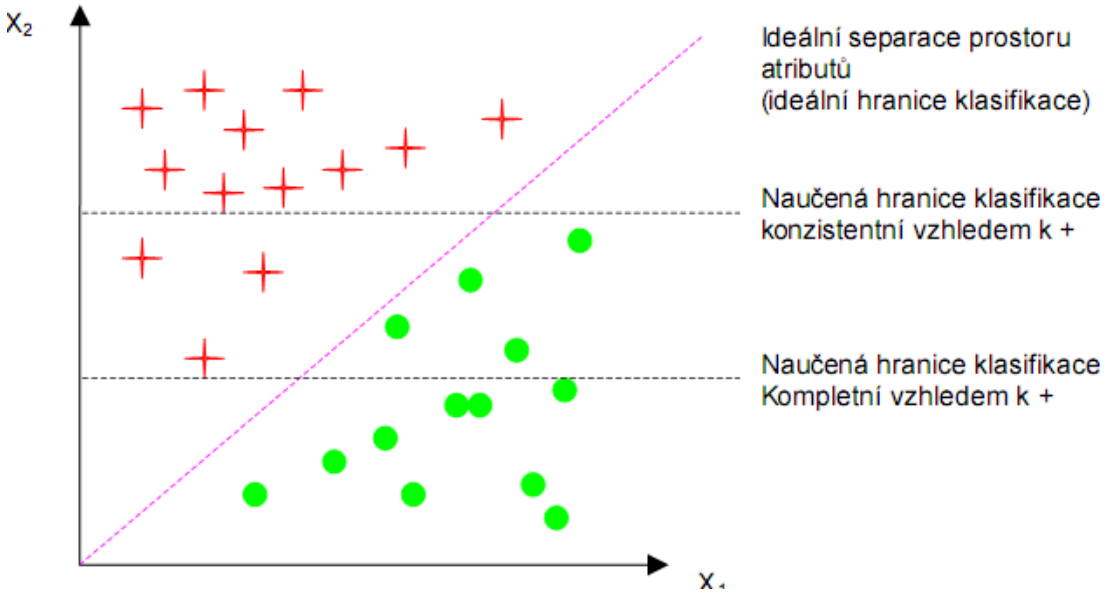
Klasifikátor: Algoritmus, schopný při vhodné množině znalostí úspěšně rozdělovat vstupní data s hodnotami atributů (příznaků), do výstupních předem zvolených skupin (tříd). Vhodná volba klasifikačního algoritmu představuje nutnou podmínku k úspěšné klasifikaci. Modely klasifikátoru mohou být kompletní nebo konzistentní. Kompletní model pokrývá všechny pozitivní případy, ale pravděpodobně pokrývá i některé negativní případy. Konzistentní model nepokrývá žádný negativní případ, ale pravděpodobně nepokrývá i některý pozitivní případ. *[12]*



Obr 3. Vstupy a výstupy klasifikátoru

Postup použití klasifikační metody:

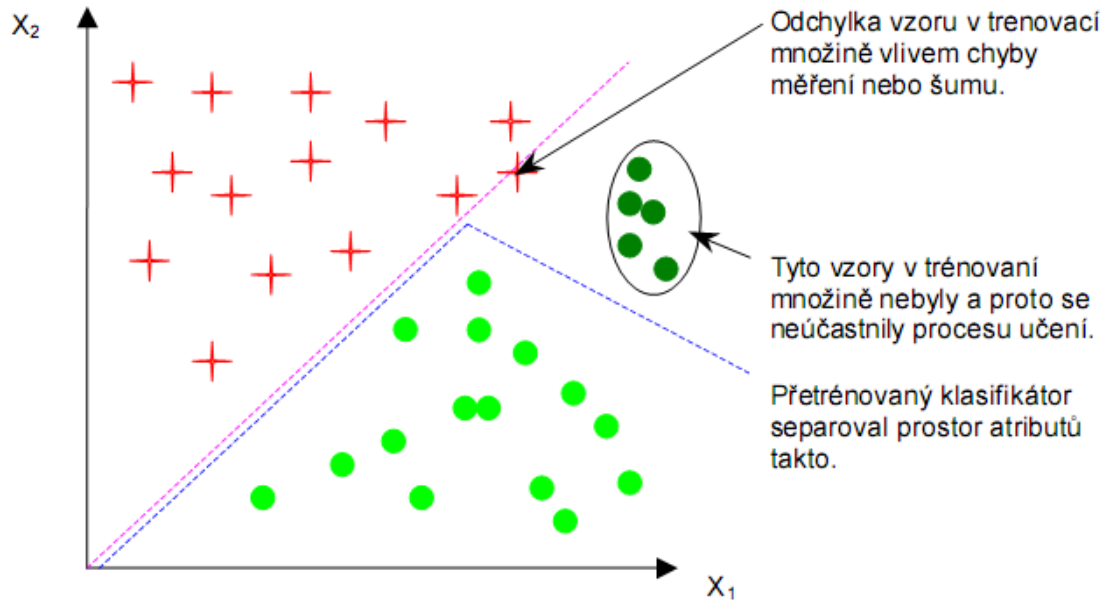
1. Učení - generování znalosti (modelu) s ohledem na typ klasifikátoru
2. Ověřování - verifikace znalostí na jiných datech, než byli použity při učení a výpočet přesnosti klasifikace
3. Klasifikace - běžný provoz naučeného klasifikátoru



Obr 4. Kompletní a konzistentní model klasifikátoru

Přetrénování klasifikátoru:

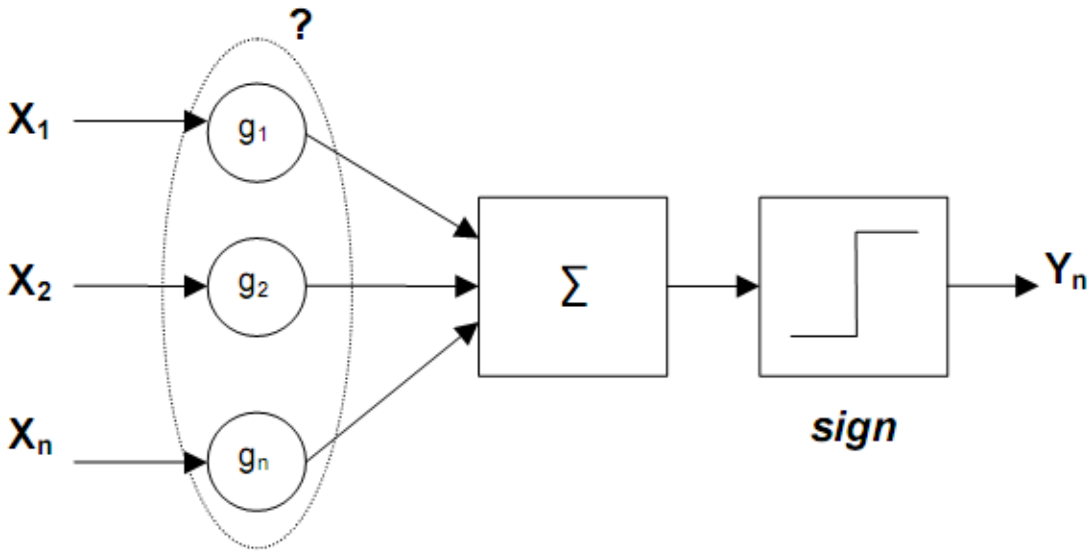
Na obrázku je patrný problém přetrénování klasifikátoru. Proto je lépe netrvat na konzistenci a úplnosti popisu. Příliš přesný popis vzhledem k trénovací množině má velmi často nižší přesnost vzhledem ke skutečným datům.



Obr 5. Přetrénování klasifikátoru

5.1 Lineární klasifikátor:

Představuje jednoduchou klasifikační metoda založenou na rozdělení prostorů příznaku pomocí částech lineárními úseky. Prostor příznaků je obecně prostor s mnoha dimenzemi proto hovoříme o separaci prostoru příznaků nadrovinami (nadplochami)



Obr 6. Schéma lineárního klasifikátoru

$$s_r = g_1 \cdot X_1 + g_2 \cdot X_2 + \ldots g_n \cdot X_n$$

Úkolem učení klasifikátoru je určit vhodné konstanty g (případně parametry funkce *sign*). Pokud lze tuto podmínku separace splnit při chybě klasifikace 0% hovoříme o lineární separabilitě prostoru příznaků.

Pro dichotomické úlohy klasifikace není nutné vyhodnocovat hodnoty diskriminační funkce vždy pro oba etalony, stačí vyhodnotit rozdíl obou diskriminačních funkcí a klasifikovat vzor na základě znaménkového rozdílu. *[12]*

5.2 Bayesův klasifikátor:

Patří do skupiny statických příznaků, umožňuje inkrementální i dávkové učení. Naučená znalost (model) je reprezentován pravděpodobnostním rozložením tříd. Při klasifikaci je zvolena třída s největší pravděpodobností.

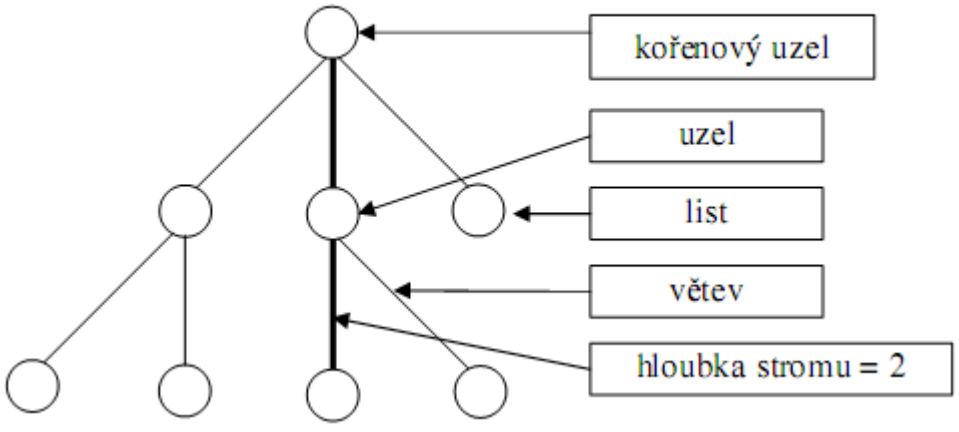
Podmíněná pravděpodobnost závislá na konjunkci jevů se nahradí funkcí podmíněných pravděpodobností jednoduchých jevů. Proto metoda požaduje úplnou vzájemnou statickou nezávislost atributů. *viz[1]*

5.3 Rozhodovací stromy:

V současnosti asi nejpoužívanější metoda pro analýzu dat a jejich hierarchickou reprezentaci. Rozhodovací stromy jsou primárně určeny pro klasifikaci kvalitativních závislých proměnných na základě vstupních proměných.Tato metoda se využívá zejména při „Data Mining“. *Viz[1]*

Druhy rozhodovacích stromů:

- CART – využití především ve statistice, kvantitativní a kvalitativní vstupy, výstupy kvantitativní nebo nominální, binární topologie, algoritmus CART
- CLS – využití při učení konceptu, , nominální a ordinální vstupy, výstupy nominální, topologie dle atributů, algoritmus CLS, ID3, C4.5 , C5
- AID – využití pro komplexní a statistické vztahy, vstupy i výstupy mohou být libovolné typy dat, topologie-všechny typy, algoritmus AID,THAID



Obr 7. Rozhodovací strom

5.4 Metoda nejbližšího souseda:

Metoda k-nejbližšího souseda neboli k Nearest Neighbor (k-NN) je používána v případech, kdy se není možné vytvořit dobrý model úkolu. Potřebujeme-li zjistit vlastnost nějakého objektu a máme k dispozici jen soubor zaznamenaných referenčních případů, postupujeme tak, že najdeme nejpodobnější případ, jehož vlastnosti jsou známy. O neznámé hodnotě atributu se pak předpokládá, že je stejná jako u podobného zaznamenaného případu

Podmínkou k-NN je, že všechny příklady (instance) odpovídají bodům v n-rozměrném prostoru. Nejprve definujeme vzdálenostní metriku d, která slouží ke kvantitativnímu ohodnocení podobnosti dvou příkladů. Na základě této metriky vyhledáme pro právě předpovídanou instanci k nejbližších sousedů v množině trénovacích příkladů. Hodnota závislé veličiny je poté stanovena na základě jádrové funkce, která vhodně kombinuje hodnoty závislých veličin všech nalezených sousedů. Nejjednodušší jádrovou funkcí je aritmetický průměr:

$$y(i) = \frac{1}{k}\sum_{j=1}^{k} y(j)$$

Předpověď pro i-tý objekt je pak průměrnou hodnotou závislé veličiny zjištěnou u k nejbližších sousedů nalezených v trénovacích datech(trénovací příklady jsou indexovány dle vzdálenosti d(x(i), x(j)) tak, že y(1) je hodnota cílové funkce nejbližšího souseda, y(2) druhého nejbližšího souseda. Složitější jádrová funkce používá funkci vzdálenostní, která určuje váhu s jakou se jednotlivé hodnoty y(j) aplikují.

Kvalita k-NN je závislá na volbě parametrů, jimž jsou počet sousedů k, vzdálenostní metrika d a jádrová funkce. K optimalizaci parametrů dochází ve fázi učení.

Zlepšení k-NN je možno pomocí váhování přínosů jednotlivých sousedů pomocí vzdálenosti. Bližší soused přispívá k výsledku více než soused vzdálenější.Tato vylepšená metoda je odolná k výsledku šumu v trénovacích datech a velmi efektivní, pokud je k dispozici dostatek trénovacích dat. Vzdálenost mezi

instancemi se počítá pomocí všech atributů (všech os euklidovského prostoru). Tímto se liší od metod založených na rozhodovacích stromech a pravidlech, kde se pro vytváření hypotézy používá jen podmnožina.V případě nejbližšího souseda jsou koncepty (třídy) reprezentovány svými typickými představiteli. V procesu klasifikace se pak nový příklad zařadí do třídy na základě podobnosti Jde tedy o metodu která vychází ze shlukové analýzy. Klíčovým pojmem je koncept podobnosti, resp. vzdálenosti dvou příkladů. *[1]*

6. Meta-learning:

Metoda strojového učení docházející k predikci kombinací více modelů, jedná se o tzv. metaznalost, to je znalost o znalosti.

6.1 Přehled metod meta-learningu:

Existuje mnoho metod meta-learningu, zde jsou uvedeny pouze ty nejzákladnější a nejdůležitější.

1. Bumping
2. Bagging
3. Stacking
4. Boosting (adaBoosting)
5. Radom forest

6.1.1 Bumping

Tato metoda využívá na rozdíl od ostatních pouze jeden model , který je ale vybraný z několika modelů naučených na stejných trénovacích datech. Metodou bootsrap (popsáno u metody Bagging) je z trénovacích dat o N prvcích vytvořeno M trénovacích množin o N prvcích. Poté je spočtena chyba modelu ze všech trénovacích dat a podle tohoto jsou vybrány nejlepší parametry modelu b z M vytvořených modelů.

$$\hat{b} = \arg\min_{b_j = b_1 \ldots b_M} \sum_{i=1}^{N} LF(y_i, f(\hat{b_j}, x_i))$$

Výsledky na nových testovacích datech dokládají, že tento způsob vede k výběru lepšího modelu než při výpočtu ze všech trénovacích dat. Tato metoda je velice robustní proti uvíznutí v lokálních extrémech.

6.1.2.Bagging

Bagging neboli Bootstrap Aggregation je metoda vytvořená Loe Breimanem z Univerzity v Berkeley v roce 1996. Jedná se o jednu z jednodušších metod meta-learningu, která je velmi stabilní a snižuje riziko přeučení. Často se používá při nedostatku dat a potřebě zvýšení stability a přesnosti. Metoda je založena vyloženě na experimentech a není tedy zaručeno zlepšení pro libovolný příklad.

Princip metody:

Při této metodě je využito více trénovacích množin, tedy z N trénovacích dat je vytvořeno M množin o velikosti N. Náhodně je s M opakováním vybráno N prvků z trénovacích dat, na tuto množinu je naučen model.(tzv. Bootstrap). Všechny z M modelů jsou obvykle stejného typu, nejčastěji rozhodovací stromy nebo neuronové sítě (pro lineární a robustní IBL modely je tato metoda zpravidla neúčinná vzhledem

k její stabilitě) Poté když je všech M modelů vytvořeno, dochází k predikci. Neznámá hodnoto je klasifikována každým modelem zvlášť a je určena její třída. Následně se určí pro kterou třídu byla neznáma nejčastěji predikována (nebo průměrná hodnota v případě regrese) a ta je výstupem

Klasifikátor: $\hat{G}(x) = \arg\max_{i=1...C} \sum_{j=1}^{M} \hat{f}_j(x)$;kde C je množství tříd

Regresní : $\hat{f}(x) = \frac{1}{M} \sum_{j=1}^{M} \hat{f}_j(x)$

Vliv opakování na kvalitu predikce:

Na množství opakování (množství modelů) ve skutečnosti mnoho nezáleží, Leo Breiman při ověřování této skutečnosti provedl pokus při kterém data předikoval při 10, 25, 50 a 100 opakování. Jeho výsledky byli:

Počet opakování	**Chyba v určení třídy [%] (Missclassification rate)**
10	21.8
25	19.5
50	19.4
100	19.4

Z tohoto je zřejmé že konkrétně v tomto případě bylo 10 opakování dostatečné množství. Breiman osobně se domnívá že pro číselnou hodnotu výstupu dostačuje menší množství opakování a naopak více opakování je potřeba pro rostoucí množství tříd.*[2],[6]*

6.1.3.Stacking:

Princip:

V tomto případě se jedná o metodu kombinující více modelů různých typů, které jsou rozděleny do dvou úrovní, úrovně 0 a úrovně 1. K učení modelů dochází postupně přičemž se nejprve naučí modely úrovně 0. Ty mohou být libovolného typu a poté modely úrovně 1 tzv. metalerner. Tyto modely mají jednoduchou strukturu, jako například rozhodovací stromy a podobně. Každý model úrovně 0 má určenu kromě predikce také svoji váhu, tedy míru důvěryhodnosti své predikce.

Učení:

Data jsou rozdělena na trénovací a testovací. Na trénovací data jsou naučeny modely úrovně 0. Jejich výstupy budou vstupy do úrovně 1. Jako trénovací data pro metamodel, tj. model úrovně 1 jsou však použita odložená testovací data. Ta jsou prohnána modely úrovně 0, vzniknou tak trénovací data pro model úrovně 1. Po jeho nastavení jsou použita pro přenastavení všech modelů úrovně 0 všechna trénovací data, avšak již bez zásahu do metamodelu úrovně 1 (zlepšení celkové přesnosti)
[9]

6.1.4.Boosting:

Boosting je nejpoužívanější metoda meta-learningu Základní myšlenkou této metody je posílit (Boost) tzv. slabé učící algoritmy na učící algoritmy silné tím, že se použije více modelů stejného typu se slabým učícím algoritmem. Slabý učící algoritmus může být jen o něco přesnější než náhodné hádání, zpravidla je úspěšnost klasifikovaných prvků jen o něco vyšší než 50%. Na počátku je každému trénovacímu záznamu přidělena váha $w_i(x) = 1$. Po vytvoření prvního modelu jsou trénovací prvky modelem klasifikovány, přičemž váha úspěšně klasifikovaných je oslabena (při vytváření následujícího modelu se budou méně významně podílet na jeho nastavení), naopak váha neúspěšně klasifikovaných prvku je posílena

Existuje mnoho variant této metody:

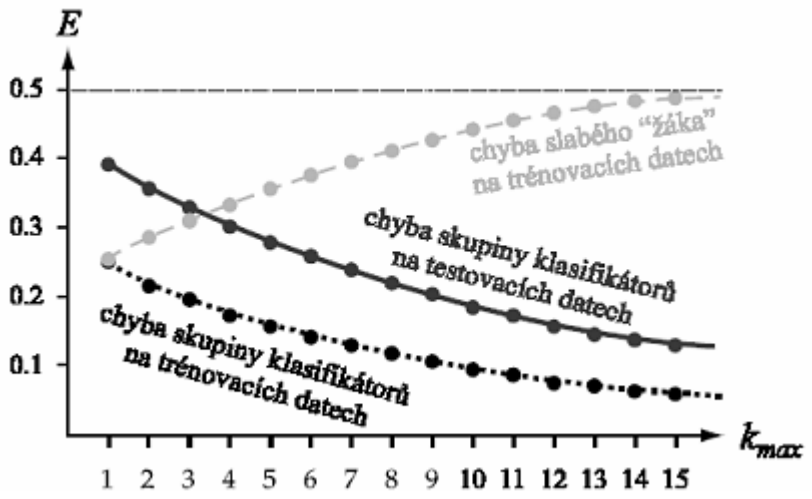
- Boosting bez váhy
- Boosting s váhou
- adaBoosting (řešení problému 2 tříd)
- adaBoosting.M1, adaBoosting.M2 (řešení problému vice tříd)
- adaBoostingR (regrese)

Boosting bez váhy:

Při realizaci nejsou brány v potaz jednotlivé váhy nýbrž cílený výběr trénovacích dat podle úspěšnosti klasifikace předešlých modelů. Na první model je použito omezené množství dat, například třetina nebo čtvrtina bez opakování. Neexistuje přesně daný způsob pro výběr dat, který by byl vždy správný. Z nich je model nastaven s přesností vyšší než 50%. Další model je již nastaven z dat, která první model klasifikuje s úspěšností lepší než 50%. Takto se pokračuje do vyčerpání trénovací množiny. Klasifikace nakonec probíha jako hlasování jednotlivých modelů.

Boosting s váhami:

Tato metoda Boostingu ji pracuje s váhami jednotlivých záznamů. Modely jsou tentokrát trénovány ze všech trénovacích dat a pro chybu klasifikace u každého prvku se bere v potaz jeho váha. Tento postup se specializuje na problémové prvky a vede ke zvětšování celkové chyby modelu. Každému vytvořenému modelu je přidělena tzv. váha modelu, která ovlivňuje jeho vliv na konečnou klasifikaci. S rostoucím počtem modelů se jejich dílčí chyba zvětšuje ale celková chyba na trénovacích a testovacích datech se zmenšuje



Obr 8. Závislost chyb na počtu modelů při adaBoost

adaBoost:

adaBoost je algoritmus pro tvorbu silných klasifikátorů jako lineární kombinaci ze jednoduchých slabých klasifikátorů $h_t(x)$

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

Tato metoda zmenšuje trénovací chybu exponenciálně v závislosti na rostoucím počtu klasifikátorů v hlasující skupině. Pokud každý klasifikátor v hlasující skupině dává výsledky lepší než náhodné, váhové rozhodnutí celé skupiny zaručuje pokles chyby na trénovacích datech. Existuje však nebezpečí přeučení algoritmu, zvláště u složitých dílčích modelů.

Princip adaBoostu:

Uvažujme trénovací sadu s N prvky. Při každém opakování vzorku $t = 0,1,\ldots.T$, prvek i má váhu w_i^t, kde $w_i^1 = \frac{1}{N}$ pro všechny i

pro každý vzorek se opakuje:

- vytvořeni klasifikátoru g^t z trénovací instancí s využitím váhy w_i^t
- četnost chyb u g^t na trénovacích datech (ε^t) je suma všech vah w_i^t ze špatně určených tříd (missclasified instances)
- pokud je $\varepsilon^t = 0$ nebo $\varepsilon^t \geq \frac{1}{2}$, je proces ukončen
- jinak jsou váhy w_i^{t+1} pro další vzorky nastaveny tak, že četnost chyb u g^t na trénovací sadě užívající váhy w_i^{t+1} je přesně $\frac{1}{2}$

Konečný složený klasifikátor g^* je vytvořen kombinací klasifikátorů g^t za použití hlasování *[4]*

Komentáře k adaBoostingu:

- Freud a Schapire prokázali, že četnost chyb u g^* na nezvážených trénovacích instancích se přibližuje nule exponenciálně rychle podle nárustu T. *[8]*
- Breiman poznamenal, že Boosting nutí klasifikátory mít 50% četnost chyb na znovuzvážených trénovacích instancích, což způsobuje že učící systém produkuje na dalším vzorku trochu jiné klasifikátory. To vede k rozsáhlým zkoumáním ve světě klasifikátorů. *[5]*
- Quinlan provedl několik testu s Boostingem a zjistil, že výhody této metody se vytrácejí pokud zde je nezanedbatelný šum v učící sadě. To je neočekávané, protože by Boosting měl bý díky používání kombinace klasifikátorů robustni na přítomnost šumu. Dále věří že důvod, proč Boosting zvyšuje přesnost klasifikace nemusí nijak souviset s konvergencí jeho vlastností na trénovací data. *[6]*
- Breiman dále poznamenal, že k selhání Boostingu dochází pravděpodobněji s relativně malými datovými sadami. *[5]*

6.2 Přesnost modelu

6.2.1 Bootstrap

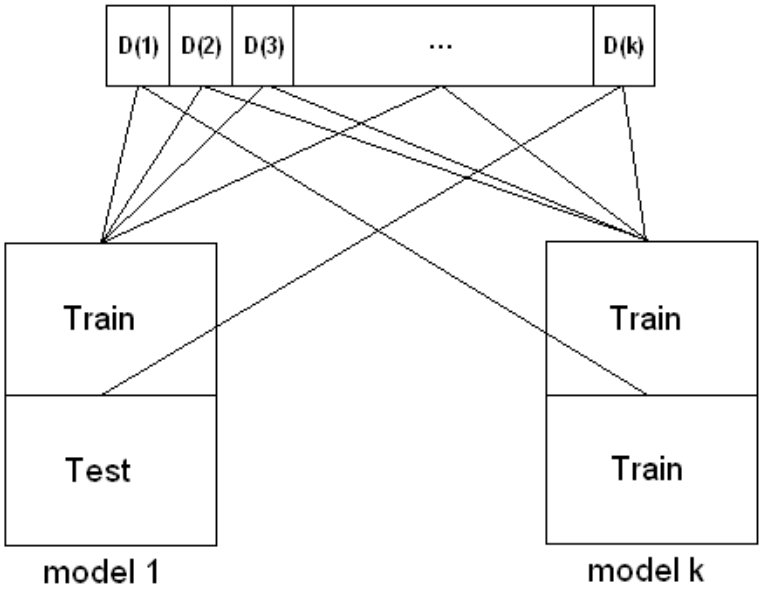
Jedná se o postup pro rozdělení dat na trénovací a testovací při nedostatu dostupných dat a následné spočítání chyby modelu. Ze dostupného souboru dat výběrem s opakováním vygenerujeme velké množství trénovacích souborů. Obecně se doporučuje asi 2000 souborů (může být ale i vyšší).

Výpočet chyby(data o N záznamech a B_i výběrů s opakováním):

$\hat{Err} = \frac{1}{N} \cdot \frac{1}{|B|} \sum_{j=1}^{|B|} \sum_{i=1}^{N} LF(y_i, \tilde{f}^{Bj}(x_i))$, kde $\tilde{f}^{Bj}$ je model naučený na B_j výběr

6.2.2 Cross Validation

Princip této metody spočívá v rozdělení dat na k stejně velkých množin (běžně k = 2, 5, 10) . Vždy jsou z těchto množin sestavena trénovací data z $\frac{k-1}{k}$ množství dat a data testovací ze zbylé $\frac{1}{k}$. Poté pro všechny kombinace sestaveno k modelů které jsou naučeny na trénovací data tak, že jsou vždy testovací data sestavena z jiné kombinace k množin. Tím získáme k různě nastavených modelů. Celková chyba se získá jako aritmetický průměr chyb jednotlivých modelů: $Err = \frac{\sum Err_k}{k}$



Obr 9. Schéma Cross Validation

Nejčastěji se používá tzv. Tenfold Cross Validation, tedy k=10. Jedná se o poměrně přesnou metodu, která ovšem v některých případech tzv. nadhodnocuje chybu. Pro eliminaci tohoto jevu je vhodné zvýšit hodnotu k.

7. Vypracování:

K vypracování zadaného úkolu byli zvoleny programy MATLAB a RapidMiner.

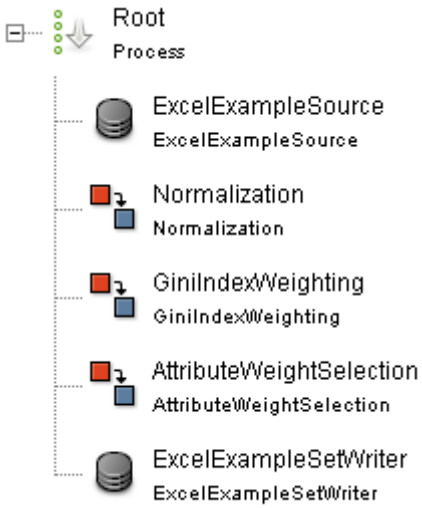
7.1 Vstupní data:

Zdroj dat:

Jako vstupní data byla použita benchmarkova data nalezena na internetu na strance http://theoval.sys.uea.ac.uk/matlab/default.html#benchmarks. Soubor benchmarks.mat obsahuje 13 různých datových sad, všechny jsou binární (třídy jsou kódovány -1/1). Jako nejvhodnější byla zvolena sada s názvem waveform, jelikož obsahuje 5000 záznamů a 21 atributů. Data byla následně rozdělena v polovině na trénovací a testovací.

Předzpracování dat:

Pro předzpracování byl použit program RapidMiner, pro tento účel bylo potřeba převést data do excelovského formátu. Jako vhodné metody předzpracování byly vybrány normalizace a váhování s následnou selekcí. Normalizace upraví jednotlivé hodnoty na předem určený rozsah a váhování spolu se selekcí vyberou pouze ty atributy, které se na výsledné veličině projevují nejvíce. V tomto případě bylo z 21 atributů vybráno 11 nejvhodnějších. Soubor předzpracovaných dat byl opět uložen do excelovského formátu, který bylo potřeba importovat a převést na vhodný tvar do programu MATLAB



Obr 10. Předzpracování dat v RapidMineru

Pro většinu modelů tvořených pomocí programu RapidMiner a pro rozhodovací strom CART vytvořený v programu MATLAB bylo potřeba upravit tvar kódování tříd. Třída -1 byla nahrazena Negat a třída 1 byla nahrazena Positive

7.2 adaBoost:

Pro metodu adaBoost byl použit jako jednoduchý model Decision stump. Algoritmus adaBoost je rozdělen do několika m-file souborů, kde každý obstarává jinou část programu.

7.2.1. Algoritmus:

onedim_stump.m a **d_stump** obsahují algoritmus učení modelu decision stump
evaluace_d_stump.m obsahuje algoritmus výpočtu decision stump
adaboost.m obsahuje algoritmus učení samotného modelu adaBoostu
evaluace_adaboost.m se stará o výpočet adaboostu
run_adaboost.m slouží ke spuštění algoritmu adaboostu a vykreslení grafu
crossval.m je určen k výpočtu Cross Validation (konkrétně náhodný výběr dat)
run_crossval_adaboost.m spouští výpočet chyby pomocí metody Cross Validation

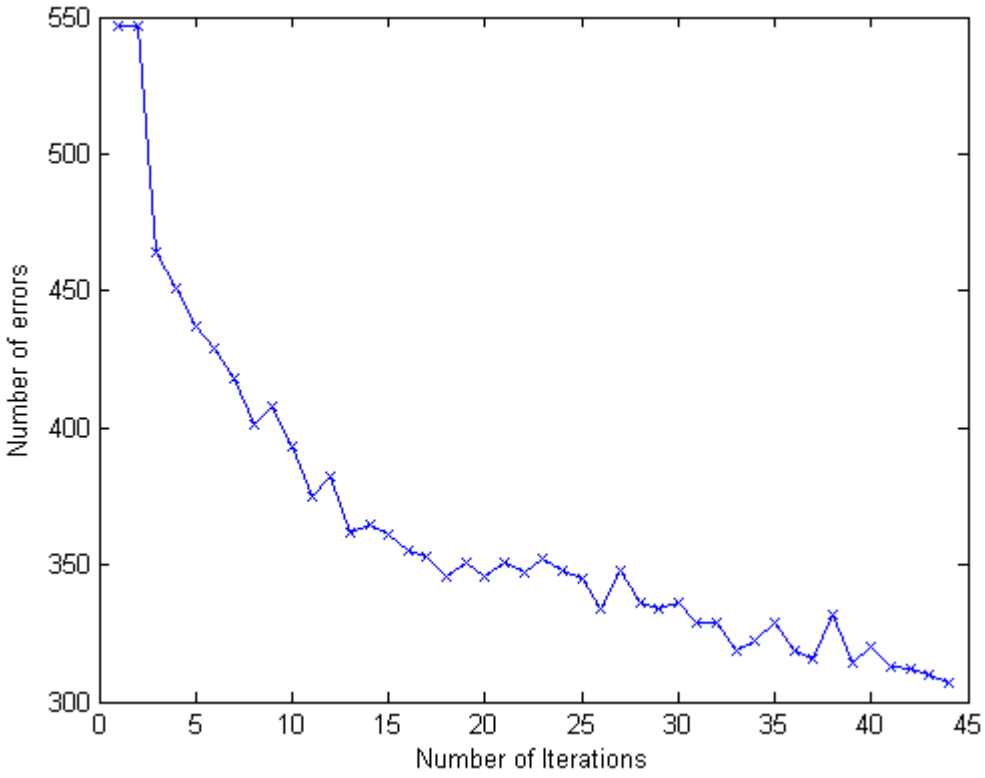
Všechny tyto algoritmy jsou uvedeny v přílohách

7.2.2. Výsledky:

Tento algoritmus vykazuje nejlepší výsledek při 44. iteracích, kdy je z 2500 vzorků špatně klasifikováno 307 což odpovídá chybě 12,28%

Při použití metody Cross Validation (run_crossval_adaboost.m) je zjištěna chyba 12,72% a přesnost 87,28%

Graf:



Obr 11. Graf závislosti množství chyb na počtu iterací

7.3 Další metody Strojového učení

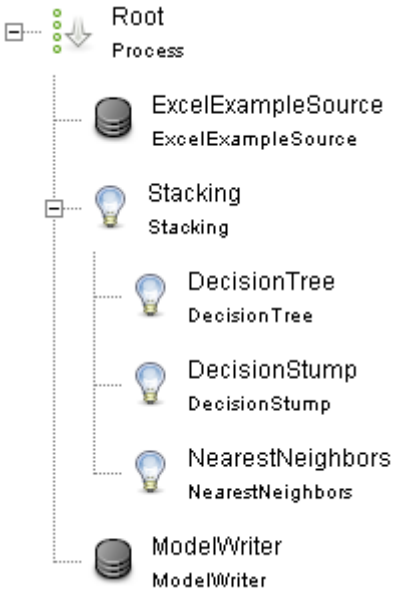
RapidMiner:

Jedná se o software přímo určený pro strojové učení, program je volně stažitelný ze stránek www.rapidminer.com Tento program byl použit na následující metody:

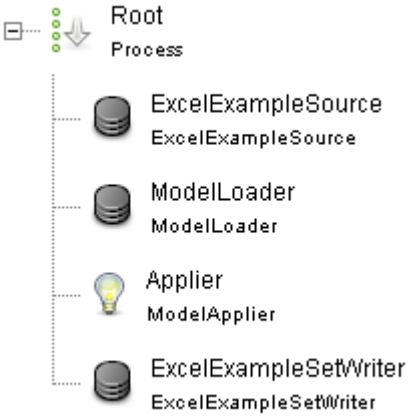
- Meta-Lerning metody
  - Stacking
  - Bagging
  - adaBoosting
- Běžné modely
  - Rozhodovací stromy ID3
  - k-Nejbližších sousedů (kNN)
  - Naivní Bayesův klasifikátor
  - Decision stump

7.3.1 Stacking:

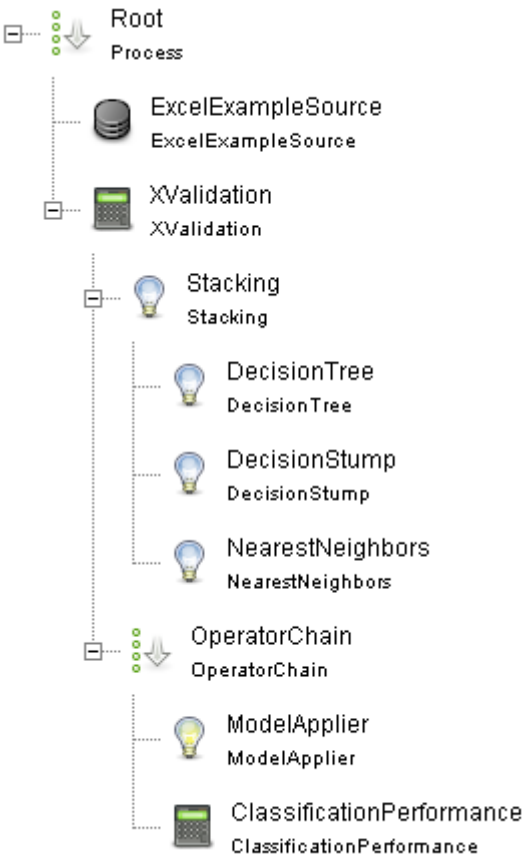
Pro metodu Meta-learningu Stacking byly aplikovány modely Rozhodovacího stromu, Decision stump a k-Nejbližších sousedů. Učení viz. Obrázek… Na dalším obrázku… je znázorněno jakým způsobem je naučený model aplikován na testovací data. Poslední obrázek… představuje metodu Cross Validation, která slouží k určení skutečné chyby modelu. Při testování modelu bylo 134 vzorků klasifikováno chybně, což činí 5,36% z celkových 2500 vzorků. Metoda Cross Validation stanovila chybu na 14,2% a přesnost na 85,8%.



Obr 12. Učení modelu Stacking



Obr 13. Aplikace na testovací data



Obr 14. Cross Validation pro Stacking

7.3.2 Bagging:

Metoda Bagging byla aplikována na model Naivního Bayesovského klasifikátoru. (Obrázky učení, testování a Cross Vaidation jsou uvedeny v přílohách) Nastavení bylo: Sample ratio = 0,9 a počet iterací = 10. Při tomto nastavení bylo chybně klasifikováno 554 vzorů dat což je 22,16%. Metodou Cross Validation byla chyba stanovena na 21,04% a přesnost na 78,96%

7.3.3 adaBoosting:

Pro metodu adaBoosting byly stejné jako v případě algoritmu vytvořeném v programu MATLAB použity jednoduché modely Decision stump. (Obrázky učení, testování a Cross Vaidation jsou uvedeny v přílohách) Při tomto nastavení bylo

chybně klasifikováno 610 vzorů dat což je 24,4%. Metodou Cross Validation byla chyba stanovena na 26,16% a přesnost na 73,84%

7.3.4 Rozhodovací stromy ID3:

Při procesu učení byl model rozhodovacího stromu ID3 nastaven pro minimální velikost listu na hodnotu 3 (nejlepší výsledky pro tento případ). Při testování naučeného modelu bylo 415 vzorků z 2500 klasifikováno nesprávně což je 16,6%. Cross Validation určil chybu na 16,5% a přesnost na 83,6%.

7.3.5 k-Nejbližších sousedů:

Při procesu učení byl model k-Nejbližších sousedů nastaven k na hodnotu 3 (nejlepší výsledky pro tento případ). Při testování naučeného modelu bylo 317 vzorků z 2500 klasifikováno nesprávně což je 12,68%. Cross Validation určil chybu na 12,84% a přesnost na 87,16%.

7.3.6 Naivní Bayesův klasifikátor:

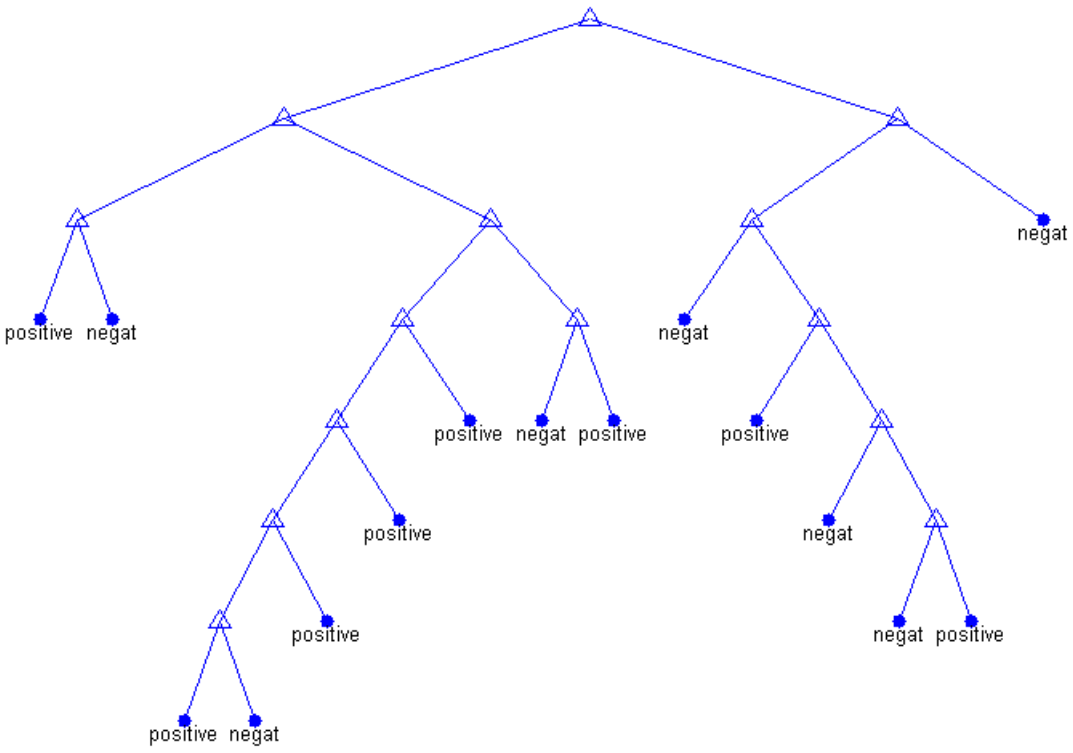
Při procesu učení byl použit model Naivniho Bayesovskeho klasifikátoru. Při testování naučeného modelu bylo 551 vzorků z 2500 klasifikováno nesprávně což je 22,04%. Cross Validation určil chybu na 20,8% a přesnost na 79,2%.

7.3.7 Decision stumps:

Při procesu učení byl použit model Decision stump s nastavenou minimální velikostí listu 3. Při testování naučeného modelu bylo 694 vzorků z 2500 klasifikováno nesprávně což je 27,76%. Cross Validation určil chybu na 26,36% a přesnost na 73,64%.

7.3.8 Rozhodovací strom CART:

Pro vytvoření rozhodovacího stromu byla použita funkce treefit z programu MATLAB, tato funkce vytvoří rozhodovací strom algoritmem CART. Parametry funkce jsou: t = treefit(data.xtrain,data.ytrain,'splitmin',250), kde data.xtrain je matice vzorků a data.ytrain obsahuje odpovídající třídy. Hodnoty split 250 pro prořezávání stromu bylo dosaženo experimentálně. Kritériem byl optimální poměr mezi přesností modelu a složitostí stromu. K zobrazeni byla použita funkce treedisp(t) s parametrem t, který obsahuje výsledné prvky navržené pomocí algoritmu(viz. Obr…). Pro aplikaci naučeného stromu na testovaci data byla použita funkce treeval: sfit = treeval(t, data.xtest). Chyba modelu je vypočtena pomocí metody Cross Validation (viz. přílohy).



Obr 15. Rozhodovací strom CART

Algoritmus:

CART.m naučí a aplikuje model CART na data a vykreslí rozhodovací strom
run_crossval_CART.m obsahuje algoritmu pro výpočet chyby a přesnosti modelu metodou Cross Validation

Výsledky:

Tento algoritmus vykazuje správně predikuje 81,84% dat, což odpovídá chybě 18,16%. Při použití metody Cross Validation (run_crossval_CART.m) je zjištěna chyba 17,64% a přesnost 82,36%

7.4 Cross Validation

Jak již bylo uvedeno v kapitole 5.2 princip metody Cross Vaidation spočívá v rozdělení dat na k množin. V našem případě bylo k=10, tedy trénovací data tvořila 9/10 celkových dat a testovací byla ze zbylé 1/10.

V programu MATLAB bylo potřeba naprogramovat, aby toto rozdělení bylo náhodné. Celý algoritmus rozdělení dat spolu s trénováním, testování a počítáním chyby je uveden v příloze…

7.5 Porovnání výsledků jednotlivých modelů

V tabulce jsou uvedeny všechny vytvořené modely i s programy, které byli pro tvorbu použity. Hodnota E[%] představuje chybu modelu vyjádřenou v procentech a Acc[%] přesnost, taktéž v procentech. Tyto hodnoty byli získány metodou Cross Validation s počtem opakování 10.

metoda	program	E [%]	Acc [%]	Misclass
adaBoost	Matlab	10,68	89,32	307
Stacking	RapidMiner	14,2	85,8	134
Bagging	RapidMiner	21,04	78,96	554
adaBoost	RapidMiner	26,16	73,84	610
ID3	RapidMiner	16,4	83,6	415
kNN	RapidMiner	12,84	87,16	317
Bayes	RapidMiner	20,8	79,2	551
Decision Stump	RapidMiner	26,36	73,64	694
CART	Matlab	17,64	82,36	454

Tabulka 1. Porovnání jednotlivých metod strojového učení

8. Závěr:

Tato práce se věnuje rešerši metod Meta-Learningu a tvorbou algoritmu metody adaBoosting. Nejprve jsou uvedeny základní metody strojového učení. Prvotním těžištěm práce je popis metod Meta-Learningu jako jsou Bumping, Stacking, Bagging, Boosting, které tyto základní modely používají.

Hlavním úkolem této práce byla tvorba algoritmu metody adaBoost, který je vytvořen v programu MATLAB. Dále bylo v programu MATLAB vytvořen rozhodovací strom metodou CART a program na posouzení kvality modelu tzv. Cross Validation. Tento algoritmus vypočte chybu a přesnost modelu. Další metody využívající Meta-Learning jako Bagging, Stacking a opět adaBoosting byli vytvořeny v programu RapidMiner spolu s ostatními metodami, které Meta-Learning nevyužívají. Patří tam: Rozhodovací stromy vytvořené metodou ID3, Decision stumps, Naivní Bayesův klasifikátor a k-Nejbližších sousedů. Výstupy všech těchto modelů jsou uvedeny v tabulce. Z výsledků je vidět, že v matlabu vytvořený algoritmus je podle metody ověření kvality modelu nejpřesnější. Pouze při aplikaci na testovací data, vykazovala metoda Stacking vytvořená v RapidMineru menší množství špatně predikovaných dat.

9. Použitá literatura

[1] P.Honzík, Strojové učení, skripta a přednáškyVUT 2006

[2] L.Alexandre, Boosting and Bagging, 2004

[3] R.Schapire, The Boosting Approach to Machina learning

[4] Y.Freud, R.Schapire, A short introduction to Boosting, 1999

[5] L.Breiman, Arcing classifers. Technical Report 460 statistics 1996.

[6] L.Breiman, Bagging predictors. MachineLearning, 1996.

[7] J.R.Quinlan, Bagging,Boosting, and C4.5 1996.

[8] J.R.Quinlan, Boosting first-order learning 1996

[9] W.W.Cohen V.R.Carvalho, Stacked Seuential Learning

[10] L.Breiman, Random Forest, Machina Learning, 2001

[11] V.Jirsík P.Hráček Neuronové sítě, expertní systémy a rozpoznávání řeči

[12] P.Petyovský, Klasifikátory Strojové učení Automatické třídění 1

10. Seznam příloh

10.1 Příloha 1: Algoritmus adaBoostingu

d_stump.m

```
function [stump] = d_stump(X,y,w)

d = size(X,2);
w = w/sum(w);
stump = cell(d,1);
werr = zeros(d,1);

for i=1:d,
  stump{i} = onedim_stump(X(:,i),y,w);
  stump{i}.ind = i;
  werr(i) = stump{i}.werr;
end

[min_werr,ind] = min(werr);
stump = stump{ind(1)};
```

onedim_stumo.m

```
function [stump] = onedim_stump(x,y,w)

[xsorted,I] = sort(x);
Ir = I(end:-1:1);
score_left  = cumsum(w(I).*y(I));
score_right = cumsum(w(Ir).*y(Ir));
score = -score_left(1:end-1) + score_right(end-1:-1:1);
Idec = find( xsorted(1:end-1)<xsorted(2:end) );


if (length(Idec)>0),
  [maxscore,ind] = max( abs(score(Idec)) );
  ind = Idec(ind(1));
  stump.werr = 0.5-0.5*maxscore;
  stump.x0   = (xsorted(ind)+xsorted(ind+1))/2;
  stump.s    = sign(score(ind));
else
  stump.werr = 0.5;
  stump.x0   = 0;
  stump.s    = 1;
end
```

adaboost.m

```
function [model] = adaboost(X,y,n)

W = ones(size(X,1),1);
W = W/sum(W);
alfa = 1;
model = cell(n,1);
H = zeros(size(X,1),1);

for i=1:n
    stump = d_stump(X,y,W);

    if (stump.werr<0.5)
        h = evaluate_d_stump(stump,X);
        alfa = 0.5*log((1-stump.werr)/stump.werr);
        H = H + alfa*h;

        W=exp(-H.*y);
        W=W/sum(W);;

        model{i} = stump;
        model{i}.alfa = alfa;
    else
        i = i-1;
    end
end
model = model(1:i);
```

evaluate_adaboost.m

```
function [h,sum_alfa] = evaluate_adaboost(model,X);
h = zeros(size(X,1),1);
sum_alfa = 0;
for i=1:length(model)
  h = h + model{i}.alfa*evaluate_d_stump(model{i},X);
  sum_alfa = sum_alfa + model{i}.alfa;
end
```

evaluate_d_stump.m

```
function [h] = evaluate_d_stump(stump,X);
h = sign( stump.s*(X(:,stump.ind)-stump.x0));
```

run_adaboost.m

```
clear all;
close all;
clc;

load waveformDATA.mat
iter=44;
error=zeros(iter,1);

model=adaboost(data.xtrain(:,1:11),data.ytrain,iter);

for k=1:iter
    y_pred=sign(evaluate_adaboost(model(1:k),data.xtest(:,1:11)));
    error(k)=sum(y_pred ~= data.ytest);
end

plot(error,'x-');
ylabel('Number of errors');
xlabel('Number of Iterations');
```

10.2 Příloha 2: Algoritmus CART

CART.m

```
clear all
close all
clc

load dataprostrom.mat

t = treefit(data.xtrain, data.ytrain,'splitmin',250);
treedisp(t)
sfit = treeval(t, data.xtrain);
sfit = t.classname(sfit);
mean(strcmp(sfit,data.ytrain));

sfit = treeval(t, data.xtest);
sfit = t.classname(sfit);
err=1-(mean(strcmp(sfit,data.ytest)));

E_per=err*100
Acc_per=100-E_per
```

10.3 Příloha 3: Algoritmus Cross Validation

run_crossval_adaboost.m

```
clear all;
close all;
clc;

load waveformDATA.mat;
ldata=length(data.xtrain);
knum=10;
num_iter=44;
[itrain,itest]=crossval(ldata,knum);
ltest=length(itest{1});
ltrain=length(itrain{1});
E=0;

for i=1:knum
    train=itrain{i};
    test=itest{i};

    for l=1:ltrain
        xdata.xtrain(l,:)=data.xtrain(train(l),:);
        xdata.ytrain(l,1)=data.ytrain(train(l),1);
    end

    for l=1:ltest
        xdata.xtest(l,:)=data.xtest(train(l),:);
        xdata.ytest(l,1)=data.ytest(train(l),1);
    end

    model=adaboost(xdata.xtrain(:,1:11),xdata.ytrain,num_iter);

    for k=1:num_iter

y_pred=sign(evaluate_adaboost(model(1:k),xdata.xtest(:,1:11)));
        error(k)=sum(y_pred ~= xdata.ytest);
    end

    Ferr= error(end)/ltest;
    E=E+Ferr;
end

E_abs=E/knum
E_per=E_abs*100
Acc_per=100-E_per
```

run_crossval_CART.m

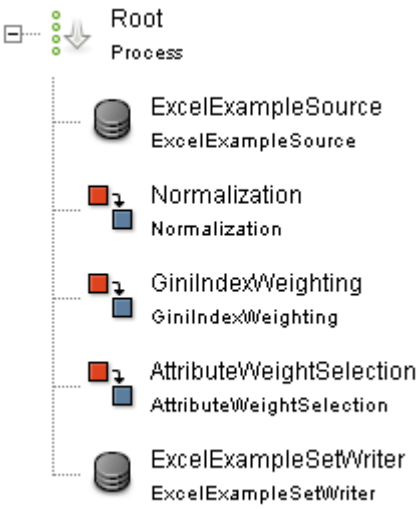
```
clear all;
close all;
clc;

load dataprostrom.mat
ldata=length(data.xtrain);
knum=10;
num_iter=44;
[itrain,itest]=crossval(ldata,knum);
ltest=length(itest{1});
ltrain=length(itrain{1});
E=0;
FE=0;

for i=1:knum
    train=itrain{i};
    test=itest{i};

    for l=1:ltrain
        xdata.xtrain(l,:)=data.xtrain(train(l),:);
        xdata.ytrain(l,1)=data.ytrain(train(l),1);
    end

    for l=1:ltest
        xdata.xtest(l,:)=data.xtest(train(l),:);
        xdata.ytest(l,1)=data.ytest(train(l),1);
    end

        t = treefit(data.xtrain, data.ytrain,'splitmin',250);
        sfit = treeval(t, xdata.xtest);
        sfit = t.classname(sfit);
        E = 1-(mean(strcmp(sfit,xdata.ytest)));

        FE = FE+E;
end

E_abs=FE/knum
E_per=E_abs*100
Acc_per=100-E_per
```

crossval.m

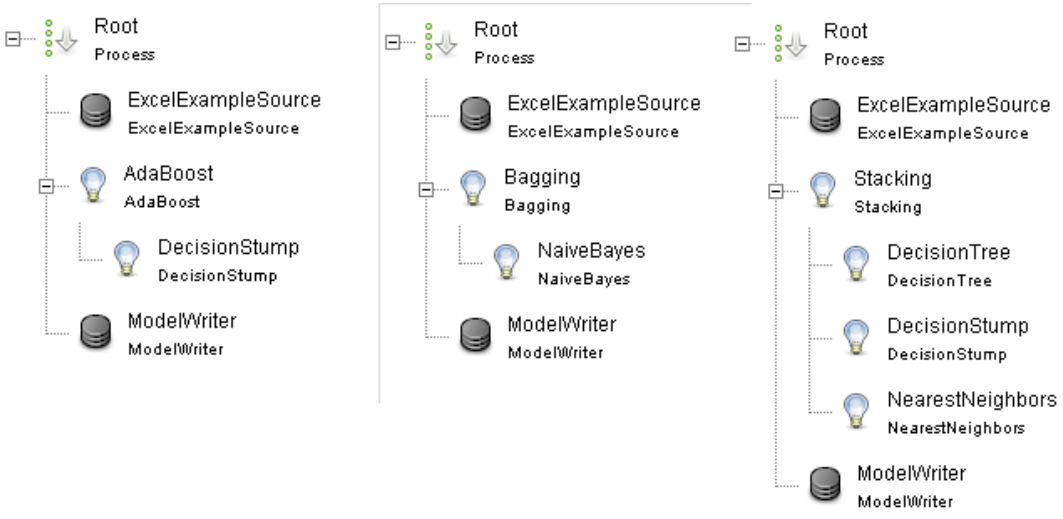
```
function [itrain,itest]=crossval(data,knum)

inx=randperm(data);
itrain=cell(1,knum);
itest=cell(1,knum);
column=ceil(data/knum);
p=[1:data zeros(1,column*knum-data)];
p=reshape(p,column,knum)';

for i=1:knum,
  test_inx=p(i,:);
  test_inx=test_inx(find(test_inx));
  train_inx=p(find([1:knum]~=i),:);
  train_inx=train_inx(find(train_inx));
  itrain{i}=inx(train_inx);
  itest{i}=inx(test_inx);
end
```

10.4 Příloha 4: RapidMiner modely

Předzpracování dat



Učení Meta-Learning modelů



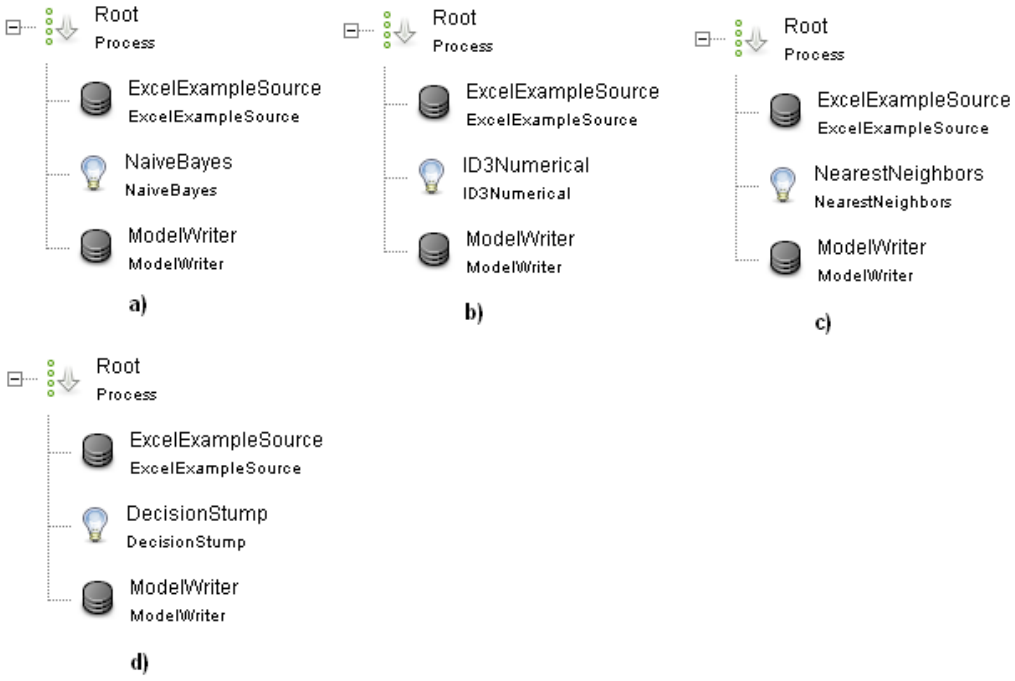
a) b) c)

a) **adaBoosting**

b) **Bagging**

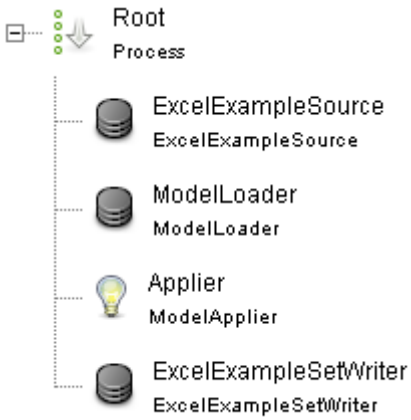
c) **Stacking**

Učení modelů nevyužívajících Meta-Learning

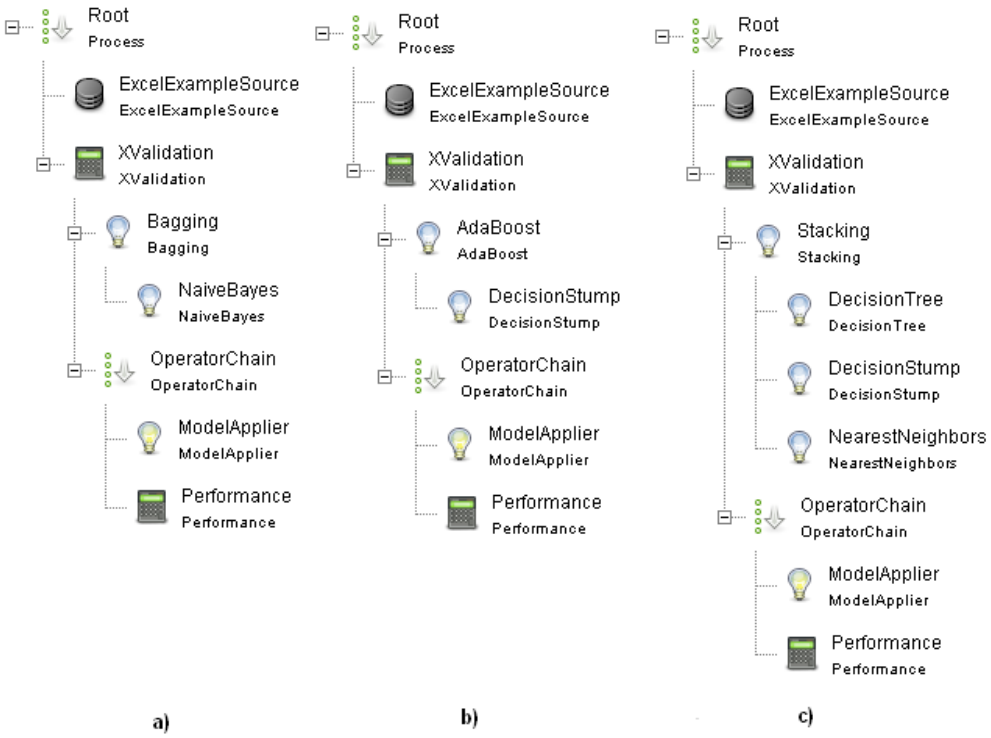


a) **Naivní Bayesův klasifikátor**
b) **Rozhodovací strom, metoda ID3**
c) **k-Nejbližších sousedů**
d) **Decision Stump**

Testování naučených modelů



Cross Validation pro Meta-Learning modely



a) **Bagging**

b) **adaBoosting**

c) **Stacking**

Cross Validation pro modely nevyužívající Meta-Learning

a) **Naivní Bayesův klasifikátor**

b) **Rozhodovací strom, metoda ID3**

c) **k-Nejbližších sousedů**

d) **Decision Stump**