Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

# Bakalářská práce

# Metody strojové klasifikace pro výběr optimálního estimátoru křivosti

Plzeň 2019 Filip Hácha

ZÁPADOČESKÁ UNIVERZITA V PLZNI
**Fakulta aplikovaných věd**
Akademický rok: **2018/2019**

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Filip HÁCHA** |
| Osobní číslo: | **A16B0036P** |
| Studijní program: | **B3902 Inženýrská informatika** |
| Studijní obor: | **Informatika** |
| Název tématu: | **Metody strojové klasifikace pro výběr optimálního estimátoru křivosti** |
| Zadávající katedra: | **Katedra informatiky a výpočetní techniky** |

Zásady pro vypracování:

1. Seznamte se s daty získanými automatickým testováním metod odhadu křivosti trojúhelníkových sítí.
2. Seznamte se s metodami strojové klasifikace a navrhněte způsoby, jak automaticky z naměřených vlastností určit efektivní estimátor. Uvažte vhodné metody redukce dimenze prostoru vlastností, zejména PCA.
3. Analyzujte teoreticky vlastnosti navržených postupů a zvolte vhodné metody pro experimentální implementaci.
4. Implementujte zvolené metody a ověřte jejich efektivitu na dostupných datech.
5. Zdokumentujte a analyzujte dosažené výsledky.

Rozsah grafických prací: **dle potřeby**

Rozsah kvalifikační práce: **doporuč. 30 s. původního textu**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

**Dodá vedoucí bakalářské práce.**

Vedoucí bakalářské práce: **Doc. Ing. Libor Váša, Ph.D.**
Nové technologie pro informační společnost

Datum zadání bakalářské práce: **10. října 2018**

Termín odevzdání bakalářské práce: **2. května 2019**

Doc. Dr. Ing. Vlasta Radová
děkanka

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
L.S.

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 15. října 2018

# Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V bakalářské práci jsou použity názvy programových produktů, firem apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

V Plzni dne 29. dubna 2019

Filip Hácha

# Poděkování

Děkuji panu doc. Ing. Liborovi Vášovi, Ph.D. za ochotu při vedení bakalářské práce a rady s jejím vypracováním. Dále děkuji panu doc. Ing. Pavlovi Královi, Ph.D. za poskytnutí konzultací k problematice neuronových sítí.

# Abstract

Machine learning methods of selection optimal curvature estimator.
The goal of this thesis is to design and create mechanism for selecting optimal curvature estimator for polygonal meshes. Several approaches of machine learning, such as decision trees, neural networks, clustering methods and linear regression have been tried to solve this problem. The created solution allows prediction of the optimal estimator based on the properties of a polygonal mesh with higher success than mechanism implemented in the research, which is directly linked to this work.



# Abstrakt

Cílem této práce je návrh a konstrukce mechanismu pro výběr optimálního estimátoru křivosti polygonálních sítí.

Pro vyřešení problému bylo vyzkoušeno několik přístupů z oblasti strojového učení, jako rozhodovací stromy, neuronové sítě, metody shlukování a lineární regrese.

Vytvořené řešení umožňuje predikci estimátoru na základě vlastností polygonální sítě s vyšší úspěšností, než nabízí využití mechanismům implementovaných ve výzkumu, na který tato práce přímo navazuje.

# Obsah


**1 Úvod** **1**

**2 Diferenciální geometrie** **2**
2.1 Křivka . . . . . 2
2.2 Plocha . . . . . 2
2.3 Křivost . . . . . 3
2.4 Křivost plochy . . . . . 3

**3 Polygonální sítě** **5**
3.1 Trojúhelníkové sítě . . . . . 5
3.2 Křivost v trojúhelníkových sítích . . . . . 5
3.3 Aplikace křivosti v praxi . . . . . 6

**4 Strojové učení** **8**
4.1 Metody učení . . . . . 8
4.2 Volba reprezentace . . . . . 8
4.3 Základní druhy úloh . . . . . 9

**5 Popis úlohy** **10**

**6 Předzpracování dat** **11**
6.1 Normalizace vlastností . . . . . 12
6.1.1 Normalizace rozpětím . . . . . 12
6.1.2 Normalizace pomocí směrodatné odchylky . . . . . 12
6.2 Analýza hlavních komponent . . . . . 12

**7 Metody klasifikace** **14**
7.1 Rozhodovací strom . . . . . 14
7.1.1 Jednovrstvý rozhodovací strom . . . . . 14
7.1.2 Dvouvrstvý rozhodovací strom . . . . . 15
7.1.3 Použití genetických algoritmů pro konstrukci víceúrovňového rozhodovacího stromu . . . . . 16
7.1.4 Simulované žíhání . . . . . 16
7.2 Neuronová síť . . . . . 17
7.3 Shlukování . . . . . 19
7.4 Aproximace metodou nejmenších čtverců . . . . . 20

**8 Metody vyhodnocení** **23**
8.1 Relativní chyba . . . . . 23
8.2 Přesnost . . . . . 23

**9 Implementace** **25**
9.1 Konfigurace výběru algoritmů . . . . . 25
9.2 Vyhodnocení vybraných algoritmů . . . . . 26
9.3 Služba pro vyhodnocení úspěšnosti metod klasifikace . . . . 27
9.4 Parsování vstupních dat . . . . . 29
9.5 Aplikace pro výběr optimálního estimátoru . . . . . 30
9.6 Rozhraní mezi prostředím .NET a Python . . . . . 31
9.7 Logování . . . . . 32
9.8 Reprezentace dat . . . . . 32
9.9 Implementace algoritmů . . . . . 32
9.9.1 Algoritmy pro předzpracování dat . . . . . 32
9.9.2 Algoritmy pro klasifikaci dat . . . . . 33
9.9.3 Metody vyhodnocení . . . . . 36

**10 Uživatelská příručka** **37**
10.1 Zdrojové kódy . . . . . 37
10.2 Překlad a instalace databáze . . . . . 38
10.3 Služba pro vyhodnocení úspěšnosti metod klasifikace . . . . 38
10.4 Aplikace pro výběr optimálního estimátoru . . . . . 38
10.5 Nastavení aplikací . . . . . 39

**11 Výsledky** **40**

**12 Závěr** **41**

**Literatura** **42**

**A Obsah CD** **45**

**B Kompletní výsledky** **46**

# 1 Úvod

Obsah této práce navazuje na výsledky výzkumu v oblasti metod pro odhad křivosti v polygonálních sítích, které publikovali L. Váša, P. Vaneček, M. Prantl, V. Skorkovská, P. Martínek a I. Kolingerová v roce 2016 na Eurographics Symposium on Geometry Processing [20], kde byla popsána úspěšnost klasifikátoru polygonálních sítí do tříd podle optimální metody odhadu křivosti realizovaného pomocí úplného jednovrstvého a neúplného dvouvrstvého binárního rozhodovacího stromu.

Všechny metody klasifikace uvedené v této práci byly vyhodnocovány nad téměř stejnou množinou testovacích dat, díky čemuž jsou výsledky dosažené v této práci snadno porovnatelné s výsledky dosaženými v rámci předešlého výzkumu.

Pro sestavení klasifikátoru bylo vyzkoušeny metody klasifikace pomocí rozhodovacího stromu, neuronové sítě, shlukování a lineární regrese. Práce dále zahrnuje implementaci algoritmů pro normalizaci vstupních dat, konkrétně analýzu hlavních komponent, normalizaci rozpětím a normalizaci pomocí směrodatné odchylky.

# 2 Diferenciální geometrie

Diferenciální geometrie je věda zabývající se popisem geometrických objektů pomocí diferenciálního počtu. Základními geometrickými objekty, kterými se tato disciplína zabývá jsou křivky a plochy v trojrozměrném euklidovském prostoru. [14]

Otázky, kterými se diferenciální geometrie zabývá, můžeme rozdělit zpravidla na dva druhy. První část otázek se týká lokálních vlastností geometrických objektů. Lokálními vlastnostmi objektů myslíme vlastnosti křivek a ploch definované na okolí bodu. Druhá skupina otázek je tvořena studováním globálních vlastností, které se zabývají geometrickými objekty jako celky. [2] Jednou z lokálních vlastností, kterou se diferenciální geometrie zabývá je křivost křivek a ploch.

## 2.1 Křivka

Uvažujeme hladké rovinné křivky, tj. diferencovatelné 1-rozměrné objekty vložené do prostoru $\mathbb{R}^2$. Taková křivka může být reprezentována parametrickou formou pomocí vektorové funkce $\boldsymbol{x} : [a, b] \to \mathbb{R}^2$, $\boldsymbol{x}(u) = (x(u), y(u))^T$ pro $u \in [a, b] \subset \boldsymbol{x}$.

Předpokládejme, že souřadnice x a y jsou diferencovatelné funkce parametru $u$. Vektor tečny $\boldsymbol{x}'(u)$ ke křivce v bodě $\boldsymbol{x}(u)$ je definován jako první derivace funkce souřadnic, tj. $\boldsymbol{x}'(u) = (x'(u), y'(u))^T$.

Například v bodové mechanice trajektorie bodu je křivka parametrizovaná časem ($u = t$) a tečna $x'(t)$ odpovídá vektoru rychlosti v čase $t$.

Je-li křivka definována jako obraz funkce $\boldsymbol{x}$, můžeme získat stejnou křivku použitím rozdílných parametrizací. [6]

## 2.2 Plocha

Regulární plochou nazveme podmnožinu $S \subset \mathbb{R}^3$, platí-li, že pro každé $p \in S$ existuje okolí $V$ v $\mathbb{R}^3$ a zobrazení $\boldsymbol{x} : U \to V \cap S$ otevřeného intervalu $U \subset R^2$ na $V \cap S \ \subset R^3$ takové, že:

1. Zobrazení $\mathbf{x}$ je diferencovatelné, což znamená, že můžeme napsat

$$\boldsymbol{x}(u, v) = (x(u, v), y(u, v), z(u, v)), (u, v) \in U \tag{2.1}$$

kde zobrazení $x(u,v)$, $y(u,v)$, $z(u,v)$ mají spojité parciální derivace podle obou složek.

2. Zobrazení $\boldsymbol{x}$ je homeomorfismus. Je-li $\boldsymbol{x}$ spojité podle podmínky (1), znamená to, že k němu existuje inverzní zobrazení $\boldsymbol{x}^{-1} : V \cap S \rightarrow U$, které je taktéž spojité. Inverzní zobrazení $\boldsymbol{x}^{-1}$ je tedy restrikcí spojitého zobrazení $F : W \subset \mathbb{R}^3 \rightarrow \mathbb{R}^2$ definovaného na otevřeném intervalu $W$ obsahujícím $V \cap S$.

3. Pro všechna $q \in U$ je zobrazení $dx_q : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ prosté.

Zobrazení $\boldsymbol{x}$ se nazývá parametrizací nebo lokálním systémem souřadnic, v okolí bodu $p$. [10]

## 2.3 Křivost

Jednou z důležitých charakteristik křivek je křivost. Křivost v daném bodu určuje míru vychýlení křivky od tečny. Je-li křivka parametrizována obloukem, pak je křivost přímo rovna velikosti vektoru druhé derivace křivky v bodě.

Parametrizace obloukem představuje pohyb po křivce který je "rovnoměrný ve smyslu velikosti rychlosti". Pro takovou parametrizaci $\boldsymbol{x(t)}$ platí, že velikost její první derivace podle dráhy $s$ je rovna jedné. (Vzorec 2.2)

$$\left\| \frac{d\boldsymbol{x}}{ds} \right\| = \left\| \frac{d\boldsymbol{x}}{dt} \right\| \left\| \frac{dt}{ds} \right\| = 1 \tag{2.2}$$

Převrácená hodnota křivosti určuje poloměr křivosti křivky v daném bodě, tj. poloměr oskulační kružnice sestrojené ke křivce v daném bodě. [14]

## 2.4 Křivost plochy

Při řezu plochy v libovolném směru bodem $T$ získáme křivku $y$. Tvoří-li tečny všech křivek $k_i$ rovinu $\tau$, pak bod $T$ nazveme regulárním bodem plochy a rovinu $\tau$ tečnou rovinou plochy.

Přímku $\overrightarrow{n}$, která je ve všech směrech kolmá na tečnou rovinu $\tau$ nazveme normálou plochy. Po sestrojení normály plochy $\overrightarrow{n}$ a výběru konkrétního směru řezu plochou definující křivku $y$ můžeme vypočítat normálovou křivost $k_n$. (Vzorec 2.3).

$$k_n = y'' \cdot \overrightarrow{n} \tag{2.3}$$

Směry, ve kterých normálová křivost nabývá extrémních hodnot $k_1$, $k_2$ (extrémní normálové křivosti), nazýváme hlavní směry plochy.

Z hodnot extrémních normálových křivostí $k_1$ a $k_2$ jsme dále schopni určit hodnoty Gaussovy křivosti (Vzorec 2.4), střední křivosti (Vzorec 2.5) a absolutní křivosti (Vzorec 2.6). [14][10]

$$G = k_1 \cdot k_2 \tag{2.4}$$

$$H = \frac{k_1 + k_2}{2} \tag{2.5}$$

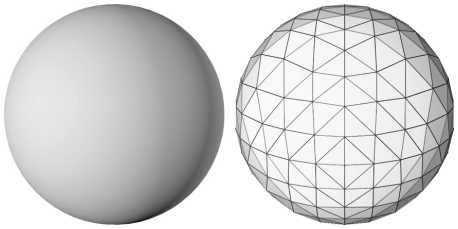
$$K_{ABS} = |k_1| + |k_2| \tag{2.6}$$

# 3 Polygonální sítě

## 3.1 Trojúhelníkové sítě

Pro účely počítačové grafiky a modelování většinou zjednodušujeme původní geometrické objekty aproximací spojitých povrchů pomocí konečného počtu vrcholů. Jedním z nejpoužívanější způsobů reprezentace modelů jsou trojúhelníkové sítě. Obrázek 3.1 ukazuje příklad aproximace původního objektu trojúhelníkovou sítí na kouli.

V mnoha aplikacích uvažujeme trojúhelníkové sítě jako množinu trojúhelníků, bez jakékoli konkrétní matematické struktury. V principu však můžeme každý trojúhelník definovat jako segment lineární reprezentace povrchu. [5]



Obrázek 3.1: Aproximace koule trojúhelníkovou sítí.

## 3.2 Křivost v trojúhelníkových sítích

Křivost je zajímavou vlastností ploch, neboť umožňuje rozpoznání rysů jako jsou údolí, roviny, konvexní, konkávní nebo sedlové tvary. Na základě křivosti můžeme plochy rozdělit do skupin a jejich vlastnosti využít při rozpoznávání objektů v polygonální síti. [9]

Zatímco výpočet křivosti v bodě na hladkých površích běžně nepředstavuje obtížnou úlohu, odhad křivosti v bodech polygonálních sítí je obtížnější úkol, pokud neplatí určité zvláštní okolnosti. [21]

Důvodem, proč je odhad křivosti v polygonálních sítích tak obtížným úkolem, je skutečnost, že při převodu reálného trojrozměrného tělesa na množinu trojúhelníků dojde k nezvratné ztrátě informace o původním tvaru v místech, kde není křivost původního tělesa nulová, neboť původní hladký povrch je zde nahrazen složením několika trojúhelníkových segmentů.

Rozdílná tělesa mohou být reprezentování stejnou trojúhelníkovou sítí, z čehož vyplývá, že nejsme schopni zpětně určit přesnou podobu původního

tělesa, ale můžeme ji pouze odhadovat.

Způsobů, jak odhadovat křivost původního tělesa na základě trojúhelníkové sítě, je více. Existují například algoritmy založené na lokálním proložení trojúhelníkové sítě hladkou plochou, pro níž dovedeme křivost vypočítat. V takovém případě zůstává otázkou, jakou hladkou plochu použít nebo jak konkrétně provést proložení sítě. Navíc pro různá data nám poskytují různé estimátory odhady s rozdílnou přesností.

Jednou z metod pro odhad křivosti je metoda založená na kvadratické aproximaci povrchu, kterou popsali J. Goldfeather a V. Interrante [11], při níž se snažíme najít kvadratický povrch, kterým nejlépe aproximujeme okolí bodu $p$, ve kterém křivost určujeme, hledáním explicitní funkce ve tvaru (Vzorec 3.1)

$$z \ = f(x,y) = \frac{A}{2}x^2 + Bxy + \frac{C}{2}y^2. \tag{3.1}$$

Po nalezení koeficientů explicitní funkce jsou křivosti analyticky vypočteny nad aproximujícím povrchem.

Jednou z dalších metod odhadu křivosti je metoda, která byla popsána pány M. Prantlem a L. Vášou [15], která je založena na hledání implicitního povrchu (Vzorec 3.2) Hermit-Birkhoffově interpolaci bodů RBF[1] funkcí.

Hermit-Birkhoffova interpolace[22] je zobecněný interpolační problém, jehož data se skládají z hodnot přiřazených bodům diferenciálních operátorů reprezentovaných funkcemi, např. $f(x) = c_0^{\boldsymbol{x}}$, $\frac{\partial f}{\partial z}(y) = c_z^{\boldsymbol{z}}$ a $\frac{\partial^2 f}{\partial x \partial y}(z) = c_{xy}^{\boldsymbol{z}}$.

$$f(x,y,z) = c \tag{3.2}$$

Kromě výše zmíněných přístupů existuje ještě řada dalších estimátorů, založených na nejrůznějších principech, jakým je např. odhad tzv. shape operátoru, diskretizace Laplace-Beltrami operátoru, využití integrálních invariantů plochy a dalších. [20] Konkrétní princip funkce estimátoru však není pro tuto práci relevantní, pročež budeme spoléhat na implementace a data, jež jsou k dispozici z předchozích prací věnovaných tomuto tématu.

## 3.3 Aplikace křivosti v praxi

Určování křivosti na tělesech aproximovaných polygonálními sítěmi má v praxi široké spektrum využití. Jednou z možných aplikací je zjednodušování polygonálních sítí se zachováním charakteristických rysů tvaru objektu. [1]

[1] RBF - Radial Basis Function

Dalším možným využitím algoritmů pro výpočet křivosti je odstraňování šumu z medicínských snímků, kdy při pořizování snímků jsou v řadě postupů aplikovaných v medicíně získaná data zatížena šumem do takové míry, že v nich není možné rozpoznat objekty nutné pro správné určení diagnózy. [8]

V neposlední řadě jsou hodnoty křivosti využívány k rozpoznání elementárních tvarů ve složitějším objektu. Jednou z konkrétních aplikací využívající postupu rozpoznávání objektů založenou na určování křivosti může být rozpoznávání lidských tváří, které je dne používáno především v oblasti zabezpečení. [19][16][24]

# 4 Strojové učení

Strojové učení je jednou z disciplín umělé inteligence, která se zabývá technikami a algoritmy umožnujícími systému přizpůsobovat se okolí nebo situaci na základě zkušeností. [3]

## 4.1 Metody učení

Metody strojového učení můžeme rozdělit na základě informací, které jsou systému během samotného procesu učení poskytovány. [3]

- Učení s učitelem - příklady, ze kterých se systém učí, jsou jasně zařazeny do tříd.
- Částečné učení s učitelem - malá část příkladů, které jsou systému poskytnuty, je zařazena do tříd a zbytek množiny příkladů je neklasifikován.
- Aktivní učení - systém se aktivně dotazuje učitele na zařazení příkladů do tříd.
- Učení nepřímými náznaky - systém odvozuje zařazení příkladů nepřímo dle změn chování učitele.
- Žádné - učení bez učitele.

Jelikož pro konstrukci klasifikátoru je dostupná množina trénovacích dat s jasnými popisy vlastností klasifikovaných objektů i jejich zařazením do tříd, jde o metodu učení s učitelem.

## 4.2 Volba reprezentace

Při řešení úloh strojového učení je důležité, jakým způsobem jsou data reprezentována. Dle základního členění můžeme data rozdělit na data reprezentovaná atributy a data, která jsou reprezentována relací.

Data reprezentovaná atributy jsou popsána pomocí binárních, nominálních či ordinálních hodnot. Při reprezentaci dat relací máme k dispozicí popis vztahů mezi jednotlivými prvky množiny. [3]

## 4.3 Základní druhy úloh

Problémy řešení v oblasti strojového učení můžeme rozdělit také dle typu úlohy. Základními typy úloh jsou klasifikace, regrese a shlukování.

Klasifikace je druh problému, kdy je našim cílem zařazení vzorku do jedné či více předem známých kategorií na základě jeho popisu. Regrese je druh problému, se snažíme určit hodnotu jedné či více vlastností prvku na základě popisu, který nám je znám. Shlukováním rozumíme problém zařazování objektů do tříd na základě jejich podobnosti. Na rozdíl od klasifikace v případě shlukování vzorků třídy obvykle dopředu neznáme a k jejích utvoření dojde až po provedení procesu shlukování. [3]

# 5 Popis úlohy

Tato práce se nezabývá konkrétními postupy pro odhad křivostí nad polygonálními sítěmi, nýbrž konstrukcí klasifikačního algoritmu pro nalezení optimálního přístupu odhadu křivosti. Při klasifikaci sítě vycházíme z parametrizace tvořené číselným popisem jejich vlastností.

Cílem úlohy je konstrukce mechanismu, který bude schopný pro zadanou trojúhelníkovou síť na základě jejích vlastností, jako jsou hustota vrcholů, pravidelnost trojúhelníků, kterými je síť tvořena, apod., predikovat, který estimátor bude optimální využít pro odhad křivosti.

Předpokladem je rozdělení konstrukce mechanismu do dvou etap. V první etapě probíhá tzv. učení s učitelem, během něhož je konstruován konkrétní model mechanismu na základě množiny trénovacích dat. Množina trénovacích dat je tvořena záznamy, které obsahují vlastnosti trojúhelníkové sítě a chybou odhadu křivosti při využití jednotlivých estimátorů.

Data, se kterými implementované algoritmy pracují obsahují pro každou polygonální síť množinu 64 vlastností, které ji popisují a na jejichž základě je prováděna klasifikace do 32 tříd, z nichž každá třída obsahuje sítě, které mají shodnou optimální metodu odhadu křivosti.

Matematicky bychom mohli takový klasifikátor popsat jako zobrazení $X : \mathbb{R}^s \rightarrow n \in \{0, 1, ..., e - 1\}$, kde $s$ je počet pozorovaných vlastností polygonálních sítí a $e$ je počet estimátorů.

# 6 Předzpracování dat

Při zpracovávání dat metodami strojového učení často narazíme na problém, kdy vstupní data nejsou ve formátu, který by byl vhodný pro algoritmy strojového učení a je tedy nutné data nejprve upravit. Postupů pro předzpracování dat je celá řada a vhodná volba konkrétního algoritmu závisí především na způsobu, jakým byla data pořízena, na jejich vzájemné korelaci, na velikosti datové množiny a na algoritmu, pro který potřebujeme data připravit. Tyto metody můžeme rozdělit do čtyřech základních skupin podle toho, k jakému typu přípravy dat jsou metody určeny. [18]

Ve spoustě aplikací mohou být pořízená data ve formátech jako je text, obraz, zvuk či video. Takto pořízená data jsou často dále nezpracovatelná a je nutné je převést do vektorové reprezentace. V našem případě jsou však pořízená data již reprezentována vektory, proto není nutné se metodami vektorizace dále zabývat.

Normalizací dat obecně rozumíme proces, při kterém dochází k úpravě dat takovým způsobem, aby bylo možně je reprezentovat předem definovanými strukturami, popř. relacemi. Z pohledu statistiky například pomocí normalizace docílíme změny měřítka pořízených dat či odstranění závislosti na použitých jednotkách.

Dalším problémem, se kterým se při práci s reálnými daty často potýkáme, mohou být chybějící hodnoty v naměřené množině. Takovýto problém se obvykle snažíme řešit vyplněním chybějící části dat hodnotou, která nejméně ovlivní výsledky následujícího zpracování. Pořízená data, se kterými při řešení problému výběru optimálního estimátoru pracujeme, jsou kompletní a u všech vzorků jsou sledovány stejné vlastnosti. Z tohoto důvodu není použití metod pro doplnění chybějících hodnot v našem případě nutné.

Poslední skupinou metod pro přípravu dat jsou metody zabývající se možnostmi extrakce vybrané podmnožiny sledovaných vlastností pořízených vzorků oproti zbylé části množiny. Tyto metody se snaží o potlačení vlastností, které nejsou z hlediska dalšího zpracování relevantní a o zesílení vlastností, které relevantní jsou.

## 6.1 Normalizace vlastností

### 6.1.1 Normalizace rozpětím

Hodnoty sledovaných vlastností polygonálních sítí nabývají řádově odlišných rozměrů, což může vést k nepřesnosti při klasifikaci, popř. až ke kompletnímu selhání klasifikačního algoritmu. Například při využití metody shlukování se dá očekávat, že rozdíl mezi rozsahy vlastností může způsobit deformaci prostoru, což může vést ke potlačení vlivu některých vlastností při výpočtu vzdálenosti bodu od těžiště.

Normalizací rozpětím jsou přepočteny vlastnosti všech sítí v množině dat do jednotkového intervalu pomocí následujícího vzorce: (Vzorec 6.1).

$$y_{ij} = \frac{x_{ij} - min(x_j)}{max(x_j) - min(x_j)} \tag{6.1}$$

### 6.1.2 Normalizace pomocí směrodatné odchylky

Normalizace pomocí směrodatné odchylky je metoda, kdy je nová hodnota vypočtena odečtením výběrového průměru od původní hodnoty a následným vydělením směrodatnou odchylkou. Směrodatná odchylka je je vypočtena jako druhá odmocnina aritmetického průměru čtverců rozdílu výběrového průměru a původních hodnot.

Výhodou normalizace pomocí rozptylu oproti normalizaci rozpětím je, že transformace hodnot vlastností je méně náchylná na odlehlé hodnoty.

Vzorec výpočtu nových hodnot je uveden níže: (Vzorec 6.2)

$$y_{ij} = \frac{x_{ij} - \bar{x_j}}{\sqrt{\frac{1}{N-1}\sum (x_{ij} - \bar{x_j})^2}} \tag{6.2}$$

## 6.2 Analýza hlavních komponent

Počet sledovaných vlastností u polygonálních sítí je značný a pro interpretaci nepřehledný, a např. pro konstrukci hlubšího rozhodovacího stromu metodou slepého prohledávání všech dostupných možností příliš vysoký. Pro zjednodušení konstrukce klasifikátoru lze pomocí analýzy hlavních komponent zkoumat, zda je možné zaznamenané vlastnosti pozorovaných sítí nahradit menším počtem jiných vlastností s co nejmenší ztrátou informace.

Uvažujeme-li popis polygonálních sítí pomocí vektorů v lineárním vektorovém prostoru dimenze 64, pomocí analýzy hlavních komponent hledáme bázi v lineárním podprostoru s nižším počtem dimenzí, kterou lze vyjádřit

jako lineární kombinaci prvků báze původního lineárního prostoru a která nejlépe popisuje vlastnosti pozorované množiny polygonálních sítí.

Sestavíme matici $\boldsymbol{A}$ použitím vektorů s vlastnostmi všech polygonálních sítí jako sloupců matice. V každém řádku matice budou tedy umístěny hodnoty právě jedné vlastnosti. Provedeme vycentrování matice $\boldsymbol{A}$ odečtením řádkového průměru od každého řádku matice, čímž získáme matici $\boldsymbol{A}_C$. Kovarianční matici $\boldsymbol{C}$ získáme jako součin $\boldsymbol{C} = \boldsymbol{A}_C \boldsymbol{A}_C^T$.

Dále určíme vlastní čísla a vlastní matice $\boldsymbol{C}$, přičemž, vlastní vektory můžeme interpretovat jako novou bázi lineárního vektorového prostoru tvořeného vlastnostmi polygonálních sítí a k nim odpovídající vlastní čísla použít jako metriku relevance příslušné vlastnosti. Nyní jsme tedy schopni vybrat pouze omezený počet nejrelevantnějších vlastností, které použijeme pro klasifikaci.

# 7 Metody klasifikace

## 7.1 Rozhodovací strom

Rozhodovací strom je struktura reprezentovaná pomocí orientovaného grafu $G = (V, E), E \subset V^2$, s konečnou množinou vrcholů $V$ rozdělenou na tři disjunktní množiny $V = \mathscr{D} \cup \mathscr{C} \cup \mathscr{T}$ rozhodovacích, pravděpodobnostních a koncových uzlů.

Pro každou hranu $e \in E$ vedoucí z vrcholu $e_1$ do vrcholu $e_2$ označíme vrchol $e_1$ jako rodičovský uzel a $e_2$ jako potomka. [13] Dále jsou pro klasifikaci použity rozhodovací stromy, které neobsahují žádné pravděpodobnostní uzly ($\mathscr{C} = \emptyset$).

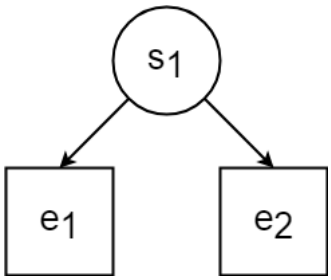
Rozhodovací uzly jsou takové uzly, které obsahují funkci $d(x) : \mathbb{R}^s \to C$, kde $s$ je dimenze vektoru, kterým jsou parametrizována vstupní data a $C$ je množina hran vedoucích z rozhodovacího uzlu. V rozhodovacím uzlu je tedy na základě vlastností klasifikovaného objektu vybrána jedna z hran, která je přidána do hledané cesty. V případě binárních rozhodovacích stromů je rozhodováno mezi průchodem levou a pravou hranou. Koncové uzly tvoří listy grafu rozhodovacímu stromu a přiřazují klasifikovanému objektu právě jednu konkrétní třídu.

Proces klasifikace je realizován jako hledání cesty z vrcholu $r$ do vrcholu $v$, kde $r$ je kořenem grafu a $v$ je libovolný koncový uzel, takové, že při průchodu všemi rozhodovacími uzly, kterými cesta prochází jsou splněny podmínky pro průchod grafem po zvolené cestě.

### 7.1.1 Jednovrstvý rozhodovací strom

Pro ověření správnosti výsledků experimentu byl znovu implementován klasifikátor realizovaný jednovrstvým úplným binárním rozhodovacím stromem, tj. strom s jedním uzlem, pro rozhodnutí na základě jedné vlastnosti polygonální $s_1$ sítě mezi dvěma estimátory $e_1$ a $e_2$ (Obrázek 7.1), popsaný v článku Mesh Statistics for Robust Curvature Estimation. [20] Jelikož oba experimenty probíhaly nad velmi podobnou množinou dat, je předpokladem správnosti výsledků malá odchylka mezi hodnotami metrik určenými pro vyhodnocení výsledků klasifikace.

Hledání optimální kombinace vlastnosti $s_1$, jejího prahu $t_1$ a estimátorů $e_1$ a $e_2$ bylo provedeno vyzkoušením všech možností kombinace jedné vlastnosti, jejích hodnot v množině trénovacích dat a dvojice estimátorů a zvolením




Obrázek 7.1: Jednovrstvý binární rozhodovací strom.

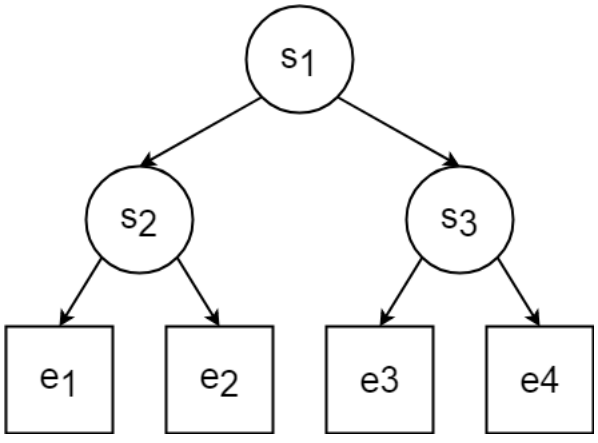
takové možnosti, jejíž relativní chyba (viz 8.1) byla minimální.

Při volbě konkrétní vlastnosti a dvojice estimátorů lze hodnotu prahu vlastnosti snadno určit prohledáním všech množností tvořených množinou hodnot dané vlastnosti, které se vyskytují v množině vstupních dat.

### 7.1.2 Dvouvrstvý rozhodovací strom

Předpokládejme, že zvětšením počtu vlastností, na základě kterých v rozhodovacím stromu klasifikujeme mezi jednotlivými estimátory a počtu estimátorů přidáním vrstvy zvětšíme přesnost klasifikátoru. Konstrukce úplného dvouvrstvého binárního rozhodovacího stromu hledáním nejlepší kombinace vlastností, jejich prahů a estimátorů, ze všech možností je však již výpočetně příliš složité. Je tedy nutné zredukovat počet možných řešení.

Použitím analýzy hlavních komponent pro redukci počtu vlastností polygonálních sítí můžeme zmenšit velikost prostoru řešení, díky čemuž jsme schopni nalézt optimální konfiguraci rozhodovacího stromu se dvěma vrstvami v kratším čase.




Obrázek 7.2: Dvouvrstvý binární rozhodovací strom.

### 7.1.3 Použití genetických algoritmů pro konstrukci víceúrovňového rozhodovacího stromu

Pro vytvoření víceúrovňového rozhodovacího stromu metodou prohledání všech možných řešení není redukce vlastností pomocí analýzy hlavních komponent vzhledem k velké výpočetní složitosti dostatečná.

Pro nalezení optimálních vlastností, prahů vlastností a estimátorů je však možné využít iterativní proces genetického algoritmu vytvořením náhodné populace vektorů reprezentujících konfiguraci rozhodovacího stromu, jejich následným křížením, mutací a výběrem nové generace prvků s nejpříznivějším ohodnocením.

Při počtu vrstev $n$ obsahující uzly, ve kterých dochází k rozhodování na základě vlastnosti polygonální sítě a jedné vrstvě obsahující listy stromu reprezentující výslednou třídu je populace inicializována jako množina vektorů s $2^n - 1$ složkami $s_i$ obsahující náhodný index jedné z pozorovaných vlastností $2^n - 1$ složkami $t(s_i)$ obsahující práh dané vlastnosti a $2^n$ složkami obsahující $e_i$ index estimátoru.

Následně je po předem stanovený počet iterací opakován proces křížení, při kterém je pro každé dva prvky populace náhodně určeno, zda a případně v kterém místě dojde ke křížení vektorů, které jsou poté přidány do populace. Dále je náhodné množství prvků zmutováno změnou náhodné složky vektoru na jinou dostupnou hodnotu, čímž opět dojde ke zvětšení prvků v populaci. Ppoté je každému prvku v populaci přiřazena pomocí ohodnocovací funkce číselná hodnota, podle které jsou vektory seřazeny a do nové populace pro následující iteraci se vybere množina prvků s nejlepším ohodnocením o velikosti původní populace.

Jako ohodnocovací funkce je vypočítána relativní chyba estimátoru (viz 8.1) tvořeným rozhodovacím stromem, který příslušný vektor reprezentuje.

### 7.1.4 Simulované žíhání

Druhou metodou, použitou pro nalezení optimálního nastavení rozhodovacího stromu větší hloubky, bez prohledávání všech možností, je simulované žíhání (angl. Simulated annealing).

Algoritmus simulovaného žíhání vychází z fyziky. Termínem žíhání označujeme fyzikální proces, při kterém je ohřáté těleso postupně ochlazováno. Důležitým faktorem žíhání je, že k chladnutí ohřátého tělesa dochází dostatečně pomalu na to, aby částicím uvnitř tělesa bylo umožněno přeuspořádání do takové struktury, která poskytne lepší fyzikální vlastnosti. Při vysoké teplotě tělesa je velmi vysoká pravděpodobnost zániku defektů jeho krystalické

mřížky, naopak se snižující se teplotou tělesa se snižuje pravděpodobnost vzniku nových defektů. Výhodou simulovaného žíhání je schopnost algoritmu překonat lokální extrémy ohodnocovací funkce a pokračovat v hledání v sousedních řešeních, i za předpokladu zhoršení aktuálního řešení. Na počátku algoritmu je vygenerováno počáteční řešení, v našem případě počáteční vektor reprezentující konfiguraci rozhodovacího stromu a nastavena počáteční teplota systému.

Následně jsou v iterativním procesu hledána sousední řešení provedením mutace aktuálního vektoru. Pro nalezená řešení je vypočteno ohodnocení, které je stejně jako při použití základního genetického algoritmu dáno úspěšností klasifikátoru, v případě, že by nalezené řešení bylo použito pro klasifikaci trénovací množiny dat.

Je-li ohodnocení nově nalezeného řešení lepší, než ohodnocené aktuálního řešení, aktuální řešení je jím nahrazeno, bez ohledu na aktuální teplotu systému. Je-li však ohodnocení nově nalezeného řešení horší, dojde k nahrazení aktuálního řešení pouze s určitou pravděpodobností, která je přímo úměrná aktuální teplotě systému. Teplota systému je v našem případě dána diskrétní klesající funkcí času, což způsobuje postupné ochlazování systému a tedy i postupné snižování pravděpodobnosti přijetí horšího řešení. [12]

## 7.2 Neuronová síť

Neuronová síť je způsob reprezentace algoritmů vycházející z biologické struktury mozku. Jako základní stavební prvky neuronových sítí jsou použity neurony, které pracují jako elementární výpočetní jednotky a synapse, které spolu neurony propojují a umožňují tak přenos informace mezi nimi. Neurony jsou v neuronových sítích umisťovány do vrstev, které můžeme dělit na tři druhy.

1. Vstupní vrstva - vrstva, který přijímá vstupní data
2. Skrytá vrstvy - vrstva, nebo více vrstev, které nejsou přístupné z venku
3. Výstupní vrstva - vrstva pro předání výsledných hodnot pro vstupy zpracované neuronovou sítí ven

Neuronové sítě jsou ideálními nástroji v případech, kdy pro složitost problému nejsme schopni určit jeho analytické řešení a jeho zjednodušení na úroveň, na které bychom jej byly schopni určit, by vedlo k příliš velké ztrátě informace. [23]

Potýkáme-li se se složitými problémy, je často užitečné pro vyjádření vztahu mezi vstupními daty a výsledky použití regresní analýzy, obvykle lineární. Výsledkem je rovnice, ve které jsou pro zjištění výsledku vstupy $x_j$ vynásobeny vahou $w_j$ a sečteny spolu s konstantou $\theta$. (Vzorec 7.1)[4]

$$y = \sum_j w_j x_j + \theta \tag{7.1}$$

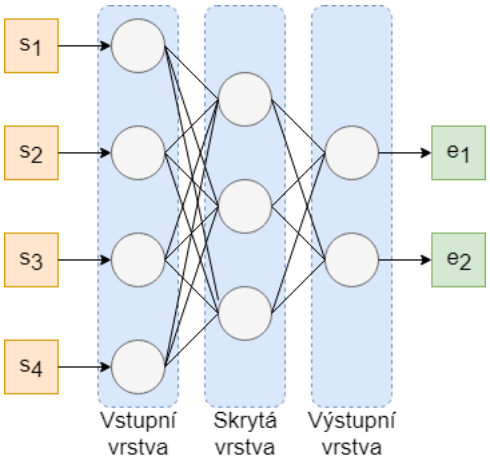
S použitím neuronových sítí jsou vstupní data $x_j$ stejným způsobem násobena vahami $w_j$, avšak tento součet je dosazen jako argument nelineární funkce (např. hyperbolické tangenty). Neuronová síť je poté dána explicitní kombinací přechodových funkcí a jednotlivými vahami. Počet aplikovaných funkcí odpovídá počtu skrytých vrstev sítě. Výslednou funkci popisující neuronovou síť tedy můžeme zapsat pomocí následujícího vzorce, (Vzorec 7.2)[4]

$$y = \sum_i w_i h_i + \theta \tag{7.2}$$

kde $h_i$ jsou výstupní hodnoty neuronů v předchozí vrstvě přivedené na vstup neuronů v další vrstvě (Vzorec 7.3).

$$h_i = tanh(\sum_j w_{ij} x_j + \theta_i) \tag{7.3}$$

Neuronové sítě můžeme rozdělit podle jejich topologie na sítě dopředné, v nichž se signál šíří výhradně jedním směrem a zpětnovazební sítě, kde dochází k šíření signálu všemi směry.




Obrázek 7.3: Ilustrační příklad třívrstvého dopředné neuronové sítě pro 4 vlastnosti a dva estimátory.

Jako aktivační funkce je v našem případě, namísto hyperbolické tangenty uvedené v příkladu, použita lineární rektifikované jednotková funkce (ReLU), pro vstupní a vnitřní vrstvy a Sigmoida pro výstupní vrstvu.

Funkce ReLU je definována jako kladná část svého argumentu (Vzorec 7.4).

$$f(x) = max(0, x) \tag{7.4}$$

Sigmoida je konkrétním případem logistické funkce a je definována jako: (Vzorec 7.5)

$$f(x) = \frac{1}{1 + e^{-x}} \tag{7.5}$$

Pro optimalizaci učení neuronové sítě je použit algoritmus gradientního sestupu SGD (angl. Stochastic Gradient Descent). SGD je populární optimalizační iterativní algoritmus v oblasti strojového učení, kde platí: (Vzorec 7.6)

$$x_{t+1} := x_t - \eta_t g_t \tag{7.6}$$

Kde $x_t$ je stav v aktuální iteraci, $\eta_t$ rychlost učení neuronové sítě a $g_t$ je gradient sestupu.

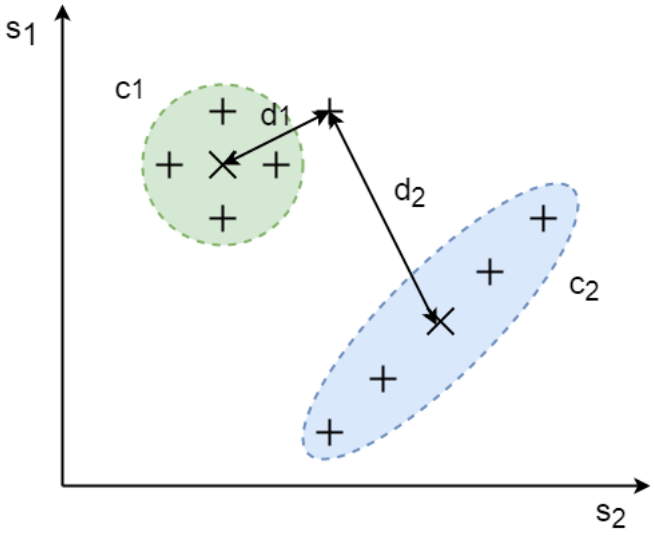
Cílem této optimalizace je zabránit neuronové síti v uvíznutí v lokálním minimu ohodnocovací funkce přičtením gradientu a pokračováním v hledání řešení z nového výchozího bodu. [17]

## 7.3 Shlukování

Mohli bychom předpokládat, že sítě s podobnými vlastnostmi, tedy body v lineárním vektorovém prostoru, které jsou si navzájem blízké, můžeme klasifikovat do tříd reprezentujících tentýž optimální estimátor. Uvažujeme-li popis polygonálních sítí pomocí vektoru vlastností jako bod v 34-dimenzionálním lineárním vektorovém prostoru můžeme se pokusit tyto body v prostoru shlukovat.

Pro každý estimátor byl tedy v rámci učení klasifikátoru vytvořen shluk bodů, reprezentujících sítě, pro něž je použití daného estimátoru optimální. V každém z těchto shluků bylo vypočteno těžiště jako vektor aritmetických průměrů jednotlivých vlastností, které tento shluk charakterizuje.

Při klasifikaci sítě z kontrolní množiny dat, je nalezen shluk, jehož těžiště leží nejblíže bodu, kterým síť reprezentujeme, ve smyslu zvolené vzdálenosti




Obrázek 7.4: Ilustrační obrázek shlukování ve 2D prostoru.

a estimátor příslušného shluku je zvolen jako optimální estimátor kontrolní sítě. (Obrázek 7.4)

Pro porovnávání vzdálenosti klasifikovaných bodů od těžišť jednotlivých shluků byly vyzkoušeny tři metriky - Euklidovská vzdálenost (Vzorec 7.7), Cosinová vzdálenost (Vzorec 7.8) a Manhattanská vzdálenost (Vzorec 7.9).

$$d_e = \sqrt{\sum_{i=0}^{n}(a_i - b_i)^2} \tag{7.7}$$

$$d_c = \arccos\left(\frac{\sum_{i=0}^{n}(a_i * b_i)}{\sum_{i=0}^{n} a_i^2 * \sum_{i=0}^{n} b_i^2}\right) \tag{7.8}$$

$$d_m = \sqrt{\sum_{i=0}^{n} \mid a_i - b_i \mid} \tag{7.9}$$

## 7.4 Aproximace metodou nejmenších čtverců

Kromě matematického popisu v kapitole 5, kde klasifikátor popisujeme jako zobrazení přiřazující vektoru vlastností polygonální sítě index estimátoru, bychom mohli hledat zobrazení přiřazující vektoru vlastností odhadovanou chybu jednotlivých estimátorů a následně hledat index optimálního estimátoru jako argument minima odhadované chyby. Hledáme tedy zobrazení $Y : \mathbb{R}^s \to \mathbb{R}^e$. Takové zobrazení nejsme schopni pro obecná vstupní data explicitně popsat a ani nevíme, zda takové zobrazení skutečně existuje. Mů-

žeme se však pokusit hledané zobrazení aproximovat metodou nejmenších čtverců.

Pomocí metody nejmenších čtverců jsme schopni aproximovat $f(x)$, jejíž předpis nám není znám, ale máme naměřené dvojice hodnot $[x_i, y_i]$, kde $x_i$ jsou argumenty funkce $f(x)$ a $y_i$ hodnoty funkce, které mohou být zpravidla zatíženy chybou měření, přímkou, polynomy vyššího stupně, popřípadě i jinou funkcí $g(x)$. Při řešení této metody hledáme funkci $g(x)$ takovou, aby její součet kvadratických odchylek $\rho^2$ v naměřených bodech od původní funkce byl minimální.

Jelikož body $[x_i, y_i]$ jsou dány měřením, závisí součet odchylek při volbě konkrétní báze prostoru, ve kterém funkci hledáme, pouze na koeficientech. V případě aproximace lineární funkcí $g(x) = c_0 + c_1 x$, tedy hledáme koeficienty $c_0$ a $c_1$. Minimalizujeme tedy funkci (viz Vzorec 7.10)[7]

$$\rho(x_0, x_1) = \sum_{i=0}^{n} (y_i - c_0 - c_1 x_i)^2 \tag{7.10}$$

Z diferenciálního počtu funkcí více proměnných je známo, že nutnou podmínkou pro to, aby tato funkce nabývala minima, je, že splnění nulové hodnoty parciálních derivací funkce podle obou proměnných. (viz Vzorec 7.11)[7]

$$\frac{\partial(\rho^2)}{\partial(c_0)} = 0, \frac{\partial(\rho^2)}{\partial(c_1)} = 0 \tag{7.11}$$

Aplikujeme-li tento postup na zobrazení $Y$ přiřazující vektoru vlastností polygonálních sítí vektor s odhady chyb jednotlivých estimátorů pro získání aproximace $Y'$ tohoto zobrazení získáme předpis. (viz Vzorec 7.12)

$$Y(x) \approx Y'(x) = C_0 + C_1 x \tag{7.12}$$

Doposud jsme předpokládali, že aproximujeme reálnou funkci jedné proměnné $f : \mathbb{R} \to \mathbb{R}$. Nyní je však třeba odvodit podobný vztah pro zobrazení $Y : \mathbb{R}^s \to \mathbb{R}^e$, které vektoru vlastností $x$ přiřadí vektor s odhady chyb estimátorů $y$. Po nahrazení lineární funkce jedné proměnné polynomem prvního stupně s počtem proměnných rovným počtu sledovaných vlastností $s$ a za předpokladu, že chyby jednotlivých estimátorů jsou navzájem nezávislé, můžeme každou složku vektoru $y$ s odhady chyb vyjádřit jako lineární kombinaci složek vektoru vlastností. (Vzorec 7.13)

$$Y(x) \approx Y'(x) = \begin{pmatrix} c_{00} + c_{01}x_1 + c_{02}x_2 + \cdots + c_{0s}x_s \\ c_{10} + c_{11}x_1 + c_{12}x_2 + \cdots + c_{1s}x_s \\ \vdots \\ c_{e0} + c_{e1}x_1 + c_{e2}x_2 + \cdots + c_{es}x_s \end{pmatrix} \tag{7.13}$$

Po určení koeficientů $c_{ij}$ pro všechna $i \in \{0, 1, ..., s\}$, kde $s$ je počet vlastností a všechna $j \in \{0, 1, ..., e-1\}$, kde $e$ je počet estimátorů jsme schopni určit odhad chyby pro každý estimátor v závislosti na vlastnostech polygonální sítě a optimální estimátor $e_{opt}$ hledat jako argument minima těchto hodnot. (Vzorec 7.14)

$$e_{opt} = \arg\min_{i} \begin{pmatrix} c_{00} + c_{01}x_1 + c_{02}x_2 + \cdots + c_{0s}x_s \\ c_{10} + c_{11}x_1 + c_{12}x_2 + \cdots + c_{1s}x_s \\ \vdots \\ c_{e0} + c_{e1}x_1 + c_{e2}x_2 + \cdots + c_{es}x_s \end{pmatrix} \tag{7.14}$$

# 8 Metody vyhodnocení

Pro porovnání odlišných metod pro klasifikaci byly použity dvě metriky, přesnost a relativní chyba. Přesnost nám umožňuje vytvořit si představu o tom, v kolika případech bylo výsledkem klasifikátoru přiřazení trojúhelníkové sítě k optimálnímu estimátoru. Relativní chyba zohledňuje i odchylku odhadnuté křivosti od reálné křivosti v případě, kdy je síť přiřazena jinému než optimálnímu estimátoru.

Z tohoto hlediska je možné, že klasifikátor, kterému se nepodaří v žádném případě určit optimální estimátor, avšak jím zvolený estimátor bude mít i přes to velkou úspěšnost, bude mít nižší relativní chybu, než klasifikátor, který pro část množiny vstupních dat určil optimální estimátor správně, ale pro zbytek množiny jsou chyby jím zvoleného estimátoru příliš vysoké. Z tohoto důvodu nám relativní chyba jako metrika v tomto případě poskytuje mnohem relevantnější informace.

## 8.1 Relativní chyba

Pro porovnání úspěšnosti výsledků jednotlivých algoritmů je použita metrika relativní chyby $d_r$, která je vypočtena jako poměr rozdílu mezi nejmenší dosaženou chybou pro danou polygonální síť (chybou optimálního estimátoru) $d_{min}$ a chybou zvoleného estimátoru $d_s$ a chyby optimálního estimátoru. (Vzorec 8.1)

$$d_r = \frac{d_s - d_{min}}{d_m in} \tag{8.1}$$

Určení relativní chyby klasifikátoru je klíčové pro porovnání s dříve dosaženými výsledky i jako ukazatel úspěšnosti klasifikace vzhledem k povaze úlohy.

## 8.2 Přesnost

Další metrikou použitou pro vyhodnocení úspěšnosti klasifikace je přesnost (Precision). Přesnost klasifikace vypočteme jako poměr součtu počtu správně klasifikovaných vzorků (TP) a celkového počtu všech vzorků.

- True positive (TP) - vzorek správně přiřazen do třídy
- True negative (TN) - vorek správně nepřiřazen do třídy

- False positivy (FP) - vzorek chybně přiřazen do třídy
- False negative (FN) - vzorek chybně nepřiřazen do třídy

Přesnost můžeme vypočítat podle následujícího vzorce. (Vzorec 8.2)

$$PRE = \frac{TP}{TP + TN + FP + FN} = \frac{TP}{ALL} \tag{8.2}$$

Ačkoli tato metrika není tak relevantním ukazatelem úspěšnosti jako výpočet relativní chyby, může nám posloužit jako ukazatel důležitosti volby optimálního estimátoru pro minimalizaci relativní chyby.

# 9 Implementace

Všechny výše uvedené metody klasifikace sítí byly implementovány v prostředí .NET Core verze 2.2 v programovacím jazyce C# a prostředí interpretovaného jazyka Python verze 3.7, který byl v rámci jednotného uživatelského rozhraní integrován do aplikace implementované v prostředí .NET Core pomocí automatizovaného spouštění Python interpretu a následné komunikace s ním.

Pro konstrukci klasifikátorů založených na neuronové síti byly použity knihovny Keras a TensorFlow, které jsou dnes jedněmi ze standardních nástrojů v oblasti strojového učení pro programovací jazyk Pyhon.

## 9.1 Konfigurace výběru algoritmů

Výběr použitých algoritmů a jejich konkrétních parametrů při zpracování dat, trénování a následném vyhodnocení je řízen pomocí serializované třídy Configuration do formátu Json. Tato třída obsahuje pole objektů implementujících rozhraní IProcessor, v němž jsou umístěny instance tříd realizující logiku pro předzpracování dat, instanci třídy implementující rozhraní IMetaEstomator pro samotnou klasifikaci a pole objektů implementujících rozhraní IEvaluator pro vyhodnocení výsledků. (Obrázek 9.1)

Typy algoritmů jsou v serializované konfiguraci reprezentovány pomocí objektů s vlastnostmi Type a Instance. Vlastnost Type určuje třídu jazyka C#, která implementuje požadovaný algoritmus a vlastnost Instance obsahuje konkrétní vlastnosti objektu, resp. jeho počáteční nastavení, jako například. hloubku rozhodovacího stromu, počet vrstev neuronové sítě, způsob výpočtu vzájemné vzdálenosti vektorů při shlukování, apod. (viz 9.1)

Pro spuštění služby dávkového zpracování pak konfigurace obsahuje ještě cestu k souboru se vstupními daty.

```
{
  "InputFilePath":"C:\\data.txt",
  "Processors":[
    {
      "Type":"MetaEstimator.Core.Processor.
         DispersionNormalization.
         DispersionNormalizationProcessor",
      "Instance":{}
    },
    {
      "Type":"MetaEstimator.Core.Processor.
         PrincipalComponentAnalysis.
         PrincipalComponentAnalysisProcessor",
      "Instance":{
        "RelevantStatsCount":5
      }
    }
  ],
  "MetaEstimator":{
    "Type":"MetaEstimator.Core.MetaEstimator.
       ClusterMetaEstimator",
    "Instance":{}
  "Evaluators":[
    {
      "Type":"MetaEstimator.Core.Evaluator.
         RelativeDivergenceEvaluator",
      "Instance":{}
    },
  ],
}
```

Listing 9.1: Ukázka serializované konfigurace použitých algoritmů

## 9.2 Vyhodnocení vybraných algoritmů

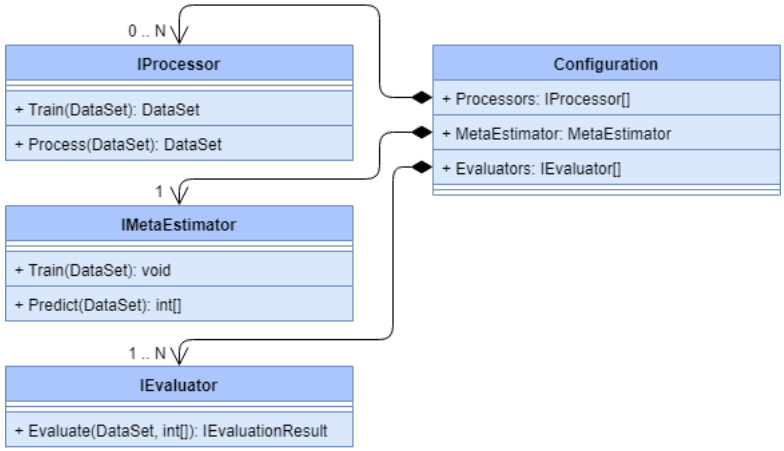
Při vyhodnocování úspěšnosti klasifikátoru s vybranými metodami pro předzpracování dat, klasifikaci a vyhodnocení výsledků program načte vstupní data popisující vlastnosti polygonálních sítí a chyby jednotlivých estimátorů ze souboru. Následně rozdělí množinu načtených dat na dvě části, z nichž jedna

je dále použita pro učení klasifikátoru a druhá pro určení úspěšnosti použitých algoritmů. Dělení dat do trénovací a ověřovací množiny probíhá rozdělením jednotlivých záznamů na liché a sudé, podle jejich pořadí v textovém souboru.

Po načtení a rozdělení vstupních dat jsou na trénovací množinu postupně aplikovány všechny algoritmy předzpracování dat uvedené v konfiguraci, během čehož jsou vypočítány potřebné statistiky pro následné předzpracování ověřovací množiny.

Předzpracovaná trénovací množina dat je přivedena na vstup klasifikátoru, který při zpracování množiny sestaví model umožňující následnou predikci optimálního estimátoru pro sítě z ověřovací množiny.

Dále jsou aplikovány algoritmy předzpracování na ověřovací množinu dat, která je následně klasifikována za pomoci naučeného klasifikačního algoritmu do disjunktních tříd jednotlivých estimátorů. Poté algoritmy pro vyhodnocení úspěšnosti porovnají indexy predikovaných optimálních estimátorů a skutečně optimálních estimátorů.




Obrázek 9.1: UML diagram popisující strukturu konfigurace výběru algoritmů.

## 9.3 Služba pro vyhodnocení úspěšnosti metod klasifikace

Pro snadnější testování různých implementací klasifikátoru a algoritmů pro předzpracování dat byla implementována služba umožňující dávkové zpracování.

Serializované konfigurace obsahující popis jednotlivých modulů jsou umístěny do tabulky relační databáze SQL Server. Služba po spuštění provede dotaz do databáze, kterým zjistí, zda obsahuje alespoň jednu nezpracovanou konfiguraci a začne ji v takovém případě zpracovávat. Po vyhodnocení úspěšnosti konfigurace zapíše její výsledek do příslušného řádku v tabulce a pokračuje zpracováním dalšího záznamu.

Konfigurace jsou v databázi uloženy v tabulce CONFIGURATION_RECORD, která sebou nese kromě informací o zvolených algoritmech také výsledky vyhodnocení, dobu běhu aplikace, a záznam o případných chybách. Názvy a datové typy jednotlivých sloupců popisuje následující obrázek. (Obrázek 9.2)

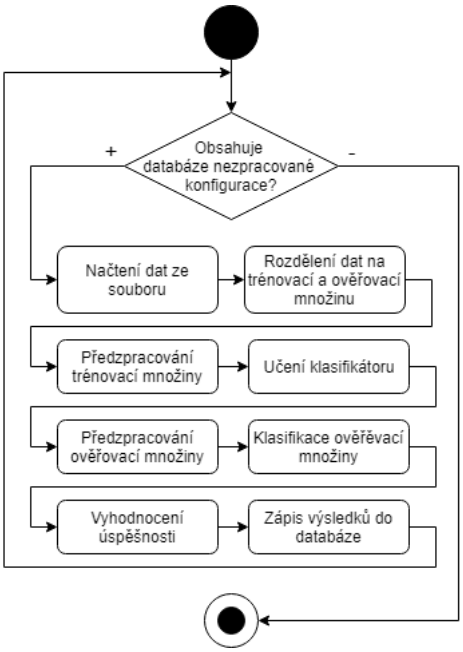
| CONFIGURATION_RECORD | | |
|---|---|---|
| PK | ID | bigint, not null |
| | NAME | varchar(250), not null |
| | CONFIGURATION | varchar(max), not null |
| | RESULT | varchar(max) |
| | STARTED | datetime |
| | FINISHED | datetime |
| | SUCCESS | bit |
| | ERROR_MESSAGE | varchar(max) |

Obrázek 9.2: Sloupce databázové tabulky CONFIGURATION_RECORD.

Jednotlivé sloupce jsou poté plněny následujícím způsobem:

- ID - Identifikátor záznamu, který je automaticky vygenerován při ukládání konfigurace do databáze.
- NAME - Automaticky generovaný název jako spojení názvů jednotlivých algoritmů pro snadnější orientaci a vyhledávání v databázi.
- CONFIGURATION - Serializovaná konfigurace s typy a počátečním nastavením jednotlivých algoritmů.
- RESULT - Serializovaný výsledek vyhodnocení.
- STARTED - Datum a čas spuštění testu konfigurace.
- FINISHED - Datum a čas dokončení testu konfigurace.
- SUCCESS - Binární hodnota příznaku, zda byl průběh testu úspěšný.
- ERROR_MESSAGE - Výpis případných chyb v průběhu testování.

Činnost služby pro dávkové zpracování znázorňuje Obrázek 9.3




Obrázek 9.3: Diagram aktivit služby pro dávkové zpracování.

## 9.4 Parsování vstupních dat

Data, která byla pro řešení úlohy poskytnuta, jsou strukturována ve dvou souborech ve formátu, který kromě sledovaných vlastností polygonálních sítí a chyb při odhadech křivost jednotlivými estimátory obsahuje další informace, které nejsou vzhledem k povaze úlohy relevantní. Z tohoto důvodu byla vytvořena jednoduchá aplikace, která je součástí projektu MetaEstimator.DataParser a která umožňuje převod dat do formátu zbaveného přebytečných informací se strukturou umožňující snadnější zpracování při konstrukci klasifikačního mechanismu.

```
dotnet MetaEstimator.DataParser.dll C:\data\stats.txt
    C:\data\log.txt C:\data\parsed.txt
```

Listing 9.2: Ukázka spuštění aplikace pro parsování dat

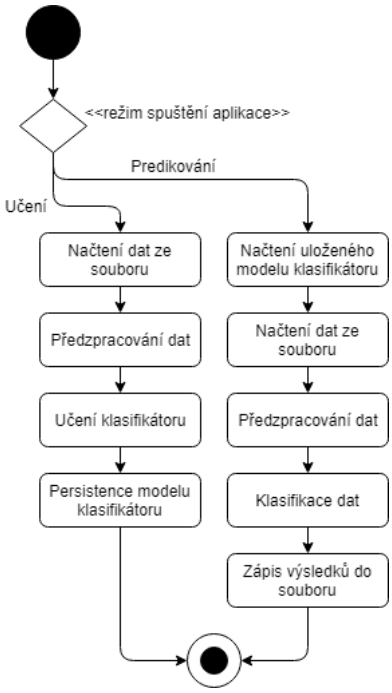
Při spuštění aplikace pro parsování dat je nutné uvést v parametrech v tomto pořadí cestu k souboru, který obsahuje popis vlastností polygonálních sítí, cestu k souboru, ve kterém jsou zachyceny úspěšnosti jednotlivých estimátorů a cestu výstupního souboru. (9.2)

Struktura výstupního souboru je tvořena jednoduchým textovým popisem, kdy každá řádka výstupního souboru odpovídá právě jedné síti. Na řádce jsou nejprve uvedeny ve sloupcích vlastnosti sítě a poté chyby jednotlivých estimátorů. Sloupce jsou od sebe odděleny znakem svislé čáry „|“

a mezi vlastnostmi sítě a chybami estimátorů je vynechán jeden prázdný sloupec.

## 9.5 Aplikace pro výběr optimálního estimátoru

Aplikace slouží pro natrénování a následné použití konkrétní kombinace zvolených algoritmů. Předpokladem pro spuštění je serializovaná konfigurace s typy a počátečním nastavením v souboru typu Json. Aplikace je nejprve spuštěna v režimu učení, ve kterém načte trénovací množinu dat, vytvoří model, který umožňuje predikci optimálního estimátoru a uloží jej do souboru. Poté je aplikace připravena pro opakované spouštění v režimu predikování, při kterém již polygonálním sítím, jejichž vlastnosti přečte z textového souboru předaného na vstupu, vybere estimátor křivosti, který považuje za optimální na základě aktuálního modelu. Obrázek 9.4 znázorňuje diagram aktivit aplikace.




Obrázek 9.4: Diagram aktivit aplikace pro výběr optimálního estimátoru.

## 9.6 Rozhraní mezi prostředím .NET a Python

Pro snazší práci s neuronovými sítěmi byly pro jejich konstrukci, učení a následnou klasifikaci použity knihovny Keras a TensorFlow, které jsou implementovány v jazyce Python. Z tohoto důvodu bylo nutné vytvořit rozhraní, které umožní spuštění skriptů napsaných v tomto jazyce, které výše uvedené knihovny využívají, z prostředí .NET Core, ve kterém je implementován zbytek aplikace.

Implementace tohoto rozhraní je realizována v knihovně MetaEstimator.PythonWrapper, konkrétně ve třídě PythonRuntime, která poskytuje požadovanou funkčnost v metodě RunPythonScript.(9.3)

```
public void RunPythonScript(
    string scriptFilePath,
    string args,
    DiscRepositorySegment inputSegment = null,
    DiscRepositorySegment outputSegment = null)
{
    ...
}
```

Listing 9.3: Hlavička metody pro spuštění Python skriptů

Metodě je na vstupu předána cesta k souboru skriptu v jazyce Python, který má být spuštěn, argumenty, které mají být předány skriptu při spuštění odděleny mezerou a nepovinnými argumenty jsou instance třídy DiscRepositorySegment, která reprezentuje dočasný soubor na pevném disku pro předání dat. Očekává-li skript nějaké vstupy, jako například počet vrstev a neuronů v neuronové síti, trénovací data, kontrolní data apod., jsou skriptu předány pomocí dočasného souboru uvedeného v parametru inputSegment. Pokud očekáváme výstup skriptu, který si poté přejeme v prostředí .NET Core číst, jako například indexy zvolených estimátorů, předáme skriptu při spuštění cestu k dočasnému souboru pomocí parametru outputSegment, do kterého skript data zapíše a my si následně tato data můžeme přečíst.

Po zavolání výše uvedené metody aplikace z konfigurace v souboru appsettings.json přečte cestu ke spustitelnému souboru interpretu jazyka Python a spustí jej jako nový proces s předanými argumenty, ke kterým se automaticky připojí i absolutní cesty k dočasným souborům pro výměnu dat. Při spuštění nového procesu dojde k přesměrování standardního textového výstupu do logu aplikace, ze které je nový proces spuštěn. Následně je do

doby, než dojde k ukončení spuštěného procesu, prováděno aktivní čekání. Po dokončení procesu je zapsán do logu návratový kód procesu a řízení předáno zpět volající proceduře, která má k dispozici data zapsaná v dočasném souboru.

## 9.7 Logování

Pro průběžné informování uživatele o činnosti aplikace a pro umožnění zpětného odhalení příčin možných chyb, jako jsou například nekonzistentní vstupní data, chybná konfigurace, selhání numerické metody, apod., aplikace po celou dobu své činnosti zapisuje klíčové informace o průběhu do textové konzole a zároveň do textového souboru.

Logování v aplikace je realizováno použitím knihovny Serilog, která je považována za jedno ze standardních řešení tohoto problému pro aplikace implementované v prostředí .NET Core.

## 9.8 Reprezentace dat

Jednotlivé prvky množiny dat jsou reprezentovány třídou DataPoint, která nese informaci o hodnotách sledovaných vlastností polygonální sítě v poli Stats a o odchylkách odhadů křivosti jednotlivých estimátorů v poli Divergences. Celá množina dat je poté reprezentována třídou DataSet, která mimo jiných funkčností poskytuje iterátor, pro průchod daty.

Pro práci s maticemi a vektory je použita knihovna MathNet.Numerics, která umožňuje provádění základních operací nad těmito objekty, jako například násobení matic, řešení soustavy reprezentované maticí, hledání inverzní matice, hledání vlastních čísel a vlastních vektorů apod.

## 9.9 Implementace algoritmů

### 9.9.1 Algoritmy pro předzpracování dat

#### Normalizace vlastností rozpětím

Normalizace vlastností polygonálních sítí je implementovány ve třídě MinMaxNormalizationProcessor. Pro použití této třídy nejsou nutné žádné další parametry (9.4)

```
IProcessor processor = new
   MinMaxNormalizationProcessor();
```

Listing 9.4: Ukázka použití normalizace rozpětím

#### Normalizace vlastností směrodatnou odchylkou

Pro normalizaci sledovaných vlastností za pomoci směrodatné odchylky slouží třída StandardDeviationNormalizationProcessor. (9.5)

```
IProcessor processor = new
   StandardDeviationNormalizationProcessor();
```

Listing 9.5: Ukázka použití normalizace rozptylem

#### Analýza hlavních komponent

Předzpracování vlastností sítí zahrnující možnost redukce počtu sledovaných vlastností a nalezení nové báze je implementováno ve třídě PrincipalComponentAnalysisProcessor. Při použití této třídy je nutné uvést nenulový počet vlastností, které mají být zachovány. (9.6)

```
IProcessor processor = new
   PrincipalComponentAnalysisProcessor
{
    RelevantStatsCount = 2
};
```

Listing 9.6: Ukázka použití analýzy hlavních komponent

### 9.9.2 Algoritmy pro klasifikaci dat

#### Rozhodovací strom

Klasifikátor s úplným binárním rozhodovacím stromem je implementován ve třídě MultilevelThresholdMetaEstimator. Při vytváření instance třídy rozhodovacího stromu je nutné uvést v parametru Depth hloubku rozhodovacího stromu, přičemž hodnota 1 odpovídá jedné vrstvě s rozhodovacími uzly a jedné vrstvě s koncovými uzly. Dále je nutné v parametru LearningType způsob jakým budou generovány konfigurace klasifikátoru, které budou následně otestovány a ze kterých bude vybrána nejlepší možná. Přípustné

hodnoty jsou LearningType.FullSearch pro prohledání všech možných konfigurací, LearningType.GeneticAlgorithm pro použití genetického algoritmu, nebo LearningType.SimulatedAnnealing pro simulované žíhání.

V případě použití úplného prohledávání se žádný další parametr neuvádí. Při použití genetického algoritmu je nutné uvést v parametru PopulationSize velikost populace, v parametru CrossbreedingProbability pravděpodobnost křížení prvků v každé iteraci, v parametru MutationProbability pravděpodobnost mutece a v parametru IterationsCount počet iterací genetického algoritmu.

Při testování optimalizace rozhodovacího stromu s použitím genetického algoritmu se ukázalo, že po již relativně nízkém počtu iterací dojde při velikosti populace čítající řádově stovky prvků k ustálení v lokálním extrému a následném snížení rychlosti prohledávání dalších řešení způsobeném nízkou diverzitou prvků v populaci. Pro odstranění tohoto problému byly zavedeny nepovinné parametry umožňující snížení stavu populace po určitém počtu iterací, což umožní opětovné křížení dosud nalezených prvků s prvky s horším ohodnocením. Parametry pro nastavení redukce populace jsou ClearingInterval určující po kolika iteracích má dojít k redukci populace a KeepAlive určující na jaké množství má být populace zredukována. Při neuvedení parametru ClearingInterval k redukci populace nedojde.

Při použití simulovaného žíhání je nutné uvést v parametrech SimulationSteps a SimulationProbabilities dvě stějně dlouhá pole, přičemž parametr SimulationSteps obsahuje postupně intervaly v sekundách, po které je v systému udržována konstatní teplota a na odpovídajícím místě v poli předaném jako parametr SimulationProbabilities je uvedena pravděpodobnost přijetí horšího řešení v daném intervalu.

Příklad použití rozhodovacího stromu je uveden níže. (9.7)

```
IMetaEstimator metaEstimator = new
   MultilevelThresholdMetaEstimator
{
    Depth = 2,
    LearningType = LearningType.SimulatedAnnealing,
    SimulationSteps = new []{ 600, 600, 600, 600,
       600, 600, 600, 600, 600, 600 },
    SimulationProbabilities = new []{ 0.9, 0.8, 0.7,
       0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0 }
};
```

Listing 9.7: Ukázka použití klasifikátoru s rozhodovacím stromem

### Neuronová síť

Logika klasifikátoru s neuronovou sítí je implementována ve třídě LinearNeuralMetaEstimator, samotná neuronová síť je pak implementována v Pythonu ve skriptech `train.py` a `predict.py`, které jsou umístěny v adresáři `MetaEstimator/LinearNeural/`, který je umístěn v knihovně `MetaEstimator.PythonLibrary`.

Při trénování klasifikátoru s neuronovou sítí nejprve aplikace převede hodnoty s odchylkami jednotlivých estimátorů na binární vektory, která mají hodnotu 1 na pozici odpovídající optimálnímu estimátoru pro danou polygonální síť a hodnotu 0 jinde. Tyto vektory jsou následně spolu s vektory vlastností zapsány na disk, načež dojde ke spuštění skriptu pro učení neuronové sítě. Skript přečte hodnoty vlastností a tříd reprezentovaných binárními vektory, počet a velikosti vrstev, sestaví model neuronové sítě a natrénuje ji na předaných datech. Po dokončení učení neuronové sítě je vytvořený model uložen na disk.

Při predikci optimálních estimátorů dojde k zápisu vlastností na disk stejně jako v případě učení neuronové sítě. Po spuštění skriptu `predict.py` dojde k načtení hodnot vlastností a uloženého modelu neuronové sítě z disku. Poté je provedena klasifikace přivedením vektorů s vlastnostmi na vstupní vrstvu neuronové sítě a třídy z výstupu sítě uloženy na disk pro následné vyhodnocení.

Zde je uveden příklad použití klasifikátoru s lineární neuronovou sítí. (9.8) V parametru Layers jsou uvedeny počty neuronů v jednotlivých vrstvách v pořadí od vstupní vrstvy po výstupní. Hodnota -1 je při vytváření neuronové sítě automaticky nahrazena počtem sledovaných vlastností vstupních dat.

```
IMetaEstimator metaestimator = new
   LinearNeuralMetaEstimator
{
    Layers = new []{ -1, 34, 34 }
};
```

Listing 9.8: Ukázka použití klasifikátoru s neuronovou sítí

### Shlukování

Třída ClusterMetaEstimator poskytuje implementaci klasifikátoru založeném algoritmu shlukování prvků se stejnými třídami. Pro použití klasifikátoru je nutné v parametru DistanceType uvést jednu z metrik pro výpočet vzdálenosti klasifikovaných bodů od těžišť jednotlivých shluků. Přípustné

hodnoty parametru jsou DistanceType.Euclidean pro použití Euklidovské vzdálenost, DistanceType Cosine pro použití Cosinové vzdálenosti, nebo DistanceType.Manhatton pro Manhattanskou vzdálenost. (9.9)

```
IMetaEstimator metaEstimator = new
   ClusterMetaEstimator
{
    DistanceType = DistanceType.Cosine
};
```

Listing 9.9: Ukázka použití klasifikátoru se shlukováním

#### Aproximace metodou nejmenších čtverců

Pro použití klasifikace založené na regresi chyb estimátorů pomocí aproximace metodou nejmenších čtverců lineárním polynomem je implementována třída klasifikátoru LeastSquaresMetaEstimator. Při použití tohoto klasifikátoru se neuvádí žádné parametry.

### 9.9.3 Metody vyhodnocení

Metriky pro vyhodnocení úspěšností klasifikace jsou implementovány ve třídách RelativeDivergenceEvaluator (relativní chyba) a AccuracyEvaluator (přesnost). Pro použití obou metod není nutné uvádět žádné další parametry.

# 10 Uživatelská příručka

## 10.1 Zdrojové kódy

Následující výčet popisuje rozdělení zdrojových kódů řešení aplikace do jednotlivých projektů (modulů).

- MetaEstimator.ConfigurationsFactory - Konzolová aplikace umožňující automatické vygenerování všech možných kombinací implementovaných algoritmů pro předzpracování dat, metod klasifikace a způsobů ohodnocení výsledku a jejich následné uložení do relační databáze.
- MetaEstimator.ConsoleApp - Konzolová aplikace pro klasifikaci polygonálních sítí do jednotlivých tříd.
- MetaEstimator.Core - Hlavní knihovna obsahující implementace jednotlivých metod klasifikace, předzpracování dat a vyhodnocen výsledků.
- MetaEstimator.Dao - Knihovna poskytující rozhraní mezi .NET Core aplikacemi a relační databází SQL Server.
- MetaEstimator.DataParser - Konzolová aplikace pro převod výchozích dat do jednotného formátu pro následující zpracování.
- MetaEstimator.Dto - Knihovna obsahující kódovou reprezentaci entit relační databáze.
- MetaEstimator.Infrastructure - Knihovna obsahující implementaci aplikačního kontextu použitého napříč konzolovými aplikacemi a logiky pro uložení dat na disk a jejich zpřístupnění interpretu jazyka Python.
- MetaEstimator.Logging - Knihovna pro logování průběhu procesů konzolových aplikací.
- MetaEstimator.PythonLibrary - Repozitář skriptů jazyka Python.
- MetaEstimator.PythonWraper - Knihovna s implementací logiky pro spouštění interpretu jazyka Python
- MetaEstimator.Service - Služba realizovaná konzolovou aplikací, umožňující dávkové zpracování předem vygenerovaných kombinací algoritmů pro předzpracování dat, klasifikaci a vyhodnocení.
- MetaEstimator.Tests - Projekt s jednotkovými testy.

## 10.2 Překlad a instalace databáze

Pro překlad zdrojových kódů aplikace je nutné mít nainstalované prostředí .NET Core verze 2.1. Překlad je možné spustit vykonáním následujícího příkazu v kořenovém adresáři řešení. (10.1)

```
dotnet build MetaEstimator.sln
```

Listing 10.1: Příkaz pro překlad zdrojových kódů

Instalaci relační databáze, kterou používá služba pro dávkové zpracování vygenerovaných konfigurací klasifikátoru je možné provést spuštěním skriptů umístěných v adresáři `SQLScripts`.

## 10.3 Služba pro vyhodnocení úspěšnosti metod klasifikace

Pro úspěšné spuštění služby je nutné otevřené připojení do databáze SQL Serveru s již existující tabulkou CONFIGURATION_RECORD. Službu lze následně spustit bez argumentů a v takovém případě postupně zpracovává všechny záznamy uložené v databázi. Službu je také možné spustit s libovolným množstvím identifikátorů záznamů s konfiguracemi uložených v databázi, v takovém případě služba zpracuje pouze záznamy s danými identifikátory.

Příkazy pro spuštění služby pro dávkové zpracování jsou uvedeny níže. První příkaz ukazuje spuštění bez argumentů, druhý spuštění pro zpracování záznamů s identifikátory 234 a 876. (10.2)

```
dotnet MetaEstimator.Service.dll
dotnet MetaEstimator.Service.dll 234 876
```

Listing 10.2: Příkazy pro spuštění služby dávkového zpracování

## 10.4 Aplikace pro výběr optimálního estimátoru

Při spuštění aplikace je na vstupu pomocí argumentů příkazové řádky předán příznak, který řídí, zda má aplikace provádět učení (-t), nebo zda má predikovat optimální estimátory křivosti (-p). Dalším argumentem je cesta

k souboru, do kterého se má naučený model klasifikátoru uložit, resp. ze kterého se má model načíst, dále cesta k souboru daty obsahující vlastnosti polygonálních sítí a v případě učení i chyby jednotlivých estimátorů. V případě spuštění v režimu predikování je posledním parametrem cesta k souboru, do kterého mají být zapsány indexy zvolených estimátorů.

Níže uvedené příkazy demonstrují postup spuštění konzolové aplikace nejprve pro trénování modelu a poté pro predikci. (10.3)

```
dotnet MetaEstimator.ConsoleApp.dll -t C:\data_learn.
   csv C:\model.json
dotnet MetaEstimator.ConsoleApp.dll -p C:\data_test.
   csv  C:\model.json C:\results.txt
```

Listing 10.3: Příklad příkazů spuštění konzolové aplikace

## 10.5 Nastavení aplikací

Pro nastavení konzolové aplikace i služby pro dávkové zpracování slouží soubor appsettings.json umístěný v adresáři příslušné aplikace. V souboru je možné nastavit cestu k souboru se spustitelným interpretem jazyka Python, cestu k existujícímu adresáři, do kterého může aplikace ukládat dočasné soubory pro přenos dat mezi .NET aplikací a skripty Pythonu, řetězec pro připojení do databáze[2] a soubor, do kterého bude aplikace průběžně logovat informace o své činnosti.

[2]Řetězec pro připojení do databáze je nutné nastavit pouze u služby pro dávkové zpracování.

# 11 Výsledky

Tabulka 11.1 obsahuje nejlepší dosažené výsledky použitých klasifikačních algoritmů napříč různými metodami pro předzpracování dat. Nejlepších výsledků z hlediska minimální relativní chyby bylo dosaženo využitím algoritmů klasifikátoru s rozhodovacím stromem o třech vrstvách s rozhodovacími uzly s využitím optimalizace pomocí genetického algoritmů. Kompletní výsledky jsou uvedeny v příloze B.

Tabulka 11.1: Tabulka porovnání výsledků klasifikačních algoritmů.

| **Předzprac.** | **Klasifikátor** | **Rel. chyba** | **Přesnost** |
|---|---|---|---|
| | Rozhodovací strom (FS, h=1) | 1.195 | 0.242 |
| PCA(3) | Rozhodovací strom (GA, h=2) | 0.764 | 0.318 |
| | Rozhodovací strom (GA, h=3) | 0.588 | 0.328 |
| Směr. odchylka, PCA(6) | Rozhodovací strom (SA, h=2) | 1.476 | 0.227 |
| Směr. odchylka | Rozhodovací strom (SA, h=3) | 0.637 | 0.346 |
| PCA(1) | Shlukování (Euclidean) | 92,543 | 0.041 |
| | Shlukování (Cosine) | 14.686 | 0.005 |
| PCA(1) | Shlukování (Manhatton) | 92.543 | 0.041 |
| Směr. odchylka, PCA(4) | Neuronová síť se dvěma vrstvami (4, 34) | 18.238 | 0.038 |
| PCA(6) | Metoda nejmenších čtverců | 98.430 | 0.023 |

Celý experiment byl realizován na množině dat čítající 3938 vzorků. Tato množina dat byla na počátku každého testu vybraných algoritmů rozdělena na dvě poloviny, přičemž jedna polovina byla použita pro natrénování klasifikačních a normalizačních algoritmů a druhá polovina pro určení úspěšnosti klasifikátoru.

[3]PCA(n) = Analýza hlavních komponent s výběrem n vlastností
[4]FS = FullSearch (Slepé prohledávání)
[5]GA = Genetický algoritmus
[6]SA = Simulované žíhání

# 12 Závěr

Použitím výše uvedených postupů klasifikace se povedlo dosáhnout lepších výsledků z hlediska velikosti relativní chyby, než při klasifikaci pomocí rozhodovacích stromů s jedním a s dvěma rozhodovacími uzly a použitím tří estimátorů křivosti. Zatímco v původní práci bylo dosaženo výsledku s hodnotou relativní chyby $d_r = 1.18$ pro rozhodovací strom s jedním rozhodovacím uzlem a $d_r = 0,86$ pro strom se dvěma rozhodovacími uzly, v této práci se podařilo zkonstruovat klasifikátor, který dosáhl relativní chyby $d_r = 0.59$. Pro ověření zlepšení výsledků klasifikátoru byl úspěšně zreplikován klasifikátor popsaný ve výchozí práci, který při použití stejné množiny testovacích dat poskytl přibližně stejné výsledky. ($d_r = 1,19$) [20]

Nejlepších výsledků bylo dosaženo s využitím analýzy hlavních komponent pro redukci původního počtu sledovaných vlastností na 3 a klasifikací pomocí úplného binárního rozhodovacího stromu se dvěma vrstvami rozhodovacích uzlů a jednou vrstvou koncových uzlů. Rozhodovací strom byl sestaven s využitím optimalizace genetickým algoritmem.

Ostatní způsoby klasifikace výsledky původního rozhodovacího stromu nepřekonaly, nicméně se dá očekávat, že při použití jiné konfigurace neuronové sítě, by mohlo dojít ke zlepšení výsledků klasifikátoru. Jedním ze způsobů, pomocí kterých bychom mohli dosáhnout zlepšení výsledků klasifikátoru s neuronovou sítí, je použití tzv. multi-label klasifikace, kdy není vzorku trénovací množiny přiřazena právě jedna třída, nýbrž je vybráno několik tříd, které pro daný vzorek dat splňují požadovaná kritéria.

Další možností je použití sofistikovanějších algoritmů (např. TDIDT[7] nebo ID3[8]) pro konstrukci rozhodovacích stromů s větším počtem vrstev či větším větvením.

[7] TDIDT - Top Down Induction od Decision Trees
[8] ID3 - Iterative Dichotomiser 3

# Literatura


[1] Bac, A. et al. Application of discrete curvatures to surface mesh simplification and feature line extraction. *Courbure discrète: théorie et applications*. 2013, s. 31.

[2] Banchoff, T. – Lovett, S. *Differential Geometry of Curves and Surfaces*. CRC Press, 2010. ISBN 9781439894057.

[3] Berka, P. *Dobỳvání znalostí z databází*. Academia, 2003. Dostupné z: `https://books.google.cz/books?id=tGvFAAAACAAJ`. ISBN 9788020010629.

[4] Bhadeshia, H. Neural networks and information in materials science. *Statistical Analysis and Data Mining: The ASA Data Science Journal*. 2009, 1, 5, s. 296–305.

[5] Botsch, M. et al. Geometric Modeling Based on Polygonal Meshes. 2008. Dostupné z: `https://hal.inria.fr/inria-00337991`. This document is the support of a course given at the Eurographics 2008 conference (Crete, Greece, April 14-18).

[6] Botsch, M. et al. *Polygon Mesh Processing*. A K Peters, Ltd., 2010. ISBN 978-1-56881-426-1.

[7] Břetislav, F. – Růžičková, I. Matematika 3. *Fakulta elektrotechniky a komunikačních technologií VUT v Brně*. 2003.

[8] Chen, Q. et al. Adaptive total variation denoising based on difference curvature. *Image and Vision Computing*. 03 2010, 28, s. 298–306. doi: 10.1016/j.imavis.2009.04.012.

[9] D. Gatzke, T. – Grimm, C. Estimating Curvature on Triangular Meshes. *International Journal of Shape Modeling*. 06 2006, 12, s. 1–28. doi: 10.1142/S0218654306000810.

[10] Carmo, M. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976. ISBN 9780132125895.

[11] Goldfeather, J. – Interrante, V. A Novel Cubic-Order Algorithm for Approximating Principal Direction Vectors. *ACM Trans. Graph.* 01 2004, 23, s. 45–63. doi: 10.1145/966131.966134.

[12] Henderson, D. – Jacobson, S. – Johnson, A. The Theory and Practice of Simulated Annealing. *Handbook of Metaheuristics*. 04 2006, s. 287–319. doi: 10.1007/0-306-48056-5_10.

[13] Kamiński, B. – Jakubczyk, M. – Szufel, P. A framework for sensitivity analysis of decision trees. 2017. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5767274/.

[14] Kolář, I. – Pospíšilová, L. Diferenciální geometrie křivek a ploch. 2007. Dostupné z: https://is.muni.cz/elportal/estud/prif/ps08/geom/web/dl/dgpdf_bez.pdf.

[15] Prantl, M. – Váša, L. Estimation of differential quantities using Hermite RBF interpolation. *The Visual Computer*. 09 2017. doi: 10.1007/s00371-017-1438-x.

[16] Prantl, M. et al. Curvature-Based Feature Detection for Head Modeling. *Procedia Computer Science*. 12 2017, 108, s. 2323–2327. doi: 10.1016/j.procs.2017.05.105.

[17] Stich, S. U. – Cordonnier, J.-B. – Jaggi, M. Sparsified SGD with Memory. In Bengio, S. et al. (Ed.) *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2018. s. 4447–4458. Dostupné z: http://papers.nips.cc/paper/7697-sparsified-sgd-with-memory.pdf.

[18] Subramanian, V. *Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch*. Packt Publishing, 2018. ISBN 9781788626071.

[19] Tanaka, H. T. – Ikeda, M. – Chiaki, H. Curvature-based face surface recognition using spherical correlation. Principal directions for curved object recognition. In *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition*, s. 372–377, April 1998. doi: 10.1109/AFGR.1998.670977.

[20] Váša, L. et al. Mesh Statistics for Robust Curvature Estimation. *Eurographics Symposium on Geometry Processing*. 2016.

[21] Váša, L. – Kühnert, T. – Brunnett, G. Multivariate analysis of curvature estimators. *Computer-Aided Design and Applications*. 2017, 14, 1, s. 58–69. doi: 10.1080/16864360.2016.1199756. Dostupné z: https://doi.org/10.1080/16864360.2016.1199756.

[22] Wendland, H. *Scattered Data Approximation*. 17. Cambridge University Press, 01 2005.

[23] Ziman, J. – Ziman, P. *Models of Disorder: The Theoretical Physics of Homogeneously Disordered Systems*. Cambridge University Press, 1979. ISBN 9780521217842.

[24] Šigut, P. – Vanecek, P. – Váša, L. Analytic Surface Detection in CAD Exported Models. s. 278–285, 01 2019. doi: 10.5220/0007396702780285.

# A Obsah CD

Přílohou této práce je CD obsahující elektronickou formu práce ve formátu PDF včetně zadání, kompletní zdrojové kódy aplikace a vstupní data. Adresářová struktura na disku je znázorněna níže.

- `01_Dokumentace` - Adresář se zadáním a elektronickou formou písemné části bakalářské práce
- `02_Zdrojove_kody` - Adresář se zdrojovými kódy
- `03_Data` - Adresář se vstupními daty

# B Kompletní výsledky

| **Předzpracování** | **Klasifikace** | **Relativní chyba** | **Přesnost** |
|---|---|---|---|
| | Shlukování (Cosine) | 14,686 | 0,005 |
| Směr. odchylka, PCA (6) | Neuronová síť [6,34] | 2209,803 | 0,007 |
| Směr. odchylka, PCA(6) | Shlukování (Manhatton) | 1365,7 | 0,009 |
| Směr. odchylka, PCA(6) | Shlukování (Euclidean) | 1366,121 | 0,008 |
| Směr. odchylka, PCA(6) | Shlukování (Cosine) | 110,864 | 0,021 |
| Směr. odchylka, PCA(9) | Rozhod. strom (h=2, simulované žíhání) | 4,649 | 0,219 |
| Směr. odchylka, PCA(6) | Neuronová síť [6,34,34] | 41,305 | 0,235 |
| Směr. odchylka, PCA(5) | Rozhod. strom (h=2, genetický alg.) | 4,617 | 0,233 |
| Směr. odchylka, PCA(5) | Rozhod. strom (h=2, genetický alg.) | 1,845 | 0,223 |
| Směr. odchylka, PCA(5) | Rozhod. strom (h=1, úplné prohledávání) | 3,433 | 0,162 |
| Směr. odchylka, PCA(5) | Neuronová síť [5,34,34,34] | 41,305 | 0,235 |
| Směr. odchylka, PCA(5) | Neuronová síť [5,34,34] | 41,305 | 0,235 |
| Směr. odchylka, PCA(5) | Neuronová síť [5,34] | 64,111 | 0,013 |
| Směr. odchylka, PCA(6) | Neuronová síť [6,34,34,34] | 41,305 | 0,235 |
| Směr. odchylka, PCA(6) | Rozhod. strom (h=1, úplné prohledávání) | 2,154 | 0,224 |
| Směr. odchylka, PCA(6) | Rozhod. strom (h=2, simulované žíhání) | 1,476 | 0,227 |
| Směr. odchylka, PCA(6) | Rozhod. strom (h=2, genetický alg.) | 1,886 | 0,229 |
| Směr. odchylka, PCA(6) | Rozhod. strom (h=2, genetický alg.) | 1,814 | 0,22 |
| Směr. odchylka, PCA(5) | Shlukování (Manhatton) | 1365,699 | 0,008 |
| Směr. odchylka, PCA(5) | Shlukování (Euclidean) | 1366,138 | 0,008 |
| Směr. odchylka, PCA(5) | Shlukování (Cosinova vzd.) | 110,864 | 0,021 |
| Směr. odchylka, PCA(3) | Rozhod. strom (h=2, genetický alg.) | 4,667 | 0,213 |

| **Předzpracování** | **Klasifikace** | **Relativní chyba** | **Přesnost** |
|---|---|---|---|
| Směr. odchylka, PCA(3) | Rozhod. strom (h=1, úplné prohledávání) | 3,433 | 0,162 |
| Směr. odchylka, PCA(3) | Neuronová síť [3,34,34,34] | 41,305 | 0,235 |
| Směr. odchylka, PCA(3) | Neuronová síť [3,34,34] | 41,305 | 0,235 |
| Směr. odchylka, PCA(3) | Neuronová síť [3,34] | 2525,623 | 0,014 |
| Směr. odchylka, PCA(3) | Shlukování (Manhatton) | 1365,9 | 0,009 |
| Směr. odchylka, PCA(3) | Shlukování (Cosinova vzd.) | 110,864 | 0,021 |
| Směr. odchylka, PCA(3) | Shlukování (Euclidean) | 1365,885 | 0,009 |
| Směr. odchylka, PCA(4) | Rozhod. strom (h=1, úplné prohledávání) | 3,433 | 0,162 |
| Směr. odchylka, PCA(4) | Neuronová síť [4,34,34,34] | 41,305 | 0,235 |
| Směr. odchylka, PCA(4) | Neuronová síť [4,34,34] | 41,305 | 0,235 |
| Směr. odchylka, PCA(4) | Neuronová síť [4,34] | 18,238 | 0,038 |
| Směr. odchylka, PCA(4) | Shlukování (Manhatton) | 1365,727 | 0,008 |
| Směr. odchylka, PCA(4) | Shlukování (Euclidean) | 1366,142 | 0,008 |
| Směr. odchylka, PCA(4) | Shlukování (Cosinova vzd.) | 110,864 | 0,021 |
| Směr. odchylka, PCA(2) | Rozhod. strom (h=1, úplné prohledávání) | 6,765 | 0,102 |
| Směr. odchylka, PCA(2) | Neuronová síť [2,34,34,34] | 41,305 | 0,235 |
| PCA(1) | Rozhod. strom (h=1, úplné prohledávání) | 1,314 | 0,255 |
| PCA(1) | Neuronová síť [1,34,34,34] | 41,305 | 0,235 |
| PCA(1) | Neuronová síť [1,34,34] | 41,305 | 0,235 |
| PCA(1) | Neuronová síť [1,34] | 109,578 | 0,021 |
| PCA(1) | Shlukování (Euclidean) | 92,543 | 0,041 |
| PCA(1) | Shlukování (Cosinova vzd.) | 110,864 | 0,021 |

| **Předzpracování** | **Klasifikace** | **Relativní chyba** | **Přesnost** |
|---|---|---|---|
| Rozpětí | Rozhod. strom (h=3, simulované žíhání) | 0,822 | 0,313 |
| Rozpětí | Rozhod. strom (h=3, genetický alg.) | 1,073 | 0,224 |
| Rozpětí | Rozhod. strom (h=3, genetický alg.) | 0,774 | 0,333 |
| PCA(1) | Shlukování (Manhatton) | 92,543 | 0,041 |
| PCA(3) | Shlukování (Cosinova vzd.) | 110,864 | 0,021 |
| PCA(2) | Rozhod. strom (h=1, úplné prohledávání) | 1,204 | 0,242 |
| PCA(2) | Neuronová síť [2,34,34,34] | 41,305 | 0,235 |
| PCA(2) | Neuronová síť [2,34,34] | 41,305 | 0,235 |
| PCA(2) | Neuronová síť [2,34] | 109,417 | 0,021 |
| PCA(2) | Shlukování (Manhatton) | 92,626 | 0,042 |
| PCA(2) | Shlukování (Euclidean) | 92,919 | 0,03 |
| PCA(2) | Shlukování (Cosinova vzd.) | 110,864 | 0,021 |
| PCA(3) | Shlukování (Euclidean) | 93,097 | 0,023 |
| Rozpětí | Rozhod. strom (h=1, úplné prohledávání) | 1,195 | 0,242 |
| Směr. odchylka | Shlukování (Cosinova vzd.) | 110,864 | 0,021 |
| | Rozhod. strom (h=3, simulované žíhání) | 0,84 | 0,274 |
| | Rozhod. strom (h=3, genetický alg.) | 1,233 | 0,245 |
| | Rozhod. strom (h=3, genetický alg.) | 0,588 | 0,328 |
| Směr. odchylka | Shlukování (Euclidean) | 1366,121 | 0,008 |
| | Rozhod. strom (h=1, úplné prohledávání) | 1,195 | 0,242 |
| | Shlukování (Manhatton) | 127,038 | 0,026 |
| | Shlukování (Euclidean) | 127,004 | 0,025 |

| **Předzpracování** | **Klasifikace** | **Relativní chyba** | **Přesnost** |
|---|---|---|---|
| Směr. odchylka | Shlukování (Manhatton) | 1365,521 | 0,008 |
| Směr. odchylka | Rozhod. strom (h=3, simulované žíhání) | 0,637 | 0,346 |
| Směr. odchylka | Rozhod. strom (h=3 ,genetický alg.) | 1,127 | 0,246 |
| Směr. odchylka | Rozhod. strom (h=1, úplné prohledávání) | 1,195 | 0,242 |
| PCA(3) | Shlukování (Manhatton) | 93,153 | 0,023 |
| Směr. odchylka,PCA(1) | Shlukování (Cosinova vzd.) | 110,864 | 0,021 |
| PCA(6) | Shlukování (Manhatton) | 126,884 | 0,024 |
| PCA(6) | Shlukování (Euclidean) | 126,901 | 0,024 |
| PCA(6) | Shlukování (Cosinova vzd.) | 110,864 | 0,021 |
| PCA(6) | Rozhod. strom (h=1, úplné prohledávání) | 1,204 | 0,242 |
| Směr. odchylka, PCA(1) | Shlukování (Euclidean) | 1364,709 | 0,008 |
| Směr. odchylka, PCA(1) | Shlukování (Manhatton) | 1364,709 | 0,008 |
| Směr. odchylka, PCA(2) | Shlukování (Manhatton) | 1366,039 | 0,008 |
| Směr. odchylka, PCA(2) | Shlukování (Euclidean) | 1366,039 | 0,008 |
| Směr. odchylka, PCA(2) | Shlukování (Cosinova vzd.) | 110,864 | 0,021 |
| Směr. odchylka, PCA(1) | Rozhod. strom (h=1, úplné prohledávání) | 55,194 | 0,015 |
| PCA(4) | Shlukování (Manhatton) | 101,162 | 0,031 |
| PCA(4) | Shlukování (Euclidean) | 110,66 | 0,031 |
| PCA(4) | Shlukování (Cosinova vzd.) | 110,864 | 0,021 |
| PCA(3) | Rozhod. strom (h=2, genetický alg.) | 1,015 | 0,271 |
| PCA(3) | Rozhod. strom (h=2, genetický alg.) | 0,764 | 0,318 |
| PCA(3) | Rozhod. strom (h=1, úplné prohledávání) | 1,204 | 0,242 |

| **Předzpracování** | **Klasifikace** | **Relativní chyba** | **Přesnost** |
|---|---|---|---|
| PCA(4) | Rozhod. strom (h=1, úplné prohledávání) | 1,204 | 0,242 |
| PCA(5) | Rozhod. strom (h=1, úplné prohledávání) | 1,204 | 0,242 |
| PCA(5) | Shlukování (Manhatton) | 126,886 | 0,024 |
| PCA(5) | Shlukování (Euclidean) | 126,901 | 0,024 |
| PCA(5) | Shlukování (Cosinova vzd.) | 110,864 | 0,021 |
| Směr. odchylka, PCA(6) | Metoda nejmenších čtverců | 388,262 | 0,002 |
| Směr. odchylka, PCA(5) | Metoda nejmenších čtverců | 345,282 | 0,021 |
| Směr. odchylka, PCA(4) | Metoda nejmenších čtverců | 343,431 | 0,023 |
| Směr. odchylka, PCA(3) | Metoda nejmenších čtverců | 247,856 | 0,019 |
| Směr. odchylka, PCA(2) | Metoda nejmenších čtverců | 245,23 | 0,017 |
| PCA(6) | Metoda nejmenších čtverců | 98,43 | 0,023 |
| PCA(5) | Metoda nejmenších čtverců | 106,561 | 0,014 |
| PCA(4) | Metoda nejmenších čtverců | 110,763 | 0,021 |
| PCA(3) | Metoda nejmenších čtverců | 112,445 | 0,021 |
| PCA(2) | Metoda nejmenších čtverců | 112,443 | 0,021 |
| PCA(1) | Metoda nejmenších čtverců | 110,862 | 0,021 |
| Směr. odchylka, PCA(1) | Metoda nejmenších čtverců | 246,512 | 0,02 |