Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Testování rozhodovacích algoritmů na notebooku Jupyter

Bc. Miroslav Kálal

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Miroslav Kálal

Systémové inženýrství a informatika
Informatika

Název práce

Testování rozhodovacích algoritmů na notebooku Jupyter

Název anglicky

Testing decision algorithms on Jupyter notebook

Cíle práce

Cílem práce je navrhnout metodiku testování a vzájemného porovnávání efektivity rozhodovacích algoritmů založených na strojovém učení na notebooku Jupyter. Navrženou metodiku student otestuje na datech dostupných na internetu a zhodnotí její výhody a nevýhody.

Metodika

Student vysvětlí architekturu a fungování notebooků Jupyter a popíše jejich použití pro zpracování datových souborů. Dále podá přehled moderních rozhodovacích algoritmů založených na strojovém učení a navrhne obecný způsob jejich učení a testování na notebooku Jupyter. Pro vybranou rozhodovací úlohu otestuje několik algoritmů strojového učení na datovém souboru, který získá z některého ze standardních úložišť na internetu. Výsledný soubor Jupyteru (soubor .ipynb) bude obsahovat kromě programové realizace rozhodovacích algoritmů také analýzu datového souboru, výsledky získané v procesu učení a výsledky testování naučeného algoritmu. Výsledky, pro které to bude vhodné, budou prezentovány v grafické formě.

Doporučený rozsah práce

60

Klíčová slova

rozhodovací algoritmy, strojové učení, jupyter notebook

Doporučené zdroje informací

Galea, A.: Beginning Data Science with Python and Jupyter, Packt, 2018
SUMMERFIELD, M. *Python 3 : výukový kurz.* Brno: Computer Press, 2010. ISBN 978-80-251-2737-7.

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

doc. Ing. Arnošt Veselý, CSc.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 30. 3. 2021

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 30. 3. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 30. 03. 2021

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Testování rozhodovacích algoritmů na notebooku Jupyter" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31. března 2021 .....

Poděkování

Rád bych touto cestou poděkoval vedoucímu práce doc. Ing. Arnoštovi Veselému, CSc. za cenné rady, konstruktivní připomínky a univerzitě za poskytnutí přístupu do databází odborných zdrojů.

Testování rozhodovacích algoritmů na notebooku Jupyter

Abstrakt

Tato práce se zabývá Strojovým učením a testováním jeho algoritmů v aplikaci Jupyter notebook. Strojové učení je větev umělé inteligence, která zasahuje do oblasti statistiky. Technika učení s učitelem využívá předem připravenou trénovací sadu s označenými daty, na které se algoritmy učí řešení úlohy klasifikace, nebo regrese. Při klasifikaci algoritmus vrací zařazení vstupu do některé z definovaných tříd. Při regresi odhaduje hodnotu proměnné. Výkon klasifikátoru lze měřit procentuální úspěšností jeho predikcí na testovacích datech. Algoritmus SVM se při učení klasifikace snaží najít optimální lineární oddělovač mezi třídami. Nejbližších k sousedů vrací třídu, která je nejčastěji zastoupena mezi sousedy zadaného vstupu, které jsou mu v určitém smyslu nejblíže. Neuronové sítě jsou rodina algoritmů, která vychází z biologických neuronových sítí. skládá se z neuronů, které jsou navzájem propojeny a jejichž výstup závisí na překonání určité hranice hodnotou jejich vstupů. Aplikace Jupyter notebook je lokálně spouštěná webová aplikace, která umožňuje kombinovat formátovaný text a zdrojový kód různých programovacích jazyků v jednom souboru. Zdrojový kód lze vkládat do oddělených buněk, které jsou samostatně spustitelné. Představený postup učení algoritmů se skládá z analýzy a přípravy trénovací sady, volby trénovací základny, tréninku křížovou validací a optimalizaci parametrů validačními křivkami, včetně metodiky testování algoritmů. Postup je demonstrován na klasifikaci typu skla.

Klíčová slova Strojové učení, učení s učitelem, křížová validace, klasifikace, Jupyter notebook, Python, datová analýza

Testing decision algorithms on Jupyter notebook

Abstract

This thesis deals with Machine learning and testing of its algorithms in the Jupyter notebook application. Machine learning is a branch of Artificial intelligence which is connected to statistics. The technique of supervised learning uses prepared training dataset with labeled data from which the algorithms are trained for solving a classification or regression task. Classification is a task of classifying the given input as a member of one of the defined classes, regression is a target variable value aproximation task. A classifier performance can be measured by the percentage of successful predictions on testing data. The SVM algorithm solves the classification problem by finding the optimal linear separator of the defined classes. K Nearest neighbours returns the class which appears the most frequently among the given input's neighbors which are in some sense the nearest. Neural networks are a family of algorithms, which is derived from biological neural networks. A Neural network is composed of neurons which are connected to each other and whose output is determined by their inputs surpassing a certain threshold. Jupyter notebook is a locally run web application which allows the user to combine formatted text with source codes of different programming languages in a single file. The code is placed in cells which can be run separately. The suggested method of algorithm training is composed of analysis and training dataset preparation, cross-validation training and parameter optimalization with the use of validation curves, including the algorithm testing methodology. The method is demonstrated on glass type classification.

Keywords Machine Learning, supervised learning, cross-validation, classification, Jupyter notebook, Python, data analysis

Obsah

Seznam obrázků

1 Úvod

Strojové učení představuje alternativní přístup vůči algoritmickému řešení problémů pomocí počítače, který je v některých případech nezbytný či výhodnější než jiná řešení. V současnosti je jedním z předních odvětví Počítačových věd. Tato práce se zabývá jednou z jeho technik a sestavením metodiky testování a vzájemného porovnávání jeho algoritmů v aplikaci Jupyter notebook.

Obsah práce se skládá z několika kapitol. Kapitola Teoretická východiska obsahuje souhrn teoretických znalostí o Strojovém učení, používaných algoritmech a krocích učení a seznámení s aplikací Jupyter notebook. Tyto znalosti jsou využity v kapitole Realizace, která obsahuje postup učení, sestavený na jejich základě, včetně testovací metodiky. Obsahuje také demonstraci postupu na vybrané úloze. Výsledky práce shrnuje kapitola Závěr.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je navrhnout metodiku testování a vzájemného porovnávání efektivity rozhodovacích algoritmů založených na strojovém učení na notebooku Jupyter. Navržená metodika bude otestována na datech dostupných na internetu s následným zhodnocením jejích výhod a nevýhod.

2.2 Metodika

Práce je rozdělena na získání základních teoretických znalostí a jejich uplatnění při realizaci a předvedení metodiky. Teoretické znalosti byly získány z odborných a učebních materiálů. Tyto znalosti zahrnují charakteristiku Strojového učení, včetně jeho zařazení do oblasti umělé inteligence, charakteristiku úloh učení s učitelem, metod používaných v procesu učení a přehled algoritmů pro řešení úloh učení s učitelem. Součástí této části je i seznámení s aplikací Jupyter notebook a běžně používanými technologiemi a postupy. Realizace metodiky testování vychází z teoretických poznatků. Nejdříve je popsán postup učení, zahrnující metodiku a poté předveden při učení několika algoritmů pro řešení vybrané úlohy, za použití dat dostupných na internetu.

3 Teoretická východiska

Tato kapitola obsahuje přiblížení pojmu umělá inteligence a seznámení s problematikou strojového učení, která je součástí oboru umělé inteligence a poskytuje přehled algoritmů strojového učení pod dohledem pro úlohy klasifikace a regrese. Dále obsahuje představení aplikace Jupyter notebook a technologií používaných při trénovaní algoritmů strojového učení v Jupyteru.

3.1 Umělá inteligence

Vytvoření ucelené definice pojmu Umělá inteligence (*Artificial Intelligence* – zkráceně AI) je poměrně komplikované. Jedním z prvních pokusů o jeho vymezení je *Turingův test*. Pokud umělá entita testem projde, říkáme, že entita je inteligentní.

Test spočívá ve schopnosti člověka rozlišit, zda komunikuje s dalším člověkem, či umělou entitou (AI). V testu vystupují 3 aktéři, každý v oddělené místnosti. Prvním aktérem je člověk v roli testujícího, který pokládá dotazy zbývajícím dvěma aktérům, kterými jsou testovaná entita (např. ve formě programu běžícího na počítači) a další člověk, který v tomto testu slouží jako reference: pokud z odpovědí na pokládané otázky nedokáže testující určit, který ze dvou aktérů je člověk, entita testem prošla. Jak ale ukázal Searle v experimentu čínské místnosti, tento test nedokáže rozlišit skutečné inteligentní chování a jeho simulaci. [1]

Searle v experimentu poukazuje na problém, že osoba aktéra může mít k dispozici přepis postupu, který vykonává umělá entita (v experimetu počítačový program) a interagovat s testujícím stejným způsobem, jako testovaná entita. Tím simuluje její chování. V experimentu uvádí program, který dokáže komunikovat v čínštině, čehož může Searle v roli aktéra dosáhnout také (za pomoci slovníku a přepisu programu), ačkoli čínštinu neovládá a tudíž ji ani entita nemusí ovládat, ale pouze tuto schopnost simulovat. [1]

Na základě tohoto experimentu lze umělou inteligenci dělit na tzv. *slabou* a silnou umělou inteligenci. Slabá, na rozdíl od silné AI, pozbývá schopnost skutečně chápat a pouze simuluje inteligentní chování. V praxi se setkáváme pouze se slabou umělou inteligencí. [1]

Existuje také nemálo pokusů o robustní definici pojmu umělé inteligence, které

však, podobně jako Turingův test, mají své slabiny. Některé z nich řeší definice Elaine Richové: „Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better“, kterou lze volně přeložit jako: *Umělá inteligence je studium možností jak umožnit počítačům vykonávat činnosti, ve kterých je člověk zatím lepší.* [2]

3.2 Strojové učení

Strojové učení je součást oboru AI, která se prolíná s různými matematickými oblastmi, zejména se statistikou a dobýváním znalostí (*data mining*). Jeho cílem je umožnit počítačům se samostatně učit a adaptovat své chování na základě dostupných dat tak, aby vedlo k přesnějšímu řešení dané úlohy. V některých oblastech je nezbytná schopnost učit se kontinuálně, např. filtrování nevyžádané emailové pošty, tzv. spamu. Existuje nespočet různých druhů spamů a neustále se objevují další, není tedy v lidských silách napsat program, který by veškerý spam dokázal odlišit od běžné komunikace. [3]

Algoritmy strojového učení jsou programy, které umožňují počítačům se naučit a řešit úlohy, jako je např. predikce. Cílem učení je zkonstruovat *model*, který pro předložený vstup vrací správný výstup. [4] Podle znalosti struktury vstupních dat a typu řešené úlohy lze strojové učení rozdělit na 4 techniky:

1. Učení s učitelem (*Supervised learning*)

   Algoritmus se učí z označených dat, kterým se říká *trénovací sada*. Technika se využívá pro řešení úloh *klasifikace* a *regrese*, kde pro daný vstup vrací algoritmus označení třídy nebo číselný výsledek, který má algoritmus na základě tohoto vstupu vypočítat. U označených dat jsou správné výsledky předem známé. Například u spamu jsou vstupními daty pro učení obsah emailových zpráv, který je označený buď jako spam, nebo jako běžná zpráva. Algoritmus strojového učení se z takových dat učí klasifikovat, tedy zařadit každou zprávu do jedné ze tříd „spam“ a „běžná zpráva“. Označené záznamy slouží i k testování algoritmu. [3]

2. Učení bez učitele (*Unsupervised learning*)

   Algoritmus se učí z neoznačených dat, výsledky tedy nejsou předem známé. Jedním z nejčastějších postupů je shluková analýza (*clustering*). Při ní se data

shlukují do skupin podle podobností, výsledné skupiny nemají předem známé charakteristiky a otestovat správnost takového rozdělení není prakticky možné, jelikož nemáme podklady podle kterých bychom mohli o jejich správnosti rozhodnout. Např. lze shlukovat zákazníky podle nákupních preferencí do skupin, na které lze cílit rozdílné marketingové kampaně. [3]

3. Kombinace učení s učitelem a bez učitele (*Semi-supervised learning*)
   Algoritmus se učí z dat, z nichž je malá část označená a převážná část neoznačená. Učení probíhá ve třech fázích. V první fázi se algoritmus učí z označených dat. Takto nastavený model ve druhé fázi označí zbývající neoznačenou část dat. Ve třetí fázi se všechna označená data spojí do jedné sady, ze které vzejde finální model řešení dané úlohy. [3]

4. Zpětnovazební učení (*Reinforcement learning*)
   Algoritmus se učí z výsledků svých akcí za běhu pomocí zpětné vazby. Algoritmus obdrží vstupní stav, na základě kterého provede akci. Po jejím provedení algoritmus obdrží zpětnou vazbu ve formě hodnocení a uloží si informace o dvojici stav-akce. Díky tomu může algoritmus optimalizovat své reakce na příchozí vstupy. [4]

3.3 Úlohy a algoritmy učení s učitelem

Algoritmy strojového učení, kterými se zabývá tato práce, slouží pro řešení jedné ze dvou úloh učení s učitelem, kterými jsou klasifikace a regrese, představené v následujících sekcích.

3.3.1 Klasifikace

Úloha klasifikace spočívá ve správném přiřazení třídy danému vstupu. Vstupním příkladům říkáme trénovací vzory (v anglické literatuře se lze setkat s pojmem *feature vectors*, „vektory vlastností“, přičemž se nemusí jednat doslova o vektory tak, jak jsou známy z lineární algebry). Trénovací sada se při klasifikaci skládá z uspořádaných dvojic $(x_i,\ y_i),\ i \in N$, kde x_i je i-tý trénovací vzor a y_i označení třídy, do které by měl být vzor zařazen. [5]

Při učení hledáme *klasifikátor*, který optimalizuje nějakou nákladovou funkci. Klasifikátor je procedura, která jako vstup přijímá množinu vlastností a jako výstup vrací označení třídy, kterou vstupu přiřadila. Testování přesnosti klasifikátoru na trénovacích datech není oproti testování na dosud neviděných datech podstatné a naopak může vést k přetrénování. Pro nalezení co nejpřesnějšího klasifikátoru je nutné, aby trénovací sada co nejlépe reprezentovala reálná data, se kterými bude klasifikátor pracovat, proto by sada měla být tzv. IID – *Independent and Identically Distributed*, nezávislá a pravděpodobnostně stejně rozdělená, jako sada všech reálných dat. [5]

Klasifikaci můžeme dělit podle počtu tříd, které se algoritmus učí rozlišovat. Při binární klasifikaci se vstupy zařazují do jedné ze dvou možných tříd. Vstupy jsou při trénování rozhodovacího algoritmu rozděleny na pozitivní a negativní. Jednu třídu vybereme jako pozitivní, příklady, které do ní patří, bude klasifikátor označovat výstupem $y = 1$ za pozitivní a ostatní $y = -1$ za negativní (patří do druhé třídy). Klasifikace s více než dvěma třídami se označuje termínem *multi-class classification*. Pro tento typ klasifikace se často používá metoda porovnávání jedné třídy vůči ostatním. Pro každou třídu je vytvořen model, který porovnává, zda vstup patří do zvolené třídy nebo do některé z ostatních. Jako výstup je zvolena třída, do které je vstup zařazen s nejvyšší mírou jistoty. [6]

3.3.2 Výkonnost klasifikátoru

Pro měření výkonnosti či přesnosti klasifikace modelu se používají různé ukazatele, z nichž některé se navzájem doplňují. Nejpřímočařeji lze měřit přesnost klasifikátoru procentuální úspěšností klasifikace trénovacích vzorů: *chyba* je procento špatně klasifikovaných vzorů, *přesnost* je procento vzorů klasifikovaných správně. Tyto ukazatele však obecně mají jen omezenou vypovídací hodnotu, protože nezohledňují různé pravděpodobnosti výskytu vzorů z různých tříd. [5]

Na základě pravděpodobnosti zařazení do tříd je založeno porovnávání se základnou. Základna je řešení, které vychází z logiky dané úlohy, nebo je náhodné a nevyžaduje algoritmus učení ani sadu označených dat. Obecně lze jako základnu použít chybu náhodného přiřazování tříd trénovacím vzorům, která je rovna $1 - \frac{1}{C}$ (C je počet tříd) a to za předpokladu, že se třídy vyskytují stejně často. Pokud se četnost výskytu různých tříd liší, stačí označit všechny vzory nejčastěji se vyskytující třídou, čímž

získáme základnu s nižší chybovostí než u chyby náhodného přiřazování. S takto zvolenou základnou můžeme porovnat, zda námi zvolený klasifikátor vykazuje menší chybu než naivní řešení. [5]

Kromě výše zmíněné chyby patří mezi ukazatele chybovosti *Tréninková chyba* – míra chybovosti algoritmu testovaném na stejné trénovací sadě, na které byl trénován – a *Testovací chyba* – míra chybovosti klasifikace dříve neviděných vzorů. [5]

3.3.3 Regrese

Úloha regrese spočívá v odhadu (predikci) hodnoty proměnné y na základě trénovacího vzoru x. Proměnnou y nepočítáme jako funkci trénovacího vzoru, trénovací sadu považujeme za soubor z náhodného rozdělení $\mathrm{P}(X,\ Y)$. Výstup je pak odhadem střední hodnoty $P(Y \mid \{X\ =\ x\})$. Můžeme tak sestavit model, který např. predikuje schopnost čtení u dětí na základě velikosti jejich chodidla, přestože na sobě tyto dvě veličiny nijak nezávisí (schopnost číst závisí – mimo jiné – na věku dítěte). Kromě predikce může regresní model sloužit k porovnávání datových trendů. [7]

3.4 Učení a testování modelu

Při Strojovém učení je nutné brát ohled na data, na kterých se model učí a na jakých je následně testován. Pokud jsou testovací data stejná jako trénovací sada, nelze vyloučit problém *přeučení* (*overfitting*) modelu, při kterém model vykazuje vysokou přesnost odhadu na datech pro učení, ale při zpracování dosud neviděných dat je jeho přesnost nízká. Proto se z trénovací sady typicky vyčlení část dat pro testování modelu na dosud neviděných datech. [4]

Pouhé rozdělení dat na trénovací a testovací však nutně nevede k výběru nejlepšího modelu, toho lze docílit metodou křížové validace (*cross-validation*), která v určitém počtu iterací rozděluje trénovací sadu na 2 části, trénovací a validační, na trénovacích datech je algoritmus trénován, na validačních pak testován. Rozdělení je pro každou iteraci náhodné a stejnoměrné, čímž se snižuje vliv konkrétního rozdělení datových sad na výsledný výkon modelu. Vylepšená varianta pro trénování klasifikace je tzv. rozvrstvená (stratified) křížová validace, při které jsou validační části vytvářeny tak, aby obsahovaly záznamy z každé třídy. Alternativně lze místo dělení náhodně vyčlenit jeden

záznam z trénovací sady (metoda *leave-one-out*). Výstupem této metody je průměrná testovací chyba či průměrná přesnost, jako nejlepší je vybrán model s nejmenší chybou, resp. nejvyšší přesností. V praxi se jako výchozí používá 10 validačních částí (iterací), menší počet připadá v úvahu pokud je trénovací sada příliš velká. [5, 6]

Výkon modelu ovlivňuje nastavení *hyperparametrů*, což jsou parametry algoritmu, které nedokáže odvodit z trénovacích dat, ale musí být nastaveny ručně před začátkem učení. Pro nalezení takového nastavení hyperparametrů, které vede k co nejlepšímu výkonu modelu lze použít *validační křivky*, které vizualizují průměrnou přesnost predikce trénovaného modelu na trénovací a validační části s různým nastavením vybraného hyperparametru při křížové validaci. Diagram validačních křivek obsahuje 2 křivky: průměrnou přesnost predikce na části trénovací a na části validační, kolem kterých jsou (např. barevně) vyznačeny oblasti, jejichž šířka odpovídá směrodatné odchylce přesnosti při jednotlivých volbách validační části. V místě, kde jdou křivky přes sebe, eliminuje nastavení hyperparametrů riziko přeučení. Pokud je křivka validační přesnosti pod křivkou trénovací, dochází k přeučení. V praxi se s opačnou variantou (validační přesnost nad trénovací) nesetkáváme. Tam, kde je přesnost modelu oproti jiným nastavením hyperparametru nízká u obou křivek, dochází k *nedoučení* (underfitting) modelu. [6]

I metoda křížové validace má svá úskalí a k výběru nejlepšího modelu nemusí stačit. V praxi typicky neznáme přesné parametry nejlepšího modelu a tak nevíme, jaký odhad je správný, přičemž může nastat i situace, že model, námi vybraný jako nejlepší, má na našich datech menší průměrnou chybovost než skutečně nejlepší model. Tento problém vyřešíme zvolením dostatečně velké sady trénovacích dat, kterou následně rozdělíme na část, která poslouží jako vstupní sada pro křížovou validaci a na část pro finální otestování vybraného modelu. Také se může stát, že více modelů bude mít podobnou průměrnou chybu. [5]

3.4.1 Algoritmy strojového učení

Následující sekce popisují algoritmy strojového učení, které patří mezi metody využívané při technice učení s učitelem. Na několika z nich je také prezentována testovací metodika.

3.4.1.1 Naivní bayesovský klasifikátor

Jednou z nejjednodušších metod klasifikace je Naivní bayesovský (někdy též Bayesův) klasifikátor, který klasifikuje na základě pravděpodobnosti $P(y \mid x)$ – trénovacímu vzoru x přiřadí takové označení třídy y jehož pravděpodobnost je nejvyšší. Výpočet pravděpodobnosti stojí na transformaci přes Bayesovu větu. Podle ní platí:

$$P(y \mid x) = \frac{P(x \mid y)P(y)}{P(x)} \tag{3.1}$$

Metoda dále předpokládá, že jednotlivé komponenty vzoru $x^{(j)}$ jsou na sobě nezávislé za podmínky přiřazení do stejné třídy $P(x^{(j)} \mid y)$ a pro pravděpodobnost vzoru x za podmínky y platí:

$$P(x \mid y) = (\prod_{j} P(x^{(j)} \mid y)P(y) \tag{3.2}$$

Dosazením za $P(x \mid y)$ a dalšími algebraickými úpravami dostaneme vzorec, který slouží k výpočtu pravděpodobností klasifikace x. V praxi je číselný rozsah pro výpočty omezen a tudíž může docházet ke ztrátě přesnosti – vypočtená pravděpodobnost je příliš malá pro rozsah námi použité datové reprezentace. Proto se používá logaritmovaná varianta vzorce:

$$(\sum_{j} log\ P(x^{(j)} \mid y)) + logP(y) \tag{3.3}$$

[5]

3.4.1.2 Nejbližších k sousedů

Metoda Nejbližších k sousedů klasifikuje trénovací vzory pomocí vzorů, které jsou k němu v nějakém smyslu nejblíže. Předpokládáme, že pokud jsou dva vzory x_1 a x_2 dostatečně blízko u sebe, jsou pravděpodobnosti jejich zařazení do stejné třídy podobné. Protože může nastat situace, že existuje více podobně vzdálených vzorů x_i, které patří do různých tříd s podobně vysokou pravděpodobností, rozhoduje klasifikátor na základě k vzorů, které jsou nejblíže ke vzoru, který je klasifikován. Pokud mezi nejbližšími převládají vzory z jediné třídy, jejichž počet je alespoň l, pak klasifikátor označí vstupní vzor touto třídou. V opačném případě jej označí jako neznámý. Při učení jsou k a l parametry, které se klasifikátoru nastaví ručně před učením. V praxi si lze většinou vystačit s $k = 3$. [5]

Problémem u této metody je volba výpočtu vzdálenosti mezi vzory. Pokud jsou komponenty trénovacích vzorů rozdílné povahy (např. barva a úhel), je nutné při jejich převodu/přepočtu udržet jejich vypovídací schopnost a váhu. Toho lze docílit dvěma způsoby: škálováním a tzv. bělením. U škálování je nutné zachovat stejný rozptyl, aby nedošlo k posílení vlivu některých vlastností na úkor jiných. Bělení znamená transformovat komponenty tak, aby byly navzájem nezávislé. Pokud je dimenze prostoru vzorů příliš vysoká, je obtížné takovou transformaci nalézt. [5]

3.4.1.3 SVM

Support vector machines jsou lineární klasifikátory, které při učení hledají optimální lineární oddělovač tříd. Trénovací sada je v tomto modelu pokládána za soubor vektorů z nějakého lineárního prostoru. Lineárním oddělovačem může být přímka, rovina apod. (záleží na dimenzi prostoru) pocházející ze stejného prostoru. Klasifikace pak spočívá ve vyhodnocení, na které straně od oddělovače se daný vektor x nachází. Algoritmy SVM jsou schopny řešit i úlohy klasifikace, ve kterých jsou třídy odděleny nelineárním oddělovačem (např. kružnicí). Řešení spočívá v takovém mapování vstupních vektorů na vektory z jiného lineárního prostoru, ve kterém je lze oddělit lineárně. [8]

SVM používá mapovací funkci $F(x)$, která jednotlivým vstupům přiřazuje vektory z nového lineárního prostoru. Obecně lze při mapování s dostatečným počtem dimenzí vždy nalézt lineární oddělovač (nadrovinu): N bodů lze vždy lineárně oddělit v lineárním prostoru s takovou dimenzí d, pro kterou platí $d \geq N - 1$. [8]

V d-rozměrném prostoru je lineární oddělovač definován rovnicí s d parametry. Pokud se d blíží N, hrozí přeučení modelu (viz sekce 3.4). Pro zamezení tohoto problému hledá algoritmus SVM *optimální* lineární oddělovač, který má největší minimální vzdálenost od vektorů oddělovaných tříd. Pro nalezení lineárního oddělovače stačí pouze vektory, nacházející se v hraniční oblasti kolem oddělovače a které jsou od něj stejně vzdálené. Těmto vektorům se říká *podpůrné vektory* a od nich je odvozen i název celého modelu. [8, 9]

Samotný lineární oddělovač SVM hledá v novém lineárním prostoru daném mapováním vektorů, kde je dán rovnicí s nelineární, tvz. jádrovou funkcí (*kernel function*), která nahrazuje výpočty mapovací funkce jednotlivých vektorů takovým skalárním součinem dvojic vektorů, který odpovídá danému prostoru. Proto jsou algoritmy SVM

známy též jako jádrové metody (kernel methods). [8, 9]

Parametry oddělovače volíme tak, aby minimalizovaly nákladovou funkci, která zajišťuje, aby každý trénovací vzor byl na správné straně oddělovače, nebo alespoň v jeho blízkosti a zároveň zajišťuje penalizaci špatně klasifikovaných vzorů. Sílu penalizace určuje její váha (číselná konstanta), jejíž optimální nastavení je nutné provést ručně (v implementacích bývá reprezentována parametrem alpha, nebo parametrem $C = 1/\text{alpha}$). Původní SVM řeší úlohu binární klasifikace, ale existují i varianty pro klasifikaci s více třídami. Existuje také rozšíření pro řešení regresních úloh. [9, 10, 11]

3.4.1.4 Rozhodovací stromy a Random Forests

Přímočarým algoritmem řešení klasifikace je *Rozhodovací strom*. Každý vrchol (resp. uzel) stromu představuje test nějaké vlastnosti vstupního trénovacího vzoru s výjimkou listů, které představují popisky tříd. Při klasifikaci prochází vzor stromem od kořene, ze kterého se přejde do dalšího vrcholu na základě výsledku testu. Takto postupuje vzor až do listu s popiskem třídy, který je mu přiřazen. Typicky se používají binární stromy, typy stromů složených z vrcholů s více potomky pro tento model nic principiálně nemění ani nepřináší zásadní výhody. [10]

Existuje mnoho algoritmů pro sestavení rozhodovacího stromu. V praxi se používají algoritmy se značným množstvím náhody, což znamená, že při každém trénování sestavíme jiný strom. Kvůli této náhodnosti a faktu, že při tréninku nehledáme ten nejlepší, je sestavený rozhodovací strom sám o sobě slabým klasifikátorem, avšak efektivně sestavitelným. Proto se z více rozhodovacích stromů sestavuje tzv. Rozhodovací les (*Decision forest*). Pro klasifikaci se používá jedna ze dvou strategií:

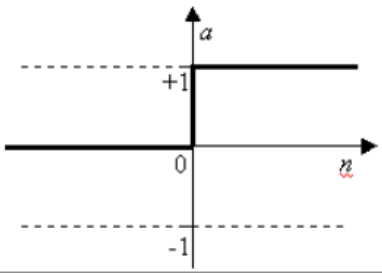
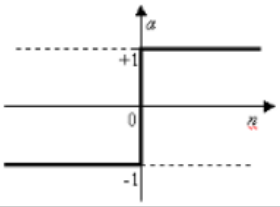
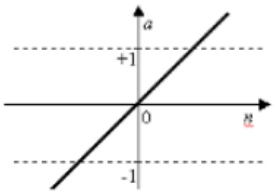
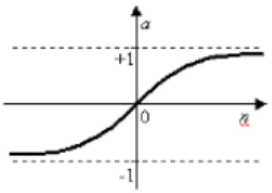
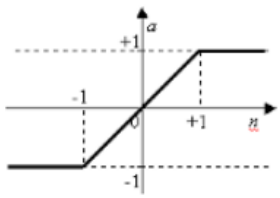
1. Vzor projde všemi stromy, při vstupu do listu se přičte jeden hlas pro daný popisek. Ten s nejvíce hlasy se vybere jako výstup.

2. Vzor projde všemi stromy, při vstupu do listu se zapíše n hlasů pro daný popisek, kde n je počet výskytů popisku v trénovací sadě. Popisek s nejvíce hlasy se vybere jako výstup.

[10]

3.4.1.5 Neuronové sítě

Model neuronových sítí je odvozen od neuronů tvořících lidský mozek. Většina Neuronových sítí se skládá z jednotek – *neuronů*, z nichž každý realizuje výpočet na základě vstupů $(x_1, x_2, ..., x_i)$ a vah spojů $(w_1, w_2, ..., w_i)$. Na výsledek i-tého neuronu x'_i je poté aplikována aktivační funkce $x_i = f(x'_i)$. [9, 12]

	Funkce jednotkového skoku (*hardlim*). Standardní výstup nenabuzeného neuronu je hodnota *0*, po nabuzení *-1*.
	Aktivační funkce signum (*hardlims*). Y(x) = sign(x) = 1 pro x > 0 a Y(x) = sign(x) = -1 pro x < 0.
	Lineární aktivační funkce (*purelin*). Y(x) = x
	Aktivační funkce hyperbolický tangent (*tansig*). Využití u Elmanových neuronových sítí.
	Sigmoidální aktivační funkce (*logsig*). $Y(x) = 1/1+e^{-x}$ Využití zejména u vícevrstvých perceptronových sítí.
	Saturační přenosová funkce (*satlins*). Využívána pro pro omezení velikosti vstupního signálu.

Obrázek 3.1: Aktivační funkce. Zdroj: [13]

Existuje mnoho aktivačních funkcí, nejjednodušší je funkce identitní $f(x'_i) = x'_i$.

Její použití ale znamená neomezenou velikost výsledků, které tak mohou růst nade všechny meze. Problém lze odstranit nahrazením identity prahovou funkcí

$$H_\Theta(x_i') = \begin{cases} 0, & pokud\ x_i' < \Theta \\ 1, & \text{jinak} \end{cases} \tag{3.4}$$

kde Θ je prahová konstanta. Další běžně používané aktivační funkce ukazuje obrázek 3.1. Během let byly vyzkoušeny různé nelineární funkce, v současnosti je za nejlepší považována jednotka ReLU (*rectified linear unit* – usměrněná lineární jednotka) realizující funkci $F(u) = max(0,\ u)$, kde u je vstup daného neuronu. [9] Nelinearita aktivační funkce je klíčová pro klasifikaci, naopak pro regresi se ve výstupní vrstvě sítě využívají funkce lineární. [12]

Jedním z principů učení Neuronových sítí je zesílení synapse – propojení mezi dvěma neurony. U biologických sítí platí, že synapse je tím silnější čím více elektrických signálů musí přenést. Tento princip je do modelu Neuronových sítí zaveden *Hebbovým pravidlem*:

Definice 1 *Pokud existuje propojení w_{ij} neuronů i, j a mezi nimi jsou opakovaně posílány takové signály, které vedou k simultánní aktivitě obou neuronů i, j, pak je w_{ij} (váha) zesílena. Výpočtem zesílení je $\Delta w_{ij} = \eta x_i x_j$, kde η je velikost jednotlivých kroků při učení.*

Existují modifikace tohoto pravidla, které vyhovují různým typům sítí. Např. pokud je síť tvořena neurony, jejichž výstupní hodnoty pocházejí z intervalu $(0, 1)$, mohou váhy časem pouze růst. Aby mohly neurony také oslabovat své spoje (nebo „odumřít“ – jinak by síť měla jen omezené možnosti adaptace) lze zavést konstantní faktor 0,99, kterým se v každém kroku váhy násobí. [9]

Další typy neuronových sítí:

Vícevrstvá perceptronová síť

Skládá se z typu neuronů zvaných *perceptron*, uspořádaných do vrstev. Perceptron realizuje výpočet váženého součtu vstupů

$$x_i' = \sum_{j=1}^{n} w_{ij} * x_j - \theta \tag{3.5}$$

kde x_j je j-tý vstupní vektor (data, pro která se realizuje predikce), w_{ij} je váha j-tého spoje i-tého neuronu a θ je prahová konstanta. Jako jeho aktivační funkce se používá sigmoida (viz 3.1). Všechny neurony v dané vrstvě jsou napojeny na každý neuron vrstvy následující. Stejně jsou přivedeny vstupy sítě na neurony první vrstvy. Poslední vrstva se nazývá výstupní vrstva (*output layer*), ostatní jsou skryté vrstvy (*hidden layers*). Počet vrstev a neuronů ve vrstvě jsou parametry sítě, které se nastavují podle konkrétní úlohy, kterou má síť řešit. Učení probíhá nastavením vah tak, aby chyba sítě (definovaná jako součet chyb vektorů z trénovací sady) byla minimální. Chyba i-tého trénovacího vzoru je definována jako

$$E_i \;=\; \frac{1}{2}\sum_{j=1}(y_j - d_{ij}) \tag{3.6}$$

kde y_j je výstup j-tého neuronu výstupní vrstvy a d_{ij} je j-tý element (j-tá komponenta) výsledku i-tého trénovacího vzoru. [14]

Učící vektorová klasifikace

Síť typu LVQ (*Learning Vector Quantization*) je *Kohonenova síť* modifikovaná pro učení s učitelem (klasifikaci). Původní Kohonenova síť aproximuje pravděpodobnostní rozdělení v trénovacích datech. Skládá se z jediné (*kompetiční*) vrstvy neuronů, které jsou propojeny se všemi vstupy a pomocí postranních vazeb i mezi sebou navzájem do zvolené topologické mřížky. Počet neuronů je volitelným parametrem sítě, v praxi se pohybuje v řádu desítek až stovek neuronů. Aktivní je vždy jen jeden neuron a to ten s minimální vzdáleností

$$d_i \;=\; \sum_{j=1}^{n}(x_i \;-\; w_{ij})^2 \tag{3.7}$$

kde n je počet neuronů v síti, x_i je i-tý element vstupního vzoru a w_{ij} je váha. Před učením jsou váhy nastaveny náhodně.

Klasická Kohonenova síť při učení aproximuje pravděpodobnostní rozdělení trénovacích vzorů pomocí rozmisťování neuronů v topologické mřížce. Síti jsou postupně předkládány náhodně vybrané trénovací vzory, které jsou porovnávány s vektory uloženými v neuronech. Jakmile je nalezen neuron, který je vzoru nejblíže, je upravena jeho váha spolu s váhami všech neuronů v jeho okolí. Okolí neuronu je definováno vzhledem k topologické vzdálenosti neuronů, na začátku

může obsahovat až všechny neurony sítě, jeho velikost se během učení postupně zmenšuje. Proces učení se opakuje tak dlouho, dokud nedosáhneme předem zvoleného počtu kroků.

Při učení LVQ nejprve proběhne fáze učení jako u klasické Kohonenovy sítě, čímž dosáhneme hrubého nastavení sítě. Poté jsou síti postupně předkládány trénovací vzory i s informací o jim přiřazených třídách. V příslušném nejbližším neuronu jsou pak uloženy četnosti výskytu tříd v dosud předložených vzorech. Po předložení všech vzorů se u každého vektoru ponechá třída s nejvyšší četností. Při klasifikaci neznámého vzoru je výstupem sítě třída přiřazená nejbližšímu váhovému neuronu (resp. v něm uloženém vektoru). Existují různé verze LVQ, které se od sebe liší způsobem hledání nejlepší hranice mezi třídami. [14]

3.5 Jupyter Notebook

Jupyter Notebook je aplikace typu klient-server, kde server je program Jupyteru a klient je webový prohlížeč, ve kterém se uživateli zobrazí webová stránka Notebooku. Po spuštění je uživatel automaticky přesměrován na úvodní stránku klientské aplikace, na které lze vytvářet nové notebooky nebo spouštět již existující (viz obrázek 3.2). Pod záložkou *File* se zobrazuje obsah pracovního adresáře, ze kterého byl Jupyter spuštěn, jednotlivé notebooky se spouštějí kliknutím na příslušný soubor s koncovkou *.ipynb*. Následně se notebook otevře v prohlížeči jako samostatná webová stránka. Na úvodní stránce jsou běžící notebooky indikovány zelenou ikonou vedle názvu souboru notebooku. Seznam běžících notebooků je pod záložkou *Running*, odtud je lze také vypínat přes tlačítko *Shutdown* (zavřením okna prohlížeče se program notebooku neukončí).

Obrázek 3.2: domovská stránka Jupyteru

V notebooku lze interaktivně pracovat jak s kódem tak s formátovaným textem, umožňuje vizualizovat a analyzovat výstupy aj. Každý notebook se skládá z buňek, do

kterých lze vkládat spustitelný kód některého z podporovaných programovacích jazyků, příkazy podporovaných interpretrů příkazové řádky, nebo text formátovaný pomocí jazyka *Markdown* [6]. Formátování je specifikováno pomocí značek (*tags*) Markdownu, které jsou při zpracování převedeny do formátu HTML, který interpertuje webový prohlížeč. [15] Příklad ukazuje obrázek 3.3, na kterém je zachycena buňka s textem a značkami pro nadpis, odsazení a blok kódu jazyka Python, obrázek 3.4 zachycuje stejnou buňku po interpretaci značek.

````
### Nadpis 3. řádu
- odrážka <!-- komentář - text, který je při interpretaci ignorován -->

<!-- prázdná řádka se interpretuje jako nový řádek -->
```python
print('zvýraznění bloku kódu programovacího jazyka python') # komentář v Pythonu
```
````

Obrázek 3.3: Neformátovaný text se značkami Markdownu v buňce Jupyter notebooku

Nadpis 3. řádu

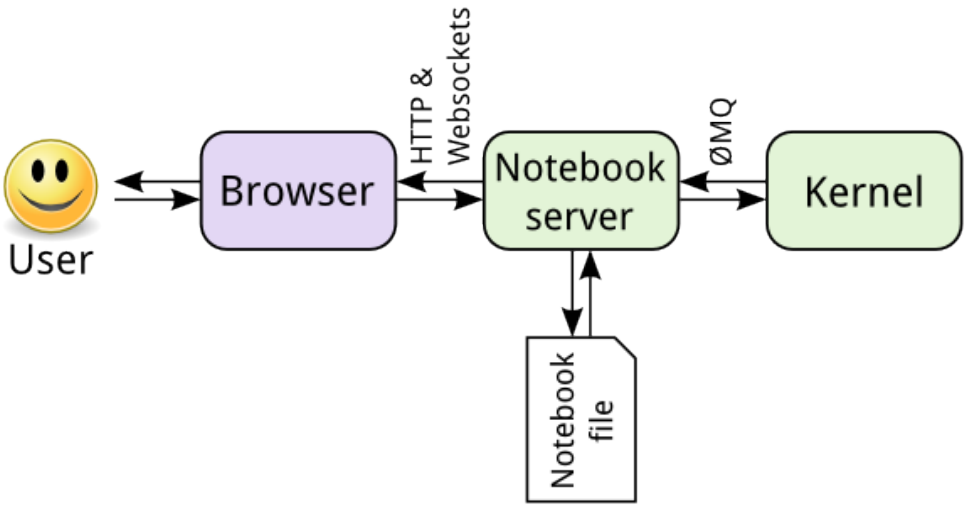
- odrážka

```
print('zvýraznění bloku kódu programovacího jazyka python') # komentář v Pythonu
```

Obrázek 3.4: Formátovaný text buňky Jupyter notebooku

Buňky lze spouštět jednotlivě (tj. spustit kód nebo zformátovat text v buňce), po skupinách definovaných uživatelem, nebo všechny buňky v rámci jednoho notebooku. Výsledek je zobrazen buď na místě obsahu buňky (v případě formátovaného textu), nebo pod ní (v případě výstupu spustitelného kódu). Notebook má vždy přiřazený kernel, který interpretuje kód daného programovacího jazyka nebo příkazové řádky zadaný v buňkách, mezi podporované programovací jazyky patří Python, Javascript, R, Ruby a další. Z interpretrů je podporován např. Bash (interpret linuxové příkazové řádky). Zmíněné technologie jsou podporované nativně, tzn. není potřeba stahovat žádné přídavné kernely nebo jiné moduly. [6]

Nativní podpora je zajištěna prostřednictvím vestavěných příkazů Jupyteru, tzv. magických funkcí (*magic functions*), které se uvozují symbolem %. Notebook lze trans-



Obrázek 3.5: komponenty Jupyteru. Zdroj: [16]

formovat do skriptu, který obsahuje veškerý spustitelný kód, který se nacházel v buňkách notebooku a textový obsah buňek jako komentáře. [6]

3.5.1 Architektura

Základním stavebním kamenem Jupyteru je technologie *IPython*, která zajišťuje zpracování kódu psaného v Pythonu, zpracování magických funkcí a komunikaci s dalšími součástmi Jupyteru (notebooky, kernely atd.). IPython běží jako samostatný proces (IPython kernel), komunikace mezi ním a dalšími součástmi Jupyteru probíhá prostřednictvím JSON zpráv – JSON je formát pro textovou reprezentaci dat odvozený od syntaxe objektů v programovacím jazyce *Javascript*. Samotnou komunikaci zajišťuje knihovna *ZeroMQ*. [16]

Komunikaci mezi IPython kernelem a notebooky ukazuje obrázek 3.5. Kernel běží v samostatném procesu a s notebooky nekomunikuje přímo. Mezi nimi stojí Notebook server, který funguje jako prostředník odstiňující obě strany, komunikuje s webovým rozhraním notebooků (webovým prohlížečem) přes protokol HTTP a s pomocí technologie WebSockets, s kernelem komunikuje přes ZeroMQ. Kromě komunikace zajišťuje ukládání obsahu notebooků ve formě JSON souborů s koncovkou *.ipynb* a načítání jejich obsahu do aplikace. Díky odstínění notebooků od kernelu lze notebooky psát a ukládat i při absenci kernelu, který by interpretoval jejich obsah. [16]

3.6 Strojové učení v Jupyteru

Trénování a testování algoritmů strojového učení v Jupyteru je v této práci realizované v programovacím jazyce Python, pro který existuje řada knihoven podporujících práci s daty a výzkum v oblasti umělé inteligence. Jednotlivé algoritmy není třeba implementovat, některé jsou součástí takovýchto knihoven, nebo existují v samostatném modulu. Tím se celý proces učení zjednodušuje a uživatel se může soustředit na samotný proces učení, zejména přípravu trénovacích dat a analýzu výsledků.

3.6.1 Manipulace s daty

Při strojovém učení hrají velkou roli trénovací sady dat, jelikož jejich složení a rozsah určují výsledné rozhodování algoritmu. V Pythonu lze datové sady reprezentovat pomocí *datasetů*.

Dataset je kolekce dat, která může odpovídat tabulce, kde se na řádcích vyskytují konkrétní hodnoty jednotlivých proměnných – ty představují sloupce tabulky. Dataset může být reprezentován pomocí formátu *CSV – comma-separated variable*. Data v tomto formátu lze do notebooku nahrát přes Python pomocí knihovny *pandas*. [6] Nahrát lze soubor z lokálního úložiště nebo z internetu pomocí funkce *read_csv*, která jako argument přijímá cestu k uloženému souboru, resp. URL adresu, ze které lze soubor stáhnout. [17]

Dataset, nahraný výše zmíněným způsobem, je reprezentován třídou *DataFrame* (z knihovny pandas). Ta reprezentuje data jako 2D pole potenciálně heterogenních dat (v různých sloupcích a řádcích mohou být data různých datových typů), ke kterému lze přistupovat jako k tabulce. Přístup k prvkům je možný pomocí indexů (číselné hodnoty vyjadřující pořadí řádku a sloupce, ve kterém se prvek nachází, začínající v nule) nebo pomocí názvů sloupců. [18]

3.6.2 Zpracování dat

Pro zajištění co nejlepších výsledků učení procházejí trénovací sady zpracováním (*preprocessing*), při kterém se ověřuje homogenita dat (všechny hodnoty daného sloupce mají stejný datový typ), úplnost dat (kde a kolik je chybějících položek) apod. Řešení chybějících údajů většinou nespočívá v odstranění celého řádku z datasetu, jelikož

bychom tím mohli přijít o cenná data, ovlivňující výsledný algoritmus. Místo toho se data nahrazují pomocí různých technik, např. průměrem hodnot daného sloupce. Odstranit celý sloupec má smysl tehdy, když obsahuje málo dat. [6]

Součástí zpracování je i analýza dat, ve které pomocí statistických ukazatelů a grafických postupů analyzujeme tréninkovou sadu a jednotlivé proměnné. *Korelační koeficient* měří sílu a směr lineární závislosti mezi dvěma proměnnými (ve strojovém učení mezi atributy trénovacích vzorů). Koeficient nabývá hodnot z intervalu $< -1; 1 >$, záporná hodnota značí nepřímou (zvyšující se hodnoty jedné proměnné znamenají snižující se hodnoty druhé), kladná přímou lineární závislost proměnných. Korelační koeficienty mohou být zobrazeny v korelační matici. [2] Na diagonále matice se nacházejí pouze hodnoty 1 (každá proměnná „koreluje“ sama se sebou), nad i pod diagonálou se nacházejí stejné hodnoty koeficientů, pouze jejich pořadí je zrcadlově obrácené (způsobené prohozeným pořadí proměnných). Pro snažší určení silně korelovaných proměnných lze využít *teplotní mapu*, která sílu závislosti vizualizuje pomocí barevných polí. Pozice pole v mapě odpovídá pozici koeficientu v matici [6].

Atributy, které jsou na sobě silně závislé mohou do modelu přinášet redundantní, resp. zbytečnou informaci, přičemž odstranění takovýchto atributů vede ke snížení složitosti úlohy, čímž také vede ke snížení časové a paměťové složitosti. Proces odstranění takovýchto atributů se nazývá *výběr vlastností* (feature selection). [3]

4 Realizace

Tato kapitola začíná popisem postupu trénování algoritmů strojového učení včetně metodiky jejich testování. Poté následuje uvedení platformy a technologií použitých pro předvedení postupu při trénování algoritmů pro řešení konkrétního klasifikačního problému. Kapitolu zakončuje analýza výsledků testování a výběru algoritmu pro řešení úlohy.

4.1 Přehled postupu učení

Celý postup učení se skládá z následujících kroků v pořadí, ve kterém jsou uvedeny. Každý krok je podrobně popsán v samostatné sekci, následující za přehledem:

1. **Příprava datové sady**
2. **Průzkumová analýza dat**
3. **Vytvoření trénovací sady**
4. **Výběr testovací základny**
5. **Trénování algoritmů**
6. **Otestování algoritmů a výběr nejlepšího modelu**

4.2 Příprava datové sady

Standardizace dat a oprava či vyřazení chybných trénovacích vzorů. Nejprve provedeme analýzu datových typů všech proměnných ve všech vzorech. Poté analyzujeme počet chybějících hodnot u každé proměnné a z trénovací sady vyřadíme ty, u kterých jich je příliš mnoho. Pokud v sadě zbývají vzory s chybějícími hodnotami, doplníme je na základě průzkumové analýzy v následujícím kroku. Pro každou ze zbývajících proměnných určíme jeden datový typ, na který budou její hodnoty převedeny a provedeme převod.

4.3 Průzkumová analýza dat

Analyzujeme rozdělení proměnných (atributů vzorů / sloupců sady) podle predikované proměnné (názvu třídy při klasifikaci nebo číselné hodnoty při regresi) a podle jednotlivých proměnných a změříme sílu korelací mezi proměnnými.

4.4 Vytvoření trénovací sady

Na základě výsledků průzkumové analýzy odstraníme z datové sady proměnné, které jsou silně závislé na jiných proměnných. Z každé dvojice na sobě silně závislých proměnných vyřadíme z trénovací sady tu, jejíž korelace s predikovanou proměnnou je slabší. Pokud se v proměnných stále nacházejí chybějící hodnoty, nahradíme je buď průměrnou hodnotou proměnné, nebo průměrem ze vzorů seskupených podle hodnot jiné proměnné, která s danou proměnnou koreluje. Výsledkem uvedených úprav je trénovací sada, se kterou se pracuje v dalších krocích.

4.5 Výběr testovací základny

Na základě trénovací sady vybereme naivní řešení s co nejvyšší přesností jako testovací základnu. Ta poslouží jako spodní hranice přesnosti a její překročení je jednou z nutných podmínek pro výběr nejlepšího modelu.

4.6 Trénování algoritmů

Trénink probíhá pomocí křížové validace, při které natrénujeme a otestujeme vybrané algoritmy s výchozím nastavením hyperparametrů, daných jejich implementací. Počet částí standardně stanovíme na 10, pokud není trénovací sada příliš velká nebo v případě klasifikace nemáme dost dat pro validaci všech tříd, pak je na zvážení snížení počtu částí.

Následně je každý algoritmus samostatně trénován. Srovnáme výkonnost modelu na různých kombinacích trénovací a validační části. Určíme hodnoty hyperparametrů algoritmů na základě validačních křivek tak, aby riziko přeučení bylo minimální se zachováním co největší přesnosti predikce. Diagramy křivek budou zobrazeny s osou přesnosti začínající na hodnotě dané testovací základnou. V dalších krocích pracujeme

pouze s hyperparametry, se kterými dosahují modely co nejvyšší průměrné přesnosti při co nejmenším riziku přeučení.

4.7 Otestování algoritmů a výběr nejlepšího modelu

Jednotlivé algoritmy s nastavením hyperparametrů z křížové validace znovu natrénujeme a průměrnou přesnost výsledných modelů porovnáme s přesností testovací základny i modelů trénovaných s výchozím nastavením. Modely, které nedosáhly lepší přesnosti než testovací základna jsou vyřazeny. Ze zbývajících vybereme jako nejlepší model ten, který dosáhl nejvyšší přesnosti.

4.8 Platforma a technologie

Celý postup učení byl realizován v aplikaci Jupyter notebook, verze 6.0.3, nainstalované na přenosném počítači s operačním systémem Windows 10 od společnosti Microsoft. Aplikace byla nainstalována a provozována v distribuci Anaconda programovacího jazyka Python. Implementace celého procesu učení, včetně manipulace s daty a prezentace výsledků učení, je realizována v programovacím jazyce Python. Dílčí části implementace jsou převzaté z [6]. V implementaci byly použity tyto knihovny (uvedené s popisem jejich využití):

scikit learn

Implementace algoritmů strojového učení, rozdělení trénovací sady na trénovací/testovací část, trénink křížovou validací, měření přesnosti algoritmů.

pandas

Manipulace s daty

seaborn

Vizualizace dat pomocí různých typů diagramů

matplotlib

Knihovna implementující vizualizace (využívaná knihovnou seaborn), sama umožňuje vytváření diagramů, nastavení jejich vzhledu, vykreslování více diagramů najednou apod.

numpy

Pomocné výpočty a funkce

4.9 Výběr řešeného problému a algoritmů

Pro předvedení strojového učení v aplikaci Jupyter notebook byl zvolen problém klasifikace druhu skla na základě jeho složení a indexu lomu. Data pocházejí z datové sady Glass Identification Data Set (dostupná z repozitáře UCI [19] zde: `https://archive.ics.uci.edu/ml/datasets/Glass+Identification`), která byla vytvořena pro účely výzkumu motivovaném odhalováním kriminální činnosti – kus skla nalezený na místě činu může být důkazním materiálem, pokud je správně identifikován. Datová sada obsahuje celkem 214 záznamů. Každý záznam se skládá z 10 sloupců – 9 proměnných a popisku třídy (druhu skla), do které záznam patří. Třídy čítají 7 druhů skla. Pro trénink byly zvoleny algoritmy SVM, Nejbližších k sousedů, Rozhodovací strom a Random Forest.

4.10 Postup učení

Veškeré zde prezentované části zdrojového kódu, včetně výstupů, jsou součástí souboru notebooku *GlassTypesLearning.ipynb* v elektronické příloze práce.

Prvním krokem je stažení datové sady a seznámení se s jejím obsahem. Samotná sada neobsahuje názvy sloupců (proměnných), ty je třeba doplnit, sloupce jsou popsány na stránce zdroje datové sady. Proměnnými jsou:

1. RI: index lomu
2. Na: sodík (jednotka: procento hmotnosti skla odpovídajícího oxidu, stejně je tomu i u atributů 4 až 10)
3. Mg: hořčík
4. Al: měď
5. Si: křemík
6. K: draslík
7. Ca: vápník
8. Ba: Baryum
9. Fe: železo

10. typ skla - možné hodnoty:

    1 – plavené sklo z okna budovy

    2 – neplavené sklo z okna budovy

    3 – plavené sklo z vozidla

    4 – neplavené sklo z vozidla

    5 – skleněný obal

    6 – skleněné nádobí

    7 – sklo ze světlometu

Obrázek 4.1 ukazuje zdrojový kód stažení datové sady a ověření, zda bylo úspěšné vypsáním názvů jejích sloupců, čímž je zároveň ověřeno i jejich doplnění do sady z pole *features*.

Nahrání datové sady z internetového zdroje

```
features = ["RI", "Na", "Mg", "Al", "Si", "K", "Ca", "Ba", "Fe", "label"]
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data',
                   sep=',', header=None, names=features)
```

Zobrazení nahraných sloupců

```
data.columns
```

```
Index(['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe', 'label'], dtype='object')
```

Obrázek 4.1: Stažení datové sady a jeho ověření

4.10.1 Příprava trénovací sady

Sada neobsahuje prázdné ani poškozené položky, reprezentované hodnotou NaN (obrázek 4.2). Všechny sloupce obsahují homogenní data (stejného datového typu), jak ukazuje obrázek 4.3). Z výstupu na obrázku je vidět, že sloupce obsahují data číselných typů (float64 pro reálná čísla, int64 pro celá), tudíž není třeba provádět konverze. Obrázek 4.4 ukazuje detekci a odstranění duplicitních záznamů. Výstup funkce duplicates ukazuje, že sada obsahuje 2 shodné záznamy, tedy 1 duplicitní, který je odstraněn funkcí drop_duplicates. Nastavením parametru *inplace* na hodnotu *True* je

zvoleno provedení změny přímo v objektu *data* (jinak by funkce vytvořila nový DataFrame bez odstraněných záznamů a původní by zůstal beze změny).

Detekce chybějících a vadných údajů

Prázdné hodnoty

```
data.isnull().sum()
```

```
RI       0
Na       0
Mg       0
Al       0
Si       0
K        0
Ca       0
Ba       0
Fe       0
label    0
dtype: int64
```

Obrázek 4.2: Detekce prázdných hodnot

Datové typy ve sloupcích

```
different_types_found = False

for column in list(data.columns):
    types = {}
    type_occurences = dict()
    for value in data[column]:
        _type = type(value)
        occurences = type_occurences.get(_type, 0)
        type_occurences[_type] = occurences + 1
    if len(type_occurences) > 1:
        different_types_found = True
        print('Ve sloupci ', column, ' byly nalezeny tyto datové typy:\n', type_occurences)
        print('\n')

if not different_types_found:
    print('Každý sloupec obsahuje data jediného datového typu (viz níže).')
    print(data.dtypes)
```

```
Každý sloupec obsahuje data jediného datového typu (viz níže).
RI       float64
Na       float64
Mg       float64
Al       float64
Si       float64
K        float64
Ca       float64
Ba       float64
Fe       float64
label      int64
dtype: object
```

Obrázek 4.3: Detekce datových typů ve sloupcích

```
# Detekce duplicit
data[data.duplicated(keep=False)]
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	label
39	1.52213	14.21	3.82	0.47	71.77	0.11	9.57	0.0	0.0	1
40	1.52213	14.21	3.82	0.47	71.77	0.11	9.57	0.0	0.0	1

```
# Odstranění duplicit
data.drop_duplicates(inplace=True)
```

```
# Ověření, že sada již duplicity neobsahuje
data[data.duplicated(keep=False)]
```

RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	label

Obrázek 4.4: Detekce a odstranění duplicitních záznamů

4.10.2 Průzkumová analýza dat

Analýza datové sady začíná popisem charakteristik jednotlivých proměnných a analýzou tvaru jejich rozdělení. Následovně jsou analyzovány korelace mezi proměnnými a rozdělení tříd napříč záznamy.

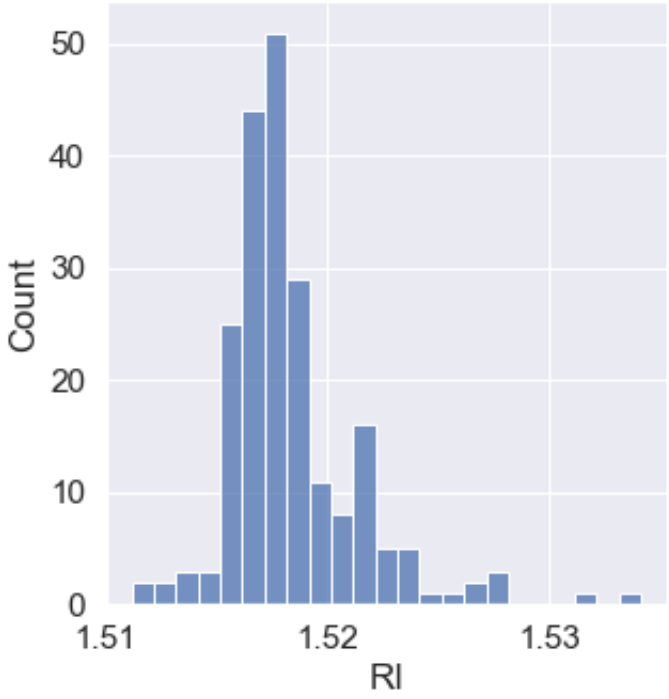
4.10.2.1 Analýza jednotlivých proměnných

Všechny proměnné trénovací sady (mimo popisky tříd) jsou popsány ve výstupu 4.5 pomocí základních statistických charakteristik (těm předchází počet hodnot – sloupec *count* – dané proměnné, kterých je po odstranění duplicit 213): průměru, směrodatné odchylky, minimální hodnoty, 25., 50. a 75. percentilu a maximální hodnoty proměnné. Následující analýzy jednotlivých proměnných mohou obsahovat zaokrouhlené hodnoty, pro přesné viz zmíněný výstup. U každé proměnné je vyšetřena normalita rozdělení pomocí histogramu. Pokud její rozdělení neodpovídá normálnímu, je nadále analyzován výskyt.

	count	mean	std	min	25%	50%	75%	max
RI	213.0	1.518348	0.003033	1.51115	1.51652	1.51768	1.51915	1.53393
Na	213.0	13.404085	0.816662	10.73000	12.90000	13.30000	13.81000	17.38000
Mg	213.0	2.679202	1.443691	0.00000	2.09000	3.48000	3.60000	4.49000
Al	213.0	1.449484	0.495925	0.29000	1.19000	1.36000	1.63000	3.50000
Si	213.0	72.655070	0.773998	69.81000	72.28000	72.79000	73.09000	75.41000
K	213.0	0.498873	0.653185	0.00000	0.13000	0.56000	0.61000	6.21000
Ca	213.0	8.954085	1.425882	5.43000	8.24000	8.60000	9.15000	16.19000
Ba	213.0	0.175869	0.498245	0.00000	0.00000	0.00000	0.00000	3.15000
Fe	213.0	0.057277	0.097589	0.00000	0.00000	0.00000	0.10000	0.51000

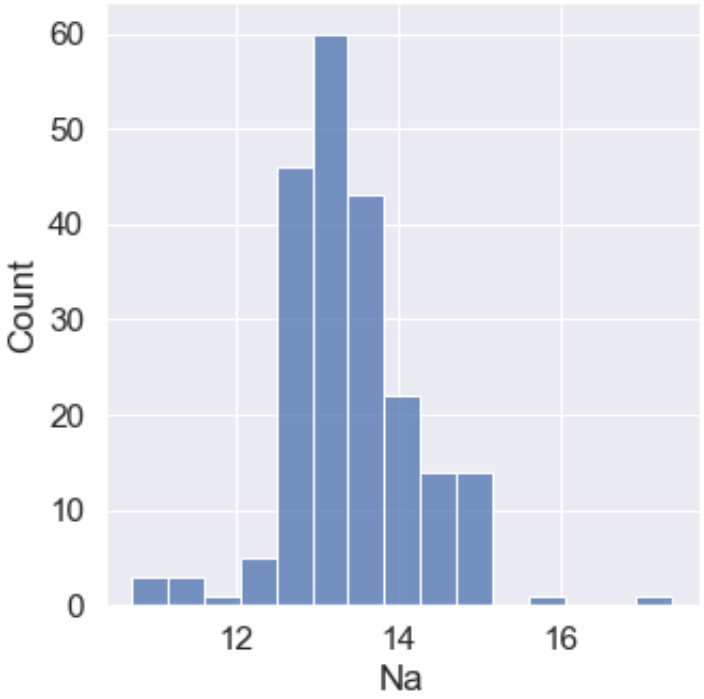
Výstup 4.5: Základní statistické charakteristiky proměnných

Indexu lomu (RI) má malou směrodatnou odchylku 0,003 – hodnoty proměnné se nacházejí v těsném intervalu od zhruba 1,5 do zhruba 1,534. Z histogramu na výstupu 4.6 je zřejmé, že rozdělení RI odpovídá normálnímu.



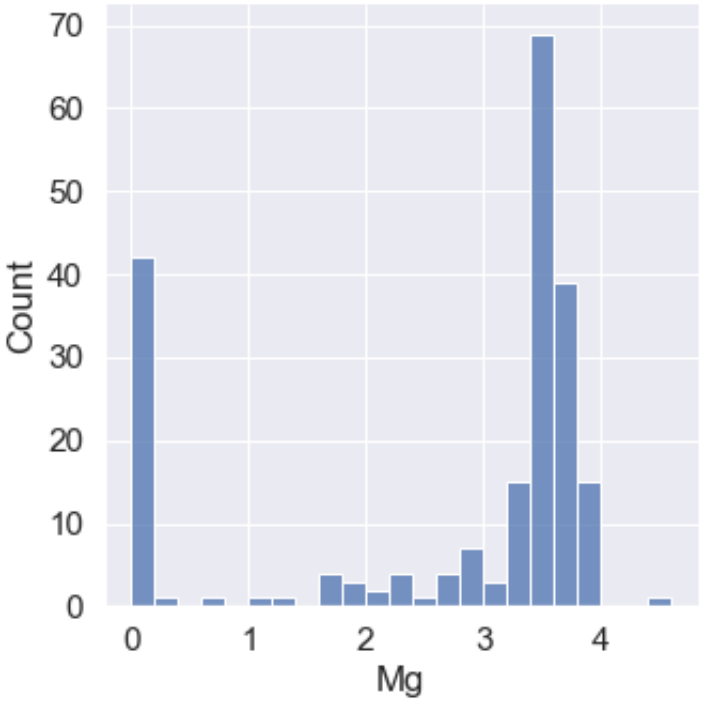
Výstup 4.6: Histogram proměnné RI – index lomu

Množství sodíku (Na) se pohybuje v intervalu od 10,73 do 17,38. Průměr 13,408 je velmi blízko 50. percentilu (13,3), z toho vyplývá, že proměnná má poměrně souměrné rozložení hodnot. Histogram 4.7 ukazuje, že rozdělení proměnné odpovídá normálnímu.



Výstup 4.7: Histogram proměnné Na – sodík

Množství hořčíku (Mg) se pohybuje v intervalu od nuly do 4,49. Průměr 2,684533 se nachází pod 50. percentilem, který je velmi blízko 75. percentilu. Histogram 4.8 ukazuje, že rozdělení proměnné odpovídá normálnímu až na poměrně vysoký počet hodnot rovných nebo blížících se nule. Obrázek 4.9 ukazuje modus proměnné, kterým je nula se 42 výskyty, tedy nula tvoří zhruba 19,7% hodnot proměnné. Tento podíl může znamenat, že se hořčík nevyskytuje v některé ze tříd. Boxplot na výstupu 4.10 ukazuje, že v záznamech zařazených do třídy 7 se ve skutečnosti nachází několik nenulových hodnot, zobrazených jako odlehlá pozorování, lze tedy usoudit, že převážnou většinu hodnot tvoří nula. Boxplot zároveň ukazuje výrazný rozdíl v množství hořčíku mezi jednotlivými třídami, vyjma tříd 1 a 3. Proměnná tedy do modelu klasifikace přináší hodnotnou informaci.

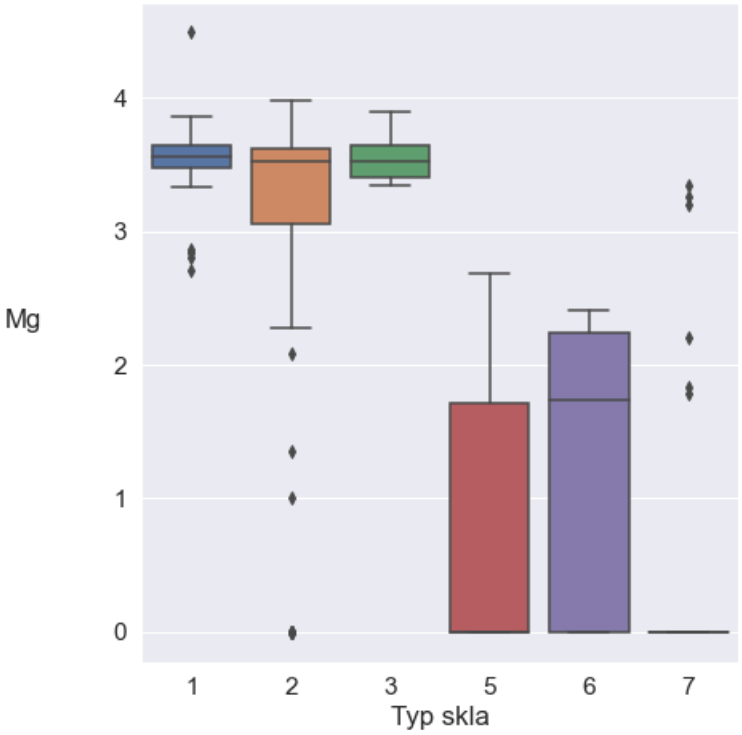


Výstup 4.8: Histogram proměnné Mg – hořčík

```
data['Mg'].value_counts()[0:0]
```

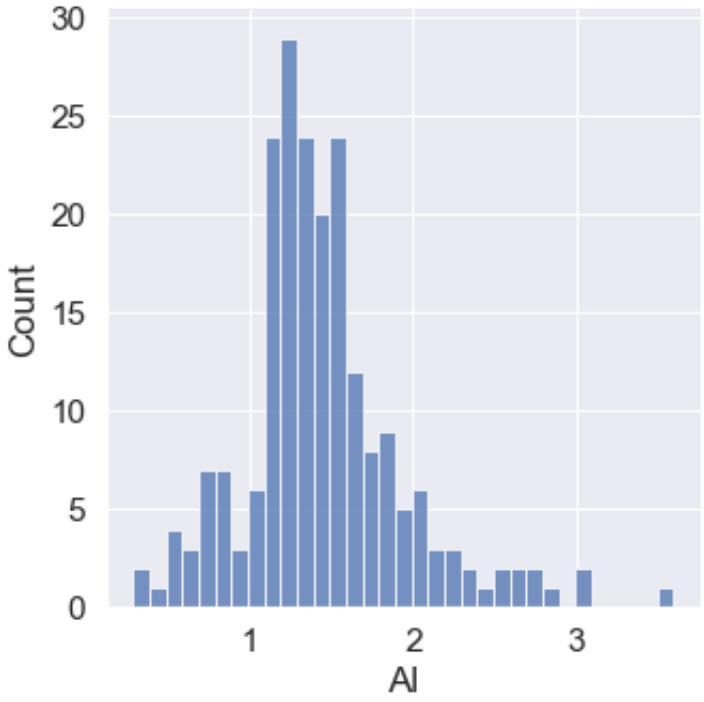
```
0.0    42
Name: Mg, dtype: int64
```

Obrázek 4.9: Modus proměnné Mg a jeho počet výskytů



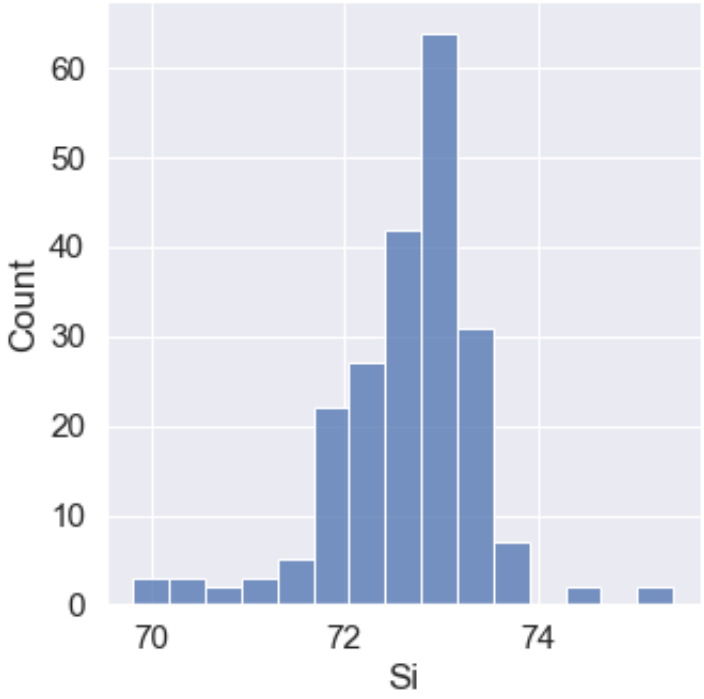
Výstup 4.10: Boxplot proměnné Mg

Množství mědi (Al) se pohybuje v intervalu od 0,29 do 3,5. Průměr, 50. i 75. percentil se pohybuje okolo hodnoty 1,5. Histogram 4.11 ukazuje, že proměnná má rozdělení odpovídající normálnímu.



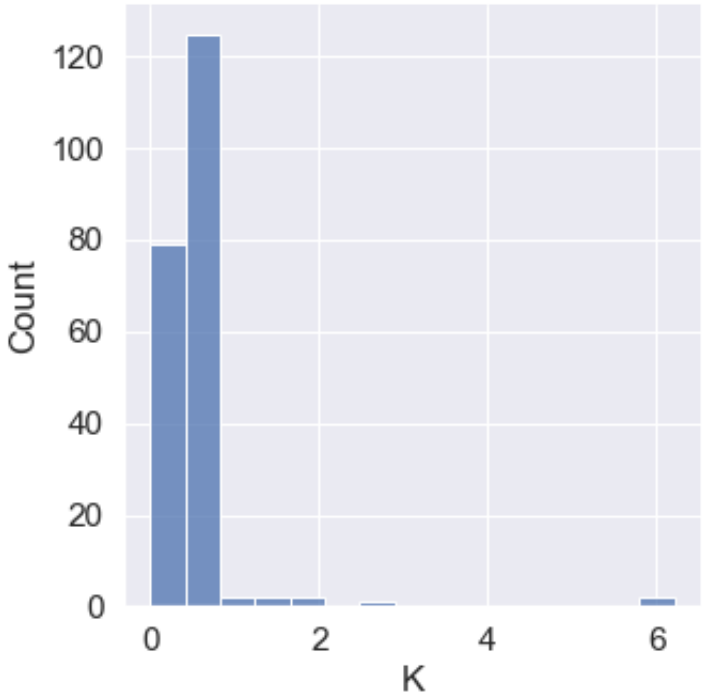
Výstup 4.11: Histogram proměnné Al – měď

Množství křemíku (Si) nabývá ze všech materiálů nejvyšších hodnot – pohybuje se v intervalu od 69,81 do 75,41. Průměr 72,651 se nachází velmi blízko 50. percentilu (72,79). Histogram 4.12 ukazuje, že proměnná má rozdělení odpovídající normálnímu.

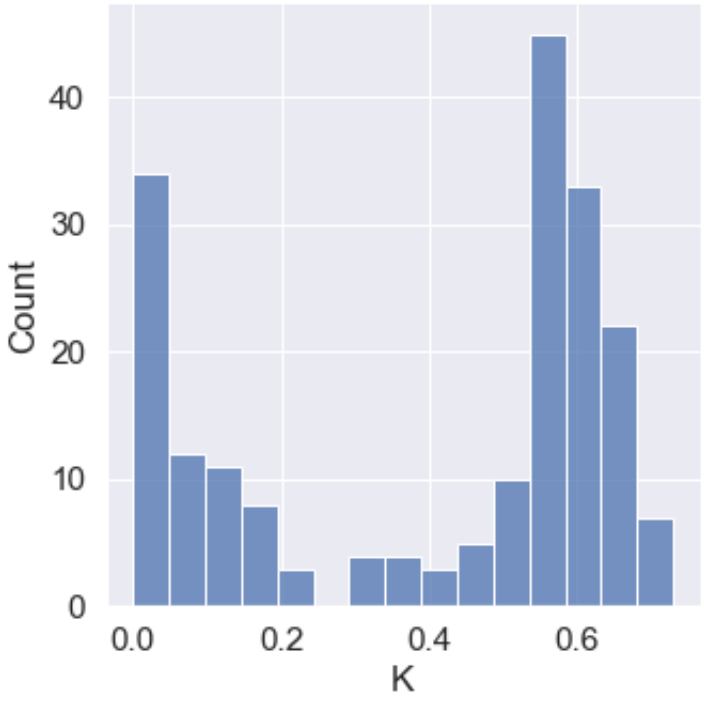


Výstup 4.12: Histogram proměnné Si – křemík

Množství draslíku (K) se pohybuje v intervalu od nuly do 6,21. Průměr (zhruba 0,497) a 50. percentil jsou velmi blízko nejmenší hodnoty, zatímco maximální hodnota (6,21) je více než desetkrát větší než 75. percentil (0,61). Proměnná je tedy výrazně asymetrická. Z histogramu 4.13 nelze přesně vyčíst rozdělení proměnné, protože ho zkreslují vysoké hodnoty, kterých je však malé množství. Na výstupu 4.14 je histogram, který zachycuje hodnoty proměnné do 95. percentilu, čímž je odstraněno pouze 5% hodnot, jejichž absence nemá výrazný vliv na tvar celého rozdělení. Histogram ukazuje, že rozdělení proměnné přibližně odpovídá normálnímu, přičemž se v něm vyskytuje anomálie hodnot blížících se (či rovné) nule. Modus proměnné na obrázku 4.15 je roven právě nule, která se objevuje 30 krát.



Výstup 4.13: Histogram proměnné K – draslík



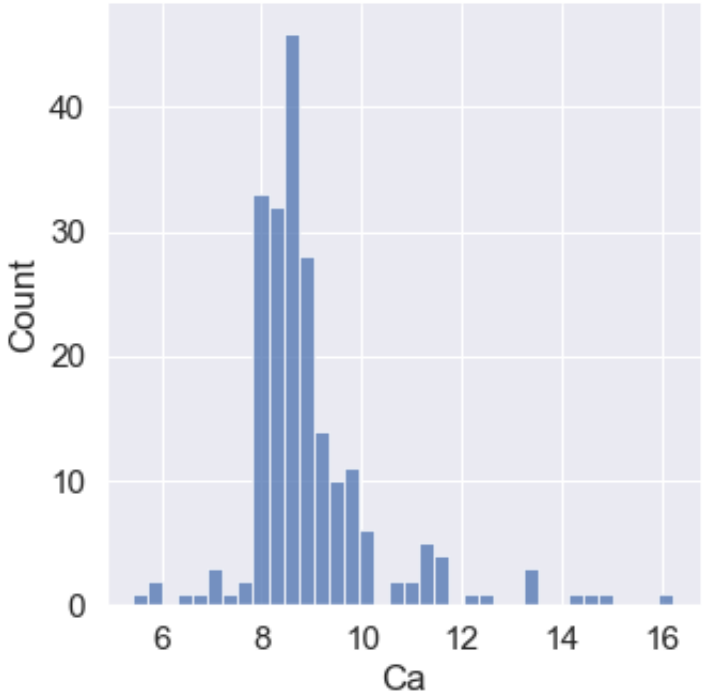
Výstup 4.14: Histogram proměnné K s hodnotami do 95. percentilu

```
data['K'].value_counts()[0:0]
```

```
0.0    30
Name: K, dtype: int64
```

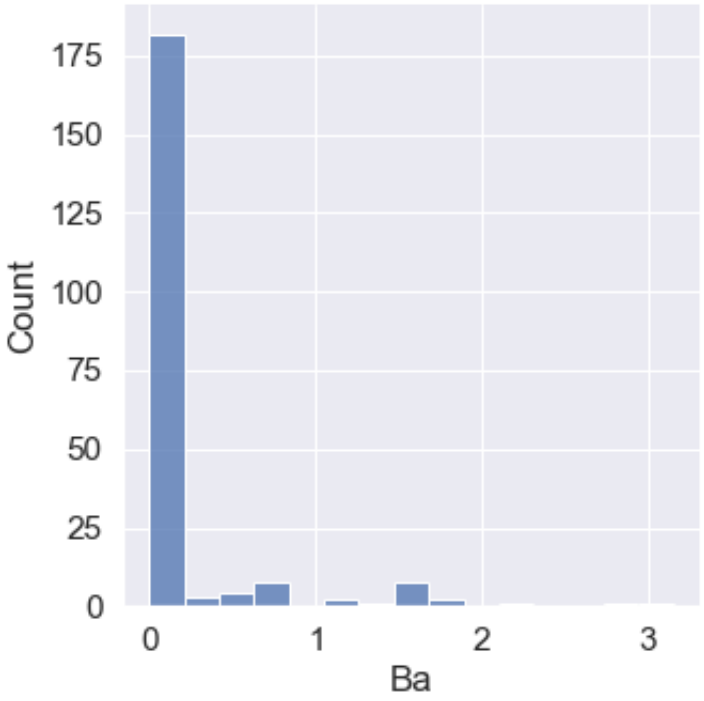
Obrázek 4.15: Modus proměnné K a jeho počet výskytů

Množství vápníku (Ca) se pohybuje v intervalu od 5,43 do 16,19. Všechny tři vybrané percentily se pohybují mezi 8 a 9,2, to znamená, že polovina všech hodnot se nachází ve zhruba jedné desetině intervalu. Histogram 4.16 ukazuje, že rozdělení proměnné odpovídá normálnímu.



Výstup 4.16: Histogram proměnné Ca – vápník

Množství barya (Ba) má při nejmenším 75% hodnot (tedy výraznou většinu) rovnou nule – průměr i vybrané percentily jsou nulové. Silná převaha těchto hodnot je vidět i na histogramu 4.17 a proměnná nemá normální rozdělení. Obrázek 4.18 ukazuje modus proměnné, kterým je podle předpokladů nula se 175 výskyty. Boxplot na výstupu 4.19 ukazuje 5 číselný souhrn proměnné rozdělené podle tříd, zde je vidět, že nenulové hodnoty se vyskytují především u třídy 7, u zbývajících se jich vyskytuje pouze několik, zobrazených jako odlehlá pozorování. Proměnná Ba tudíž přináší do modelu informaci pro zařazení záznamu do třídy 7 (sklo ze světlometu).

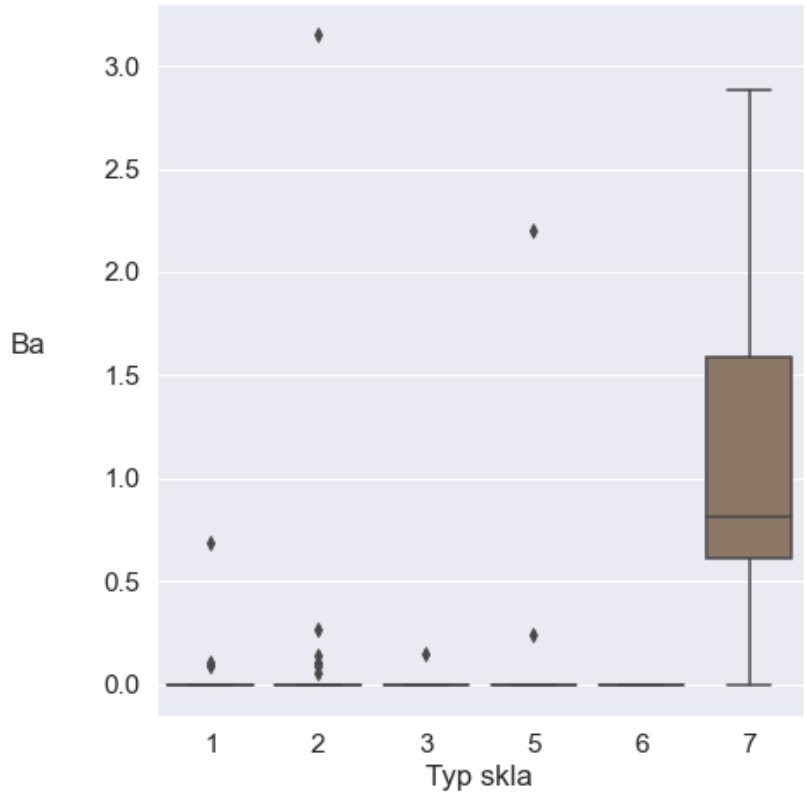


Výstup 4.17: Histogram proměnné Ba – Baryum

```
data['Ba'].value_counts()[0:0]
```

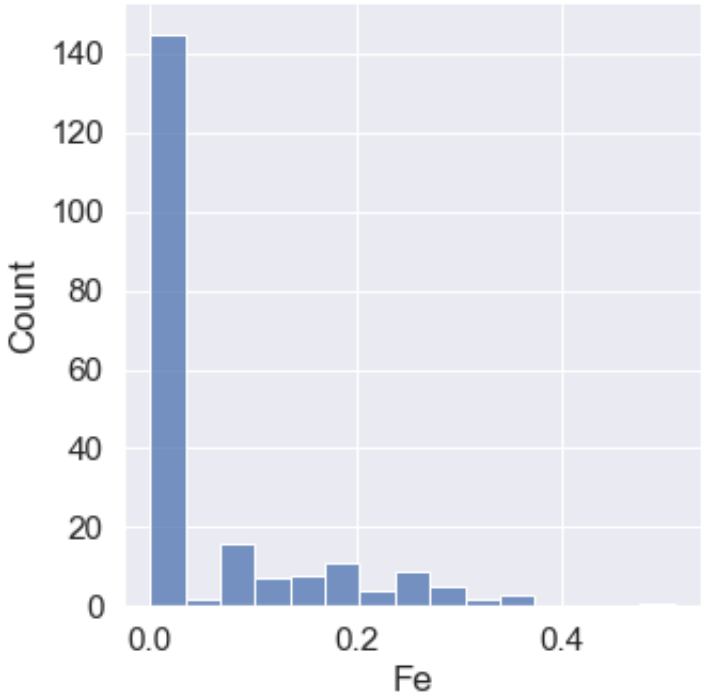
```
0.0    175
Name: Ba, dtype: int64
```

Obrázek 4.18: Modus proměnné Ba a jeho počet výskytů



Výstup 4.19: Boxplot proměnné Ba

Množství železa (Fe) nabývá hodnot z intervalu od nuly do 0,51, hodnota odchylky je 0,097439. Průměr činí 0,057, je tedy velmi blízko minimální hodnotě. Z těchto parametrů lze usoudit, že proměnná často nabývá nízkých či přímo nulových hodnot. To ukazuje i histogram 4.20. Z výstupu 4.21 je zřejmé, že nízký průměr je způsoben tím, že modusem proměnné je nula, která se vyskytuje 143 krát a tvoří zhruba 67% všech jejích hodnot. Boxplot 4.22 ukazuje, že je tento poměr způsoben rozdělením drtivé většiny nenulových hodnot mezi třídami 1, 2 a 3, u tříd 5 a 7 se jich nachází pouze několik, zobrazených jako odlehlá pozorování.

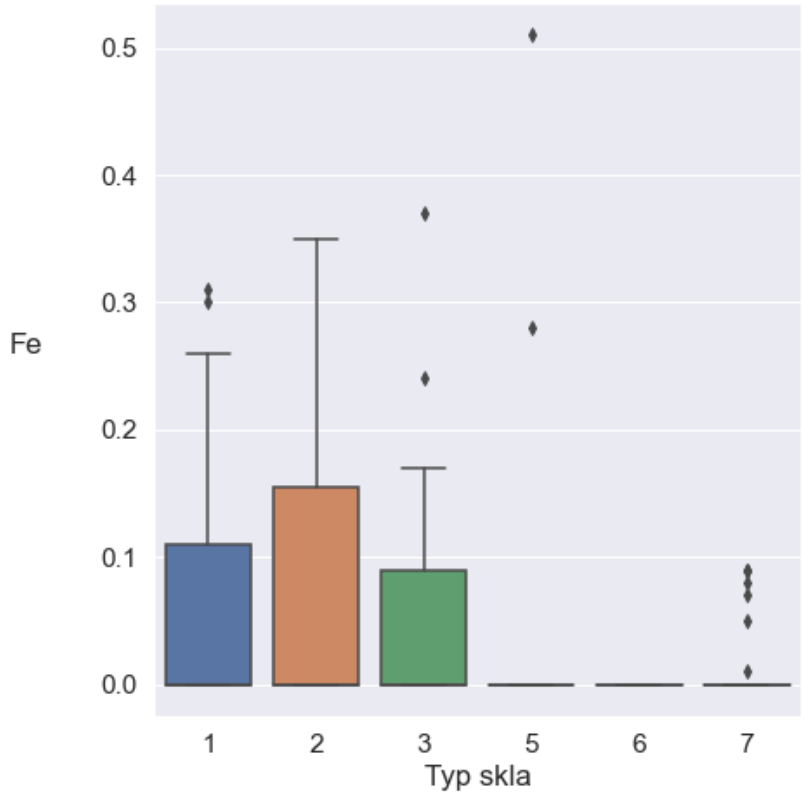


Výstup 4.20: Histogram proměnné Fe – železo

```
data['Fe'].value_counts()[0:0]
```

```
0.0    144
Name: Fe, dtype: int64
```

Výstup 4.21: Modus Fe a jeho počet výskytů



Výstup 4.22: Boxplot proměnné Fe

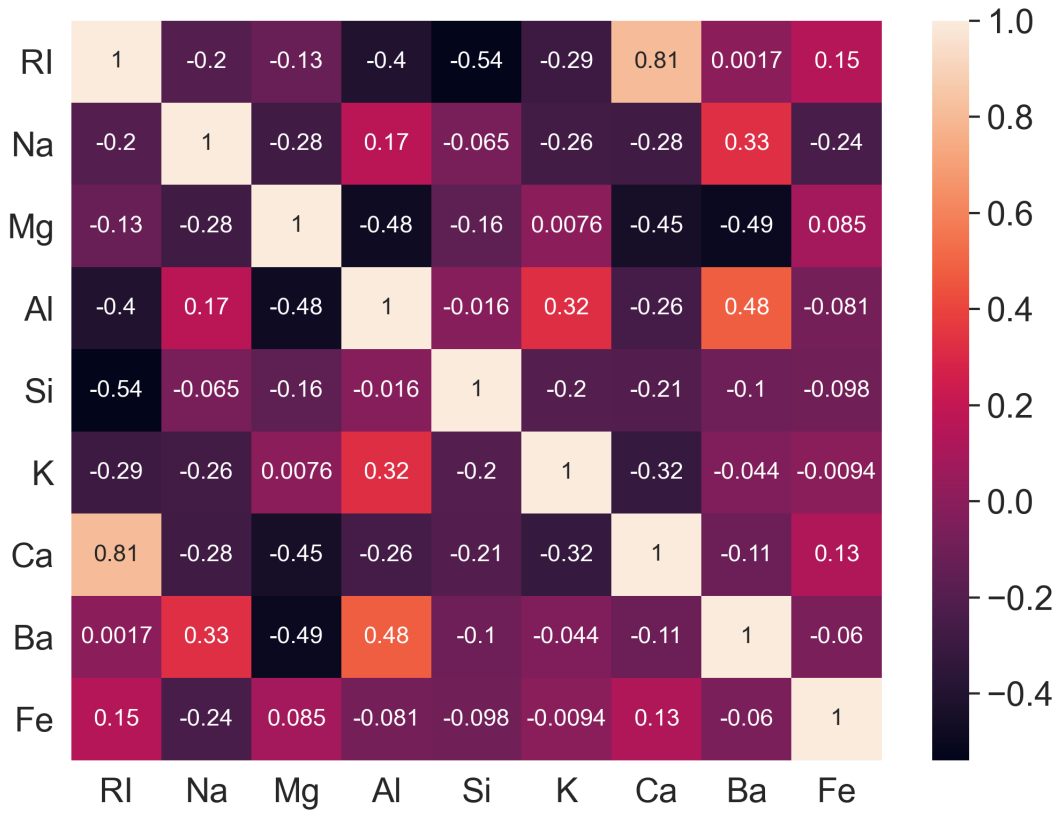
4.10.2.2 Korelace a analýza tříd

Korelační matice na obrázku 4.23 ukazuje párové korelace mezi jednotlivými proměnnými. Korelace jsou vypočteny pomocí Pearsonova koeficientu. Pro jednodušší nalezení korelací je matice vizualizována pomocí teplotní mapy na obrázku 4.24. Čím je barva pole mapy světlejší, tím vyšší je hodnota odpovídajícího korelačního koeficientu, čím je barva tmavší tím je jeho hodnota nižší.

Vizualizace ukazuje, že korelace se vyskytuje mezi proměnnými RI a Ca. Jejich korelační koeficient činí 0,810403, mezi proměnnými je silná, přímá závislost. Koeficient proměnných RI a Si činí -0,542052, mezi proměnnými je mírná, nepřímá závislost. Koeficienty zbývajících proměnných činí v absolutní hodnotě méně než 0,5, mezi těmito proměnnými je buď jen nízká (přímá nebo nepřímá) závislost a nebo jsou proměnné nezávislé.

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe
RI	1.000000	-0.198802	-0.127526	-0.400973	-0.539000	-0.287645	0.811183	0.001679	0.147083
Na	-0.198802	1.000000	-0.278420	0.167735	-0.064885	-0.264158	-0.278194	0.329080	-0.239374
Mg	-0.127526	-0.278420	1.000000	-0.479575	-0.162437	0.007617	-0.446197	-0.491818	0.085426
Al	-0.400973	0.167735	-0.479575	1.000000	-0.016195	0.323683	-0.258068	0.480642	-0.080583
Si	-0.539000	-0.064885	-0.162437	-0.016195	1.000000	-0.197281	-0.207145	-0.104389	-0.097717
K	-0.287645	-0.264158	0.007617	0.323683	-0.197281	1.000000	-0.317032	-0.043653	-0.009372
Ca	0.811183	-0.278194	-0.446197	-0.258068	-0.207145	-0.317032	1.000000	-0.112208	0.126314
Ba	0.001679	0.329080	-0.491818	0.480642	-0.104389	-0.043653	-0.112208	1.000000	-0.059729
Fe	0.147083	-0.239374	0.085426	-0.080583	-0.097717	-0.009372	0.126314	-0.059729	1.000000

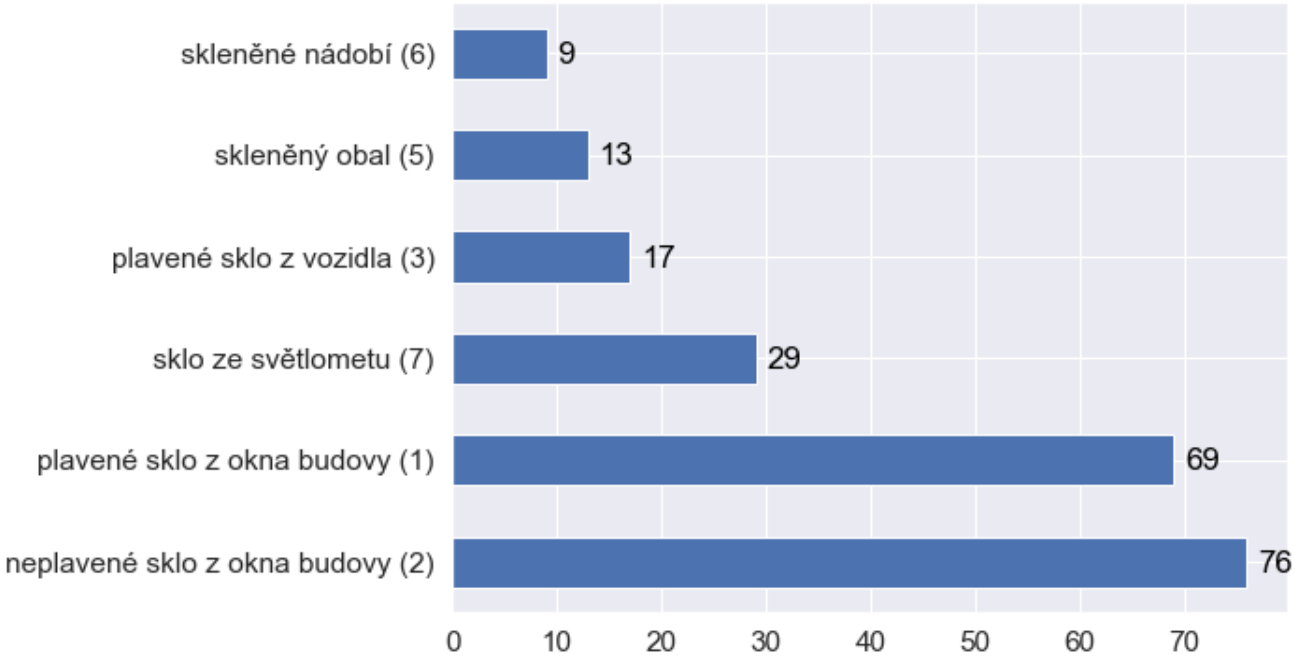
Výstup 4.23: Korelační matice



Výstup 4.24: Tepelná mapa korelační matice

Na výstupech v příloze C jsou korelace vizualizovány pomocí bodových (korelačních) diagramů. Ty potvrzují, že se vysoká korelace vyskytuje mezi proměnnými RI a Ca (výstup C.1, 2. řádek 3. sloupec), jejichž diagram se zhruba blíží přímce. Mezi proměnnými RI a Si lze vidět podobný jev (výstup C.1, 2. řádek 1. sloupec), avšak s větším množstvím odlehlých bodů, korelace zde není tolik silná. Na žádném jiném diagramu se již podobné obrazce připomínající korelované proměnné nevyskytují.

Při analýze tříd jsou pro snadnější orientaci nahrazena číselná vyjádření názvy, které jsou při dalším postupu učení opět nahrazeny čísly. Výstup na obrázku 4.25 ukazuje, že v sadě jsou nejvíce zastoupeny třídy plaveného a neplaveného skla z okna budovy, které představují více než polovinu výskytů všech tříd. Třída číslo 4 – neplavené sklo z vozidla – se v sadě nevyskytuje.



Výstup 4.25: Rozdělení tříd

4.10.2.3 Závěr analýzy

Většina proměnných datové sady má normální rozdělení. Mezi proměnnými RI a Ca je silná korelace. Více než polovina záznamů patří buď do třídy 2, nebo 1, třída 4 se v datech nevyskytuje. Třídy 6 a 5 obsahuje jen malý počet záznamů a přesnost jejich predikce je těžko měřitelná.

4.10.3 Výběr proměnných a vytvoření trénovací sady

Proměnné RI a Ca jsou silně korelované, pro trénink je proto ponechána jen ta z nich, která má vyšší korelaci s atributem třídy. Na obrázku 4.26 je zdrojový kód výběru odstraněné proměnné, vytvoření trénovací sady a jeho výstup. Z původní datové sady *data* jsou vybrány proměnné RI, Ca a label (atribut třídy), vypočtena korelační matice, ze které funkce nlargest vybere řádky s nejvyššími hodnotami korelace, z nich je vybrán název 2. řádku (1. je korelace sama na sebe) a uložen do proměnné *rm_feature*. Poté je vypsán její název. Pomocí funkce drop je vytvořena kopie datové sady bez odstraněné proměnné a uložena do proměnné *training_set*. Ze sady byla odstraněna proměnná Ca. To potvrzuje i výstup funkce head.

```
cm = data[['RI', 'Ca', label]].corr()
rm_feature = cm[label].nlargest().index[0:].tolist()[1]
print('Odstraněná proměnná: ', rm_feature)

training_set = data.drop(columns=rm_feature)
training_features = features.copy()
training_features.remove(rm_feature)
training_set.head()
```

```
Odstraněná proměnná:  Ca
```

	RI	Na	Mg	Al	Si	K	Ba	Fe	label
1	1.52101	13.64	4.49	1.10	71.78	0.06	0.0	0.0	1
2	1.51761	13.89	3.60	1.36	72.73	0.48	0.0	0.0	1
3	1.51618	13.53	3.55	1.54	72.99	0.39	0.0	0.0	1
4	1.51766	13.21	3.69	1.29	72.61	0.57	0.0	0.0	1
5	1.51742	13.27	3.62	1.24	73.08	0.55	0.0	0.0	1

Obrázek 4.26: Vytvoření trénovací sady

4.10.4 Výběr testovací základny

Nejlepší testovací základna byla vybrána pomocí funkce baseline_search, která vytvoří jednotlivé základny jako instance třídy DummyClassifier z knihovny pandas. Typy základen jsou zadané volitelným parametrem *strategies* (pokud není zadán, jsou vybrané všechny dostupné strategie). Poté funkce nastaví jednotlivé základny podle trénovacích dat a provede predikci na testovací části. Nakonec pomocí funkce score

vypočítá průměrnou přesnost základen a vrátí tu s nejvyšší přesností. Pro přehled vrací také pole dvojic základna-přesnost. Výstup 4.27 ukazuje, že nejvyšší přesnosti dosahují shodně základny *most_frequent* a *prior*, jako nejlepší byla vybrána most_frequent.

```
Typy základen a jejich skóre:
     stratified      0.286
  most_frequent      0.357
          prior      0.357
        uniform      0.174

Nejlepší základna: most_frequent
```

Výstup 4.27: Výběr testovací základny

4.10.5 Trénování algoritmů

Nejprve jsou jednotlivé algoritmy otestovány s výchozími hodnotami hyperparametrů daných implementací. Testování provádí funkce cv_accuracy, která vypíše průměrnou přesnost predikce zadaného algoritmu při rozvrstvené křížové validaci (včetně směrodatné odchylky) a to jak celkovou, tak pro jednotlivé třídy. Implementace funkce předpokládá, že předaný algoritmus je instance příslušné třídy z knihovny scikit learn, jejíž rozhraní zahrnuje funkce fit a predict. Počet částí pro křížovou validaci je nastaven na 9, aby v každé byl zástupce i pro třídu 6, kterých je v trénovací sadě právě 9. Výpočet přesnosti predikce jednotlivých tříd vychází z výpočtu *confusion matrix*, tzv. chybové matice (též matice záměn), realizovaného shodně pojmenovanou funkcí knihovny scikit learn. Řádky matice tvoří predikce vrácené modelem, sloupce tvoří skutečné třídy, do kterých předložené vzory patří. Přesnost predikce tříd je vypočítána jako procento správných přiřazení (hodnoty na diagonále matice dělené celkovým počtem predikcí převedené na procenta). Výstup 4.28 ukazuje přesnost jednotlivých algoritmů, která se výrazně liší. Přesnost predikce tříd 5 a 6 je ovlivněna malým počtem trénovacích i testovacích vzorů, proto jim nelze přikládat příliš velký význam.

První výstup patří algoritmu SVM (označen jako SVC, což je název implementační třídy), který byl schopen predikovat pouze vzory třídy 2 a to se 100% úspěšností, což pravděpodobně znamená silné přeučení modelu. Jeho celková průměrná přesnost je 16,7%.

Druhý výstup patří algoritmu Nejbližších k sousedů, který nebyl schopen predikovat vzory ze třídy 3. Jeho celková průměrná přesnost 53,8% je více než trojnásobná oproti SVM.

Třetí výstup patří algoritmu rozhodovacího stromu. Strom dosáhl průměrné přesnosti 60,5%, přičemž byl schopen predikovat část vzorů z každé třídy. U třídy 3 dosahuje nejnižší přesnosti predikce, která činí 33,33%. Přesnosti predikce nejčastěji zastoupených tříd 1 a 2 jsou blízko průměru.

Poslední výstup patří algoritmu Random Forest, který dosahuje podobné průměrné přesnosti jako rozhodovací strom, která činí 66,8% a je nejvyšší z testovaných modelů. Model dosahuje u většiny tříd vysoké přesnosti, jedinou výjimku tvoří přesnost predikce třídy 3, která je pouze 16,67% a třídy 5, která činí 55,56%. Stejně jako model Rozhodovacího stromu byl schopen správně predikovat alespoň malou část vzorů každé klasifikační třídy.

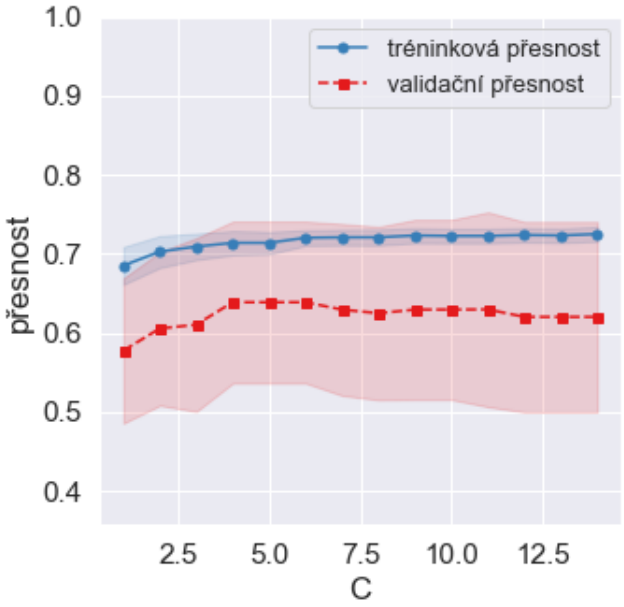
Dalším krokem je hledání co nejlepšího nastavení hyperparametrů pro minimalizování přeučení modelu pomocí křížové validace. Počet validačních částí je opět nastaven na 9. Parametry jsou vybírány podle přesnosti modelu vizualizované validačními křivkami, doplněnými o barevně vyznačené zóny, jejichž hranice jsou stanovené směrodatnou odchylkou přesnosti při daném nastavení modelu. Přesnost je v grafech na ose y, zobrazovány jsou hodnoty od přesnosti testovací základny (zhruba 0,37) do maximální hodnoty 1 (100% přesnost).

Pro SVM je nastavován hyperparametr C, křivka na výstupu 4.29 ukazuje, že nelze vyloučit nebezpečí přeučení, přičemž ani přesnost není příliš vysoká. Nejlepší nastavení hyperparametru je 5, kde jsou si trénovací a validační přesnosti nejblíže a při kterém se přesnost pohybuje mezi nejvyššími hodnotami.

Pro Nejlepších k sousedů je nastavován hyperparametr n_neighbors (počet sousedů) v kombinaci s metrikami výpočtu vzdálenosti mezi sousedy, které nabízí implementační třída z knihovny scikit learn. Každý diagram na výstupu 4.30 odpovídá jedné metrice. Počet sousedů se pohybuje v sudých číslech od 14 do 36. Výstup ukazuje, že křivky pro nastavení s 20 a méně sousedy jsou od sebe viditelně vzdálenější, lze předpokládat, že nižší počet sousedů nepřinese zlepšení. Ze výstupu lze vyčíst 3 kandidáty pro co nejlepší nastavení: 28 sousedů pro metriku euclidean, 26 pro manhattan a 28 pro metriku minkowski, jejíž validační křivky mají zřejmě stejný nebo velmi

```
--------------------------------
model:  SVC
průměrná přesnost: 16.7%, směrodatná odchylka: 0.37
průměrné přesnosti pro jednotlivé třídy:
třída  1 : 0.00%
třída  2 : 100.00%
třída  3 : 0.00%
třída  5 : 0.00%
třída  6 : 0.00%
třída  7 : 0.00%
--------------------------------
--------------------------------
model:  KNeighborsClassifier
průměrná přesnost: 53.8%, směrodatná odchylka: 0.42
průměrné přesnosti pro jednotlivé třídy:
třída  1 : 74.80%
třída  2 : 56.48%
třída  3 : 0.00%
třída  5 : 50.00%
třída  6 : 66.67%
třída  7 : 75.00%
--------------------------------
--------------------------------
model:  DecisionTreeClassifier
průměrná přesnost: 60.5%, směrodatná odchylka: 0.39
průměrné přesnosti pro jednotlivé třídy:
třída  1 : 64.29%
třída  2 : 60.49%
třída  3 : 33.33%
třída  5 : 55.56%
třída  6 : 66.67%
třída  7 : 82.41%
--------------------------------
--------------------------------
model:  RandomForestClassifier
průměrná přesnost: 66.8%, směrodatná odchylka: 0.37
průměrné přesnosti pro jednotlivé třídy:
třída  1 : 80.75%
třída  2 : 76.39%
třída  3 : 16.67%
třída  5 : 55.56%
třída  6 : 88.89%
třída  7 : 82.41%
--------------------------------
```

Výstup 4.28: Přesnosti algoritmů bez úprav hyperparametrů

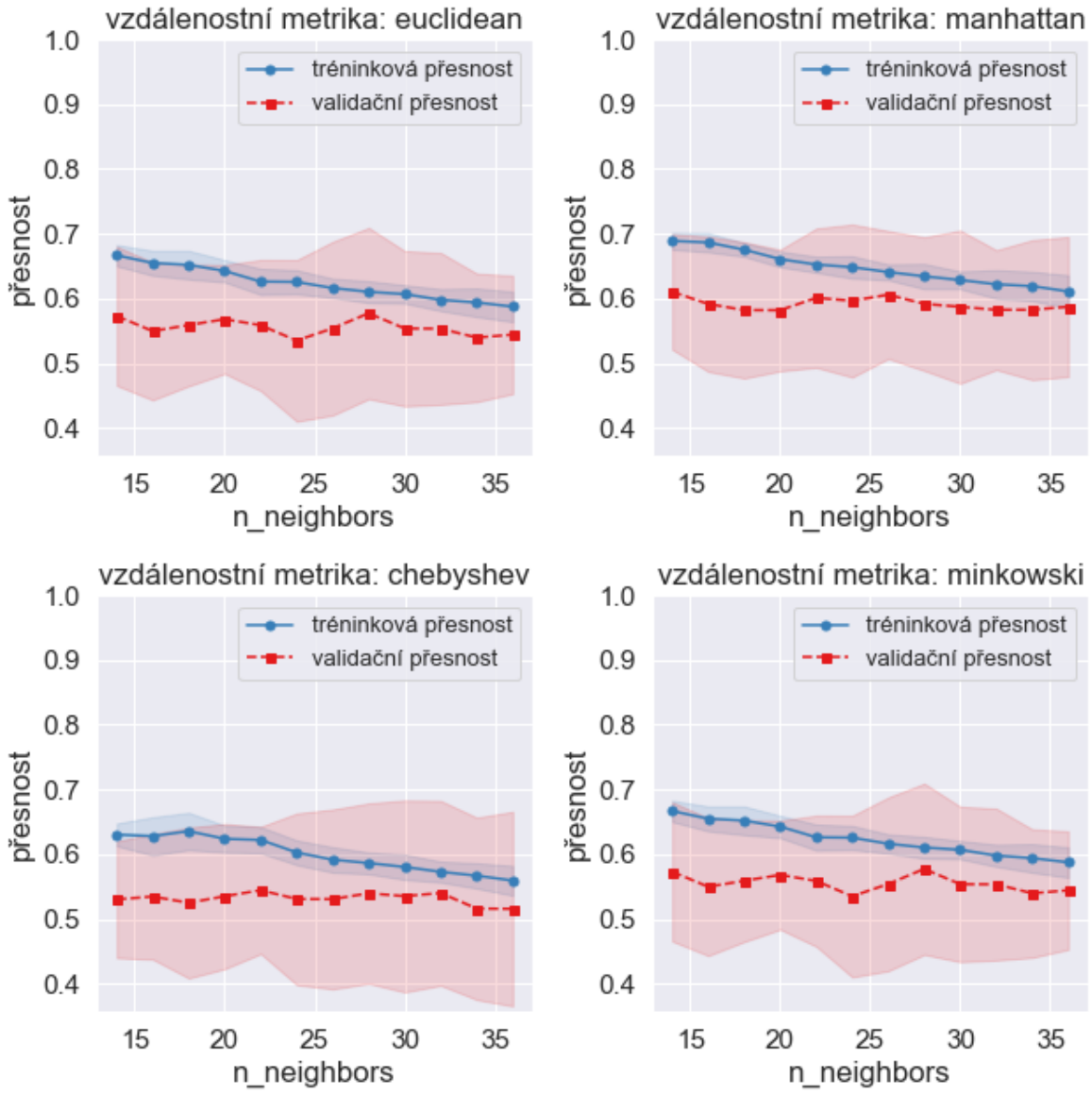


Výstup 4.29: Validační křivky SVM

podobný průběh jako křivky metriky euclidean. Tyto hodnoty představují nejnižší riziko přeučení v rámci jednotlivých metrik při zachování poměrně vysoké přesnosti oproti ostatním nastavením. Rozdíly mezi těmito variantami jsou jen těžko rozpoznatelné, proto je upřednostněna varianta s nejvyšší průměrnou přesností. Výstup 4.31 ukazuje hodnoty přesnosti. Nejlepším nastavením algoritmu je metrika manhattan se 26 sousedy.

Pro Rozhodovací strom je nastavován hyperparametr maximální hloubky stromu max_depth, pro variantu se shodnou váhou všech tříd a nastavením balanced pro vyvažování tříd podle frekvence jejich výskytu v trénovacích vzorech. Výstup 4.32 ukazuje, že ani u jedné z variant nevede větší hloubka k výraznému zlepšení přesnosti, navíc se křivky od sebe výrazně vzdalují, což značí přeučení. Strom s vyvažováním má při nejlepší nastavení maximální hloubku 3 úrovně, dosahuje vyšší přesnosti, ale také většího rizika přeučení, než strom s konstantními váhami s hloubkou 2. Přesnost této varianty patří spíše k nižším hodnotám daných křivek, což znamená nedoučení. Druhá varianta dosahuje jedné z nejvyšších přesností, proto je upřednostněna. Nejlepším nastavením je maximální hloubka 3 při vyvážených váhách tříd.

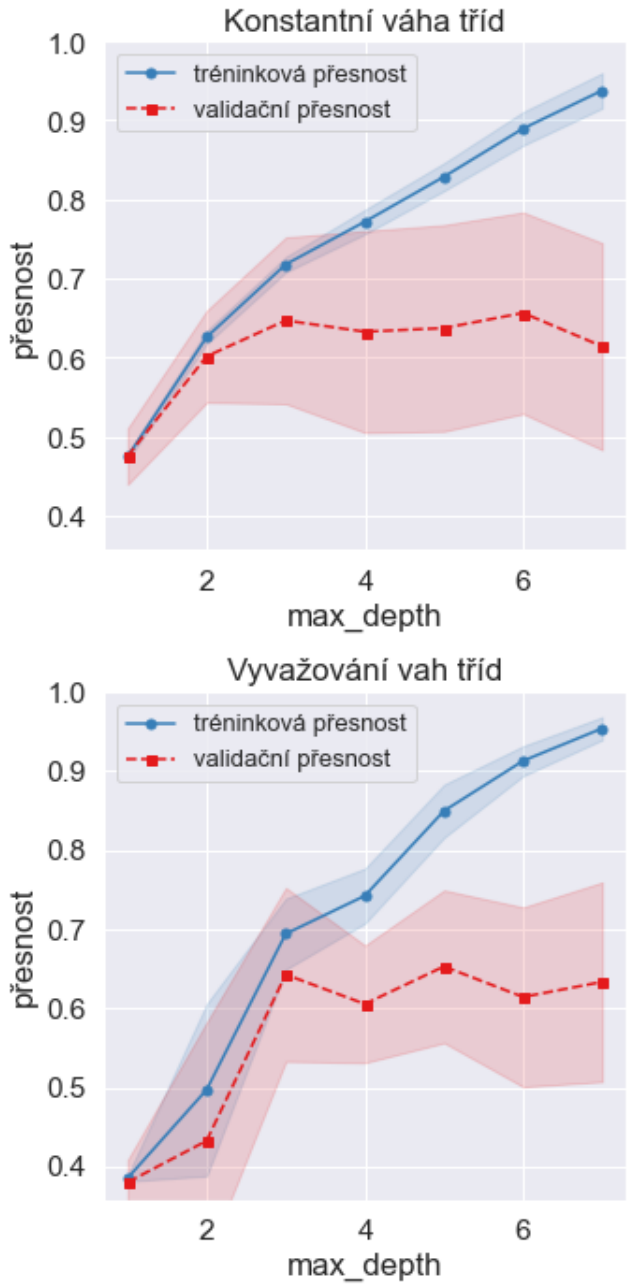
Pro Random Forest jsou nastavovány hyperparametry maximální hloubky max_depth a počtu stromů n_estimators. Na výstupu 4.33 vidíme, že ani řádový rozdíl v počtu stromů nevede k zásadním změnám průběhu křivek, u všech je však vidět poměrně stabilní míra



Výstup 4.30: Validační křivky Nejbližších sousedů

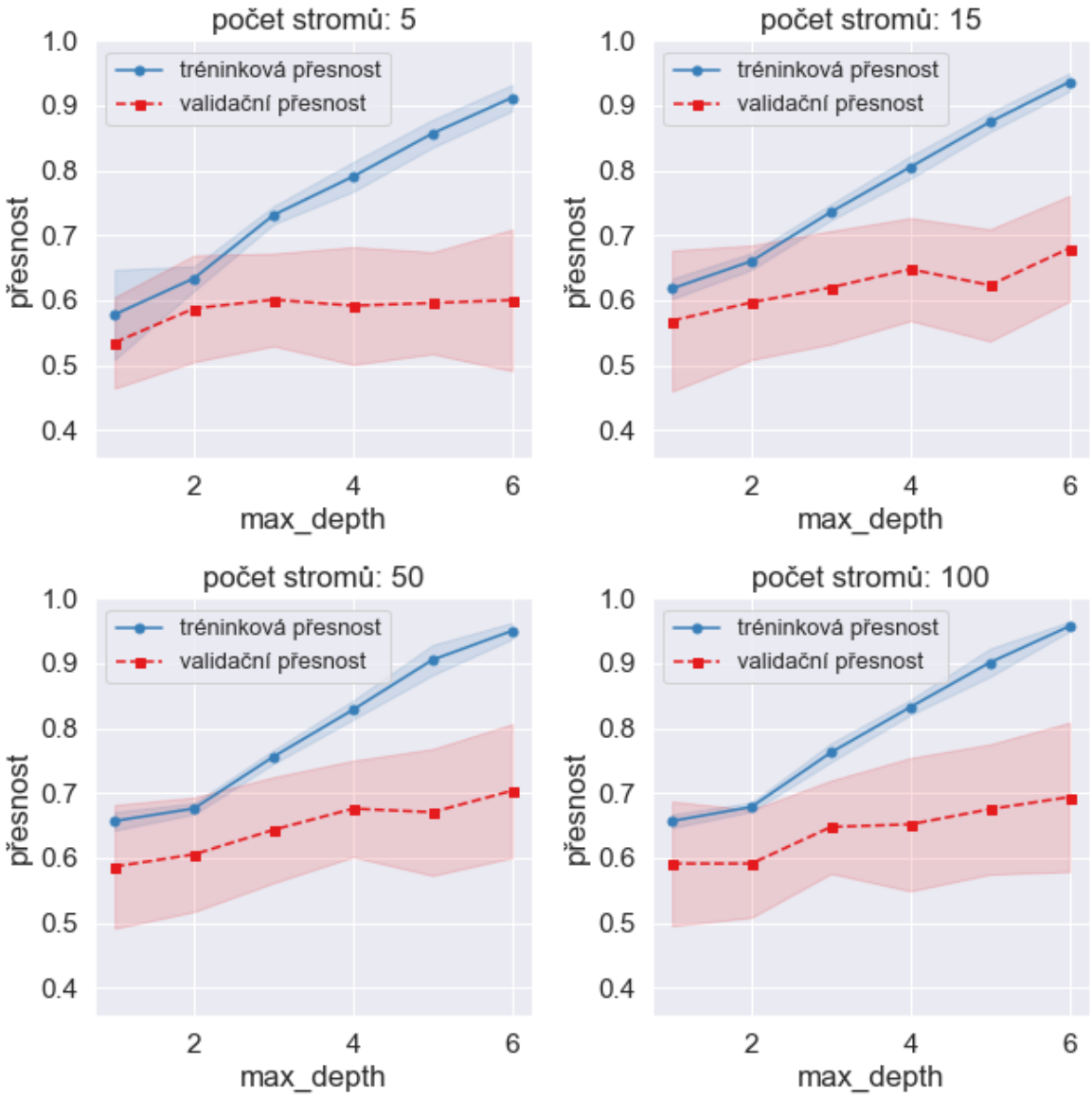
```
metrika euclidean, počet sousedů: 28, přesnost: 0.577 +/- 0.132
metrika manhattan, počet sousedů: 26, přesnost: 0.606 +/- 0.099
metrika minkowski, počet sousedů: 28, přesnost: 0.577 +/- 0.132
```

Výstup 4.31: Průměrná přesnost nejlepších nastavení Nejbližších k sousedů

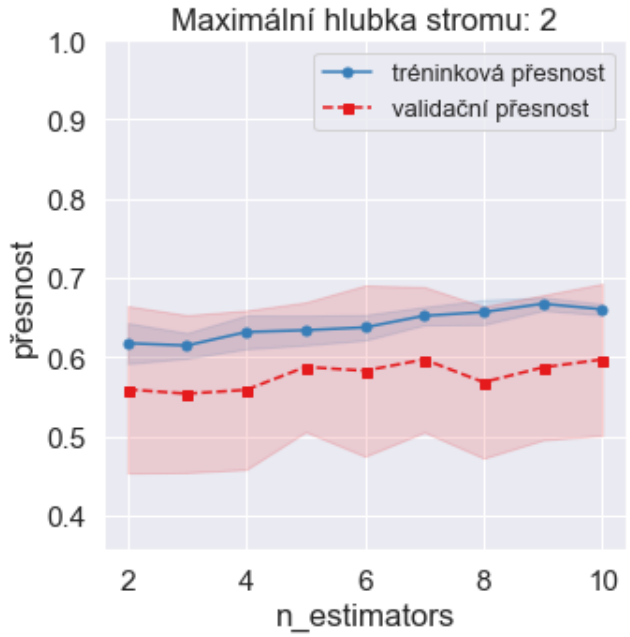


Výstup 4.32: Validační křivky Rozhodovacího stromu

rizika přeučení při nastavení maximální hloubky na 1 nebo 2 úrovně, které jsou ze všech hodnot nejméně rizikové na všech grafech. Protože vyšší počet stromů nepřináší výrazné zlepšení přesnosti ani snížení rizika přeučení, je z důvodu nižší výpočetní náročnosti zvolen počet stromů od 2 do 10. Protože hloubka 2 úrovně je na grafu pro 5 stromů nejlepší variantou, je na této hodnotě zafixována při hledání optimálního počtu stromů. Výstup 4.34 ukazuje, že v rozsahu do 10 stromů je nejlepší variantou 5 stromů. Nejlepším nastavením je tedy 5 stromů s maximální hloubkou 2.



Výstup 4.33: Validační křivky Random Forest



Výstup 4.34: Validační křivky Random Forest pro maximální hloubku stromu 2 úrovně

4.10.6 Otestování algoritmů a výběr nejlepšího modelu

Výstup 4.35 ukazuje otestování algoritmů s nastavením hyperparametrů podle jejich přesnosti na validačních křivkách. Nejlepšího zlepšení dosáhl algoritmus SVM, který více než zdvojnásobil svou průměrnou přesnost oproti výchozímu modelu a dosáhl celkové přesnosti 60,1%. Výchozí model správně predikuje pouze třídu 2, u které dosahuje 100% přesnosti, model s optimalizovanými hyperparametry má o 33,02% menší přesnost predikce této třídy, ale kromě třídy 3 dosahuje u všech nenulových přesností.

```
--------------------------------
model:  SVC
průměrná přesnost: 60.1%, rozdíl oproti výchozí: 43.4%
průměrné přesnosti pro jednotlivé třídy:
třída  1 : 66.67%, rozdíl oproti výchozí: 66.67%
třída  2 : 66.98%, rozdíl oproti výchozí: -33.02%
třída  3 : 0.00%, rozdíl oproti výchozí: 0.00%
třída  5 : 77.78%, rozdíl oproti výchozí: 77.78%
třída  6 : 66.67%, rozdíl oproti výchozí: 66.67%
třída  7 : 82.41%, rozdíl oproti výchozí: 82.41%
--------------------------------
--------------------------------
model:  KNeighborsClassifier
průměrná přesnost: 40.9%, rozdíl oproti výchozí: -12.9%
průměrné přesnosti pro jednotlivé třídy:
třída  1 : 85.52%, rozdíl oproti výchozí: 10.71%
třída  2 : 58.95%, rozdíl oproti výchozí: 2.47%
třída  3 : 0.00%, rozdíl oproti výchozí: 0.00%
třída  5 : 22.22%, rozdíl oproti výchozí: -27.78%
třída  6 : 0.00%, rozdíl oproti výchozí: -66.67%
třída  7 : 78.70%, rozdíl oproti výchozí: 3.70%
--------------------------------
--------------------------------
model:  DecisionTreeClassifier
průměrná přesnost: 70.8%, rozdíl oproti výchozí: 10.3%
průměrné přesnosti pro jednotlivé třídy:
třída  1 : 74.21%, rozdíl oproti výchozí: 9.92%
třída  2 : 45.83%, rozdíl oproti výchozí: -14.66%
třída  3 : 72.22%, rozdíl oproti výchozí: 38.89%
třída  5 : 72.22%, rozdíl oproti výchozí: 16.67%
třída  6 : 88.89%, rozdíl oproti výchozí: 22.22%
třída  7 : 71.30%, rozdíl oproti výchozí: -11.11%
--------------------------------
--------------------------------
model:  RandomForestClassifier
průměrná přesnost: 37.5%, rozdíl oproti výchozí: -29.2%
průměrné přesnosti pro jednotlivé třídy:
třída  1 : 78.77%, rozdíl oproti výchozí: -1.98%
třída  2 : 60.34%, rozdíl oproti výchozí: -16.05%
třída  3 : 0.00%, rozdíl oproti výchozí: -16.67%
třída  5 : 0.00%, rozdíl oproti výchozí: -55.56%
třída  6 : 0.00%, rozdíl oproti výchozí: -88.89%
třída  7 : 86.11%, rozdíl oproti výchozí: 3.70%
--------------------------------
```

Výstup 4.35: Přesnosti algoritmů s nastavením hyperparametrů z křížové validace

Algoritmus Nejbližších k sousedů dosáhl o 12,9% nižší přesnosti než výchozí model, u tříd 1 a 2 se jeho přesnost zlepšila o 10,71%, resp. 2,47%. Třídu 6 není při tomto nastavení schopen predikovat.

Algoritmus Rozhodovacího stromu dosáhl průměrné přesnosti 70,8% predikce třídy 2, o 10,3% více, než při výchozím nastavení. Podobně se zlepšila přesnost predikce u třídy 1 na 74,21%, naopak u třídy 2 se jeho predikce zhoršila na 45,83%. Jako jediný algoritmus predikuje všechny třídy.

Algoritmus Random Forest má s přenastavenými parametry nižší přesnost jak průměrnou, tak napříč všemi třídami s výjimkou třídy 7. Jeho průměrná přesnost se snížila o 29,2% na 37,5%. Třídy 3, 5 a 6 nepredikuje vůbec.

Nejlepší celkové přesnosti z optimalizovaných modelů dosahuje Rozhodovací strom. Jeho celková přesnost je nejvyšší ze všech modelů, predikuje všechny třídy s nenulovou průměrnou přesností, hodnocení sráží pouze nízká přesnost u 2. nejzastoupenější třídě trénovací sady. Optimalizace jeho parametrů také znamená nižší riziko výrazného poklesu jeho přesnosti při nasazení v praxi. Jako nejlepší model je vybrán Rozhodovací strom.

Pro účely možného využití vybraného modelu je vybraný model exportován pomocí funkce dump z knihovny joblib, která je součástí scikit learn. Jak ukazuje obrázek 4.36, model je nejdříve natrénován pomocí celé trénovací sady a poté je exportován do souboru *glass_classification_decision_tree.joblib*, následně je úspěšný export ověřen nahráním souboru funkcí load. Tento soubor lze nahrát do aplikace napsané v jazyce Python či do jiných Jupyter notebooků.

Naučení modelu na celé sadě

```
np.random.seed(1)
best_dtc = DecisionTreeClassifier(class_weight='balanced', max_depth=3, random_state=1)
best_dtc.fit(training_set[training_features].values, training_set[label].values);
```

Export modelu a otestování jeho nahrání

```
from joblib import dump, load

filename = 'glass_classification_decision_tree.joblib'

# export
dump(best_dtc, filename)

# nahrání pro otestování exportu
test_model_import = load(filename)
if test_model_import != None:
    print('Soubor modelu byl úspěšně nahrán.')
else:
    print('Soubor modelu nelze nahrát.')
```

```
Soubor modelu byl úspěšně nahrán.
```

Obrázek 4.36: Naučení a export vybraného modelu

Závěr

Aplikace Jupyter notebook je užitečný nástroj pro trénink algoritmů strojového učení. Díky podpoře jazyka Python a dalších nabízí mnoho nástrojů pro datovou analýzu, vizualizace, trénink a vyhodnocování výkonu algoritmů v přehledném prostředí. Výkon algoritmů pro řešení úloh učení pod dohledem závisí zejména na složení a velikosti trénovací sady dat, pokud neobsahuje dostatek adekvátních trénovacích vzorů, ať už celkově, či v případě klasifikace pro některé třídy, lze jen obtížně dosáhnout dostatečné přesnosti natrénovaného modelu, což se ukázalo na předvedeném postupu trénování algoritmů pro úlohu klasifikace druhů skla. Postup je sám o sobě poměrně přímočarý a může být nedostačující pro výběr nejlepšího modelu s nejvyšší možnou přesností. Na vybrané sadě nedokázal dojít k takovému výkonu klasifikátoru, který by byl vhodný k reálnému nasazení, což je při nejmenším z části způsobeno strukturou a velikostí datové sady, kde je pro některé třídy příliš málo trénovacích vzorů. Pro možná zlepšení je třeba jej otestovat na dalších, různorodých úlohách a datových sadách většího rozsahu.

Literatura

[1] NEAPOLITAN, R., JIANG, X. *Artificial Intelligence*. New York: Chapman and Hall/CRC, 2018 [cit. 2020-07-12]. 2nd Edition, 2. edice. ISBN 9781315144863. Dostupné z: https://doi-org.infozdroje.czu.cz/10.1201/b22400.

[2] ERTEL, Wolfgang. *Introduction to Artifical Intelligence*. Cham: Springer International Publishing, 2017 [cit. 2020-09-19]. ISBN 978-3-319-58486-7.

[3] WLODARCZAK, Peter. *Machine Learning and its Applications*. Boca Raton: CRC Press, 2019 [cit. 2020-06-19]. 1st Edition, 1. edice. ISBN 9780429448782. Dostupné z: https://doi-org.infozdroje.czu.cz/10.1201/9780429448782.

[4] MOHAMMED, M., KHAN, M., BASHIER, E. *Machine Learning*. Boca Raton: CRC Press, 19. 8. 2016 [cit. 2020-07-23]. 1st Edition, 1. edice. ISBN 9781315371658. Dostupné z: https://doi-org.infozdroje.czu.cz/10.1201/9781315371658.

[5] FORSYTH, David. Learning to classify. In: FORSYTH, David. *Applied Machine Learning*. Cham: Springer International Publishing, 2019, s. 3-19 [cit. 2020-07-21]. ISBN 978-3-030-18114-7.

[6] GALEA, Alex. *Beginning Data Science with Python and Jupyter*. Birmingham: Pact Publishing, 2018 [cit. 2020-04-15]. ISBN 978-1-78953-202-9.

[7] FORSYTH, David. Regression. In: FORSYTH, David. *Applied Machine Learning* [online]. (PDF). Cham: Springer International Publishing, 2019, s. 205-237 [cit. 2020-07-22]. ISBN 978-3-030-18114-7.

[8] MUNI, Masarykova univerzita. *SVM* [online]. [cit. 2020-07-18]. Dostupné z: https://is.muni.cz/el/1433/podzim2006/PA034/09_SVM.pdf.

[9] ERTEL, Wolfgang. Neural Networks. In: ERTEL, Wolfgang. *Introduction to Artifical Inteligence* [online]. (PDF). Cham: Springer International Publishing, 2017, s. 245-286 [cit. 2020-08-11]. ISBN 978-3-319-58487-4.

[10] FORSYTH, David. SVMs and Random Forests. In: FORSYTH, David. *Applied Machine Learning* [online]. (PDF) Cham: Springer International Publishing, 2019, s. 21-48 [cit. 2020-07-22]. ISBN 978-3-030-18114-7.

[11] SCIKIT-LEARN developers. *Support Vector Machines* [online]. [cit. 2021-02-27]. Dostupné z: https://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation.

[12] FORSYTH, David. Simple Neural Networks. In: FORSYTH, David. *Applied Machine Learning*. Cham: Springer International Publishing, 2019 [cit. 2020-08-16], s. 367-390. ISBN 978-3-030-18114-7.

[13] MENDELU, Mendelova univerzita v Brně. *Neuronové sítě*. [cit. 2020-04-04]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=21471.

[14] MAŘÍK, Vladimír a kol. *Umělá inteligence (4)*. 1. vydání. Praha: ACADEMIA, 2003 [cit. 2020-10-15]. ISBN 80-200-1044-0.

[15] CONE, Matt. *Getting started* [online]. [cit. 2020-05-12]. Dostupné z: https://www.markdownguide.org/getting-started/.

[16] JUPYTER team. *Jupyter Documentation* [online]. [cit. 2020-04-09]. Dostupné z: https://jupyter.readthedocs.io/en/latest/projects/architecture/content-architecture.html.

[17] PANDAS development team. *csv text files* [online]. [cit. 2020-03-27]. Dostupné z: https://pandas.pydata.org/docs/user_guide/io.html#csv-text-files.

[18] PANDAS development team. *DataFrame* [online]. [cit. 2020-03-28]. Dostupné z: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html#pandas.DataFrame.

[19] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

A Seznam použitých zkratek

HTML Hypertext Markup Language

AI Artificial intelligence

SVM Support Vector Machine

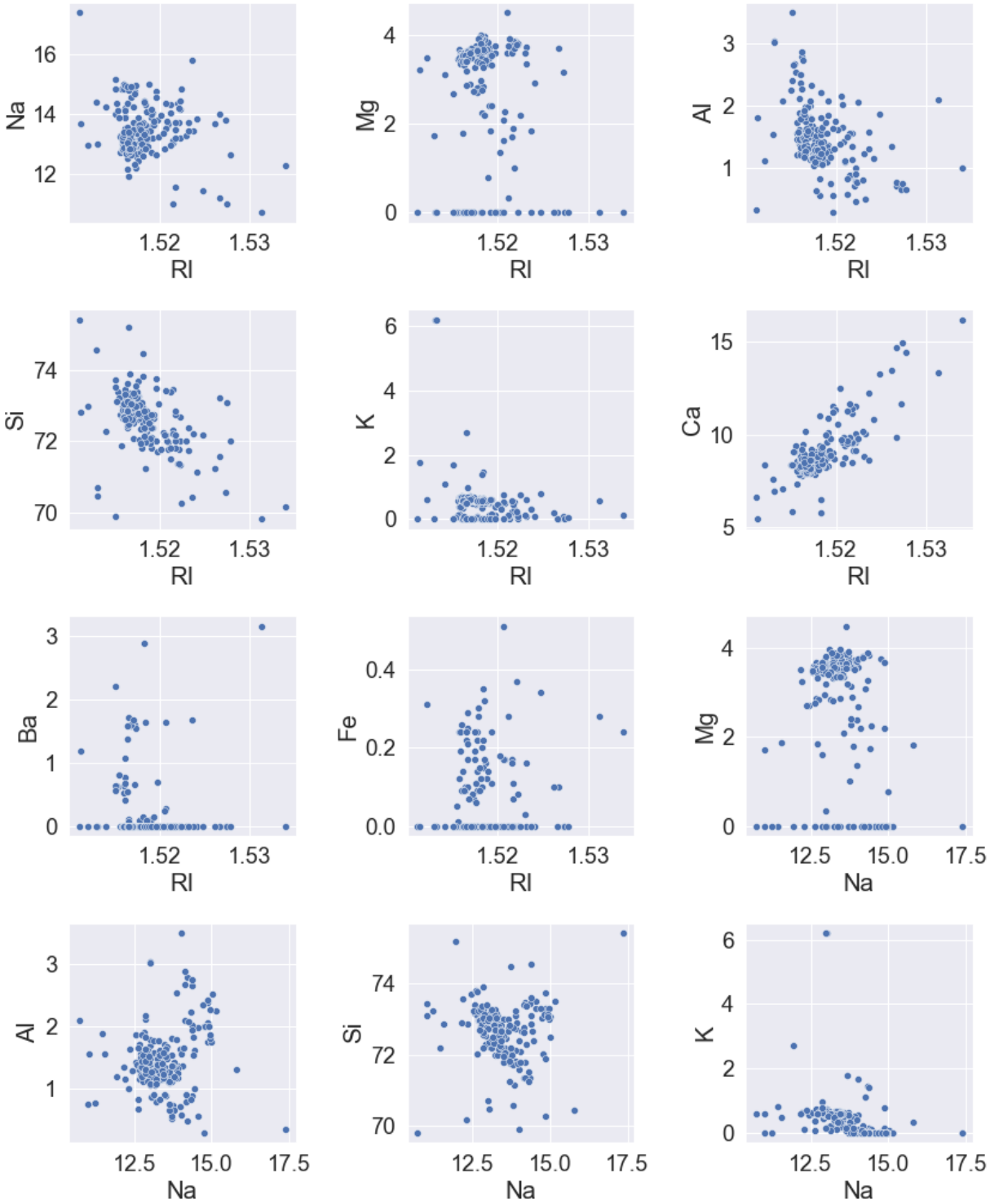
JSON Javascript Object Notation

B Obsah elektronické přílohy

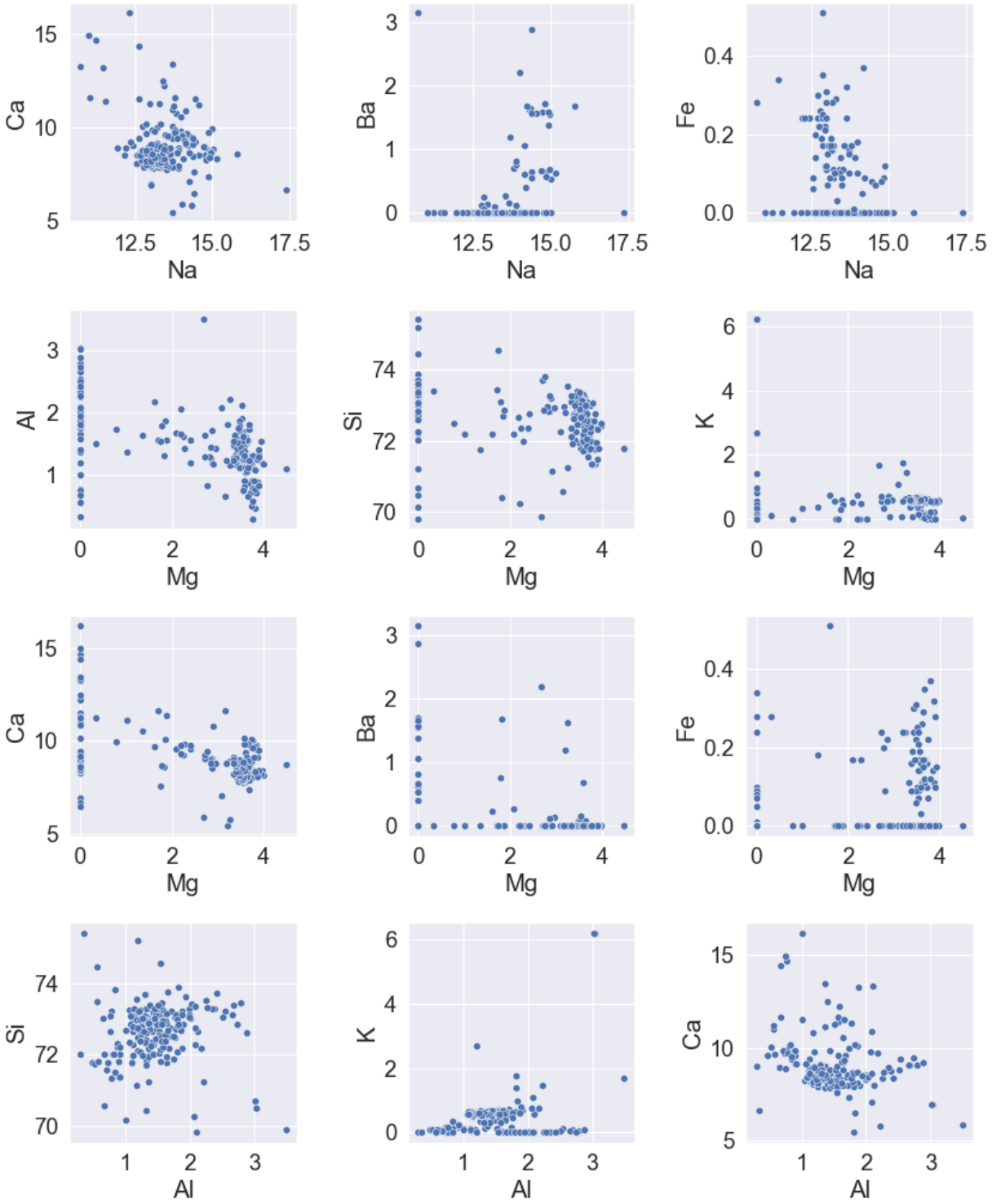
`GlassTypesLearning.ipynb` ... Soubor Jupyter notebooku s demonstrací postupu učení popsaného v práci

`glass_classification_decision_tree.joblib` ... Export natrénovaného modelu

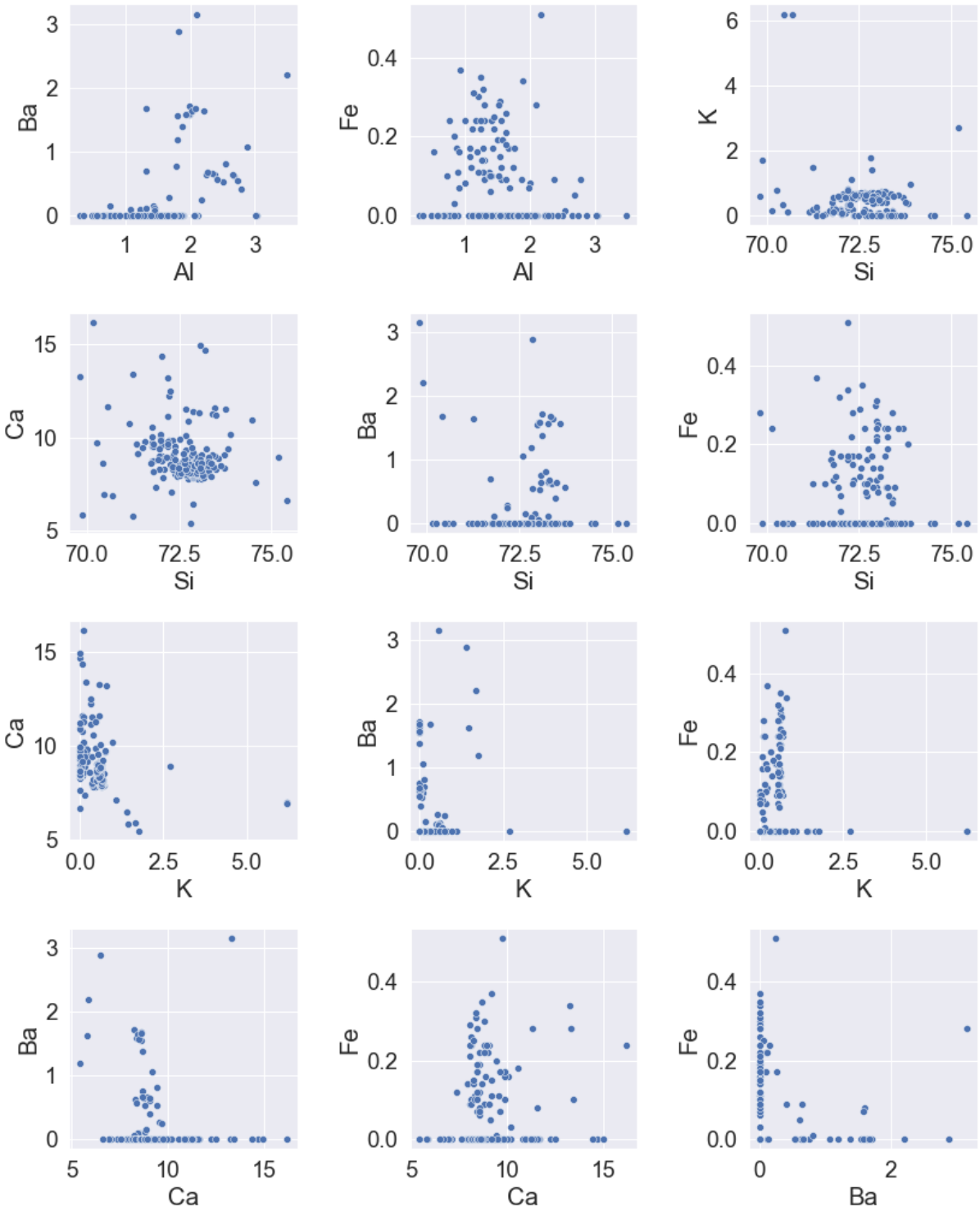
C Bodové diagramy proměnných datové sady



Obrázek C.1: bodové diagramy 1



Obrázek C.2: bodové diagramy 2



Obrázek C.3: bodové diagramy 3