České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství

Obor: Aplikace softwarového inženýrství

Automatizace předzpracování dat ve strojovém učení

Automation of Data Preprocessing in Machine Learning

BAKALÁŘSKÁ PRÁCE

Vypracoval: Anna Gruberová
Vedoucí práce: Ing. Václav Mácha
Rok: 2021

České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství Akademický rok 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	**Anna Gruberová**
Studijní program:	**Aplikace přírodních věd**
Obor:	**Aplikace softwarového inženýrství**
Název práce česky:	**Automatizace předzpracování dat ve strojovém učení**
Název práce anglicky:	**Automation of Data Preprocessing in Machine Learning**

Pokyny pro vypracování:

1. Seznamte se se základními modely strojového učení. Pozornost věnujte tomu, jakým způsobem musí být zpracována data pro trénování, validaci a testování těchto modelů.
2. Seznamte se s jednotlivými typy datových sad používaných ve strojovém učení jako jsou například tabulková data nebo obrazová data. Pro každý typ vyberte vhodné zástupce a navrhněte způsob, jak tyto data předzpracovat pro snadné použití ve strojovém učení.
3. Navrhněte a implementujte balík v programovacím jazyku Julia, který umožní snadné stažení vybraných datových sad a jejich předzpracování. Balík by měl splňovat následující kitéria:
   - Balík by měl stahovat pouze data, která chce uživatel skutečně použít.
   - Předzpracování dat se stejnými vstupními parametry by vždy mělo vést na stejný výsledek bez ohledu na to, na jakém počítači proběhlo, tzn. mělo by být reprodukovatelné.
   - Přidání nové datové sady by mělo být co nejjednodušší, tzn. funkce pro předzpracování by měly být obecné.

Doporučená literatura:

[1] BEZANSON, J., EDELMAN, A., KARPINSKI, S. and SHAN, V.B., *Julia: A fresh approach to numerical computing.* In: SIAM review, 59(1), 2017, pp.65-98.

[2] ZHENG, A. and CASARI A. *Feature engineering for machine learning: principles and techniques for data scientists.* Sebastopol: O'Reilly Media, 2018. ISBN 978-149-1953-242.

[3] HASTIE, T., TIBSHIRANI, R. and FRIEDMAN, J. H.. *The elements of statistical learning: data mining, inference, and prediction.* 2nd edition. New York: Springer, 2009. Springer series in statistics. ISBN 978-0-387-84857-0.

Jméno a pracoviště vedoucího práce:

Ing. Václav Mácha
Katedra matematiky, Fakulta jaderná a fyzikálně inženýrská, ČVUT v Praze

Jméno a pracoviště konzultanta:

–

Václav Mácha
.....
vedoucí práce

Datum zadání bakalářské práce: 29. 9. 2020

Termín odevzdání bakalářské práce: 7. 7. 2021

Doba platnosti zadání je dva roky od data zadání.

.....
garant oboru

.....
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská
-1-

.....
děkan

V Praze dne 29. 9. 2020

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracovala samostatně a použila jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne ....

Anna Gruberová

Poděkování

Chtěla bych poděkovat Ing. Václavu Máchovi za vedení mé bakalářské práce, za cenné rady a připomínky k tvorbě této práce a za čas strávený touto pomocí.

Anna Gruberová

Název práce:

Automatizace předzpracování dat ve strojovém učení

Autor: Anna Gruberová

Studijní program: Aplikace přírodních věd

Obor: Aplikace softwarového inženýrství

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Václav Mácha
Katedra matematiky, Fakulta jaderná a fyzikálně inženýrská, České vysoké učení technické v Praze

Konzultant: –

Abstrakt: Cílem této bakalářské práce je navrhnout a implementovat nástroj, který ulehčí stahování a předzpracování datových sad pro strojové učení. Nástrojů pro usnadnění práce s daty existuje velká řada, nicméně tyto nástroje sice poskytují funkce na stažení a další zpracování vybraných datových sad, ale neumožňují uživateli snadné přidání nové datové sady. Navíc ani tato data nepřevádí do jednotného výstupního formátu, takže není možné na nová data aplikovat již existující funkce. Stěžejní částí této práce je návrh nového nástroje, který řeší zmíněné nedostatky, a popis jeho implementace v programovacím jazyce Julia, včetně kompletního příkladu použití na konkrétní datové sadě.

Klíčová slova: Strojové učení, Předzpracování dat, Programovací jazyk Julia

Title:

Automation of Data Preprocessing in Machine Learning

Author: Anna Gruberová

Abstract: The goal of this bachelor thesis is to design and implement a tool, that facilitate downloading and preprocessing of data sets for machine learning. There is a large number of such tools, however, these tools provide functions for downloading and further processing of selected data sets, but do not allow the user to easily add a new data set. In addition, they do not even convert this data into a uniform output format, so it is not possible to apply existing functions to new data. The main part of this thesis is the layout of the new tool that solves the mentioned shortcomings, and a description of its implementation in the Julia programming language, including a complete example of its usage on a specific dataset.

Key words: Machine Learning, Data Preprocessing, Julia Programming Language

Obsah

Úvod

Tato bakalářská práce se zabývá předzpracováním datových sad pro výpočty ve strojovém učení. Cílem práce je vytvořit nástroj, který bude umožňovat stahování a základní úpravu datových sad. Důraz je kladen na jednotný format uložení a načtení dat a na jednoduchost přidávání nových datových sad, aby bylo možné balík snadno rozšiřovat. Tyto dvě vlastnosti mají za cíl co nejvíce usnadnit použití koncovým uživatelům.

První kapitola se věnuje strojovému učení a na jednoduchých příkladech ilustruje vybrané modely strojového učení a také se zabývá tím, jaký typ vstupních dat tyto modely předpokládají.

V následující kapitole jsou představeny typy datových sad. Dále jsou popsány možné formáty těchto dat a možné problémy, které je potřeba zohlednit při stahování a upravování dat. Dále se v této kapitole zabývám rozborem předzpracování dat, které je nutné provést před vložením dat do modelu. Popsáno jsou základní způsoby dělení dat pro trénování modelu a nejčastěji používané typy normalizace dat.

Ve třetí kapitole se nachází popis a porovnání již existujících nástrojů, které se používají ve strojovém učení pro stahování a předzpracování dat. Součástí je také představení programovacího jazyka Julia, v němž je vytvořen balík, jehož vytvoření je hlavním cílem této práce.

Stěžejní část práce je obsažena ve čtvrté kapitole. Jedná se o návrh nového nástroje a především o podrobný popis implementace celého balíku. Popis je strukturován do jednotlivých kroků, které popisují proces přípravy datasetu od stažení, přes předzpracování dat, které se provádí před vložením dat do modelu.

Poslední kapitola obsahuje ucelenou ukázku použití naimplementovaného balíku na konkrétní datové sadě doplněnou o komentáře k použitým funkcím.

Kapitola 1

Strojové učení

Každý den se můžeme v reálném světě setkat s řadou problémů, které je z různých důvodů potřeba řešit. V hledání řešení těchto problémů nám může pomoci matematika. Bohužel většina skutečných problémů je příliš složitá, jsou ovlivněné mnoha faktory a v současné době nejsme schopni je přesně popsat. Z tohoto důvodu se musíme často uchylovat k popisu problému pomocí vybraných měřitelných veličin, které daný problém nejlépe charakterizují. Na základě těchto veličin můžeme poté využít matematické modely k aproximativnímu popisu problému a k jeho řešení. Obor zabývající se touto problematikou se nazývá strojové učení. [1]

Reálných problémů, které lze řešit pomocí metod strojového učení je nepřeberné množství. Pro ilustraci zmiňme alespoň několik příkladů.

- Jednou z nejčastějších motivací pro vyřešení problému pomocí strojového učení je snaha o zvýšení zisku. Například podle analýzy vytíženosti prodejny lze určit optimální distribuci pracovníků u pokladen. Pokladny by neměly být přetížené, aby tak podnik nepřicházel o zisk. Zároveň je žádoucí, aby práce byla efektivní a nedocházelo ke zbytečným prostojům, které jsou pro podnik ztrátou. Též analýzou toho, co lidé nakupují, je možné zvýšit zisk podniku. Je třeba upravovat nabídku zboží a služeb tak, aby poptávka byla co nejvíce spokojená, protože nespokojení zákazníci se do podniku nebudou vracet, a podnik tak opět může tratit.

- Se ziskem úzce souvisí pojem marketing. Z údajů v kolik hodin sleduje televizi nejvíce lidí, lze zvolit nejvhodnější dobu pro reklamu, aby daná společnost získala nejvíce potenciálních zájemců o své produkty.

- Dalším důvodem zkoumání dat je bezpečnost. A to jak kyberbezpečnost, kdy je možné podle textu či způsobu sdílení příspěvků odhalit roboty na sociálních sítích. Či bezpečnost na reálných místech, kdy analýza dat z kamer dokáže odhalit nebezpečného člověka.

1.1 Typy strojového učení

Podle struktury dat pro trénování lze strojové učení rozdělit do dvou hlavních skupin: učení s učitelem a učení bez učitele.

1.1.1 Strojové učení s učitelem

Strojové učení s učitelem, také nazývané *supervised learning*, spočívá v tom, že model obdrží množinu s trénovacími daty

$$\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}, \tag{1.1}$$

kde $\boldsymbol{x}_i \in \mathcal{X}$, kde $\mathcal{X}$ představuje prostor vstupních hodnot, a $y_i \in \mathcal{Y}$, kde $\mathcal{Y}$ představuje množinu možných výstupů, N odpovídá počtu vzorků (anglicky *samples*). Hodnoty $\boldsymbol{x}_i$ představují zjednodušené popisy zkoumaného jevu, tzv. příznaky (anglicky *features*). $\boldsymbol{x}_i$ obsahuje vstupní hodnoty všech příznaků pro i-tý vzorek. Výstupní hodnota y_i odpovídá sledovanému závěru řešené úlohy pro i-tý vzorek. Množina $\mathcal{D}$ obsahuje tedy N dvojic vstupů a výstupů. Na základě těchto správně označených dat z množiny $\mathcal{D}$ se model naučí předpovídat tyto hodnoty u dalších, neoznačených dat. [21]

Mezi často používané algoritmy strojového učení s učitelem patří [12]:

- lineární regrese,
- logistická regrese,
- *random forest* (náhodný les),
- metoda SVM (*support vector machines* neboli metoda podpůrných vektorů).

Tuto skupinu můžeme dále rozdělit podle problému, který je řešen, na klasifikační a regresní úlohu.

Klasifikace

Klasifikace je úloha, kdy se ze vstupních dat snažíme určit, do které z možných tříd daný subjekt patří. Počet tříd musí být konečný a může se jednat o fixní hodnoty nebo to mohou být například slovní označení sledovaných skupin. Vždy můžeme úlohu převést do podoby, kdy každou třídu reprezentujeme pomocí čísla, což znamená, že y_i z rovnice (1.1) je z množiny $\mathcal{Y} = \{1, 2, \ldots, K\}$. V případě, že K, počet tříd, je rovno dvěma, mluvíme o binární klasifikaci. [21]

Jako příklad můžeme uvést úlohu klasifikace kosatců, která je popsána v datasetu Iris. V této úloze má vektor vstupu $\boldsymbol{x}_i$ z definice (1.1) délku čtyři a obsahuje šířku a délku kališních lístků a šířku a délku korunních lístků. Cílem je na základě těchto údajů určit o jaký druh kosatce se jedná. N je rovno 150, tedy celkem obsahuje

tento jednoduchý dataset 150 vzorků. Pro názornost budeme uvažovat jen dva druhy vstupů (šířka, délka korunních lístků) a dvě třídy výstupů. Ukázka tohoto zjednodušeného datasetu je v tabulce 1.1.

petal length	petal width	species
4.7	1.4	versicolor
4.5	1.5	versicolor
4.9	1.5	versicolor
5.2	2.0	virginica
5.4	2.3	virginica
5.1	1.8	virginica

Tabulka 1.1: Ukázka zjednodušeného datasetu Iris

Pro řešení výše uvedeného problému lze použít například logistickou regresi, která je definovaná následovně [13]

$$\begin{aligned} l(w) &= \sum_{i=1}^{N} \Big\{ y_i \log p\left(bmx_i; w\right) + (1 - y_i) \log\left(1 - p\left(bmx_i; w\right)\right) \Big\} \\ &= \sum_{i=1} \Big\{ y_i w^\top bmx_i - \log\left(1 + e^{w^\top bmx_i}\right) \Big\} \end{aligned} \tag{1.2}$$

Provedeme-li minimalizaci této funkce podle $\boldsymbol{w}$

$$\min_{\boldsymbol{w}} \quad l(\boldsymbol{w}), \tag{1.3}$$

získáme vektor vah $\boldsymbol{w}$, který specifikuje oddělující nadrovinu $\boldsymbol{w}^\top x = 0$. Tato nadrovina rozdělí s určitou přesností vzorky na ty, které jsou příslušné jedné nebo druhé třídě.

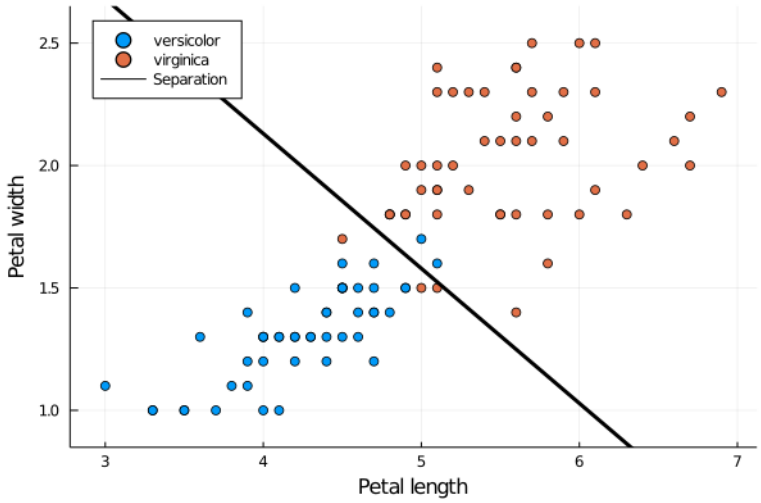
Logistická regrese předpokládá, že originální vstupy upravíme tak, že přidáme před každý vzorek hodnotu jedna. To nám poté zaručí, že budeme mít k oddělující rovině jednak sklon i posun, protože pak bude platit

$$\boldsymbol{w}^\top \hat{\boldsymbol{x}} = w_1 \cdot 1 + w_2 \cdot x_1 + w_3 \cdot x_2, \tag{1.4}$$

kde $\hat{\boldsymbol{x}}$ je nový vektor rozšířený o jedničku na prvním místě a složky x_1, x_2 jsou složky původního vektoru. Na obrázku 1.1 lze vidět, jak jsou data rozdělena pro část datasetu Iris. Všechny vzorky nad nadrovinou jsou vyhodnoceny jako třída virginica a všechny pod jako druh versicolor. Výsledné rozdělení sice není stoprocentně přesné, ale v případě podobných parametrů by stejně tak i člověk nedokázal druh rostliny bezchybně rozlišit. [17]

Regrese

V regresní úloze jsou labely spojité hodnoty, tzn. $\mathcal{Y} = \mathbb{R}$, např. to může být mzda zaměstnance, cena nemovitosti, výška nebo váha člověka. Podíváme-li se opět na Iris



Obrázek 1.1: Rozmístění dvou tříd (virginica, versicolor) datasetu Iris v závislosti šířky korunního lístku na jeho délce. [17]

dataset, můžeme za labely považovat vlastnost délky korunního lístku (petal length) a předpovídat ji v závislosti na šířce korunního lístku (petal width). Předpověď můžeme získat lineární regresí, jejíž model má tvar [17]

$$l(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{w}^\top bmx_i - y_i\right)^2, \tag{1.5}$$

kde vektor bmx_i je rozšířen o 1 stejně jako v případě logistické regrese, N je počet vzorků a $\boldsymbol{w}$ jsou neznámé koeficienty, které získáme opět minimalizací funkce l

$$\min_{\boldsymbol{w}} \quad l(\boldsymbol{w}) \tag{1.6}$$

Na obrázku 1.2 je vidět funkce, jejíž podoba je určena koeficienty z vektoru w. [17]

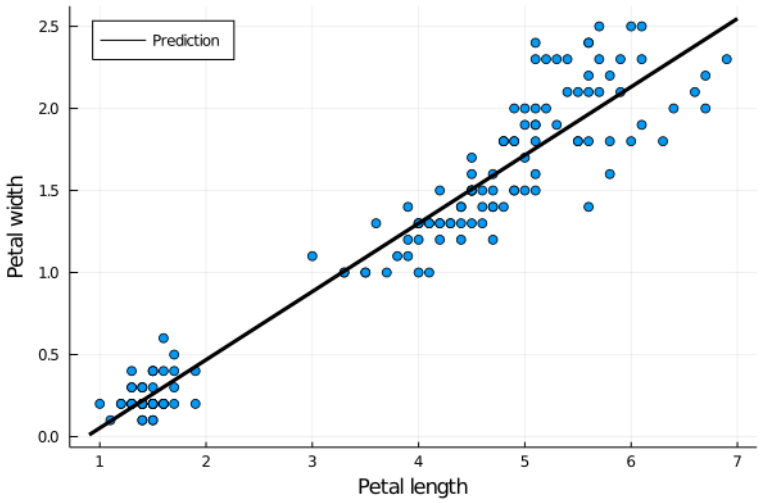
Zmíněné příklady jsou velmi jednoduché a obdobné závěry by bylo možné vyslovit i bez použití matematických modelů. Reálný život ale zahrnuje mnohonásobně komplexnější problémy, ve kterých hledat souvislosti je pro člověka příliš časově náročné nebo dokonce nemožné, například neupravený dataset Iris vyžaduje práci ve 4-dimenzionálním prostoru, který už nejsme schopni vizualizovat.

1.1.2 Strojové učení bez učitele

Druhým přístupem strojového učení je učení bez učitele. V takovém případě model dostane jen neoznačená data.

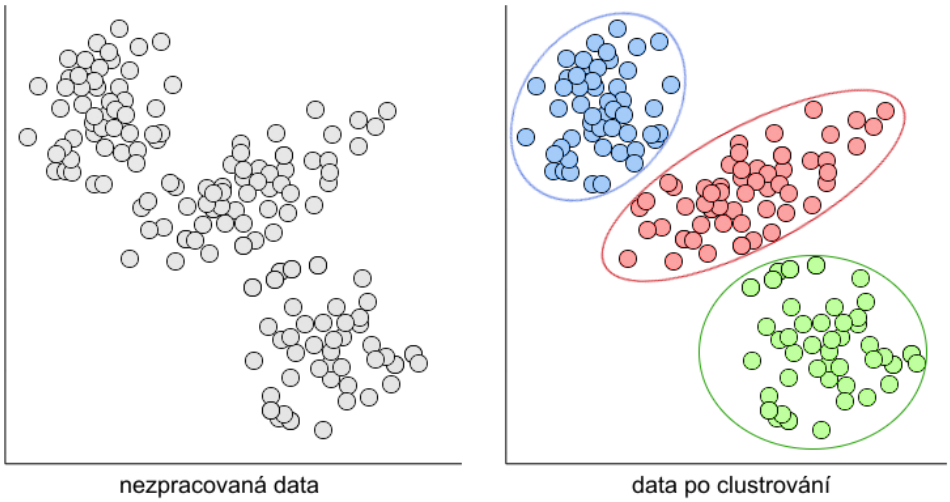
$$\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{N}, \tag{1.7}$$

kde $bmx_i \in \mathcal{X}$ je vektor vstupů a N je počet vzorků. Cílem modelu je pak najít souvislosti a vypozorovat vzorce chování dat. Tento přístup je náročnější pro model, protože neví, na které vlastnosti se má zaměřit, a také není možné využívat k trénování výstupní hodnoty. Výhodou nicméně je, že není potřeba datovou sadu předem ručně, tedy člověkem, popsat. [21]



Obrázek 1.2: Závislosti šířky korunního lístku na jeho délce podle dat z datasetu Iris. [17]

Při učení bez učitele se aplikuje shluková analýza dat neboli clusterování, kdy je cílem rozdělit data do skupin podle vzájemné podobnosti v příznakovém prostoru. Mezi metody shlukové analýzy patří např. metoda nejbližšího souseda nebo k-means metoda. Znázornění aplikace clusterování na neupravená data je na obr. 1.3. [12]



Obrázek 1.3: Shluková analýza

Kapitola 2

Definice problému

Z příkladů uvedených v předchozí kapitole lze vidět, že každý problém závisí na datech. Tato data existují v různých formátech a aby mohla být použita v modelu, je nutné je upravit do vhodné, jednotné podoby. Standardní postup řešení úlohy strojového učení je následující: Nejdříve se data musí prozkoumat a pak se sjednotí jejich formát do požadované podoby a rozdělí se na části. Poté se podle povahy dat a problému vybere model. Tento model se následně na upravených datech natrénuje a vyhodnotí se přesnost jeho předpovědí.

Cílem této práce je vytvořit nástroj, který usnadní první část zmíněného procesu. Toto zpracování by mělo být nezávislé na zařízení, na kterém se data zpracovávají, a stejné i při opakovaném zpracovávání. Navíc přidání vybrané, nové datové sady by mělo pro uživatele co nejjednodušší.

Databází, které poskytují volně ke stažení datové sady popisující nejrůznější reálné problémy, existuje celá řada. Uvedeno je několik příkladů těchto databází:

- Google nabízí vyhledávač Google Dataset Search, který umožňuje nalézt přes 25 milionů různých datasetů, které jsou dostupné na různých úložištích a odkazuje na ně. Vyhledávač se nachází na URL adrese `https://datasetsearch.research.google.com`. [10]
- UCI Machine Learning Repository nabízí přes 500 datasetů, které lze stahovat přímo z URL adresy. Úložiště je dostupné na `http://archive.ics.uci.edu`.
- OpenML je úložiště, které sdružuje přes 21000 datových sad z jiných zdrojů, např. z UCI Machine Learning Repository. URL adresa tohoto úložiště je `https://www.openml.org`.
- Kaggle obsahuje rozsáhlou databázi datasetů, která je pravidelně aktualizována o nové datasety. Datasety lze stáhnout po přihlášení z webových stránek `https://www.kaggle.com` nebo pomocí Kaggle API.

Protože datové sady pocházejí z různých zdrojů není nikde určena jednotná podoba, jak by měla být data předzpracována a v jakém formátu by měla být uložena.

Toto bohužel platí i v rámci jednotlivých databází. Data jsou tak uložená různými způsoby, což je činí nepřehlednými. Při práci s modely strojového učení se k datům přistupuje opakovaně, a tudíž se opakovaně musí provádět počáteční přezkoumání dat, jak vlastně vypadají, a jejich podoba se musí převézt do podoby vhodné pro model. Vzhledem k tomu, že každý dataset je uložen v odlišném formátu, není tento převod jednotný a nelze jej tedy snadno automatizovat. Dokonce je možné, že i tentýž dataset stažený z různých úložišť má jinou podobu, takže při práci s takovým datasetem je nutné odkazovat nejen na jeho jméno, ale i na místo jeho uložení.

Ve zbylé části této kapitoly se budeme zabývat popisem dvou nejpoužívanějších datových formátů a popisem základních metod předzpracování.

2.1 Tabulková data

V případě tabulkových dat jsou vstupy $\boldsymbol{x}_i$ z definice (1.1) vektory, jejichž délka odpovídá počtu příznaků. Label y_i obsahuje jednu položku s označením. Řádky tabulky jsou tvořeny vektory $\boldsymbol{x}_i$ a hodnotou y_i. Sloupce tedy potom obsahují jednotlivé příznaky, popřípadě labely. Problémy v nejednotné reprezentaci uložení tabulkových dat mohou být následující.

1. Rozdíl, který je viditelný ještě před otevřením souboru s daty, je formát souboru. Obvykle se pro tabulková data používá formát CSV, ale můžeme se setkat i s formáty XLS, XLSX, TXT, PDF aj. Navíc mohou být soubory ještě komprimované.

2. Standardní podoba tabulkových dat je, že jednotlivé příznaky jsou sloupce a vzorky jsou řádky, může tomu být ale i naopak. Dále budeme uvažovat první případ.

3. Neexistuje standardní označení sloupce s labely ani jednotná poloha. Ve většině případů se jedná o poslední sloupec, ale bývá to i první anebo jakýkoli jiný.

4. Názvy sloupců neboli hlavička (anglicky *header*) mohou být součástí tabulky s daty, a to obvykle na prvním řádku, nebo mohou být v samostatném souboru.

5. Původní názvy sloupců jsou často komplikované, a tedy je lepší přistupovat k jednotlivým sloupcům ne podle vlastnosti, kterou sloupec obsahuje, ale podle standardizovaného pojmenování sloupce (např. na základě pořadí sloupce).

6. Tabulka s daty může obsahovat chybějící hodnoty, a ty mohou být v původním souboru různě reprezentované. Pro další práci je vhodné sjednotit tuto reprezentaci. Podobně pro hodnoty `true` a `false`, původně mohou být reprezentovány např. pomocí T a F, yes a no nebo Y a N.

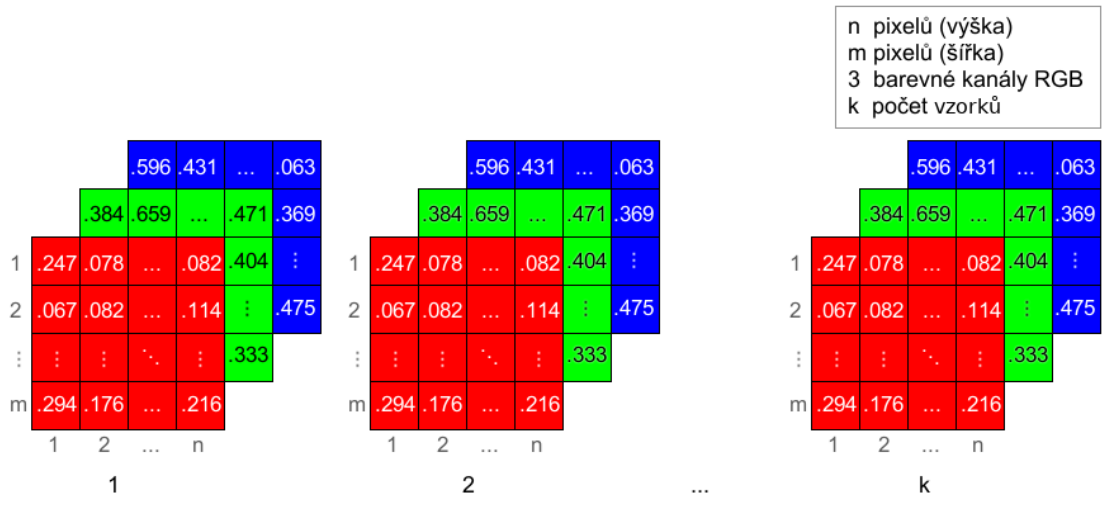
7. Data mohou být obsažená ve více souborech. Například již zmíněné názvy sloupců mohou být v jiném souboru, stejně tak i hodnoty labelů, nebo mohou být rozdělená samotná data do několika částí již před stažením.

2.2 Obrazová data

Obrázky se běžně vyskytují ve formátech jako JPG, PNG, GIF aj. Pokud s nimi chceme pracovat ve strojovém učení, převádí se jejich reprezentace do n-dimenzionálního pole. Tato podoba se může lišit v závislosti na používaném programovacím jazyku a rovněž i podle toho, zda se jedná o barevný nebo černobílý obrázek.

Struktura, kterou chceme po načtení obrazového datasetu získat je v případě černobílých (šedých) obrázků následující: pro vstupy $\boldsymbol{x}_i$ platí, že každý vzorek (jeden obrázek) je matice (dvourozměrné pole) hodnot, kde souřadnice pole odpovídají souřadnicím pixelů obrázku. Hodnoty odpovídají úrovni šedé barvy která je v uložených datech reprezentována jako číslo mezi 0 a 255, ve formátu načtených dat jako vstup do modelu jsou hodnoty převedeny na desetinná čísla mezi 0 a 1. Všechny vzorky dohromady poté tvoří trojrozměrné pole, kde buď poslední nebo první rozměr obsahuje jednotlivé vzorky, a to obvykle v závislosti na programovacím jazyku. My budeme dále pracovat s umístěním v poslední dimenzi.

Reprezentace barevných obrázků je složitější, neboť barvy jsou uložené v několika kanálech. Nejpoužívanější je barevný model RGB, kdy máme tři barevné kanály pro červenou, zelenou a modrou barvu. Všechny vzorky datasetu jsou uložené ve čtyřrozměrném poli, poslední dimenze opět obsahuje jednotlivé vzorky, předposlední (třetí) dimenze obsahuje pro každý obrázek tři barevné kanály reprezentující červenou, zelenou a modrou barvu. V prvních dvou dimenzích jsou pak souřadnice pixelů obsahující úroveň příslušného barevného kanálu. Vstup $\boldsymbol{x}_i$ tedy obsahuje tři matice, jejichž velikost je rovna rozměrům obrázku v pixelech. Znázornění této reprezentace je na obrázku 2.1.



Obrázek 2.1: Reprezentace datasetu s barevnými obrázky ve vícedimenzionálním poli

2.3 Dělení dat

Pri řešení problémů chceme, aby získaný model byl co nejlepší. Abychom byli schopni nezaujatě ohodnotit, jak se model chová, musíme dodržet pravidlo, že trénování a vyhodnocení modelu by mělo vždy probíhat na různých datových množinách. Kdyby tomu tak nebylo, tak by výsledky byly ovlivněné a nejednalo by se o nezaujaté hodnocení modelu. Z tohoto důvodu je vhodné dataset rozdělit na dvě části: trénovací a testovací. Tento postup je analogií postupu trénování modelu v praxi, kdy se model trénuje na datech dostupných v daném okamžiku a predikce se potom provádějí na datech, která v dané chvíli nebyla známá nebo dokonce ještě neexistovala. [4]

Mohou nastat ale i situace, kdy rozdělení datasetu není vhodné. Například je-li dataset příliš malý, rozdělením by výsledná přesnost modelu mohla být nepřiměřeně dobrá či špatná. Dalším důvodem k nerozdělení může být, že jednotlivé třídy nejsou v datové sadě zastoupené v dostatečném množství. Mohlo by se tedy stát, že testovací data budou obsahovat třídy, které v trénovacích datech vůbec zastoupena nejsou, a naopak. Podle toho jak vypadají data se volí různé poměry, ve kterých se data dělí. Neexistuje žádný ideální poměr, ale nejčastěji se trénovací data skládají z přibližně 70 % vzorků a zbylých 30 % je určeno na testování. [4]

Pokud model obsahuje více parametrů tzv. hyperparametrů[1], které je třeba volit ručně, za účelem co nejúspěšnějšího modelu ve svých predikcích, dělí se dataset na tři části, a to kromě dat pro trénování a testování ještě na data pro validaci. Validační set se používá pro výběr nejlepší kombinace hyperparametrů pro testování a případné použití modelu. Při dělení se často postupuje tak, že data se rozdělí nejprve pouze na trénovací a testovací a poté se z trénovacích dat oddělí část pro validaci v požadovaném poměru k trénovacím datům. [17, 5]

Při rozdělení na trénovací, validační a testovací sady se velice často používá náhodné dělení, které by mělo zajistit, ze všechny sady budou obsahovat podobné zastoupení labelů. Náhodný výběr vzorků nezaručuje rovnoměrné rozdělení, přesto se v praxi používá spíš, než náročnější dělení, kdy se data roztřídí podle jednotlivých labelů na skupiny a ty se pak rozdělí v daném poměru. Právě při náhodném výběru vzorků může docházet k tomu, že při opětovném dělení téhož nerozděleného datasetu budou výsledné rozdělené části různé. Sice budou mít při zachování poměru stejnou velikost, ale zastoupení a pořadí prvků bude jiné, a to může vést při práci se stejným modelem k různým výsledkům. Proto je zapotřebí při dělení dat zaručit, aby pořadí rozdělených složek bylo vždy stejné, pomocí nastavení vlastního random seedu[2] pro generování pseudonáhodných čísel.

[1] Hyperparametry jsou proměnné, které určují jak se model bude trénovat. V neuronových sítích to je například počet skrytých vrstev a počet neuronů v těchto vrstvách. [23]

[2] Česky někdy nazývané náhodné semínko.

2.4 Další zpracování dat

Pro předzpracování dat existuje nepřeberné množství technik, jak data upravit. Výběr, jaké techniky použít závisí vždy na konkrétním problému a na zvoleném modelu, který chceme trénovat. V teto sekci představíme několik základních úprav dat, které se používají nejčastěji.

Binarizace labelů

Řada klasifikačních modelů uvažuje pouze binární problémy a pro jejich trénování je potřeba problém více tříd převést na problém binární. Pokud tedy labely obsahují více jak dvě kategorie je možné vybrané labely označit jako pozitivní a zbylé jako negativní.

Normalizace dat

Po rozdělení datasetu bývá vhodné data obsahující číselné hodnoty přeškálovat. V případě reálných dat se může velice snadno stát, že různé příznaky mohou nabývat různých velikostí. Například jeden příznak může mít hodnoty z rozmezí $[0, 1]$ a jiný z $[0, 1^{-5}]$. U těchto dat by značná část modelů předpokládala, že příznaky s malou hodnotou maji menší význam. Abychom tomuto jevu zabránili, je potřeba zaručit, že jednotlivé příznaky budou stejně škálované. Mezi nejčastěji používané normalizace patří standardizace, normalizace euklidovskou normou a min-max škálování. [17, 28]

- **Standardizace**

  Cílem standardizace je zaručit, že střední hodnota přes všechny hodnoty jednoho příznaku bude nulová a rozptyl jednotkový. Toho lze docílit aplikaci následujícího vzorce:

  $$\tilde{x} = \frac{x - \mu}{\sigma}, \tag{2.1}$$

  kde x je vektor hodnot, μ je průměr hodnot, σ směrodatná odchylka.

- **Min-max škálování**

  Min-max škálování zaručuje, že minimální hodnota vektoru příznaku je rovna 0 a maximální 1. Tato normalizace je definována následovně

  $$\tilde{x} = \frac{x - \min(x)}{\max(x) - \min(x)}, \tag{2.2}$$

  kde minimum, resp. maximum je minimální, resp. maximální hodnota ze všech hodnot škálované vlastnosti. [28]

- **Normalizace**

  Cílem normalizace je upravit hodnoty tak, že norma hodnot příznaku je rovna 1. Většinou se používá Euklidova norma, proto se tato normalizace nazývá také L2-normalizace. Je definována jako

  $$\tilde{x} = \frac{x}{\|x\|_2}, \tag{2.3}$$

  tedy hodnoty vektoru podělíme normou vektoru. Euklidova norma se počítá následovně $\|x\|_2 = \sqrt{x_1^2 + \ldots + x_n^2}$.[28]

Kapitola 3

Software

Programovací jazyk Julia byl vytvořen v roce 2012 se zaměřením na matematické výpočty. Jedná se o svobodný a otevřený software[1] pod licencí MIT. Julia kombinuje výhody dalších programovacích jazyků především jednoduchost psaní kódu Pythonu a rychlost jazyka C. [6]

Julia, stejně jako například Python, je dynamicky typovaný jazyk. To znamená, že při práci s proměnnými není třeba definovat jejich typy, na rozdíl od staticky typovaných jazyků (např. C, Fortran), kde je nutné typy definovat. Staticky typované jazyky jsou výkonnější (rychlejší), neboť již předem znají typ dat, tedy způsob, jak s daty pracovat. Na druhou stranu dynamicky typované jazyky jsou přehlednější a „pohodlnější“ pro programátora, protože se nemusí zabývat reprezentací dat v počítači. Julia umožňuje oba dva přístupy. Při vytváření knihoven je nezbytné typy specifikovat, aby se zvýšil výkon a eliminovali se některé případné chyby. Při práci s již existujícími funkcemi z knihoven to už nezbytné není. Julia podporuje uživatelem definované typy, a to nejen složené datové typy, ale i abstraktní a primitivní typy. Výpočty s uživatelem definovanými typy mají srovnatelný výkon se zabudovanými[2] typy. [3, 6, 15, 17]

S typovým systémem souvisí další výhoda Julie tzv. *multiple dispatch*. Slovem funkce se označuje objekt, který souboru argumentů přiřazuje návratovou hodnotu. V Julii funkce obsahují metody, metody pro danou funkci mají stejný název ale liší se argumenty, a to jak počtem, tak jejich typy. Na rozdíl od single dispatche multiple dispatch rozhoduje, kterou metodu vybrat na základě typů všech argumentů. Zatímco single dispatch rozlišuje metody pouze podle typu jednoho parametru. Dynamický dispatch použitý v Julii se rozhoduje až při běhu programu oproti statickému, který potřebuje znát typ argumentů už při kompilaci programu. [3, 15, 17]

Jazyk Julia se rychle rozvíjí a k roku 2021 bylo zaregistrováno přes 4 000 balíků v obecném registru[3]. Další balíky existují i mimo tento registr a jsou dostupné ke stažení z webové adresy, kde jsou uložené. Balíky lze stahovat přímo pomocí správce

[1] anglicky *open source* software

[2] z anglického *built-in*

[3] Obsah obecného registru (anglicky *general registry*) je dostupný na úložišti Git `https://github.com/JuliaRegistries/General`.

balíků v REPLu[4] jazyka Julie a po načtení je hned použít. Při vytváření nového balíku se využívá funkcí z již existujících balíků. [26]

Vzhledem k uvedeným vlastnostem jazyka Julia, které jsou vhodné pro řešení úloh strojového učení, byl právě tento jazyk zvolen pro implementaci praktické části. V následující části této kapitoly jsou představeny existující nástroje pro práci s daty ve strojovém učení.

3.1 Existující nástroje

V této kapitole jsou představeny a porovnány knihovny pro strojové učení, které umožňují mimo jiné stahování a předzpracování datových sad. Ve srovnávání jsem se zaměřila především na ty funkce knihoven, které souvisí s předzpracováním dat.

TensorFlow

TensorFlow je framework určený pro neuronové sítě. Primárně byl navržen pro jazyk Python, ale postupně je propojován do jiných programovacích jazyků, např. jazyky C++, Java, JavaScript, Julia, R, Matlab. TensorFlow obsahuje především nástroje pro vytváření, trénování a vyhodnocování modelů strojového učení, ale také i nástroje pro předzpracování různých typů dat (textová, obrazová, zvuková aj.). [25, 11]

PyTorch

PyTorch je knihovna pro jazyk Python, která se zaměřuje v oblasti strojového učení na práci s neuronovými sítěmi. Nabízí obdobnou funkcionalitu jako již zmíněný TensorFlow. [22]

Scikit-learn

Scikit-learn je rozsáhlá knihovna jazyka Python pro strojové učení. Obsahuje algoritmy a modely používané jak při učení s učitelem, tak bez učitele. Vedle toho jsou součástí i funkce pro předzpracování dat. Scikit-learn umožňuje například normalizaci dat, binarizaci tříd, dělení dat na trénovací a testovací s možností ovlivnit reprodukovatelnost dělení. Pomocí scikit-learn je možné stahovat datasety, které jsou dostupné na úložišti OpenML, o těchto datasetech lze po stažení zjistit dodatečné informace (zdroj, verzi, kontrolní součet atd.). Načítání dat z dalších zdrojů je možné pomocí nástrojů z jiných knihoven Pythonu.[24]

V jazyce Julia byl vytvořen balík ScikitLearn.jl, který používá funkce z knihovny scikit-learn z jazyka Python.

[4]Zkratka REPL znamená *read-eval-print loop* a jde o interaktivní příkazovou řádku, kde napsaný příkaz se hned vyhodnotí.

MLJ

MLJ je balík jazyka Julia, který nabízí nástroje pro práci s modely strojového učení. Balík je z části naimplementovaný v jazyce Julia a z části využívá modely z knihovny Pythonu scikit-learn. Stejně jako tato knihovna MLJ načítá datasety z OpenML. [20]

JuliaML

Prostředí JuliaML obsahuje řadu balíků, které se používají pro strojové učení. Balík MLDatasets.jl se stará o stahování často používaných datasetů pro strojové učení. Obsahuje dva tabulkové datasety, datasety pro modelování jazyka, analýzu textu a především několik obrazových datasetů. Počet datasetů, které tento balík obsahuje, je pouze deset. Pro přidávání nových datových sad není specifikovaný požadovaný formát.[18]

Další balík z prostředí JuliaML MLDataUtils.jl obsahuje funkce pro předzpracování již stažených dat. Obsahuje například funkce pro dělení, náhodné procházení jednotlivých složek dat, normalizaci příznaků, úpravu labelů včetně jejich binarizování.[19]

UCIData

Balík UCIData.jl jazyka Julia umožňuje stahovat vybrané datasety, které jsou k dispozici na úložišti UCI Machine Learning Repository. Poslední verze nabízí ke stažení 139 datasetů. Balík kromě stahování převádí tabulková data (kde řádky jsou vzorky) do formátu, kde poslední sloupec obsahuje labely a první ID. V hlavičce u názvů sloupců je označeno, zda je příznak numerického nebo kategorického typu. Přidání nového tabulkového datasetu (obzvláště ze zmíněného úložiště) je možné, neboť registrace dat má tutéž strukturu pro všechny datasety. Doporučovaný postup pro přidání nové sady, ale popsaný není.

RDatasets

RDatasets je sbírka téměř 1500 datasetů, které byly původně zveřejněny spolu s dalšími statistickými nástroji pro jazyk R. Tyto datasety jsou uloženy ve formátu CSV na úložišti na GitHubu.

V jazyce Julia existuje stejnojmenný balík RDatasets.jl, který umožňuje načítání části těchto datasetů v prostředí tohoto jazyka. [2]

Caret

Balík Classification And Regression Training (caret) používaný v jazyce R obsahuje funkce používané pro zpracování dat a při trénování modelů. Značná část balíku je věnována nástrojům pro trénování modelů, mimo to ale balík umožňuje i dělení

datasetů s možností předem zvolit random seed, centrování a škálování dat. Součástí balíku je několik datasetů, které lze načíst. Načítání jiných než těchto dat musí být provedeno pomocí jiného nástroje. [16]

V tabulce 3.1 je přehled zmíněných datasetů v souvislosti s funkcemi, které jsou očekávány od nástroje pro zpracování dat, jehož vytvoření je cílem této práce.

Tyto požadované nároky pro práci s daty jsou následující:

- stahování datasetů,
- úprava do jednotného formátu,
- dělení dat na trénovací, validační a testovací, které je reprodukovatelné (tedy ve stejném poměru a pořadí při znovupoužití na stejná data),
- normalizace dat,
- binarizace labelů.

knihovna	stahování datasetu	jednotný formát	dělení dat	normalizace	binarizace labelů
TensorFlow	✓	✗	✓	✓	✗
PyTorch	✓	✗	✓	✓	✗
scikit-learn	✓	✗	✓	✓	✓
MLJ	✓	✗	✓	✗	✗
MLDatasets	✓	✗	✗	✗	✗
MLDataUtils	✗	✗	✓	✓	✓
UCIData	✓	✓	✗	✗	✗
RDatasets	✓	✗	✗	✗	✗
caret	✓	✗	✓	✓	✗

Tabulka 3.1: Tabulka porovnání vybraných knihoven pro strojové učení.

Popsané existující nástroje mají širokou funkcionalitu, ale žádný nesplňuje všechny požadované operace pro předzpracování dat, jak je vidět z tabulky 3.1. Navrhovaný balík poskytuje všechny tyto operace. Navíc je balík navržen tak, aby přidání nového datasetu bylo co nejsnazší pro koncového uživatele. Funkce pro strojové učení nejsou u tohoto balíku prioritou, neboť pro výpočty a modelování je již k dispozici řada nástrojů.

Kapitola 4

Implementace

Jak již bylo zmíněno v předchozích kapitolách, cílem práce je vytvořit nástroj, který usnadní zpracování datových sad. Toto zpracování je nutné provést pokaždé před samotnými výpočty a prací s modelem. Zpracovávání je poměrně zdlouhavé, obzvlášť proto, že není automatizované. Navrhovaný nástroj by měl tento proces co nejvíce zkrátit. Především je žádoucí, aby si nástroj pro již jednou zpracovaný dataset pamatoval o něm informace a uživatel je tak nemusel znovu dohledávat. Důraz je kladen na to, aby registrace nového datasetu byla jednoduchá a nevyžadovala mnoho času.

Protože implementovaný balík se nejvíce zaměřuje na předzpracování dat, byl jako jeho název zvolen anglický překlad tohoto účelu – PreprocessData.jl[1].

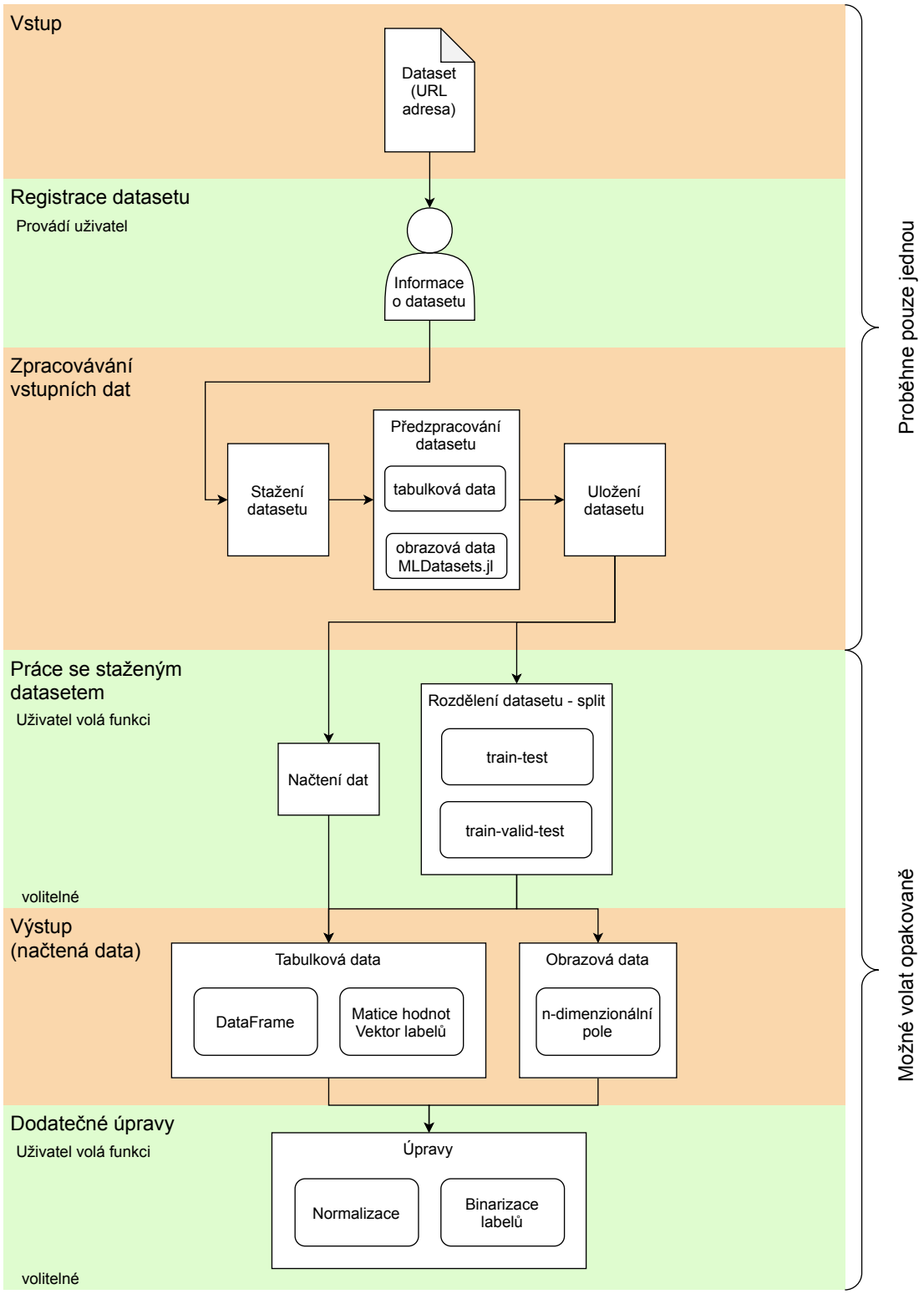
4.1 Návrh balíku PreprocessData.jl

Na obrázku 4.1 je znázorněno schéma popisující průběh zpracování datasetu pomocí tohoto nástroje.

Proces lze rozdělit na dvě části – na část, která pro daný dataset proběhne jen jednou a na funkce, které může uživatel volat opakovaně. Vstupní jednotkou je dataset, který je uložený a volně dostupný ke stažení na URL adrese. Aby balík PreprocessData.jl mohl s datasetem pracovat musí k němu uživatel vytvořit registrační skript se základními údaji. Poté se mohou data již stáhnout a při tomto stažení se upraví do jednotné podoby a uloží se na lokální úložiště. Stažená data může uživatel dále načíst v upravené podobě anebo je rozdělit v požadovaném poměru zavoláním funkce na daný dataset. Na načtená data či na již rozdělená data bude možné aplikovat dodatečné funkce, jako například funkci na normalizaci či binarizaci labelů, viz kapitola 2.

Popis balíku lze analogicky jako na obrázku 4.1 rozdělit na čtyři hlavní části: registrace, zpracování dat, práce se staženým datasetem a dodatečné úpravy dat. Na závěr budou navíc popsány doplňkové funkce balíku.

[1] jl. je přípona souborů obsahujících zdrojový kód jazyka Julia.



Obrázek 4.1: Základní struktura práce s datasetem

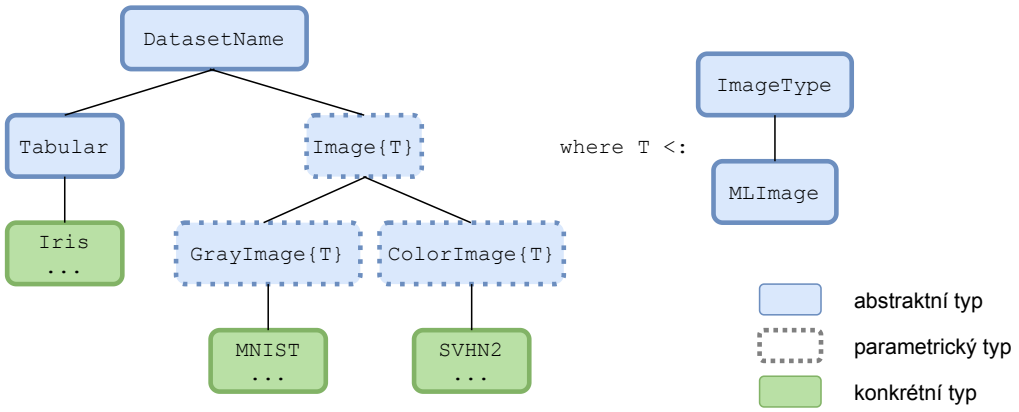
4.2 Registrace datasetu

Uživatel si vybere dataset, který je volně přístupný ke stažení na webových stránkách (příklady, odkud je možné stahovat datové sady, jsou uvedeny v kapitole 2). O tomto vybraném datasetu uživatel zjistí základní informace a uloží je do samostatného souboru (skriptu). Napsání tohoto skriptu zahrnuje pro daný dataset pouze dodefinování krátkých metod, které lze většinou zapsat přehledně do jednoho řádku. Tento soubor je třeba uložit do složky s kódy balíku PreprocessData.jl do příslušné podsložky s již registrovanými datasety `/PreprocessData/src/datasets/`. Název souboru by měl kvůli přehlednosti odpovídat názvu stahovaného datasetu a samozřejmě musí mít příponu .jl. Při dalším novém načtení balíku PreprocessData.jl bude možné tento dataset již stáhnout a pracovat s ním.

Dříve než budou rozebrány jednotlivé kroky přidání nového datasetu, je třeba popsat, jak jsou v balíku datasety reprezentované.

4.2.1 Reprezentace datasetů

Jak bylo zmíněno v kapitole 3 jazyk Julia podporuje multiple dispatch. Z toho důvodu je vhodné k rozlišování metod pro různé datasety využít typový systém. Datasety jsou reprezentovány jako nově vytvořené konkrétní datové typy, které jsou podtypy abstraktních typů. Hierarchie vytvořených typů lze vidět na obrázku 4.2.



Obrázek 4.2: Hierarchie typů pro reprezentaci datasetu

Definovala jsem společného předka pro všechny typy datasetů `DatatsetName`, jehož nadřazeným typem (supertypem) je již pouze typ `Any`. Potomky typu `DatatsetName` jsou abstraktní typy `Image` pro obrazové a `Tabular` pro tabulkové datové sady. V případě obrazových dat existují ještě podtypy pro barevné a šedé obrázky, tedy `ColorImage` a `GrayImage`. Typy reprezentující obrazová data jsou parametrické typy, tzn. ke každému takovému typu je možné specifikovat parametry, a tak vytvářet nové kombinace typů. Důvodem je, že obrazová data mohou být různě uložená a mohou tak potřebovat různá předzpracování, zároveň je ale možné psát metody obecně pro obrazová data (barevná i černobílá). Parametry, které lze vkládat do typů `Image`,

`GrayImage` a `ColorImage` jsou opět typy, které označují druh obrazových dat. Jejich nadřazeným typem je typ `ImageType`, viz obrázek 4.2.

V současné verzi balíku PreprocessData.jl existuje jeden podtyp `MLImage`, který pracuje s datasety z MLDatasets.jl. Balík MLDatasets jsem využila proto, že je v něm naimplementována řada funkcí pro stažení a zpracování vybraných obrazových datasetů. Je výhodnější použít tyto již existující kódy, než vytvářet nové se stejnou funkcionalitou. V případě, že by uživatel potřeboval pracovat s jinými obrazovými datasety, lze snadno vytvořit nový podtyp a k němu dodefinovat metody, které předzpracují data do formátu, který se požaduje v obecných funkcí balíku PreprocessData.jl. Tento postup platí i pro zavedení zcela nového druhu datasetu mimo tabulkové a obrazové.

Jednotlivé datasety jsou reprezentovány jako konkrétní typy, které jsou podtypy příslušného abstraktního typu, který odpovídá druhu datasetu. Na obrázku 4.2 jsou znázorněny zelenými obdélníky s příklady názvů reálných datasetů. Od konkrétních typů lze vytvářet instance, a tak k datasetům přistupovat. Například zavedení datasetu Iris vypadá takto:

```
struct Iris <: Tabular end
```

Kód 4.1: Definice konkrétního typu tabulkového datasetu Iris.

Znak `<:` znamená, že typ `Iris` je podtypem typu `Tabular`. Definice konkrétního typu datasetu se nachází v samostatném souboru (skriptu), v tomto souboru jsou také definovány vlastnosti daného datasetu. Tyto vlastnosti jsou definovány jako metody, jejichž jediným argumentem je instance typu daného datasetu. Když je funkce později volána, Julia díky multiple dispatchi vybere metodu podle typu datasetu, který byl funkci předán. Například počet vzorků by se pro dataset Iris definoval následovně:

```
size(::Iris) = (150, 0, 0)
```

Kód 4.2: Funkce `size` pro dataset Iris.

Pro pojmenovávání nových typů je vhodné dodržet konvenci, že první písmeno je velké a zbytek jsou malá písmena. V případě, že se název datasetu skládá z více slov, slova se bez mezer, podtržítek apod. sloučí a první písmeno každého slova bude velké. Pokud je název zkratkou, je možné použít pro její zápis i více velkých písmen.

Při definování konkrétního typu obrazového datasetu je nutné specifikovat ještě parameter, jak vidíme např. u datasetu MNIST.

```
struct MNIST <: GrayImage{MLImage} end
```

Kód 4.3: Definice konkrétního typu obrazového datasetu MNIST.

4.2.2 Metody pro registraci

Předtím než popíšeme vlastnosti a k nim příslušné metody, které jsou obsahem registračního souboru, je třeba představit nástroj, s jehož pomocí se data stahují.

Stahování dat je v balíku PreprocessData.jl zprostředkováno pomocí balíku DataDeps.jl. Ten umožňuje stahovat soubory, které jsou uložené na webovém serveru a přístupné přes HTTP či HTTPS, do lokálního úložiště. Soubory jsou stahovány do složky datadeps, která se nachází v pracovní složce jazyka Julia pro aktuálního uživatele[2]. V této složce je pro každou sadu dat, která je registrovaná v rámci jednoho registračního souboru, vytvořena podsložka. Jméno podsložky odpovídá návratové hodnotě metody `name` (viz níže) daného datasetu. Do této složky se uloží všechny soubory daného datasetu stažené v balíku PreprocessData.jl.

DataDeps.jl se kromě samotného stažení stará i o to, aby se již jednou stažený soubor nestahoval podruhé. Aby bylo možné soubory stáhnout je potřeba je registrovat, tj. definovat potřebné specifikace. Část údajů je nutné sepsat do registračního bloku, aby se soubor s daty správně stáhl, zbytek údajů je nepovinný, avšak umožňuje další operace. [8]

Povinné údaje v registračním bloku DataDeps:

- Jméno (`name`): tímto jménem bude ke stahovanému souboru (resp. souborům) přistupováno a takový bude název podsložky, která stažená data obsahuje.
- Zpráva (`message`): obvykle obsahuje název datasetu, URL adresu, popřípadě požadavky na citace.
- Cesta (`remote_path`): URL adresa, odkud se má soubor stáhnout.

Nepovinné údaje:

- Kontrolní součet (`checksum`): pro kontrolu, zda se soubory správně stáhly. Řetězec obsahující kontrolní součet, nejčastěji se používá hashovací algoritmus SHA256. Tento údaj sice není povinný, ale pakliže nebude vyplněný, tak se zobrazí varování, které obsahuje kontrolní sumu pro stahovaný soubor.
- Metoda pro načítání (`fetch_method`): předefinováním výchozí metody lze stahovat přes HTTP s autorizací, nebo přes jiný protokol.
- Metoda po načtení (`post_fetch_method`): zde lze definovat jakékoli úpravy na stažených datech, například rozbalení komprimovaných souborů. Defaultně je `post_fetch_method` prázdná.

Aby uživatel, který chce přidat nový dataset nemusel mít přesnou znalost, jak funguje balík DataDeps.jl, vytvořila jsem v balíku PreprocessData.jl funkce, které předávají požadované informace do registračního bloku DataDeps. Stručný přehled se nachází v tabulce 4.1, podrobněji jsou metody popsány dále.

[2]Cestu k pracovní složce Julie lze zjistit vypsáním obsahu konstanty `DEPOT_PATH` v Julia REPLu. Složka obsahuje mimo jiné stažené balíky a k nim příslušné soubory.

metoda	typ návratové hodnoty*	popis
`url`	`String` `Vector{String}`	URL adresa, která odkazuje přímo na soubor s daty.
`checksum`	`String` `Vector{String}`	Kontrolní součet, viz sekce 4.2.2.
`prep`	obecná funkce	Metoda, která zajišťuje úpravu stažených dat. Je to metoda, která se předává metodě `post_fetch_method` z DataDeps.jl, viz 4.2.2. Tato metoda je podrobněji popsána dále.

* V případě, že je uvedeno více typů, možné je použít jeden z nich v závislosti na konkrétních datech. Vektor hodnot se používá v případě, že chce uživatel stáhnout více souborů v rámci jednoho datasetu.

Tabulka 4.1: Tabulka metod používaných v registračním souboru datasetu, které se přímo uplatňují v registračním bloku DataDeps.

Získání hodnoty `name` požadované v DataDeps obstarává stejnojmenná metoda `name`. Tato metoda je nezávislá na typu datasetu, a tak jsem ji definovala pro nadřazený typ všech datasetů `DatasetName`, viz kód 4.4.

```
function name(dataset::DatasetName)
    lowercase(String(nameof(typeof(dataset))))
end
```

Kód 4.4: Definice metody `name`.

Metoda `name` vrací název typu datasetu malými písmeny. V případě, že pro některý typ datasetu, by tento tvar názvu nebyl vhodný, stačí dodefinovat novu metodu pro konkrétní případ, a to klidně i pro nadřazený abstraktní typ, jako je to pro typ datasetu `MLImage`, viz kód 4.5. Názvy obrazových datasetů z MLDatasets.jl jsou totiž zkratky a bývá zvykem, aby zkratky nebyly napsané pouze malými písmeny.

```
function name(dataset::Image{MLImage})
    String(nameof(typeof(dataset)))
end
```

Kód 4.5: Definice metody `name` pro typ `Image{MLImage}`.

Pro předání adresy, odkud se má soubor stáhnout (v DataDeps označená jako `remote_path`), slouží metoda pojmenovaná `url`. Metoda `checksum` je sice nepovinná, ale pokud není vyplněná správně nebo vůbec, tak balík DataDeps.jl vypíše varování se správnou hodnotou kontrolní sumy. Tuto hodnotu lze zkopírovat a následně vložit do registračního souboru. Je to nejčastější způsob při používání DataDeps, jak správně zjistit hodnotu kontrolní sumy.

Metoda `prep` se předává do registračního bloku balíku DataDeps.jl metodě `post_fetch_method`, která povoluje pouze jeden vstupní argument. Tímto argumentem je cesta ke staženému souboru `path`, samotná metoda nevrací žádnou hodnotu a je definovaná jako anonymní funkce. V případě tabulkových dat je v těle této funkce

metoda `preprocess`, která zpracovává CSV tabulková data. Tvar metody vypadá následovně (v závislosti na konkrétním datasetu, zde je uveden dataset Iris):

```
prep(::Iris) = path -> preprocess(path, Iris())
```

Kód 4.6: Obvyklý formát funkce `prep` pro tabulkový dataset.

Pokud je stahovaný soubor komprimovaný, musí být před úpravou extrahován. K extrakci dat poskytuje balík PreprocessData funkci `extract`, která se zavolá na argument `path` v funkci `preprocess`. Jako příklad můžeme uvést dataset Nuclear:

```
prep(::Nuclear) = path -> preprocess(extract(path), Nuclear
    ())
```

Kód 4.7: Funkce `prep` pro komprimovaný dataset.

Metodě `prep` lze ještě předat keyword argumenty, které se dále předávají metodě `CSV.File`. To jsou argumenty, které ovlivňují, jak se soubor načte, například od a do kterého řádku se mají data číst, jaký je oddělovač dat v tabulce, jak jsou zapsané boolovské hodnoty. Seznam možných argumentů lze zjisit v dokumentaci balíku CSV.jl[3] Použití keyword argumentů pro funkci `CSV.File` můžeme ilustrovat na metodě `prep` pro dataset Adult.

```
path -> preprocess(path, Adult(); delim=", ", skipto=2)
```

Kód 4.8: Funkce `prep` s dodatečnými argumenty pro dataset Adult.

Pomocí argumentu `delim` lze specifikovat, jak jsou oddělené buňky tabulky[4]. Argumentem `skipto` se nastavuje od jakého řádku souboru, se data budou načítat. Na začátku souboru mohou být totiž například informace o obsahu datasetu.

Metody `url`, `checksum` a `prep` souvisí přímo s balíkem DataDeps.jl. Dále popsané metody se využívají při zpracovávání dat. Jejich přehled se nachází v tabulce 4.2.

Všechny zmíněné metody jsou vhodné a ve většině případů povinné pro jakýkoli typ datasetu. V případě, že by povinná metoda nebyla pro konkrétní typ datasetu vyplněna, vypíše se chybová hláška. Tato chyba je definována pro nadřazený typ `DatasetName`. Pakliže je definovaná stejnojmenná metoda pro typ, který je podřazený typu `DatasetName`, proběhne konkrétnější metoda díky multiple dispatchi jazyka Julia. Předpisy metod pro obecný typ datasetu se nachází v kódu 4.9.

```
url(dataset::DatasetName) = error("URL address not specified
    for $dataset dataset.")
checksum(::DatasetName) = ""
size(::DatasetName) = error("Size not specified for $dataset
    dataset.")
problem(::DatasetName) = error("Type of problem not
   specified for $dataset dataset.")
message(::DatasetName) = ""
function prep(::DatasetName) end
```

Kód 4.9: Metody registračního souboru pro obecný typ datasetů.

[3]Dokumentace balíku CSV.jl se nachází na adrese `https://csv.juliadata.org/stable/`.

[4]`CSV.File` určuje oddělovač na základě prvních několika řádků, ale v některých případech je lepší oddělující znak, či znaky předem zadat.

metoda	typ návratové hodnoty	popis
`size`	`Tuple{Int, Int, Int}`	Počet vzorků pro trénování, validaci a testování. Obvykle jsou stažená data nerozdělená, v takovém případě je první číslo (data pro trénování) počet všech vzorků.
`problem`	`DataType`*	Druh problému, který se pro daný dataset řeší. Dva typy, klasifikace a regrese, jsou představeny v sekci 1.1.1.
`message`	`String`	Dodatečná zpráva s informacemi o datasetu, například informace o licenci. (nepovinné)

* Pro specifikaci druhu problému jsem vytvořila abstraktní typ `Problem`, který má dva abstraktní podtypy `Classification` a `Regression`. V případě, že by bylo třeba zavést nový druh úlohy, stačí vytvořit další podtyp typu `Problem`.

Tabulka 4.2: Metody registračního souboru, které souvisí se zpracováním datasetu.

Některé typy datasetů mohou vyžadovat další metody, které poskytnou informace nutné k jejich zpracování. Tyto vlastnosti mohou mít smysl pouze pro některé druhy datasetů, a tak nemá smysl je definovat obecně. U tabulkových dat jsou takovou vlastností například názvy sloupců tabulky, nebo umístění sloupce s labely. Pro tabulková data jsem proto zavedla několik metod, jejichž popis je v tabulce 4.3.

V kódu 4.10 je kompletní obsah registračního souboru pro již zmíněný tabulkový dataset Iris.

```
struct Iris <: Tabular end
url(::Iris) = "http://archive.ics.uci.edu/ml/machine-
    learning-databases/iris/iris.data"
checksum(::Iris) = "6
    f608b71a7317216319b4d27b4d9bc84e6abd734eda7872b71a458569e2
     656c0"
prep(::Iris) = path -> preprocess(path, Iris())
target(::Iris) = 5
size(::Iris) = (150, 0, 0)
headers(::Iris) = ["sepal length", "sepal width", "petal
    length", "petal width", "class"]
problem(::Iris) = Classification
```

Kód 4.10: Registrační soubor datasetu Iris.

Vidíme, že každá metoda má argument, jehož typ je typ registrovaného datasetu. Tento argument nemusí mít název, neboť není v metodě dále využíván, ale slouží pro účely multiple dispatche tak, že jazyk Julia vybere pro daný dataset správnou metodu. Na pořadí jednotlivých metod v souboru nezáleží, pouze definice typu musí předcházet metodám.

Pro data z MLDatasets.jl, která jsou reprezentována typem `Image{MLImage}`, není nutné definovat metody potřebné pro balík DataDeps.jl. Přesto je vhodné je dodefinovat, kdyby došlo k jejich zavolání nějakou funkcí, viz kód 4.11.

metoda	typ návratové hodnoty*	popis
`target`	`Int` `DataType`**	Pořadí sloupce, ve kterém se nachází cílové hodnoty (labely).
`headers`	`Int` `Vector{String}`	V této metodě lze specifikovat hlavičku, neboli názvy sloupců, které přísluší k datasetu. Pokud stahovaný soubor obsahuje kromě dat jednotlivých vzorků, také názvy sloupců, návratovou hodnotou této metody je číslo, které odpovídá pořadí řádku s názvy. Pokud názvy sloupců nejsou v souboru, ale jsou známé, je možné je v této metodě předat jako vektor textových řetězců. (nepovinné)
`categorical`	`Int` `Vector{Int}` `Range`	Označení pořadí sloupců, které obsahují kategorické hodnoty. (nepovinné)
`transposed`	`Bool`	Pokud je dataset transponovaný oproti obvyklému uspořádání*** metoda by měla vracet hodnotu `true`, jinak je tato metoda nepovinná.

* V případě, že je uvedeno více typů, možné je použít jakýkoli z nich v závislosti na konkrétních datech.

** Pro případy, kdy se stahuje soubor se samostatnou hlavičkou, nebo labely jsem vytvořila abstraktní typ `File`, který má dva abstraktní podtypy `Header` a `Labels`.

*** Při práci s tabulkovými daty předpokládáme, že řádky jsou vzorky a sloupce jsou vlastnosti.

Tabulka 4.3: Tabulka metod používaných v metodě `preprocess` pro tabulková data.

```
url(dataset::Image{MLImage}) = @info "URL address for $
    dataset can be found in MLDatasets.jl."
checksum(dataset::Image{MLImage}) = @info "Checksum address
    for $dataset can be found in MLDatasets.jl."
```

Kód 4.11: Povinné metody pro datasety typu `MLImage`.

4.2.3 Datová sada s více soubory

V podkapitole 2.1 bylo zmíněno, že jeden dataset může být z různých důvodů rozdělený na více částí. Balík DataDeps.jl umožňuje stahování více souborů v rámci jednoho registračního bloku. Protože v balíku PreprocessData.jl je registrační blok vyplňován pomocí metod, stačí se zaměřit na metody z tabulky 4.1 a na metodu `Target`. Chceme-li stáhnout více souborů nebudou metody z tabulky 4.1 vracet jednu samostatnou hodnotu, ale vektor příslušných hodnot. Délka vektoru odpovídá počtu stahovaných souborů. V `url`, resp. `checksum` se jedná o vektor několika URL adres, resp. kontrolních sum. Metoda `prep` může zůstat stejná jako při stahování jednoho souboru, v takovém případě se na všechny soubory aplikuje stejný postup

předzpracování dat tabulky. Důvodem proč není žádoucí aplikovat na všechny soubory stejnou funkci pro předzpracování může být, že stahovaný soubor obsahuje pouze sloupec labelů, nebo názvy sloupců tabulkového datasetu. Stahujeme-li soubor s labely, funkce `prep` má tvar `path -> preprocess(path, Labels)` a metoda `Target` neobsahuje číslo, ale hodnotu `Labels`. Stahujeme-li soubor s názvy sloupců pak má tvar `path -> preprocess(path, Header)`. Jednotlivé funkce `preprocess` jsou popsány v podkapitole 4.3.1.

Kód 4.12 obsahuje registrační skript pro dataset Gisette. Tento dataset se skládá ze čtyř souborů: soubor s trénovacími daty, soubor s daty pro validaci a dva soubory s příslušnými labely. Metody `url`, `checksum` a `prep` vrací vektor se čtyřmi hodnotami. Protože metoda, je kvůli délce vektoru delší než jednořádková, používá se k její definici dlouhý tvar, tedy `function end` blok namísto rovnítka, které se používá u jednořádkových funkcí[5]. Velmi důležité je upozornit, že na pořadí hodnot ve vektoru záleží. První položce z metody `url` odpovídá první položka z metod `checksum` a `prep`. Zároveň je nutné, aby soubory s labely předcházely souborům se samotnými daty, neboť při zpracování tabulkového datasetu se labely připojují k tabulce a soubor s labely se maže.

4.2.4 Registrované datasety

V současné verzi balíku PreprocessData.jl se nachází deset tabulkových datasetů a pět obrazových datasetů z balíku MLDatasets. Vybrala jsem takové zástupce, aby byla názorně ukázána funkcionalita balíku, takže jsou zastoupeny sady, které jsou například už rozdělené na části pro trénování a testování, které jsou v komprimovaném souboru, které obsahují názvy sloupců, které nikoli atd. Seznam registrovaných datasetů lze snadno rozšiřovat podle potřeb uživatelů. Datasetů je velké množství, a je tak těžké zaručit, aby balík obsahoval potřebný dataset. Z tohoto důvodu byl při implementaci kladen důraz na to, aby bylo přidání nového datasetu jednoduché a uživatel si mohl zaregistrovat datasety, které potřebuje. Přidání sad jiného druhu než jsou tabulková data je možné, nicméně náročnější a zahrnuje v sobě i dopsání několika nových metod pro předzpracování nového typu dat.

V tabulce 4.4 je přehled registrovaných tabulkových datasetů, stručný popis a typ problému, který se v daném datasetu řeší.

Tabulka 4.5 obsahuje seznam registrovaných datasetů z balíku MLDatasets.jl a jejich krátký popis. Všechny tyto datasety řeší problém klasifikace, a tak není nutné tento fakt psát v tabulce pro každý dataset.

[5]Zápis s pomocí `function end` bloku je možné použít i pro jednořádkové funkce, ale vzhledem,k rychlosti a přehlednosti se preferuje zápis s rovnítkem.

```
struct Gisette <: Tabular end
function url(::Gisette)
    [
        "https://archive.ics.uci.edu/ml/machine-learning-
           databases/gisette/gisette_valid.labels",
        "https://archive.ics.uci.edu/ml/machine-learning-
           databases/gisette/GISETTE/gisette_valid.data",
        "https://archive.ics.uci.edu/ml/machine-learning-
           databases/gisette/GISETTE/gisette_train.labels",
        "https://archive.ics.uci.edu/ml/machine-learning-
           databases/gisette/GISETTE/gisette_train.data",
    ]
end

function checksum(::Gisette)
    [
        "a6b857a0448023f033c4dda2ef848714b4be2ae45ce598d088f
        b3efb406e08c5",
        "5cea897956dd172a006132738254a27a8f61ecc1ceb6f5b2063
        9c281d2942254",
        "42bd681fe51b161f033df773df14a0116e492676555ab14616c
        1b72edc054075",
        "6d4c5e998afe67937b9e77a3334e03c85e545ebc65a6eb1333f
        fc14125cfc389"
    ]
end

function prep(::Gisette)
    [
        path -> preprocess(path, Labels),
        path -> preprocess(path, Gisette()),
        path -> preprocess(path, Labels),
        path -> preprocess(path, Gisette())
    ]
end

target(::Gisette) = Labels
size(::Gisette) = (6000, 1000, 0)
problem(::Gisette) = Classification
```

Kód 4.12: Ukázka registračního skriptu se stahováním více souborů v rámci jednoho datasetu.

název	popis	typ problému
Abalone	Dataset s 4 177 vzorky a devíti vlastnostmi, které jsou fyzickým popisem měkkýše abalone. Cílem je předpovědět jeho věk.	klasifikace
Adult	Dataset obsahující přes 40 000 vzorků a který je již rozdělený na data pro trénování a testování. Cílem je předpovědět, zda příjem jedince překročí 50 tisíc dolarů za rok.	klasifikace
Boston	Dataset obsahuje 506 vzorků. Cílem je předpovědět cenu domu na základě informací o domu a jeho lokalitě.	regrese
Car evaluation	Dataset obsahuje 1 728 vzorků. Cílem je určit, zda auto popsané šesti vlastnostmi je v dobrém stavu, či ne.	klasifikace
Forest fires	Dataset obsahuje 517 vzorků s údaji o lokalitách. Cílem je odhadnout plochu, která bude spálena při lesním požáru v dané lokalitě.	regrese
Gisette	Dataset, který je rozdělen na trénovací a validační data (existují i testovací data, ale protože k nim nejsou zveřejněny cílové hodnoty, nejsou zahrnuta v tomto balíku). Stahovaná data mají 7 000 vzorků a každý vzorek má 5 000 vlastností. Cílem datasetu je rozlišit ručně napsané, podobné číslice 4 a 9.	klasifikace
Ionosphere	Dataset s 351 vzorky a 35 příznaky. Cílem je klasifikovat údaje z radaru jako dobré, nebo špatné.	klasifikace
Iris	Malý dataset obsahující 150 vzorků a 4 vlastností. Cílem je rozpoznat jeden ze tří druhů kosatců podle popisu květu.	klasifikace
Nuclear	Malý dataset s pouze 32 vzorky a 11 vlastnostmi. Cílem je předpovědět cenu jaderné elektrárny.	regrese
Wine	Krátký dataset, kde je cílem určit na základě dat z chemické analýzy typ vína.	klasifikace

Tabulka 4.4: Tabulka registrovaných tabulkových datasetů v PreprocessData.jl. [27]

název	popis
MNIST	Dataset obsahující 70 000 šedých obrázků ručně psaných číslic od 0 do 9.
FashionMNIST	Dataset obsahující 70 000 šedých obrázků módy, které pochází od společnosti Zalando.
CIFAR-10	Dataset obsahující 60 000 obrázků z deseti tříd (např. letadlo, kočka, pes). Obrázky jsou rovnoměrně zastoupené, tj. každá třída obsahuje 6 000 zástupců.
CIFAR-100	Dataset se 100 třídami, z nichž každá obsahuje 6 000 obrázků. Třídy jsou navíc ještě seskupeny do dvaceti kategorií. Dataset má tím pádem dva vektory s labely.
SVHN-2*	Jedná se o dataset s více než 630 000 barevnými obrázky, na kterých jsou fotky čísel domů. Vedle datasetu SVHN-2 existuje ještě dataset SVHN. První zmíněný se liší od druhého tím, že obrázky s čísly jsou oříznuté a zmenšené na pevnou velikost 32x32 pixelů. [18]

* Zkratka SVHN znamená The Street View House Numbers.

Tabulka 4.5: Tabulka registrovaných obrazových datasetů v PreprocessData.jl. [18]

Další datasety

Vzhledem k tomu, že reprezentace datasetů je vytvořena pomocí typů, které mají logickou hierarchii, je možné vytvořit nový typ a začlenit ho do hierarchie. Funkce v balíku PreprocessData.jl jsou napsány co nejobecněji (tzn. pro nejnadřazenější typ `DatasetName`) a postupně jsou zkonkrétňovány podle druhu datasetu nebo přímo podle jednotlivých datasetů. Přidání úplně nového typu tedy neznamená nové definování všech metod, ale pouze části metod, které převedou data do určité podoby. V této podobě pak lze data použít v obecných metodách.

Aby byla dodržena podobná struktura metod i pro nově přidané typy datasetů, je vhodné využívat již existující funkce a metody specifikovat pro nové typy.

4.3 Zpracovávání vstupních dat

Proces zpracovávání vstupních dat, lze rozdělit na tři části: stažení, předzpracování a uložení datasetu. Všechny tyto kroky probíhají v rámci jedné funkce z balíku DataDeps.jl, která je volána v metodách balíku PreprocessData.jl.

Data se stahují pomocí DataDeps.jl, který potřebuje ke stažení souboru o něm informace. O registraci datasetu a požadovaných údajích pojednává podkapitola 4.2.

4.3.1 Předzpracování a ukládání dat

Předzpracování dat probíhá ve fázi, kdy balík DataDeps.jl volá metodu `post_fetch_method`. V balíku PreprocessData.jl je této metodě předávána metoda `prep`, obě zmíněné metody jsou představeny v části 4.2.

Tabulková data

Při registraci tabulkového datasetu se musí definovat funkce `prep(::Dataset) = path -> preprocess(path, Dataset())`, kde místo textu `Dataset()` je instance konkrétního typu datasetu. Proměnná `path` obsahuje cestu ke staženému, neupravenému souboru s daty (včetně názvu tohoto souboru). Funkce `preprocess` se pak stará o úpravu dat. Ve funkci `preprocess` je možné nastavit ještě keyword argumenty, které se dále předávají funkci `CSV.File`, jak již bylo popsáno v sekci 4.2.2, tyto argumenty ovlivňují načítání dat ze souboru. Již přednastavené jsou argumenty, díky kterým lze určit, jak mohou vypadat chybějící a boolovské hodnoty:

```
missingstrings = ["", "NA", "?", "*", "#DIV/0!", "missing",
    "NaN"],
truestrings = ["T", "t", "TRUE", "true", "y", "yes", "Y"],
falsestrings = ["F", "f", "FALSE", "false", "n", "no", "N"];
```

Kód 4.13: Možná reprezentace hodnot `true`, `false` a `missing`.

Tabulková data se upravují do podoby, kdy řádky obsahují vzorky a sloupce vlastnosti. Sloupec, který obsahuje cílové hodnoty, je přesunutý na konec tabulky. V prvním řádku tabulky, jsou názvy sloupců ve formátu Column 1, Column 2, atd., poslední sloupec cílových hodnot má název Target. Pokud jsou pro dataset definované sloupce, které mají kategorické hodnoty, k názvům těchto sloupců je připojeno „-C“ jako Categorical. Pokud původní soubor obsahoval názvy sloupců, tak ty se uloží do samostatného souboru s názvem header.csv do stejné složky, kde je uložený soubor s datasetem. Upravený dataset se uloží do nového souboru a původní se smaže.

Aby se k nově uloženým souborům přistupovalo snadněji, sjednotila jsem jejich názvy. Názvy souborů jsou ve formátu `data-typrozdeleni.csv`, kde `typrozdeleni` je nahrazeno jedním ze slov `train`, `test` nebo `valid`. V samotném názvu souboru je tedy již vidět, zda obsahuje data pro trénování, validaci či testování. Pojmenování souboru se děje automaticky na základě původního názvu stahovaného souboru. Pokud původní soubor obsahuje v názvu `valid`, resp. `test`, vyhodnotí se jako data pro validaci, resp. testování. Pokud obsahuje slovo `train` nebo neobsahuje `valid` či `test`, vyhodnotí se jako data pro trénování. Tedy když dataset obsahuje jen jeden soubor s daty, u kterých není určeno, zda jsou již rozdělena, považují se za trénovací data.

Cesta k souborům požadovaného datasetu se pro všechny tabulkové datasety liší pouze názvem podsložky, ve které jsou tyto soubory uloženy. Např. pro dataset Iris je relativní cesta k souboru s trénovacími daty `./datadeps/iris/data-train.csv` a pro dataset Ionosphere `./datadeps/ionosphere/data-train.csv`. Díky tomuto zjednodušení názvu souboru nemusí být pro každý dataset jiná metoda pro načítání

dat, anebo tato metoda alespoň nemusí prohledávat soubory v podsložce datasetu, ale rovnou může požadovat soubor s předem známým názvem.

Stahují-li se labely v samostatném souboru funkce `prep` má podobu `path -> preprocess(path, Labels)`. V takovém případě se soubor stáhne a uloží. Poté, co se labely načtou ve funkci `preprocess` pro soubor s vlastnostmi, se soubor s labely smaže.

Pokud se stahuje soubor s hlavičkou funkce `prep` má podobu `path -> preprocess(path, Header)`. Soubor se uloží pod názvem header.csv. Je třeba, aby názvy sloupců byly v souboru ve formátu, kdy každý název je na samostatném řádku. Tuto úpravu, v případě, že data v takovémto formátu již nejsou, je potřeba udělat ručně.

Obrazová data

Výsledná podoba obrazových dat stažených a zpracovaných balíkem MLDatasets.jl odpovídá popisu v podkapitole 2.2, tj. šedé obrázky jsou reprezentovány 3D polem, barevné 4D polem, kde poslední rozměr obsahuje jednotlivé vzorky. Labely se nachází v odděleném souboru. MLDatasets rovněž využívá ke stahování souboru DataDeps.jl, a tak se podsložky s názvem datasetu nachází ve stejné složce `datadeps`.

4.4 Práce se staženým datasetem

4.4.1 Načítání dat

Data se načítají pomocí funkce `load`. Tato funkce načítá data pomocí funkce `getdata`, která hledá uložený soubor s daty. Pokud data ještě nejsou stažená, spustí se během této funkce stahovací proces.

Konkrétní typy, které reprezentují jednotlivé datasety nejsou exportovány, protože uživatel obvykle nepotřebuje pracovat s větším množstvím datasetů naráz. To znamená, že ve všech funkcích, kde voláme instanci konkrétního typu nějakého datasetu je nutné před jméno instance připojit řetězec `PreprocessData`. Například volání datasetu Iris vypadá takto: `PreprocessData.Iris()`. Důvodem psaní předpony je, že typ není exportovaný. Připsáním názvu balíku určíme, z jakého balíku daný typ (nebo případně metoda v jiných případech) pochází. Psaní předpony může být zdlouhavé, proto existuje způsob, jak se tomu u neexportovaných metod a typů vyhnout. Například pracuje-li se s datasety Iris a Ionosphere, stačí v REPLu jazyka Julia napsat příkaz:

```
using PreprocessData: Iris, Ionosphere
```

Kód 4.14: Dodatečné exportování vyrbaných konkrétních datasetů z balíku PreprocessData.

Tabulková data

Pro načtená tabulková data jsem zvolila dva možné výstupní formáty – matici hodnot a vektor labelů nebo tabulku DataFrame. Typ DataFrame jsem zvolila proto, že umožňuje přehlednou práci s tabulkovými daty. Pochází z balíku DataFrames.jl, který je inspirovaný balíkem pandas z jazyka Python či dplyr z jazyka R. Data se nachází v přehledné tabulce, jejíž sloupce lze libovolně pojmenovat. K položkám v tabulce lze přistupovat pomocí řádkových nebo sloupcových indexů popřípadě názvů sloupců. Balík nabízí funkce pro třídění, dělení a slučování tabulek v závislosti na jejich obsahu. Dále je podporována práce s kategorickými a chybějícími daty. [9]

V případě, že uživatel zvolí jako výstupní formát tabulku, je možné dále určit, zda hlavička této tabulky bude mít automatické názvy sloupců, nebo názvy sloupců ze souboru header.csv, rozdíl v těchto dvou volbách se diskutuje v sekci 4.3.1.

Metodu `load` pro tabulková data jsem vytvořila následovně:

```
function load(
    dataset::Tabular,
    type::Type{<:Split}=Train;
    toarray::Bool=false,
    header::Bool=false,
    )
    return postprocess(dataset, getdata(dataset, type),
        toarray, header)
end
```

Kód 4.15: Definice metody `load`.

Při volání metody `load` je samozřejmě nutné specifikovat, jaký dataset chce uživatel načíst. Dále může zvolit typ dat, které chce načíst, a to buď `Train`, `Test` nebo `Valid`. Hodnota `Train` je předdefinovaná, takže v případě trénovacích dat není nutné typ uvádět. Poté má metoda ještě dva volitelné keyword argumenty. Argument `toarray` umožňuje uživateli rozhodnout, v jakém formátu se data načtou. Pokud je hodnota `toarray` rovna false, pak se načtou jako DataFrame, pokud je rovna true, načtená data budou typu Tuple se dvěma prvky, kde první prvek obsahuje matici s daty a druhý vektor s labely. Pokud je druhý keyword argument `header` roven true a zároveň se data načítají jako DataFrame, použijí se jako názvy sloupců v tabulce názvy ze souboru header.csv. Pokud takový soubor pro daný dataset neexistuje, vypíše se informace, že nebyl nalezen a jako názvy se ponechají názvy automaticky generované při předzpracování datasetu. Pakliže se data načítají ve formátu matice a vektoru, názvy sloupců nemají pro tento tvar žádnou roli. Obsah souboru lze zjistit pomocí funkce `getheader(dataset::Tabular)`, kde argumentem je tabulkový dataset.

Dva typy výstupů – DataFrame a Tuple – lze dodatečně změnit. Převod z DataFramu na pole umožňuje funkce `df_to_array(df::DataFrame; cols::Int=1)`. Prvním argumentem je DataFrame, kterých chceme převést. Druhý argument je volitelný a určuje velikost pole s labely, které se oddělí od matice příznaků, a to směrem od posledního sloupce matice. Jeho hodnota je přednastavena na jedna, a tudíž se oddělí pouze jeden sloupec, tj. vektor s labely. Důvodem, proč může být vhodné oddělit i více sloupců, je například situace, že DataFrame obsahuje i sloupec

s binarizovanými labely. Sloučení matice a vektoru do DataFramu, je možné pomocí funkce `DataFrame` z balíku DataFrames.jl.

4.4.2 Dělení

Dataset je možné rozdělit buď na dvě části – trénovací a testovací, nebo na tři části – pro trénování, validaci a testování. Princip dělení datasetu je vysvětlený v podkapitole 2.3. Podle toho, které rozdělení uživatel chce, použije buď funkci `split_traintest`, nebo `split_trainvalidtest`.

Hlavička funkce `split_traintest` se nachází v kódu 4.16.

```
function split_traintest(
    dataset::DatasetName;
    trainSize::Float64=0.8,
    seed::Int=12345,
    kwargs...
)
```

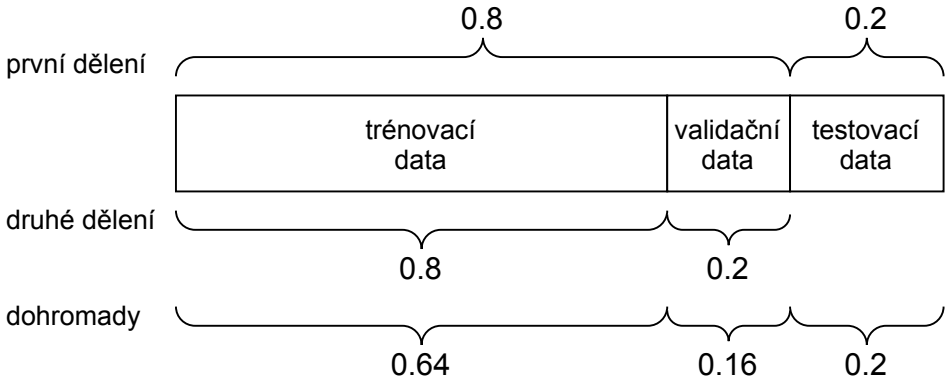
Kód 4.16: Hlavička funkce pro dělení datasetu.

První a jediný povinný argument je název datasetu, tedy jeho konkrétní typ. Keyword argument `trainSize` určuje poměr, ve kterém budou data pro trénování a testování. Jeho hodnota je přednastavena na 0.8, tzn. 80 % vzorků se použije pro trénování a zbylých 20 % pro testování. Hodnota argumentu musí být z intervalu $\langle 0, 1\rangle$, jinak funkce vrátí chybu. Pomocí argumentu `seed` lze nastavit random seed. Argument je přednastavený na hodnotu 12345, takže pokud uživatel `seed` nezmění, stejná data se promíchají vždy do stejného pořadí. Změnou lze docílit jiného náhodného promíchání, a tím pádem i rozdělení. Další keyword argumenty označené v definici jako `kwargs...` jsou stejné jako v případě funkce `load`. Jde o argumenty `toarray` a `header`, jejich význam je popsaný v předchozí sekci o načítání dat 4.4.1.

Funkce, která zajišťuje dělení na tři části `split_trainvalidtest` se svými argumenty od funkce `split_traintest` liší jen přidáním nového argumentu `validSize`, který určuje velikost dat pro validaci. Data pro validaci se oddělují až po rozdělení dat na trénovací a testovací, takže poměr rozdělení udaný `validSize` se počítá až z části, která obsahuje trénovací data. Zvolíme-li například `trainSize` 0.8 a `validSize` 0.2, tak výsledný poměr všech tří částí – trénovací, validační, testovací v tomto pořadí – bude 0.64 ku 0.16 ku 0.2. Příklad tohoto dělení je znázorněn na obrázku 4.3.

Obě funkce dělení jsou závislé na metodě `size` z balíku PreprocessData, která je povinná v registračním souboru pro každý dataset. V této metodě je zaznamenané, zda je dataset již rozdělený, nebo ne. Pokud je dataset již rozdělený tak funkce pro dělení toto rozdělení ponechávají a velikosti `trainSize` a `validSize` se neuplatní. Pakliže je rozdělený dataset jen částečně, například na trénovací a validační data, z trénovacích dat se v poměru specifikovaném v `trainSize` oddělí data pro testování. Obdobně pro ostatní kombinace.

Samotné dělení probíhá tak, že poté, co se zjistí, zda jsou data již aspoň částečně



Obrázek 4.3: Příklad dělení na části datasetu.

rozdělená, se data načtou. Pak se vytvoří pole s čísly od 1 do n, kde n je počet vzorků, které chceme rozdělit. Toto číslo se získává z metody `size` daného datasetu. Poté proběhne náhodná permutace těchto čísel v závislosti na hodnotě argumentu `seed` a tato permutace se rozdělí v zadaném poměru. Výsledkem jsou dvě pole náhodně seřazených čísel. Prvky tohoto pole určují indexy, podle kterých se rozdělí vzorky na dvě části. Pokud se dělí dataset na tří části, tento proces proběhne dvakrát.

Volání funkce `split_trainvalidtest` se všemi možnými keyword argumenty je předvedeno na datasetu Iris v kódu 4.17.

```
train, valid, test = split_trainvalidtest(
    PreprocessData.Iris(),
    trainSize=0.7,
    validSize=0.15,
    header=true,
    toarray=true,
    seed=42
)
```

Kód 4.17: Hlavička funkce pro dělení datasetu na tři části.

Z příkladu mimo jiné vidíme, že na pořadí keyword argumentů nezáleží. Výstupem tohoto volání funkce jsou tři dvouprvkové `Tuply` s názvy `train`, `test`, `valid`, z nichž každý obsahuje na prvním místě matici příznaků a na druhém vektor labelů.

Metody pro dělení jsou univerzální pro jakýkoli typ datasetu. V případě přidání zcela nových typů datasetů je zapotřebí naimplmentovat pouze funkci `getdata`, která vrací data příslušného datasetu a funkci, která vybírá ze skupiny vzorků ty s vybranými indexy.

4.5 Dodatečné úpravy

Na rozdělená nebo pouze načtená data je možné aplikovat ještě funkce, které upravují obsah dat. Tyto funkce umožňují data normalizovat a převést labely na binární klasifikaci. Dodatečné úpravy se na rozdíl od funkce `load` a funkcí pro dělení nea-

plikují na typ datasetu, ale na data, která jsou výstupem těchto funkcí.

4.5.1 Normalizace

Druhy normalizace, které jsem v tomto balíku naimplementovala, jsou podrobně popsány v sekci 2.4. Jedná se o normalizaci podle průmeru a směrodatné odchylky, podle euklidovské normy a o min-max škálování.

Název metody je pro všechny normalizace stejný – `normalize!`. V jazyce Julia platí konvence, že funkce končící vykřičníkem označují funkce, které přímo mění vstupní argument, tedy nevracejí jeho upravenou kopii. Prvním argumentem funkce je typ normalizace, kterou chce uživatel použít. Protože jazyk Julia umožňuje již několikrát zmíněný multiple dispatch, je výhodné vytvořit nové abstraktní typy, které budou sloužit k rozlišení metod pro jednotlivé normalizace, namísto toho, aby se výběr požadované normalizace řídil podmínkami if-else podle hodnoty nějakého stringu s názvem normalizace. Vybrané normalizace jsou reprezentovány abstraktními typy:

- `Std` pro normalizaci pomocí směrodatné odchylky a průměru, neboli standardizaci.
- `MinMax` pro normalizaci podle minimální a maximální hodnoty dat.
- `L2` pro normalizaci podle Euklidovy (L2) normy.

Všechny mají společný nadřazený typ `Normalization`, přidání dalšího druhu normalizace by znamenalo vytvoření nového typu samozřejmě nové metody, kde se vypočtou nové hodnoty.

Normalizace dat se provádí na všechny rozdělené části datasetu, avšak je důležité, aby hodnoty, podle kterých se normalizuje, byly vypočítány jen z části pro trénování. Důvodem je to, že chceme úplně oddělit trénování od testování. Když děláme normalizaci pred trénováním, tak už se jedná o úpravu dat, která patří do procesu trénování, a tedy ta stejná úprava by měla být provedena na testovacích datech. Testovací data nejsou zahrnuta do výpočtu normalizačních konstant, protože při trénování bychom správně neměli mít k testovacím datům přístup. Z tohoto důvodu jsem vytvořila dvě metody pro každý typ normalizování, které se liší počtem argumentů. Celkem tím pádem existuje pro tři typy normalizace šest metod.

Nejprve se zaměříme na standardizaci, ta se určí typem `Std`. Metoda `normalize! (type::Type{Std}, data, mean, std; kwargs...)` je určena pro normalizaci validačních nebo testovacích dat, této funkci je potřeba předat průměr (`mean`) a směrodatnou odchylku (`std`). Tyto hodnoty je ale potřeba vypočítat předem z dat pro trénování pomocí funkce `meanstd`. Tuto funkci `normalize!` lze samozřejmě použít i pro trénovací data. Pro ně ale existuje i další funkce `normalize!(type::Type{Std}, data; kwargs...)`, které není třeba předávat hodnoty `mean` a `std`, neboť se počítají v těle funkce přímo z obdržených dat. Data, které lze funkci předat mohou být typu DataFrame nebo matice. Pokud se jedná o DataFrame není nutné, aby uživatel vybíral jen numerické hodnoty, funkce je naimplementována tak, aby sloupce (včetně

sloupce s labely), které neobsahují pouze čísla byla z normalizace vynechána. V případě, že data jsou matice, rovněž se vynechají nenumerické hodnoty. Samostatný vektor s labely se funkci nepředává.

Pokud by uživatel chtěl použít škálování hodnot pomocí minima a maxima má metoda tvar `normalize!(type::Type{MinMax}, data, min, max; kwargs...)`. Postup je obdobný jako u standardizace, funkce pro získání minima a maxima se jmenuje `minmax`. V případě L2 normalizace má metoda tvar `normalize!(type::Type{L2}, data, norm; kwargs...)`, postup je opět obdobný, jen se předává navíc pouze jeden argument `norm`. Pro jeho získání se použije funkce `l2norm`.

Všechny metody `normalize!` mají ještě volitelný keyword argument `dims`. Tímto argumentem se určí, ve které dimenzi se bude normalizace a příslušné potřebné hodnoty počítat. Hodnota `dims` je přednastavená na 1, takže data jsou normalizovány po sloupcích.

V následujících kódech 4.18 a 4.19 je příklad postupu při normalizování datasetu Iris, který je rozdělen na data pro trénování a testování ve formátu matice s vektorem. Indexem `[1]` vybíráme z Tuplu `train` respektive `test` první položku, tedy matici s příznaky.

```
train, test = split_traintest(PreprocessData.Iris(), toarray
    =true)
normalize!(L2,train[1])
mean, std = meanstd(train[1])
normalize!(L2,test[1], mean, std)
```

Kód 4.18: Ukázka normalizace dat.

Stejný výsledek dostaneme i tímto způsobem:

```
train, test = split_traintest(PreprocessData.Iris(), toarray
    =true)
mean, std = meanstd(train[1])
normalize!(L2,train[1], mean, std)
normalize!(L2,test[1], mean, std)
```

Kód 4.19: Ukázka normalizace dat.

4.5.2 Binarizace labelů

Pokud je řešeným problémem klasifikace, tak binarizací labelů rozumíme převedení většího počtu cílových hodnot na pouze dvě – pozitivní a negativní. Funkci `binarize` uživatel předá seznam labelů (může být i jeden), které chceme považovat za pozitivní, a také v argumentech předá vektor nebo jeden sloupec DataFramu s labely. Seznam možných labelů pro daný dataset lze zjistit zavoláním funkce `classes(dataset::DatasetName)`. Na rozdíl od funkce pro normalizování dat binarizace nemění původní data, ale pouze vrací vektor boolovských hodnot, tedy pozitivní label má hodnotu true (1) a negativní false (0).

Binarizace dat pro dataset Iris se nachází v kódu 4.20.

```
data = load(PreprocessData.Iris())
binarize(data[:,:Target], "Iris-setosa")
```

Kód 4.20: Ukázka binarizace labelů.

K poslednímu sloupci, který obsahuje labely díky sjednocené podobě, můžeme přistoupit pomocí jeho názvu Target.

4.6 Doplňkové funkce

Vedle funkcí, které souvísí se zpracováním nebo stahováním dat, balík PreprocessData.jl obsahuje ještě funkce pro spávu datasetů.

Funkce `listdatasets` vypíše seznam všech registrovaných datasetů. Datasety jsou rozdělené podle typů a pak jsou seřazeny podle abecedy. Jednotlivé typy a příslušné datasety jsou pro přehlednost barevně odlišené a odsazené. U funkce je navíc možné specifikovat typ datasetů, které chce uživatel vypsat. Tyto typy jsou v závislosti na typu datasetu: `Tabular, Image, GrayImage, ColorImage`, jedná se přesně o názvy typů, které se používají při registraci datasetů. Dalším možným parametrem pro vypsání je typ úlohy, kterou řeší, tj. `Classification` nebo `Regression`. V obou případech platí, že pokud bude zaveden nový typ datasetu, nebo úlohy `listdatasets` bude umět zobrazit i datasety tohoto typu. Například všechny datasety řešící regresi získá uživatel takto: `listdatasets(Regression)`.

Funkce `info(dataset::DatasetName)` vypíše základní informace o datasetu. Pro každý typ datasetu vypíše jeho jméno, typ, adresu odkud byl stažený, velikost (počet vzorků ve stažených souborech, viz metoda `size` v sekci 4.2.2), typ řešeného problému, dodatečnou zprávu (obsah metody `message`, viz 4.2.2) a informaci o tom, zda je dataset stažený nebo ne a cestu k podsložce se staženým souborem[6]. V případě tabulkových dat se navíc zobrazuje i původní pozice sloupce s labely `Target column`. Na následujícím příkladu je vidět výstup metody `info(PreprocessData.Boston())`, tedy všechny zmíněné hodnoty pro dataset Boston.

```
PreprocessData.Boston()
    Name:           boston
    Type:           Tabular
    Downloaded:     No
    Source:         https://raw.githubusercontent.com/
       JuliaML/MLDatasets.jl/master/src/BostonHousing/
       boston_housing.csv
    Size:           506 (train data)
                    0 (valid data)
                    0 (test data)
    Problem type:   Regression
    Message:        Boston Housing Dataset
    Target column:  14
```

Kód 4.21: Metoda `info` zavolaná na dataset Boston.

[6]Za stažený se dataset považuje tehdy, existuje-li podsložka s názvem datasetu ve složce datadeps (viz 4.2.2). Samostatnou informaci o stažení datasetu lze získat funkcí `isdownloaded(dataset::DatasetName)`, která vrací true, nebo false.

Další pomocnou funkcí je funkce `remove(dataset::DatasetName)`. Ta umožňuje uživateli smazat celou podsložku datasetu i s obsahem přímo v REPLu jazyka Julie, aby uživatel složku se soubory nemusel hledat a mazat v prohlížeči souborů či v příkazovém řádku.

Každá funkce, včetně neexportovaných funkcí, je popsána přímo v souborech s kódy balíku pomocí tzv. docstring. Tuto stručnou dokumentaci si může uživatel kdykoli dohledat v REPLu jazyka Julie v režimu nápovědy. Tento mód získáme v REPLu napsáním otazníku „?“ a názvu požadované funkce.

Kapitola 5

Práce s balíkem

5.1 Přístup k balíku

Všechny balíky jazyka Julia mají soubory s kódy volně přístupné přes URL adresu na Git repozitáři[1], nejčastěji se používá úložiště GitHub. V takovém případě uživatel nebo organizace, který balík vytváří, založí na svém profilu na GitHubu repozitář, který obsahuje veškeré kódy daného balíku. Pokud se obsah balíku změní, obnoví se obsah Git repozitáře. Balíky jazyka Julia lze instalovat do počítače pomocí správce balíku v juliovském REPLu. Je-li balík v hlavním registru jazyka Julia, lze jej stahovat pouze pomocí názvu, nejsou-li jeho součástí, pak je možné je instalovat právě pomocí odkazu na Git repozitář. Balík PreprocessData.jl je možné nainstalovat na zařízení, kde je instalovaný jazyk Julia, následujícím příkazem v REPLu:

```
(@v1.6) pkg> add https://github.com/GruAna/PreprocessData.jl
```

Kde pravou hranatou závorkou „]“ se přepne uživatele z REPLu jazyka Julie do REPLu správce balíků. Na adrese `https://github.com/GruAna/PreprocessData.jl` se nachází veškeré kódy k balíku PreprocessData.jl.

5.2 Ukázka

V této části se nachází ucelená ukázka použití dostupných funkcí v rámci práce s jedním z registrovaných datasetů, datasetem Abalone. Příkazy, které uživatel zadává do Julia REPLu jsou uvozeny slovem `julia>` .

Výběr datasetu

Nejprve musí uživatel balík PreprocessData načíst.

```
julia> using PreprocessData
```

[1] anglicky *repository*

Předpokládáme, že dataset, se kterým chce uživatel pracovat, je již registrovaný. Pokud si uživatel nepamatuje název přesně, může použít funkci `listdatasets`, která vypíše seznam všech dostupných datasetů.

```
julia> listdatasets()
DatasetName
Image
  ColorImage
    CIFAR10
    CIFAR100
    SVHN2
  GrayImage
    FashionMNIST
    MNIST
Tabular
  Abalone
  Adult
  Boston
  Carevaluation
  Forestfires
  Gisette
  Ionosphere
  Iris
  Nuclear
  Wine
```

Funkce `listdatasets` poskytuje i možnost filtrování datasetů podle typu problému. Například pokud uživatele zajímají pouze klasifikační problémy, může zavolat funkci `listdatasets` a jako argument jí předat typ Classification, čímž funkce vyfiltruje pouze klasifikační datasety.

```
julia> listdatasets(Classification)
Classification datatsets
Abalone (Tabular)
Adult (Tabular)
Carevaluation (Tabular)
Gisette (Tabular)
Ionosphere (Tabular)
Iris (Tabular)
Wine (Tabular)
FashionMNIST (GrayImage)
MNIST (GrayImage)
CIFAR10 (ColorImage)
CIFAR100 (ColorImage)
SVHN2 (ColorImage)
```

V případě, že uživatel potřebuje získat podrobnější informace o nějakém konkretním datasetu, tak může využít funkci `info`. Tato funkce vypíše o daném datasetu doplňující informace, mimo jiné to, zda je dataset již stažený. Tento krok samozřejmě není nezbytný, ale pokud bude dataset nestažený může uživatel takto zjistit, že pro použití tohoto datasetu potřebuje připojení k internetu kvůli stažení dat. Uživatel si pro další práci vybere dataset Abalone.

```
julia> info(PreprocessData.Abalone())

PreprocessData.Abalone()
    Name:           abalone
    Type:           Tabular
    Downloaded:     No
```

```
      Source:          https://archive.ics.uci.edu/ml/machine
         -learning-databases/abalone/abalone.data
      Size:            4177 (train data)
                       0 (valid data)
                       0 (test data)
      Problem type:    Classification
      Target column:   9
```

Při volání funkce `info` byla použita předpona `PreprocessData.` pro přístup k typu `Abalone`. Nyní už víme, že z datasetem budeme dále pracovat, takže je výhodné si typ `Abalone` exportovat dodatečně. Z funkce `info` ještě vidíme, že data nejsou vůbec rozdělená.

```
julia> using PreprocessData: Abalone
```

Načtení dat

Data můžeme načíst pomocí `load`. Protože dataset Abalone není stažený, jak víme z výpisu funkce `info`, proběhne před načtením stažení dat. Při stahování musí uživatel potvrdit, že chce data skutečně stáhnout.

```
julia> data = load(Abalone())
This program has requested access to the data dependency
abalone which is not currently installed. It can be
installed automatically, and you will not see this message
again.

    Dataset: abalone
    Website: https://archive.ics.uci.edu/ml/machine-learning
       -databases/abalone/abalone.data

Do you want to download the dataset from https://archive.ics
.uci.edu/ml/machine-learning-databases/abalone/abalone.data
to "/home/vaclav/.julia/datadeps/abalone"?
[y/n]
y

Info: Downloading
  source = "https://archive.ics.uci.edu/ml/machine-learning-
     databases/abalone/abalone.data"
  dest = "/home/anna/.julia/datadeps/abalone/abalone.data"
  progress = 1.0
  time_taken = "0.33 s"
  time_remaining = "0.0 s"
  average_speed = "573.016 KiB/s"
  downloaded = "187.376 KiB"
  remaining = "0 bytes"
  total = "187.376 KiB"
```

Pro získání představy o datech si zobrazíme obsah proměnné `data`.

```
julia> first(data, 6)
6x9 DataFrame
 Row | Column 1-C  Column 2  Column 3  Column 4  Column 5
     | String      Float64   Float64   Float64   Float64
-----------------------------------------------------------
   1 | M               0.455     0.365     0.095     0.514
   2 | M               0.35      0.265     0.09      0.2255
```

```
   3 | F               0.53       0.42       0.135     0.677
   4 | M               0.44       0.365      0.125     0.516
   5 | I               0.33       0.255      0.08      0.205
   6 | I               0.425      0.3        0.095     0.3515
                                                4 columns omitted

julia> first(data[:, (end-1):end], 6)
6x2 DataFrame
 Row | Column 8  Target
     | Float64   Int64
------------------------
   1 |     0.15       15
   2 |     0.07        7
   3 |     0.21        9
   4 |     0.155      10
   5 |     0.055       7
   6 |     0.12        8
```

Tabulka datasetu je příliš velká, aby se celá zobrazila na obrazovce, ale je možné si nechat vypsat jen část tabulky (např. poslední dva sloupce). Z ukázky vidíme, že dataset obsahuje sloupce s kategorickými i číselnými hodnotami. Během stahování došlo i k přejmenování sloupců. Jak je vidět, první kategorický sloupce má za univerzálním názvem dodané „-C“ a poslední sloupec obsahuje labely a má název Target.

Data lze načíst i jako dvě matice namísto DataFramu. V takovém případě navolíme hodnotu argumentu `toarray` na true.

```
julia> features, labels = load(Abalone(), toarray=true)
(Any["M" 0.455 … 0.101 0.15; "M" 0.35 … 0.0485 0.07; … ; "F"
0.625 … 0.261 0.296; "M" 0.71 … 0.3765 0.495], [15, 7, 9,
10, 7, 8, 20, 16, 9, 19  …  9, 8, 10, 10, 8, 11, 10, 9, 10,
12])

julia> features
4177x8 Matrix{Any}:
 "M"  0.455  0.365  0.095  0.514   0.2245  0.101   0.15
 "M"  0.35   0.265  0.09   0.2255  0.0995  0.0485  0.07
 "F"  0.53   0.42   0.135  0.677   0.2565  0.1415  0.21
 :                                 :
 "F"  0.625  0.485  0.15   1.0945  0.531   0.261   0.296
 "M"  0.71   0.555  0.195  1.9485  0.9455  0.3765  0.495

julia> labels
 4177-element Vector{Int64}:
  15
   7
   :
  10
  12
```

V následujících případech budeme pracovat s DataFramem, ale všechny funkce lze aplikovat i na pole.

Dělení dat

Data nyní můžeme rozdělit, uživatel si vybere dělení na dvě části (trénovací a testovací). Použije funkci `split_traintest`. V případě, že chceme použít originální hlavičku, tak nastavíme argument `header` na hodnotu true. (Hlavičku, nebo informaci zda vůbec existuje lze zjistit zavoláním funkce `getheader`.)

```
julia> getheader(Abalone())
9-element Vector{String}:
 "Sex"
 "Length"
 "Diameter"
 "Height"
 "Whole weight"
 "Shucked weight"
 "Viscera weight"
 "Shell weight"
 "Rings"
```

Rozdělená data budou uložena do proměnných s názvy `train` a `test`.

```
julia> train, test = split_traintest(Abalone(), header=true)
Info: Dataset abalone has only train data. Separating
test data from train data.
(3342x9 DataFrame
  Row | Sex     Length   Diameter  Height   Whole weight  ...
      | String  Float64  Float64   Float64  Float64       ...
------------------------------------------------------------
    1 | F         0.585     0.46     0.17          1.0835  ...
    2 | M         0.275      0.2     0.08           0.099  ...
  :   |   :         :         :        :              :
 3341 | F         0.565     0.505    0.21           1.2765  ...
 3342 | M         0.45      0.335    0.125          0.349   ...
                           4 columns and 3338 rows omitted,

835x9 DataFrame
  Row | Sex     Length   Diameter  Height   Whole weight  ...
      | String  Float64  Float64   Float64  Float64       ...
------------------------------------------------------------
   1 | I         0.62      0.45     0.135         0.924    ...
   2 | F         0.67      0.505    0.175         1.0145   ...
  :  |   :         :         :        :              :
 834 | I         0.38      0.275    0.095         0.2505   ...
 835 | F         0.505     0.405    0.18          0.606    ...
                             3 columns and 831 rows omitted)
```

Při porovnání výstupů z funkce `load` a `split_traintest` vidíme, že řádky tabulky nejsou ve stejném pořadí, a že k výběru řádků byla použita náhodná permutace indexů.

Dodatečné úpravy načtených dat

Nyní lze rozdělená data normalizovat, např. normalizací podle průměru a směrodatné odchylky.

```
julia> mean,std = meanstd(train);

julia> normalize!(Std, train)

julia> normalize!(Std, test, mean, std)

julia> train
3342x9 DataFrame
  Row | Sex     Length      Diameter    Height    ...
      | String  Float64     Float64     Float64   ...
------------------------------------------------------
    1 | F         0.509051    0.527426    0.713696 ...
    2 | M        -2.05009    -2.07126    -1.38881  ...
  :   |   :          :           :           :
 3341 | F         0.343945    0.977198    1.64814  ...
 3342 | M        -0.605412   -0.721941   -0.337555 ...
                     5 columns and 3338 rows omitted

julia> test
835x9 DataFrame
 Row | Sex     Length      Diameter    Height    ...
     | String  Float64     Float64     Float64   ...
-----------------------------------------------------
   1 | I         0.797985    0.427477   -0.103944 ...
   2 | F         1.21075     0.977198    0.830501 ...
  :  |   :          :           :           :
 834 | I        -1.18328    -1.32164    -1.03839  ...
 835 | F        -0.151371   -0.0222957   0.947307 ...
                     5 columns and 831 rows omitted
```

Opět můžeme porovnat s původními hodnotami a zjistíme, že data byla úspěšně normalizována.

Dále může uživatel převést tuto klasifikační úlohu na binární klasifikaci. Nejprve je vhodné si zjistit, jaké labely jsou v datasetu zastoupeny a z těch vybrat ty, které budou považovány za pozitivní a binarizovat sloupec s labely.

```
julia> classes(Abalone())
28-element Vector{Int64}:
  1
  2
  3
  4
  :
 25
 26
 27
 29
```

Můžeme se podívat do dokumentace k funkci `binarize`, abychom se ujistili, v jakém pořadí má funkce argumenty, a zda se vkládá celá tabulka, či jen sloupec s labely.

```
help?> binarize
search: binarize

  binarize(data, pos_labels)

  Binarize data for specified positive labels. (Data must
  contain just labels no feature valeus.)
```

Jako pozitivní labely budeme uvažovat kategorie, které jsou označené číslem menším než 10. Takže budeme vkládat vektor obsahující čísla 1 až 9, k tomu můžeme použít jednu ze základních funkcí jazyka Julia `collect`.

```
julia> bin_train=binarize(train[:, end], collect(1:9))
3342-element Vector{Bool}:
 0
 1
 0
 0
 :
 1
 1
 0
 0

julia> bin_test=binarize(test[:, end], collect(1:9))
835-element Vector{Bool}:
 0
 0
 0
 0
 :
 1
 0
 1
 0
```

Nyní tyto vektory můžeme připojit k příslušným DataFramům s daty.

```
julia> train.Binarized = bin_train;

julia> train[:, end-3:end]
3342x4 DataFrame
  Row | Viscera weight  Shell weight  Rings  Binarized
      | Float64         Float64       Int64  Bool
------------------------------------------------------
    1 |       1.32514       0.621851      14      false
    2 |      -1.42         -1.492          5       true
    3 |       0.238903      0.10951       10      false
  :   |        :              :           :        :
 3341 |       0.89792       0.83682       12      false
 3342 |      -0.679175     -0.882927      10      false
                                    3337 rows omitted
```

Zobrazením posledních čtyř sloupců DataFramu `train` můžeme zkontrolovat připojení binarizovaných labelů.

Smazání nepoužívaných datasetů

V případě, že už uživatel daný dataset nechce používat a potřebuje uvolnit místo na disku, je možné využít funkci `remove`, která smaže veškerá uložená data související s daným datasetem. Pro kontrolu, zda je soubor nestažený, lze použít funkci `isdownloaded`.

```
julia> remove(Abalone())

julia> isdownloaded(Abalone())
false
```

Závěr

Cílem práce bylo vytvořit nový balík na stahování a předzpracování dat pro programovací jazyk Julia. Tento balík PreprocessData.jl je možné stáhnout z volně dostupného úložiště Git pomocí správce balíků jazyka Julia. Stažený balík je pak po načtení možné aktivně používat přímo v interaktivním REPLu či využívat jeho funkce ve vlastních skriptech.

V rámci vytváření návrhu nového nástroje jsem se seznámila se základními modely strojového učení a s typy dat, se kterými se v těchto modelech pracuje. Rovněž jsem popsala i metody předzpracování, které se na tyto data nejčastěji aplikují. Vybrala jsem zástupce těchto typů dat a na jejich základě jsem navrhla nástroj, který umožňuje snadné stažení, načtení a předzpracování dat. Dále jsem se seznámila s programovacím jazykem Julia, který byl pro své vlastnosti zvolen jako jazyk pro vytvoření nástroje. V tomto jazyce jsem navržený nástroj naimplementovala.

Prozkoumala jsem existující nástroje, které se nejčastěji používají ve strojovém učení a nabízí funkce pro stahování a předzpracování datových sad. Porovnáním funkcionality těchto nástrojů jsem zjistila, že žádný nesplňuje současně úplně všechny požadavky, které jsou kladeny na nový nástroj. Představené nástroje umožňují ve většině případů stahovat vybrané datové sady, nicméně tato data obvykle nemají po načtení jednotný formát. Jejich další nevýhodou je, že popsané knihovny nejsou navržené tak, aby přidání nového datasetu do množiny stažitelných datových sad bylo pro uživatele jednoduché. Kvůli nejednotnosti načteného formátu dat rovněž není možné pro nově přidaný dataset používat funkce daného nástroje. Ve vytvořeném balíku PreprocessData.jl jsem se zaměřila na odstranění zmíněných nedostatků u již existujících nástrojů. Vzhledem ke struktuře balíku, má uživatel možnost přidávat nové typy datových sad ke stažení a zpracování pomocí několika povinných funkcí k práci s daty.

Balík PreprocessData.jl poskytuje funkce pro stahování a načítání dat, rovněž i pro dělení dat, které je nutné provést před vložením dat do modelu. Dále nabízí i funkce umožňující různé způsoby normalizování dat a také funkci, která převede řešení klasifikační úlohy na binární klasifikaci. Součástí balíku jsou mimo funkcí, které pracují přímo s daty, i funkce, díky kterým uživatel získá přehled o již registrovaných datasetech a o jejich vlastnostech. Každá naimplementovaná funkce balíku PreprocessData.jl je doplněná o stručný popis své funkcionality, který je možné získat v nápovědě jazyka Julia.

Struktura balíku je navržena tak, aby přidávání nových typů datových sad bylo co nejjednodušší a aby již implementované metody byly použitelné i pro nové typy.

Díky této struktuře je balík možné neustále dál rozšiřovat nejen o nové typy datových sad, ale i o další možnosti předzpracování. Samotný balík obsahuje několik předpřipravených datasetů s různými specifickými vlastnostmi. Tito vybraní zástupci datových sad slouží jako názorná ukázka toho, co obnáší registrace nových datových sad a jak se s registrovanými daty v balíku pracuje.

Literatura

[1] ALPAYDIN, Ethem. *Machine Learning: The New AI*. MIT press, 2016. ISBN 978-026-2529-518 [cit. 2021-04-27].

[2] AREL-BUNDOCK, Vincent. *A collection of datasets originally distrubuted in R packages* [online]. [cit. 2021-06-04].
Dostupné z: https://github.com/vincentarelbundock/Rdatasets

[3] BEZANSON, Jeff, EDELMAN, Alan, KARPINSKI, Stefan and SHAN, Viral B., *Julia: A fresh approach to numerical computing* [online]. 19. 7. 2020. Dostupné z: https://arxiv.org/abs/1411.1607

[4] BROWNLEE, Jason. *Train-Test Split for Evaluating Machine Learning Algorithms*. In *Machine Learning Mastery* [online]. 26. 8. 2020 [cit. 2020-05-05]. Dostupné z: https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/

[5] BROWNLEE, Jason. *What is the Difference Between Test and Validation Datasets?*. In *Machine Learning Mastery* [online]. 26. 8. 2020 [cit. 2020-05-05]. Dostupné z: https://machinelearningmastery.com/difference-test-validation-datasets/

[6] CABUTTO, Tyler A. and HEENEY, Sean P. and AULT, Shaun V. and MAO, Guifen and WANG, Jin. *An Overview of the Julia Programming Language* [online]. Association for Computing Machinery, 2018. ISBN 9781450365406. Dostupné z: https://doi.org/10.1145/3277104.3277119

[7] Co je strojové učení?. In: *Oracle Česká Republika* [online]. [cit. 2021-04-27]. Dostupné z: https://www.oracle.com/cz/data-science/machine-learning/what-is-machine-learning/

[8] *DataDeps Documentation* [online]. [cit. 2021-05-25]. Dostupné z: https://www.oxinabox.net/DataDeps.jl/stable/

[9] *DataFrames.jl* [online]. [cit. 2021-06-07]. Dostupné z: https://dataframes.juliadata.org/stable/

[10] *Dataset Search: Oficiální webová stránka* [online]. [cit. 2021-05-21]. Dostupné z: https://datasetsearch.research.google.com/

[11] DUBOVNIKOV, Kirill. *PyTorch vs TensorFlow — spotting the difference*. [online] 20. 6. 2017 [cit. 2021-06-24]. Dostupné z: https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b

[12] FUMO, David. *Types of Machine Learning Algorithms You Should Know* [online]. 15. 6. 2017 [cit. 2021-05-21]. Dostupné z: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861

[13] HASTIE, T., TIBSHIRANI, R. and FRIEDMAN, J. H.. *The elements of statistical learning: data mining, inference, and prediction*. 2nd edition. New York: Springer, 2009. [cit. 2021-06-29]. ISBN 978-0-387-84857-0. Dostupné z: https://web.stanford.edu/ hastie/Papers/ESLII.pdf

[14] HONZÍK, Petr. *Strojové učení* [online]. Brno: FEKT Vysoké učení technické, 2006 [cit. 2021-04-27]. Dostupné z: http://midas.uamt.feec.vutbr.cz/STU/others/Honzik - Strojove_uceni_S.pdf

[15] Julia Documentation - The Julia Language [online]. Dostupné z: https://docs.julialang.org/en/v1/

[16] KUHN, Max. *The caret Package* [online]. [cit. 2021-06-04]. Dostupné z: https://topepo.github.io/caret/

[17] MÁCHA, Václav a ADAM, Lukáš. *Julia for Machine Learning* [online]. 2021. Dostupné z: https://vaclavmacha.github.io/JuliaCourse/stable/

[18] *MLDatasets.jl's Documentation* [online]. [cit. 2021-06-05]. Dostupné z: https://juliaml.github.io/MLDatasets.jl/stable/

[19] *MLDataUtils.jl's documentation* [online]. [cit. 2021-06-05]. Dostupné z: https://mldatautilsjl.readthedocs.io/en/latest/

[20] *MLJ* [online]. [cit. 2021-06-05]. Dostupné z: https://alan-turing-institute.github.io/MLJ.jl/stable/

[21] MURPHY, Kevin P. *Machine learning: a probabilistic perspective* [online]. MIT press, 2012. [cit. 2021-04-30].

[22] *PyTorch Documentation* [online]. [cit. 2021-06-24]. Dostupné z: https://pytorch.org/docs/stable/index.html

[23] RADHAKRISHNAN, Pranoy. *What are Hyperparameters ? and How to tune the Hyperparameters in a Deep Neural Network?* [online]. 9. 8. 2017 [cit. 2020-05-06] Dostupné z: https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a

[24] *scikit-learn: Machine Learning in Python* [online]. [cit. 2021-06-05]. Dostupné z: https://scikit-learn.org/stable/index.html

[25] *TensorFlow* [online]. [cit. 2021-06-24]. Dostupné z: https://www.tensorflow.org/

[26] *The Julia Programming Language* [online]. [cit. 2021-06-06]. Dostupné z: https://julialang.org/

[27] *UCI Machine Learning Repository* [online]. [cit. 2021-06-18]. Dostupné z: https://archive.ics.uci.edu/ml/index.php

[28] ZHENG, Alice a CASARI, Amanda *Feature Engineering for Machine Learning*. O'Reilly Media, Inc., 2018. ISBN 978-1-491-95324-2 [cit. 2021-05-13].

Příloha A

Obsah přiloženého CD

BP_Gruberova.pdf – soubor s elektronickou verzí této bakalářské práce.

PreprocessData – složka se zdrojovými kódy naimplementovaného balíku PreprocessData.jl.