Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Aplikace Machine learningu pro získávání dat z dokumentů

Adéla Šolarová

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Adéla Šolarová

Informatika

Název práce

Aplikace Machine learningu pro získávání dat z dokumentů

Název anglicky

Machine learning application for obtaining data from documents

---

Cíle práce

Hlavním cílem bakalářské práce je navrhnout program automatizující proces čtení konkrétních informací z definované množiny dokumentů.
Dílčí cíle práce jsou:
- zpracovat literární rešerši na téma Machine learning se zaměřením na oblasti významné z hlediska získávání dat z dokumentů,
- navrhnout a popsat Machine learning algoritmus schopný extrahovat definovaná data z definované množiny dokumentů,
- implementovat algoritmus v jazyce Python pro vybraný typ dokumentů.

Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Praktická část je zaměřena na vývoj programu v jazyce Python, schopného přečíst konkrétní informace z definované množiny dokumentů. Na základě syntézy teoretických poznatků a výsledků praktické části budou formulovány závěry bakalářské práce.

Doporučený rozsah práce

30-40 stran

Klíčová slova

python, PDF, machine learning, analýza textu, text mining, klasifikace dokumentů, document intelligence

---

Doporučené zdroje informací

AHADH, Abdhul, Govind Vallabhasseri BINISH a Rajagopalan SRINIVASAN. Text mining of accident reports using semi-supervised keyword extraction and topic modeling. Process Safety and Environmental Protection [online]. 2021, 155, 455-465 [cit. 2022-04-26]. ISSN 09575820. Dostupné z: doi:10.1016/j.psep.2021.09.022

BUI, Duy Duc An, Guilherme DEL FIOL a Siddhartha JONNALAGADDA. PDF text classification to leverage information extraction from publication reports. Journal of Biomedical Informatics [online]. 2016, 61, 141-148 [cit. 2022-04-26]. ISSN 15320464. Dostupné z: doi:10.1016/j.jbi.2016.03.026

RASCHKA, Sebastian, MIRJALILI, Vahid. Python Machine Learning – Third Edition. Birmingham: Packt Publishing, 2019. 770 s. ISBN 978-1-78995-575-0.

VIRIUS, Miroslav. Základy algoritmizace. Vydání druhé. Praha: České vysoké učení technické, 2008. ISBN 978-80-01-04003-4.

Předběžný termín obhajoby

2022/23 LS – PEF

Vedoucí práce

doc. Ing. Jan Tyrychtr, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 31. 10. 2022

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 25. 11. 2022

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci ”Aplikace Machine learningu pro získávání dat z dokumentů” jsem vypracovala samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autorka uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušila autorská práva třetích osob.

V Praze dne 15.3.2023 _____

Poděkování

Ráda bych touto cestou poděkovala svému vedoucímu doc. Ing. Janu Tyrychtrovi, Ph.D. za příkladné vedení práce a jeho bleskovou komunikaci. Dále bych ráda poděkovala institucím Státní ústav radiační ochrany a Státní úřad pro jadernou bezpečnost za povzbuzení věnovat se danému tématu a za svolení použít k testování programu reálné protokoly z měření na rentgenových zařízeních. V neposlední řadě děkuji také své rodině za podporu při studiu.

Aplikace Machine learningu pro získávání dat z dokumentů

Abstrakt

Tato práce se zabývá vytvořením programu, schopného automatizovat proces čtení určitých předem definovaných informací z velkého množství strukturovaných PDF dokumentů. Velký důraz je kladen na klasifikaci dokumentů, z důvodu definování malé skupiny vzájemně si velmi podobných dokumentů a následného usnadnění identifikace informací z této jasně vymezené skupiny dokumentů. Pro testování programu byly vybrány protokoly z měření na rentgenových zařízeních. Byly porovnány dvě klasifikační metody učení bez učitele, konkrétně K-Means a DBSCAN. Obě vybrané klasifikační metody byly shledány vhodnými pro aplikaci na třízení vybraného typu dokumentů. Dále program využívá metodu učení s učitelem, konkrétně rozhodovacího stromu, pro nalezení potřebných informací. Program byl aplikován na jednu vybranou třídu protokolů a čtení informací bylo testováno na třech typech textových informací. Bylo zjištěno, že pokud se potřebná informace nachází v dokumentu na podobném místě, je metoda rozhodovacího stromu relativně vhodná, ovšem pro čtení informací, které se v dokumentech nachází pokaždé na jiném místě, nemusí být tato metoda příliš efektivní.

Klíčová slova: python, PDF, machine learning, analýza textu, text mining, klasifikace dokumentů, document intelligence

Machine learning application for obtaining data from documents

Abstract

This work presents a software program designed to automate the process of extracting predefined data from a large set of structured PDF documents. The focus is on document classification, which aims to group together very similar documents, thus streamlining the information retrieval process. The chosen document type for the study is measurement protocols for X-ray devices. Two unsupervised learning methods, namely K-Means and DBSCAN, were evaluated for their effectiveness in document classification. Both methods were found to be suitable for sorting the target document type. A supervised learning method, specifically decision tree, was employed for reading the predefined data. The program was tested on one selected protocol class, and the performance was evaluated for three different types of textual information. The results indicate that the decision tree method is relatively effective when the predefined information is consistently located in a similar position in every document. However, for data located in different positions within the document, this method may not be as effective.

Keywords: python, PDF, machine learning, text analysis, text mining, document classification, document intelligence

Obsah

1 Úvod

S rozvojem počítačů a informačních technologií se dnešní doba nevyhnutelně dostala do bodu, kdy jsou stále častěji kladeny různé nároky na digitalizaci dat. Tato skutečnost otevřela zcela nové možnosti dostupnosti informací. Enormní množství shromážděných dat je však vysoce neefektivní analyzovat manuálně a proto se stále častěji setkáváme s různými formami automatizace procesu jejich analyzování, třídění a získávání cílených informací. Díky rozvoji umělé inteligence začaly vznikat rovněž různé její podoblasti zabývající se právě hromadnou analýzou velkého objemu dat. Vývoj technologií umělé inteligence výrazně přispěl ke zjednodušení a zefektivnění mnoha náročných procesů spojených nejen s datovou analýzou.

Tato práce se zabývá využitím jedné z oblastí umělé inteligence, konkrétně strojového učení, k automatizaci hromadného zpracování digitalizovaných dat v podobě PDF dokumentů. Účelem navrženého programu je zjednodušení práce s daty v jednom z výzkumných ústavů. Tento ústav má přístup k velkému množství protokolů z měření ve formátu PDF, které obsahují cenné informace z hlediska porovnávání a zpracovávání statistických průzkumů. Protože manuální dohledávání dat v jednotlivých protokolech zabírá enormní množství času, zůstává velký potenciál dostupných informací nevyužit. Cílem navrženého programu je navrhnout možný způsob automatizace tohoto procesu čtení potřebných dat s co možná nejmenší nutnou interakcí ze strany uživatele a získaná data uložit ve formě snadno čitelné pro pracovníky ústavu.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem bakalářské práce je navrhnout program automatizující proces čtení konkrétních informací z definované množiny dokumentů.
Dílčí cíle práce jsou:

- zpracovat literární rešerši na téma Machine learning se zaměřením na oblasti významné z hlediska získávání dat z dokumentů,
- navrhnout a popsat Machine learning algoritmus schopný extrahovat definovaná data z definované množiny dokumentů,
- implementovat algoritmus v jazyce Python pro vybraný typ dokumentů.

2.2 Metodika

Teoretická část práce se opírá především o studium odborné literatury a slouží jako rešerše řešené problematiky se zaměřením na algoritmy a metody použité v praktické části práce. Pro vyhledávání odborných zdrojů byla využita rovněž vědecká databáze ScienceDirect s využitím různých kombinací klíčových slov ”python”, ”machine learning”, ”text mining”, ”PDF”, ”document classification” a další.

Praktická část je zaměřena na vývoj programu schopného přečíst konkrétní informace z definované množiny PDF dokumentů. Program je psán v jazyce Python za použití knihovny sklearn. Velká pozornost byla věnována přípravě dat, především klasifikaci dokumentů do malých velmi podobných skupin. Pro tento účel byly porovnány dvě klasifikační metody dostupné v knihovně sklearn, K-Means a DBSCAN (kapitola 3.2.4.2). Data byla následně čtena z jedné vybrané skupiny protokolů a byla k tomu využita metoda rozhodovacího stromu (kapitola 3.2.4.1). Kromě knihovny sklearn byly dále použity python knihovny os, sys, shutil, io, numpy, pandas, csv, matplotlib, filecmp a pdfminer.
Pro vývojové diagramy je použita následující anotace:

- **šipka** - směr zpracování
- **oddélník se zaoblenými rohy** - počátek, nebo ukončení zpracování
- **kosočtverec** - větvení postupu na základě splnění podmínky
- **obdélník** - dílčí krok zpracování (funkce, nebo skupina funkcí vykonávající určitý úkol)
- **obdélník s vlnovkou na spodní straně** - soubor, případně skupina souborů

Závěry bakalářské práce jsou formulovány na základě syntézy teoretických poznatků a výsledků praktické části.

3 Teoretická část práce

3.1 Umělá inteligence

Umělá inteligence (Artificial Intelligence, AI) je obor informatiky zabývající se tvorbou systémů schopných napodobovat některé lidské dovednosti, například schopnost vnímat, chápat, plánovat, rozhodovat se a učit se [1].

3.1.1 Zaměření umělé inteligence

Disciplíny AI se v současnoti zaměřují především na následující prvky lidské inteligence: učení, uvažování, řešení problémů, vnímání a používání jazyka [2].

3.1.1.1 Učení

Schopnost učit se je aplikována v nejrůznějších formách AI. Nejjednodušším příkladem učení je tzv. rote learning (mechanické učení), neboli učení metodou pokus - omyl. Princip rote learningu je, že program nahodile provádí určitou akci, dokud nedospěje k požadovanému výsledku. Jakmile uspěje, zapamatuje si jak k výsledku dospěl, aby při příští příležitosti již nemusel nahodile hádat, ale mohl postupovat cíleně. Náročnější z hlediska učení je metoda generalizace. Cílem této metody je, aby systém dokázal využít předchozích zkušeností na nové problémy podobného charakteru. Na rozdíl od rote learningu by tedy systém měl být schopen najít odpověď i na problém, se kterým se dosud explicitně nesetkal tím, že aplikuje řešení dle nějakého předchozího vzoru. [2]

3.1.1.2 Uvažování

U současných AI systémů rozlišujeme dva hlavní typy uvažování, deduktivní a induktivní [2]:

- **Deduktivní** - charakteristické pro deduktivní uvažování je, že pravdivost vstupů garantuje pravdivost výstupu. Například: *Pan Novák může být pouze v práci, nebo doma. Doma není, to znamená, že musí být v práci*. S deduktivním uvažováním se setkáváme v matematice a logice, kde je na základě několika málo axiomů[1] vybudován rozsáhlý a nevyvratitelný systém vět (výroků, teorémů).
- **Induktivní** - v případě induktivního uvažování je zahrnut element nejistoty, tedy že pravdivost vstupů do jisté míry podporuje pravdivost výstupu, ale bez žádné garance. Například: *Předchozí nehody podobného charakteru byly způsobeny selháním lidského*

[1] Axiom je tvrzení, které je předem pokládáno za pravdivé [3].

faktoru, proto i tato nehoda byla zapříčiněna selháním lidského faktoru. S induktivním uvažováním se můžeme setkat ve vědě, kdy jsou na základě shromážděných dat vytvářeny modely popisující a předpovídající další chování. Jakmile se vyskytnou anomální data, musí být vytvořený model upraven.

Ve vývoji modelů uvažování v současné AI bylo dosaženo signifikantních pokroků, nicméně k opravdovému uvažování nestačí pouze vyvozovat závěry, ale vyvozovat závěry relevantní ve vztahu ke konkrétní situaci. V tomto ohledu se uvažování řadí mezi nejsložitější problematiky moderní AI. [2]

3.1.1.3 Řešení problémů

V souvislosti s AI můžeme řešení problémů definovat jako systematické prohledávání množiny možných akcí, aby bylo dosaženo nějakého předem definovaného cíle, nebo řešení [2]. Řešení problémů dělíme na speciální a obecné [2]:

- **Speciální (special purpose)** - je šité na míru pro konkrétní problém a může tedy využívat velmi specifické charakteristiky situace, do níž je problém zasazen.
- **Obecné (general purpose)** - je aplikováno na široký rozptyl různých typů problémů.

Příkladem řešení problémů může být volba vítězného tahu, nebo sekvence tahů v deskové hře, nebo vytvoření matematického důkazu [2].

3.1.1.4 Vnímání

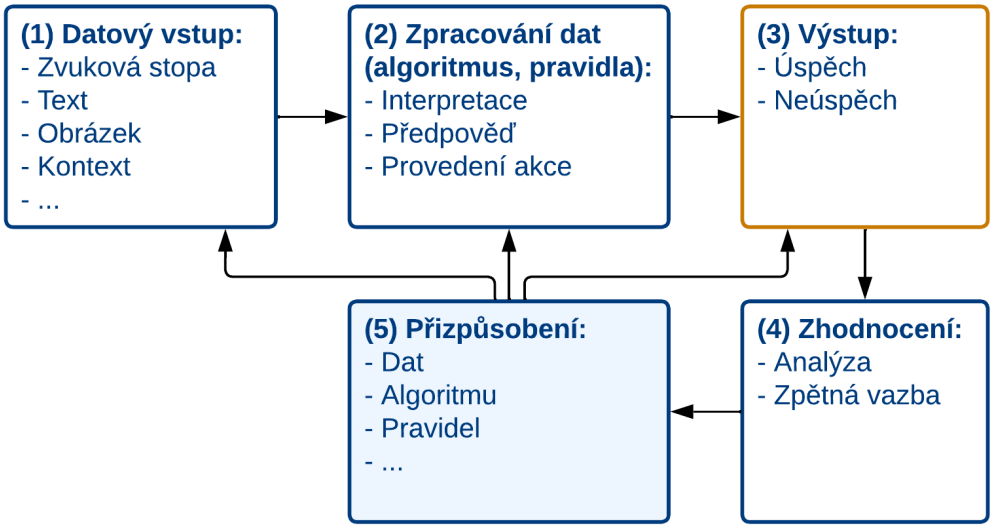
Jedná se o schopnost snímat okolní prostředí pomocí různých smyslových receptorů - nejčastěji obraz a zvuk. Analýza obrazu je zkomplikována faktem, že se určitý objekt může jevit odlišně v závislosti na úhlu, z něhož je pozorován, a na osvětlení. Rozpoznávání okolních objektů může být u AI systémů využito v oblastech jako rozpoznávání tváří, řízení autonomních vozidel, nebo navigování robota po budově a sbírání definovaných předmětů. [2]

3.1.1.5 Používání jazyka

Poslední oblastí, na níž se vývoj AI zaměřuje, je schopnost interpretovat a používat lidskou řeč [2]. Jazyk může být nejen mluvený projev, nebo souvislý text, ale také množina symbolů a znaků s určitým významem dle společenské konvence (například výstražné symboly, nebo dopravní značení) [2]. Specifikum lidského jazyka oproti pouhému souboru značení je však jeho schopnost generovat neomezené množství vět a formulovat libovolnou myšlenku [2]. Vědecká komunita se neshoduje v názorech, co znamená, že počítač ”rozumí lidské řeči” [2]. Ačkoliv počítač teoreticky dokáže komunikovat na úrovni rodilého mluvčího, sám o sobě

není schopen chápat význam jednotlivých vět, pouze spojuje knihovnu slov a pravidel do souvislé řeči (argument čínského pokoje) [4].

3.1.2 Základní princip fungování umělé inteligence



Obrázek 1: Základní ukázka fungování AI (upraveno podle [1]).

Na obrázku 1 je popsán základní princip fungování AI. (1) Vstupními daty pro AI může být například záznam řeči, text, obrazová data, apod. (2) AI následně zpracovává velké objemy vstupních dat a prostřednictvím souboru algoritmů a pravidel data interpretuje, vytváří předpovědi a přijímá rozhodnutí. (3) Výstupem každého cyklu je buď úspěch - předpověď se povedla, bylo přijato správné rozhodnutí, nebo neúspěch - předpověď se nepovedla, výstup je chybný. (4) Na základě výstupů provede AI zhodnocení a (5) navrhne změny procesních algoritmů, pravidel, atd. pro zlepšení výsledků v dalším kroku cyklu. Nový soubor upravených algoritmů a pravidel znovu aplikuje na vstupní data a cyklus opakuje, dokud není dosaženo požadovaných výsledků. [1]

3.1.3 Některé důležité podoblasi umělé inteligence

Podoblastí a aplikací AI je nepřeberné množství, jmenujeme alespoň některé nejdůležitější:

- **Machine learning (Strojové učení)** - algoritmy a techniky AI schopné automatického učení a vylepšování se na základě souboru předchozích zkušeností, bez potřeby explicitního programování [1].
- **Data mining (Vytěžování dat)** - metody filtrování, třídění a klasifikace dat z větších datových celků, jejichž účelem je odhalení méně patrných vzorců a souvislostí mezi jednotlivými daty [1].

- **Robotika** - konstrukce robotů [1].
- **Neuronové sítě** - výpočetní model AI, simulující chování struktury neuronů v lidském mozku [1].
- **Computer vision (Počítačové vidění)** - algoritmy AI schopné interpretovat obrazová data (grafy, tabulky, obrázky, videa) [1].
- **Multiagentní systémy** - simulace prostředí, v němž se nachází několik samostatných agentů, z nichž každý má nějaké vlastní cíle a funkce [5].
- **Natural language processing (Zpracování přirozeného jazyka)** - nástroj AI pro pochopení, rozpoznání, interpretování a vytváření přirozeného lidského jazyka a řeči [1].

3.1.4 Stručný historický přehled vývoje umělé inteligence

Zrod myšlenky umělé inteligence

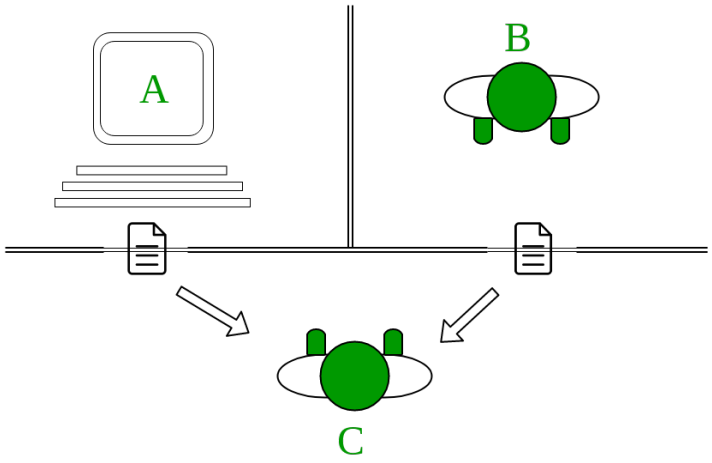
Koncept strojů, schopných inteligentního uvažování, někdy dokonce natolik inteligentního, že stroje převezmou plnou kontrolu nad světem, člověka odjakživa fascinoval a první vize umělé inteligence a jejích možností se neobjevily nikde jinde než v literatuře. Už v řecké mytologii se objevují zmínky o bronzovém mechanickém obru jménem Talós, který střežil ostrov Krétu [6]. V průběhu 20. století se myšlenka myslících strojů a robotů stále více rozvíjela [7]. Slavný autor sci-fi literatury Isaac Asimov představil kolem roku 1950 v souboru povídek Já, robot slavné Tři zákony robotiky, shrnující etiku chování AI ve vztahu k člověku a základní požadavky na budoucí vývoj a využívání AI [8].

V polovině 20. století zároveň došlo k přechodu od pouhých nápadů a představ v beletrii k reálný debatám ve vědeckých kruzích [7]. Myšlenkou na skutečné vytvoření umělé sítě neuronů, simulující některé funkce skutečného lidského mozku, se začala zaobírat stále širší komunita vědců, matematiků a filosofů, jedním z nichž byl například i britský matematik a kryptoanalytik Alan Turing [7]. Turing v roce 1950 ve své práci Computing Machinery and Intelligence koncipoval principy a požadavky pro postavení reálného inteligentního stroje a způsoby jak otestovat jeho inteligenci [2], [9].

Turingův test

Základní otázkou, kterou se Turing ve své práci zabýval bylo, zda je stroj schopen myslet a jak definovat hranici kdy řekneme, že je již stroj schopen myslet [9]. Tuto problematiku nastínil pomocí hry s názvem Imitation Game, která později vešla ve známost jako Turingův test (obrázek 2). Principem testu je, že jedna osoba (obrázek 2 - C) pokládá otázky dvěma neznámým respondentům, z nichž jeden je stroj (obrázek 2 - A) a druhý člověk (obrázek 2

- B) [9]. Jako hranici, kdy již můžeme tvrdit, že je stroj schopen myslet, definoval Turing okamžik, kdy osoba pokládající otázky (obrázek 2 - C) již nebude schopna rozlišit, který z respondentů je stroj a který člověk [9].



Obrázek 2: Grafické znázornění Turingova testu: A a B představují respondenty - počítač a člověka, C osobu pokládající otázky ([10]).

Vývoj počítačů

V době kdy Turing přišel s konkrétními koncepty AI však za myšlenkovými pochody vědecké komunity silně zaostával vývoj počítačů. Tehdejší počítače se řadily do tzv. první generace počítačů a dokázaly pouze provést konkrétní pokyny, ale již nebyly schopné tyto pokyny ukládat [7], [11]. Navíc provoz tehdejších počítačů byl finančně náročný a přístup k nim si mohly dovolit pouze velké firmy, nebo prestižní univerzity [7]. V období od padesátých let začaly vznikat první počítače druhé generace, vybavené prvními tranzistorovými klopnými obvody a feritovými paměťmi [11]. Počítače druhé generace disponovaly oproti počítačům první generace vyšší operační rychlostí, vlastní řídící jednotkou a nebyly tak finančně náročné na provoz, což umožnilo další rozvoj AI [11].

Zrod prvních programů

Prvními programy aplikující principy AI byly programy simulující různé hry. Roku 1951 napsal Christopher Strachey, někdejší Turingův spolužák a později ředitel výzkumné skupiny Programming Research Group na Oxfordské univerzitě, první program hrající klasickou hru dáma. Roku 1952 byl Anthony Oettingerem z Cambridgeské univerzity představen program s názvem Shopper, který se stal první demonstrací strojového učení (Machine Learning, ML). Hra simulovala prostředí nákupního domu s osmi obchody a postavu nakupujícího. Jakmile dostal nakupující instrukci, kterou položku má zakoupit, začal náhodně procházet obchody a hledal danou položku v nabídce jednotlivých obchodů. Při každé návštěvě si zároveň nakupující zapamatoval několik náhodných jiných položek z nabídky daného obchodu, takže

pokud při další instruktáži dostal za úkol nakoupit stejnou položku, nebo některou, kterou již lokalizoval, nehledal již náhodně, ale šel cíleně přímo do správného obchodu. [2]

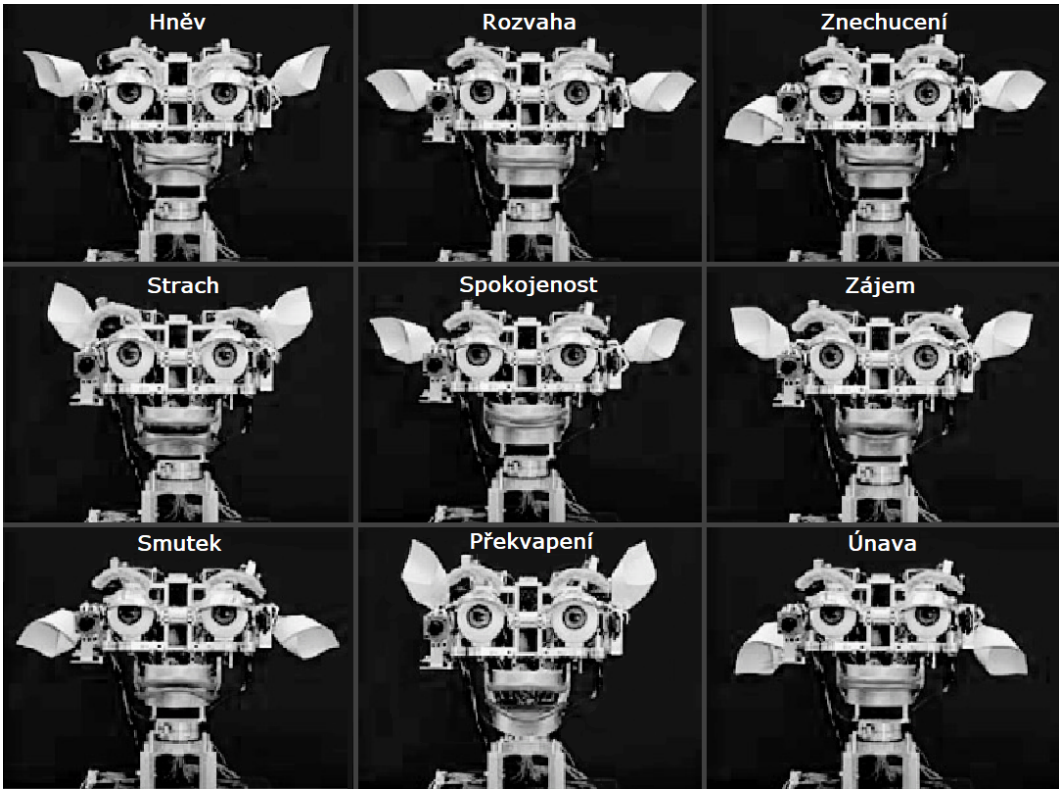
Logic Theorist

Každý nový projekt či koncept je ve svém vývoji do určité míry závislý na finanční podpoře a dostupných zdrojích. Vzhledem k tomu, že celý koncept AI existoval převážně ve fantaskní literatuře a myšlenkových konstruktech, bylo pro přilákání pozornosti investorů třeba dokázat, že je koncept AI realizovatelný. Významným milníkem se proto stala konference Dartmouth Summer Research Project on Artificial Intelligence, na níž byl roku 1956 představen program Logic Theorist. Program, jehož autory byly Allen Newell, Cliff Shaw a Herbert Simon, napodoboval schopnosti člověka řešit problémy a je mnohými považován za první program umělé inteligence. Tato událost nejen prokázala, že je myšlenka zrodu AI realizovatelná, ale rovněž přitáhla pozornost mnoha autorit oboru a přispěla k významnému rozvoji AI v následujících dvaceti letech. [7]

Rozvoj AI ve 20. století

V následujícím období docházelo k prudkému rozvoji počítačů a souběžnému rozvoji AI. Počítače dokázaly uchovávat větší množství informací a pokynů, byly stále rychlejší, levnější a dostupnější [7]. Docházelo k rozvoji vyšších programovacích jazyků a s nimi i k vylepšování algoritmů strojového učení [7]. Vznikalo mnoho slibných projektů, z nichž některé přesvědčily i vládní instituce, aby investovaly do dalšího vývoje AI [7]. Mezi takové projekty patřil například General Problem Solver (Simons a Newell, 1972), který byl schopen řešit dobře definované problémy jako dokazování logických, či geometrických teorémů, řešení slovních úloh a šachů [7], [12]. Dalším takovým projektem byl například projekt ELIZA (Weizenbaum, 1966), chatovací bot, schopný vést s uživatelem primitivní konverzaci [7], [13]. Jednalo se o raný příklad zpracování přirozeného jazyka (Natural language processing) [13].

V osmdesátých letech 20. století již existovaly počítače čtvrté generace a k rozvoji AI rovněž přispělo rozšíření nástrojů algoritmizace a další finanční investice [7], [11]. Byly představeny první techniky hlubokého učení, neboli deep learningu (Hopfield, Rumelhart), které umožňovaly počítačům učit se na základě předchozích zkušeností, a vznikl první expertní systém (Feigenbaum) - program, který shromažďoval odpovědi na všechny možné myslitelné otázky od expertů v oboru, na základě čehož byl pak schopen již samostatně odpovídat na pokládané odborné otázky [7]. Roku 1997 došlo k dalšímu významnému zlomu ve vývoji AI, když byl velmistr světa v šachu Gary Kasparov poražen počítačem, konkrétně programem Deep Blue od společnosti IBM, což byl velký úspěch v oblasti vývoje programů založených na rozhodování [7]. Rovněž došlo k velkému pokroku v oblasti rozpoznávání řeči a rozpoznávání emocí (například robot Kismet, obrázek 3) [7].



Obrázek 3: Robot Kismet, 2003, schopný rozpoznávat a napodobovat lidské emoce (upraveno podle [14]).

3.1.5 Využití umělé inteligence v současnosti

Umělá inteligence je v současné době aplikována na obrovské množství oborů a služeb. Většina z nás ji využívá v každodenním životě aniž by si toho možná byla vědoma.
Nejtypičtějšími příklady AI v současnosti jsou:

- **Vyhledávač Google** - co dělá z webového vyhledávače od firmy Google velmi mocný nástroj současnosti je právě AI. Tento vyhledávač je dnes schopen nejen účinně vyhledávat na základně zadaných hesel, ale je také schopen doporučit konkrétní část videa s možnou odpovědí, nebo na základě zadaných hesel doporučit soubor otázek a odpovědí, které mohl mít uživatel na mysli [15]. Zároveň umí Google seskupovat různé související zprávy dle jejich tematického zaměření.

- **Doporučení ve webových službách** - různé webové služby začaly AI využívat pro přizpůsobení různých doporučení pro konkrétního uživatele. Velmi rozšířené je využití AI na různých sociálních sítích pro zobrazování přizpůsobeného obsahu, či návrh kontaktů (Twitter, Facebook, Instagram) [15]. Některé E-shopy využívají AI pro návrhy dalšího zboží, které by uživatel mohl chtít zakoupit (Amazon, Alza) [15], [16].

- **Streamovací služby** - ať už jde o filmy, videa, muziku, různé streamovací služby využívají AI pro doporučení nových médií, šitých uživateli přímo na míru (Netflix, YouTube, Spotify) [15].
- **Internetová reklama** - mnohý jistě zažil ten nepříjemný okamžik, kdy si prohlížel nějaké zboží a následujících několik dní na něj viděl reklamu na každé webové stránce, kterou navštívil. I toto má na svědomí AI. Účelem je přizpůsobit zboží nabízené reklamou tak, aby se reklama dostala k lidem, které by doopravdy mohla zaujmout, a měla tak co nejefektivnější dopad [15].
- **Navigace, mapy, cestování** - některé služby map (Google, Apple) a navigací využívají AI například k tomu, aby poskytly aktuální informace o provozu a vypočítaly nejrychlejší trasu. Služba Uber využívá AI k vyhledání vhodného auta a nacenění služby. Stejně tak využívají AI i některé platformy pro výběr letenek a další. [15]
- **Videohry** - herní průmysl je pravděpodobně jeden z prvních, který začal AI hojně využívat. Začalo to jednoduchým generováním levelů a obtížností, dále AI algoritmy vytvářejí počítačové oponenty, imitující reálného hráče, v závodních hrách jsou ostatní vozidla rovněž řízena AI. Velmi zajímavou aplikací AI v herním průmyslu jsou počítačem vytvořené charaktery (NPC) v herní sérii Middle Earth, které se vyvíjejí na základě interakcí s hráčem a dalších herních faktorů. [15]

AI je aplikována rovněž například na rozpoznávání spamů v emailové poště, pro různé virtuální asistenty (Siri, Cortana, Alexa), převody textu na řeč, rozpoznávání tváří, různé druhy předpovědí (předvídání poruchy stroje, vývoje akcií, apod.), optimalizace cen, práce s Big Data a jiné datové analýzy [17]. Výčet dalších využití může pokračovat do nekonečna.

3.2 Strojové učení

Strojové učené (Machine learning, ML) je jednou z podoblastí AI zaměřující se na schopnost stroje učit se [18]. Kombinací zpracování dat a použitím různých metod a algoritmů se ML snaží imitovat lidskou schopnost učit se z předchozích zkušeností a postupně zlepšovat přesnost svého úsudku [18].

ML se stal významnou součástí datové vědy [18]. Nachází uplatnění ve zpracování obrovského množství dat, jejich klasifikaci a identifikaci jemných vazeb a vzorců [18].

3.2.1 Využití strojového učení

ML je velmi populární a extrémně rozšířenou podoblastí AI. Jeho aplikací je obrovské množství, uvedeme některé z těch nejznámějších [18]:

- **Rozpoznání řeči** - jedná se o schopnost převádět mluvenou řeč na text. ML algoritmy k tomu využívají metod natural language processingu (zpracování přirozeného jazyka), více v kapitole 3.1.3. Rozpoznávání řeči bývá v současnosti již běžnou výbavou většiny mobilních zařízení, využívají jej virtuální asistenti apod.

- **Zákaznické služby** - díky ML je dnes mnoho webových stránek nabízející služby zákazníkům vybaveno chatboty. Chatbot je automatizovaný nástroj, který je schopen komunikovat se zákazníkem, odpovídat na často kladené otázky, poskytovat různé rady přizpůsobené preferencím zákazníka, navrhovat zákazníkům podobné, nebo související produkty a při prodeji oblečení, nebo bot například navrhovat správné velikosti.

- **Computer vision (Počítačové vidění)** - viz kapitola 3.1.3.

- **Systémy doporučení** - schopnost nabízet personalizované služby nebo produkty. Tato oblast využití ML byla již rozebírána v několika bodech v kapitole 3.1.5, například metody nabízení souvisejících produktů na platformách jako Amazon, či Alza, nebo doporučení na různých streamovacích platformách.

- **Automatické obchodování s akciemi** - díky ML mají zájemci o obchodování s akciemi možnost automatizovat mnoho souvisejících úkonů. Nástroje ML jsou navrženy pro optimalizaci obchodních portfolií a automaticky provádí tisíce až miliony obchodů denně.

3.2.2 Princip fungování strojového učení

Proces učení ML lze rozdělit na tři hlavní části [18]:

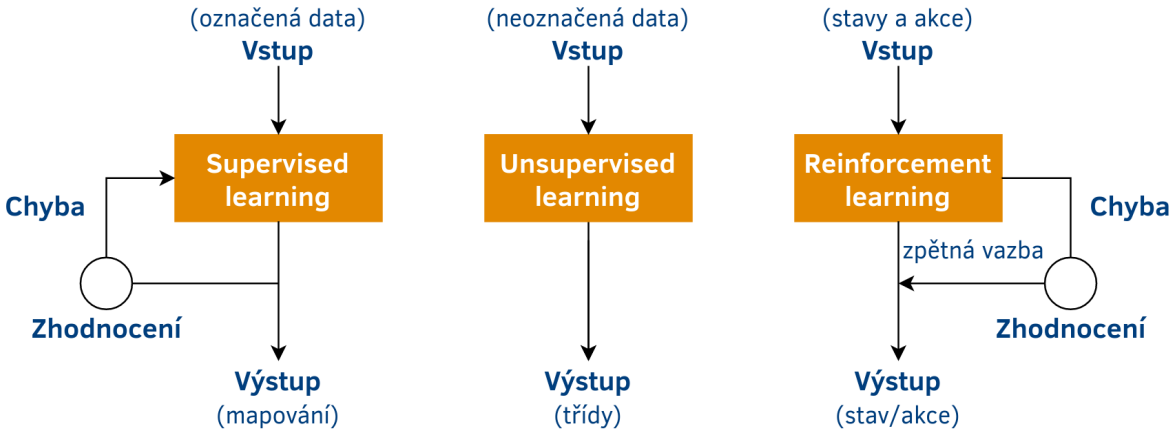
1. **Proces rozhodování** - ML algoritmy pracují se vstupními daty a vytváří rozhodovací model, jehož výstupem jsou odhady, nebo předpovědi. Obecně vzato jsou algoritmy ML využívány buď k predikci, nebo klasifikaci.

2. **Chybová funkce** - tato funkce slouží k popisu pravdivosti výstupů modelu. Pokud jsou dostupné informace o správných výstupech, chybová funkce může porovnat výstupy modelu s očekávanými reálnými výstupy a zhodnotit chybovost vytvořeného modelu.

3. **Proces optimalizace modelu** - proměnné modelu jsou upraveny tak, aby byla snížena odchylka mezi odhady modelu a očekávanými výstupy. Algoritmus bude opakovat cyklus zhodnocení a optimalizace modelu tak dlouho, dokud nedosáhne požadované přesnosti.

3.2.3 Základní metody strojového učení

Existuje několik možností, jakým způsobem přistupovat k ML problémům. Na obrázku 4 jsou symbolicky znázorněny průběhy procesu učení u tří z těchto přístupů.



Obrázek 4: Symbolické grafické znázornění procesu učení pro různé druhy ML (upraveno podle [19]).

3.2.3.1 Učení s učitelem)

Učení s učitelem (Supervised Machine Learning, SML) je metoda, kde algoritmus nejprve pracuje s daty, u nichž je známý požadovaný výstup (data jsou ”označena”). Model je postupně optimalizován, aby vyhovoval označeným vstupním datům. Následně model zpracovává nová, již neoznačená data a snaží se na základě označeného setu dat odhadnout jejich výstup. Typickým příkladem SML je lineární regrese. [20]

3.2.3.2 Učení bez učitele

Učení bez učitele (Unsupervised Machine Learning, UML) je metoda, kde algoritmus pracuje s daty, u nichž není předem známý správný výstup (data jsou ”neoznačena”). Jsou hledány vzájemné podobnosti a souvislosti mezi jednotlivými daty, na základě čehož jsou data shlukována do menších celků. Typickým příkladem UML jsou různé druhy třízení. [20]

3.2.3.3 Částečné učení s učitelem

Částečné učení s učitelem (Semi-Supervised Machine Learning, SSML) je metoda, která nabízí kombinaci SML a UML. Během učení využívá algoritmus menší set označených dat a zároveň extrahuje soubor vlastností z větší skupiny neoznačených dat. Toto řešení je ideální v případech, kdy z jakéhokoliv důvodu není možné dodat SML algoritmu dostatek označených dat k vytrénování modelu. [18]

3.2.3.4 Zpětnovazební učení

Zpětnovazební učení, někdy zvané také posilující učení (Reinforcement Machine Learning, RML), je metoda principem velmi podobná SML [18]. Při RML se algoritmus učí určité akce v závislosti na souboru vstupních stavů [19]. Po provedení akce shromáždí algoritmus zpětnou vazbu z okolního prostředí, která je následně využita k posílení dalšího cyklu učení a zpřesnění modelu [19], [18]. Jedná se o metodu pravděpodobně nejpodobnější principu skutečného lidského učení - algoritmus provádí akce metodou pokus-omyl, negativní zpětná vazba z okolí je pro něj signálem ke změně postupu [19], [18]. Zpětné vazbě se také někdy říká odměna - čím pozitivnější zpětná vazba je, tím vyšší odměna [19].

3.2.4 Modely strojového učení

V souvislosti s ML systémy je tzv. modelem myšlena matematická reprezentace výstupu procesu učení, neboli ML model je něco jako program, který generuje výstupy a předpovědi [19]. ML model je odlišný pojem od algoritmu ML. Algoritmus je procedura, nebo metoda, která pracuje s daty, aby odhalila skryté souvislosti a opakující se vzorce, čímž generuje model [19]. Formulováno jinými slovy, algoritmus, který je trénován na datech, se stává modelem [19]. Modely ML jsou rozdělovány do tří základních skupin na základě metod ML (viz kapitola 3.2.3): modely SML, modely UML a modely RML.

3.2.4.1 Modely učení s učitelem

Modely SML ještě dále dělíme na dvě podskupiny - regrese a klasifikace [19].

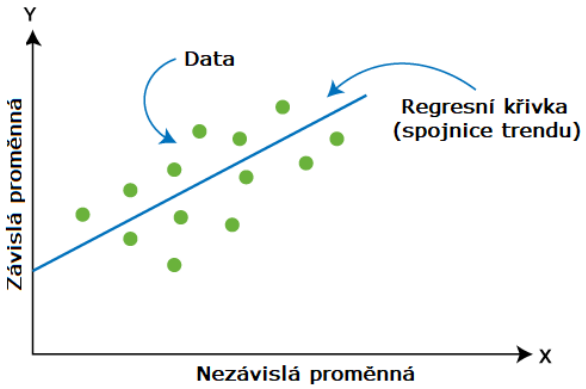
Regrese

U modelů regrese je výstupem jediná proměnná v podobě konstanty [19]. Příklady regrese mohou být:

1. **Lineární (popř. obecná, polynomická) regrese** - jedná se o jeden z nejjednodušších modelů ML [19]. Slouží k předpovědi jedné výstupní hodnoty, na základě množiny vstupních hodnot [19]. Matematickou reprezentací lineární regrese je rovnice přímky, tedy výstupní hodnotu získáme za základě rovnice:

$$Y = ax + b \tag{1}$$

   kde x značí vstupní hodnoty a a, b jsou parametry přímky [19]. Grafické znázornění modelu se nachází na obrázku 5. Cílem modelu je nalézt rovnici přímky, která nejlépe vyhovuje trendu vstupních dat.



Obrázek 5: Grafické znázornění lineární regrese (upraveno podle [19]).

Stejný princip jako model lineární regrese mají i modely obecné lineární regrese (multiple linear regression) a polynomické regrese [19]. V případě obecné lineární regrese pracujeme se soustavou n lineárních rovnic:

$$Y_i = b_0 + b_1 \cdot x_{i1} + b_2 \cdot x_{i2} + ... + b_n \cdot x_{in} \tag{2}$$

kde $i \in \{1, 2, ..., n\}$, $x_{i1}, ..., x_{in}$ značí vstupní hodnoty a $b_0, ..., b_n$ jsou parametry přímky [21].

V případě polynomické regrese pracujeme s rovnicí:

$$Y = b_0 + b_1 \cdot x + b_2 \cdot x^2 + ... + b_n \cdot x^n \tag{3}$$

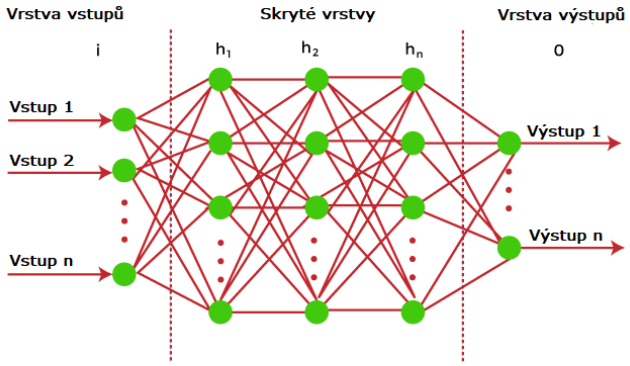
kde x značí vstupní hodnoty a $b_0, ..., b_n$ jsou parametry polynomu [22].

2. **Rozhodovací strom** - oblíbený model ML, který lze využít jak pro regresi, tak pro klasifikaci [19]. Jedná se o rozhodovací proces větvící se v každém novém uzlu. Uzly reprezentují testy, nebo atributy, větve symbolizují výstup daného procesu [19]. Výhodou tohoto modelu je jeho snadná implementace, nevýhodou je, že je obtížné až nemožné zajistit přesnost výstupu [19].

3. **Náhodný les** - tento model pracuje se souborem velkého množství rozhodovacích stromů [19]. Každý rozhodovací strom samostatně zhodnotí vstupní hodnoty a vyhodnotí výstup [19]. Tímto vznikne množina mnoha výstupů jednotlivých stromů [19]. Při regresních problémech se za finální výstup považuje průměr celé množiny výstupů [19].

4. **Neuronová síť** - jak již bylo naznačeno v kapitole 3.1.3, neuronová síť je model, který simuluje funkci neuronů a nervových spojení v lidském mozku, pomocí propojení milionů virtuálních neuronů [19]. Grafické znázornění stavby neuronové sítě se

nachází na obrázku 6. Z obrázku 6 lze vyčíst, že se model neuronové sítě skládá z množství vrstev, z nichž jedna tvoří řadu vstupů a jiná zase řadu výstupů. Mezi nimi se nachází jedna, nebo více skrytých vrstev, obsahující vzájemně propojené neurony. Neuron reprezentuje matematickou funkci, která přijímá a třídí informace v závislosti na specifické architektuře sítě [23].

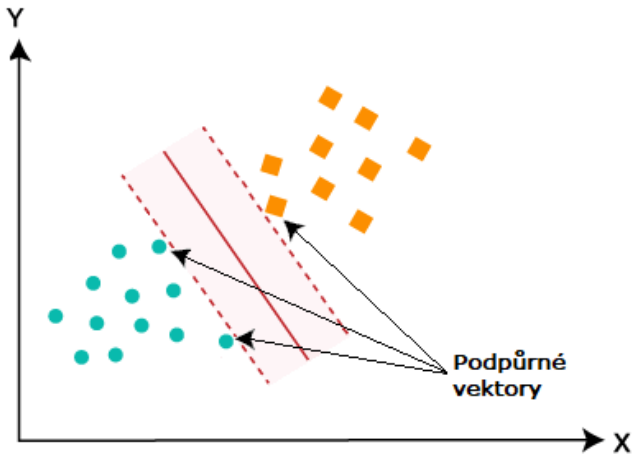


Obrázek 6: Grafické znázornění neuronové sítě (upraveno podle [19]).

Klasifikace

Klasifikační modely slouží k tomu, aby na základě technik SML byly schopny roztřídit vstupní data do několika kategorií [19]. V případech, kdy problém má pouze dvě možné kategorie, mluvíme o tzv. binární klasifikaci (Binary classification) [19]. V případech, kdy má problém více než dvě kategorie, mluvíme o tzv. klasifikaci s více třídami (Multi-class classification) [19]. Příklady klasifikace mohou být:

1. **Rozhodovací strom** - princip již popsán v této kapitole v části Regrese.
2. **Náhodný les** - princip již popsán v této kapitole v části Regrese. Jediným rozdílem je zhodnocení výstupní hodnoty - při klasifikačních problémech je jako finální výstup vybrána hodnota, která se v množině výstupů jednotlivých stromů vyskytuje nejčastěji [19].
3. **Logistická regrese** - tento model je používán pro klasifikační problémy a principem se podobá lineární regresi [19]. Jeho výstupem je číslo mezi 0 a 1, reprezentující pravděpodobnost, s jakou daný vstup patří do určité kategorie [19].
4. **Support vector machine (Podpůrný vektor)** - tento model lze použít i pro řešení regresních problémů, ale častěji bývá používán u klasifikačních problémů [19]. Principem tohoto modelu je hledání hranice, jíž by bylo možné rozdělit skupinu dat do jednotlivých kategorií [19]. Grafické znázornění tohoto modelu se nachází na obrázku 7



Obrázek 7: Grafické znázornění modelu support vector machine (upraveno podle [19]).

5. **Naïve Bayes (Naivní Bayesův klasifikátor)** - název je odvozen od Bayesovy věty[2], na které je model vystavěn [19]. Výstupní kategorie je určena na základě vztahu:

$$P(Y|x) = \frac{P(x|Y) \cdot P(Y)}{P(x)} \tag{4}$$

kde $x = (x_1, x_2, ..., x_n)$ je vektor závislých vstupních vlastností, $P(Y)$ je apriorní pravděpodobnost (class probability) a $P(Y|x)$ je věrohodnost (conditional probability) [25], [26]. Předpokladem modelu je, že jednotlivé vlastnosti jsou na sobě vzájemně nezávislé a každá má pro určení výstupu stejnou váhu [26]. Princip modelu vysvětlíme na příkladu hry golfu. Mějme množinu vstupních vlastností $x = (rain, hot, false)$ a výstup roven $y = Ne$ [26]. Potom $P(Y|x)$ v rovnici 4 značí pravděpodobnost, že nebudeme hrát golf, pokud podmínky počasí budou a) nejspíš bude pršet, b) je vedro, c) nefouká [26].

3.2.4.2 Modely učení bez učitele

Modely UML jsou využívány převážně k řešení následujících tří činností [19]:

1. **Klastrování** - nebo také shlukování, je model umožňující seskupování dat do menších skupin na základě vzájemných souvislostí a podobností. Klastrováním lze segmentovat obraz[3], nebo analyzovat statistická data. Mezi často užívané algoritmy klastrování patří například K-means Clustering, Hierarchal Clustering, nebo DBSCAN

[2]Bayesova věta v teorii pravděpodobnosti uvádí vztah mezi dvěma pravděpodobnostmi vzájemně opačných podmíněných jevů [24].

[3]Segmentace obrazu znamená rozdělení obrazu na několik částí (podobrazů), z nichž pixely v každé z nich mají příbuzné vlastnosti [27].

(Density-Based Spatial Clustering of Applications with Noise). Grafické znázornění principu klastrování se nachází na obrázku 8.

Obrázek 8: Grafické znázornění klastrovacího modelu (upraveno podle [19]).

2. **Association Rule Learning (Asociační analýza)** - cílem tohoto modelu je identifikovat závislosti jednoho vstupu na jiném a dokáže rozpoznat velmi zajímavé vztahy jednotlivých proměnných uvnitř velkého souboru dat. Model nachází nejčastější uplatnění v oblastech jako Market Basket Analysis[4], nebo Web Usage Mining[5]. Mezi často užívané algoritmy association rule learningu patří například algoritmus Apriori, Eclat, nebo FP-growth algoritmus.

3. **Redukce dimenze** - dimenze souboru dat označuje množství vlastností, nebo proměnných definovaných uvnitř tohoto souboru. Ačkoliv bývá výše dimenze obvykle spojená s přesnějším výstupem, příliš velké množství proměnných může vést k některým problémům, jako například overfitting (viz kapitola 3.2.5.1), nebo pokles efektivity algoritmu spojený se spotřebou příliš velkého množství výpočetního výkonu. V takových případech se nabízí možnost automatického snížení dimenze, aniž by došlo k příliš velkým změnám v poskytnutých informacích. Mezi algoritmy redukce dimenze patří například PCA (Principal Component Analysis), nebo Singular Value Decomposition.

3.2.4.3 Modely zpětnovazebního učení

Algoritmy RML dělíme na model-based a model-free. Model-based algoritmy fungují na principu pochopení okolního prostředí a vytvoření modelu na základě interakcí s tímto prostředím - vždy se snaží najít nejefektivnější možnou akci k dosažení požadovaného výsledku [30]. Model-free algoritmy se učí prostřednictvím zjištění následků svých akcí skrze

[4]Market Basket Analysis (analýza nákupního košíku) je metoda identifikace souvislostí mezi objekty pozorováním toho, které skupiny produktů, nebo služeb se často vyskytují společně v jedné transakci [28].

[5]Web Usage Mining je podoblast data miningu zabývající se extrakcí širokého spektra informací dostupných skrze webové stránky [29].

zkušennosti - tedy nejprve několikrát provedou určitou akci a teprve na základě zjištěných výstupů optimalizují svůj postup pro zvýšení pozitivní zpětné vazby [30]. Příklady modelů RML jsou následující [19]:

1. **Q-Learning** - model-free algoritmus založený na Bellmanově rovnici[6], který je navržen pro prostředí v němž nejsou předem známy žádné preferované stavy ani není definováno, co je považováno za pozitivní zpětnou vazbu [19], [30]. Q-learning není vhodný pro soubory dat s velkým počtem možných stavů a akcí [30].

2. **Deep Q-Network** - model využívající Q-Learningové algoritmy uvnitř neuronové sítě [19]. Řeší problém Q-Learningu pro prostředí s velkým počtem možných stavů a akcí [30].

3. **SARSA (State-Action-Reward-State-Action)** - jedná se o algoritmus založený na markovském rozhodovacím procesu[7] [19].

3.2.5 Komplikace spojené s aplikací strojového učení

Jakkoliv fascinující jsou možnosti ML a jakkoliv nesčetné jsou jeho aplikace v usnadňování našeho každodenního života, existuje i odvrácená strana, sestávající z různých limitů, problémů a nepřesností. Podívejme se na některé komplikace spojené s ML, s nimiž je nutné počítat.

3.2.5.1 Overfitting a underfitting

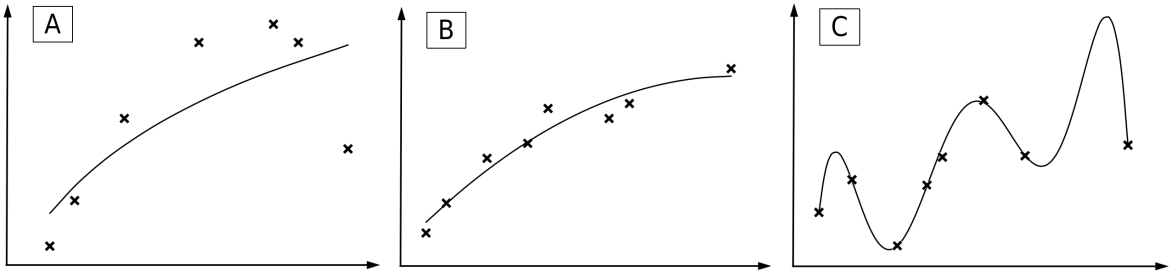
Důležitou vlastností každého ML modelu musí být schopnost vhodně zobecnit dostupná data, aby byl model schopen co nejpřesněji předpovídat další výstupy. V případech, kdy je model nastaven nevhodně, může docházet k overfittingu, nebo underfittingu [33]. Na obrázku 9 jsou oba tyto problémy graficky znázorněny na příkladu regrese.

Underfitting (obrázek 9 - A) označuje případ, kdy je model natolik nevhodně koncipován, že nejen není schopen předpovídat trend pro nová (neoznačená) data, ale nedokáže dostatečně zobecnit ani již dostupná (označená) data [33]. Underfitting nebývá problémem tak často jako overfitting, protože existují mnohé prostředky jak jej snadno a včas odhalit [33].

K overfittingu (obrázek 9 - C) dochází v případech, kdy je model příliš přesně napasován na dostupná data, v důsledku čehož již nevystihuje správný trend pro nová

[6]Bellmanova rovnice uvádí, že dlouhodobá odměna z určité akce je rovna kombinaci odměny z aktuální akce a očekávané odměny z budoucích akcí [31].

[7]Markovský rozhodovací proces je matematický proces používaný pro modelování rozhodovacích problémů v prostředí, kde je předem známa část preferovaných stavů a správných akcí [32].



Obrázek 9: Generalizace dat u ML modelů - A underfitting, B optimální fit, C overfitting.

data [33]. K overfittingu dochází častěji u nelineárních modelů, nebo modelů bez parametrů, jako jsou například rozhodovací stromy (viz kapitola 3.2.4.1, část Regrese). Overfitting je při sestavování ML modelů velmi častým problémem, protože to, že model umí pracovat s určitým vzorkem dostupných dat neznamená, že bylo dosaženo optimálního zobecnění [33]. Metody jak omezit riziko overfittingu jsou následující [33]:

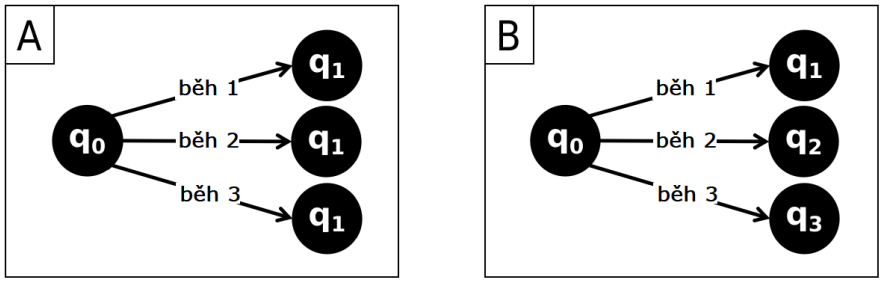
- **Resamplovací metody** - resamplování [8] umožňuje zhodnotit přesnost vytvořeného ML modelu. Jednou z oblíbených resamplovacích technik je k-fold cross validation, která umožňuje vytrénovat model několikrát po sobě (k krát) na různých podmnožinách původního souboru dat. Díky tomu lze vytvořit již poměrně dobrý odhad, s jakou přesností se bude model chovat pro nová data.
- **Validační soubor dat** - pokud máme k dispozici dostatečné množství dat, je další možností vytrénovat ML model pouze na části dostupných dat a jejich druhou část uchovat mimo trénovací proces k ověření (validaci). Validační soubor dat, u nichž jsou předem známy správné výsledky, může být poté využit pro otestování chování modelu.

3.2.5.2 Deterministické problémy

Algoritmy ML jsou stochastického (náhodného), nikoliv deterministického charakteru. Rozdíl mezi stochastickým a deterministickým algoritmem je znázorněn na obrázku 10. Zjednodušeně řečeno, deterministický algoritmus bude produkovat při stejných vstupních podmínkách stále stejné výstupy [35]. Stochastický algoritmus ovšem může i při stejných vstupních podmínkách generovat při každém svém spuštění trochu jiný výstup [35]. Použití ML tedy může být problematické pro některé deterministické problémy, při nichž očekáváme stejné chování modelu při daném souboru vstupních podmínek [36]. Příkladem deterministického problému mohou být například modely pro předpověď počasí [36].

[8] Resampling je soubor statistických technik sloužící ke zjištění dodatečných informací o daném souboru dat [34]. Použitím některé z těchto technik můžeme docílit zvýšení celkové přesnosti souboru, nebo v rámci souboru odhalit nejistoty [34].

Ačkoliv je možné vytvořit a vytrénovat ML model pro předpověď počasí, model nebude schopen do hloubky chápat fyzikální zákony, jimiž se opravdové předpovědní modely řídí [36]. ML model může být za určitých podmínek úspěšný i při předpovědi počasí, ale už nebude chápat, že například hustota nemůže být záporná [36].



Obrázek 10: Grafické znázornění principu deterministických (A) a stochastických (B) algoritmů: q_0 jsou vstupní podmínky, q_1, q_2 a q_3 jsou výstupy pro různá spuštění algoritmu.

3.2.5.3 Nedostatek dat

Všechny ML modely jsou založeny na procesu učení a mnohé z nich k tomuto procesu potřebují velké množství dat. Pakliže je model trénován na nedostatečném množství dat, lze očekávat, že jeho výstupy a odhady budou velmi nepřesné a model může být zcela nepoužitelný, nebo jen málo užitečný. [36].

Souvisejícím problémem je nedostatek kvalitních dat. Aby ML model fungoval správně a byl schopen co nejpřesnějších předpovědí, musí být trénován na pravdivých datech. Pakliže je model trénován například na datech obsahujících mnoho chyb a nejistot, lze očekávat, že takový model bude rovněž vysoce chybový při činění dalších předpovědí. [36]

Problematickým se rovněž ukázalo použití ML modelu, který byl trénován na souboru dat v určité situaci, na data z jiné situace [36]. Příkladem může být použití ML algoritmů s mamografickou[9] databází [36]. Mamografické databáze obsahují ve velké většině data pacientek s bílou barvou pleti [36]. Nicméně se ukázalo, že hrozba úmrtí na rakovinu prsu je výrazně vyšší pro pacientky s černou barvou pleti, kvůli celé řadě genetických i situačních faktorů [36], [38], [39]. V takovém případě skutečnost, že jsou ML modely trénovány převážně na pacientkách s bílou barvou pleti, má nepříznivé dopady použije-li se takový model pro vytváření předpovědí pro pacientky s černou barvou pleti [36].

[9]Mamografie je název rentgenového vyšetření ženské prsní tkáně [37].

3.2.5.4 Nesprávná aplikace

Prudký vývoj různých ML metod přirozeně způsobil, že ML začal expandovat do různých odvětví průmyslu i vědy. V roce 2019 na vědecké konferenci varovala statistička Genevera Allen z Rice University, že nesprávné používání metod ML způsobuje krizi ve vědecké komunitě [40]. Vědci totiž začali využívat metody ML pro vyhledávání souvislostí mezi daty ve svých výzkumech, aniž by dobře porozuměli principům zvolených ML metod [40]. Slepé používání ML takovýmto způsobem často vedlo k tomu, že se ML fixoval na šum v datech, místo na reálné souvislosti dat [40]. Kvůli tomu nebyly výsledky reprodukovatelné a výzkumníci zkoumající stejný problém na podobných datech za použití ML metod shledávali, že se jejich výsledky vůbec nepřekrývají, ačkoliv by měly [40].

Použití náhodných ML metod, bez pochopení toho, jak model došel k daným výsledkům, se může velmi vymstít [36]. ML model může totiž zdánlivě fungovat i když bude aplikován nesprávně [36]. Příkladem může být i již v kapitole 3.2.5.2 zmíněný problém využití ML k předpovědi počasí. ML metody budou fungovat i když budou aplikovány na deterministický problém jako je předpověď počasí, ale nebudou rozumět zákonům fyziky a na jejich výsledky se proto nebude možné spoléhat [36]. Případy nesprávné aplikace u stochastických problémů můžeme rozdělit na dva hlavní případy [36]:

- **P-hacking** - tento pojem označuje proces, kdy jsou ML metody použity na prohledávání obrovských objemů dat, dokud není nalezena statisticky významná souvislost. Takto rozsáhlé soubory dat mohou mít tisíce až miliony proměnných a hladina statistické významnosti pro většinu výzkumů je $p < 0.05$, tedy nalezení statisticky významných souvislostí není velký problém. Problémem je, že takovýmto způsobem nalezené souvislosti mohou být falešné, protože se ML model může zafixovat na šum v datech a pokud výsledky interpretuje osoba nedostatečně vzdělaná v oblasti ML, může tyto falešné výsledky vydávat za reálné souvislosti mezi daty.

- **Rozsah analýzy** - existuje významný rozdíl mezi analýzou pomocí ML modelů a statistickým modelováním. Zatímco statistické modelování je potvrzujícího charakteru, ML modely mají průzkumný charakter. Při statistickém modelování je nejprve vybrán konkrétní problém, který chceme popsat teoretickým modelem, je vybrán určitý počet faktorů, které bude model zahrnovat a následně je pečlivě navržena hypotéza a řada experimentů jak hypotézu otestovat a potvrdit. Pokud jsou ale objemy dat příliš velké a množství proměnných (faktorů) příliš vysoké, je nemožné stanovit konečný počet hypotéz pro podobný model. Průzkum dat pomocí ML metod může být v takovém případě jediný způsob, jak odhalit souvislosti v datech, ale tento druh průzkumu bude postrádat mnoho kvalit klasického statistického modelu. V mnoha případech je ML aplikován i na problémy, kde by bohatě stačily jiné statistické metody,

které by byly schopné poskytnout výrazně větší množství informací a byly méně chybové než ML.

3.2.5.5 Interpretovatelnost

Bez ohledu na to, jak přesný a užitečný dokáže ML model být, jeho výsledky bude číst a interpretovat člověk. Pakliže nejsou výstupy modelu interpretovatelné, může být celý model investory smeten ze stolu jako nepoužitelný. To, jestli bude daný model zhodnocen daným člověkem jako interpretovatelný, se odvíjí od celé řady faktorů, které dalece překračují hranice technických dovedností. Podstatnou součástí interpretovatelnosti modelu je zdůvodnění jak bylo dosaženo daných výsledků. ML model může dosáhnout vysoké přesnosti dalších predikcí, ale již nemusí být schopen odůvodnit a validovat svá rozhodnutí, která k výsledku vedla. Jak by v takovém případě bylo možné přesvědčit investora, aby důvěřoval výstupům daného modelu, pakliže ani samotný vývojář neví, jak bylo výsledků dosaženo. Z těchto důvodů je interpretovatelnost jednou z velmi důležitých kvalit každého ML modelu, který je zamýšlen pro použití v praxi. [36]

3.2.5.6 Etika

Obecně jsou s výrazným pokrokem v technologiích většinou spojeny i nějaké formy sociálních a etických kontroverzí. ML a obecně celá věda o AI se ve světě zasloužili o obrovský posun ve sběru dat a práci s daty a programy v této oblasti jsou již natolik vyvinuté, že vyvstal v celku očekávaný problém - lidé začali v některých případech věřit datům a algoritmům více, než vlastnímu úsudku a logice. Důvěra v technologie má své výhody i nevýhody. Technologie a programy výrazně usnadňují, zefektivňují a automatizují naši práci. Na druhou stranu občas některé technologie z pohodlnosti svádí ke slepé důvěře, jako například GPS navigace. Mnoho lidí má tendence důvěřivě následovat navigací navrženou trasu, aniž by je napadlo úsudek navigace zpochybnit a vybrat raději trasu dle vlastního úsudku. Další často diskutovaným problémem je, kdo převezme zodpovědnost v případě, že AI způsobí nějakou škodu, například když autonomní vozidlo způsobí nehodu. [36]

4 Praktická část práce

Cílem praktické části této bakalářské práce je sestrojení algoritmu (programu), který bude schopen číst předem definovaná data ze skupiny příbuzných PDF dokumentů. Praktická část této práce vznikla ve spolupráci s výzkumnou institucí Státní ústav radiační ochrany (SÚRO). Program je navržen pro specifické potřeby tohoto ústavu, aby pomáhal se sběrem dostupných dat při řešení výzkumných úkolů. Program byl testován na konkrétních reálných protokolech, poskytnutých se svolením Státního úřadu pro jadernou bezpečnost (SÚJB).

4.1 Podrobný popis řešeného problému

SÚRO je veřejnou výzkumnou institucí zabývající se odbornou činností v oblasti ochrany obyvatelstva před ionizujícím zářením[10]. Pro podporu své činnosti má ústav přístup k velké databázi protokolů z měření na různých zdravotnických rentgenových, či terapeutických zařízení.

Každé zařízení, schopné generovat ionizující záření, je vedeno v evidenci SÚJB. Každý držitel takového zařízení, je ze zákona[11] povinen zajistit, aby byla v pravidelných intervalech kontrolována funkčnost zařízení [43]. Tyto kontroly jsou prováděny v intervalech a rozsahu definovaných ve Vyhlášce o radiační ochraně a zabezpečení radionuklidového zdroje ([43]) pomocí jedné z následujících zkoušek [43]:

- **Přejímací zkouška (PZ)** - provádí se při zakoupení nového zařízení, nebo například při jeho přesunu na jiné pracoviště.
- **Zkouška dlouhodobé stability (ZDS)** - provádí se v pravidelných intervalech (např. jednou za rok, jednou za 3 roky apod.) a dále při dalších mimořádných situacích, například při důvodném podezření na nesprávnou funkci přístroje, apod.
- **Zkouška provozní stálosti (ZPS)** - provádí se v pravidelných intervalech, zpravidla měsíčních, za účelem průběžné kontroly správné funkčnosti zařízení.

V případě zkoušek PZ a ZDS je vystaven protokol, který vystavuje osoba, či firma mající oprávnění danou zkoušku vykonávat. Tyto protokoly jsou shromažďovány v centrálním registru SÚJB ve formátu PDF. Vzhledem k počtu zařízení na území České Republiky a poměrně častého intervalu jejich povinné pravidelné kontroly je v registru SÚJB shromážděno obrovské množství dokumentů. Protokoly obsahují z výzkumného hlediska cenné informace, které jsou však pohřbeny v obrovském množství PDF dokumentů, protože manuální vypisování potřebných informací zabírá obrovské množství času.

[10] Ionizující záření je označení pro takové záření, které má dostatečnou energii k odtržení elektronu z elektronového obalu atomu, schopné způsobit chemické změny v látce [41]

[11] Atomový zákon (zákon č. 263/2016 Sb.) [42]

Pro začátek bylo cílem prozkoumat možnosti automatizace procesu čtení těchto dat z PDF protokolů, tedy vytvořit program, který by dokázal protokoly vhodnou metodou zpracovat a aplikovat vybrané ML algoritmy. Program by měl vyžadovat co nejnižší časovou investici ze strany uživatele a jeho výstupy by měly být dobře interpretovatelné pro zaměstnance SÚRO, potažmo SÚJB.

4.2 Volba programovacího jazyka

Pro implementaci programu byl zvolen jazyk Python. Existuje celá řada programovacích jazyků vhodných pro konstrukci ML modelů, kromě Pythonu jsou to například JavaScript, R, nebo C++ [44]. Pro implementaci programu v této práci byl zvolen jazyk Python z toho důvodu, že pracovníci SÚRO, kteří budou program využívat, mají zkušennost právě s tímto programovacím jazykem a budou proto schopni program snadněji spravovat. Kromě toho je Python nejen velmi intuitivní a oblíbený jazyk na vzestupu, ale rovněž v něm existuje mnoho velmi užitečných knihoven přizpůsobených pro práci s ML [44].

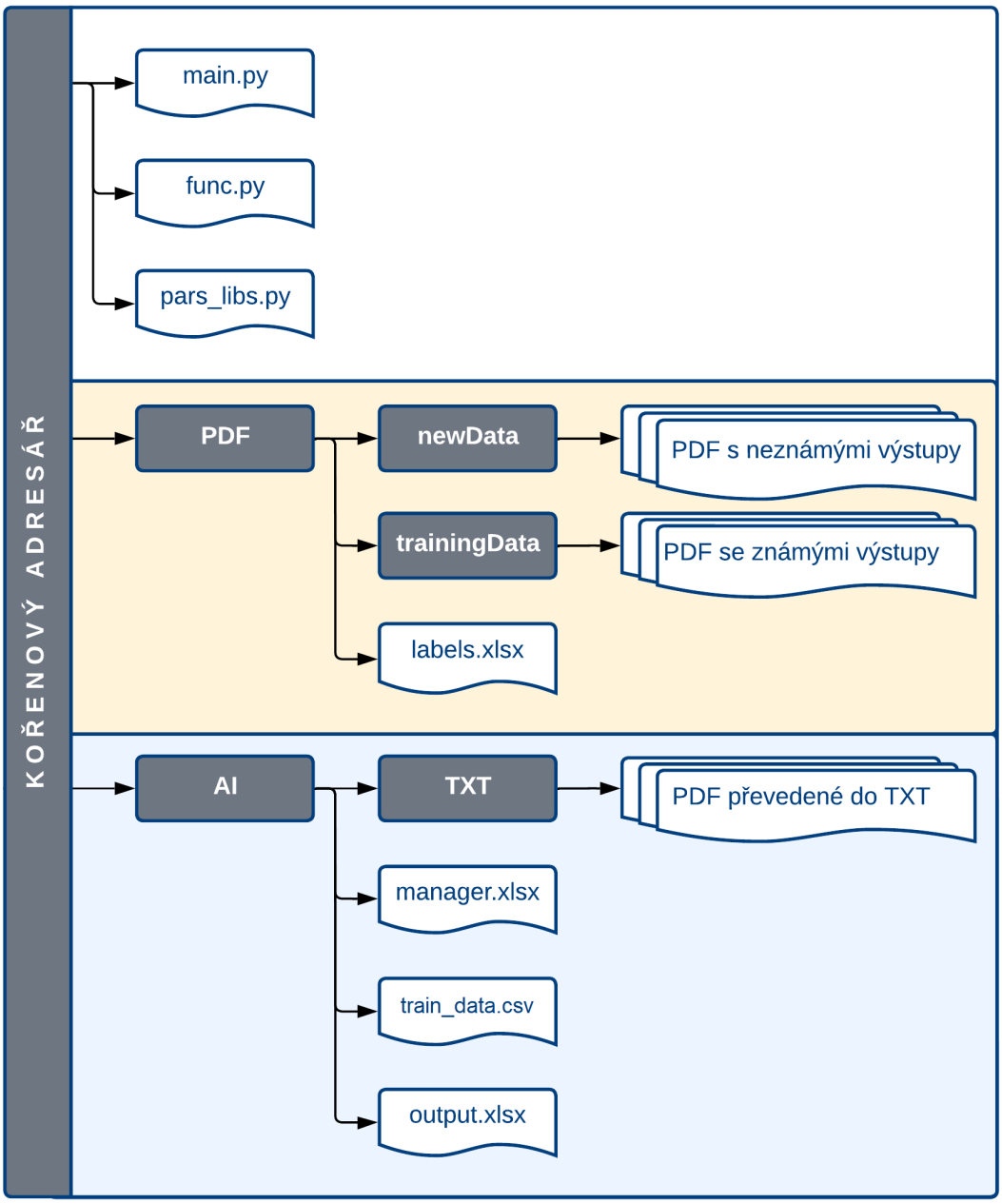
4.3 Stavba dokumentů

Jak již bylo zmíněno v části 4.1, dokumenty, které bude program v praxi zpracovávat, jsou PDF protokoly z měření na různých zařízeních schopných generovat ionizující záření. Pro účely této práce byly vybrány pouze protokoly z měření na jednom typu rentgenového zařízení. Všechny protokoly mají legislativou stanovený rozsah uvedených informací a testovaných parametrů, díky čemuž je předem dáno, jaké kapitoly a informace musí každý protokol obsahovat. Není však stanovena žádná jednotná šablona pro podobu protokolu, takže ačkoliv je o obsahu protokolu předem rozhodnuto, každá osoba/firma vystavující protokol má trochu jiný způsob zápisu potřebných informací a používá svou vlastní šablonu. Osob, či firem provádějící měření je na území České Republiky omezené množství, takže jeden typ zařízení nemívá více než 15 až 20 různých šablon protokolů. Dále většina protokolů vystavených konkrétní osobou, či firmou mívá pouze velmi omezené množství variant, zpravidla ne více než 3.

4.4 Souborový systém

Pro účely snadné orientace ve výstupech programu a bezproblémové interakce uživatele s programem byl navržen systém souborů v adresáři, v němž bude program pracovat. Souborová struktura je znázorněna na obrázku 11. Program sestává ze tří souborů:

- ***pars_libs.py*** - obsahuje všechny uživatelem definované parametry a použité knihovny
- ***func.py*** - obsahuje všechny funkce
- ***main.py*** - řídí celý program.



Obrázek 11: Popis souborové struktury - šedé podbarvení značí adresáře, bílé soubory.

Tyto tři soubory jsou umístěny v kořenovém adresáři, v němž se dále nachází dva adresáře:

- ***PDF*** - tento adresář je určen pro interakci s uživatelem. Obsahuje:
  - ***labels.xlsx*** - soubor, do něhož uživatel manuálně doplňuje určitá data z vybraného vzorku protokolů (příprava pro SML). Ukázka struktury souboru se nachází v tabulce 1.
  - **adresář *newData*** - adresář sloužící k nahrávání všech nových PDF dokumentů. Obsahuje všechny neoznačené dokumenty (tedy ty, které nemají záznam v *labels.xlsx*).
  - **adresář *trainingData*** - obsahuje všechny označené dokumenty PDF (ty, z nichž byla vypsána data do *labels.xlsx*).
- **AI** - tento adresář je interním pracovním adresářem programu a všechny soubory v něm jsou spravovány a tvořeny výhradně programem, bez jakékoliv intervence ze strany uživatele. Obsahuje:
  - ***manager.xlsx*** - tento soubor program vytváří a využívá za účelem získávání informací o dostupných dokumentech a pro ukládání nových informací. Ukázka struktury souboru se nachází v tabulce 2.
  - **adresář *TXT*** - obsahuje všechny dokumenty PDF převedené to textové formy, která je pro program čitelnější.
  - ***train_data.csv*** - soubor sloužící jako vstup pro SML algoritmus. Soubor je vytvářen automaticky a obsahuje název textového souboru, obsah textového souboru, informaci hledanou v daném souboru (přečteno ze souboru *labels.xlsx*) a řádek na němž se hledaná informace v souboru nachází.
  - ***output.xlsx*** - soubor obsahující hlavní výstup celého programu, tedy přečtená určitá data z nových dokumentů PDF.

file_name	**txt_name**	**info1**	**info2**	**info3**	**...**
file1.pdf	file1.txt	...	...	...	...
file2.pdf	file2.txt	...	...	...	...
file3.pdf	file3.txt	...	...	...	...
file4.pdf	file4.txt	...	...	...	...
file5.pdf	file5.txt	...	...	...	...
...	...	...	...	...	...

Tabulka 1: Struktura souboru */PDF/labels.xlsx*.

file name	location	labeled	converted	class1	class2	...
file1.pdf	C:\root\PDF\trainingData\file1.pdf	1	1	0	1	...
file2.pdf	C:\root\PDF\trainingData\file2.pdf	1	1	0	1	...
file3.pdf	C:\root\PDF\newData\file3.pdf	0	1	-1	0	...
file4.pdf	C:\root\PDF\newData\file4.pdf	0	1	-1	0	...
file5.pdf	C:\root\PDF\newData\file5.pdf	0	1	-1	2	...
file6.pdf	C:\root\PDF\trainingData\file6.pdf	1	1	-1	0	...
file7.pdf	C:\root\PDF\newData\file7.pdf	0	1	2	3	...
...	...	...	...	...	...	...

Tabulka 2: Struktura souboru */AI/manager.xlsx*.

4.5 Skladba programu

Program je rozdělen do tří logických celků: iniciace, zpracování souborů a část umělé inteligence, jejíž součástí je jednak klasifikace dokumentů, jednak čtení dat z dokumentů. Zjednodušený vývojový diagram průběhu programu se nachází na obrázku 12.

4.5.1 Iniciace

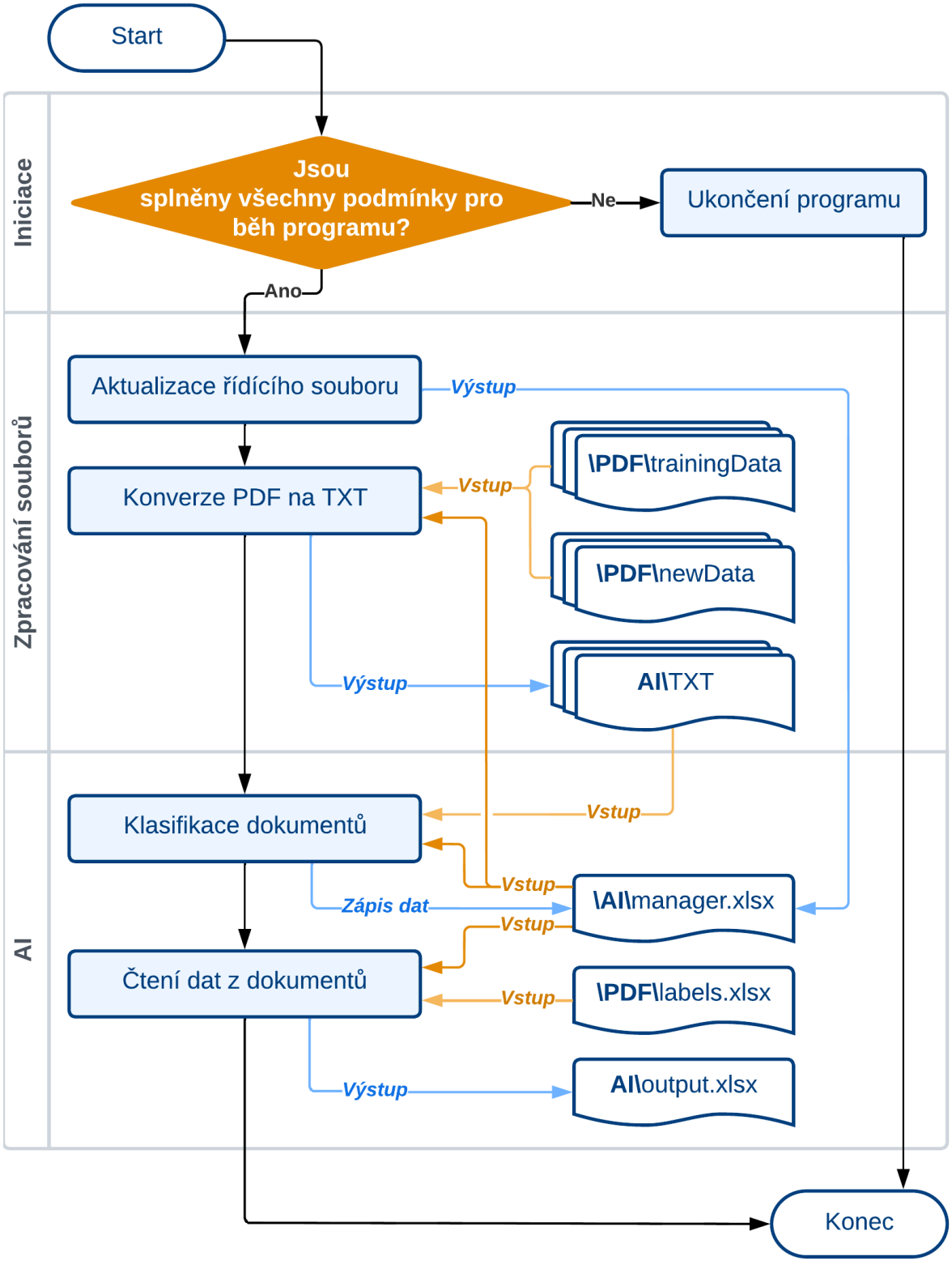
Úkolem iniciační části je zjistit, zda jsou splněny všechny nutné předpoklady pro běh programu. Pakliže předpoklady splněny nejsou a iniciace je tudíž neúspěšná, program bude ukončen. Kromě kontroly nutných předpokladů se iniciační část rovněž stará o další potřebné úkony, nutné k hladkému běhu programu. Akce provedené během iniciace jsou následující:

1. **Kontrola pracovních adresářů** - kontrola existence pracovních adresářů (viz souborová struktura na obrázku 11). Pokud některý z adresářů dosud neexistuje, je založen.

2. **Podmínka č.1** - kontrola zda existuje soubor */PDF/labels.xlsx*. Jeho neexistence má za následek ukončení programu.

3. **Správné umístění dokumentů** - jsou načtena data ze souboru */PDF/labels.xlsx* a je zajištěno, aby se všechny v souboru uvedené (označené) dokumenty nacházely v adresáři */PDF/trainingData*, zatímco všechny ostatní dokumenty aby byly umístěny adresáři */PDF/newData*.

4. **Podmínka č.2** - kontrola jestli se v adresáři */PDF/trainingData* stále nachází nějaké dokumenty (pokud se i po předchozí operaci v adresáři stále nachází nějaké dokumenty znamená to, že existují nějaké označené dokumenty pro tréning SML modelu). Pokud je adresář prázdný, program je ukončen.



Obrázek 12: Zjednodušený vývojový diagram programu pro čtení dat z dokumentů.

Jak je již naznačeno v předchozích bodech, pro úspěšnou iniciaci je nezbytné, aby existoval soubor */PDF/labels.xlsx* a aby složka */PDF/trainingData* nebyla ke konci iniciačního procesu prázdná. Důvody pro tyto podmínky jsou prosté. Program má sloužit pro čtení dat z dokumentů, k čemuž využívá SML. Pakliže neexistují žádné označené dokumenty, na nichž by bylo možno vytrénovat model, nemá vůbec smysl pokračovat.

4.5.2 Zpracování souborů

V této části dochází ke všem nezbytným přípravám dat pro sestavení ML modelů. Jedná se o dva hlavní úkoly:

1. **Aktualizace řídícího souboru** - založení/přepsání souboru */AI/manager.xlsx*. Je vytvořen seznam všech dostupných dokumentů (tedy těch nacházejících se buď v adresáři */PDF/trainingData*, nebo */PDF/newData*) a ke každému dokumentu jsou do souboru */AI/manager.xlsx* uloženy následující informace:
   - název dokumentu
   - kompletní adresa dokumentu (kde se dokument nachází)
   - zda je dokument označen, či nikoliv
   - zda byl dokument již konvertován do textového formátu, či nikoliv.

   Řídící soubor slouží ostatním funkcím programu pro snadné a rychlé získání potřebných informací o souborech a k ukládání nových informací.

2. **Konverze na text** - ze souboru */AI/manager.xlsx* zjistí, které dokumenty PDF dosud nebyly konvertovány do textového formátu, tyto soubory konvertuje a vzniklé TXT verze dokumentů uloží do adresáře */AI/TXT*.

4.5.3 Umělá inteligence

AI sekce se rozpadá na dvě samostatné aplikace ML. Aplikace na klasifikaci dokumentů (UML) a aplikace na čtení dat z dokumentů (SML).

4.5.3.1 Klasifikace dokumentů

Práce byla v ohledu klasifikace dokumentů volně inspirována publikací *Text mining of accident reports using semi-supervised keyword extraction and topic modeling* ([45]), v níž je řešen obdobný problém - publikace se věnuje navržení vhodného způsobu extrakce informací z velkého množství reportů o nehodách. Publikace se věnuje různým metodám klasifikace těchto reportů do menších skupin v závislosti na příčinně nehody, aby byla usnadněna následná analýza nehod. V našem případě pracujeme s částečně strukturovanými dokumenty ve formě protokolů, jejichž struktura se liší v závislosti na tom, jaký subjekt protokol vystavil.

Klasifikace dokumentů je tedy důležitá z toho hlediska, že podaří-li se rozdělit protokoly do malých skupin dle zhotovitele, daná malá skupina protokolů bude již mít velmi obdobnou strukturu a bude snazší vyhledat v dokumentu potřebné informace. Klasifikace (respektive spíše klastrování) dokumentů v této bakalářské práci probíhá za použití všech dokumentů dostupných v podadresářích */PDF* formou UML. Pro účely nalezení optimálního modelu pro klasifikaci dokumentů byly implementovány dvě různé UML metody:

1. **K-Means** - tato metoda klastrování patří mezi nejoblíbenější pro svou jednoduchou implementaci a výpočetní efektivitu. Nevýhodou této metody je, že je potřeba předem stanovit počet tříd/shluků *k*, do kterých se budou data dělit. Pro přibližný odhad kolik tříd je pro zvolenou metodu optimální byla využita metoda lokte (Elbow Method). Tato metoda spočívá v tom, že je metoda K-Means aplikována na daný dataset s měnící se hodnotou počtu shluků *k* a pro každou aplikaci je stanovena suma čtverců vzdálenosti každého bodu k centru nejbližšího shluku. Takto vzniklé body jsou vyneseny do grafu, v jehož tvaru by měl být patrný zlom v určité hodnotě *k*, z čehož lze usoudit optimální hodnotu *k* pro daný dataset. [20]

2. **DBSCAN** - narozdíl od metody K-Means, tato metoda nepotřebuje žádný předpokládaný počet tříd. Počet tříd je určen automaticky na základě analýzy hustoty shluků v jednotlivých oblastech. Důležitým parametrem této metody je parametr *epsilon*, definující určitou vzdálenost - body vzdálené od sebe v rámci této vzdálenosti jsou považovány za třídu/shluk. Body, které se nevejdou do definované vzdálenosti jsou považovány za šum (algoritmus jim přiřadí hodnotu -1). [20]

Pro účely testování klasifikace byla vybrána skupina cca 180 protokolů jediného typu rentgenového zařízení od sedmi různých zhotovitelů. Účelem klasifikace bylo roztřídit protokoly dle jednotlivých zhotovitelů (tedy do sedmi tříd). U každého protokolu byla předem zjištěna třídní příslušnost a protokoly byly pro přehlednost pojmenovány shodně dle příslušné třídy (dvoupísmenným kódem skupiny a číslem).

4.5.3.2 Čtení dat z dokumentů

Ze souboru */AI/manager.xlsx* byla zvolena jedna třída dokumentů a pro tuto třídu byla z každého dokumentu do souboru */PDF/labels.xlsx* vypsána informace o typu rentgenového zařízení, o firmě zhotovující protokol a o naměřené dávkové veličině při jednom z testů. Pro čtení dat byla zvolena metoda SML rozhodovací strom. Algoritmus se pokouší uhodnout číslo řádku, na němž se nachází požadovaná informace. Pro zpřesnění výsledků byla do algoritmu implementována vlastnost, která určuje skupinu výrazů, které se na řádku určitě nemohou vyskytovat (například jedná-li se o název typu rentgenového zařízení, název nikdy neobsahuje tečky, dvojtečky apod.). Pakliže algoritmus určí řádek obsahující některý z těchto

zakázaných výrazů, odhad zahodí a pokusí se určit řádek znovu. Pro účely srovnávání výsledků byla data přečtena ze skupiny zkušebních protokolů čtyřmi různými způsoby:

1. **způsob 1** - set dat použitých pro trénink modelu obsahoval 10 dokumentů (pro vyzkoušení výkonu modelu trénovaném na malém počtu dokumentů)
2. **způsob 2** - set dat použitých pro trénink modelu obsahoval 10 dokumentů a navíc vlastnost rozpoznávání zakázaných výrazů
3. **způsob 3** - set dat použitých pro trénink modelu obsahoval alespoň 50 dokumentů (pro účely srovnání výkonu modelu trénovaném na malém a středním počtu dokumentů)
4. **způsob 4** - set dat použitých pro trénink modelu obsahoval alespoň 50 dokumentů a navíc vlastnost rozpoznávání zakázaných výrazů

Definovaný model pracuje především s pozicí dané informace v textu, tedy v novém dokumentu se snaží odhadnout číslo řádku, na němž se bude potřebná informace nacházet. Číslo řádku pak převádí zpět na textovou informaci a celý výstup ukládá do souboru */AI/output_classX.xlsx*, kde ”X” symbolizuje číslo identifikované třídy.

Skript si umí poradit i s případem, kdy jsou dokumenty k analýze rozloženy do několika tříd. Rozdělí tréningový dataset dle těchto tříd a trénuje model vždy na setu dokumentů ze stejné třídy. Tento model následně aplikuje na set nových dokumentů, opět pouze na dokumenty klasifikované ve stejné třídě, pro níž byl model trénován.

5 Výsledky a diskuse

5.1 Klasifikace dokumentů

Ilustrační ukázka toho, jak vypadal výstup klasifikace dokumentů a nachází na obrázku 13.

	A	B	C	D	E	F
1	**file name**	**location**	**labeled**	**converted**	**class_DBSCAN**	**class_K-Means**
2	**xi1.pdf**	C:\Users\I	0	1	-1	4
3	**xi10.pdf**	C:\Users\I	0	1	0	4
4	**xi11.pdf**	C:\Users\I	0	1	0	4
5	**xi12.pdf**	C:\Users\I	0	1	-1	4
6	**xi13.pdf**	C:\Users\I	0	1	-1	4
7	**xi2.pdf**	C:\Users\I	0	1	0	4
8	**xi3.pdf**	C:\Users\I	0	1	0	4
9	**xi4.pdf**	C:\Users\I	0	1	-1	4
10	**xi5.pdf**	C:\Users\I	0	1	0	4
11	**xi6.pdf**	C:\Users\I	0	1	0	4
12	**xi7.pdf**	C:\Users\I	0	1	-1	4
13	**xi8.pdf**	C:\Users\I	0	1	0	4
14	**xi9.pdf**	C:\Users\I	0	1	0	4
15	**fe1.pdf**	C:\Users\I	0	1	1	0
16	**fe2.pdf**	C:\Users\I	0	1	1	0
17	**fe3.pdf**	C:\Users\I	0	1	1	0
18	**fe4.pdf**	C:\Users\I	0	1	1	0
19	**fe5.pdf**	C:\Users\I	0	1	1	0
20	**fe6.pdf**	C:\Users\I	0	1	1	0
21	**lu1.pdf**	C:\Users\I	0	1	2	5
22	**lu10.pdf**	C:\Users\I	0	1	2	5
23	**lu11.pdf**	C:\Users\I	0	1	2	5
24	**lu12.pdf**	C:\Users\I	0	1	2	5
25	**lu13.pdf**	C:\Users\I	0	1	2	5
26	**lu14.pdf**	C:\Users\I	0	1	2	5
27	**lu15.pdf**	C:\Users\I	0	1	2	5
28	**lu16.pdf**	C:\Users\I	0	1	2	5
29	**wq1.pdf**	C:\Users\I	0	1	3	2
30	**wq10.pdf**	C:\Users\I	0	1	3	2
31	**wq11.pdf**	C:\Users\I	0	1	3	2
32	**wq12.pdf**	C:\Users\I	0	1	3	2

Obrázek 13: Ukázka části výstupu klasifikace programu - na obrázku je ukázán vzhled souboru */AI/manager.xlsx*. Sloupce obsahující klasifikaci byly pro přehlednost obarveny.

	počet	epsilon = 0.5 třída	epsilon = 0.5 zastoupení	epsilon = 0.8 třída	epsilon = 0.8 zastoupení	epsilon = 1 třída	epsilon = 1 zastoupení
skupina 1	13	0	38%	0	62%	0	100%
		-1	62%	-1	38%		
skupina 2	6	1	100%	1	100%	1	100%
skupina 3	16	-1	100%	2	100%	2	100%
skupina 4	54	2	20%	3	97%	3	100%
		3	59%	4	9%		
		4	9%				
		-1	11%				
skupina 5	65	5	60%	5	98%	4	100%
		6	22%	-1	2%		
		-1	18%				
skupina 6	6	-1	100%	6	100%	5	100%
skupina 7	21	7	86%	7	100%	6	100%
		-1	14%				

Tabulka 3: Klasifikační metoda DBSCAN pro různě zvolené hodnoty parametru *epsilon*. Hodnota *epsilon = 0.5* odpovídá 10 třídám, *epsilon = 0.8* odpovídá 9 třídám a *epsilon = 1* odpovídá 7 třídám. Sloupec počet uvádí počet dokumentů v dané skupině, sloupec zastoupení uvádí procentuální zastoupení dokumentů dané třídy v dané skupině dokumentů.

	počet	k = 10 třída	k = 10 zastoupení	k = 9 třída	k = 9 zastoupení	k = 7 třída	k = 7 zastoupení
skupina 1	13	6	100%	4	100%	4	100%
skupina 2	6	7	100%	0	100%	5	100%
skupina 3	16	4	100%	5	100%	3	100%
skupina 4	54	9	20%	2	100%	1	100%
		3	70%				
		2	9%				
skupina 5	65	5	97%	7	68%	0	91%
		1	3%	1	26%	6	9%
				6	6%		
skupina 6	6	7	100%	8	100%	5	100%
skupina 7	21	8	90%	3	100%	2	100%
		0	10%				

Tabulka 4: Analýza klasifikační metody K-Means pro různě zvolené hodnoty parametru *k*. Hodnota *k* odpovídá počtu tříd. Sloupec počet uvádí počet dokumentů v dané skupině, sloupec zastoupení uvádí procentuální zastoupení dokumentů dané třídy v dané skupině.

Na obrázku 13 můžeme sledovat strukturu popsanou v tabulce 2. V souboru se nachází dva sloupce se třídami, každý ze sloupců je výstupem jiné klasifikační metody. Sloupce obsahující třídy byly pro přehlednost dodatečně obarveny. Z obrázku je patrné, že každá klasifikační metoda používá nezáporná celá čísla (tedy včetně nuly) pro označení třídy, metoda DBSCAN navíc používá hodnotu -1 pro dokumenty, které identifikovala jako šum (nezařadila je do žádné třídy). Podrobnější analýza výstupu klasifikace metodou DBSCAN se nachází v tabulce 3 a metodou K-Means v tabulce 4.

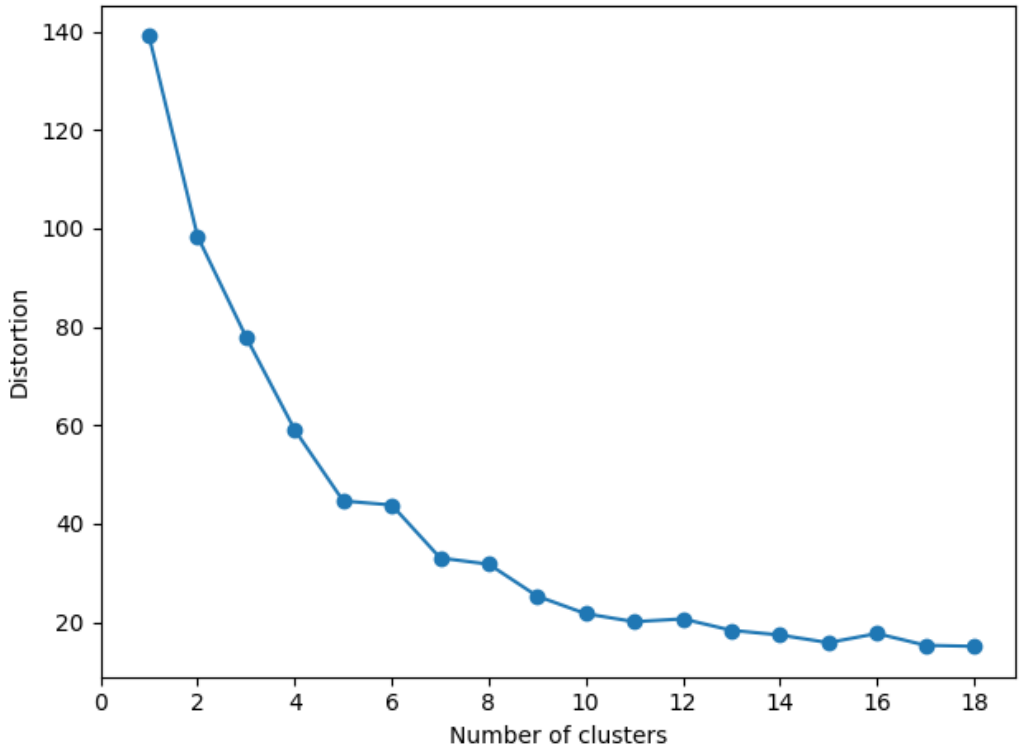
Pro účely srovnání obou metod byly pro obě metody vybrány trojice parametrů tak, aby byl výstupní počet tříd u obou metod stejný. V tabulce 3 si lze povšimnout, že metoda DBSCAN pro volbu *epsilon = 0.5* nebyla schopná zařadit velké množství protokolů do žádné třídy (hodnota -1), někdy nebyla schopná celé skupiny protokolů identifikovat jako samostatné třídy (např. skupinu 3 a skupinu 6). Díky postupnému zvyšování parametru *epsilon* byla metoda schopná identifikovat větší množství protokolů a pro hodnotu parametru *epsilon = 1* bylo dosaženo přesné shody. Pokud by ovšem tato metoda byla aplikována na neznámý dataset s neznámým počtem tříd, může být vhodné stanovení parametru *epsilon* obtížné, protože neexistuje žádná pevná reference dle níž by bylo možné určit, zda je stanovený počet tříd správný.

Metoda K-Means (tabulka 4) vykazuje velice dobré výsledky pro všechny tři volby parametru *k*. Ani v jednom případě nebylo sice dosaženo přesné shody, nicméně třídy byly určeny více méně správně, všechny dokumenty byly přiřazeny nějaké třídě, popřípadě došlo k jemnějšímu dělení do více tříd v rámci jedné skupiny protokolů. V tabulce 4 si lze rovněž povšimnout skutečnosti, že skupiny 2 a 6 byly metodou K-Means ve dvou případech identifikovány jako jedna třída, neboť protokoly od příslušných dvou zhotovitelů mají podobnou stavbu.

Aby mohla být metoda K-Means použita pro neznámý dataset, je třeba nejprve určit přibližný počet tříd, který lze v datasetu očekávat. Pro tento účel byla využita metoda lokte (kapitola 4.5.3.1). Pro sestavení grafu byl využit následující kód:

```
distortions = []
for i in range(1, 19):
    km = KMeans(n_clusters=i, init='k-means++', max_iter=100, n_init=1)
    km.fit(data)
    distortions.append(km.inertia_)
plt.plot(range(1, 19), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.tight_layout()
plt.xticks(np.arange(0, 20, 2))
plt.show()
```

Uvedený kód provádí osmnáct iterací pro parametr $k = 1$ až $k = 18$ a počítá chybovost metody pro daný počet tříd. Výsledný graf pro náš zkušební dataset se nachází na obrázku 14.



Obrázek 14: Výstup metody lokte (Elbow Method).

Obrázek 14 neukazuje žádný významný zlom, který by jednoznačně určoval optimální počet tříd, přesto lze z tvaru výsledné křivky usoudit, že optimální počet tříd bude pravděpodobně někde mezi $k = 5$ a $k = 8$.

Počty protokolů v jednotlivých skupinách jsou velmi nerovnoměrně rozdělené, skupiny 4 a 5 obsahují větší množství přes 50 protokolů, skupiny 2 a 6 jsou naopak velmi malé. Obě klasifikační metody si byly schopné s touto nerovnováhou poradit a na výsledné určení tříd tato skutečnost nehrála významný vliv. To je dobré znamení, protože u datasetu neznámých protokolů se rovněž těžko podaří zajistit stejnoměrné rozdělení jednotlivých skupin protokolů a je proto dobře, že to pro vybrané klasifikační metody neznamená žádnou překážku.

Pro aplikaci ve výsledném programu byla nakonec vybrána metoda K-Means, protože u ní lze díky metodě lokte určit přibližný počet tříd, které lze v datasetu očekávat a protože vždy všechny protokoly bez výjimky přiřadí nějaké určité třídě.

5.2 Čtení dat z dokumentů

Pro část algoritmu hledající potřebnou informaci v textu je důležitá správná klasifikace dokumentů. Algoritmus pro čtení dat se spoléhá na nalezení správné pozice informace v textu.

K tomu potřebuje, aby skupina protokolů, na nichž je model trénován, byla co nejpodobnější a požadovaná informace v nich byla na co nejpodobnějším místě.

Výstup čtení dat z dokumentů, které bylo realizováno pomocí SML metody rozhodovacího stromu, se nachází v tabulkách 5, 6 a 7. Mezi výstupy pro jednotlivé typy informací jsou na první pohled patrné velké rozdíly.

V případě čtení informace o typu rentgenového zařízení (tabulka 5), lze z tabulky vyčíst, že u způsobů, kde byl pro trénink modelu použit dataset pouze deseti protokolů, jsou výsledky velmi chabé. Ani dataset o počtu 55 protokolů se nezdá plně dostačující. V případě způsobu 1 a 3 se algoritmus několikrát trefil do řádku bezprostředně sousedícím s požadovaným řádkem (oranžově podbarvená pole). Z tabulky 5 lze vyčíst, že na zpřesnění výsledků nemá tak výrazný vliv zvětšení datasetu tréninkových dokumentů, jako spíš implementace vlastnosti rozpoznávání zakázaných výrazů. V případě způsobu 2 a 4 byly zakázány výrazy dvojtečka (:), tečka (.) a prázdný řádek. Pakliže algoritmus při svém odhadu určil buď prázdný řádek, nebo řádek obsahující jeden ze zakázaných výrazů, pokusil se učinit nový odhad. Nejlepších výsledků bylo dosaženo aplikací způsobu 4, tedy použitím většího datasetu pro tréning modelu a aplikací vlastnosti rozpoznávání zakázaných výrazů. Je potřeba rovněž zmínit, že informace o typovém označení rentgenového zařízení se v protokolu nachází relativně blízko začátku, ale málokdy se nachází vždy na přesně stejném řádku.

Informace o firmě zhotovující protokol byla naopak nalezena téměř pokaždé (tabulka 6), bez ohledu na velikost tréninkového datasetu a další úpravy algoritmu. Tato informace se v protokolech nachází velmi blízko začátku a nachází se téměř pokaždé na stejném řádku, pro algoritmus bylo proto velice snadné tuto informaci správně identifikovat. V případě způsobu 3 a 4 v jednom případě nebyla informace o firmě nalezena. Příčinnou je pravděpodobně stochastický charakter ML metod, nic neindikuje, že by byl způsob 3 a 4 pro určení obdobné informace méně vhodný, než způsob 1 a 2.

Nejhorší shody bylo dosaženo při hledání infoirmace o dávkové veličině (tabulka 7). Při použití způsobu 4 byla v několika případech nalezena správná informace, ale chybovost odhadů v ostatních případech byla značná. Tento výsledek není nijak překvapivý, protože hledaná informace se v protokolech nachází téměř ke konci, ale co je důležitější, existují rovněž značné rozdíly v poloze dané informace, protože u daného testu existuje několik různých podpřípadů, jakým způsobem se daná dávková veličina stanovuje, a proto se tato veličina vždy nachází v textu na trochu jiném místě. Vzhledem k okolnostem bylo tedy naopak překvapením, že byl algoritmus v případě způsobu 4 schopen si i s takto vysokou variabilitou poradit a informaci v textu hned v několika případech nalézt. Přesto pouhá aplikace metody rozhodovacího stromu pravděpodobně pro vyhledávání takového typu informace není příliš vhodná. Pro extrakci podobného typu informace by bylo vhodnější nalézt více komplexní řešení, možná prozkoumat i možnosti algoritmů nesouvisejících se strojovým učením.

Velikost tréninkového datasetu v počtu pouhých 10, nebo 50 dokumentů je relativně malá a značným zvětšením tréninkového datasetu by pravděpodobně mohlo být dosaženo lepších výsledků. Důležité je však si uvědomit, že čím větší bude tréninkový dataset, tím větší množství času je nutné investovat do manuálního vyhledávání potřebných informací. Navíc, kdyby měl být program aplikován na neroztřízenou skupinu dokumentů, musela by být velikost tréninkového datasetu značně větší, aby každá třída měla dostatečné zastoupení v tréninkovém datasetu, ale v takovém případě již značně narůstá množství dokumentů, které je nutno zpracovat ručně. Vzhledem k tomu, že má program šetřit co největší množství času uživatele, byla jeho výkonnost testována na malé až střední velikosti tréninkového datasetu pro otestování, zda-li i takto malé množství dokumentů postačuje pro trénink modelu. Z výsledků v tabulkách 5, 6 a 7 lze učinit závěr, že i model trénovaný na takto malém datasetu může dosahovat uspokojivých výsledků, je-li aplikován pro extrakci informací, které se v textu nachází na podobných pozicích.

5.3 Funkční program

Součástí výsledků práce je rovněž funkční program. Přepisy kódů jednotlivých souborů programu se nachází v příloze A. Program poskytuje uživateli rovněž zpětnou vazbu v podobě textového výstupu, v níž uživatele informuje o úspěšnosti provedení jednotlivých operací. Ukázka textového výstupu programu se nachází v příloze B.

		text přečtený metodou rozhodovacího stromu - typové označení rentgenového zařízení			
	očekávaný text	**způsob 1**	**způsob 2**	**způsob 3**	**způsob 4**
1	X-Mind DC	Výtisk č. :		Satelec	X-Mind DC Satelec
2	X-Mind DC	Kryt rentgenky:		Tubus:	X-Mind DC
3	ProX	výrobní číslo		Planmeca Oy	jméno a přijmení
4	CS 2200	Kryt rentgenky:			
5	CS 2200	Kryt rentgenky:			
6	Prostyle Intra	Prostyle Intra	výrobní číslo Prostyle Intra		Prostyle Intra
7	RX DC Hypersphere	RX DC Hypersphere	RX DC Hypersphere		HyperSphere
8	RX DC Myray	CEFLA s.c.r.l.	RX DC Myray	CEFLA s.c.r.l.	RX DC Myray
9	Gendex eXpert DC	ZAŘÍZENÍ		Gendex	Gendex eXpert DC Gendex
10	Mindent 70 DC	BMT a.s.	Mindent 70 DC		Rozdělovník
	počet dokumentů:	**10**	**10**	**55**	**55**
	zakázané výrazy:	**ne**	**ano**	**ne**	**ano**

Tabulka 5: Ukázka dat přečtených metodou rozhodovacího stromu. Čtení bylo provedeno čtyřmi způsoby, doplňující informace k jednotlivým způsobům jsou uvedeny pod tabulkou: počet dokumentů značí kolik dokumentů bylo použito pro trénink modelu, zakázané výrazy značí, zda byla v algoritmu použita dodatečná vlastnost detekce zakázaných výrazů. Zeleně podbarvená pole označují správně uhodnuté informace, oranžově podbarvená pole označují případ, kdy se algoritmus trefil do řádku bezprostředně sousedícího s hledanou informací. Text obsahující citlivé informace byl nahrazen červeným textem popisujícím jaká informace se v textu původně nacházela.

		text přečtený metodou rozhodovacího stromu - zhotovitel			
	očekávaný text	**způsob 1**	**způsob 2**	**způsob 3**	**způsob 4**
1	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.
2	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.
3	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.
4	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.
5	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.
6	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.
7	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.
8	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.
9	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	IČ: číslo	
10	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.	Firma XY s. r. o.
	počet dokumentů:	**10**	**10**	**55**	**55**
	zakázané výrazy:	**ne**	**ano**	**ne**	**ano**

Tabulka 6: Ukázka dat přečtených metodou rozhodovacího stromu. Čtení bylo provedeno čtyřmi způsoby, doplňující informace k jednotlivým způsobům jsou uvedeny pod tabulkou: počet dokumentů značí kolik dokumentů bylo použito pro trénink modelu, zakázané výrazy značí, zda byla v algoritmu použita dodatečná vlastnost detekce zakázaných výrazů. Zeleně podbarvená pole označují správně uhodnuté informace. Text obsahující citlivé informace byl nahrazen červeným textem popisujícím jaká informace se v textu původně nacházela.

		text přečtený metodou rozhodovacího stromu - kerma			
	očekávaný text	**způsob 1**	**způsob 2**	**způsob 3**	**způsob 4**
1	Ki = 1,60 mGy	5.3 Polotloušťka	0,30		Ki = 1,60 mGy
2	Ki = 0,51 mGy				
3	Ki = 0,60 mGy	Kerma na konci...	Rozlišení při vysokém...	vzdálenost ohnisko - detektor	
4	Ki = 0,77 mGy	Strana: 4/9		Na snímku se nesmí...	Ki = 0,77 mGy
5	Ki = 0,77 mGy	Strana: 4/9		Na snímku se nesmí...	Ki = 0,77 mGy
6	Ki = 0,36 mGy	ANO		Na snímku fantomu nesmí...	
7	Ki = 0,49 mGy		3,5 x 4,5 cm	Přítomnost artefaktů...	Přítomnost artefaktů...
8	Ki = 0,74 mGy		Ki = 0,74 mGy		Ki = 0,74 mGy
9	Ki = 0,82 mGy	Strana: 4/7	Strana: 4/7	shoda	U
10	Ki = 1,46 mGy	Referenční OD v intervalu...	Rozlišení při vysokém...	60 kV ≥ 1,5	Minimální OD ≤ 0,3...
	počet dokumentů:	**10**	**10**	**50**	**50**
	zakázané výrazy:	**ne**	**ano**	**ne**	**ano**

Tabulka 7: Ukázka dat přečtených metodou rozhodovacího stromu. Čtení bylo provedeno čtyřmi způsoby, doplňující informace k jednotlivým způsobům jsou uvedeny pod tabulkou: počet dokumentů značí kolik dokumentů bylo použito pro trénink modelu, zakázané výrazy značí, zda byla v algoritmu použita dodatečná vlastnost detekce zakázaných výrazů. Zeleně podbarvená pole označují správně uhodnuté informace.

6 Závěr

Cílem této bakalářské práce bylo vytvořit program v jazyce Python, který dokáže číst předem definované informace z definované skupiny PDF dokumentů. S využitím metod strojového učení se podařilo naplnit tento cíl a vytvořit program, který umí 1) klasifikovat dokumenty metodou DBSCAN a K-Means a 2) číst předdefinované informace z daných dokumentů pomocí metody rozhodovacího stromu. Program byl testován na skupině protokolů z měření na rentgenových zařízeních. Čtení definovaných informací bylo testováno na jedné vybrané třídě protokolů s velikostí tréninkového datasetu v počtu 10, nebo 50 protokolů. I s takto malým tréninkovým datasetem bylo dosaženo relativně uspokojivých výsledků v případě, že se definovaná informace v dokumentech nacházela na přibližně podobném místě a když byl algoritmus doplněn o vlastnost, kdy z výsledků vylučoval skupinu definovaných zakázaných výrazů. V případě, kdy se informace v dokumentech nacházely na velmi rozdílných místech, metoda rozhodovacího stromu již nebyla příliš efektivní a tréninkový dataset v počtu 50 protokolů byl nedostatečný.

7 Seznam použitých zdrojů

1. KANADE, Vijay. *What is Artificial Intelligence (AI)? definition, types, goals, challenges, and trends in 2022* [online]. 2022 [cit. 2022-10-19]. Dostupné z: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-ai/.
2. COPELAND, Brian J. *Artificial Intelligence* [online]. 2022 [cit. 2022-10-19]. Dostupné z: https://www.britannica.com/technology/artificial-intelligence/Alan-Turing-and-the-beginning-of-AI.
3. KNEBLÍK, Radovan; KONČEL, Jan; MORAVCOVÁ, Vlasta; MORAVEC, Luboš; RICHTER, Jaroslav; TRKOVSKÁ, Dana. *Poznámka o výstavbě matematiky* [online]. MFF UK, 2011 [cit. 2022-10-24]. Dostupné z: https://www2.karlin.mff.cuni.cz/~portal/logika/?page=vyst.
4. COLE, David. *The Chinese Room Argument* [online]. Stanford University, 2020 [cit. 2022-10-24]. Dostupné z: https://plato.stanford.edu/entries/chinese-room/.
5. RUSSELL, Stuart J.; NORVIG, Peter. *Artificial Intelligence: A Modern Approach*. 3. vyd. Pearson Education, 2009. ISBN 978-0-13-604259-4.
6. *Talos* [online]. 2021 [cit. 2022-10-17]. Dostupné z: https://www.greekmythology.com/Myths/Creatures/Talos/talos.html.
7. ANYOHA, Rockwell. *The history of Artificial Intelligence* [online]. Harvard Graduate School of the Arts a Sciences, 2020 [cit. 2022-10-17]. Dostupné z: https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/.
8. TATE, Karl. *History of A.I.: Artificial intelligence (infographic)* [online]. Purch, 2014 [cit. 2022-10-17]. Dostupné z: https://www.livescience.com/47544-history-of-a-i-artificial-intelligence-infographic.html.
9. TURING, Alan M. Computing Machinery and intelligence. *Mind* [online]. 1950, roč. LIX, č. 236, s. 433–460 [cit. 2022-10-18]. Dostupné z DOI: 10.1093/mind/lix.236.433.
10. *Turing test in Artificial Intelligence* [online]. 2022 [cit. 2022-10-24]. Dostupné z: https://www.geeksforgeeks.org/turing-test-artificial-intelligence/.
11. KOVÁŘ, Petr. *Obecný přehled generací počítačů* [online]. [B.r.] [cit. 2022-10-18]. Dostupné z: https://historiepocitacu.cz/obecny-prehled-generaci-pocitacu.html.

12. *General problem solver (A. newell & H. simon)* [online]. 2018 [cit. 2022-10-18]. Dostupné z: https://www.instructionaldesign.org/theories/general-problem-solver/.

13. NORMAN, Jeremy. *Joseph Weizenbaum writes Eliza: A pioneering experiment in Artificial Intelligence Programming* [online]. [B.r.] [cit. 2022-10-19]. Dostupné z: https://www.historyofinformation.com/detail.php?id=4137.

14. BREAZEAL, Cynthia. Emotion and sociable humanoid robots. *International Journal of Human-Computer Studies*. 2003, roč. 59, č. 1-2, s. 119–155. Dostupné z DOI: 10.1016/s1071-5819(03)00018-1.

15. SHA, Arjun. *22 examples of artificial intelligence you're using in Daily Life* [online]. 2021 [cit. 2022-10-19]. Dostupné z: https://beebom.com/examples-of-artificial-intelligence/.

16. *Umělá inteligence Alzee Zlepšuje Zákaznický Servis v alza.cz* [online]. [B.r.] [cit. 2022-10-24]. Dostupné z: https://www.alza.cz/umela-inteligence-alzee-zlepsuje-zakaznicky-servis.

17. KOĎOUSKOVÁ, Barbora. *Umělá inteligence (AI): Historie a trendy pro rok 2022* [online]. 2022 [cit. 2022-10-19]. Dostupné z: https://www.rascasone.com/cs/blog/umela-inteligence-ai-trendy.

18. IBM CLOUD EDUCATION. *Machine Learning* [online]. 2020 [cit. 2022-10-20]. Dostupné z: https://www.ibm.com/cz-en/cloud/learn/machine-learning.

19. *Machine learning models* [online]. [B.r.] [cit. 2022-10-24]. Dostupné z: https://www.javatpoint.com/machine-learning-models.

20. RASCHKA, Sebastian; MIRJALILI, Vahid. *Python Machine Learning: Machine Learning and deep learning with python, scikit-learn, and tensorflow 2*. 3. vyd. Packt Publishing, 2019. ISBN 978-1-78355-513-0.

21. LACEY, Michelle [online]. 1998 [cit. 2022-10-24]. Dostupné z: http://www.stat.yale.edu/Courses/1997-98/101/linmult.htm.

22. *Machine learning polynomial regression* [online]. [B.r.] [cit. 2022-10-24]. Dostupné z: https://www.javatpoint.com/machine-learning-polynomial-regression.

23. CHEN, James. *What is a neural network?* [Online]. 2022 [cit. 2022-10-24]. Dostupné z: https://www.investopedia.com/terms/n/neuralnetwork.asp.

24. NECKAR, Jan. *Bayesova věta* [online]. 2016 [cit. 2022-10-24]. Dostupné z: https://www.algoritmy.net/article/45037/Bayesova-veta.

25. HONZÍK, Petr. *Bayesovské učení* [online]. FEKT VUT, 2014 [cit. 2022-10-24]. Dostupné z: https://docplayer.cz/92557678-Obsah-prednasky-1-bayesuv-teorem-6-naivni-bayesovsky-klasifikator-nbk.html.

26. *Naive Bayes classifiers* [online]. 2022 [cit. 2022-10-24]. Dostupné z: https://www.geeksforgeeks.org/naive-bayes-classifiers/.

27. ŠPANĚL, Michal; BERAN, Vítězslav. *Obrazové segmentační techniky - Přehled existujících metod* [online]. 2006 [cit. 2022-10-25]. Dostupné z: http://www.fit.vutbr.cz/~spanel/segmentace/.

28. LI, Susan. *A gentle introduction on Market Basket Analysis - Association rules* [online]. 2017 [cit. 2022-10-25]. Dostupné z: https://towardsdatascience.com/a-gentle-introduction-on-market-basket-analysis-association-rules-fa4b986a40ce.

29. *What is web usage mining?* [Online]. 2022 [cit. 2022-10-25]. Dostupné z: https://www.geeksforgeeks.org/what-is-web-usage-mining/.

30. ODEMAKINDE, Elisha. *Model-based and model-free reinforcement learning: Pytennis case study* [online]. 2022 [cit. 2022-10-25]. Dostupné z: https://neptune.ai/blog/model-based-and-model-free-reinforcement-learning-pytennis-case-study.

31. *Bellman equation* [online]. 2021 [cit. 2022-10-25]. Dostupné z: https://www.geeksforgeeks.org/bellman-equation/.

32. JAGTAP, Rohan. *Understanding markov decision process (MDP)* [online]. Towards Data Science, 2021 [cit. 2022-10-25]. Dostupné z: https://towardsdatascience.com/understanding-the-markov-decision-process-mdp-8f838510f150.

33. BROWNLEE, Jason. *Overfitting and underfitting with machine learning algorithms* [online]. 2019 [cit. 2022-10-25]. Dostupné z: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/?_cf_chl_tk=j9vTkNTkbY10OrDi4jy6bPv5PKxw85DePQA0GnOp7c4-1666698488-0-gaNycGzNCGU.

34. *What is resampling? (with definition and types)* [online]. 2022 [cit. 2022-10-28]. Dostupné z: https://www.indeed.com/career-advice/career-development/what-is-resampling.

35. *Difference between deterministic and non-deterministic algorithms* [online]. 2022 [cit. 2022-10-28]. Dostupné z: https://www.geeksforgeeks.org/difference-between-deterministic-and-non-deterministic-algorithms/.

36. STEWART, Matthew. *The limitations of machine learning* [online]. 2020 [cit. 2022-10-25]. Dostupné z: https://towardsdatascience.com/the-limitations-of-machine-learning-a00e0c3040c6.

37. *Mammography* [online]. U.S. Department of Health a Human Services, [b.r.] [cit. 2022-10-28]. Dostupné z: https://www.nibib.nih.gov/science-education/science-topics/mammography.

38. DESANTIS, Carol E.; FEDEWA, Stacey A.; GODING SAUER, Ann; KRAMER, Joan L.; SMITH, Robert A.; JEMAL, Ahmedin. Breast cancer statistics, 2015: Convergence of incidence rates between black and white women. *CA: A Cancer Journal for Clinicians*. 2015, roč. 66, č. 1, s. 31–42. Dostupné z DOI: 10.3322/caac.21320.

39. YEDJOU, Clement G.; SIMS, Jennifer N.; MIELE, Lucio; NOUBISSI, Felicite; LOWE, Leroy; FONSECA, Duber D.; ALO, Richard A.; PAYTON, Marinelle; TCHOUNWOU, Paul B. Health and racial disparity in breast cancer. *Advances in Experimental Medicine and Biology*. 2019, s. 31–49. Dostupné z DOI: 10.1007/978-3-030-20301-6_3.

40. CHRISTIAN, Jon. *Statistician: Machine learning is causing a "crisis in science"* [online]. Futurism, 2019 [cit. 2022-11-03]. Dostupné z: https://futurism.com/neoscope/machine-learning-crisis-science.

41. *Ionizující záření: NZIP* [online]. NZIP, [b.r.] [cit. 2022-11-04]. Dostupné z: https://www.nzip.cz/rejstrikovy-pojem/3236.

42. *Atomové Právo* [online]. SÚJB, [b.r.] [cit. 2022-11-04]. Dostupné z: https://www.sujb.cz/legislativa/atomove-pravo.

43. SBÍRKA ZÁKONŮ ČESKÁ REPUBLIKA. *422/2016 Sb. Vyhláška o radiační ochraně a zabezpečení radionuklidového zdroje*. 2016.

44. TEAM, Codecademy. *7 Top Machine Learning Programming languages* [online]. Codecademy News, 2022 [cit. 2022-11-03]. Dostupné z: https://www.codecademy.com/resources/blog/machine-learning-programming-languages/.

45. AHADH, Abdhul; BINISH, Govind Vallabhasseri; SRINIVASAN, Rajagopalan. Text mining of accident reports using semi-supervised keyword extraction and topic modeling. *Process Safety and Environmental Protection*. 2021, roč. 155, s. 455–465. Dostupné z DOI: 10.1016/j.psep.2021.09.022.

8 Seznam obrázků, tabulek a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam použitých zkratek

AI	-	Artifical inteligence (Umělá inteligence)
ML	-	Machine learning (Strojové učení)
SML	-	Supervised Machine Learning (Učení s učitelem)
UML	-	Unsupervised Machine Learning (Učení bez učitele)
SSML	-	Semi-Supervised Machine Learning (Částečné učení s učitelem)
RML	-	Reinforcement Machine Learning (Zpětnovazební učení)
NPC	-	Non-player character (Počítačem vytvořený herní charakter)
DBSCAN	-	Density-Based Spatial Clustering of Applications with Noise (Název algoritmu)

Příloha A Přepisy zdrojových kódů

A.1 pars_libs.py

```
import os
from os.path import splitext
import sys
import shutil
import io
import numpy as np
import pandas as pd
import csv
import matplotlib.pyplot as plt
import filecmp

from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfpage import PDFPage

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.tree import DecisionTreeClassifier


"""
================================================================================
USER INFORMATION
================================================================================
- place all new PDFs to the /PDF/newData directory
- pick some of the new PDFs and fill out their data to /PDF/labels.xlsx
- fill the variables in 'USER DEFINED variables' section!
- run /main.py
let the AI take care of the rest...
--------------------------------------------------------------------------------
WARNINGS:
- don't change any file or folder names
--------------------------------------------------------------------------------
```

```
NOTES:
- program will place PDFs correctly within the newData or trainingData directories,
  you don't have to care about it
- program will treat every determined class individually, training and applying
  reading algorithm always on only one class at time
-------------------------------------------------------------------------------
"""

# -------------------------------------------------------------------------------
# USER DEFINED variables
# -------------------------------------------------------------------------------

# define column from labels to process
feature = 'company'

# define forbidden expressions for given feature
forbidden = ['Měření', 'IČ:']

# -------------------------------------------------------------------------------
# no-changes-needed variables
# -------------------------------------------------------------------------------

# number of characters to read from TXTs within the classification part, selection
# should contain firm header, not confusing AI with rest of the document
charNo = 300  # setting to -1 will mean to read whole TXT

# length of lines splitting captions in console prints
caption_bar_length = 40
caption_bar_char1 = '='
caption_bar_char2 = '-'

# default working folder (folder where main.py is placed)
defaultDir = os.getcwd()

# following folders will be automatically created, if non existent
pdf_mainDir = fr'{defaultDir}\PDF'
pdf_trainingDataDir = fr'{pdf_mainDir}\trainingData'
pdf_newDataDir = fr'{pdf_mainDir}\newData'
ai_mainDir = fr'{defaultDir}\AI'
txt_mainDir = fr'{ai_mainDir}\TXT'
```

```
77
78  # source/result files
79  labels = fr'{pdf_mainDir}\labels.xlsx'
80  manager = fr'{ai_mainDir}\manager.xlsx'
81  train_data = fr'{ai_mainDir}\train_data.csv'
```

A.2 func.py

```
from pars_libs import *

# ------------------------------------------------------------------------------
# simple functions
# ------------------------------------------------------------------------------

# checks folder existence - creates it if not existent
def checkFolder(path):
    """
    :param path: folder path
    :return -

    Checks if given folder exists. Only when not, creates the folder including all
    parent directories and informs user changes were made.
    """

    pathExist = os.path.exists(path)
    os.makedirs(path, exist_ok=pathExist)
    if not pathExist:
        print(f"- new directory created. {path}")


# terminates the program (console friendly)
def terminate():
    """
    :return -

    Terminating the program in a dignified manner when running from console. User
    will be informed what probably went wrong and the window will be closed after
    user interaction.
    """

    input("\nWorking data discrepancy or absence, program will be terminated."
          "\nMake sure there are some records in /PDF/labels.xlsx file, and that "
          "those records have corresponding PDFs available at /PDF/trainingData."
          "\nPress enter to exit.")
    sys.exit()


```

```
40 # lists all files with certain extension in defined folder
41 def listFiles(path, extension):
42     """
43     :param path: folder path
44     :param extension: type of extension to find ('.pdf', '.txt', ...)
45                       'ignore extension' option will result in listing all files
46     :return LIST
47 
48     Returns files in given folder (either all, or those with chosen extension).
49     """
50 
51     if extension == 'ignore extension':
52         fileList = [f for f in os.listdir(path)
53                     if os.path.isfile(os.path.join(path, f))]
54     else:
55         fileList = [f for f in os.listdir(path)
56                     if os.path.isfile(os.path.join(path, f)) and extension in f]
57 
58     return fileList
59 
60 
61 # get number of line in txt containing certain text
62 def getLineNumber(fileName, searchString):
63     with open(fileName, encoding="utf8") as f:
64         for i, line in enumerate(f, start=1):
65             if searchString in line:
66                 return i
67 
68 
69 # joins list of lists into one flatt list
70 def flatten(listOfLists):
71     """
72     :param listOfLists: list of multiple list variables
73     :return LIST
74 
75     Function takes values from multiple lists and joins them into single list.
76     """
77 
78     flatted_list = [item for sublist in listOfLists for item in sublist]
79 
```

```
80      return flatted_list
81
82
83  # converts PDF to text
84  def convertPdfToText(sourcePath):
85      """
86      :param sourcePath: path to PDF file
87      :return STRING
88
89      This is simple PDF to text converter.
90      """
91
92      with io.StringIO() as retstr:
93          with TextConverter(PDFResourceManager(), retstr, codec='utf-8',
94                             laparams=LAParams()) as device:
95              with open(sourcePath, 'rb') as f:
96                  interpreter = PDFPageInterpreter(PDFResourceManager(), device)
97                  for page in PDFPage.get_pages(f, set(), maxpages=0, password='',
98                                                caching=True, check_extractable=True):
99                      interpreter.process_page(page)
100
101                 return retstr.getvalue()
102
103
104 # converts TXT to one string
105 def txtToString(sourcePath, extent=-1):
106     """
107     :param sourcePath: path to TXT file
108     :param extent: if set to a positive number, it will return first x characters
109                    of TXT file, otherwise it will return whole TXT as one string
110     :return STRING
111     """
112
113     with open(sourcePath, 'r', encoding='utf8') as f:
114         string = f.read()
115         if extent > 0:
116             string = string[:extent]
117
118     return string
119
```

```python

# ------------------------------------------------------------------------------
# complex functions (depending on other functions)
# ------------------------------------------------------------------------------

# checks agreement between labels and documents, ensures right document placement
def checkLabels(sourcePath, trainingDataDir, newDataDir):
    """
    :param sourcePath: path to labels file
    :param trainingDataDir: path to training PDFs
    :param newDataDir: path to new PDFs
    :return -

    This is a complex function ensuring all documents within a PDF folder are
    correctly placed
    1) checks duplicates between trainingData and newData folders, keep files in
       trainingData and deletes duplicates from newData folder
    2) checks labels.xlsx records and ensures:
       - all files with a record (labeled data) are placed in trainingData
       - the rest (unlabeled data) is placed in newData
       - records lacking a corresponding PDF file are deleted
    """

    # get a list of all files with a record in labels (exclude nan values)
    labels_df = pd.read_excel(sourcePath, index_col=0)
    labels_files = labels_df.index.values.tolist()
    labels_files = [x for x in labels_files if str(x) != 'nan']
    #print(labels_files)

    # get a list of all PDF files in trainingData folder
    training_files = listFiles(trainingDataDir, '.pdf')
    #print(training_files)

    # check if files from trainingData have some duplicates in newData, delete them
    duplicates = filecmp.cmpfiles(trainingDataDir, newDataDir, training_files)[0]
    if duplicates:
        print(f"- deleting duplicates from newData. {duplicates}")
        for file in duplicates:
            os.remove(fr'{newDataDir}\{file}')
```

```
160
161     # get a list of all PDF files in newData folder
162     new_files = listFiles(newDataDir, '.pdf')
163     #print(new_files)
164
165     # iterate through labeled documents and check their placement/existence
166     move_files = []
167     drop_files = []
168     for file in labels_files:
169         if file in training_files:
170             # document exists and is correctly placed
171             pass
172         elif file in new_files:
173             # document exists but is wrongly placed
174             move_files.append(file)
175         else:
176             # document doesn't exist in expected folders
177             drop_files.append(file)
178
179     # if there are some misplaced documents, move them
180     if move_files:
181         print(f"- moving from newData to trainingData. {move_files}")
182         for file in move_files:
183             shutil.move(fr'{newDataDir}\{file}',
184                         fr'{trainingDataDir}\{file}')
185
186     # if there are some labels lacking corresponding PDF, ask permission and delete
187     if drop_files:
188         answer = input(f"- documents not found. {drop_files} Do you wish to "
189                        r"delete their records in /PDF/labels.xlsx? (y/n)")
190         if answer == 'y':
191             labels_df_drop = labels_df.drop(drop_files)
192             labels_df_drop.to_excel(labels)
193             print(r"-- /PDF/labels.xlsx file updated.")
194         else:
195             terminate()
196
197     # if there are PDFs in trainingData, which lack a label, move them to newData
198     unlabeled = list(set(training_files) - set(labels_files))
199     if unlabeled:
```

```
200         print(f"- moving unlabeled PDFs to newData. {unlabeled}")
201         for file in unlabeled:
202             shutil.move(fr'{trainingDataDir}\{file}',
203                         fr'{newDataDir}\{file}')
204
205
206 # updates manager
207 def updateControlFile(control_file, trainingDataDir, newDataDir, txtDir):
208     """
209     :param control_file: path to manager
210     :param trainingDataDir: path to training PDFs
211     :param newDataDir: path to new PDFs
212     :param txtDir: path to TXTs
213     :return -
214
215     This function checks out content of /PDF and /TXT folders and creates
216     the manager.xlsx file. This file contains information about available PDFs,
217     if they were converted to TXT and if they are labeled or not. Later this file
218     serves the AI algorithms to get and store needed information.
219     """
220
221     # create blank dataFrame
222     control_df = pd.DataFrame(data={'file name': [],
223                                     'location': [],
224                                     'labeled': [],
225                                     'converted': [],
226                                     })
227     control_df = control_df.set_index('file name')
228
229     # get a list of all PDF files in trainingData folder
230     training_files = listFiles(trainingDataDir, '.pdf')
231
232     # get a list of all PDF files in newData folder
233     new_files = listFiles(newDataDir, '.pdf')
234
235     # get a list of all TXT files in TXT folder
236     txt_files = listFiles(txtDir, '.txt')
237
238     # go through all available PDFs
239     for file in flatten([training_files, new_files]):
```

```
240
241         # find out, if file has record in labels
242         if file in training_files:
243             labeled = True
244             location = fr'{trainingDataDir}\{file}'
245         else:
246             labeled = False
247             location = fr'{newDataDir}\{file}'
248
249         # find out, if file was converted to TXT
250         if f'{splitext(file)[0]}.txt' in txt_files:
251             converted = True
252         else:
253             converted = False
254
255         # update control dataFrame
256         new_data_df = pd.DataFrame([[location, labeled, converted]],
257                                    columns=['location', 'labeled', 'converted'],
258                                    index=pd.Index([file], name='file name'))
259         control_df = control_df.append(new_data_df)
260
261     # write changes to control file
262     control_df = control_df.sort_index()  # sorting control dataFrame
263     control_df.to_excel(control_file)
264     print("- /AI/manager.xlsx created.")
265
266
267 # convert all PDF from list to TXT in /TXT folder
268 def pdfToTxt(file_list, txt_targetDir):
269     """
270     :param file_list: list of file paths (those files will be converted)
271     :param txt_targetDir: path to folder, where converted files will be saved
272     :return -
273     """
274
275     # convert only new documents, ignore those already converted
276     if file_list:  # if the list is not blanc, then
277         print(f"- converting {len(file_list)} files.")
278         # progress bar - defining length
279         print("-" * len(file_list))
```

```
280
281         for file in file_list:
282             # extract file name (no path, no extension)
283             file_name = splitext(os.path.basename(file))[0]
284             # define new file path (where TXT will be saved)
285             file_target_path = fr'{txt_targetDir}\{file_name}.txt'
286
287             with open(file_target_path, 'w', encoding='utf-8') as f:
288                 f.write(convertPdfToText(file))
289                 f.close()
290
291             # progress bar to show something is happening (conversion takes time)
292             print("|", end='')
293         print()
294
295
296 # prepares data for a model
297 def classification_dataPre(control_df, sourceDir, no_of_chars):
298     """
299     :param control_df: dataFrame from control file
300     :param sourceDir: path to folder containing TXT files
301     :param no_of_chars: number of first X character from TXT file for the ML to
302            work with (this number should be sufficient to contain all header
303            information, but not too large to confuse ML with rest of the document)
304     :return SPARSE MATRIX, LIST
305     """
306
307     # get data from control file
308     txt_files = [f'{splitext(file_name)[0]}.txt' for file_name
309                  in control_df.index[control_df['converted'] == True].tolist()]
310
311     # convert relevant TXT documents into one list of strings (input data format)
312     documents = []  # to store documents in form of strings
313     file_names = []  # to store corresponding PDF documents names
314     for file_name in sorted(txt_files):
315         documents.append(txtToString(fr'{sourceDir}\{file_name}', no_of_chars))
316         file_names.append(f'{splitext(file_name)[0]}.pdf')
317
318     # pre-process string inputs to vectors
319     vectorization = TfidfVectorizer(max_df=0.7)  # max_df 0.7 - 1.0
```

```
320     X = vectorization.fit_transform(documents)
321
322     return X, file_names
323
324
325 # generates elbow method diagram
326 def classification_elbowMethod(X, max_range):
327     """
328     :param X: vectorized documents data
329     :param max_range: maximum number of classes for the diagram
330     :return: -
331     """
332
333     distortions = []
334     for i in range(1, max_range):
335         km = KMeans(n_clusters=i, init='k-means++', max_iter=100, n_init=1)
336         km.fit(X)
337         distortions.append(km.inertia_)
338     plt.plot(range(1, max_range), distortions, marker='o')
339     plt.xlabel('Number of clusters')
340     plt.ylabel('Distortion')
341     plt.tight_layout()
342     plt.xticks(np.arange(0, (max_range+1), 2))
343     plt.show()
344
345
346 # builds up classification model with a DBSCAN method
347 def classification_DBSCAN(control_df, X, file_names, eps):
348     """
349     :param control_df: dataFrame from control file
350     :param X: vectorized documents data
351     :param file_names: relevant indices of the control file dataFrame
352     :param eps: maximum distance between two samples in one class (model parameter)
353     :return DATAFRAME
354     """
355
356     # create classifying model
357     cl_model = DBSCAN(eps=eps, min_samples=5)
358     cl_model.fit(X)
359     cluster_numbers = cl_model.labels_
```

```
360     total_clusters = len(np.unique(cluster_numbers))
361
362     # save classes and corresponding PDF names in a dataFrame format
363     classification_dic = {'file name': file_names,
364                           'class_DBSCAN': cluster_numbers}
365     predicted_classes_df = pd.DataFrame(data=classification_dic)
366     predicted_classes_df = predicted_classes_df.set_index('file name')
367
368     # merge data from control file with determined classes
369     control_df = pd.concat([control_df, predicted_classes_df], axis=1)
370     print(f"- DBSCAN method used, total {total_clusters} classes determined"
371           f" for epsilon {eps}.")
372
373     return control_df
374
375
376 # builds up classification model with a K-Means method
377 def classification_Kmeans(control_df, X, file_names, clusters):
378     """
379     :param control_df: dataFrame from control file
380     :param X: vectorized documents data
381     :param file_names: relevant indices of the control file dataFrame
382     :param clusters: number of classes
383     :return DATAFRAME
384     """
385
386     # create classifying model
387     cl_model = KMeans(n_clusters=clusters, init='k-means++', max_iter=100, n_init=1)
388     cl_model.fit(X)
389
390     cluster_numbers = cl_model.labels_
391     total_clusters = len(np.unique(cluster_numbers))
392
393     # save classes and corresponding PDF names in a dataFrame format
394     classification_dic = {'file name': file_names,
395                           'class_K-Means': cluster_numbers}
396     predicted_classes_df = pd.DataFrame(data=classification_dic)
397     predicted_classes_df = predicted_classes_df.set_index('file name')
398
399     # merge data from control file with determined classes
```

```
    control_df = pd.concat([control_df, predicted_classes_df], axis=1)
    print(f"- K-Means method used, {total_clusters} classes in total.")

    return control_df


# prepares training dataset
def data_reading_dataPre(feature_name, label_data, output_file):
    """
    :param feature_name: name of column in labels
    :param label_data: dataFrame from labels
    :param output_file: csv training data file
    :return:
    """

    with open(output_file, 'w', newline='', encoding="utf8") as csv_file:
        writer = csv.writer(csv_file)
        writer.writerow(['file', 'text', 'target information', 'line number'])

        for filename in label_data['txt_name'].tolist():
            with open(os.path.join(txt_mainDir, filename),
                      'r', encoding="utf8") as text_file:
                text = text_file.read()
            feature = label_data.loc[label_data['txt_name'] == filename,
                                     feature_name].iloc[0]
            line_number = getLineNumber(os.path.join(txt_mainDir, filename),
                                        feature)

            writer.writerow([filename, text, feature, line_number])


# uses decision tree method to extract information
def data_reading_decisionTree(train_file, new_docsDir, new_docs, output_file):
    """
    :param train_file: csv training data file, created with other function
    :param new_docsDir: the TXT directory
    :param new_docs: list of TXT names of documents to process with trained model
    :param output_file: files containing results
    :return: -
    """
```

```
440
441         # Load the training data (manually labeled examples)
442         train_df = pd.read_csv(train_file)
443
444         # Preprocess the text in the training data
445         vectorizer = CountVectorizer()
446         #vectorizer = TfidfVectorizer()
447         train_X = vectorizer.fit_transform(train_df['text'])
448         #train_X = vectorizer.fit_transform(train_df['text'])
449
450         # Train a decision tree classifier to predict the line number
451         clf = DecisionTreeClassifier()
452         #clf = RandomForestClassifier(n_estimators=100)
453         clf.fit(train_X, train_df['line number'])
454
455         # Create an empty DataFrame to store the extracted information and line numbers
456         extracted_df = pd.DataFrame(columns=['text', 'line_number'])
457
458         # Loop over each new document and extract the target information
459         for doc in new_docs:
460             doc_path = os.path.join(new_docsDir, doc)
461             with open(doc_path, 'r', encoding="utf8") as f:
462                 new_text = f.read()
463
464             # Preprocess the text in the new document
465             new_X = vectorizer.transform([new_text])
466
467             # Use the decision tree classifier to predict the line number
468             predicted_line = clf.predict(new_X)[0]
469
470             # Extract the predicted line from the new document
471             lines = new_text.split('\n')
472             if predicted_line < len(lines):
473                 extracted_info = lines[predicted_line].strip()
474             else:
475                 extracted_info = ''
476
477             # Add the extracted information and line number to the DataFrame
478             extracted_df = extracted_df.append({'text': extracted_info,
479                                                 'line_number': predicted_line},
```

```
                                            ignore_index=True)

    # Write the extracted information and line numbers to the CSV file
    extracted_df.to_excel(output_file, index=False)


# uses decision tree method with addition of forbidden expr. to extract information
def data_reading_decisionTree_expressions(train_file, new_docsDir, new_docs,
                                          forbidden_expressions, output_file):
    """
    :param train_file: csv training data file, created with other function
    :param new_docsDir: the TXT directory
    :param new_docs: list of TXT names of documents to process with trained model
    :param forbidden_expressions: list of string expressions, that can never appear
           in searched feature
    :param output_file: files containing results
    :return: -

    """

    # Load the training data (manually labeled examples)
    train_df = pd.read_csv(train_file)

    # Preprocess the text in the training data
    vectorizer = CountVectorizer()
    train_X = vectorizer.fit_transform(train_df['text'])

    # Train a decision tree classifier to predict the line number
    clf = DecisionTreeClassifier()
    clf.fit(train_X, train_df['line number'])

    # Create an empty DataFrame to store the extracted information and line numbers
    extracted_df = pd.DataFrame(columns=['text', 'line_number'])

    # Loop over each new document and extract the target information
    for doc in new_docs:
        doc_path = os.path.join(new_docsDir, doc)
        with open(doc_path, 'r', encoding="utf8") as f:
            new_text = f.read()

```

```
        # Preprocess the text in the new document
        new_X = vectorizer.transform([new_text])

        # Use the decision tree classifier to predict the line number
        predicted_line = clf.predict(new_X)[0]

        # Search for the desired information in the vicinity of the predicted line
        lines = new_text.split('\n')
        # Search up to 10 lines before the predicted line
        start_line = max(predicted_line - 10, 0)
        # Search up to 10 lines after the predicted line
        end_line = min(predicted_line + 10, len(lines))
        extracted_info = None
        for line_idx in range(start_line, end_line):
            # Extract the line of text and check if it contains any forbidden expr.
            line_text = lines[line_idx].strip()
            if any(expr in line_text for expr in forbidden_expressions):
                continue  # Skip this line if it contains a forbidden expression

            # Extract the desired information from the line
            if line_idx < predicted_line:
                extracted_info = line_text
            elif line_idx == predicted_line:
                if extracted_info is None:
                    extracted_info = line_text
                else:
                    extracted_info += ' ' + line_text
            else:
                break  # Stop searching after the predicted line

        # Add the extracted information and line number to the DataFrame
        extracted_df = extracted_df.append({'file_name': os.path.basename(doc_path),
                                            'text': extracted_info,
                                            'line_number': predicted_line},
                                           ignore_index=True)

    # Write the extracted information and line numbers to the CSV file
    extracted_df.to_excel(output_file, index=False)
```

A.3 main.py

```
from func import *

if __name__ == '__main__':
    # ================================================================================
    # Program initiation
    # ================================================================================
    print('\n' + caption_bar_char1 * caption_bar_length)
    print("Program initiation")
    print(caption_bar_char2 * caption_bar_length)
    # For the successful initiation, there must be some records in /PDF/labels.xlsx
    # and corresponding PDFs must be placed in the trainingData or newData folder.
    # If these conditions are not met, there is no reason to continue.
    # Part of the initiation is also ensuring all PDFs are placed correctly.

    # create working folders, if non existent (stating subdirectories suffice)
    print("\nChecking working folders...")
    for folder in [pdf_trainingDataDir, pdf_newDataDir, txt_mainDir]:
        checkFolder(folder)
    print("Working folders ok.")

    # if excel with labels doesn't exist, terminate
    if not os.path.isfile(labels):
        terminate()

    # check agreement between training PDFs placement and records in labels
    print("\nChecking labels and file placement...")
    checkLabels(labels, pdf_trainingDataDir, pdf_newDataDir)
    print("Labels ok, placement ok.")

    # if there are no files in trainingData folder, terminate
    if not os.listdir(pdf_trainingDataDir):
        terminate()


    # ================================================================================
    # Data processing
    # ================================================================================
    print('\n' + caption_bar_char1 * caption_bar_length)
    print("Data processing")
```

```
40      print(caption_bar_char2 * caption_bar_length)
41      # This section ensures, there is the control /AI/manager.xlsx file and keeps it
42      # up to date. It is safe to delete control file, program will create new one,
43      # but only reason for that would be user intervention in the file, which is
44      # strongly forbidden.
45      # Other task of this section is PDF to TXT conversion, if not done already.
46
47      # update information in manager (create manager if necessary)
48      print("\nCreating manager...")
49      updateControlFile(manager, pdf_trainingDataDir, pdf_newDataDir, txt_mainDir)
50      print("Manager ok.")
51
52      # read updated data from manager
53      manager_df = pd.read_excel(manager, index_col=0)
54
55      # list all files that weren't converted to TXT yet
56      convert_name_list = manager_df.index[manager_df['converted'] == False].tolist()
57      # get paths to those files
58      convert_file_list = []
59      for name in convert_name_list:
60          convert_file_list.append(manager_df.loc[name, 'location'])
61
62      # convert given files to TXT and save them in /TXT folder
63      print("\nPDF to TXT conversion...")
64      pdfToTxt(convert_file_list, txt_mainDir)
65      print("Conversion ok.")
66
67
68      # ============================================================================
69      # AI
70      # ============================================================================
71      print('\n' + caption_bar_char1 * caption_bar_length)
72      print("AI")
73      print(caption_bar_char2 * caption_bar_length)
74      # Unsupervised machine learning is used to all available PDFs (in both training
75      # and new folders) to determine document classes. Results are saved to manager.
76      # Supervised machine learning is used to training PDFs. Model is trained
77      # separately for each document class stated in manager.
78
79      print("\nClassifying documents...")
```

```
80      # get actualized manager information
81      manager_df = pd.read_excel(manager, index_col=0)
82      # preprocess data
83      data, indices = classification_dataPre(manager_df, txt_mainDir, charNo)
84
85      # classify document with DBSCAN method
86      distance = 1
87      manager_df = classification_DBSCAN(manager_df, data, indices, distance)
88
89      # show elbow method diagram for K-means
90      classification_elbowMethod(data, 19)
91
92      # user number of classes input for K-means with check
93      while True:
94          try:
95              classes_number = int(input("- enter number of classes:"))
96              break
97          except ValueError:
98              print("WARNING! You have to enter an integer - try again.")
99              continue
100
101     # classify document with K-Means method
102     manager_df = classification_Kmeans(manager_df, data, indices, classes_number)
103
104     manager_df.to_excel(manager)  # save all to control file
105     print("- classes saved to /AI/manager.xlsx.")
106
107     print("Classification ok.")
108
109
110     print("\nReading documents...")
111
112     # get actualized manager information
113     manager_df = pd.read_excel(manager, index_col=0)
114
115     # read labeled data and preprocess them for a model
116     label_df = pd.read_excel(labels, index_col=0)
117     data_reading_dataPre('company', label_df, train_data)
118     print("- creating /AI/train_data.csv.")
119
```

```
# loop through determined classes
for class_number in set(manager_df['class_K-Means']):

    # define list of new documents to process (always from one class at time)
    # all documents in given class
    docs_list_all = manager_df.index[
        manager_df['class_K-Means'] == class_number].tolist()
    # all new documents to process
    docs_list_new = listFiles(pdf_newDataDir, '.pdf')
    # documents to process from the given class in TXT format
    docs_list = [f.replace('.pdf', '.txt') for f in
                 [x for x in docs_list_all if x in docs_list_new]]

    # define output file name
    output_file = fr'{ai_mainDir}\output{class_number}.xlsx'

    # extract information with decision tree method and forbidden expressions
    if docs_list:
        data_reading_decisionTree_expressions(train_data, txt_mainDir, docs_list,
                                              forbidden, output_file)
        print(f"- saving results for class {class_number} to "
              f"{os.path.basename(output_file)} ({len(docs_list)} documents).")
    else:
        print(f"- no documents to process for class {class_number}.")

print("Reading ok.")
```

Příloha B Textový výstup programu

log.txt

```
========================================
Program initiation
----------------------------------------

Checking working folders...
Working folders ok.

Checking labels and file placement...
- moving from newData to trainingData. ['ra1.pdf', 'ra24.pdf', 'ra40.pdf']
- moving unlabeled PDFs to newData. ['ra64.pdf', 'ra65.pdf']
Labels ok, placement ok.

========================================
Data processing
----------------------------------------

Creating manager...
- /AI/manager.xlsx created.
Manager ok.

PDF to TXT conversion...
Conversion ok.

========================================
AI
----------------------------------------

Classifying documents...
- DBSCAN method used, total 4 classes determined for epsilon 1.
- enter number of classes:3.5
WARNING! You have to enter an integer - try again.
- enter number of classes:3
- K-Means method used, 3 classes in total.
- classes saved to /AI/manager.xlsx.
Classification ok.

Reading documents...
- creating /AI/train_data.csv.
- no documents to process for class 0.
- saving results for class 1 to output1.xlsx 5 documents.
```

```
- saving results for class 2 to output2.xlsx 4 documents.
Reading ok.
```
