VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ZÍSKÁVÁNÍ ZNALOSTI Z DAT V JAZYCE PYTHON

DATAMINING IN PYTHON

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TAMARA KRESTIANKOVÁ**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. JAROSLAV ZENDULKA, CSc.**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Krestianková Tamara**
Obor: Informační technologie
Téma: **Získávání znalosti z dat v jazyce Python**
Data Mining with Python
Kategorie: Data mining

Pokyny:

1. Seznamte se se základy získávání znalostí z dat (data mining).
2. Seznamte se dostupnými prostředky na podporu dolování z dat v jazyce Python.
3. Po dohodě s vedoucím zvolte vhodnou úlohu a vhodná data pro demonstraci použití jazyka Python pro tyto účely.
4. Podrobněji se seznamte s přístupy a algoritmy pro řešení zvolené úlohy a potřebnými knihovnami jazyka Python.
5. Řešení úlohy implementujte a použijte na zvolených datech.
6. Na základě získaných zkušeností diskutujte výhody a nevýhody použití jazyka Python pro řešení zvolené úlohy.

Literatura:

- Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Third Edition. Morgan Kaufmann Publishers, 2012, 703 p., ISBN 978-0-12-381479-1. (Kapitola 1 a další informace relevantní ke zvolené úloze a použitým datům).
- Vettigli, G.: Practical Data Mining with Python. Dostupné na http://guidetodatamining.com/.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 4.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zendulka Jaroslav, doc. Ing., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Táto práca sa zaoberá procesom dolovania dát, prostriedkami pre podporu dolovania dát v programovacom jazyku Python a demonštráciou využitia tohto jazyka pre účely dátovej analýzy, so zameraním na klasifikáciu a tvorbu klasifikačných modelov. Tieto modely sú schopné na základe dát z biomedicínskych hlasových meraní s určitou presnosťou klasifikovať testované subjekty do dvoch katégórií - ľudí trpiacich Parkinsonovou chorobou a zdravých ľudí.

Abstract

This thesis deals with principles of data mining process, available Python packages for data mining and a demonstration of Python script capable of data analyisis focused on classification techniques. Created classifiers are able to classify subjects into two groups - healthy people and people suffering from Parkinson's disease - based on their biomedical vocal analysis data.

Klíčová slova

dátová analýza, dolovanie dát, python, spracovanie dát, scikit-learn, pandas, numpy, klasifikácia, rozhodovacie stromy, parkinsonova choroba

Keywords

data analysis, data mining, python, data processing, scikit-learn, pandas, numpy, classification, decision trees, parkinson's disease

Citace

KRESTIANKOVÁ, Tamara. *Získávání znalosti z dat v jazyce Python.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Jaroslav Zendulka, CSc.

Získávání znalosti z dat v jazyce Python

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne pod vedením pána Doc. Ing. Jaroslava Zendulku, CSc.. Uviedla som všetky literálne pramene a publikácie, z ktorých som čerpala.

.....
Tamara Krestianková
12. května 2018

Poděkování

V tejto časti práce by som sa chcela poďakovať pánovi docentovi Jaroslavovi Zendulkovi za ochotu, čas a cenné rady, ktoré mi poskytol počas tvorby tejto bakalárskej práce.

Obsah

Kapitola 1

Úvod

Žijeme v dobe informácií a denne sa okolo nás vyprodukuje obrovské množstvo dát. Každý užívateľ internetu denne prispieva novými a novými záznammi sociálnym sieťam, internetovým obchodom či iným spoločnostiam. Mnoho krát ľudia nemajú ani najmenšiu predstavu o tom, koľko údajov o ich osobe a aktivite si s ich súhlasom tieto spoločnosti zhromažďujú a ďalej využívajú vo svoj prospech a rast.

Internet však nie je jediná oblasť, ktorá zhromažďuje dáta. Medzi ďalšie oblasti patrí najmä veda a výskum. V tejto oblasti rôzne spoločnosti či organizácie zbierajú dáta potrebné na rôzne štúdie, hypotézy alebo vývoj nových technológií.

Zdroj dát môže byť teda veľmi rôznorodý, no zakaždým sú tieto dáta zhromaždené s rovnakým cieľom, ktorým je získanie nových a užitočných informácií. Samotné dáta zvyčajne na prvý pohľad neposkytujú žiadnu prospešnú informáciu a preto je potrebné ich lepšie analyzovať a nové informácie hľadať na základe rôznych asociácií, ktoré medzi nimi vznikajú. Ktorý človek je však schopný analyzovať *milióny* záznamov? V tomto momente prichádzajú do hry technológie a to konkrétne oblasť zaoberajúca sa analýzou dát, ktorou je získavanie znalostí z dát nazývaná aj data mining.

Cieľom tejto práce je oboznámiť sa so základmi získavania znalostí z dát a s dostupnými prostriedkami na podporu dolovania dát v jazyku Python. Ďalej je cieľom získané znalosti aplikovať a demonštrovať použitie jazyka Python na zvolenej dolovacej úlohe a množine dát a na záver vyhodnotiť výhody a nevýhody použitia práve tohto programovacieho jazyka.

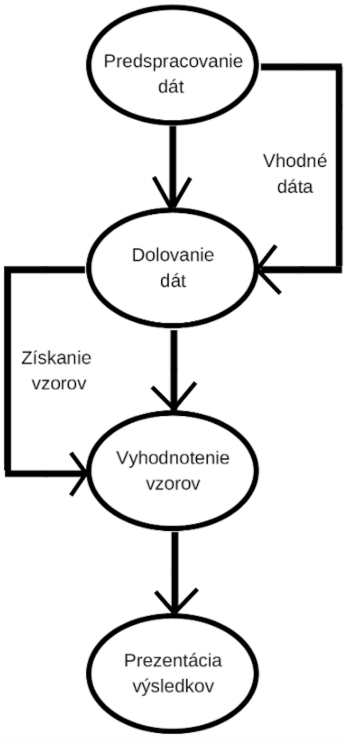
V kapitole 2 je opísaný úvod do získavania znalostí z dát, popis tejto oblasti a jednotlivé kroky tohto procesu. Nasledujúce kapitoly sú venované detajlnejšiemu pohľadu na jednotlivé kroky získavania znalostí z dát, začínajúc kapitolou 3, ktorá hovorí o predspracovaní dát. 4. kapitola opisuje pojem datamining a najčastejšie používané metódy dolovania dát. Nakoľko zvolená dolovacia úloha je typickou klasifikačnou úlohou, tak je tejto metóde venovaná celá kapitola 5. Posledná, 6. kapitola teoretickej časti tejto práce, je venovaná práve dostupným prostriedkom na podporu dolovania dát v jazyku Python. Praktická časť tejto bakalárskej práce demonštruje použitie jazyka Python pre získavanie znalostí z dát, začínajúc 7. kapitolou. Jedná sa o obsiahlejšiu kapitolu, ktorá popisuje zvolenú analytickú úlohu, použitú dátovú sadu a potom aj samotnú implementáciu riešenia tejto úlohy a jej výsledky. Na záver sa v kapitole 8 popisujú výhody a nevýhody použitia jazyka Python pre tieto účely.

Kapitola 2

Získavanie znalostí z databází

Získavanie znalostí z databází je proces získavania a objavovania znalostí spravidla z veľkých objemov dát. Informácie, ktoré je možné vďaka tomuto procesu získať, sú vždy netriviálne, skryté a potenciálne užitočné[21]. Netriviálnosť v tomto prípade znamená, že sa jedná o informácie, ktoré nie je možné získať napríklad jednoduchým výpočtom alebo SQL dotazom nad databázou. Skrytosť informácií zase znamená, že informácie nie je na prvý pohľad vidieť a je nutné tieto informácie určitým *netriviálnym* spôsobom nájsť. Poslednou vlastnosťou získavaných informácií je ich užitočnosť. Užitočnosť získaných informácií sa vždy meria významom pre určitú hypotézu alebo rozhodnutie[35].

Proces získavania znalostí z databází je zložený zo štyroch dôležitých krokov, ktorých postup je graficky znázornený na obrázku 2.1.



Obrázek 2.1: Proces získavania znalostí z databází

Prvým krokom je predspracovanie dát. Jedná sa o proces prípravy a úpravy dát do takého tvaru, ktorý najviac vyhovuje danej dolovacej úlohe. Vďaka predspracovaniu sú z dát odstránené prázdne hodnoty a ďalšie faktory, ktoré potenciálne zhoršujú výsledok a spomaľujú výpočet. Predspracovanie dát bude detajlnejšie opísané v kapitole 3.

Druhý krok, ktorý je zároveň najdôležitejší, je samotné dolovanie dát. V tomto kroku prebieha získavanie požadovaných informácií aplikovaním vhodnej dolovacej metódy na vstupné dáta. Niekedy však nie je na začiatku výskumu jasné, ktorý dolovací model je pre danú úlohu najvhodnejší. Z toho dôvodu sa zvyčajne v procese výskumu skúša viacero metód, aby bolo možné ich porovnať a získať čo najlepšie výsledky. Dolovaniu dát a metódam dolovania dát bude venovaná kapitola 4.

Predposledným krokom je vyhodnotenie modelov a nájdených vzorov. Je potrebné si uvedomiť, že proces dolovania dát môže generovať obrovské množstvo vzorov. Nie všetky sú však *zaujímavé* a žiadúce. Existujú 4 základné vlastnosti, ktoré charakterizujú zaujímavosť získaného vzoru a sú to:

- jednoduchá zrozumiteľnosť pre človeka
- novosť
- potenciálna užitočnosť
- platnosť pre nové alebo testovacie dáta

Vyhodnotenie zaujímavosti vzoru môže byť subjektívna ale aj objektívna záležitosť, ktorú je opäť možné vyčísliť a posúdiť. Získané zaujímavé vzory potom predstavujú *znalosť*[35].

Posledným krokom je prezentácia získaných znalostí. Ako to už vo väčšine výskumov býva zvykom, na konci práce sa prezentujú výsledky. Tieto výsledky môžu byť rôzne vizualizované a reprezentované so snahou zaujať publikum alebo prípadne budúceho užívateľa.

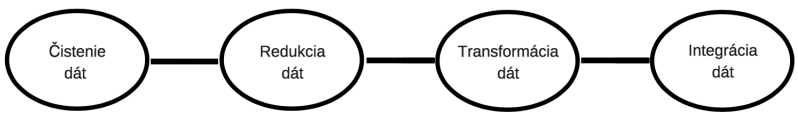
Kapitola 3

Predspracovanie dát

Prvým krokom k úspešnému dolovaniu dát je predspracovanie dát alebo takzvaný *preprocessing*. Databázy reálneho sveta nie sú ani z ďaleka dokonalé. Bežne sa môžeme stretnúť s dátami, ktoré sú:

- nekompletné
- zašumené
- nekonzistentné

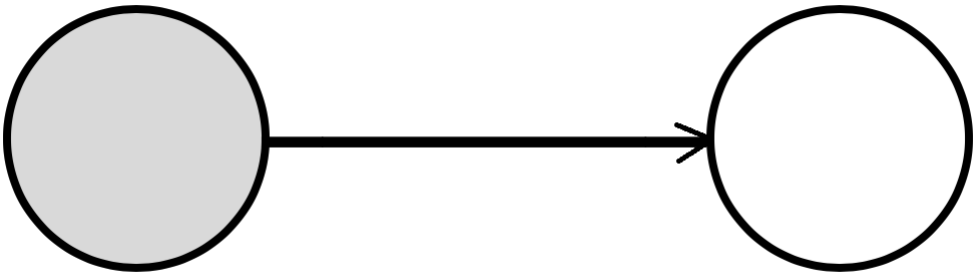
Dôvodov, prečo sa v databáze nachádzajú neúplné dáta môže byť niekoľko. Ako príklad si môžeme predstaviť internetový formulár. Veľa takýchto formulárov obsahuje polia označené ako nepovinné, kde si užívateľ môže vybrať, či danú hodnotu formulára vyplní alebo nie. Dôsledkom toho je potrebné rátať s tým, že hodnota poľa mohla ostať prázdna a neobsahuje žiadnu informáciu[35]. Ďalšou nedokonalosťou je spomínaný šum dát. Ako zašumené označujeme tie dáta, ktoré sú nesprávne a nedávajú vo svojom kontexte zmysel. K takejto chybe môže dôjsť zlyhaním ľudského faktora pri vkládaní dát do databázy alebo tiež zlyhaním technológie, ktorá meria a určuje hodnoty atribútov (napríklad teplomer). Poslednou spomínanou nedokonalosťou databázy reálneho sveta, s ktorou je potrebné počítať a ktorú je nevyhnutné odstrániť je nekonzistentnosť dát. K nekonzistentnosti dát alebo redundantnosti dát môže dochádzať vtedy, keď sa používa niekoľko rôznych dátových zdrojov. V takomto prípade existuje pravdepodobnosť, že sa medzi jednotlivými zdrojmi objaví nekonzistentnosť v pomenovaní, formáte a tak podobne. Ak chceme pri dolovaní dát dosiahnuť čo najlepšie výsledky, je potrebné všetky tieto nedokonalosti odstrániť[21]. Práve to je hlavnou úlohou predspracovania dát. Hlavné kroky predspracovania dát sú graficky znázornené na obrázku 3.1[1].



Obrázek 3.1: Kroky predspracovania dát

[1]Poradie jednotlivých procesov záleží od každej úlohy zvlášť. Niekedy je potrebné najprv integrovať a potom redukovať či transformovať a pod.

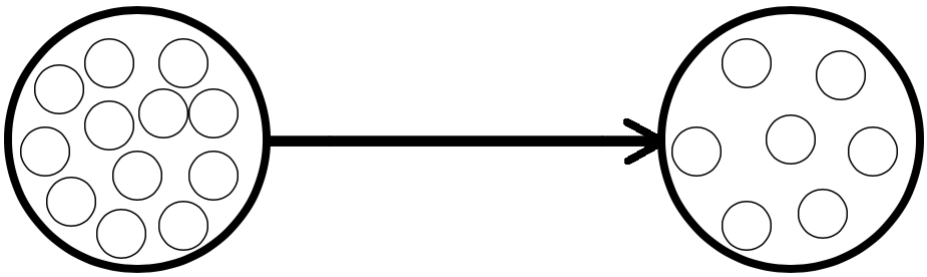
3.1 Čistenie dát



Obrázek 3.2: Čistenie dát

Ako už bolo spomenuté, v databázach sa mnoho krát objavujú neúplné, zašumené alebo nekonzistentné dáta. Práca s takouto databázou môže byť neefektívna a výsledky zase nedôveryhodné. Práve tento problém rieši čistenie dát, ktoré je graficky znázornené na obrázku 3.2. Problém neúplnosti sa najčastejšie rieši odstránením záznamov s chýbajúcimi hodnotami alebo dosadením určitej hodnoty tam, kde hodnota chýba. Existuje však ešte niekoľko iných metód, ktoré vrámci tejto práce nebudú spomenuté. Šum dát sa najčastejšie riši plnením (*binding*). Je to proces takzvaného lokálneho vyhladzovania, počas ktorého sa zohľadňujú hodnoty v blízkom okolí. To napomáha vyhladeniu rôznych odchýlok a drobných chýb, ktoré v databázach vznikajú[35]. Medzi ďalšie metódy určené na odstráňovanie šumu patrí regresia alebo zhlukovanie. Proces čistenia dát je veľmi dôležitý a výrazne ovplyvňuje kvalitu a dôveryhodnosť výsledkov dolovacích úloh.

3.2 Redukcia dát

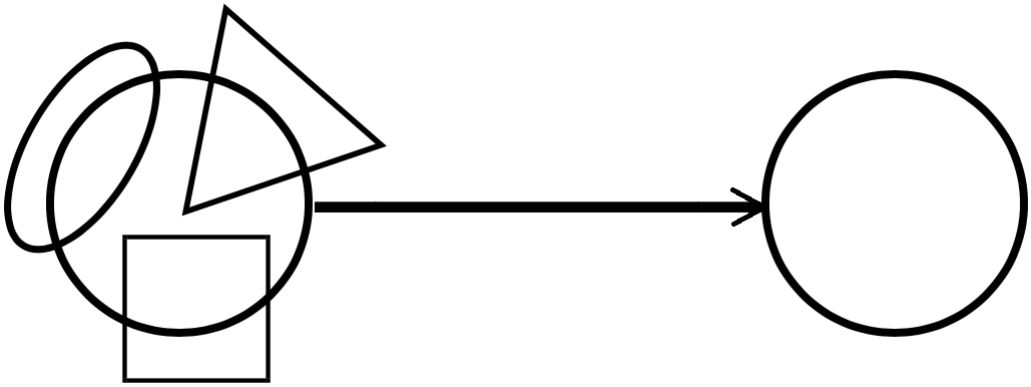


Obrázek 3.3: Redukcia dát

Ďalšou úlohou predspracovania, ktorá vedie k lepším výsledkom dolovania dát je ich redukcia, ktorá je graficky znázornená na obrázku 3.3. Vrámci procesu redukcie dát dochádza k zníženiu ich objemu a to znížením počtu dátových objemov/záznamov (vzorkovanie) alebo znížením počtu atribútov výberom podmnožiny atribútov alebo extrakciou (redukcia dimenzionality). Vďaka tomu sa neskôr urýchľujú akékoľvek výpočty realizované nad takouto sadou dát. Zdrojové dátové sady môžu obsahovať stovky atribútov, z ktorých môžu byť mnohé redundantné alebo nerelevantné pre danú dolovaciu úlohu[35]. Takéto atribúty

je vhodné pred procesom dolovania dát odstrániť. Na čo je však potrebné dávať si pozor, je zachovanie integrity a charakteru pôvodnej množiny[35]. Je veľmi dôležité, aby aj po znížení objemu zdrojových dát boli dosiahnuté takmer rovnaké výsledky ako pred ich redukciou. Jedným z hlavných cieľov je urýchliť výpočet, čo môže byť v mnohých prípadoch dolovacích úloh kritické.

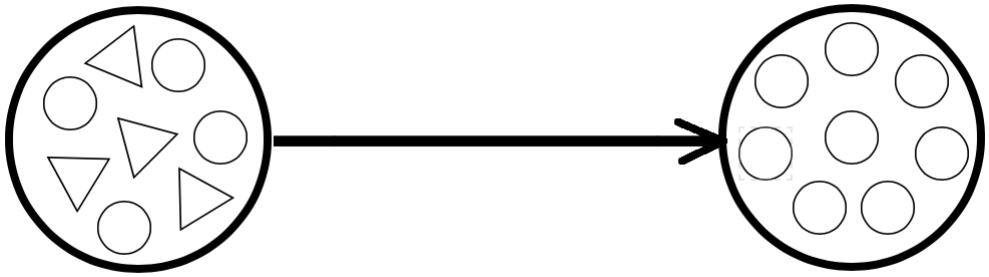
3.3 Integrácia dát



Obrázek 3.4: Integrácia dát

Dolovanie dát si mnoho krát vyžaduje zlučovanie dát z rôznych dátových zdrojov, pričom zvyčajne dochádza ku nadobudnutiu redundancie a nekonzistentnosti dát. Proces, ktorý sa vrámci predspracovania dát snaží tento problém minimalizovať sa nazýva *integrácia dát*, viď obrázok 3.4. Hlavnou úlohou integrácie je teda z mnohých nekoherentných zdrojov vytvoriť jeden koherentný. V tomto prípade sa redundanciou rozumie okrem výskytu rovnakých atribútov aj výskyt atribútov (v rôznych zdrojoch dát), ktoré majú približne rovnaký význam alebo takých, ktorých hodnotu je možné odvodiť od hodnôt iných atribútov[35]. Spolu s integráciou veľmi súvisí aj transformácia dát, o ktorej bude zmienka v sekcii 3.4.

3.4 Transformácia dát



Obrázek 3.5: Transformácia dát

Posledným procesom súvisiacim s predspracovaním dát je ich *transformácia*, ktorej proces je znázornený na obrázku 3.5. Hlavnou úlohou transformácie dát je previesť zdrojové dáta do takého tvaru, ktorý najviac vyhovuje riešeniu danej dolovacej úlohy. To zabezpečí efektívnejší dolovací proces a jednoduchšie pochopenie nájdených vzorov. Medzi najznámejšie transformácie patrí normalizácia a agregácia dát. Normalizácia je proces, ktorej cieľom je namapovať numerické hodnoty na konkrétny interval, ktorým je zvyčajne <-1.0, 1.0> alebo <0.0, 1.0>[35]. Agregácia dát je na druhej strane proces, kedy dochádza ku zoskupeniu niekoľkých dát do jednej entity, čím sa zároveň redukuje objem množiny dát.

Kapitola 4

Dolovanie dát

Už v predchádzajúcich kapitolách bol pojem *dolovanie dát* niekoľko krát spomenutý. V tejto kapitole bude bližšie vysvetlené, čo tento pojem znamená, ako vznikol a aké základné metódy dolovania dát existujú.

4.1 Pojem dolovanie dát

Proces získavania znalostí z databázy sa dnes bežne označuje ako dolovanie dát alebo *data mining*. Okolo tohto pojmu vzniká neustále veľa diskusií z dvoch dôvodov. Jedným z nich je fakt, že dolovanie dát je len jednou časťou procesu získavania znalostí z dát a preto môže byť tento pojem občas zavádzajúci[21]. Druhým dôvodom je samotný význam slov *data mining*. Dolovanie alebo *mining* je slovo, ktoré vyjadruje proces získavania rôznych nerastných surovín z bane. Podstata je tá, že žiadanej nerastej suroviny je v bani málo a treba ju selektovať z obrovského množstva iných nerastov. Tento význam bol prenesený do oblasti informatiky a získavania znalostí z dát a dolovaním sa začalo označovať získavanie informácií z obrovských objemov dát. Tu ale vzniká otázka, prečo sa potom celý tento proces neoznačuje ako *information minig*, ktorý presne vystihuje podstatu tohto procesu, nakoľko sa nedolujú dáta, ale dolujú sa informácie z obrovského množstva dát. V konečnom dôsledku sa aj tak zaužíval pojem *data mining*, nakoľko je to jednoduchšie a kratšie ako *information mining*.

4.2 Metódy dolovania dát

Keďže existuje mnoho rôznych odvetví a druhov dát, existuje aj mnoho rôznych typov dolovacích úloh. Cieľom niektorých úloh môže byť dáta klasifikovať do určitého počtu tried alebo predikovať nejakú novú hodnotu. Inokedy môže byť cieľom nájsť zaujímavú asociáciu medzi dostupnými dátami alebo dáta určitým spôsobom zhlukovať. O týchto druhoch problémov a metódach, ktoré ich riešia, bude zmienka v ďalšej časti tejto kapitoly.

4.2.1 Dolovanie asociačných pravidiel a frekventovaných množín

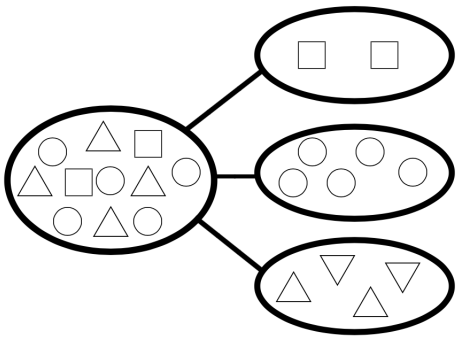
Prvou z metód dolovania dát, o ktorej bude zmienka, je dolovanie asociačných pravidiel a frekventovaných množín. Získavanie asociačných pravidiel spočíva vo výhľadávaní rôznych zaujímavých asociácií a korelácií nad veľkým objemom dát[35]. Tieto techniky sú v dnešnej dobe veľmi populárne pre rôzne internetové obchody. Pomocou týchto metód je možné

analyzovať nákupy a predaný tovar. Z týchto dát je potom možné získať informácie o tom, o aký tovar sa zákazníci zaujímajú a aké položky ich môžu zaujímať a na základe toho im môžu predajcovia ponúkať tovar.

4.2.2 Klasifikácia a predikcia

Druhá technika, o ktorej bude v tejto práci zmienka a na ktorú bude v kapitole 5 upriamená väčšia pozornosť, je klasifikácia. Klasifikácia je proces, ktorého hlavnou úlohou je radiť objekty do určitých prislúchajúcich tried na základe ich vlastností. Týchto tried je vždy konečný počet[35]. Príkladom klasifikácie v reálnom živote je separovanie takzvanej nežiadúcej emailovej pošty. Cieľom tohto procesu je rozhodnúť, či prichádzajúci email je alebo nie je žiadúci a na základe toho ho zaradiť do priečinku prijatej (*inbox*) alebo nevyžiadanej (*spam*) pošty. Metód, akými je možné tento výsledok docieliť, je niekoľko a jednou z nich je napríklad spracovanie textu obsiahnutého v danom emaile do jedného vektora, ktorý je vstupom pre klasifikátor. Existuje približne 57 slov ktoré sú často obsiahnuté v nežiadúcich emailoch a práve ich výskyt a početnosť sa vo vstupnom vektore prehľadáva. Pokiaľ táto početnosť presiahne predom stanovený prah, email sa klasifikuje ako *spam* a táto informácia je zároveň výsledkom klasifikátora[5]. Najprv je však nutné, aby sa klasifikátor naučil správne túto poštu klasifikovať na vzorkoch, o ktorých je známe, do akej triedy patria. Potom je takýto klasifikátor schopný predikovať triedu každému ďalšiemu prichádzajúcemu emailu.

Predikcia je naopak proces, ktorého cieľom je, na základe vlastností daného objektu, predpovedať mu istú hodnotu[35]. Typickým príkladom je predikcia výsledkov rôznych športových podujatí, kedy je možné na základe doterajšej úspešnosti a rôznych štatistických údajov predikovať, kto je najpravdepodobnejším kandidátom na víťaza podujatia.

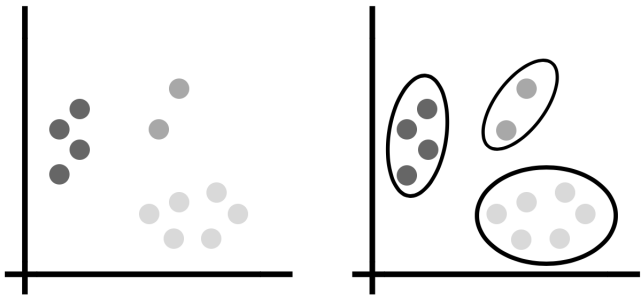


Obrázek 4.1: Klasifikácia objektov

4.2.3 Zhluková analýza

Technika zhlukovej analýzy (*clustering*) je založená na rozdeľovaní objektov do tried (*clusters*) na základe ich podobnosti. O objektoch v jednotlivých triedach je možné tvrdiť, že sú si podobné a že zároveň si nie sú podobné s objektami v iných triedach[35]. To, na základe čoho sa táto podobnosť určuje, je vždy predom daná. Zvyčajne sa vyberú konkrétne atribúty alebo takzvaná vzialenostná funkcia[6]. Príkladom z reálneho života môže byť sieť reštaurácie s donáškou jedla. Ak by bolo v jednom meste dostupných 5 sídiel tejto reštaurácie a 30 ľudí by si vytvorilo objednávku (za predpokladu, že každý sídli na inej ulici), bolo

by vhodné tieto objednávky správne rozdeliť a každú prideliť niektorej z reštaurácií. Toto rozdelenie je možné vykonať na základe rôznych faktov a to napríklad podľa vzdialenosti od sídla reštauráicie alebo podľa typu objednávky a podobne. V takejto situácií by bolo možné využiť metódu zhlukovej analýzy, ktorá by vo svojom výsledku vytvorila 5 tried, ktorých obsahom by boli objednávky prislúchajúce jednému z piatich sídiel danej reštaurácie.

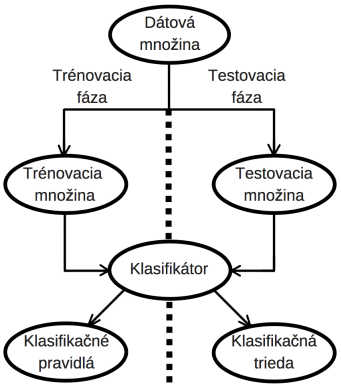


Obrázek 4.2: Výsledné zhluky zhlukovej analýzy

Kapitola 5

Klasifikácia

V tejto kapitole bude upriamená pozornosť na jednu zo spomínaných metód dolovania dát a tou je klasifikácia, nakoľko sa jedná o metódu využitú v prípadovej štúdii v kapitole 7. Ako už bolo v prechádzajúcej kapitole spomenuté, klasifikácia je proces, ktorého úlohou je radiť objekty do predom známych tried na základe ich vlastností. Najzákladnejší princíp vytvárania klasifikačného modelu je graficky zobrazený na obrázku 5.1. Z obrázku je možné vyčítať, že tvorba klasifikačného modelu prebieha v dvoch fázach, pričom sa pracuje s dvoma dátovými množinami. Bližšie informácie budú podrobnejšie opísané v nasledujúcich častiach tejto kapitoly. Poslednou, treťou fázou, je aplikácia vytvoreného klasifkátora na dáta doposiaľ neznámej triedy.



Obrázek 5.1: Základný princíp tvorby klasifikačného modelu

5.1 Fázy klasifikácie

Proces klasifikácie je možné rozdeliť na niekoľko častí. Najčastejšie sa však delí na dve základné fázy a tými sú:

- fáza učenia
- fáza testovania

5.1.1 Fáza učenia

To, čo je hlavnou úlohou prvej fázy klasifikácie je zrejmé už z jej názvu. V tejto fáze je potreba naučiť klasifikátor čo najlepšie klasifikovať. Ako sa dá docieliť, že sa klasifikátor naučí klasifikovať? Pred začiatkom klasifikácie je nutné dátovú množinu s ktorou sa bude pracovať poznať a zároveň vedieť, čo má byť cieľom klasifikácie a na základe akých vlastností je možné tento cieľ dosiahnuť. Cieľom je trieda, kam bude objekt zaradený a zvyčajne sa nazýva cieľový atribút alebo náveštie (*label*). Naopak vlastnosti objektu, ktoré sú vstupom pre klasifikátor, sa označujú ako rysy (*feature*). Takúto množinu dát je potrebné rozdeliť na dve podmnožiny:

- trénovacia množina - táto množina sa využíva v prvej fáze klasifikácie a slúži na tréning klasifikátora.
- testovacia množina - táto množina sa naopak využíva až v druhej fáze, kedy sa testuje úspešnosť klasifikátora.

Veľkosť trénovacej a testovacej množiny nie je pevne stanovená, no zvyčajne predstavuje trénovacia množina približne 75% a testovacia množina 25% z celkového objemu dát.

O dátach v trénovacej množine je nutné dopredu vedieť, do akej triedy patria. Tieto dáta sú vstupom pre klasifikátor a práve vďaka tomu, že je predom známe do akej triedy patria, je klasifikátor schopný zistiť klasifikačné pravidlá, pomocou ktorých sa neskôr riadi a objekty klasifikuje do určitých tried[35]. Klasifikačné pravidlo $h(x)$ je funkcia, ktorá priradí každý element z množiny rysov klasifikačnej triede $y = h(x)$. V prípade binárnej klasifikácie, klasifikačná trieda (y) nadobúda binárne hodnoty 0 a 1.

5.1.2 Fáza testovania

Druhá fáza spočíva v testovaní úspešnosti klasifikátora. Tentokrát sa použije testovacia množina, ktorej dáta sú nezávislé na predchádzajúcich dátach (sú teda pre klasifikátor nové). Je dôležité podotknúť, že aj v tomto prípade je známa klasifikačná trieda. Dáta testovacej množiny sú opäť vstupom pre klasifikátor. *Naučený* klasifikátor pomocou klasifikačných pravidiel, ktoré v predchádzajúcej fáze vytvoril, klasifikuje tieto vstupné dáta. Vďaka tomu, že je predom známe do akej triedy dáta naozaj patria, je možné percentuálne vyhodnotiť úspešnoť klasifikátora a celého procesu klasifikácie[35]. Viac informácií o hodnotení klasifikačných modelov sa nachádza v kapitole 5.3. To, či je klasifikátor dostatočne úspešný a či je ho možné použiť na dátach, o ktorých sa predom nepozná klasifikačná trieda, je otázkou rozhodnutia daného človeka zodpovedného za daný klasifikátor. Neexistuje však presne stanovený prah, ktorý by určoval, či je klasifikátor dostatočne úspešný alebo nie.

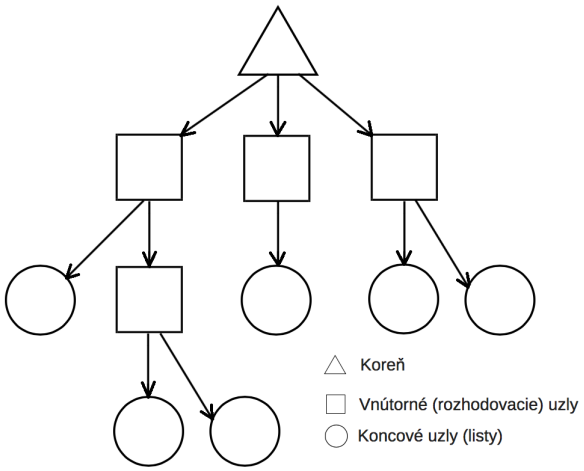
5.2 Metódy klasifikácie

V nasledujúcej časti tejto kapitoly budú spomenuté najčastejšie používané metódy klasifikácie.

5.2.1 Klasifikácia pomocou rozhodovacích stromov

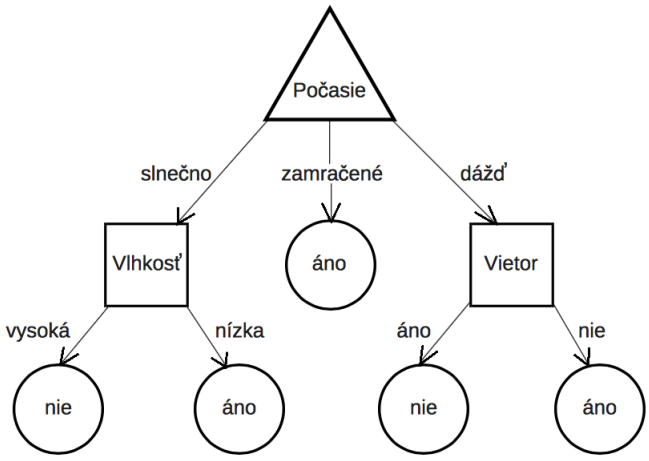
Prvou metódou, o ktorej bude v tejto kapitole zmienka, je klasifikácia pomocou rozhodovacích stromov. Rozhodovací strom je graf stromovej štruktúry, ktorý je zložený z koreňa,

vnútorných uzlov a koncových uzlov (listy)[35]. Jednoduchý graf rozhodovacieho stromu je znázornený na obrázku 5.2.



Obrázek 5.2: Rozhodovací strom

Na tejto grafovej štruktúre je založená celá metóda klasifikácie pomocou rozhodovacích stromov. Koreň stromu je vždy tvorený atribútom s najvyššou rozhodovacou schopnosťou. Atribút s najvyššou rozhodovaciu schopnosťou je ten atribút, na základe ktorého začína celá klasifikácia. Extrémy stromu tvoria koncové uzly alebo listy. Listy v tomto kontexte reprezentujú finálnu klasifikačnú triedu. Medzi koreňom a listami sa nachádzajú rozhodovacie alebo inak nazývané aj *splitting* uzly. Tieto uzly reprezentujú testy hodnôt atribútov a rozhodujú do akej klasifikačnej triedy daná hodnota patrí, prípadne do akého ďalšieho rozhodovacieho uzlu daná hodnota poputuje. Každý takýto rozhodovací strom môže byť pretransformovaný na klasifikačné pravidlá. Príklad konkrétneho rozhodovacieho stormu je znázornený na obrázku 5.3 a znázorňuje rozhodnutie, či hrať alebo nehrať futbalový zápas.



Obrázek 5.3: Rozhodovací strom - príklad

Klasifikačné pravidlá pre tento storm by vyzerali nasledovne:

```
if Počasie == 'slnečno' and Vlhkosť == 'vysoká' then return áno
```

```
if Počasie == 'slnečno' and Vlhkosť == 'nízka' then return nie
if Počasie == 'zamračené' then return áno
if Počasie == 'dážď' and Vietor == 'áno' then return nie
if Počasie == 'dážď' and Vietor == 'nie' then return áno
```

Tento proces však má svoje chyby a jednou z nich je vznik nežiadaných vetví z dôvodu výskytu dátového šumu[35]. Pre skvalitnenie presnosti predpovede rozhodovacieho stromu je potrebné takéto vetvy odstrániť. Existujú dve základné eliminačné metódy:

- prepruning - tieto nežiadúce vetvy sa vôbec negenerujú. Namiesto testu atribútu sa táto časť stromu rovno ukončí listom, ktorý reprezentuje takú klasifikačnú triedu, ktorá je priradená najväčšiemu počtu vzorkov, ktoré odpovedajú tejto časti stormu[35].
- postpruning - nepotrebné vetvy sú odstránené (orezané) až po vytvorení celého rozhodovacieho stromu.

5.2.2 Bayesovská klasifikácia

Bayesovská klasifikácia je metóda založená na štatistike. Jej princíp je však úplne jednoduchý. Zakaždým, keď klasifikátor dostane na vstup nový objekt, ktorý je potrebné zaradiť do určitej triedy, tak sa podľa konkrétnych štatistických metód vypočíta s akou pravdepodobnosťou patrí do jednotlivých tried. Ako výsledok sa vyberie tá trieda, pre ktorú bola vypočítaná najvyšia pravdepodobnosť[35]. Pre výpočet pravdepodobnosti sa využíva dobre známy Bayesov vzorec[2]:

$$p(C_k \mid x) = \frac{p(C_k) \times p(x \mid C_k)}{p(x)}$$

Premenná x je vektor, ktorý reprezentuje hodnoty *feature* atribútov, respektíve rysy. Výraz

$$p(C_k \mid x)$$

potom vyjadruje pravdepodobnosť, s ktorou tieto hodnoty spadajú do jednej z k možných klasifikačných tried.

5.2.3 Klasifikácia pomocou podporných vektorov

Tento druh klasifikácie je tiež známy ako SVM alebo *Support Vector Machine*. Princíp SVM klasifikátora pri binárnej klasifikácii je založený na rozdelení dát z trénovacej množiny v takzvanom bodovom diagrame na dve protiľahlé oblasti, ktoré reprezentujú jednotlivé triedy dát. SVM rozdeľuje priestor nadrovinou (*hyperplane*), ktorá sa tiež označuje ako rozhodovacia hranica, ktorá oddeľuje na bodovom diagrame dve triedy a určuje, ktoré body patria do ktorej triedy. Pre optimálnu nadrovinu platí, že by mala byť umiestnená v čo najväčšom odstupe od krajných bodov, ktoré sa nazývajú podporné vektory (*support vectors*). Rozlišujeme dva druhy SVM - lienárnu a nelineárnu - a to podľa schopnosti rozdeliť priestor[27].

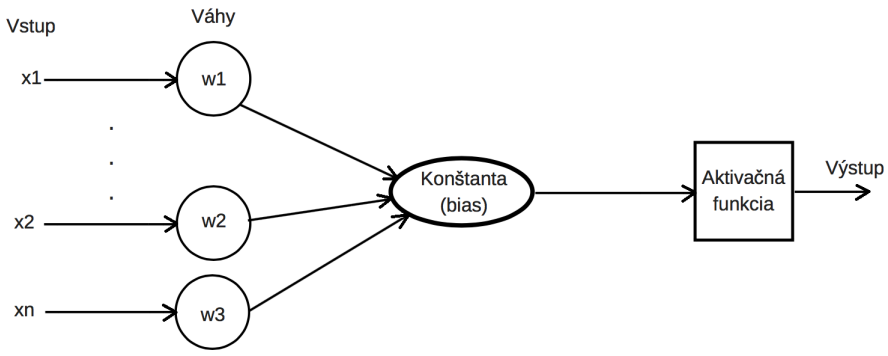
5.2.4 Klasifikácia založená na k-najbližšom susedstve

Tento druh klasifikácie je často označovaný za jeden z najjednoduchších. Pri klasifikácii objektu sa pozerá na susediace objekty, ktorých trieda je známa. Počet objektov, na ktoré sa berie ohľad je daný parametrom k a nazýva sa tiež okolie. Okolie vždy nadobúda kladné hodnoty a typicky sa jedná o malé celé číslo. Klasifikovanému objektu je priradená tá trieda,

ktorá je v okolí k najpočetnejšia. Ak by nastal prípad, že hodnota $k = 1$, tak by bol objekt klasifikovaný do rovnakej triedy, ako najbližšie sa vyskytujúci objekt v jeho okolí[19].

5.2.5 Klasifikácia pomocou neurónových sietí

Posledným modelom, o ktorom bude v tejto kapitole zmienka, sú takzvané neurónové siete. Princíp neurónových sietí bol odvodený od princípu biologických neurónov, ktoré sú základnou stavebnou jednotkou neurovej sústavy. Biologický neurón je zložený z tela (*soma*), vstupov (*dendritov*) a jedného výstupu (*axómu*). Jednotlivé neuróny komunikujú prostredníctvom synapsií, ku ktorej dochádza prepojením dendritov jedného neurónu a axómu druhého neurónu[35]. Umelý neurón (viď obrázok 5.4) má niekoľko vstupov, na ktoré sú zavedené vstupné hodnoty. Tieto vstupy sú zároveň výstupmi iných umelých neurónov. Každý umelý neurón zároveň obsahuje hodnoty, ktoré reprezentujú synapsie, a sú nazývané *váhy*. Ich hodnoty sú nastavené tak, aby bol zo vstupu dosiahnutý požadovaný výstup. Každý neurón vo svojom vnútri dokáže transformovať konkrétnym výpočtom vstup na výstup pomocou konštanty nazývanej bias a aktivačnej funkcie. Tento výstup je potom opätovne vstupom ďalšieho neurónu alebo reprezentuje výstupnú hodnotu, na ktorej bude vykonaná klasifikácia[35].



Obrázek 5.4: Umelý neurón

5.3 Výber a vyhodnotenie modelov

Vytvorením klasifikačného modelu proces klasifikácie nekončí. Po jeho vytvorení nastáva fáza, kedy je potrebné tento model určitým spôsobom ohodnotiť a odhadnúť, ako sa model bude správať ďalej pri jeho aplikácii na dáta s doposiaľ neznámou triedou. V praxi to znamená, že tento klasifikátor bude ďalej používaný zakaždým na nových dátach a mal by byť spoľahlivý a presný[21].

V tejto sekcii budú opísané niektoré bežne používané metriky, ktoré odhadujú presnosť a spoľahlivosť klasifikačných modelov.

5.3.1 Pojmy súvisiace s hodnotením klasifikačných modelov

Učenie klasifikátora na trénovacej množine a zároveň určovanie jeho úspešnosti na tejto množine môže byť zavádzajúce. Úspešnosť klasifikátora môže byť v tomto prípade dosť vysoká. To však nemusí odpovedať realite, pretože klasifikátor je v tomto bode na tieto dáta

špecializovaný[21]. Vždy je dôležité úspešnosť klasifikátora hodnotiť na testovacej množine, pri ktorej sa klasifikátor stretne so zatiaľ neznámou množinou dát.

V tejto práci bude upriamená pozorosť na binárnu klasifikáciu. Binárna klasifikácia je druh klasifikácie, v ktorom sa vstupné dáta klasifikujú do dvoch tried, pričom cieľový atribút môže nadobúdať hodnoty z množiny {0,1}, respektíve {áno (*true*), nie (*false*)}. Trieda s hodnotou 1, respektíve s hodnotou áno (*true*), sa nazýva pozitivita (*positivity*). Naopak trieda s hodnotou 0, respektíve s hodnotou nie (*false*), sa nazýva negativita (*negativity*). Pre lepšie pochopenie, čo tieto dva pojmy znamenajú, je možné uviesť príklad klasifikačnej úlohy znázornenej na obrázku 5.3. V tomto prípade je pozitivita n-tica dát, ktorej výsledkom klasifikácie je 1 (áno, zápas sa odohrá) a naopak negativitou je n-tica dát, ktorej výsledkom klasifikácie je 0 (nie, zápas sa neodohrá). Ešte pred rozprávaním o konkrétnych metrikách je potrebné vysvetliť štyri dôležité súvisiace pojmy:

- Skutočne pozitívne (*true positive*, TP) - jedná sa počet n-tíc pri ktorých došlo k pozitívnej zhode. V konkrétnej úlohe z obrázku 5.3 to znamená, že klasifikátor správne určil že sa zápas odohrá.
- Falošne pozitívne (*false positive*, FP) - jedná sa o počet n-tíc, ktorých výsledky boli falošne pozitívne. Klasifikátor teda určil, že sa zápas odohrá aj keď reálne sa zápas neodohrá.
- Skutočne negatívne (*true negative*, TN) - jedná sa počet n-tíc pri ktorých došlo k negatívnej zhode. To znamená, že klasifikátor správne určil, že sa zápas neodohrá.
- Falošne negatívne (*false negative*, FN) - jedná sa o počet n-tíc, ktorých výsledky boli falošne negatívne. To znamená, že klasifikátor určil, že sa zápas neodohrá, no reálne sa zápas odohrá.

5.3.2 Matica zmien

Matica zmien (*confusion matrix*), znázorená tabuľkou 5.1, sumarizuje pojmy uvedené v sekcii 5.3.1. Táto matica slúži predovšetkým na vizualizáciu toho, ako dobre dokáže klasifikátor rozlišovať pozitivitu a negativitu. TP a TN sú hodnoty, ktoré určujú kedy bol klasifikátor úspešný a klasifikoval správne. Naopak hodnoty FP a FN určujú, kedy klasifikátor nebol úspešný a neklasifikoval správne.

		predicted	
		0	1
actual	0	TN	FN
	1	FP	TP

Tabulka 5.1: Matica zmien

Súčet skutočne a falošne pozitívnych sa označuje aj písmenom T. Naopak súčet skutočne a falošne negatívnych sa označuje písmenom N.

5.3.3 Metriky

Po objasnení týchto dôležitých pojmov je možné rozprávať o jednotlivých metrikách. Medzi najčastejšie používané metriky určené na hodnotenie klasifikačných modelov patrí celková správnosť modelu, senzitivita, špecificita[18], chybovosť, presnosť a F1 skóre[21].

Celková správnosť modelu

Celková správnosť modelu (*accuracy*) je definovaná ako podiel správne klasifikovaných subjektov ku všetkým subjektom, čiže:

$$\frac{TP + TN}{P + N}$$

Senzitivita

Senzitivita (*sensitivity, recall*) je daná pomerom skutočne pozitívnych prípadov ku všetkým pozitívnym prípadom, čiže:

$$\frac{TP}{P}$$

Špecificita

Špecificita (*specificity*) je daná podielom skutočne negatívnych prípadov ku všetkým negatívnym prípadom, čiže:

$$\frac{TN}{N}$$

Chybovosť

Chybovosť (*error rate*) je daná podielom súčtu falošne pozitívnych a falošne negatívnych prípadov ku súčtu všetkých prípadov, čiže:

$$\frac{FP + FN}{P + N}$$

Presnosť

Presnosť (*precision*) je daná podielom skutočne pozitívnych prípadov ku súčtu všetkých skutočne pozitívnych a falošne pozitívnych prípadov, čiže:

$$\frac{TP}{TP + FP}$$

F1 skóre

F1 skóre (*f1 score*) je metrika, ktorej vzorec je tvorený kombináciou dvoch predchádzajúcich metrík, a to presnosti a senzitivity. Vzorec určujúci F1 skóre vyzerá nasledovne:

$$\frac{2 \times precision \times senzitivity}{precision \times senzitivity}$$

Kapitola 6

Podpora jazyka Python pre dolovanie dát

Python je pomerne jednoduchý a všestranne využiteľný programovací jazyk. Svoje uplatnenie si našiel napríklad vo vývoji backendových webových aplikácií, umelej inteligencii, vedeckých výpočtoch alebo aj v dátovej analýze. Nakoľko je s týmto jazykom možné vytvárať od hier cez aplikácie, až po najrôznejšie nástroje, získal si v poslednej dobe vysokú obľúbenosť a popularitu[29]. Takzvaný *Python Package Index* alebo skrátene *PyPI*[15] hostuje tisíce modulov tretích strán, ktoré ešte o niečo viac rozširujú možnosti využitia tohto jazyka. V tejto práci bude pozornosť upriamená na takzvané *Scientific & Numeric* moduly, medzi ktoré patrí napríklad `Pandas`, `NumPy`, `SciPy` a v neposlednom rade bude spomenutý voľne dostupný softvér `scikit-learn`, založený práve na `NumPy` a `SciPy` knižniciach.

6.1 `Pandas`

`Pandas` je voľne dostupná knižnica jazyka Python, ktorá poskytuje vysoko výkonné a jednoducho použiteľné nástroje a dátové štruktúry. Slúži na analýzu dát, ktoré je možné reprezentovať 2D tabuľkou. Príkladom takýchto dát sú SQL tabuľky, CSV súbory alebo rôzne tabuľkové precesory[34]. Základné dátové typy, ktoré táto knižnica ponúka sú:

- `DataFrame` - dátový typ identický dobre známej tabuľke, ktorá má riadky (záznamy) a stĺpce (atribúty).
- `Series` - dátový typ reprezentujúci stĺpec tabuľky, ktorý obsahuje sériu hodnôt (podobne ako zoznam). Navyše má svoje meno, dátový typ a index, ktorý jednotlivé hodnoty pomenúva[34].

V jazyku Python je možné k hodnotám dátového objektu `DataFrame` rôzne pristupovať. K stĺpcom tabuľky sa dá prisupovať atribútom `columns` a k hodnotám tabuľky je možné pristupovať atribútom `values`. Ďalej je možné zistiť rozmery tabuľky (počet stĺpcov a riadkov) prostredníctvom atribútu `shape`. Pre príklad je uvedená tabuľka 6.1.

Animal	Legs	Country
Snake	0	South America
Tiger	4	India
Spider	8	Australia

Tabulka 6.1: Tabuľka reprezentujúca dátový objekt X typu DataFrame

Nech je táto tabuľka dátového typu DataFrame a nazýva sa X, potom je možné jej atribúty a hodnoty vyčítať nasledovne:

```
print X.columns
>>> Index(['Animal', 'Legs', 'Country'])

print X.values
>>> array(['Snake', 0, 'South America'],['Tiger', '4', 'India'],
['Spider', '8', 'Australia'])

print X.shape
>>> (3,3)
```

Táto knižnica zároveň ponúka metódu, vďaka ktorej je možné konvertovať dátovu sadu v CSV formáte na dátový typ `DataFrame`. Jej použitie môže vyzerať nasledovne:

```
import pandas as pd
dataframe_object = pd.read_csv('original_data.csv')
```

6.2 NumPy

Numerický Python alebo skrátene `NumPy` je základnou knižnicou jazyka Python, ktorá je zameraná najmä na rôzne vedecké a matematické výpočty. táto knižnica poskytuje multidimenzionálne objekty typu pole, nad ktorými je možné vykonávať matematické, logické a ďalšie operácie veľmi rýchlo a efektívne[22].

To, čo robí túto knižnicu výnimočnou, je jednoznačne objekt `ndarray`. Tento objekt zapúzdruje n-dimenzionálne polia homogénnych dátových typov a rôzne operácie, ktoré je nad nimi možné vykonávať. Objekt `ndarray` sa od iných Pythonovských sekvencií - ako je `list` alebo `tuple` - líši napríklad tým, že:

- Veľkosť `ndarray` je daná pevne už pri inicializácii, zatiaľ čo klasické `listy` sa môžu dynamicky meniť aj po ich inicializácii.
- Narozdiel od bežných sekvencií, všetky prvky `ndarray` objektu musia byť rovnakého dátového typu.[22]

Numpy poskytuje radu metód, ktoré je možné aplikovať na objekty dátového typu `ndarray`. Medzi ne patrí napríklad metóda `arange()`, ktorá vráti rovnomerne rozmiestnené hodnoty na danom intervale. Táto metóda uľahčuje iterácie v rôznych cykloch a to aj pri práci s dátovým objektom `DataFrame`. Priklad použitia:

```
import numpy as np

number_of_attributes = X.shape[1] # number_of_attributes = 3
```

```
for attribute_index in np.arange(number_of_attributes):
    print X.columns[i]

Output:
name_of_attr_1
name_of_attr_2
name_of_attr_3
```

Ďalšou užitočnou metódou je metóda `mean()`, ktorá je schopná vypočítať aritmetický priemer (numerických) prvkov poľa typu `ndarray`:

```
list_x = [1,2,3,4,5]
np_array = np.array(list_x) # converts Python list into ndarray
result = np.mean(nd_array)
print result

>>> 2.5
```

6.3 SciPy

`SciPy` vzniklo skrátením slov Scientific Python a tento názov zodpovedá samotnej funkcii tejto voľne dostupnej knižnice. `SciPy` obsahuje moduly určené na optimalizáciu, lineárnu algebru, interpoláciu, spracovanie obrazu a omnoho viac. V tomto bode môže vznikať otázka, aký je rozdiel medzi `NumPy` a `SciPy`. Knižnica `NumPy` a jej objekt `ndarray` tvoria základ, na ktorom je postavená knižnica `SciPy` a jej ďalšia funkcionalita. Je možné tvrdiť, že `SciPy` je jej nadstavbou a spolu s využitím ďalších modulov ako je `Pandas` alebo `Matplotlib` tvorí `SciPy` skvelú podporu jazyka Python pre vedecké a výpočetné úlohy[30]. V tejto práci a jej praktickej časti však nenašiel tento modul svoje využitie.

6.4 Scikit-learn

Ako posledný nástroj, ale zároveň najdôležitejší, o ktorom bude v tejto práci zmienka, je práve `scikit-learn`. Je to nástroj postavený na základoch knižnice `SciPy` a špeciálne slúži na dátovú analýzu a dolovanie dát[11]. Tento nástroj rozširuje jazyk Python o radu metód a tried, ktoré umožňujú strojové učenie, prácu s dátovými množinami, zhlukovanie a ďalšie užitočné úkony dátovej analýzy.

Tento prostriedok bol základným stavebným kameňom demonštrácie dolovania dát v jazyku Python v kapitole 7. Tento softvér ponúka okrem iných aj plnú podporu pre klasifikačné dolovacie úlohy.

V prvom rade sú to metódy určené na predspracovanie dát. `Scikit-learn` ponúka radu metód určených na čistenie, redukciu, transformáciu a integráciu dát[14].

Ďalej tento softvér ponúka triedy a ich metódy určené na križovú validáciu a sú to napríklad triedy `LeaveOneOut`, `LeavePOut`, `KFold` alebo `RepeatedKFold` ktoré sú súčasťou balíka `models_selection`[7]. Ako príklad je možné uviesť základnú funkcionalitu križového validátora `KFold`.

```
import numpy as np
from sklearn.model_selection import KFold:
```

```
data = ["a", "b", "c", "d"]
cross_validator = KFold(n_splits=2)
for train, test in cross_validator.split(X):
    print({0}{1}".format(train, test))

>>> [2 3] [0 1]
>>> [0 1] [2 3]
```

Ďalej sa tento softvér môže pýšiť širokou škálou dostupných klasifikačných modelov. Každý z dostupných modelov je reprezentovaný ako trieda, ktorá má v každom prípade, okrem iných metód, implementovanú metódu `fit()`, ktorá trénuje klasifikátor a metódu `predict()`, ktorá aplikuje vytrénovaný klasifikátor na testovaciu množinu dát. Medzi základné klasifikačné modely, ktoré `scikit-learn` ponúka, patrí `DecisionTreeClassifier`, ktorý je založený na metóde rozhodovacích stromov a je súčasťou balíka `tree`. Ďalej je to metóda `GaussianNB()`, založená na Bayesovskej klasifikáci, ktorá je súčasťou balíka `naive_bayes`, alebo metóda `NearestCentroid`, ktorá je súčasťou balíka `neighbors` a je založená na metóde k-najbližšieho susedstva[16]. Ako príklad je možné uviesť inicializáciu modelu `DecisionTreeClassifier` a použitie jeho základných metód:

```
from sklearn import tree

X = [[0, 0], [1, 1]]
y = [0, 1]

classifier = DecisionTreeClassifier()
classifier = classifier.fit(X, y)

# after being fitted, the classifier can
# then be used to predict the class of samples
classifier.predict([[2., 2.]])

>>> array([1])
```

Pre dokončenie klasifikácie nesmú tomuto softvéru chýbať metódy určené na vyhodnotenie modelov. Všetky známe metriky sú implementované v balíku `metrics`. Medzi najčastejšie využívané metódy patrí `accuracy_score`, `f1_score`, `recall_score` alebo `precision_score`[12]. Ako príklad je možné uviesť použitie metriky `accuracy_score`:

```
import numpy as np
from sklearn.metrics import accuracy_score
values_predicted = [0, 0, 1, 0, 1]
values_true = [0, 1, 1, 0, 1]
accuracy_score(values_true, values_predicted)

>>> 0.8
```

Kapitola 7

Dolovanie dát v jazyku Python

Cieľom výskumu je vytvoriť skript v jazyku Python, ktorý bude slúžiť ako ukážka získavania znalostí z dát v jazyku Python. Analytická úloha spočíva v klasifikácií ľudí do dvoch kategórií - ľudia trpiaci Parkinsonovou chorobou a zdraví ľudia. Pre túto úlohu bola k dispozícií dátová sada, ktorá obsahuje dáta biomedicínskych hlasových meraní zdravých ľudí a ľudí trpiacich Parkinsonovou chorobou, ktorá je voľne dostupná na webovej stránke UCI Machine Learning Repository[33]. Táto analytická úloha bola inšpirovaná výskumom M. Vadovského a J. Paraliča s názvom *Použitie spracovaných záznamov reči pacientov pre určenie štádia Parkinsonovej choroby*[32].

7.1 Parkinsonova choroba

Pred akýmkoľvek dolovaním dát je potrebné oboznámiť sa so skúmanou oblasťou a získať čo najviac informácií, ktoré by mohli ovplyvniť nasledujúci výskum. V tomto prípade sa jedná o Parkinsonovu chorobu a preto je potrebné zistiť, čo vlastne Parkinsonova choroba je a ako sa prejavuje.

7.1.1 Čo je Parkinsonova choroba

Parkinsonova choroba je neurodegeneratívne ochorenie, ktoré ovplyvňuje prevažne neuróny produkujúce látku dopamín v špecifickej časti mozgu, nazývanej čierna substancia alebo *substantia nigra*[26]. Hlavnou úlohou tejto látky je prenos signálov v mozgu, zodpovedných najmä za pohyb a taktiež emócie. Ako sa choroba postupne vyvíja, bunky produkujúce dopamín odumierajú až do bodu, kedy sa dopamín v mozgu neprodukuje takmer vôbec[24]. V takom prípade pacient okrem iného úplne stratí schopnosť pohybu a samostatnosti.

7.1.2 Príznaky Parkinsonovej choroby

Jedná sa o veľmi špecifickú chorobu, ktorej príznaky sa vyvíjajú postupne v priebehu rokov. Vývoj Parkinsonovej choroby a jej symptómov závisí od pacienta k pacientovi. Je vedecky dokázané, že každý človek trpiaci touto chorobu ju pociťuje inak. Existuje však niekoľko symptómov, ktoré majú všetci títo pacienti spoločné a na základne ktorých je možné túto chorobu diagnostikovať[23]. Tieto príznaky je možné zaradiť do dvoch hlavných kategórií - motorické a kognitívne. Medzi motorické príznaky môžeme zaradiť tremor, stratu rovnováhy, celkové spomalenie pohybového ústrojenstva a stratu flexibility[24]. Medzi kognitívne príznaky patria napríklad problémy so sústredením, problémy s pamäťou, nechuť do jedla,

depresia, strata hmotnosti a v neposlednom rade aj postupná strata schopnosti prirodzene rozprávať[25].

7.2 Pochopenie a príprava dát

Prvým krokom k pochopeniu dát je oboznámenie sa s Parkinsonovou chorobou (viď sekciu 7.1). Ako už bolo spomenuté, Parkinsonova choroba sa môže prejaviť okrem iného aj v samotnej reči pacienta. Človek trpiaci Parkinsonovou chorobou má hlas o dosť tichší a jemnejší. V horších štádiách choroby má okolie problém tohto človeka vôbec počuť a komunikovať s ním. Nakoľko títo ľudia mávajú problémy s pamäťou, zabúdajú slová a tým pádom je reč omnoho pomalšia a zároveň monotónnejšia ako u zdravého človeka. Tieto príklady sú len jedni z mnohých faktorov postihnutia prirodzenej rečovej schopnosti u pacientov s Parkinsonovou chorobou[28]. Nakoľko sú reč a hlas merateľné a dajú sa analyzovať, boli pre túto úlohu zvolené práve rečové dáta, ktoré sú voľne dostpuné na UCI Machine Learning Repository[33]. Tieto dáta sú prispôsobené tomuto typu výskumu - získavaniu znalostí z dát - a dajú sa aplikovať na metódu klasifikácie. Klasifikátor je schopný sa na tejto dátovej sade naučiť určovať, či pacient trpí alebo netrpí Parkinsonovou chorobou.

7.2.1 Dátova sada

Použité dáta obsahujú radu biomedicínskych hlasových meraní 31 ľudí, z ktorých 23 trpí Parkinsonovou chorobou a zvyšní sú zdraví ľudia, ktorí netrpia žiadnou rečovou závadou. Konkrétne sa jedná spolu o 195 rečových záznamov, čo je približne 5-6 záznamov na jedného pacienta. Každý atribút tejto dátovej sady prestavuje špecifické hlasové meranie, ktoré bolo vykonané na danom subjekte. Sada taktiež obsahuje atribút `name` ktorý identifikuje subjekt a atribút `status`, ktorý určuje, či je daný subjekt zdravý (0) alebo chorý (1). Dáta sú k dispozícií v ASCII CSV formáte. Pre získavanie znalostí z dát tento formát jazyku Python plne vyhovuje a nie je potreba ho nijak transformovať.

Atribúty dátovej sady

- `name` - meno subjektu a číslo záznamu v ASCII formáte
- `MDVP:Fo[Hz]` - priemerná hlasová frekvencia
- `MDVP:Fhi[Hz]` - maximálna hlasová frekvencia
- `MDVP:Flo[Hz]` - minimálna hlasová frekvencia
- `NHR, HNR` - dve miery pomeru šumu a tónových komponent v hlase
- `RPDE, D2` - dve nelineárne dynamické merania komplexnosti hlasu
- `DFA` - signál fraktálového exponentu škálovania
- `spread1, spread2, PPE` - tri nelineárne merania základnej frekvenčnej odchýlky
- `status` - zdravotný stav pacienta
- `MDVP:Jitter[%], MDVP:Jitter[Abs], MDVP:RAP, MDVP:PPQ, Jitter:DDP` - miery variability v hlasovej frekvencii

- `MDVP:Shimmer, MDVP:Shimmer[dB], Shimmer:APQ3` - miery variability v amplitúde
- `Shimmer:APQ5, MDVP:APQ, Shimmer:DDA` - miery variability v amplitúde

7.2.2 Príprava dát

Príprava dát spočíva v predspracovaní dát, o ktorom bola zmienka v kapitole 3. Je to preto, aby ich nasledujúce dolovanie bolo čo najefektívnejšie a najúspešnejšie. Ako už bolo spomenuté, hlavnou úlohou predspracovania dát je odstránenie chýbajúcich hodnôt, šumu a nekonzistencie. Dostupná množina dát však neobsahuje žiadne chýbajúce hodnoty ani zašumené dáta a rovnako sa v dátovej sade nevyskytuje žiadna nekonzistencia. Dáta sú teda pripravené na dolovanie. Čo je však možné vykonať, je redukcia dát, konkrétne redukcia dimenzionality (viď sekcia 3.2). Počas výskumu sa pracovalo s celou množinou dostupných dát a neskôr s podmnožinou, v ktorej bola odstránená kolinearita výpočtom takzvaného VIF faktoru. O tomto procese bude ďalej zmienka v sekcii 7.3.2.

7.3 Implementácia riešenia úlohy v jazyku Python

Hlavným cieľom tejto práce je demonštrovať využitie programovacieho jazyka Python na účely získavania znalostí z dát. V nasledujúcej časti bude podrobne opísaný postup riešenia a všetkých využitých prostriedkov, ktoré jazyk Python ponúka. Niektoré dostupné knižnice a softvéry, ktoré umožňujú prácu s dátami v jazyku Python boli spomenuté v kapitole 6. Počas výskumu boli najviac využité triedy a ich metódy, ktoré ponúka softvér `scikit-learn` a dátové typy a metódy modulov `Pandas` a `NumPy`. Pre upresnenie je potrebné spomenúť, že pre implementáciu bola použitá verzia 3.6 jazyka Python.

7.3.1 Načítanie dát

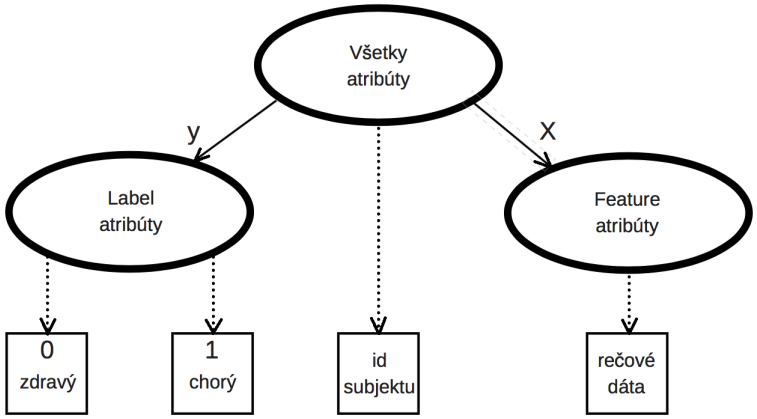
Použitá dátová sada je v CSV (*Comma Separated Values*) formáte. Jedná sa o jednoduchý a štandardizovaný textový formát určený na reprezentáciu tabuľkových dát, kde sú jednotlivé riadky tabuľky oddelené čiarkou[31]. Pre načítanie súboru tohto typu bola použitá metóda `read_csv()` z knižnice `Pandas`, ktorá konvertuje vstupné dáta do dátového typu `DataFrame`. Vďaka tomu je možné pracovať s dátami ako s bežnou SQL tabuľkou, kde môžeme pristupovať k atribútom a ich hodnotám jednoducho indexovaním.

7.3.2 Odstránenie kolinearity

Z hľadiska experimentovania bola snaha o vylepšienie výsledkov tým, že sa zredukoval počet atribútov odstránením kolinearity. Kolinearita vzniká vtedy, ak je korelácia medzi hodnotami dvoch atribútov tak vysoká, že je možné jeden z nich vylúčiť, nakoľko je možné tvrdiť, že na základe jedného je možné predikovať druhý a naopak. Tento problém je možné riešiť pomocou metódy `variance_inflation_factor()`, ktorá je súčasťou modulu `StatsModel`. Táto metóda umožňuje vypočítať inflačný faktor rozptylu[17] pre každý atribút. Ak sa hodnota VIF faktoru pre daný atribút nachádza v rozmezí 5-10, existuje vysoká pravdepodobnosť výskytu kolinearity a atribút je vhodné odstrániť. V dátach s ktorými sa pracovalo sa našlo presne osem atribútov s hodnotou inflačného faktoru rozptylu vyššou ako päť. Tieto atribúty boli z dátovej sady odstránené a jedná sa konkrétne o atribúty `Shimmer:APQ3`, `Jitter:DD`, `MDVP:Shimmer`, `MDVP:Jitter[%]`, `DFA`, `MDVP:Shimmer[dB]`, `Shimmer:APQ5` a `MDVP:RAP`. Pôvodný počet atribútov 21 bol zredukovaný na 13.

7.3.3 Určenie vlastností a cieľových atribútov

Na začiatku procesu získavania znalostí z dát je potrebné položiť si otázku, akú informáciu je potrebné zistiť a na základe čoho je možné túto informáciu dostať. Ako už bolo spomenuté v kapitole 5, atribúty dátovej množiny sa rozdeľujú na cieľové (*label*) atribúty a na rysy (*feature*). Pri implementácii sa zvyknú *label* atribúty označovať písmenom `y` a naopak *feature* atribúty písmenom `X`. Cieľom tejto analytickej úlohy bolo určiť, či je daný subjekt zdravý alebo chorý. Nech hodnota 1, ktorá reprezentuje chorého človeka a hodnota 0, ktorá reprezentuje zdravého človeka, sú hodnoty klasifikačnej triedy, ku ktorej je cieľom sa dopracovať. Rozhodnutie, či je testovaný subjekt zdravý alebo chorý, je možné predpovedať na základe získaných rečových dát. Preto boli ako *label* dáta zvolené hodnoty stĺpca `status` (0,1) a ako *feature* dáta všetky stĺpce obsahujúce spracovanú reč daného subjektu. Stĺpec `name` reprezentuje identifikáciu záznamu konkrétneho subjektu a nehrá v tejto klasifikácii žiadnu rolu. Preto nepatrí ani do jednej zo spomínaných skupín. Toto rozdelenie je graficky znázornené na obrázku 7.1.



Obrázek 7.1: Rozdelenie dátovej množiny na label a feature atribúty

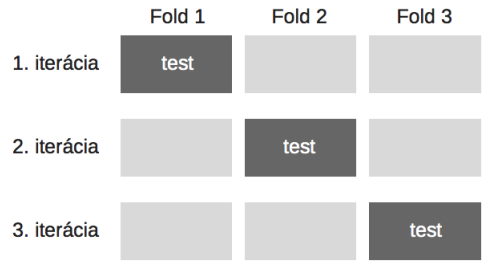
7.3.4 Križová validácia

Učenie parametrov predpovedať klasifikačné triedy a ich testovanie na neustále rovnakej množine dát je hrubá metodologická chyba. Takto vytrénovaný model by síce v danej množine dát mohol dosahovať dobré výsledky, ale nie je všestranne použiteľný. V prípade predikcie na novej množine dát by pravdepodobne vôbec neuspel. Takúto situáciu nazývame preučenie (*overfitting*). Konkrétne sa jedná o stav, kedy sa klasifikátor naučí veľmi presne klasifikovať hodnoty danej množiny dát, ale v prípade výskytu nových dát nie je vôbec úspešný. Aby sa zabránilo vzniku tohto problému, je potrebné dáta rozdeliť na trénovaciu a testovaciu množinu, pričom sa model učí na trénovacej množine a neskôr sa vyhodnocuje jeho aplikáciou na zvolenej testovacej množine. Tento prístup však pri nesprávnom nastavení potrebných parametrov môže opäť viesť k tzv. preučeniu[20]. Najideálnejší prístup je preto využitie križovej validácie. V tejto práci boli využité dva rôzne prístupy:

`KFold()`

`K-Fold` križový validátor rozdelí množinu dát na `k` podmnožín. Každá z týchto podmnožín tvorí minimálne raz testovaciu množinu, zatiaľ čo zvyšných `k-1` podmnožín tvorí trénovaciu

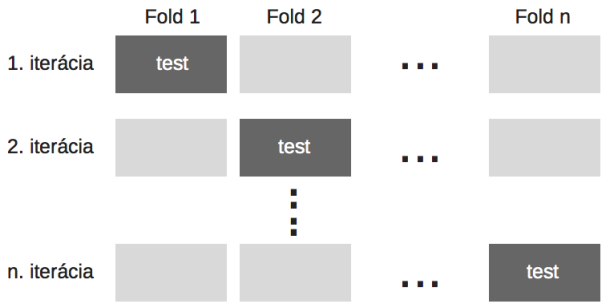
množinu. Pri inicializácii tohto krížového validátora je možné určiť, na koľko podmnožín bude množina dát rozdelená. V tejto práci bola zvolená 10-násobná krížová validácia a teda bol parameter `n_splits` nastavený na hodnotu 10. Na obrázku 7.2 je možné vidieť princíp, na akom použitý algoritmus funguje. Na obrázku je znázornený krížový validátor s parametrom `n_splits` nastaveným na hodnotu 3.



Obrázek 7.2: K-Fold krížová validácia pre k=3

`LeaveOneOut()`

Princíp `Leave One Out` krížového validátora je podobný spomínanému `K-Fold` validátoru. Rozdiel spočíva práve v množstve podmnožín. Pri `Leave One Out` krížovej validácií je tento parameter určený celkovým množstvom vzokov n, v tomto prípade 195. Tento validátor je možné inicializovať aj ako `K-Fold(n_splits=n)`, čiže v prípade tejto dolovacej úlohy by mal parameter `n_splits` hodnotu 195. Oba tieto validátory sú súčasťou voľne dostupného `scikit-learn` softvéru.



Obrázek 7.3: Leave One Out krížová validácia

Spolu s využitím tejto metódy je potrebné správne vyhodnotiť úspešnosť klasifikátora. Po vykonaní klasifikácie s využitím krížovej validácie bol pre celkový výsledok úspešnosti klasifikátora vypočítaný priemer všetkých čiastočných výsledkov (úspechov) v jednotlivých iteráciách krížového validátora a tento priemer sa zvolil ako finálna úspešnosť klasifikátora. Vyhodnotenie klasifikácie je podrobnešie popísané v sekcii 7.3.6.

7.3.5 Výber a inicializácia modelov

Ďalším krokom je výber vhodných klasifikačných modelov. V mnohých prípadoch je ťažké hneď na úvod stanoviť, aký klasifikačný algoritmus je pre danú úlohu najvhodnejší. Preto boli pre tento výskum použité takmer všetky dostupné modely, ktoré poskytuje

`scikit-learn` softvér. Takýmto spôsobom bolo možné skúmať ich správanie a nakoniec vyhodnotiť, ktorý je najúspešnejší a tým pádom najvhodnejší. Použité modely boli spravidla inicializované s prednastavenými hodnotami vstupných parametrov. Pre experimentovanie boli zvolené tieto modely:

`sklearn.tree.DecisionTreeClassifier`

Prvým použitým klasifikátorom je `DecisionTreeClassifier`. Tento klasifikátor je založený na princípe rozhodovacích stromov (viď kapitola 5) a dá sa pomocou neho vykonávať ako binárna, tak aj multitriedna klasifikácia (klasifikácia, ktorej výsledkom je 0 až K-1 klasifikačných tried). Tento klasifikátor je súčasťou `scikit-learn` softvéru, v ktorom bol pre implementáciu rozhodovacieho stromu zvolený optimalizovaný algoritmus `CART`[4].

`sklearn.gaussian_process.GaussianProcessClassifier`

Ďalším použitým modelom je `GaussianProcessClassifier`. Tento model je založený na pravdepodobnosti a implementuje Gaussove procesy za účelom klasifikácie[8]. Pre jeho inicializáciu boli opätovne zvolené prednastavené hodnoty všetkých parametrov. Tieto parametre špecifikujú priebeh všetkých potrebných výpočtov rôznych pravdepodobností. Pre účely tejto úlohy však nie je potrebné tieto výpočty nijako modifikovať.

`sklearn.linear_model.SGDClassifier`

Celý názov modelu `SGDClassifier` je *Stochastic Graient Descent Classifier*. Tento model, ako už z názvu vyplýva, implementuje metódy učenia formou stochastického gradientu.

`sklearn.svm.SVC`

V poradí štvrtým použitým modelom je model `SVC`. Tento model je založený na metóde podporných vektorov alebo inak známej pod skratkou *SVM*.

`sklearn.naive_bayes.GaussianNB`

Predposledným použitým modelom je `GaussianNB`. Jedná sa o model založený na Bayesovskej klasifikácii.

`sklearn.neighbors.nearest_centroid.NearestCentroid`

Posledným použitým modelom je `NearestCentroid`. Tento model je založený na klasifikácii založenej na k-najbližšom susedstve. Pre inicializáciu tohto modelu je potrebné určiť metriku, podľa ktorej sa bude počítať vzdialenosť medzi jednotlivými objektami. V tomto prípade bol tento parameter `metric` nastavený na hodnotu `euclidean`.

7.3.6 Klasifikácia

Po príprave testovacej množiny a trénovacej množiny, prípadnom odstránení kolinearity a inicializácii klasifikačných modelov, nastupuje samotná klasifikácia. Priebeh klasifikácie bol skúmaný na všetkych šiestich inicializovaných modeloch úplne rovnako.

Fáza trénovania

Ako už bolo vysvetlené v kapitole 5, klasifikácia prebieha v dvoch fázach. Prvou z nich je trénovacia fáza, počas ktorej si daný klasifikátor vytvorí klasifikačný model, odpovedajúci tejto trénovacej množine. Respektíve, *naučí* sa čo najlepšie klasifikovať dáta na základe dostupnej trénovacej množiny. Tento proces sa nazýva aj *fitting*. Vďaka vymoženostiam jazyka Python nie je nutné zaoberať sa detajlmi procesu trénovania a vytvárania klasifikačných pravidiel, nakoľko softvér `scikit-learn` ponúka metódu `fit()`, ktorá tento trénovací proces zabezpečuje. V praxi vyzerá volanie tejto metódy nasledovne:

```
clf.fit(X_train, y_train)
```

Objekt `clf` reprezentuje klasifikátor, premenná `X_train` ktorá je dátového typu `DataFrame`, reprezentuje všetky hodnoty *feature* atribútov a ako posledná premenná `y_train`, ktorá je dátového typu `Series`, reprezentuje hodnoty *label* atribútu.

Fáza testovania

Po vytvorení klasifikačného modelu je model pripravený na testovanie. V tejto fáze dostane klasifikátor na vstup hodnoty *feature* atribútov testovacej množiny a jeho úlohou bude predikovať hodnoty *label* atribútu. Konkrétne povedané, klasifikátor dostane na vstup spracované biomedicínske hlasové meriania, na základé ktorých bude určovať, či daný subjekt trpí alebo netrpí Parkinsonovou chorobou. Tento proces zabezpečuje metóda `predict()`, ktorá je taktiež súčasťou `scikit-learn` softvéru. V praxi vyzerá tento proces nasledovne:

```
y_test_predict = clf.predict(X_test)
```

Premenná `y_test_predict`, ktorá je dátového typu `ndarray`, predstavuje výsledky klasifikátora pre testovaciu množinu `X_test`, ktorá je dátového typu `DataFrame` a reprezentuje *feature* atribúty.

Vyhodnotenie modelov

Po trénovacej a testovacej fáze nastáva vyhodnotenie celkovej klasifikácie. Existuje niekoľko metrík, podľa ktorých sa môže stanoviť úspešnosť daného klasifikátora (viď sekcia 5.3). Pre tieto účely bola zvolená metóda `accuracy_score()`, ktorá je súčasťou `scikit-learn` softvéru a reprezentuje metriku s názvom *celková správnosť modelu*. Táto metóda porovnáva hodnoty, ktoré sú výsledkom klasifikátora s hodnotami, o ktorých je známe, že sú reálne a pravdivé. Na to, aby klasifikátor dosiahol 100% úspešnosť, všetky predpovedané hodnoty sa musia zhodovať s reálnymi hodnotami. Výsledok úspešnosti klasifikátora sa teda pohybuje medzi hodnotami 0 až 1, pričom hodnota 1 reprezentuje 100% úspešnosť.

Nakoľko bola pre tento výskum zvolená 10-násobná krížová validícia a taktiež sa experimentovalo s takzvanou Leave One Out (ďalej len LOO) krížovou validáciou, proces klasifikácie prebiehal v niekoľkých cykloch. Pri 10-násobnej krížovej validácií sa jedná konkrétne o 10 cyklov a pri LOO krížovej validácií o 195 cyklov. Z toho vyplýva, že v každom cykle sa vyhodnotí úspešnosť klasifikátora na inej trénovacej a testovacej množine. Tieto čiastočné výsledky je potrebné uchovávať, aby bolo možné po ukončení cyklu vyhodnotiť celkovú úspešnosť klasifikátora. V tejto práci to bolo zvládnuté ukladaním čiastočných výsledkov do premennej dátového typu `list` a využitím metódy `mean()`, ktorá je súčasťou knižnice `Numpy`. Najprv sa konvertuje dátový typ `list` na dátový typ `ndarray` a potom táto metóda vypočíta aritmetický priemer prvkov poľa, pričom prvky poľa v tomto prípade predstavujú čiastočné výsledky klasifikácie.

Pre príklad je možné uviesť pseudokód popisujúci inicializáciu klasifikačného modelu a križového validátora, klasifikáciu a výpočet celkovej úspešnosti tohto modelu po klasifikácií.

```
# classification model and cross validator initialization
clf = sklearn.tree.DecisionTreeClassifier()
cross_validator = sklearn.model_selection.KFold(n_splits=10)

# array used to store classification scores
decision_tree_scores = []

for train_index, test_index in cross_validator.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    dt_score = classification(clf, X_train, X_test, y_train, y_test)
    decision_tree_scores.append(dt_score)
...
# classification() method
clf.fit(X_train, y_train)
y_test_predict = clf.predict(X_test)
return (accuracy_score(y_test.values, y_test_predict))
...
# get final score
average_score_dt = np.mean(decision_tree_scores)
```

Výsledky klasifikácie

V nasledujúcej časti budú uvedené výsledky jednotlivých klasifikačných metód. V tabuľke 7.1 sa nachádzajú výsledky po klasifikácii, pre ktorú bola zvolená 10-násobná križová validácia. V prvom stĺpci `Model` sú uvedené použité modely. V druhom stĺpci, s názvom `All`, sú uvedené výsledky jednotlivých modelov za použitia všetkých pôvodných atribútov. V poslednom stĺpci s názvom `VIF` sa nachádzajú výsledky jednotlivých modelov za použitia zredukovaného počtu atribútov - po odstránení kolienarity.

Model	All	VIF
DecisionTreeClassifier	0.7687	0.7832
GaussianProcessClassifier	0.7008	0.7008
SGDClassifier	0.5032	0.5787
SVC	0.7274	0.7484
GaussianNB	0.6695	0.7408
NearestCentroid	0.6945	0.6945

Tabulka 7.1: Výsledky klasifikácie s 10-násobným križovým validátorom

Z tejto tabuľky je možné vyčítať, že najúspešnejším modelom sa stal model založený na metóde rozhodovacích stromov - `DecisionTreeClassifier` a to v oboch prípadoch. S použitím všetkých dostupných atribútov dosiahol celkovú úspešnosť ~77% a so zredu-

kovaným počtom atribútov, po odstránení vyskytujúcej sa kolinearity, dosiahol úspešnosť ~78%. Rovnako veľmi dobré výsledky dosiahol aj model `SVC`, ktorý za použitia všetkých atribútov dosiahol úspešnosť ~73% a po odstránení kolinearity ~75%.

V tabuľke 7.2 sa nachádzajú výsledky klasifikácie, ktorá bola vykonaná s LOO križovým validátorom. Už na prvý pohľad je vidieť, že výsledky sú o niečo lepšie pre každý model. Dôvodom je najmä pomer trénovacej a testovacej množiny, ktorý je 194:1. Aj v tomto prípade dosiahol najlepšie výsledky v oboch prípadoch klasifikátor `DecisionTreeClassifier`. V klasifikácií so všetkými atribútmi dosiahol úspešnosť ~86% a v klasifikácií so zredukovaným počtom atribútov dosiahol úspešnosť ~89%. Veľmi dobré výsledky dosiahol aj klasifikátor `GaussianProcessClassifier`, ktorý v oboch prípadoch dosiahol rovnakú úspešnosť ~84%.

Model	All	VIF
DecisionTreeClassifier	0.8615	0.8872
GaussianProcessClassifier	0.8359	0.8359
SGDClassifier	0.6103	0.6718
SVC	0.7897	0.7795
GaussianNB	0.7128	0.7744
NearestCentroid	0.7179	0.7179

Tabulka 7.2: Výsledky klasifikácie s LOO križovým validátorom

Uvedené výsledky sa môžu každým spustením skriptu s implementovaným procesom klasifikácie o niečo líšiť. Dôvodom je vždy náhodný výber záznamov trénovacej a testovacej množiny. Rozdiel však spočíva v stotinách, občas desatinách čísla.

Ako najvhodnejší klasifikátor sa ukázal `DecisionTreeClassifier`, ktorý dosiahol najlepšie výsledky za použitia oboch metód križovej validácie.

V tabuľke 7.3 sa nachádza matica zmien (viď 5.3.2) pre túto metódu, ktorá znázorňuje výsledky klasifikácie, pri ktorej boli použité všetky atribúty a 10-násobná križová validácia.

Na druhom obrázku 7.4 sa nachádza matica zmien znázorňujúca výsledky klasifikácie, pri ktorej bolo zredukované množstvo atribútov a použitá 10-násobná križová validácia. Takúto tabuľku je možné zostaviť pomocou metódy `confusion_matrix()`, ktorá je súčaťou `scikit-learn` softvéru.

Z tabuľky 7.3 je možné vyčítať, že klasifikátor správne určil prítomnosť Parkinsonovej choroby (hodnota 1) v 132 prípadoch a neprítomnosť Parkinsonovej choroby (hodnota 0) v 18 prípadoch. Klasifikátor bol vo zvyšných prípadoch neúspešný a klasifikoval v 30 prípadoch prítomnosť Parkinsonovej choroby, kedy reálne výskyt tejto choroby nebol a naopak určil neprítomnosť choroby v 15 prípadoch, kedy sem výskyt tejto choroby bol. Celková úspešnosť klasifikátora je teda

$$\frac{150}{195} \cong 0,77 \cong 77\%$$

		predicted	
		0	1
actual	0	18	30
	1	15	132

Tabulka 7.3: Matica zmien pre metódu `DecisionTreeClassifier` s použitím 10-násobnej križovej validácie a všetkých atribútov

Z druhej tabuľky 7.4 je možné vyčítať, že klasifikátor správne určil prítomnosť Parkinsonovej choroby v 131 prípadoch a neprítomnosť Parkinsonovej choroby v 22 prípadoch. Klasifikátor bol vo zvyšných prípadoch neúspešný a klasifikoval v 26 prípadoch prítomnosť Parkinsonovej choroby, kedy reálne výskyt tejto choroby nebol a naopak určil neprítomnosť choroby v 16 prípadoch, kedy sem výskyt tejto choroby bol. Celková úspešnosť klasifikátora je teda

$$\frac{153}{195} \cong 0,78 \cong 78\%$$

		predicted	
		0	1
actual	0	22	26
	1	16	131

Tabulka 7.4: Matica zmien pre metódu `DecisionTreeClassifier` s použitím 10-násobnej križovej validácie a po odstránení kolinearity

Ďalšie metódy klasifikácie

Na základe predchádzajúcich experimentov s dostupnými klasifikačnými modelmi sa podarilo zistiť, že najvhodnejšou klasifikačnou metódou (podľa zvolenej metriky celkovej správnosti modelu), pre spomínaný typ dolovacej úlohy, je práve klasifikácia pomocou rozhodovacích stromov. Tento výsledok vytvoril podnet na experimentáciu s touto metódou s cieľom dosiahnuť ešte o niečo lepší výsledok.

Vo všeobecnosti, jazyk Python a knižnica `scikit-learn` ponúka metódy, nazývané kombinované metódy alebo *ensemble methods*, ktorých fungovanie je založené na kombinácií rôznych jednoduchších algoritmov strojového učenia. Niektoré z týchto metód využívajú práve princíp rozhodovacích stromov. Medzi ne patria:

- `RandomForestClassifier()`
  Náhodný les alebo *random forest* je metóda, ktorá vytvára z rozhodovacích stromov *les*. Náhodnosť je základný rozdiel oproti metóde `DecisionTreeClassifier()`. V tomto prípade je vždy výber koreňa každého stromu v lese úplne náhodný, zatiaľ čo v metóde `DecisionTreeClassifier()` sa vždy vyberá ten, ktorý má najvyššiu rozhodovaciu schopnosť. Ako prvé sa vytvorí náhodný les zložený z určitého počtu rozhodovacích stromov. Tu platí, že čím je tento les väčší (čím obsahuje väčší počet stromov), tým je klasifikátor presnejší[9]. Ďalším rozdielom oproti jednoduchej metóde rozhodovacích stromov je vyhodnotenie. Po klasifikácii pomocou metódy náhodneho lesa klasifikátor vráti najčastejšiu hodnotu (modus) tried, ktoré vrátili jednotlivé stromy[13].

- `GradientBoostingClassifier()`
  Princíp metódy zvyšovania stromového gradientu alebo *Gradient tree boosting* je opäť úplne odlišný, no rovnako ako predchádzajúca kombinovaná metóda, využíva rozhodovacie stromy. Proces klasifikácie prebieha tak, že sa na začiatku vytvorí prvý klasifikačný strom, ktorý sa učí klasifikovať na určitej, náhodne vybranej podmnožine dát z trénovacej množiny. Po naučení klasifikátora sa na jeho otestovanie použije rovnaká sada dát a prehľadávajú sa záznamy, pri ktorých klasifikácia nebola úspešná. V nasledujúcom kroku sa vytvorí ďalší strom, ktorého vstupom sú opäť náhodne

vybrané dáta z trénovaej množiny. V tomto prípade sa však vyberajú dáta s tým rozdielom, že nesprávne klasifikované dáta z predchádzajúceho stromu sa vyberú s väčšou pravdepodobnosťou. Tento postup pokračuje, až kým klasifikátor nedosiahne zvolený maximálny počet stromov. Výsledok je potom kombináciou výsledkov všetkých čiastočných vygenerovaných stromov[3].

- `BaggingClassifier()`
  Poslednou kombinovanou metódou, ktorá využíva rozhodovacie stromy je metóda nazývaná *bagging*. Táto metóda je veľmi podobná predchádzajúcej metóde *gradient tree boosting*. V tomto prípade sa však nepočíta žiadna vznikajúca chyba v predchádzajúcom strome pri tvorbe toho nasledujúceho. Najprv sa vytvorí predom určený počet stromov, ktoré dostanú na vstup úplne náhodnú podmnožinu dát z trénovacej množiny. Takto sa vytrénuje každý jeden vytvorený strom a výsledkom je opäť kombinácia výsledkov všetkých čiastočný stromov.

Takto vzniknuté a vytrénované modely sa v testovacej fáze testujú na testovacej množine a vyhodnocujú sa na základe zvolenej metriky, ktorá je pre danú dolovaciu úlohu najideálnejšia, rovnako ako v predchádzajúcej klasifikácii. V ďalšej časti praktickej časti tejto práce bude popísaný postup a výsledky klasifikácie prostredníctvom týchto kombinovaných metód.

Je potrebné brať na vedomie, že s rastúcim počtom vygenerovaných stromov sa zvyšuje výpočtová záťaž, no rozdiely v presnostiach sú už ďalej veľmi malé [32]. Preto boli tieto modely inicializované s hodnotou parametra určujúceho maximálny počet stromov (`n_estimators`) najprv na 10, potom na 30 a nakoniec na 50. Pre zvyšné parametre týchto modelov boli použité prednastavné hodnoty. Nasledujúci priebeh klasifikácie bol takmer identický s priebehom klasifikácie v sekcii 7.3.6. Rozdielom oproti predchádzajúcej klasifikácii je vynechanie `LeaveOneOut` križovej validácie a použitie iba 10-násobnej `KFold` križovej validácie. Pre túto klasifikáciu boli použité rovnaké *feature* atribúty a rovnaký *label* atribút (viď sekcia 7.3.3). Ďalej boli použité najprv všetky atribúty a neskôr bolo množstvo atribútov zredukované odstránením kolinearity (viď sekcia 7.3.2). Po vytrénovaní modelov prebehlo ich testovanie a vyhodnotenie úspešnosti podľa rovnakej metriky ako v predchádzajúcej klasifikácii a tou je celková úspešnosť modelu. Po vyhodnotení bola pre každý model vytvorená matica zmien, určená na vizualizáciu najvyššej úspešnosti daného modelu.

Výsledky klasifikácie s použitím metódy `RandomForestClassifier` sa nachádzajú v tabuľke 7.5. S použitím všetkých atribútov tento model dosiahol najlepší výsledok pri vygenerovaní 50 stromov a jeho celková úspešnosť bola ~78%. Po odstránení kolinearity bol model rovnako najúspešnejší pri počte vygenerovaných stromov 50 a dosiahol úspešnosť ~80%.

Trees	All	VIF
10	0.7571	0.7918
30	0.7524	0.7937
50	0.7776	0.8039

Tabulka 7.5: Výsledky klasifikácie s použitím metódy `RandomForestClassifier`

V matici zmien v tabuľke 7.6 je znázornená najvyššia dosiahnutá úspešnosť tohto modelu, ktorá bola docielená použitím zredukovaného počtu atribútov a vygenerovaním lesa o veľkosti 50 stromov. Z matice je možné vyčítať, že model úspešne klasifikoval chorých ľudí

v 139 prípadoch z celkových 147 prípadov a zdravých ľudí v 18 prípadoch z celkových 48 prípadov.

		predicted	
		0	1
actual	0	18	30
	1	8	139

Tabulka 7.6: Matica zmien pre metódu `RandomForestClassifier`

Výsledky ďalšej klasifikácie, tentokrát s použitím metódy `GradientBoostingClassifier`, je možné nájsť v tabuľke 7.7. S použitím všetkých atribútov tento model dosiahol najlepší výsledok pri vygenerovaní 50 stromov a jeho celková úspešnosť bola ~81.5%. Po odstránení kolinearity bol model najúspešnejší s počtom vygenerovaných stromov taktiež 50 a dosiahol úspešnosť 83%.

Trees	All	VIF
10	0.8097	0.7945
30	0.8145	0.8097
50	0.8147	0.8300

Tabulka 7.7: Výsledky klasifikácie s použitím metódy `GradientBoostingClassifier`

Pre najúspešnejšiu klasifikáciu tejto metódy bola taktiež vytvorená matica zmien, znázornená v tabuľke 7.8. Matica je vytvorená pre klasifikáciu s použitím 50 rozhodovacích stromov s redukovaným počtom atribútov. Z tejto matice je možné vyčítať, že model úspešne klasifikoval chorých ľudí v 141 prípadoch z celkových 147 prípadov a zdravých ľudí v 21 prípadoch z celkových 48 prípadov.

		predicted	
		0	1
actual	0	21	27
	1	6	141

Tabulka 7.8: Matica zmien pre metódu `GradientBoostingClassifier`

Ako posledný bol použitý model `BaggingClassifier`. Úspešnosť tohto modelu sa nachádza v tabuľke 7.9. S použitím všetkých atribútov dosiahol tento model navyššiu úspešnosť ~80% pri vygenerovaní 50 rozhodovacích stromov. Po odstránení kolinearity sa tomuto modelu podarilo dosiahnuť najlepšiu úspešnosť ~79% a to pri vygenerovaní iba 10 stromov.

Trees	All	VIF
10	0.7566	0.7879
30	0.7779	0.7779
50	0.7932	0.7826

Tabulka 7.9: Výsledky klasifikácie s použitím metódy `BaggingClassifier`

Ako v predchádzajúcom prípade, tak aj teraz, je v tabuľke 7.10 znázorená najlepšia klasifikácia metódy `BaggingClassifier`. Z tejto matice je možné vyčítať, že model úspešne

klasifikoval chorých ľudí v 133 prípadoch z celkových 147 prípadov a zdravých ľudí v 21 prípadoch z celkových 48 prípadov.

		predicted	
		0	1
actual	0	21	27
	1	14	133

Tabulka 7.10: Matica zmien pre metódu `BaggingClassifier`

Týmto experimentom sa podarilo zvýšiť úspešnosť klasifikácie založenej na metóde rozhodovacích stromov. Výsledok sa podarilo najviac vylepšiť pomocou metódy `GradientBoostingClassifier`, ktorý pri použití všetkých atribútov dosiahol úspešnosť 81.5% a po odstránení kolienarity sa mu podarilo dosiahnuť úspešnosť 83%.

Kapitola 8

Výhody a nevýhody použitia jazyka Python

V záverečnej časti tejto práce bude poskytnutý ako subjektívny, tak aj objektívny pohľad na prácu s programovacím jazykom Python pri riešení úlohy z oblasti získavania znalostí z dát a dátovej analýzy.

8.1 Jazyk Python

Jazyk Python je interpretovaný, interaktívny programovací jazyk, ktorý podporuje ako objektovo orientované, tak aj štruktúrované a funkcionálne programovanie. Jedná sa o dynamicky typovaný jazyk podporujúci množstvo dátových typov[1].

Tento jazyk je všeobecne známy ako skriptovací jazyk, no využíva sa aj na vývoj veľkých softvérových projektov a nie len na tvorbu jednoduchých shell skriptov. Tento jazyk si osvojili aj veľké spoločnosti ako je napríklad Google alebo BitTorrent.

8.2 Výhody

V prvom rade je potrebné vyzdvihnúť zopár výhod, ktoré prináša práca s týmto jazykom vo všeobecnosti.

Veľkou prednosťou tohto jazyka je interaktívny režim, ktorý ponúka jeho interpreter. Počas tvorby kódu je možné zadávať do terminálu rôzne výrazy, ktoré sa ihneď vyhodnocujú a tým pádom je možné ich testovať pred samotnou integráciou do kódu. Počas tvorby skriptu demonštrujúceho prácu s jazykom Python bol využitý interaktívny shell `IPython`[10], ktorý navyše podporuje aj automatické dopĺňanie výrazov a zvýrazňovanie textu.

Ďalšou výhodou jazyka Python je jeho jednoduchosť, skvelá čitateľnosť kódu a dynamické typovanie.

Z pohľadu dátovej analýzy je potrebné vyzdvihnúť široké spektrum modulov určených na prácu v tejto oblasti, ktoré rozširujú základnú funkcionalitu jazyka Python. Medzi ne patria najmä `Numpy` a `Pandas`, ktoré ponúkajú potrebné dátové typy, uľahčujúce prácu s dátami a dátovými tabuľkami. Víťazom tejto kategórie je však jednoznačne softvér `scikit-learn`. Tento softvér prakticky umožňuje vykonávať dátovú analýzu v jazyku Python, nakoľko mu ponúka plnú funkcionalitu a podporu pre predspracovanie dát, dolovanie dát prostredníctvom širokého spektra metód a zároveň aj ich vyhodnotenie podľa zvolených metrík.

8.3 Nevýhody

Pri tvorbe programu, demonštrujúceho dátovú analýzu v jazyku Python, neboli pociťované takmer žiadne nedostatky tohto jazyka, nakoľko sa jednalo o relatívne krátky a výpočetne nie veľmi náročný program. Je však všeobecne známe, že interpretovaný jazyk Python je oproti iným, kompilovaným jazykom ako je C alebo C++, o dosť pomalší. Pri tvorbe výpočetne náročnejšieho kódu je preto potrebné zvážiť, ako veľmi je kladený dôraz na rýchlosť programu.

Ďalšou a poslednou nevýhodou, o ktorej bude v tejto práci zmienka, opäť súvisí s interpretovanosťou tohto jazyka a zároveň s tým, že sa jedná o dynamicky typovaný jazyk. Po spustení sa program prekladá riadok po riadku a mnoho krát sa chyby v programe odhalia až počas behu programu. Preto je potrebné kód vždy dôkladnejšie otestovať.

Kapitola 9

Záver

Cieľom tejto práce bolo oboznámiť sa so základmi získavania znalostí z dát a dostupnými prostriedkami na podporu tohto procesu v programovacom jazyku Python. Ďalej bolo cieľom získané znalosti aplikovať a demonštrovať na zvolenej analytickej úlohe.

Pre splnenie tohto cieľa bolo potrebné zhromaždiť dostatočný počet materiálov a naštudovať si základné princípy získavania znalostí z dát. Nakoľko bola zvolená analytická úloha určená práve na klasifikáciu, tak bol na tento proces kladený vyšší dôraz a študovaná látka bola preberaná do väčšej hĺbky.

Pred implementáciou programu demonštrujúceho proces dolovania dát bolo potrebné naštudovať si najmä dokumentácie dostupných prostriedkov pre podporu riešenia danej úlohy a vyhodnotiť, ktoré z dostupných prostriedkov je vhodné použiť.

Konečné riešenie úlohy dostatočne a jasne demonštruje základné kroky dolovania dát a ich spracovanie pomocou dostupných modulov jazyka Python. V programe boli využité najmä prostriedky určené na klasifikáciu a vyhodnotenie dolovacieho procesu. Toto riešenie však nie je konečné a je možné s ním rôzne experimentovať podľa danej potreby a ďalších cieľov.

Literatura

[1] *Advantages and Disadvantages of Python Programming Language*. [Online; navštívené 30.04.2018].
URL https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121

[2] *Bayes' theorem*. [Online; navštívené 1.5.2018].
URL https://en.wikipedia.org/wiki/Bayes%27_theorem

[3] *Boosting*. [Online; navštívené 04.05.2018].
URL https://www.youtube.com/watch?v=GM3CDQfQ4sw

[4] *CART Algorithm*. [Online; navštívené 28.04.2018].
URL ftp://ftp.boulder.ibm.com/software/analytics/spss/support/Stats/Docs/Statistics/Algorithms/14.0/TREE-CART.pdf

[5] *Classification Problems in Real Life*. [Online; navštívené 16.04.2018].
URL https://onlinecourses.science.psu.edu/stat857/node/147

[6] *Clustering Distance Measures Essentials*. [Online; navštívené 1.5.2018].
URL http://www.sthda.com/english/articles/26-clustering-basics/86-clustering-distance-measures-essentials/

[7] *Cross-validation: evaluating estimator performance*. [Online; navštívené 18.04.2018].
URL http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-evaluating-estimator-performance

[8] *Gaussian Processes*. [Online; navštívené 30.04.2018].
URL http://scikit-learn.org/stable/modules/gaussian_process.html

[9] *How Random Forest Algorithm Works in Machine Learning*. [Online; navštívené 04.05.2018].
URL https://medium.com/@Synced/how-random-forest-algorithm-works-in-machine-learning-3c0fe15b6674

[10] *IPython - Interactive Computing*. [Online; navštívené 30.04.2018].
URL https://ipython.org

[11] *Machine Learning in Python*. [Online; navštívené 13.04.2018].
URL http://scikit-learn.org/stable/

[12] *Model evaluation: quantifying the quality of predictions*. [Online; navštívené 18.04.2018].

URL http://scikit-learn.org/stable/modules/model_evaluation.html#model-evaluation

[13] *Náhodný strom*. [Online; navštívené 04.05.2018].
URL https://cs.wikipedia.org/wiki/Náhodný_les

[14] *Preprocessing data*. [Online; navštívené 18.04.2018].
URL
http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing

[15] *Python Package Index*. [Online; navštívené 08.04.2018].
URL https://pypi.python.org/pypi

[16] *Supervised learning*. [Online; navštívené 18.04.2018].
URL http://scikit-learn.org/stable/supervised_learning.html#supervised-learning

[17] *Variance Inflation Factor (VIF) Explained*. [Online; navštívené 18.04.2018].
URL https://etav.github.io/python/vif_factor_python.html

[18] *Verifikácia predikčných modelov*. [Online; navštívené 1.5.2018].
URL http://emijournal.cz/wp-content/uploads/2017/12/03_verifikacia-predikčných-modelov.pdf

[19] Cibula, M.: *Support Vector Machines*. [Online; navštívené 21.04.2018].
URL http://smnd.sk/mcibula/index.html

[20] *Cross-validation: evaluating estimator performance*. [Online; navštívené 08.04.2018].
URL http://scikit-learn.org/stable/modules/cross_validation.html

[21] Kamber, M.; Han, J.; Pei, J.: *Data mining*. Elsevier Inc., 2012, ISBN 978-0-12-381479-1.

[22] *What is NumPy?* [Online; navštívené 13.04.2018].
URL https://docs.scipy.org/doc/numpy-1.13.0/user/whatisnumpy.html

[23] *Causes & Statistics*. Parkinson's Foundation, [Online; navštívené 08.04.2018].
URL
http://parkinson.org/Understanding-Parkinsons/Causes-and-Statistics

[24] *Movement Symptoms*. [Online; navštívené 08.04.2018].
URL http://parkinson.org/Understanding-Parkinsons/Symptoms/Movement-Symptoms

[25] *Non-Movement Symptoms*. [Online; navštívené 08.04.2018].
URL http://parkinson.org/Understanding-Parkinsons/Symptoms/Non-Movement-Symptoms

[26] *What Is Parkinson's?* [Online; navštívené 08.04.2018].
URL
http://www.parkinson.org/understanding-parkinsons/what-is-parkinsons

[27] Patel, S.: *K Nearest Neighbors Classifier*. [Online; navštívené 18.04.2018].
URL https://medium.com/machine-learning-101/k-nearest-neighbors-classifier-1c1ff404d265

[28] Patel, S.: *Symptoms – Speech Difficulties or Changes*. [Online; navštívené 20.04.2018].
URL https://parkinsonsdisease.net/symptoms/speech-difficulties-changes/

[29] *Why Learn Python?* [Online; navštívené 09.04.2018].
URL http://www.bestprogramminglanguagefor.me/why-learn-python

[30] *Scientific Computing Tools for Python*. [Online; navštívené 13.04.2018].
URL https://www.scipy.org/about.html

[31] Shafranovich, Y.: *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. SolidMatrix Technologies, Inc., [Online; navštívené 08.04.2018].
URL https://tools.ietf.org/html/rfc4180

[32] Steinberger, J.; Zíma, M.; Fiala, D.: *Data a znalosti 2017*. Západočeská univerzita v Plzni, 2017, ISBN 978-80-261-0720-0.
URL https://daz2017.kiv.zcu.cz/data/DaZ2017-Sbornik-final.pdf

[33] *Center for Machine Learning and Intelligent Systems – Parkinsons Data Set*. [Online; navštívené 08.04.2018].
URL https://archive.ics.uci.edu/ml/datasets/Parkinsons

[34] Viktorin, P.; Hrončok, M.: *Analýza dat v Pythonu*. [Online; navštívené 10.04.2018].
URL http://naucse.python.cz/lessons/intro/pandas/

[35] Zendulka, J.; Bartí, V.; Lukáš, R.; aj.: *Získávání znalostí z databází, Studijní opora*. Jaroslav Zendulka a kol., [Online; navštívené 15.02.2018].
URL http://www.fit.vutbr.cz/~zendulka/tmp/ZZN.pdf