EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 103004/I/2022/421000162356

Strojové učenie klasifikácie v Pythone

Diplomová práca

2022 **Bc. Ondrej Izakovič**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Strojové učenie klasifikácie v Pythone

Diplomová práca

Študijný program: Informačný manažment

Študijný odbor: Ekonómia a manažment

Školiace pracovisko: Katedra aplikovanej informatiky FHI

Vedúci záverečnej práce: doc. Dr. Ing. Miroslav Hudec

Bratislava 2022 **Bc. Ondrej Izakovič**

Ekonomická univerzita v Bratislave
Fakulta hospodárskej informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Ondrej Izakovič
Študijný program:	informačný manažment (Jednoodborové štúdium, inžiniersky II. st., denná forma)
Študijný odbor:	ekonómia a manažment
Typ záverečnej práce:	Inžinierska záverečná práca
Jazyk záverečnej práce:	slovenský
Sekundárny jazyk:	anglický

Názov: Strojové učenie klasifikácie v Pythone

Anotácia: Cieľom práce jeo preskúmať strojové učenie v úlohách klasifikácie a vytvoriť kód v Pythone pre učenie klasifikovania s dohľadom.

Väčšie objemy dát nie je jednoduché klasifikovať do niekoľkých (často dvoch) tried. V učení s učiteľom pre určité množstvo dát je známa príslušnosť do tried. Tieto dáta sa použijú na učenie a validáciu klasifikácie vytvorenej v Pythone. Následne sa naučený model použije na klasifikáciu ostatných dát.

Vedúci:	doc. Dr. Ing. Miroslav Hudec
Katedra:	KAI FHI - Katedra aplikovanej informatiky FHI
Vedúci katedry:	Ing. Mgr. Peter Schmidt, PhD.
Dátum zadania:	30.10.2020

Dátum schválenia: 30.10.2020

Ing. Mgr. Peter Schmidt, PhD.
vedúci katedry

Čestné vyhlásenie

Čestne vyhlasujem, že som záverečnú prácu vypracoval samostatne a že som uviedol všetku dostupnú literatúru.

Dátum: .....

(podpis študenta)

ABSTRAKT

IZAKOVIČ, Ondrej: *Strojové učenie klasifikácie v Pythone*. – Ekonomická univerzita v Bratislave. Fakulta Hospodárskej informatiky; Katedra aplikovanej informatiky FHI. – Vedúci záverečnej práce: doc. Dr. Ing. Miroslav Hudec – Bratislava: FHI EU, 2022, počet strán 58.

Záverečná práca je vypracovaná na tému Strojové učenie klasifikácie v Pythone. Cieľom záverečnej práce bolo preskúmať strojové učenie v úlohách klasifikácie a vytvoriť kód v Pythone pre učenie klasifikovania s dohľadom. Pri veľkom objeme dát nie je jednoduché ich zaradiť do tried. Avšak z časti dát, ktorých zatriedenie poznáme, je možné vytvoriť a validovať klasifikačný model v jazyku Python. Takýto model môžeme následne použiť pri klasifikácii ostatných údajov. Práca bola rozdelená do troch kapitol. Prvá kapitola sa venovala súčasnému stavu problematiky doma a v zahraničí. Zaoberala sa teoretickým vymedzením klasifikácie, problémov, na riešenie ktorých sa používa a aké nástroje využíva. Druhá kapitola opisuje cieľ a metódy použité na jeho dosiahnutie. Tretia kapitola sa zameriava na použitie algoritmov a modelov v jazyku Python. Výsledkom riešenia danej problematike je porovnanie algoritmov a klasifikačných modelov a ich výsledkov.

Kľúčové slová:

klasifikácia, klasifikačný algoritmus, Naivný Bayesov klasifikátor, Metóda najbližšieho suseda, rozhodovací strom, support vector machines

ABSTRACT

IZAKOVIČ, Ondrej: *Machine learning of the classification tasks by Python*. – University of Economics in Bratislava. Faculty of Economic Informatics; Department of Applied Informatics. – The supervisor of final work: doc. Dr. Ing. Miroslav Hudec – Bratislava: FHI EU, 2022, number of pages 58.

Topic of this paper is Machine learning of the classification tasks by Python. This paper aims to explore machine learning methods used to solve classification problems and to create sample code in Python using supervised learning. In huge datasets it becomes a challenge to classify data. While using smaller set of labelled it was possible to create and validate classifier in Python programming language. This classification model was later used to classify all the unseen instances. The thesis is divided to three chapters. The first chapter is devoted to the current state of the issue at home and abroad, which deals with the theoretical definition of classification and classification problems and methods that are used to solve these problems. The next part described the goal of this paper and methods used to achieve it. The third chapter showed how to write classifiers in Python. The final part deals with result of the solution of this issue achieved with algorithms and models used for classification.

Keywords:

classification, classification algorithm, Naïve Bayes Classifier, Nearest Neighbour classifier, decision tree, support vector machines

O B S A H str.

Úvod

S rozvojom internetu a big data rastie aj potreba analyzovať veľké množstvo údajov. Výsledkom toho je viditeľný rozvoj v oblasti dolovania údajov a data science, alebo inak vede o dátach. Klasifikácia patrí k prvým krokom pri dolovaní údajov a je azda najdôležitejším nástrojom strojového učenia používaného práve pri získavaní informácii z údajov. Klasifikácia predstavuje proces zatrieďovania údajov do dvoch alebo viacerých tried, pričom môže ísť o rôzne typy údajov a je jedno, či sú súčasťou big data, alebo len malých datasetov. Výsledky klasifikácie sú veľmi dôležité pri vykonávaní rozhodnutí či už v medicíne, podnikovej praxi, marketingu, pri riadení, ale aj pri sociálnych vedách. Táto práca sa zameriava na klasifikáciu ako dôležitý nástroj strojového učenia.

V prvej kapitole sa oboznámime s teóriou ohľadom klasifikácie do väčšej hĺbky. Prejdeme si jej využitie v praxi, ale aj to, aké údaje skúma. Predstavíme si pojmy testovacia a trénovacia množina. Následne sa oboznámime s algoritmami a metódami používanými pri klasifikácii, ako je Bayesov naivný klasifikátor, metóda najbližšieho suseda, ako sú pravidlové systémy a rozhodovacie stromy a support vector machines. V neposlednom rade si vysvetlíme, na základe čoho si vybrať správny klasifikátor.

V druhej kapitole sa bližšie oboznámime s cieľom práce, metódami a metodikami, ktoré pri jeho dosiahnutí použijeme.

V tretej kapitole si ukážeme, ako sa jednotlivé klasifikačné algoritmy tvoria v jazyku Python. Začneme tvorbou testovacej množiny. Následne vytvoríme vhodný klasifikátor. Prejdeme si taktiež výstupy a zhodnotíme efektivitu jednotlivých klasifikačných modelov.

Vo štvrtej kapitole porovnáme jednotlivé algoritmy, zhrnieme ich klady a zápory. Na záver ešte vykonáme jeden praktický test a porovnáme navzájom výsledky modelov.

1. Súčasný stav riešenej problematiky doma a v zahraničí

1.1 Klasifikácia

Klasifikácia je činnosť, ktorú vykonávame na dennej báze toho, aby sme o nej uvažovali. V podstate ide o zaraďovanie objektov do jedinečných tried, pričom každý objekt je zaradený práve do jednej triedy. To znamená, že nemôže byť zaradený do viac ako jednej triedy, ani nemôže ostať nezatriedený. Výnimkou je fuzzy klasifikácia, kedy sú prvky množiny zaradené do jednej, alebo aj viacerých tried. V takomto prípade každý prvok prislúcha do tried v miere nazývanej stupeň príslušnosti (Amo, 1999). Hlavným znakom objektov jednej triedy je, že sú si navzájom sú podobné, zatiaľ čo sa odlišujú od prvkov iných tried (Hussein, 2019).

Mnohé rozhodovacie problémy vieme naformulovať ako klasifikačné problémy, pričom ľudí, alebo iné objekty triedime do kategórii na základe určitých vonkajších vlastností. Napríklad študentov môžeme zatriediť podľa prospechu na tých, ktorí prospeli, na tých, ktorí prospeli s vyznamenaním, a na tých, ktorí neprospeli vôbec.

V praxi sa môžeme stretnúť aj so situáciou, kedy existujú len dve triedy. V takomto prípade ide o binárnu klasifikáciu. Ako príklad môžeme uviesť zákazníka, ktorý sa rozhoduje, či si kúpi, alebo nekúpi určitý tovar.

Pri riešení problémov, kedy je potrebné roztriediť určitú množinu údajov do tried, nie je klasifikácia jediným použiteľným nástrojom. Často používaná je aj zhluková analýza (clustering). Medzi zhlukovaním a klasifikáciou však existuje jeden zásadný rozdiel, a to aj napriek tomu, že výsledkom sú na prvý pohľad len údaje zatriedené do množín *podobných* údajov. Pri zhlukovaní sú údaje zaraďované do k tried na základe ich podobnosti. Pri klasifikácii sú údaje zaraďované do vopred stanoveného počtu k tried, pričom sa v prvom rade model musí naučiť triediť údaje do skupín na základe skúseností s trénovacou množinou D. Pri zhlukovaní však k počet zhlukov nie je známy. Stanovujeme rôzne k zhlukov, pričom sa snažíme minimalizovať cieľovú funkciu pri čo najmenšom počte zhlukov (Bramer, 2020). Z hľadiska strojového učenie teda zhlukovú analýzu pokladáme za typ strojového učenia nazývaného ako učenie pozorovaním, práve kvôli absencii špeciálnej množiny, na základe ktorej by sa model „naučil“ špecifiká jednotlivých tried. Klasifikácia na druhej strane patrí k takzvanému „učeniu s učiteľom“ .

1.2 Trénovacia množina

Trénovacia množina je súbor cvičných dát, z ktorých sa algoritmus strojového učenia učí nájsť vzory a súvislosti medzi údajmi. Prvky trénovacej množiny pozostávajú vždy z dvoch typov prvkov – *features* a *labels*. Features sú skúmané vlastnosti objektu, atribúty, zatiaľ čo label predstavuje príslušnosť prvku do triedy. Na týchto datasetoch sa programy učia odhadovať klasifikáciu ešte nezatriedeného prvku.

Trénovaciu množinu M môžeme vyjadriť ako maticu D rozmerov $N * d$, ktorá obsahuje N riadkov, z ktorých každý má priradenú práve jednu z k tried s príznakom $\{1...k\}$. Každý riadok predstavuje jeden prvok množiny. Na základe modelu M je možné predpovedať príslušnosť tried d-dimenzionálneho záznamu (Aggarwal, 2015).

1.3 Klasifikácia ako nástroj Data Miningu

Data Mining predstavuje, ako už samotný názov vypovedá, dolovania údajov z veľkých datasetov. Ide o proces vyhľadávania vzorov v dátach a získavaní znalostí o dátach. A práve klasifikácia patrí k najbežnejším nástrojom pri data mining-u. Vyznačuje sa totiž schopnosťou hľadania vzorcov a vzťahov pri riešení problému, ktoré sa následne dajú využiť na riešenie nových, či podobných problémov. Ak si vezmeme dataset, obsahujúci informáciou o zaradení prvkov do tried, tak na základe vzťahov medzi prvkami jednej triedy môžeme sledovať koreláciu vlastností jednotlivých prvkov a ich vplyv na zatriedenie prvkov. Na základe znalostí o týchto vlastnostiach môžeme vytvárať pravidlové systémy pre použitie pri dolovaní dát.

Väčšina metód dolovania dát, ako zhluková analýza, sa používa zvyčajne v kombinácii s inými nástrojmi. Klasifikácia je jedinečná tým, že je možné ju použiť samostatne na riešenie kompletných klasifikačných problémov, alebo ako súčasť väčšieho systému.

1.4 Využitie klasifikácie

Klasifikácia je užitočným nástrojom marketingu. Umožňuje nám totiž sledovať správanie jednotlivých cieľových skupín zákazníkov. Každý záznam je označeným príznakom – *label*, ktorý reprezentuje názor zákazníka voči produktu, či už pozitívny, alebo negatívny. Príznak môžeme zákazníkovi priradiť na základe jeho predchádzajúceho nákupného správania, alebo na základe príslušnosti k určitej demografickej skupine zákazníkov. Na základe takýchto poznatkov je možné predvídať nákupné správanie

v prípade, že zákazník bude mať záujem o produktu, o ktorom neevidujeme žiadne dáta. Poprípade jediné známe údaje o zákazníkovi sa týkajú jeho príslušnosti k určitej demografickej skupine. Príznaky sa teda priraďujú na základe už existujúcich údajov, v už spomenutej trénovacej množiny.

Klasifikácia je veľmi užitočným nástrojom aj v oblasti zdravotníctva. Umožňuje nám predvídať výskyt chorôb u určitej demografickej skupiny obyvateľstva. Okrem toho slúžia na predvídanie priebehu chorôb u pacientov. Práve preto je presnosť klasifikácie v odvetví zdravotníctva dôležitá.

Klasifikácia umožňuje poskytovanie personalizovaného obsahu na internete. Zatriedenie užívateľa je samozrejme užitočné pre firmy pracujúce pre marketingové spoločnosti, ktoré užívateľom poskytnú reklamu na produkty, o ktoré by mohli mať záujem. Profilovanie návštevníkov používajú aj spravodajské spoločnosti, blogy a sociálne siete, ktoré sa snažia užívateľom poskytnúť čo najrelevantnejší obsah. Okrem poskytovania obsahu je klasifikácia prospešná fungovaniu internetu aj tak, že umožňuje crawler-om webových vyhľadávačov triediť a katalogizovať stránky.

Sociálne siete sú plné užívateľmi generovaného obsahu, ktorý klasifikácia umožňuje využívať. Jej pomocou je možné užívateľov kategorizovať napríklad na základe ich politickej orientácie a tak identifikovať voličskú základňu. Vďaka klasifikácie je taktiež možné identifikovať krízové situácie dejúce sa v reálnom svete, čo umožňuje autoritám na ne bezodkladne reagovať.

Klasifikačné algoritmy sú súčasťou nástrojov na identifikáciu ľudí, či objektov. To je užitočné najmä v prípade identifikácie páchateľov, alebo pri hľadaní podozrivých a nebezpečných predmetov (Aggarwal 2015).

1.5 Naivný Bayesov klasifikátor

Ďalej sa budeme zaoberať jednotlivými klasifikačnými algoritmami, najmä ich teoretickými základmi. Použiteľnosť týchto algoritmov závisí priamo od klasifikačného problému. Iné algoritmy sú vhodné, keď atribúty, na základe ktorých triedime objekt sú kategorické, iné v prípade, ak sú atribúty spojité. Samozrejme, na výber vhodného algoritmu má vplyv aj veľkosť skúmaných dát, počet vlastností a množstvo rôznych hodnôt, ktoré atribúty nadobúdajú.

Naivný Bayesov klasifikátor nezaraďuje objekty na základe pravidiel, ale využíva časť matematiky zvanú Teória pravdepodobnosti. Z toho vyplýva, že objekty sú priraďované

do tried, do ktorých patria s najväčšou pravdepodobnosťou. Pravdepodobnosť, že nastane určitý jav môžeme vyjadriť číselne, na intervale [0, 1], kedy 0 znamená, že daná udalosť nemôže nastať a 1 vyjadruje, že udalosť určite nastane. Ak sme pri určitom jave pozorovali, že nastal 0,7 krát pri N pozorovaniach, môžeme predpokladať že v budúcnosti nastane s pravdepodobnosť 0,7. Čím viac meraní by sme brali do úvahy, k tým presnejšiemu odhadu by sme prišli. V zásade ale nesledujeme len jeden možný jav, ale viacero javov *E*, z ktorých môže nastať vždy len jeden. Pravdepodobnosť, že nastane každý jeden uvádzame, ako *P(E)*, a nachádza sa v intervale [0, 1]. V prípade viacerých jedinečných javov sa súčet ich výskytu musí rovnať 1.

Pri klasifikácii je často náročné a vo veľa prípadoch nemožné získať všetky potrebné a dostatočne presné údaje na zatriedenie určitého javu. Preto sa ako podklad pri použití Naivného Bayesovho algoritmu používa špeciálna množina údajov vo forme tabuľky, ktorej riadky tvoria jednotlivé merania, alebo vlastnosti týkajúce sa pozorovaného objektu. Každý riadok je už vopred klasifikovaný. Táto podmnožina sa označuje ako trénovacia množina a každý riadok ako jej inštancia. Každá inštancia pozostáva z hodnôt jednotlivých atribútov a im prislúchajúcej klasifikácie. Trénovacia množina nám pomôže predpovedať budúci výsledok klasifikácie ešte nezatriedenej inštancie. Pravdepodobnosť, že nastane určitý jav, ak poznáme jednu z hodnôt atribútov, označujeme ako podmienenú pravdepodobnosť.

Na základe pozorovaní predpovedí počasia vieme povedať, že v 50 percentách prípadov pršalo, ak bolo zamračené. Môžeme teda povedať, že trieda je *dážď* za predpokladu, že počasie je *zamračené* a môžeme to zapísať ako *P*(trieda = dážď | počasie = zamračené) = 0,5. V prípade viacerých sledovaných atribútov ako teplota, vlhkosť a tlak môžeme budúci výsledok predpovedať ako *P*(trieda = dážď| počasie = zamračené and teplota = nízka and vlhkosť = vysoká and tlak = nízky).

V trénovacích datasetoch sa nie vždy nachádzajú, alebo sa vyskytujú len v malom počte, inštancie s rovnakými hodnotami atribútov ako tie, ktoré potrebujeme klasifikovať. Aby sme dostali presnejší odhad, môžeme sa zamerať na jediný atribút. Pri takomto prístupe však môžeme prísť k inej klasifikácie pri každom atribúte. Naivný Bayesov algoritmus poskytuje možnosť zohľadniť viaceré pravdepodobnosti.

Slovo naivný v názve odkazuje na predpoklad, že vplyv každého z atribútov na výslednú klasifikáciu je nezávislý. Teda neexistuje korelácia medzi tým, že je zamračené a je chladno. Takýto prístup je naivný, lebo v reálnom svete len zriedka nájdeme dáta, pri ktorých by hodnoty jednotlivých atribútov spolu nesúviseli. Tento problém sa dá riešiť

odstránením niektorých navzájom závislých atribútov. Napriek tejto teoretickej nevýhode vie byť Bayesov algoritmus spoľahlivý a dosahovať podobné výsledky ako výpočtovo omnoho náročnejšie algoritmy, akými sú napríklad neurónové siete a support vector machines.

1.5.1 Matematický zápis Bayesovho Naivného algoritmu

Matematickým základom Bayesovho algoritmu je Bayesov teorém. Teorém pojednáva o zmene názoru vo svetle nových faktov. Vezmime si hypotézu o deji alebo stave reálneho sveta. Pridelíme jej subjektívnu pravdepodobnosť, takzvanú pravdepodobnosť *a priori*, pred jej testovaním. Po zistení nových faktov prepočítame predchádzajúcu pravdepodobnosť a získame novú pravdepodobnosť, takzvanú pravdepodobnosť *a posteriori*. Môžeme povedať, že nová pravdepodobnosť je výsledkom predchádzajúcej pravdepodobnosti správnosti hypotézy a podmienenej pravdepodobnosti vychádzajúcej z novonadobudnutých faktov vydelenej výslednou pravdepodobnosťou (McGrayne, 2012).

Bayesova formula vychádza z tohto teorému a má vzorec

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)},$$

kde $P(A \mid B)$ je podmienená pravdepodobnosť javu A za predpokladu, že nastal jav B, a naopak $P(B \mid A)$ je pravdepodobnosť javu B podmieného výskytom javu A (s P(A) > 0) (Encyklopedia of Mathematics, 2021).

1.5.2 Naivná Bayesova klasifikácia

Ak máme súbor k jedinečných tried c_1, c_2, ... , c_k, ktoré majú pravdepodobnosti a priori $P(c_1)$, $P(c_2)$, ... , $P(c_k)$ a n atribútov a_1, a_2, ... , a_n, ktoré majú pre danú inštanciu hodnoty v_1, v_2, ... , v_n, potom pravdepodobnosť a posteriori výskytu tried c_i pre danú inštanciu je proporcionálna k

$$P(c_i) * P(a_1 = v_1 \; and \; a_2 = v_2 \; and \; a_n = v_n \,|c_i)\,,$$

za predpokladu, že jednotlivé atribúty sú nezávislé, hodnota výrazu je výsledkom súčinu

$$P(c_i) * P(a_1 = v_1 \;|c_i) * P(a_2 = v_2 \;|c_i) * \ldots * P(a_n = v_n \;|c_i)\,,$$

alebo skrátene

$$P(c_i) * \prod_{j=1}^{n} (a_j = v_j \,|class = \; c_i)\,.$$

Súčin musíme vyrátať pre každé i od 1 po k. Následne vyberieme klasifikáciu s najvyššou hodnotou. Vzorec kombinuje pravdepodobnosť a priori c_i s hodnotami n podmienených pravdepodobností. Vo vzorci sa štandardne neuvádza menovateľ ako pri Bayesovom vzorci a to z toho dôvodu, že je pre každý z prvkov totožný.

Bayesov naivný algoritmus sa v posledných rokoch teší veľkej obľube, nie je však bezchybný. Azda najväčším jeho problémom je, že na vykonanie algoritmu musia byť všetky údaje diskrétne. V reálnom svete však datasety často obsahujú nie len diskrétne, ale aj kontinuálne údaje. To v praxi znamená, že dataset môže obsahovať množstvo rôznych hodnôt, ktoré sa v ňom nachádzajú len v malom počte, poprípade sú jedinečné. Možným riešením je tieto kontinuálne premenné vopred zatriediť do intervalov, ktoré sa neprekrývajú. Ďalším nedostatkom Bayesovho algoritmu je, že malá testovacia množina môže skresľovať výpočet relatívnych pravdepodobnosti. V extrémnom prípade sa môže dokonca stať to, že ak sa určitá hodnota v testovacej množine nenachádza, pravdepodobnosť jej výskytu by bola stanovená nule (Bramer, 2020).

Riešením je Laplaceova korekcia, ktorá predstavuje pridanie jedného prvku do všetkých množín, zvyčajne je týmto prvkom 1. Vychádzame pri tom z toho, že naša testovacia množina je tak veľká, že pridanie ďalšieho príkladu ovplyvní pravdepodobnosť len zanedbateľne, pričom sa ale vyhneme nulovej hodnote pravdepodobnosti. Predísť problému s nulovým výskytom sa dá aj tak, že vopred nastavím spodnú hranicu na nízku hodnotu, napríklad 0,0001 (Brédová, 2011).

1.6 Metóda najbližšieho suseda

Azda najprirodzenejším spôsobom, akým môžeme klasifikovať určitý objekt, je na základe podobnosti k iným objektom. Ak máme určitý nezaradený objekt, môžeme ho porovnať s už zaradenými objektami. Porovnáme vlastnosti jednotlivých objektov a nezaradený objekt klasifikujeme do tej istej triedy, ako objekt s podobnými vlastnosťami. V prípade, že nepoznáme význam všetkých vlastností, na základe hodnoty atribútov vieme odhadnúť, do akej miery sú si objekty blízke.

V zásade objekt neporovnávame len s jedným klasifikovaným objektom, ale s viacerými blízkymi objektami. Počet porovnávaných objektov býva zvyčajne malé celé číslo. Vtedy hovorím o metóde klasifikácie k-najbližšieho suseda. Postup algoritmu takejto klasifikácie je jednoduchý. Najprv nájdeme k najbližších objektov k ešte nezatriedenému objektu a priradíme ho do triedy, do ktorej je zaradená väčšina z daných k objektov.

1.6.1 Meranie vzdialenosti

Podobnosť objektov skúmame ako ich vzájomnú vzdialenosť v *n* dimenzionálnom euklidovskom priestore. Počet dimenzií závisí od množstva sledovaných vlastností – každá vlastnosť, atribút, predstavuje jednu dimenziu. Pri meraní vzdialenosť musíme brať do úvahy 3 podmienky:

1) vzdialenosť akéhokoľvek objektu k sebe samému je 0, teda *dist(A, A)* = 0
2) vzdialenosť z bodu *A* do bodu *B* je rovnaká ako z bodu *B* do bodu *A*, teda *dist(A, B) = dist(A, B)*
3) najkratšia vzdialenosť dvoch objektov je vždy priamka.

Pri meraní vzdialenosti sa používa viacero metrík, najbežnejšie však *Euklidova vzdialenosť*. Ak máme dva objekty $A = [a_1, a_2, \dots, a_n]$ a $B = [b_1, b_2, \dots, b_n]$. v *N*-rozmernom priestore, vypočítame vzdialenosť medzi nimi pomocou Pytagorovej vety nasledovne:

$$ED(A,B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}$$

$$ED(A,B) = \sqrt{\sum_{i=1}^{n} \left(a_i - b_j\right)^2}$$

Ďalšou metrikou je *Manhattanova vzdialenosť*, známa aj ako *cityblock* metrika. Oba názvy vychádzajú z podobnosti metriky s mestskými blokmi na Manhattane, kedy nie je možné sa dostať z bodu A do bodu B priamo, ale je možné iba obchádzať bloky a to horizontálne, alebo vertikálne. Metrika vracia sumu absolútnych hodnôt rozdielov každej dimenzie dvoch bodov v *n*-rozmernom priestore. Majme body $A = [a_1, a_2, \dots, a_n]$ a $B = [b_1, b_2, \dots, b_n]$. Manhattanova vzdialenosť – MD, týchto dvoch bodov je určená nasledovne:

$$MD(A,B) = |a_1 - b_1| + |a_2 - b_2| + \cdots + |a_n - b_n|$$

$$MD(A,B) = \sum_{i=1}^{n} |a_i - b_i|$$

Pri Euklidovej vzdialenosti môže dôjsť k stavu, kedy sú vzdialenosti medzi objektami príliš podobné, alebo rovnaké. V takom prípade je vhodná skôr Manhattanova metriky (Bramer, 2020).

V praxi sa používa aj *Kosínusová vzdialenosť*. Tá predstavuje optimálnu metódu merania vzdialenosti v datasetoch, v ktorých sú rozdiely medzi atribútmi veľmi veľké, alebo

množstvo atribútov objektov nadobúda nulové hodnoty. Táto metrika je založená na vytvorení vektorov z počiatočného bodu, v dvojrozmernom priestore *A*(0,0), ku každému objektu a meraní uhlov medzi vektormi jednotlivých objektov. Na základe veľkosti uhlov potom môžeme určiť najbližšieho suseda, pričom neberieme ohľad na vzdialenosť od počiatočného bodu (Layton, 2015).

1.6.2 Normalizácia

Pri meraní vzdialenosti sa často vyskytuje situácia, kedy hodnoty určitého atribútu nadobúdajú neúmerne vysoké hodnoty oproti ostatným atribútom, alebo sú medzi hodnotami jedného atribútu veľké rozdiely. To môže skresľovať výsledky pri meraní podobnosti. Najjednoduchším riešením je použitie alternatívnej jednotky napríklad pri prevode niekoľkých tisícov metrov na kilometre, a podobne.

Efektívnejšie riešenie však vo väčšine prípadov predstavuje *normalizácia* hodnôt spojitých atribútov. Podstata normalizácie tkvie v priradení hodnôt atribútov na intervale [0,1] a to tak, že maximálna známa hodnota atribútu *A* predstavuje 1, najmenšia známa hodnota atribútu *A* potom predstavuje hodnotu 0. Každej ostatnej hodnote *a* atribútu *A* je priradená normalizovaná hodnota ako *(a – min)/(max – min)*. Takýmto spôsobom môžeme prekonvertovať všetky hodnoty atribútu na hodnoty na intervale [0,1]. V praxi môže nastať stav, že hodnota atribútu *A* ešte nezatriedeného objektu nadobúda vyššiu, alebo nižšiu hodnotu ako bolo doterajšie maximum alebo minimum. V takom prípade jej je jednoducho pridelená hodnota 1, ak je hodnota vyššia ako pôvodné maximum, v prípade, že hodnota je menšia ako doterajšie minimum, je jej priradená 0.

Pri meraní podobnosti objektov sa často stretávame so situáciou, kedy určitý atribút považujeme za dôležitejší, než ostatné. V takom prípade môžeme priradiť jednotlivým atribútom váhový koeficient. Po doplnení váhového koeficientu zapíšeme vzorec na výpočet Euklidovskej vzdialenosti ako

$$ED(A,B) = \sqrt{w_1(a_1 - b_1)^2 + w_2(a_2 - b_2)^2 + \cdots + w_n(a_n - b_n)^2}\,,$$

kde $w_1, w_2, \ldots, w_n$ sú váhy jednotlivých atribútov. Súčet jednotlivých váh je vždy 1.

Problém pri určovaní podobnosti objektov predstavujú kategorické atribúty. Jednoduchým riešením, ako eliminovať tento problém, je priradiť objektom, ktorých hodnota sa zhoduje vzdialenosť 0, v opačnom prípade, teda ak sú hodnoty rôzne, vzdialenosť bude 1.

Pri určitých kategorických atribútoch, môžu hodnoty atribútu reprezentovať kardinalitu. Ako príklad môžeme uviesť *dobrý, priemerný, zlý*. V takomto prípade je vhodné priradiť najlepšej hodnote 1, najhoršej 0 a ostatným hodnotám parameter z intervalu [0, 1]. Uvedený príklad by po konverzii vyzeral takto: *dobrý – 1, priemerný – 0,5, zlý – 0* (Bramer, 2020).

Metóda najbližšieho suseda je špecifická v tom, že nevyžaduje použitie trénovacej množiny a tak ju môžeme označiť za lenivé učenie (Layton, 2015).

1.7 Pravidlové systémy a rozhodovacie stromy

Pravidlové systémy sú často používaným a obľúbeným nástrojom data mining-u. Pri ich aplikácii pod pravidlovým systémom rozumieme množinu $R = \{R_1 ... R_m\}$ klasifikačných pravidiel, ktoré majú formu implikácie *if – then (ak – tak potom)*. Zvyknú byť formulované v podobe:

IF *condition* THEN *conclusion.*

Podmienka na ľavej strane výrazu zvyčajne obsahuje operátory $<, \leq, >, =, \subseteq, \in$, ktoré porovnávajú atribúty klasifikovaného prvku. Na pravej strane výrazu je potom trieda, do ktorej bude prvok zaradený (Aggarwal, 2015).

Automaticky generované pravidlové systémy sú vhodnou alternatívou expertných systémov, ktoré sú, v kontraste s vhodne vytvoreným algoritmom, písané manuálne expertmi z určitej oblasti. Nevýhodou ručného písania pravidiel je časová náročnosť. Pomocou algoritmov je možné tvoriť pravidlový systém mnohonásobne rýchlejšie než konvenčnými metódami (Bramer, 2020).

Súbor pravidiel pre klasifikáciu je generovaný pri trénovacej fáze z trénovacej množiny. Po vytvorení pravidlového systému je dobré otestovať daný pravidlový systém klasifikovaním testovacej množiny. Pri testovaní pozorujeme, ktoré pravidlá sa spúšťajú. Pod spusteným pravidlom rozumieme také pravidlo, ktorého podmienka je splnená záznamom z testovacej množiny. Problémová situácia nastane, ak jeden záznam z testovacej množiny spustí viacero pravidiel, ktoré by ho zaradili do rozdielnych tried. Takýto konflikt je treba riešiť tak, že manuálne nastavíme podmienky aby boli navzájom exkluzívne. Teda, každý záznam z množiny spustí vždy len jednu podmienku. Prílišná úprava pravidlového systému môže viesť k javu nazývanému *overfitting*. To znamená, že model je presne napasovaný na trénovaciu množinu, ale ak naň použijeme reálne dáta, nemusí ich správne klasifikovať.

Druhou možnosťou je vytvorenie pravidiel pre všetky možné inštancie dát. Teda každý možný prvok spustí nejaké pravidlo. Najjednoduchšie je v takomto prípade vytvoriť jedno pravidlo, ktoré by bolo schopné pokryť všetky možné inštancie údajov, ktoré ešte nepokrývajú žiadne pravidlá. Rozhodovacie stromy aplikujú obe spomenuté riešenia (Aggarwal, 2015).

1.7.1 Rozhodovací strom

Vhodnou úpravou je možné pravidlový systém, vo väčšine prípadov, transformovať do podoby rozhodovacieho stromu. Nie nepodobne, ako pri pravidlových systémoch, triedime objekty sledovanej množiny na základe hodnôt ich atribútov. Pri rozhodovacích stromoch aplikujeme podmienky na hodnoty atribútov objektu a na základe výslednej pravdivostnej hodnoty ich „štepíme“. Proces štepenia opakujeme dovtedy, kým nemôžeme objekt spoľahlivo klasifikovať (Bramer, 2020).

Rozhodovací strom si môžeme predstaviť ako graf pozostávajúci z uzlov. Na začiatku je jeden koreňový (nelistový) uzol, ktorý sa ďalej rozvetvuje. V každom uzle prebieha vyššie spomenuté štepenie na základe jednej z vlastností objektu (Layton, 2015). Môžeme povedať, že výsledný strom je zovšeobecnením pôvodného datasetu. Rozhodovacie stromy majú okrem klasifikácie aj funkciu kompresie. To znamená, že pomocou rozhodovacieho stromu môžeme dataset reprezentovať v kompaktnej forme (Bramer, 2020).

Rozhodovací strom tvoríme z trénovacej množiny dychtivým učením. Pre tvorbu rozhodovacích stromov existuje množstvo algoritmov, s niektorými sa oboznámime v neskôr. Väčšina z nich je iteratívnych: nájdu najvhodnejší atribút na základe ktorého sa sledovaná množina bude štepiť v koreňovom uzle, potom postupne prechádzajú uzly a hľadajú ďalší najlepší atribút na štepenie. Tento proces je zastavený až v momente, keď ďalšie štepenie postráda zmysel (Layton, 2015).

1.7.2 Výber vhodného atribútu

Ako už bolo vyššie spomenuté, tvorba každého rozhodovacieho stromu a pravidlového systému začína výberom vhodného kritéria – atribútu. Objekty reálneho sveta majú množstvo vlastností, pričom nie každá zohráva pri klasifikácii rovnakú rolu. Nepodstatné vlastnosti často znižujú presnosť klasifikácie a ich spracovávanie má za následok vyššiu výpočtovú zložitosť. Nepodstatné atribúty sa vyskytujú najmä v rozsiahlych, mnohorozmerných datasetoch. Tie sa stali obľúbenými hlavne kvôli relatívnej jednoduchosti hľadania súvislostí medzi údajmi aj bez potreby sledovania historických dát.

Obsahujú však dimenzie, ktoré výsledok klasifikácie dokážu do veľkej miery skresľovať (Dorado, 2019). Preto je veľmi dôležité sa sústrediť na tie vlastnosti, ktoré majú najväčší vplyv na zaraďovanie do tried. Pre nájdenie optimálnych vlastností pre štepenie sa používajú tri základné modely (Aggarwal. 2015).

Najčastejšie využívané algoritmy na tvorbu rozhodovacieho stromu, ako ID3 a C4.5 využívajú na zvolenie atribútu metódy informačného zisku atribútov. Dané algoritmy používajú na kalkuláciu informačného zisku *kritérium entropie* a *gini index* (Dorado, 2019).

1.7.2.1 Entropia

Entropia vychádza z Shannonovej komunikačnej a informačnej teórie a vyjadruje *neurčitosť* informácie, alebo *aké množstvo informácie* obsahuje správa. V prípade rozhodovacieho stromu entropia vyjadruje neurčitosť, s ktorou je v každom uzle prvok zatriedený do triedy. Úlohou algoritmu je teda minimalizovať túto neurčitosť. Medzi neurčitosť výsledku a heterogenitou, teda rôznorodosťou výsledku klasifikácie, existuje závislosť – čím väčšia heterogenita, tým väčšia neurčitosť Závislosť medzi heterogenitou a neurčitosťou sa vo všeobecnosti vyjadruje ako

$$E(v_i) = -\sum_{j=1}^{k} p_j \, log_2 \, p_j \, ,$$

kde kde p_j vyjadruje príslušnosť sledovanej vzorky do *j*-tej z *k* tried ktoré obsahujú atribút v_i. (Gray, 2013). Celková entropia určitého atribútu sa vypočíta ako vážený aritmetický priemer *r* hodnôt, ktoré atribút nadobúda

$$E = \frac{\sum_{i=1}^{r} n_i \, E(v_i)}{n} \, ,$$

kde n_i je počet výskytov, kedy atribút nadobúda hodnotu v_i a *n* je množstvo rôznych hodnôt, ktoré nadobúda atribút v_i. Teda platí že $\sum_{i=1}^{r} n_i = n$ (Aggarwal, 2015).

1.7.2.2 Gini index

Gini index sa používa na meranie schopnosti rozdeliť sledovanú množinu určitou vlastnosťou. Túto schopnosť atribútov $v_1 \ldots v_r$ vypočítame ako

$$Gini(v_i) = 1 - \sum_{j=1}^{k} p_j^2 \, ,$$

kde p_j vyjadruje príslušnosť sledovanej vzorky do *j*-tej z *k* tried ktoré obsahujú atribút v_i. Súčet všetkých $\sum_{j=1}^{k} p_j = 1$. Ak sú prvky sledovanej množiny rovnomerne zatriedené do tried, hodnota gini indexu je 1 - 1/*k*. Gini index môžeme označiť aj ako mieru variancie príslušnosti prvkov sledovanej množiny do tried. Ak sú totiž všetky prvky zatriedené do jedinej triedy, hodnota gini indexu je 0.

Pre každý atribút sa následne vyráta vážený aritmetický priemer pre hodnoty atribútu ako

$$Gini = \sum_{i=1}^{r} \frac{n_i G(v_i)}{n},$$

kde n_i je počet výskytov, kedy atribút nadobúda hodnotu v_i a *n* je množstvo rôznych hodnôt, ktoré nadobúda atribút v_i. Pre určenie schopnosti rozdeliť sledovanú množinu platí, že čím je hodnota gini indexu nižšia, tým je táto schopnosť vyššia (Aggarwal, 2015).

Výhodou gini indexu voči entropii je, že nepočíta logaritmy. Nevýhodou gini indexu je, že je vhodný len na tvorbu binárnych rozhodovacích stromov, teda stromov, pri ktorých z každého uzla vedú maximálne dve vetvy.

Pri použití gini indexu je vhodné podniknúť aj na takzvané orezávanie stromu. Ide o proces zovšeobecňovania stromu, tak aby sa predišlo vyššie spomínanému pretrénovaniu modelu – overfitting. Na to môžeme použiť ďalej spomínaný wrapper (Aggarwal, 2015).

Metóda *Wrapper* sa zameriava na nájdenie takej množiny atribútov, tak aby sa zvýšila výkonnosť klasifikačného algoritmu a vyradili sa prebytočné a nepodstatné atribúty. Wrapper funguje na iteračnom princípe a skúmajú kvalitu, teda relevantnosť jednotlivých atribútov pre klasifikáciu. Klasifikátor opakovane vyberie množinu vlastností a následne skúma výsledok klasifikácie, pričom smerodajným merítkom je obvykle presnosť klasifikácie. Nevýhodou takéhoto prístupu je výpočtová a časová náročnosť. S výpočtovou náročnosťou je úzko spojená cena, ktorá rastie exponenciálne. Podobne výpočtová náročnosť rastie s množstvom vlastností objektov. Problém výpočtovej náročnosti sa dá do istej miery eliminovať použitím vhodného algoritmu, ktorý vie do istej miery predvídať výber vhodných atribútov. Jedným z týchto algoritmov je algoritmus *forward*, ktorý začne s prázdnou množinou atribútov, ku ktorej iteratívne pridáva atribúty tak, aby zvyšovali výkonnosť klasifikátora až kým nenarazí na žiadny ďalší atribút, ktorý ju dokáže zvýšiť. Algoritmus forward je však stále relatívne náročný. Okrem iného nedisponuje schopnosťou spätne identifikovať a vyradiť atribúty, ktoré nemajú veľký vplyv na výkonnosť

klasifikátora. Ako ďalší algoritmus uvediem *backward*, ktorý začína s celou množinou atribútov a iteratívne uberá atribúty tak, aby sa znižovala výpočtová náročnosť až do bodu, kedy už eliminácia ďalšieho atribútu nemá význam. Algoritmu forward je prakticky nepoužiteľný pri datasetoch obsahujúcich veľké množstvo vlastností prvkov (Dorado, 2019).

Ďalšou skupinou sú takzvané *embedded* modely pre výber atribútov klasifikátora. Tie sa zameriavajú na zvýšenie efektivity klasifikátora tak, aby sa predišlo prílišnému zvyšovaniu výpočtovej náročnosti. Najobvyklejším použitím takéhoto modelu je vytvorenie takého rozhodovacieho stromu, ktorý bude prvýkrát trénovaný na všetkých atribútoch. Výkonnosť takéhoto klasifikátora sa následne analyzuje, aby sa identifikovali atribúty, ktoré majú len malý, alebo žiadny vplyv na klasifikáciu. Tie sú pri opätovnom trénovaní klasifikátora vyradené. Čoraz populárnejšou aplikáciou embedded modelu sú *regularizované modely*. Regularizácia uprednostňuje menej komplexné modely, čím sa okrem iného predchádza k pretrénovaniu modelu (Tang, 2014).

1.7.3 Stromové algoritmy

Pri tvorbe rozhodovacieho stromu môžeme aplikovať rôzne algoritmy. Medzi najznámejšie patrí ID3, C4.5 a jeho nadstavba C5.0 a CART.

Algoritmus ID3 (Iterative DiChaudomiser 3) je algoritmom učenia s učiteľom. Vytvoril ho Ross Quinlan v 80. rokoch. Algoritmus vytvára z trénovacej množiny údajov stromovú štruktúru, ktorá slúži na klasifikáciu ešte nezatriedených dát. Pri tvorbe stromu ID3 využíva informačnú teóriu na nájdenie atribútu, ktorý by mal v danom uzle najvyššiu informačnú hodnotu. Informačný zisk každého atribútu je vypočítaný za pomoci už vyššie spomenutej *entropie*. Algoritmus dychtivo prehľadáva jednotlivé vetvy a zastaví sa až v prípade, keď každý podpriestor obsahuje iba prvky jednej triedy.

ID3 je vhodný na klasifikáciu prvkov s numerickými hodnotami atribútov. ID3 negarantuje optimálny výsledok, môže ním totiž byť aj len lokálne optimum. Modely vytvorené ID3 algoritmom sú náchylné na pretrénovanie, preto sa vo všeobecnosti odporúča tvoriť menšie rozhodovacie stromy. Slabou stránkou ID3 algoritmu sú atribúty s veľkým množstvom hodnôt (Hssina, 2014).

Algoritmus C4.5 je nadstavbou ID3 algoritmu. Vo všeobecnosti je v praxi najpoužívanejším algoritmom pre tvorbu rozhodovacích stromov. Ross Quinlan začal pracovať na algortime C4.5, aby odstránil chyby ID3 algoritmu, predovšetkým problém pri klasifikácii datasetov obsahujúcich atribúty s veľkým množstvom hodnôt. Pri atribútoch

nadobúdajúcich veľké množstvo hodnôt je totiž entropia vždy veľmi nízka. Preto C4.5 nevyberá atribút len na základe entropie, ale jeho normalizovaného informačného zisku – vzájomný rozdiel entropie atribútov. Vďaka tomu sa zvyšuje efektivita modelu aj pri použití nenormalizovaných dát.

C4.5 tiež dokáže pracovať s atribútmi, ktoré majú prázdne hodnoty atribútu a to tak, že hodnotu ich informačného zisku odhadne zo známych údajov. C4.5 po vytvorení modelu ešte raz preskúma strom a odstraňuje uzly, v ktorých štepenie nemá na výslednú klasifikáciu vplyv, alebo majú len malý vplyv. Takto sa dá relatívne jednoducho predísť pretrénovaniu modelu.

Ross Quinlan ďalej pokračoval s úpravou stromových algoritmov a vytvoril C5.0, ako upravenú verziu C4.5. Hlavnou výhodou C5.0 je rýchlosť a efektívnosť. Modely generované pomocou C5.0 dosahujú porovnateľné výsledky s tými, vytvorenými pomocou C4.5, avšak sú fyzicky menšie a jednoduchšie. Okrem toho je vyššia rýchlosť dosiahnutá aj schopnosťou paralelizovať beh algoritmu jeho rozdelením na viacero vlákien.

CART (Classification and Regression Trees) je podobný algoritmu C4.5. Vytvára binárne rozhodovacie stromy za použitia Gini indexu. Na druhej strane C5.0 umožňuje vytvárať uzly s viac než dvomi vetvami. V uzle sa vyberá atribút, ktorého Gini index dosahuje po štepení najnižšiu hodnotu. Podobne ako pri C4.5 sa výsledný model prečisťuje a odstraňujú sa prebytočné vetvy (Hssina, 2014).

1.8 Support Vector Machines

Support Vector Machines (SVMs), alebo inak metóda podporných vektorov, je nástrojom binárnej klasifikácie predovšetkým číselných dát. SVMs sú založené na delení priestoru nadrovinou na dva podpriestory. Táto nadrovina predstavuje hranicu medzi triedami, ktoré nadobúdajú hodnotu $\{-1, +1\}$. Nadrovina by mala deliť priestor tak, že prvky zatriedené do určitej triedy by sa mali nachádzať v priestore vymedzenom nadrovinou ako priestor danej triedy. Deliaca nadrovina, v dvojrozmernom priestore reprezentovaná priamkou, je zostrojená tak, že od prvkov tried má čo najväčší odstup. Úlohou SVMs je teda nájsť práve takú nadrovinu, ktorá bude mať maximálny odstup od prvkov. V dvojrozmernom priestore najväčší odstup predstavuje pruh z oboch strán nadroviny, v ktorom sa nenachádzajú žiadne prvky, respektíve ich je málo.

O priestore hovoríme, že je lineárne separovateľný v prípade, ak existuje taká nadrovina, ktorou je možné rozdeliť priestor tak, že v každom podpriestore sa nachádzajú

prvky len jednej triedy. V praxi sa však len zriedka stáva, že skúmané dáta je možné lineárne separovať.

1.8.1 Lineárne SVM

Ak skúmame dataset, ktorého dáta sú lineárne separovateľné, existuje nekonečne veľa možností na vytvorenie deliacej nadroviny. Cieľom SVMs je v takomto prípade nájdenie najdlhšej možnej kolmice medzi nadrovinou a najbližším prvkom triedy. V dvojrozmernom priestore je vzdialenosť od najbližšieho prvku jednej triedy a nadroviny rovnaká, ako vzdialenosť nadroviny a najbližšieho prvku druhej triedy. Ak tieto body pretneme nadrovinami paralelnými s deliacou nadrovinou, dostaneme *podporné vektory.* Priestor medzi týmito vektormi predstavuje odstup. Takéto delenie priestoru zaručuje najpresnejšiu klasifikáciu prvkov, ktorých začlenenie do triedy nám nie je známe.

V prípade, že dáta sú lineárne separovateľné, môžeme použiť najjednoduchší, takzvaný *lineárny klasifikátor*. Majme trénovaciu množinu D pozostávajúcu z n usporiadaných dvojíc $(\vec{x_i}, y_i)$ kde $\vec{x_i}$ je d-rozmerný vektor i-teho prvku a y_i je informácia o zaradení i-teho prvku do triedy, $y_i \in \{-1, +1\}$. Deliacu nadrovinu môžeme definovať ako rovnicu

$$\vec{w} * \vec{x} + b = 0 ,$$

kde $\vec{w} = (w_1 \dots w_d)$ je d-rozmerný vektor reprezentujúci normálnu rovinu a b je skalár, ktorý udáva vzdialenosť od počiatočného bodu. Vektor $\vec{w}$ určuje orientáciu nadroviny. Úlohou klasifikátora je nájdene najväčšieho možného odstupu $d+$ a $d-$ od deliacej nadroviny k najbližším prvkom tried. Tieto body pretínajú pomocné vektory. Zmenou, alebo odstránením týchto prvkov by sa zmenilo aj postavenie podporných vektorov. Podporné vektory rozdeľujú priestor na tri regióny. Keďže je priestor lineárne separovateľný, môžeme predpokladať, že žiaden prvok neleží v priestore medzi podpornými vektormi a zároveň všetky prvky ležia v jednom z dvoch zvyšných (extrémnych) regiónov. To je možné zapísať ako

$$\vec{w} * \vec{x_i} + b = +1 \; \forall \; y_i = +1$$

$$\vec{w} * \vec{x_i} + b = -1 \; \forall \; y_i = -1.$$

Pre všetky $\vec{x_i}$ patriace do triedy $y_i = +1$ teda platí

$$\vec{w} * \vec{x_i} + b \geq +1$$

a pre všetky prvky patriace do y_i = -1 platí

$$\vec{w} * \vec{x_i} + b \leq -1.$$

To je možné transformovať do jednej nerovnice

$$y_i(\vec{w} * \vec{x_i} + b) \geq +1 \; \forall \; i.$$

Vzdialenosť regiónov obsahujúcich len prvky jednej triedy od deliacej nadroviny je $1/||w\vec{}||$, kde $||w\vec{}||$ je normalizovaný vektor v Euklidovskom priestore a $|b|/||w\vec{}||$ predstavuje vzdialenosť nadroviny od počiatočného bodu. Keďže rozdiel medzi dvomi nadrovinami oddelenými pomocnými vektormi je 2, vzdialenosť, odstup, medzi týmito nadrovinami predstavuje $2/||w\vec{}||$. Úlohou SVMs klasifikátora je maximalizovať tento odstup. Zápis $2/||w\vec{}||$ nie je optimálny. Preto sa v praxi pre riešenie problému používa minimalizácia $||w\vec{}||^2/2$, čo je to isté ako maximalizácia $2/||w\vec{}||$. Pre riešenie takejto výpočtovo náročnej úlohy nelineárneho programovania s nelineárnymi obmedzeniami sa používa metóda *Lagrangeovho multiplikátora*. Podstatou tejto metódy je zaviesť *n*-rozmerovú množinu nezáporných Lagrangeových multiplikátorov $\lambda\vec{} = (\lambda_1 \ldots \lambda_n) \geq 0$ pre každé nelineárne obmedzenie. Obmedzenia sú zrelaxované použitím Lagrangeovej penalizácie

$$L_P = \frac{||\vec{w}||2}{2} - \sum_{i=1}^{n} \lambda_i [y_i(\vec{w} * \vec{x_i} + b) - 1] \, ,$$

pričom L_P sa minimalizuje vzhľadom ku $w\vec{}$ a b. Ide o konvexný problém nelineárneho programovania ako duálnu úlohu maximalizácie L_D. Aby sme zaručili, že gradient L_P vzhľadom na $w\vec{}$ a b bol nulový, musí platiť

$$\vec{w} - \sum_{i=1}^{n} \lambda_i \, y_i \, \vec{x_i} = 0 \, .$$

Po dosadení do Lagrangeovej funkcie získame duálnu funkciu

$$L_D = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \, \lambda_j \, y_i \, y_j \, \vec{x}_i \, \vec{x}_j \, .$$

Úlohou SVMs je potom maximalizácia duálnej úlohy L_D vzhľadom na obmedzenie $\lambda_i \geq 0$ a $\sum_{i=1}^{n} \lambda_i \, y_i = 0$ (Aggarwal, 2015).

Pri riešení duálnej úlohy v lineárne separovateľnom priestore, je možné použiť funkciu podobnosti. Táto funkcia umožňuje riešenie úlohy na základe podobnosti medzi

prvkami aj bez toho, aby sme poznali numerické hodnoty jednotlivých vlastností prvkov. Mapovaním vieme získať transformovaný vektor φ ($x\vec{}$). Mapovanie so sebou však prináša výpočtovú náročnosť. Nám postačuje poznať vzťah φ ($x_i\vec{}$)* φ ($x_j\vec{}$) = k($x_i\vec{}$, $x_j\vec{}$). Tak získame jadrovú funkciu k($x_i\vec{}$, $x_j\vec{}$), ktorú je možné použiť pri SVMs klasifikácii na pôvodných hodnotách prvkov ako:

$$L_D = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \, \mathrm{k}\left(\overrightarrow{x_i}, \overrightarrow{x_j}\right).$$

Ako už bolo spomenuté, datasety, ktoré by boli lineárne separovateľné sú v praxi veľmi ojedinelé. Takéto datasety je však možné aspoň približne separovať tak, že väčšina prvkov sa nachádza v správnom podpriestore. Ostatné, ktoré sa nachádzajú na nesprávnej strane podporných vektorov sú penalizované (Bishop, 2006).

1.8.2 Nelineárne SVM

V prípade, že dataset nie je lineárne separovateľný môžem použiť takzvaný *jadrový trik,* v literatúre označovaný aj ako *jadrová substitúcia* alebo *transformácia.* Pomocou tejto metódy presuniem prvky do priestoru, ktorý je lineárne separovateľný. Takto umožňuje transformovať lineárne neseparovateľnú úlohu na úlohu lineárne separovateľnú. Transformácia spočíva v pridávaní atribútov, pričom každý predstavuje novú dimenziu. Transformácia prebieha pomocou zmeny jadra, ktoré má pri použití lineárnej klasifikácie k($x_i\vec{}$, $x_j\vec{}$) (Bishop, 2006).

Medzi funkcie používané pri nelineárnych SVMs patrí:

- funkcia s polynómom *d*-teho stupňa k($\gamma\, x_i\vec{}$, $x_j\vec{}$ + r)^d
- Gaussova radiálna bázová funkcia exp(-γ|| $x_i\vec{}$ - $x_j\vec{}$||^2)
- Dvojvrstvová neurónová sieť sigmoid tanh(γ|| $x_i\vec{}$ - $x_j\vec{}$|| + r)

1.8.3 Nebinárna klasifikácia

SVMs sú vytvorené na klasifikáciu dát, ktoré sú triedené do dvoch tried, ktoré nadobúdajú hodnotu {−1, +1}. Existujú však určité metódy, ktoré nám umožňujú použiť SVMs na klasifikáciu do *k* viac než dvoch tried. Prvou je metóda *jedna-proti-ostatným* kedy je klasifikácia do *k* rôznych tried transformovaná na *k* binárnych klasifikačných modelov. Každý prvok je klasifikovaný každým modelom a to tak, že v každom modeli je klasifikovaný, či do danej triedy patrí, alebo nepatrí. Ak do danej triedy patrí, dostane pre

túto triedu bod, ak nie, bod dostane pre všetky ostatné triedy. Na konci je každý prvok začlenený do triedy, ktorá má najviac bodov. Aby sa predišlo tomu, že prvok je zatriedený do viacerých tried s rovnakým počtom bodov, je možné použiť pri bodovaní do tried hodnotu reprezentujúca príslušnosť prvku do danej triedy a to napríklad za pomoci Bayesovho naivného algoritmu. V takom prípade je prvok klasifikovaný do triedy, ku ktorej má najvyššiu príslušnosť. Ďalším možným prístupom je *jeden-proti-jednému*, kedy je model rozdelený na $k(k\text{-}1)/2$ modelov a rovnaký počet trénovacích dát. Každý model predstavuje binárnu klasifikáciu. Pre každý model je potom vybratá víťazná trieda. Ktorá metóda je efektívnejšia závisí od štruktúry dát. Napriek tomu, že k klasifikačných modelov je menej než $k(k\text{-}1)/2$, výpočtová náročnosť je redukovaná malou veľkosť trénovacích množín, ktoré predstavujú približne $2/k$ pôvodnej trénovacej množiny.

SVMs umožňujú klasifikovať aj kategorické hodnoty pomocou metódy *binarizácia*. Binarizácia je proces transformácie kategorického atribúty, ktorý nadobúda n rôznych hodnôt na n atribútov. Záznamu je potom v transformovanom atribúte priradená hodnota 1 v prípade, že mal v pôvodnom atribúte danú hodnotu. V opačnom prípade je hodnota atribútu 0 (Aggarwal, 2015).

1.9 Metriky hodnotenia kvality klasifikácie

Pri klasifikovaní určitého datasetu môžeme často použiť viacero klasifikátorov, pričom každý môže podať iný výkon, v závislosti od dát. Na základe čoho ale vybrať najvhodnejší dataset? Azda najprirodzenejšia metrika, na základe ktorej by sme vybrali klasifikátor, je *presnosť klasifikácie*. Pod presnosťou klasifikácie rozumieme pomer správne klasifikovaných prvkov k množine neznámych prvkov. Presnosť je vo všeobecnosti najdôležitejšie merítko pri výbere klasifikátora. Avšak v praxi je nutné zhodnotiť aj zložitosť algoritmu, efektívnosť, s akou klasifikátor využíva zdroje systému a pochopiteľnosť výsledného modelu.

Pri veľkom množstve údajov nie je možné stanoviť presnosť modelu, je možné ju len odhadnúť, na základe údajov ktoré neboli použité pri jeho tvorbe. Najjednoduchší prístup k meraniu presnosti klasifikátora je *rozdelenie datasetu na trénovaciu a testovaciu množinu*. Trénovacia množina, ako už bolo spomenuté slúži na vytvorenie klasifikátora. Následne potom klasifikátor triedi prvky testovacej množiny. Presnosť klasifikátora p vypočítame ako $p = C/N$, kde N je počet prvkov trénovacej množiny a C je počet správne zatriedených prvkov. V praxi sa delia údaje v rôznych pomeroch v závislosti od problému, pričom štandardne v pomeroch 1:1, 2:1, 70:30, alebo 60:40. Bežne sa stáva, že ak odhadneme

presnosť p na určitej množine údajov, získame novú množinu údajov a tie klasifikujeme, dostaneme odlišné p. Presnú hodnotu presnosti klasifikátora dostaneme len vtedy, keď ho otestujeme na všetkých možných inštanciách, čo nie je veľmi realistické. Ak by sme chceli byť presnejší, môžeme použiť smerodajnú odchýlku, ktorú zapíšeme ako $S = \sqrt{\frac{p(1-p)}{N}}$.

V prípade že počet prvkov N je nízky (ale nie len vtedy), môžeme presnosť modelu určiť vzájomnou validáciou. V takomto prípade rozdelíme dataset na k podmnožín. Klasifikátor natrénujem k-krát na k-1 podmnožinách a otestujem jeho presnosť na k-tej podmnožine. Následne vydelíme celkový počet C správne zatriedených prvkov celkovým množstvom prvkov N a tak získame odhad presnosti p.

Pri klasifikácii často triedime datasety do dvoch tried, kedy jednu triedu, ktorá nás obzvlášť zaujíma, môžeme označiť ako pozitívnu, a druhú ako negatívnu. Keď máme viacero tried, ale zaujíma nás začlenenie prvkov do jednej triedy, tak tú označíme ako pozitívnu a zvyšné triedy ako negatívne. Ak klasifikátor zatriedi prvky ako pozitívne, aj keď by mali byť v realite negatívne, hovoríme o falošnej pozitivite. Podobne, ak pozitívny prvok je zatriedený ako negatívny, hovoríme o falošnej negativite. V praxi môže mať falošná pozitivita oveľa väčší dopad ako falošná negativita. Presnosť klasifikačného modelu môžeme hodnotiť aj na základe pomeru skutočne pozitívnych prípadov, ktoré boli klasifikované ako pozitívne (Bramer, 2020).

2. Cieľ práce, metodika práce a metódy skúmania

Primárnym cieľom záverečnej práce je preskúmať strojové učenie v úlohách klasifikácie a vytvoriť kód v Pythone pre učenie klasifikovania s dohľadom.

Čiastkové ciele tejto práce sú opísanie klasifikácie a jej využitia v praxi, opísanie klasifikačných algoritmov ako napríklad Bayesov naivný algoritmus, metódu najbližšieho suseda, algoritmy na tvorbu pravidlových systémov a rozhodovacích stromov a metódu SMVs a následná aplikácia niektorých z týchto klasifikátorov.

Objektom skúmania je klasifikácia ako nástroj strojového učenia, algoritmy používané ku klasifikácii, proces tvorby klasifikačných modelov v jazyku Python a nástroje pre vizualizáciu klasifikačných modelov.

Spracované údaje v záverečnej práci sú získané z literatúry týkajúcej sa klasifikácie, klasifikačných algoritmov, matematickej teórie, informačnej teórie, data mining-u, programovacieho jazyku Python. Vychádzali sme z kníh a vedeckých článkov v anglickom jazyku. Všetky zdroje sú uvedené v zozname použitej literatúry. Za pomoci metódy selekcie boli vybrané potrebné a dôležité údaje z literatúry.

Výkonnosť použitých klasifikátorov bola meraná v percentách. Získane závery ohľadom výkonu klasifikačných modelov sme získali experimentom a dedukciou. Metódou porovnania sme zhodnotili využiteľnosť modelov pri skúmaní datasetov.

Metóda analýzy bola aplikovaná pri skúmaní súčasného stavu problematiky, klasifikačných modelov, výbere vhodných atribútov datasetu,

Indukciou sme prišli ku zovšeobecneniu aspektov jednotlivých algoritmov, ako aj modelov.

V záverečnej práci využívame na demonštráciu aplikácie klasifikácie v jazyku Python, nástroje projektu Scikit-Learn. Scikit-Learn zahŕňa širokú škálu algoritmov strojového učenia. Hlavným dôvodom je jednoduchá aplikácia týchto algoritmov, čo umožňuje rýchlu tvorbu klasifikačných modelov a bezproblémovú experimentáciu. Kódy, knižnice a dokumentácia sú voľne dostupné, na podnet autora tohto projektu uvádzam ako zdroj publikáciu *Scikit-learn: Machine Learning in Python*.

Pri práci používame komunitnú verziu vývojového prostredia *PyCharm* od spoločnosti JetBrain a vstavanú konzolu. Kód je napísaný vo verzii Python-u 3.9.10. Python je vysokoúrovňový, interpretovaný, interaktívny programovací jazyk. Python podporuje

objektovo orientované programovanie a vysokoúrovňové dátové typy. Python má rozsiahlu štandardnú knižnicu, ktorá je voľne dostupná na portáli Python-u, vrátane interpretera (Python Software Foundation, 2022).

3. Výsledky práce

Táto kapitola sa zaoberá praktickým využitím klasifikátorov, konkrétne rozhodovacím stromom, klasifikátorom najbližšieho suseda a klasifikátorom využívajúcim SVMs.

3.1 Použité dáta

Skúmanú množinu sme získali z otvoreného komunitného portálu *data.world*. Na klasifikáciu používame dataset predstavujúci zoznam pasažierov lode RMS Titanic, ktorý obsahuje okrem iného aj údaje o ich veku a pohlaví, triede, v ktorej sa plavili, cene lístka a to, či osudnú plavbu prežili. Na základe vlastností pasažierov budeme skúmať ich vplyv na prežitie pasažierov.

Dataset je vo formáte .csv dokumentu. Ten načítavame pomocou funkcie obsiahnutej v knižnici *pandas*. Pandas je veľmi robustný nastroj pre prácu s dátami. Nám umožňuje pri načítavať údaje a výberať atribúty na základe ich názvov. Niektoré klasifikátory nepodporujú prácu s *pandas dataframe*, preto je potrebná ich transformácia.

Dataset obsahuje určité osobné údaje pasažierov, ktoré sú pre modelovanie nepodstatné. Odstránime ich spolu s riadkami, v ktorých chýbajú údaje. Niektoré klasifikátory umožňujú prácu aj s chýbajúcimi dátami, ale pre naše potreby je jednoduchšie takéto prvky odstrániť. Okrem toho zmeníme atribút pohlavie na binárnu hodnotu, miesto pôvodného reťazca. Nie všetky klasifikátory sú totiž schopné pracovať s kategorickým atribútmi, vrátane klasifikátora *DecisionTreeClassifier*.

```
import pandas as pd

data = pd.read_csv("train.csv")
data.head()

data = data[['Survived', 'Pclass', 'Sex', 'Age', 'Fare']].dropna().copy()
data['Sex'] = data['Sex'].replace(to_replace='male', value=0)
data['Sex'] = data['Sex'].replace(to_replace='female', value=1)
```

Obr. 1 – Načítanie datasetu

Výsledkom je pre klasifikáciu relevantná množina štruktúrovaných údajov. Táto množina postráda záznamy s neúplnými, alebo chýbajúcimi údajmi. Po načítaní môžeme dataset vidieť pomocou metódy *print()*. Pandas nám zobrazuje množiny v kompaktnej podobe.

	Survived	Pclass	Sex	Age	Fare
0	0	3	0	22.0	7.2500
1	1	1	1	38.0	71.2833
2	1	3	1	26.0	7.9250
3	1	1	1	35.0	53.1000
4	0	3	0	35.0	8.0500
..	...	...	...	...	...
885	0	3	1	39.0	29.1250
886	0	2	0	27.0	13.0000
887	1	1	1	19.0	30.0000
889	1	1	0	26.0	30.0000
890	0	3	0	32.0	7.7500

[714 rows x 5 columns]

Obr. 2 – Prvých a posledných 5 riadkov datasetu

3.2 Rozhodovací strom v jazyku Python

Prvým modelom pre klasifikáciu, ktorý skúmame, je rozhodovací strom. Pri zostrojení modelu stromu používame už vyššie spomenutý Scikit-Learn a v ňom zabudované Python-ové knižnice, ako napríklad knižnicu *tree*, obsahujúcu funkcie na tvorbu samotného stromu, *preprocessing* na očistenie dát a prípravu dát, *model_selection* pre rozdelenie množiny a v neposlednej rade *metrics* na zhodnotenie výkonnosti výsledného modelu. Strom zobrazujeme pomocou metód knižnice *matplotlib*. Dáta získavame spôsobom opísaným v predchádzajúcej podkapitole.

V ďalšom kroku skúmané atribúty, triedu (Pclass), pohlavie (Sex), vek (Age) a cenu cestovného (Fare) a výslednú triedu (Survival) roztriedime na dve množiny: trénovaciu – *X* a testovaciu - *y*. Trénovacia množina slúži pri tvorbe modelu a testovacia na jeho validáciu. Na toto rozdelenie slúži funkcia *train_test_split*. Jej vstupnými parametrami sú v tomto prípade atribúty, množina tried, pomer veľkosti testovacej množiny k množine trénovacej a parameter určujúci, ako sa pri delení množiny atribúty premiešajú.

Ako už bolo spomenuté v prvej kapitole, pri delení množiny je možné použiť rôzne pomery (1:1, 2:1, 80:20, 70:30). Pomer veľkostí množín stanovujeme na základe opakovaných pozorovaní a meraní presnosti modelu. Tá bola konzistentne najvyššia, v priemere 80,30 % pri rozdelení datasetu v pomere 60:40. Množina na trénovanie modelu predstavuje 60 % pôvodného datasetu. Zvyšných 40 % slúži na meranie výkonnosti modelu. Na premiešavanie záznamov používame funkciu *random()* tak, aby generovala pseudonáhodné číslo na intervale [1,10]. V opačnom prípade by sme mohli ako parameter metódy použiť konštantu, ktorá by zabezpečila konzistentné výsledky pri opakovanom behu algoritmu.

```
from sklearn.model_selection import train_test_split

atributy = ['Pclass', 'Sex', 'Age', 'Fare']
X = data[atributy]
y = data.Survived

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=random.randrange(1, 10))
```

Obr. 3 – Tvorba trénovacej a testovacej množiny

Keďže dataset obsahuje vlastnosti vyčíslené v rôznych mierkach, je nutné ich normalizovať. V opačnom prípade by rozdiel mierok mohol viesť k modelu, ktorý by prikladal neúmerne veľkú váhu niektorým atribútom. Na normalizáciu slúži funkcia predspracovania dát *MinMaxScaler*, ktorá normalizuje údaje na hodnoty na intervale [0,1] a to tak, že každá hodnota *a* atribútu *A* je vynásobená s *(a – min)/(max – min)*.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Obr. 4 – Normalizácia pomocou MinMaxScaler

Dáta sú predspracované, očistené. Môžeme teda pristúpiť k samotnej klasifikácii. Scikit-Learn je vybavený klasifikátorom *DecisionTreeClassifier.* Klasifikátor používa optimalizovanú verziu CART algoritmu. Tento robustný klasifikátor umožňuje vybrať používateľovi vhodné kritérium pre štepenie, spôsob, na základe ktorého bude model štepiť, maximálnu hĺbku stromu, maximálnu šírku stromu, minimálne množstvo prvokov množinu, pri ktorom zastaví vetvenie, váhu jednotlivých atribútov atď. Pri tvorbe volíme *gini index*, keďže v priemere dosahoval vyššiu presnosť pri aplikácii na daný dataset v porovnaní s *entropiou*. Hĺbku, do ktorej klasifikátor tvorí model limitujeme na 5 uzlov, kvôli prehľadnosti vizualizácie konečného modelu. Váhy jednotlivých atribútov ponecháme rovnocenné. Pomocou metódy *fit* následne vytvárame model.

```
from sklearn.tree import DecisionTreeClassifier

DTC = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=5, class_weight=None)
DTC = DTC.fit(X_train, y_train)
```

Obr. 5 – Tvorba modelu rozhodovacieho stromu

Keď máme model vytvorený, ostáva ho len validovať a zistiť jeho výkonnosť, respektíve presnosť, s akou dokáže predpovedať zaradenie preň ešte neznámeho prvku do triedy. Metóda *predict* slúži na vytvorenie predikcie na základe atribútov testovacej množiny. Výslednú predpoveď následne porovnávame s reálnymi triedami testovacej množiny.

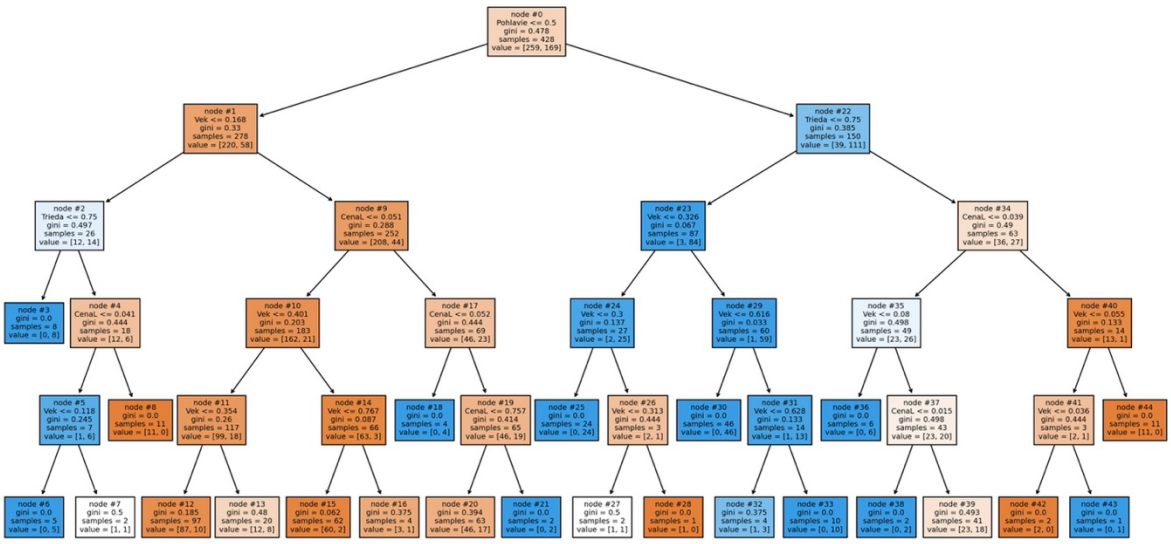
```
from sklearn import metrics

y_pred = DTC.predict(X_test)
print('Presnost modelu: ', 100*round(metrics.accuracy_score(y_test, y_pred), 6), ' %.')
```

Obr. 6 – Zistenie presnosti modelu

Model rozhodovacieho stromu vieme vizualizovať buď v textovej podobe jednoduchým exportom, alebo pomocou nástrojov, ktoré poskytuje knižnica *matplotlib*.

```
from sklearn.tree import export_text
from matplotlib import pyplot as plt
from sklearn import tree

strom = export_text(DTC, feature_names=['Trieda', 'Pohlavie', 'Vek', 'CenaL'])
print(strom)

plt.figure(figsize=(20,10))
tree.plot_tree(DTC, feature_names=['Trieda', 'Pohlavie', 'Vek', 'CenaL'], fontsize=7, filled=True, node_ids=True)
plt.savefig('tree', dpi=200)
plt.show()
```

Obr. 7 – Tvorba diagramu rozhodovacieho stromu



Obr. 8 – Vizualizácia rozhodovacieho stromu

Z vizualizácie je vidieť, že klasifikátor vyhodnotil ako najlepšie atribút na štepenie v koreňovom uzle (uzol číslo 0) atribút *Pohlavie*. Presnosť tohto modelu je 80,7692 %. Rozhodovací strom nám umožňuje presne vidieť, ako sa klasifikátor správa.

3.3 Metóda najbližšieho suseda v Pythone

Ako už bolo spomenuté v prvej kapitole, metóda najbližšieho suseda funguje na princípe klasifikovania podobných objektov do rovnakých tried. Pri riešení používame rovnaký dataset ako pri tvorbe rozhodovacieho stromu s tým rozdielom, že na klasifikáciu využívame vždy len dve vlastnosti – *Vek* a *Cenu lístka*. Dôvodom je zložitosť zobrazovania

klasifikácie pri použití viac ako dvoch atribútov. Každý atribút totiž predstavuje jednu dimenziu. Pre prehľadnosť pracujeme v dvojrozmernom euklidovskom priestore.

```
import pandas as pd
import random
import numpy as np
from sklearn import metrics
from sklearn.model_selection import train_test_split

data = pd.read_csv("train.csv")
data.head()
data = data[['Survived', 'Pclass', 'Sex', 'Age', 'Fare']].dropna().copy()

data['Sex'] = data['Sex'].replace(to_replace='male', value=0)
data['Sex'] = data['Sex'].replace(to_replace='female', value=1)

atributy = ['Age', 'Fare']
X = data[atributy]
y = data.Survived

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=random.randrange(1, 10))

X_train = X_train.to_numpy()
X_test = X_test.to_numpy()
y_train = y_train.to_numpy()
y_test = y_test.to_numpy()
```

Obr. 9 – Načítanie údajov a ich úprava

Proces načítania dát a delenia datasetu na trénovaciu a testovaciu množinu je takmer zhodný s predchádzajúcim prípadom, rozdiel je v počte atribútov, s ktorými budeme pracovať. Následne konvertujeme trénovaciu a testovaciu podmnožinu z *pandas dataframe* na *numpy array*. Klasifikátor najbližšieho suseda nedokáže pracovať optimálne s dátovými štruktúrami dataframe.

3.3.1 Výber vhodného algoritmu

Klasifikátory metódy najbližšieho suseda – *NearestNeighbors* – umožňujú použiť tri rôzne algoritmy: *Ball tree*, *k-d tree* a algoritmus používajúci „hrubú silu“, tzv. *brute-force algoritmus*. Riešiť problém hrubou silou vo všeobecnosti znamená prehľadať celý priestor možných riešení. Brute-force algoritmus vyhľadá všetky páry prvkov v datasete a meria ich vzájomnú vzdialenosť. Z toho vyplýva, že vie dosahovať veľmi dobré výsledky v malých datasetoch, kde získa veľmi presné informácie o prvkoch datasetu. Avšak s rastúcim počtom prvkov rastie náročnosť exponenciálne, čo robí brute-force algoritmus prakticky nepoužiteľným vo veľkých datasetoch. Náročnosť tohto algoritmu rastie v miere $O[DN^2]$, kde N je počet prvkov skúmanej množiny a D počet dimenzii, pričom každý atribút predstavuje jednu dimenziu. Pri výbere algoritmu by sme ako jeden z parametrov klasifikátora zadali *algorithm = 'brute'*. Pre náš dataset je nevhodné použiť hrubú silu, keďže

ide o veľkú množinu aj napriek tomu, že pracujeme len s dvojrozmerným euklidovským priestorom, teda používame len dva atribúty.

Aby sme sa vyhli problému vysokej výpočtovej náročnosti pri použití hrubej sily, môžeme použiť jeden z algoritmov využívajúcich stromové štruktúry. Takéto algoritmy znižujú množstvo potrebných kalkulácii vzdialenosti. Vychádzajú z jednoduchého predpokladu, že ak medzi prvkami *A* a *B* existuje veľká vzdialenosť, a medzi *B* a *C* je len malá vzdialenosť, tak viem, že medzi prvkami *A* a *C* je tiež veľká vzdialenosť bez toho, aby sme ju museli vyrátať. Použitie znižuje výpočtovú náročnosť na úroveň logaritmickej náročnosti *O[DN log N]*.

Jedným takýmto algoritmom je *k-d algoritmus*, ktorý vytvára binárny strom. Každý uzol tohto stromu je *k*-dimenzionálny bod, ktorý delí priestor na dva podpriestory, v prípade dvojrozmerného priestoru na dve polroviny. Delenie prebieha vždy na základe jednej dátovej osy. Keďže štepenie v uzloch prebieha vždy len na základe hodnôt jednej dimenzie, algoritmus nemusí nikdy vypočítavať vzdialenosť vo viac rozmernom priestore. To umožňuje veľmi rýchlu tvorbu stromu. Keď je strom vyhotovený, najbližší susedia sú vymedzený s náročnosťou *O[log N]*. K-d algoritmus dosahuje najvyššiu výkonnosť pri málo rozmerných datasetoch. Tento jav sa nazýva ako „kliatba dimenzionality“. Pre jeho aplikáciu sa ako parameter klasifikátora vkladá *algorithm = 'kd_tree'*.

S rastúcim počtom dimenzii sa však k-d strom stáva neefektívnym. Problém s veľkým množstvom dimenzii rieši algoritmus *Ball tree*. Ten vytvára stromovú štruktúru podobne ako k-d strom s tým rozdielom, že v uzloch nedelí priestor na podroviny, ale člení prvky do pretínajúcich sa *n-sfér*. Príslušnosť prvku do každej n-sféry je stanovená od stredu danej n-sféry. Vytváranie takéhoto stromu je vo všeobecnosti náročnejšie, ako tvorba k-d stromu, avšak pri veľkom počte dimenzii ($D > 20$) sa môže Ball tree osvedčiť. Nie je to však pravidlom, na efektivitu Ball tree algoritmu má veľký vplyv štruktúra dát.

Ďalším faktorom, ktorý má vplyv na výkonnosť klasifikátora je *k* počet najbližších susedov. S rastúcim *k* rastie výpočtová zložitosť, najmä pri tvorbe stromových štruktúr. V zásade by sa mal vybrať prístup hrubou silou ak $k > N/2$. Čo sa presnosti klasifikácie týka, všetky 3 algoritmy dosahovali veľmi podobné hodnoty. Vzhľadom na veľký počet prvkov datasetu ($N = 714$) a nízky celkový počet dimenzií ($D < 15$) budem ďalej používať k-d strom.

Pred klasifikáciou je vhodné zvážiť vplyv *váh*. V tomto prípade nehovoríme o vplyve váh jednotlivých vlastností na výsledok klasifikácie, ale na váhu pri hľadaní najbližšieho suseda. Pri uniformných váhach je všetkým prvkom prikladaná rovnaká váha. Druhou

možnosťou je priradiť prvkom proporčne vyššie váhy v závislosti od vzdialenosti od klasifikovaného prvku. Čím sú si prvky bližšie, o to väčšiu váhu vzdialenosti priradíme. Ak nie je špecifikované inak, klasifikátor používa uniformné váhy.

3.3.2 Klasifikátor k-najbližšieho suseda

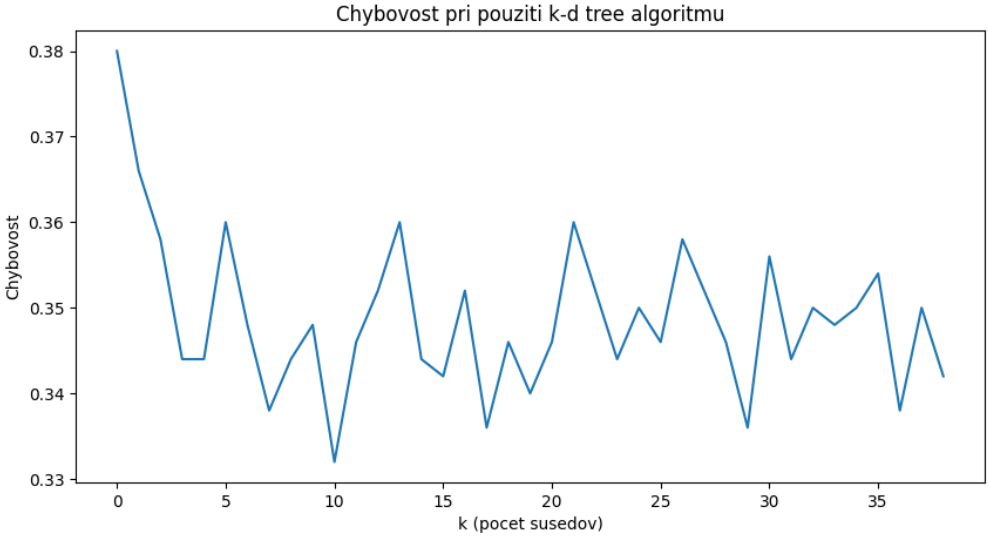
Metóda k-najbližšieho suseda je založená na porovnávaní vlastností objektu s vopred stanoveným *k* počtom najbližších prvkov. Ide o najbežnejšie použitie metódy najbližšieho suseda. Výber vhodného *k* závisí vždy od skúmaných dát. V zásade platí, že čím je vyšší koeficient *k*, tým viac sa eliminuje vplyv hluku na klasifikáciu. Na druhej strane vyššie *k* robí hranice medzi jednotlivými triedami menej zreteľnými. S rastúcim *k* taktiež rastie výpočtová zložitosť, najmä pri tvorbe stromových štruktúr.

Koeficientu *k* vyberáme na základe chybovosti algoritmu. Pomocou jednoduchého kódu klasifikátor *KNeighborsClassifier* púšťame opakovane s tým, že postupne zvyšujeme hodnotu *k* o 1 od 1 po 40. Numpy pole *chybovost* je postupne napĺňané hodnotami korešpondujúcimi priemernému rozdielu predpovedanej a výslednej hodnoty. Čím je tento rozdiel nižší, tým je klasifikátor presnejší.

```
from sklearn.neighbors import KNeighborsClassifier

chybovost = []

for i in np.arange(1, 40):
    KNN = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree')
    KNN.fit(X_train, y_train)
    y_pred = KNN.predict(X_test)
    chybovost.append(np.mean(y_pred != y_test))

fig = plt.figure(figsize=(10,5))
plt.plot(chybovost)
plt.title('Chybovost pri pouziti k-d tree algoritmu')
plt.ylabel('Chybovost')
plt.xlabel('k (pocet susedov)')
fig.savefig('chybovost.png')
plt.show()
```

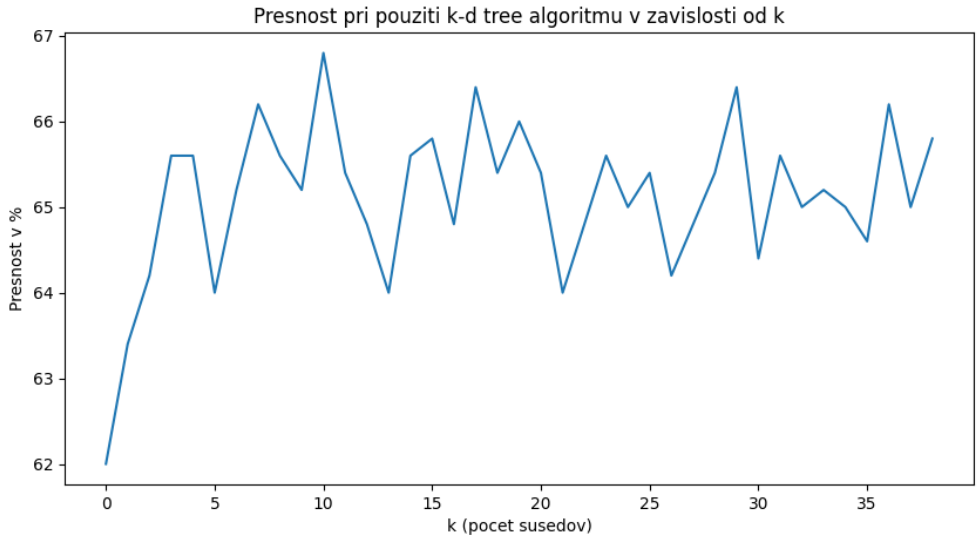
Obr. 10 – Zistenie vhodného k pri metóde k-najbližšieho suseda

Pomocou *matplotlib* následne rozdiely predikcie vizualizujeme. Na základe vytvoreného grafu môžeme vidieť, že klasifikátor poskytuje najpresnejšie výsledky, pri *k* rovnajúce sa približne 10. To znamená, že ak použijem pri klasifikácii $k = 10$, mali by som dosiahnuť najlepšie výsledky pri relatívne nízkom *k* v pomere ku *N* celkovému počtu prvkov.



Obr.11 – Chybovosť klasifikátora v závislosti od k

Podobne môžeme hodnotiť presnosť algoritmu vo vzťahu ku *k*. Na výslednom diagrame vidíme, že klasifikátor dosahuje najvyššiu presnosť predpovedí pri $k \cong 10$.



Obr.12 – Presnosť klasifikátora v závislosti od k

Po definovaní optimálneho *k* nám ostáva stanoviť najvhodnejšiu metriku vzdialenosti. Vyberáme medzi Euklidovou a Manhattanovou metrikou vzdialenosti. Tie sú bližšie opísané v 1. kapitole.

```
from sklearn.neighbors import KNeighborsClassifier

k_neighbors = 10

KNN=KNeighborsClassifier(n_neighbors=k_neighbors, algorithm= 'kd_tree', weights='uniform', metric='euclidean')
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_test)
print( 'Presnost KNN klasifikatora za pouzitia Euklidovej vzdialenosti: '
       ,100*round(metrics.accuracy_score(y_test, y_pred),6), ' %.')

KNN=KNeighborsClassifier(n_neighbors=k_neighbors, algorithm= 'kd_tree', weights='uniform', metric='manhattan')
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_test)
print( 'Presnost KNN klasifikatora za pouzitia Manhattanovej vzdialenosti: '
       ,100*round(metrics.accuracy_score(y_test, y_pred),6), ' %.')
```

Obr. 13 – Výber vhodnej metriky vzdialenosti

Po opakovanom spúšťaní algoritmu obe metriky dosahujú veľmi podobné výsledky, pričom Manhattanova metrika dosahovala vo všeobecnosti o niečo vyššiu presnosť. Rozdiel však nečiní viac ako 2 percentuálne body.

```
from sklearn.neighbors import KNeighborsClassifier

k_neighbors = 10

KNN=KNeighborsClassifier(n_neighbors=k_neighbors, algorithm= 'kd_tree', weights='uniform', metric='manhattan')
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_test)
print( 'Presnost KNN klasifikatora: ', 100*round(metrics.accuracy_score(y_test, y_pred), 6), ' %.')
```

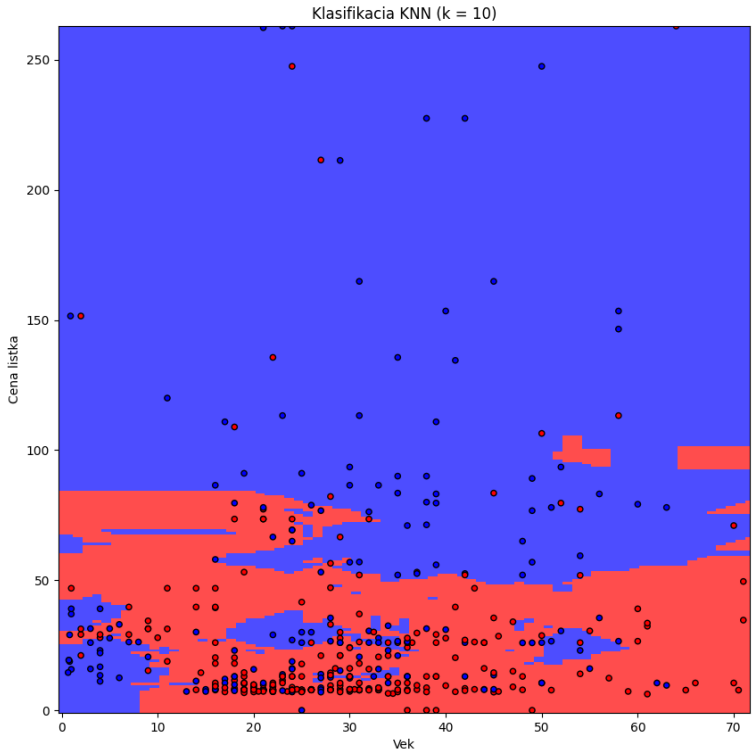
Obr. 14 – Tvorba klasifikátora k-najbližšieho suseda

Pre vizualizáciu si vytvárame farebnú mapu zo zoznamu farieb obsiahnutom v knižniciach matplotlib. V nasledujúcom kroku konštruujeme sieť a vypĺňame ju na základe príslušnosti jednotlivých bodov do tried. Pomocou metódy *scatter* vytvárame bodový graf, ktorého body predstavujú prvky sledovanej množiny. Farba bodu predstavuje jeho zatriedenie. Graf ohraničíme hraničnými bodmi sledovanej množiny, označíme osy a pridáme názov.

```
from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap

cmap_light = ListedColormap(['#FF4D4D', '#4D4DFF'])
cmap_bold = ListedColormap(['#FF0000', '#0000FF'])

x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 1),
np.arange(y_min, y_max, 1))
Z = KNN.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
fig = plt.figure(figsize=(10, 10))
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap_bold, edgecolor='#000000', s=20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title('Klasifikacia KNN (k = %d)' % (k_neighbors))
plt.xlabel('Vek')
plt.ylabel('Cena listka')
fig.savefig('KNN.png')
plt.show()
```

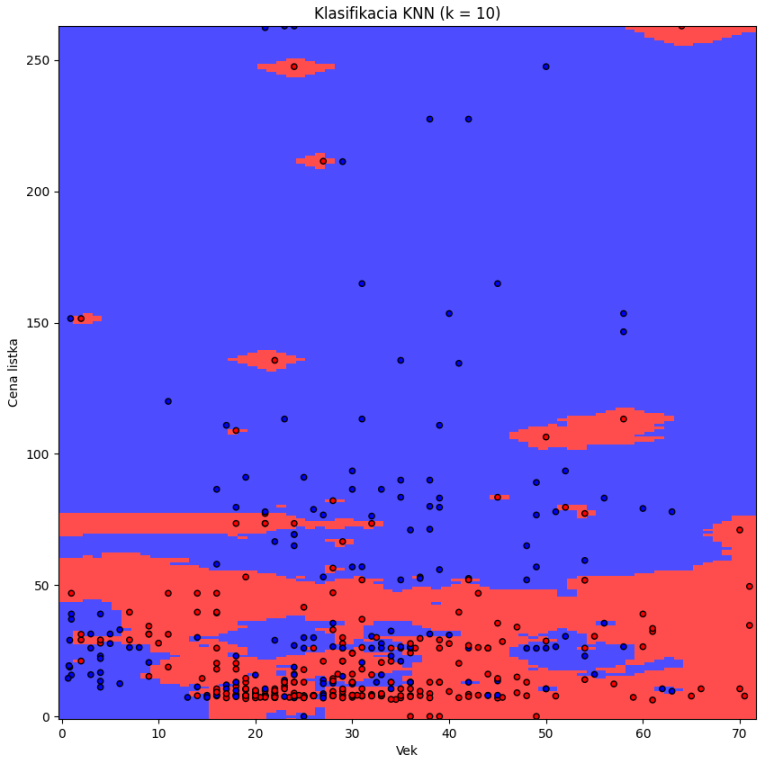
Obr. 15 – Tvorba grafu pri metóde k-najbližšieho suseda

Výsledný graf nám umožňuje sledovať vplyv ceny lístka a veku na osud pasažierov. Body znázorňujú prvky trénovacej množiny. Červené body predstavujú tých, ktorí neprežili, zatiaľ čo modré tých, ktorí prežili. Modré a červené pozadie predstavujú hranice tried. Presnosť tohto modelu je 68,1818 %



Obr.16 – Vizualizácia klasifikátora k-najbližšieho suseda

Ak pri klasifikácii nastavíme proporčné váhy pri meraní vzdialenosti, výsledný model má takýto tvar. Z diagramu je jasne vidieť vplyv váh na klasifikáciu, najmä pri prvkoch, ktorých atribút cena lístka nadobúda hodnotu vyššiu ako 150.



Obr.17 – Vizualizácia klasifikátora k-najbližšieho suseda pri použití proporčných váh

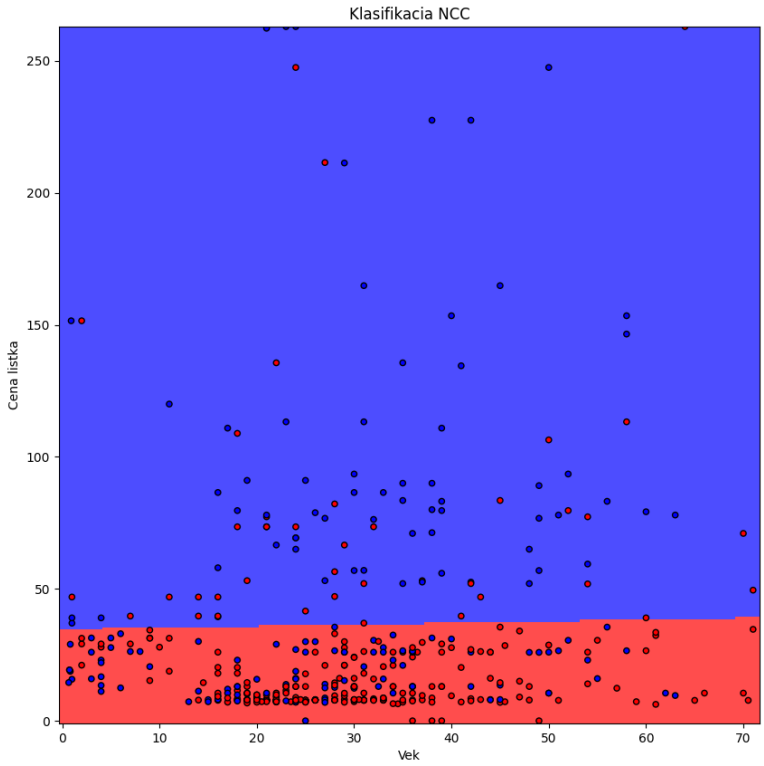
3.3.3 Metóda najbližších ťažísk

Klasifikátor metódy najbližších ťažísk využíva jednoduchý algoritmus na zatrieďovanie prvkov. Každá trieda je reprezentovaná jedným bodom – ťažiskom. Prvky množiny sú potom priraďované do triedy na základe vzdialenosti od jednotlivých ťažísk. Výhoda tohto algoritmu je jeho jednoduchosť, keďže nepotrebujeme dopredu poznať žiaden z parametrov, ako napríklad *k* pri hľadaní najbližšieho suseda.

```
from sklearn.neighbors import NearestCentroid

NCC=NearestCentroid(metric='euclidean')
NCC.fit(X_train, y_train)
y_pred = NCC.predict(X_test)
print( 'Presnost NCC klasifikatora: ', 100*round(metrics.accuracy_score(y_test, y_pred), 6), ' %.')

cmap_light = ListedColormap(['#FF4D4D', '#4D4DFF'])
cmap_bold = ListedColormap(['#FF0000', '#0000FF'])

x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 1),
np.arange(y_min, y_max, 1))
Z = NCC.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
fig = plt.figure(figsize=(10, 10))
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap_bold, edgecolor='#000000', s=20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title('Klasifikacia NCC')
plt.xlabel('Vek')
plt.ylabel('Cena listka')
fig.savefig('NCC.png')
plt.show()
```

Obr. 18 – Tvorba klasifikátora pri metóde najbližších ťažísk

Klasifikátor aj napriek jednoduchosti algoritmu pri použití v našom datasete dosahuje relatívne vysokú presnosť a to 68,88 %. Pri vizualizácii je však vidieť veľký rozdiel pri stanovení hraníc medzi jednotlivými triedami.

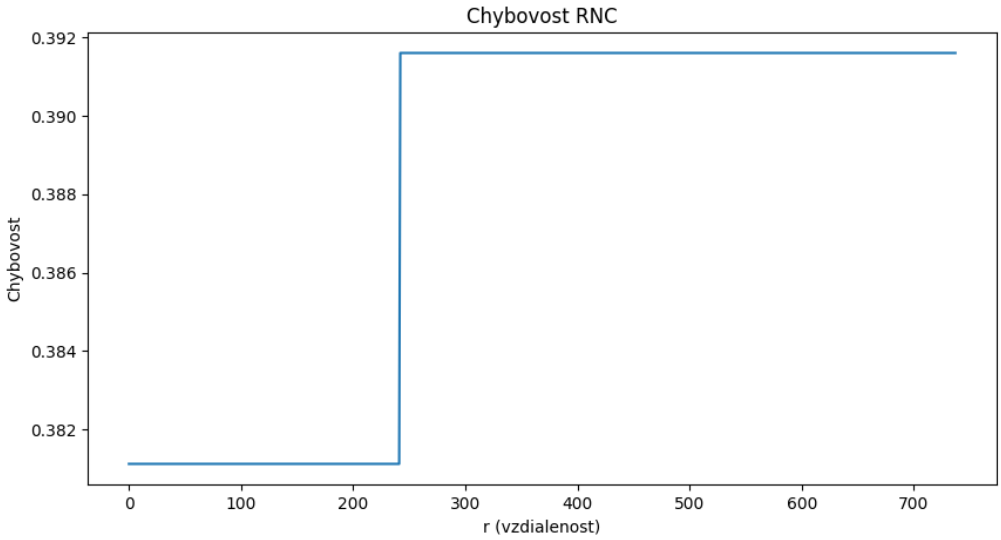


Obr. 19 – Vizualizácia klasifikátora pri metóde najbližších ťažísk

3.3.4 Klasifikátor s rozsahovým dopytom

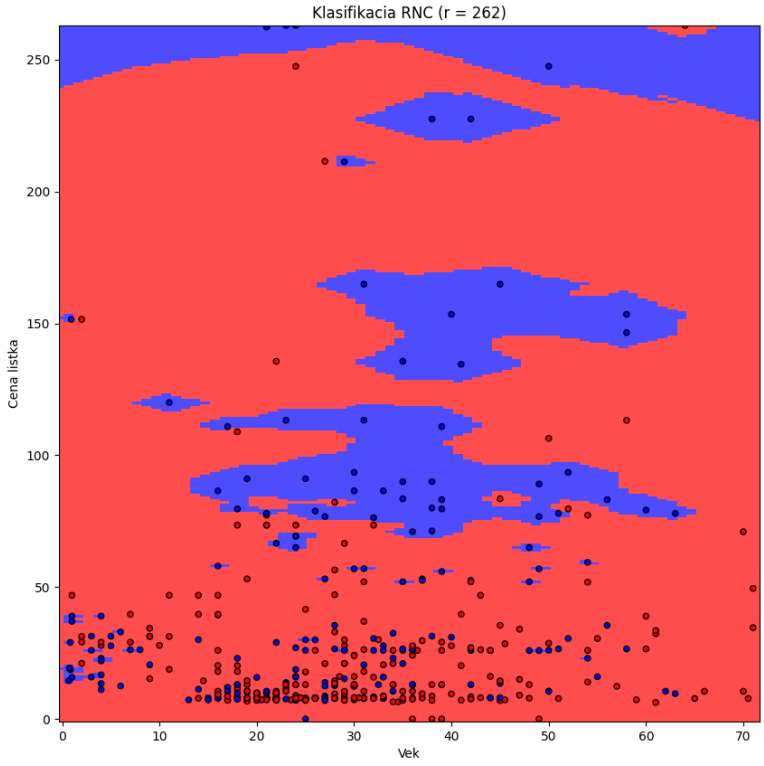
Doteraz sme sa zaoberali zatrieďovaním do tried na základe podobnosti vlastností prvkov s *k*-najbližšími susedmi, poprípade sme vytvorili akýsi pomyselný stred každej triedy. Klasifikátor najbližšieho suseda však nie je jediným nástrojom podobnostného vyhľadávania. Pri klasifikácii sa stáva, že máme presne stanovenú vzdialenosť, v rámci ktorej skúmame podobnosť prvkov. V dvojrozmernom priestore si môžeme predstaviť vzdialenosť ako kruh s polomerom *r,* ktorý predstavuje *rozsah*. Rozsah je smerodajný parameter klasifikátora s rozsahovým dopytom. V praxi môže byť za rozsah považovaná vzdialenosť v kilometroch od určitého miesta. V takomto prípade ide o rozsahový dopyt (Haspanúr, 2014).

Klasifikátor *RadiusNeighborsClassifier* slúži na klasifikáciu v prípade, že je prioritná práve fyzická vzdialenosť. Okrem prípadov z reálneho sveta, keď poznáme vzdialenosť, sa používa pri skúmaní datasetov, v ktorých sú vzdialenosti medzi prvkami príliš vysoké. Tento klasifikátor je do veľkej miery ovplyvnený už spomínanou kliatbou dimenzionality. Tá opisuje jav, kedy s rastúcim počtom dimenzií neúmerne rastie výpočtová zložitosť.



Obr.20 – Chybovosť klasifikátora v závislosti od r

RadiusNeighborsClassifier sa správa nepredvídateľne pri aplikácii na vybraný dataset. Minimálna hodnota pre rozsah $r = 262$, čo je dané štruktúrou dát v datasete. Použitie rozsahu $r > 262$ nemá však na výsledný model žiadny vplyv. Presnosť klasifikátora v tomto prípade je 59,79 %. Z uvedeného je jasné, že rozsahový klasifikátor v žiadnom prípade nie je vhodný na skúmanie tohto datasetu a ide len o demonštráciu jeho funkcionality.



Obr. 21 – Vizualizácia klasifikátora s rozsahovým dopytom

3.4 Metóda SVMs v Pythone

Ako už bolo spomenuté v predchádzajúcej kapitole, SVMs je súbor metód, ktoré klasifikujú prvky datasetu tak, že priestor rozdelia pomocou nadroviny na podpriestory, ktoré obsahujú prvky len jednej triedy. To je realistické iba v prípade, že priestor je lineárne separovateľný.

Pri aplikácii SVMs používame rovnaké údaje ako v predchádzajúcich dvoch príkladoch. Pre jednoduchú vizualizáciu v dvojrozmernom priestore opäť berieme do úvahy pri klasifikácii len dva atribúty, a to *Vek*, a *Cenu lístka.* Načítanie dát prebieha rovnako, ako pri metóde najbližšieho suseda, vrátane transformácie *panda dateframe* na *numpy array.*

Po načítaní dát vytvárame SVMs klasifikátor (SVC) a trénujeme ho na testovacích dátach. Parameter *C* je takzvaný regularizačný parameter. Tento parameter je voliteľný a smie nadobúdať kladné hodnoty. Veľkosť *C* určuje, do akej miery budú penalizované prvky nachádzajúce sa v podpriestore, ktorý nezodpovedá triede daného prvku. S rastúcim *C* rastie penalizácia. *C* reprezentuje kompromis medzi minimalizáciou chyby pri trénovaní modelu a maximalizáciou odstupu. Tentokrát model vkladáme do triedy *pipeline* spolu so *StandardScaler*. SVMs sú senzitívne na mierku vložených dát a preto je vhodné ich normalizovať.

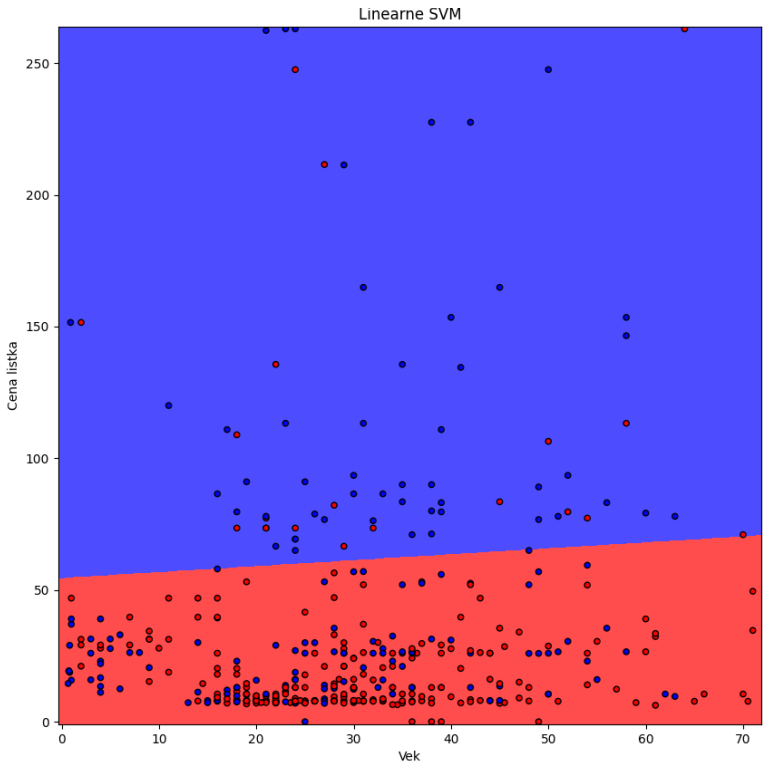
```
from sklearn import svm
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

C = 1
SVM = make_pipeline(StandardScaler(), svm.LinearSVC(C=C, loss="hinge", max_iter=400000))
SVM.fit(X_train, y_train)
y_pred = SVM.predict(X_test)
print( 'Presnost linearneho klasifikatora SVM: ', 100*round(metrics.accuracy_score(y_test, y_pred), 6), ' %.')
```

Obr. 22 – Tvorba lineárneho SVM klasifikátora

Pomocou metódy *countourf* vizualizujem rozdelenie priestoru deliacou nadrovinou. Dvojrozmerný priestor konštruujeme podobne ako v prípade metódy najbližšieho suseda, s tým rozdielom, že farebné časti priestoru predstavujú podpriestory zodpovedajúce jednotlivým triedam.

```
from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap
from mlxtend.plotting import plot_decision_regions

cmap_light = ListedColormap(['#FF4D4D', '#4D4DFF'])
cmap_bold = ListedColormap(['#FF0000', '#0000FF'])

plot_decision_regions(X_train, y_train, clf=SVM)

x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.2), np.arange(y_min, y_max, 0.2))
Z = SVM.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

fig = plt.figure(figsize=(10, 10))
plt.contourf(xx, yy, Z, cmap=cmap_light)
Z = SVM.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap_bold, edgecolor='#000000', s=20)
plt.title('Linearne SVM')
plt.xlabel('Vek')
plt.ylabel('Cena listka')
fig.savefig('SVM_LK.png')
plt.show()
```

Obr. 23 – Tvorba grafu pre vizualizáciu SVM klasifikátora



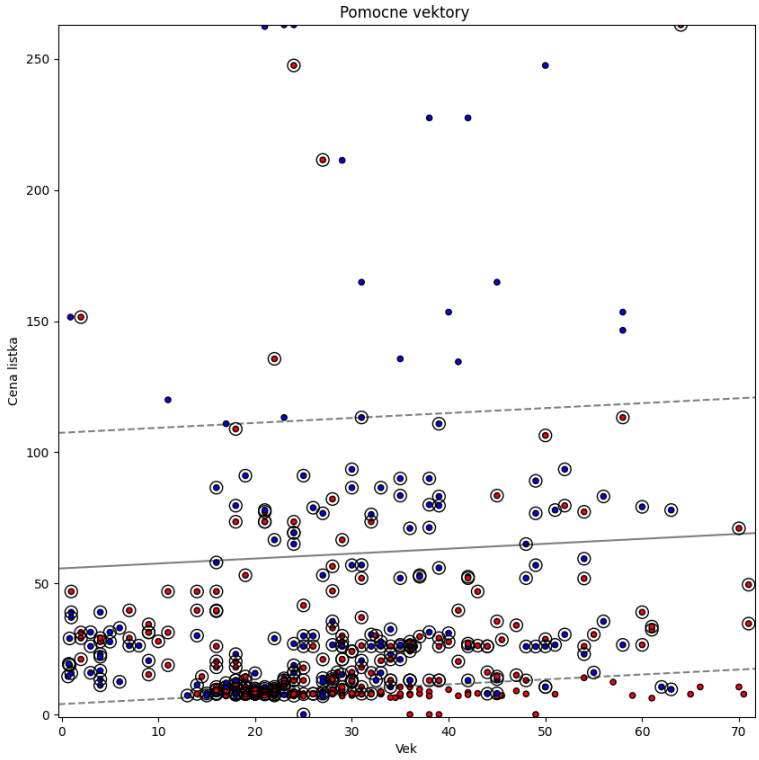
Obr. 24 – Vizualizácia lineárneho SVM klasifikátora

Už na prvý pohľad je jasne vidieť, že priestor nie je lineárne separovateľný. Deliaca nadrovina nedokáže rozdeliť priestor tak, aby v každom podpriestore by boli prvky len jednej triedy. Takýto výsledok sme mohli očakávať vzhľadom na to, že datasety obsahujúce reálne dáta sú len zriedka lineárne separovateľné. Aj napriek tomu je presnosť modelu 66,7832 %.

```
fig = plt.figure(figsize=(10, 10))
decision_function = SVM.decision_function(X_train)
support_vector_indices = np.where((2 * y_train - 1) * decision_function <= 1)[0]
podporne_vektory = X_train[support_vector_indices]

plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap_bold, edgecolor='#000000', s=20)
ax = plt.gca()
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 1), np.arange(y_min, y_max, 1))
Z = SVM.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, colors="#000000", levels=[-1, 0, 1], alpha=0.5, linestyles=["--", "-", "--"])
plt.scatter(podporne_vektory[:, 0], podporne_vektory[:, 1], s=100, linewidth=1, facecolors='none', edgecolors="#000000",
)
plt.title("Pomocne vektory ")
plt.xlabel('Vek')
plt.ylabel('Cena listka')
fig.savefig('SVM_LK_SV.png')
plt.show()
```

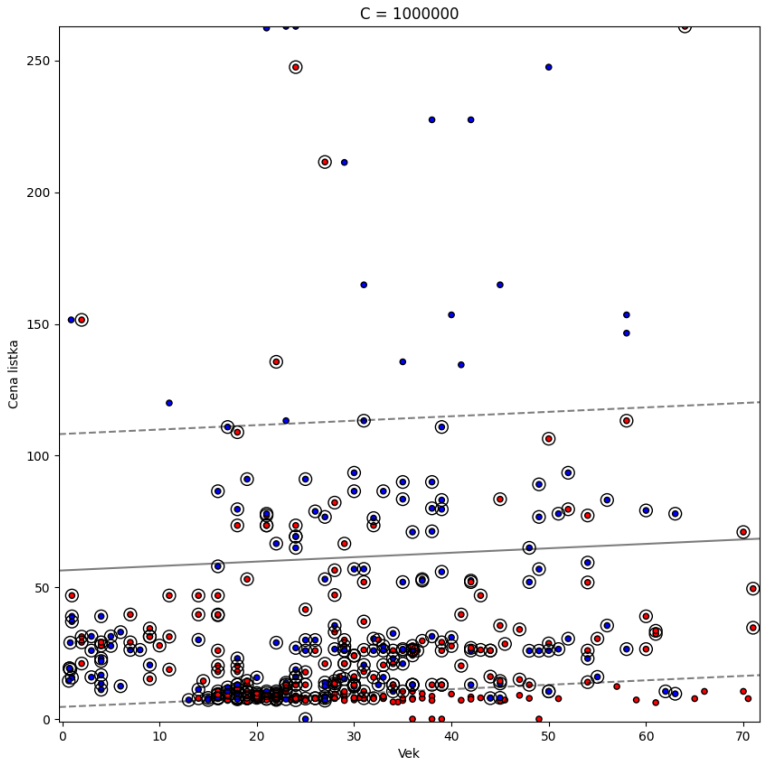
Obr. 25 – Tvorba grafu pre vizualizáciu pomocných vektorov

Pre lepšiu vizualizáciu tohto problému konštruujeme druhý graf, ktorý okrem deliacej nadroviny (plná čiara) zobrazuje aj podporné vektory (prerušované čiary). Pomocné vektory reprezentujú najväčší možný odstup.



Obr. 26 – Vizualizácia pomocných vektorov lineárneho SVM klasifikátora

Z grafu je jasne vidieť, že veľké množstvo prvkov sa nachádza v priestore medzi podpornými vektormi. Čiernym kruhom sú vyznačené všetky prvky, ktoré sa nachádzajú v priestore medzi podpornými vektormi, alebo v podpriestore prislúchajúcom druhej triede. Definitívne teda môžeme povedať, že priestor nie je lineárne separovateľný. Rastúci regulačný parameter C má len minimálny vplyv na nami použitý dataset. Aj pri veľmi vysokom $C = 1\ 000\ 000$ nastane len minimálna zmena v postavení pomocných vektorov.



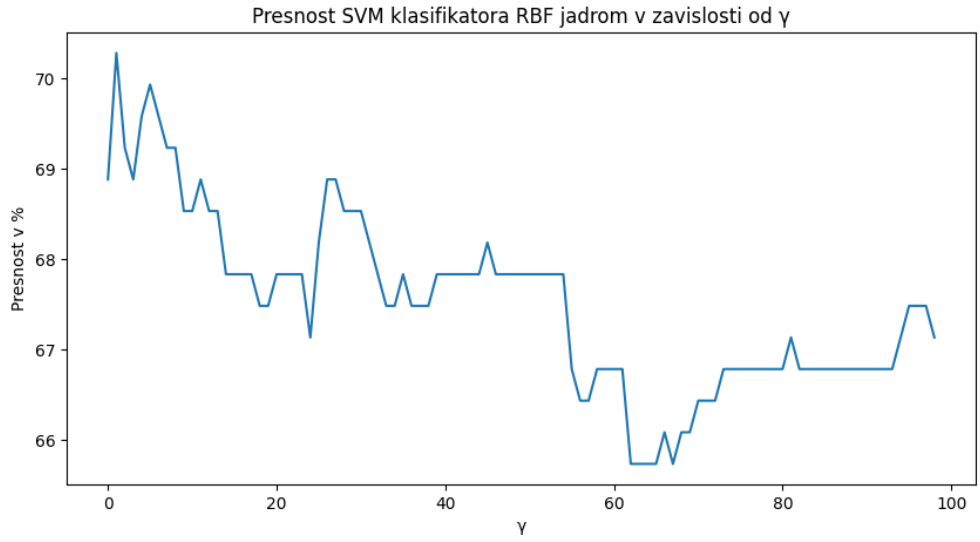
Obr. 27 – Vizualizácia vplyvu parametra C na pomocné vektory lineárneho SVC

3.4.1 SVMs v lineárne neseparovateľnom priestore

V prípade, že dataset nie je lineárne separovateľný musím pristúpiť k jadrovému triku, teda zámene jadrovej funkcie k(x_i ⃗, x_j ⃗). Táto metóda umožňuje presun prvkov datasetu z priestoru, ktorý je lineárne neseparovateľný do lineárne separovateľného priestoru.

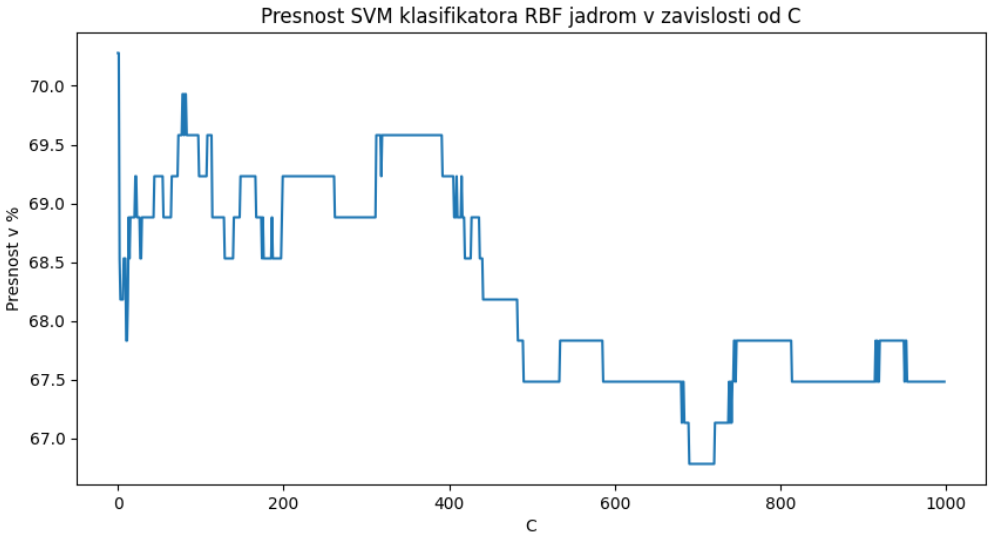
Pre prvú zmenu jadra používame Gaussovu zmenu jadra, ktorá je vstavaná do SVMs klasifikátora. Gaussova radiálna funkcia pridáva atribúty podobnosti. Tie udávajú podobnosť prvku ku orientačným bodom v priestore. Podobnosť je reprezentovaná vzdialenosťou. Orientačné body, a teda aj vzdialenosti prvkov k týmto bodom pridávame do momentu, kým nie je priestor lineárne separovateľný. Každý atribút pridáva novú dimenziu. Pridávaním týchto atribútov sa môže stať, že pôvodné vlastnosti prvkov sa stanú pri klasifikácii nepotrebnými a teda klasifikácia prebehne len za pomoci merania podobnosti, respektíve vzdialenosti k pridaným bodom. Gaussova radiálna funkcia jadra má tvar exp(-γ|| x_i ⃗ - x_j ⃗||^2). SVMs klasifikátor vyžaduje zadanie parametra γ ktorý udáva, aký vplyv budú mať jednotlivé prvky sledovanej množiny. Čím vyššiu hodnotu parameter γ nadobúda, tým menej sú ovplyvnené vzdialené prvky. Parameter γ musí byť vždy nenulový. Pre SVMs

klasifikátor parameter γ je reprezentovaný parametrom *gamma*. Okrem číselnej hodnoty umožňuje vložiť hodnotu *auto*, kedy pridelí $\gamma = 1/N$, kde N je počet prvkov datasetu. Z nasledujúceho grafu je vidieť vplyv γ na presnosť algoritmu, pričom model dosahuje najvyššiu presnosť pri $\gamma = 2$.



Obr. 28 – Vizualizácia vplyvu γ na SVMs klasifikátor s RBF

Po zistení optimálne veľkosti parametra γ ostáva ešte otestovať vplyv parametra C. V tomto prípade dosahuje klasifikátor najvyššiu presnosť pri $C = 1$. S rastúcim C presnosť klesá.

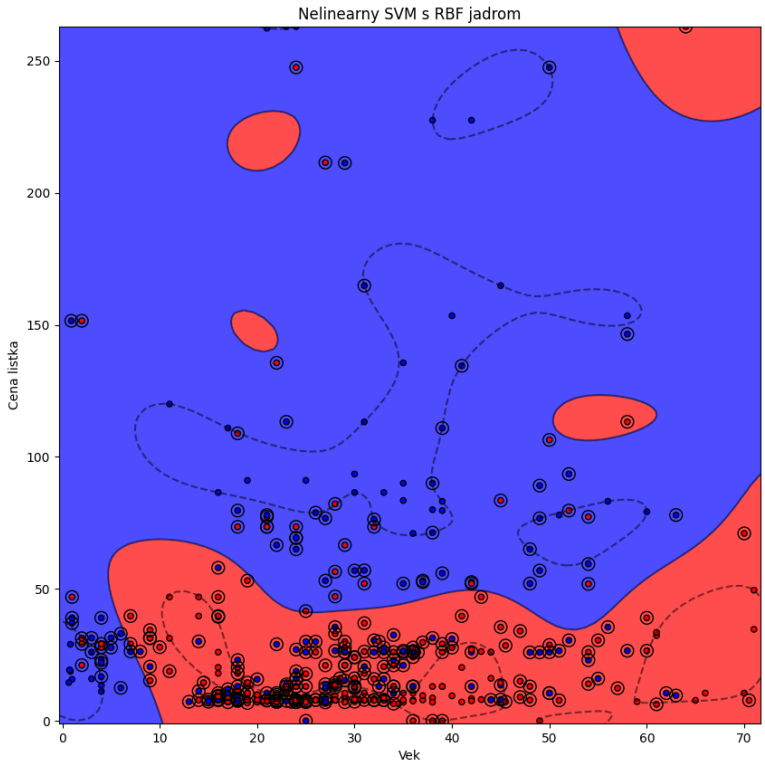


Obr. 29 – Vizualizácia vplyvu C na SVMs klasifikátor s RBF

Výsledný model tvoríme podobne ako v prípade lineárneho klasifikátora. Jadrová funkcia sa udáva do parametra *kernel*.

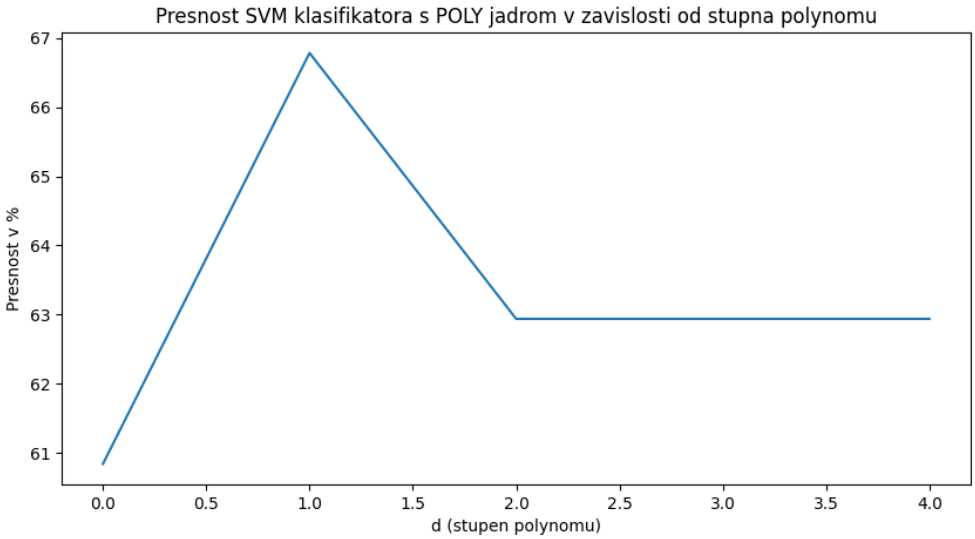
```
from sklearn import svm
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

C = 1
SVM = make_pipeline(StandardScaler(), svm.SVC(C=C, kernel='rbf', gamma=2))
SVM.fit(X_train, y_train)
y_pred = SVM.predict(X_test)
print( 'Presnost SVM s RFB jadrom: ', 100*round(metrics.accuracy_score(y_test, y_pred), 6), ' %.')
```

Výsledný model má už na prvý pohľad špecifický tvar, v porovnaní s lineárnym SVMs klasifikátorom. V dvojrozmernom priestore sa javí byť priestor preťatý na viacerých miestach. Deliaca nadrovina už nemá tvar priamky. Aj napriek tomu sa však nájdu prvky, ktoré sa nachádzajú v podpriestoroch náležiacich inej triede. Klasifikátor dosahuje presnosť 70.2797 %. Môžeme si všimnúť, že niektoré z novovytvorených podpriestorov neobsahujú žiadny z prvkov trénovacej množiny.



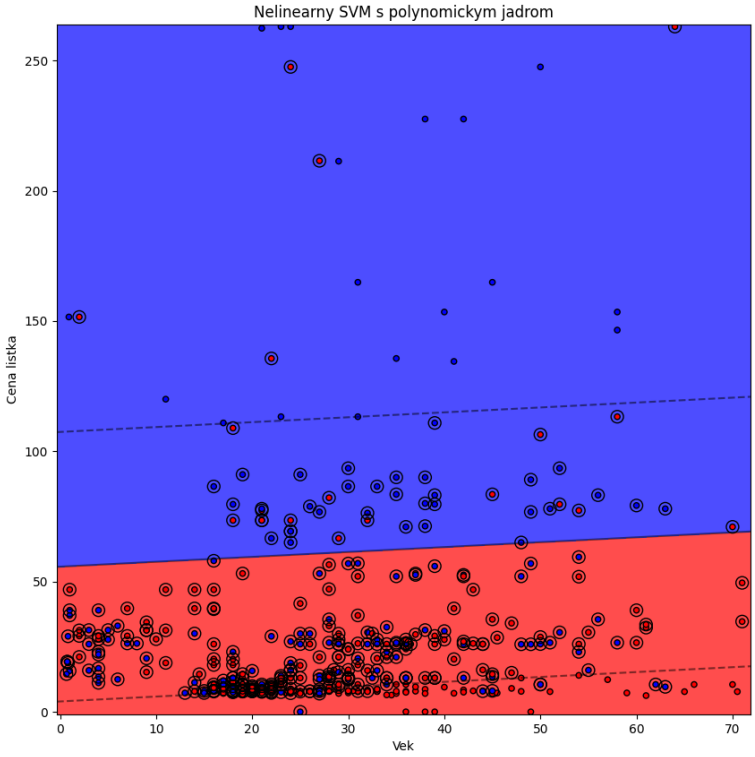
Obr. 30 – Vizualizácia nelineárneho SVMs klasifikátora s RBF jadrom

Druhou možnosťou, ako sa vysporiadať s lineárne neseparovateľným priestorom je použitie polynomického jadra. Pre použitie polynomickej funkcie jadra v SVC zadáme *kernel='poly'*. Funkcia jadra s polynómom má podobu $k(\gamma\, \vec{x_i},\ \vec{x_j} + r)^d$, kde parameter d udáva stupeň polynómu a r je voľný parameter. Prvé, čo je nutné zistiť je veľkosť optimálneho stupňa d pre klasifikačný model.



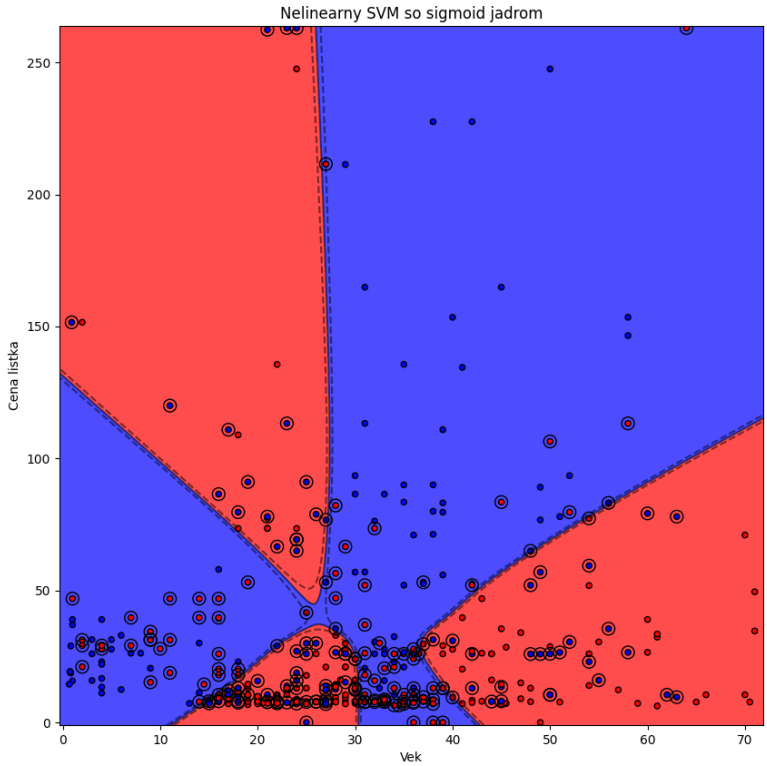
Obr.31 – Presnosť SVMs klasifikátora v závislosti od stupňa polynómu

Na základe grafu vidíme, že najvyššiu presnosť klasifikátor dosahuje pri použití lineárneho polynóm, teda polynómu prvého stupňa. Takisto veľkosť voľného parametra *r* ani veľkosť koeficientu *C* nevyzerá mať vplyv na efektivitu modelu. Polynomická funkcia 1. stupňa je totožná s lineárnym SVMs klasifikátorom.



Obr. 32 – Vizualizácia nelineárneho SVMs klasifikátora s polynomickým jadrom

Ďalšou možnosťou, ako riešiť problém lineárne neseparovateľného priestoru je použiť *Hyperbolické tangensové jadro*, takzvané *jadro sigmoid*. Jadro sigmoid patrí do modelov viacvrstvových neurónových sietí, nazývaných perceptróny. Neurónové siete sú modely simulujúce funkcionalitu ľudského mozgu. Neurónové siete fungujú na princípe uzlov – neurónov, ktoré reagujú na vstupy z ostatných neurónov, vykonajú určitú matematickú operáciu a výstup posunú ďalším neurónom. Vykonaná operácia je závislá od intenzity vstupu. Na základe zmien intenzity vstupu je možné prispôsobovať výstupy a tak trénovať neurónovú sieť. Jadrová funkcia sigmoidu má tvar $\tanh(\gamma\| \vec{x_i} - \vec{x_j} \| + r)$. Takáto jadrová funkcia vytvára dvojvrstvovú neurónovú sieť. Prvá vrstva predstavuje množinu d neurónov, kde d zodpovedá počtu dimenzií. Tieto neuróny prijímajú a ďalej vysielajú signál bez toho, aby vykonali akúkoľvek matematickú operáciu. Neuróny prvej vrstvy len prideľujú váhy vstupu. Druhá vrstva pozostáva z jedného neurónu, ktorý je zodpovedný za vykonanie matematickej operácie, v tomto prípade na základe vstupu z prvej vrstvy prideľuje váhu $\{-1, +1\}$. Táto váha predstavuje predikciu zatriedenia prvku (Aggarwal, 2015). Rovnakým postupom ako pri predchádzajúcom SVMs klasifikátora nájdeme parametre C a r. Model dosahuje najvyššiu presnosť 63,2867 % pri $C = 4$ a bez použitia parametra r.



Obr. 33 – Vizualizácia nelineárneho SVM klasifikátora s jadrom sigmoid

4. Diskusia

V predchádzajúcej kapitole sme mohli vidieť praktické využitie metód a algoritmov strojového učenia pri klasifikácii. Modely vytvárané pomocou klasifikátorov Scikit dosahovali aj pri aplikácií na nehomogénne dáta obsahujúce množstvo hluku relatívne dobré výsledky. Na druhej strane sme mohli pozorovať, že jednotlivé algoritmy dosahovali lepšie výsledky, ako iné. To je samozrejme dané rôznorodosťou použitých metód a špecifikami skúmaného datasetu.

Naivný Bayesov klasifikátor, spomenutý v druhej kapitole, je vhodným nástrojom pre klasifikáciu dokumentov. Vyznačuje sa tým, že pri trénovaní jeho modelu je potrebná len relatívne malá trénovacia množina. Osvedčil sa napríklad pri odchytávaní správ obsahujúcich spam. Jeho výhodou oproti zložitejším modelom je rýchlosť. Naivný Bayesov klasifikátor dokáže pracovať taktiež s kategorickými údajmi. Vo všeobecnosti ide o solídny klasifikátor, ktorý však nie je vhodný na predikciu, pretože je naivný. Teda považuje všetky atribúty za nezávislé, čo je v praxi zriedka kedy pravda.

Klasifikátory využívajúce metódu najbližšieho suseda, aj napriek relatívnej jednoduchosti ich tvorby, sú považované za veľmi účinné nástroje pre klasifikáciu. V praxi sa ukázali byť efektívne pri analýze obrazových záznamov, satelitných máp a pri rozpoznávaní rukopisu. Veľmi dobré výsledky tieto klasifikátory dosahujú pri skúmaní datasetov, pri ktorých je nepravidelná hranica medzi jednotlivými triedami. Treba poznamenať, že práve pri klasifikátore najbližšieho suseda sme narazili na problém s neefektívnym využívaním pamäte, ktorý znateľne spomaľoval chod programu. Klasifikátory najbližšieho suseda nie sú stavané na prácu s kategorickými atribútmi. Kategorickým atribútom však môžeme priradiť kardinalitu a tým sa stanú použiteľnými.

Rozhodovacie stromy sú efektívnym nástrojom pri predpovedaní výsledku na základe naučených pravidiel. Výhodou klasifikačného modelu v podobe rozhodovacieho stromu je jeho jednoduchá reprezentácia. V takomto strome je jednoduché sa orientovať. Pri tvorbe klasifikačného modelu sú potrebné často len minimálne úpravy vstupných dát. Veľkou výhodou stromovej štruktúry je nízka výpočtová náročnosť, ktorá s pribúdajúcimi prvkami rastie logaritmicky. Rozhodovacie stromy sú vhodné na klasifikáciu tak kategorických údajov, ako aj numerických údajov. Pri tvorbe rozhodovacích stromov sa často stretávame s problémom pretrénovania modelu, ktorý je výsledkom prehnane veľkého stromu. Preto je vo väčšine prípadov nutné stromy zjednodušovať, orezávať. Pretrénovaniu je však možné predísť aj vhodným stanovením hĺbky a šírky, do ktorej sa model tvorí.

Ďalšou z nevýhod rozhodovacích stromov je, že sú citlivé na zmeny v dátach a dominantné triedy. Aj drobná zmena v dátach môže viesť k úplne inému stromu.

Poslednou triedou klasifikátorov, na ktoré sa táto práca zameriava sú support vector machines. Ich veľkou výhodou je, že dokážu pracovať s mnohorozmernými datasetmi. SVMs vedia byť efektívne dokonca aj v prípade, keď počet dimenzii prevyšuje počet hodnotených vlastností prvkov. V porovnaní s ostatnými klasifikátormi nevyžadujú veľké množstvo pamäte. To je spôsobené malým množstvom prvkom potrebných na klasifikáciu. Tie tvoria pomocné vektory. Tento aspekt bol pozorovateľný najmä pri porovnaní s klasifikátorom najbližšieho suseda, ktorý využíval desaťnásobne viac operačnej pamäte. Možnosť pracovať s rôznymi jadrovými funkciami dáva SVMs veľkú adaptabilitu. V prípadoch, kedy štandardné jadrové funkcie nie sú postačujúce, SVMs umožňujú používateľovi definovať vlastnú jadrovú funkciu. Pri výbere jadrovej funkcie je nutné byť opatrný, výber nesprávnej funkcie totiž môže viesť k pretrénovaniu modelu. (Pedregosa, 2011). SVMs samé o sebe nie sú vhodné na použitie, ak sú údaje kategorické. Opäť, ak je možné tieto kategorické údaje zoradiť, dajú sa nahradiť numerickými hodnotami.

Je nutné poznamenať, že klasifikátory Scikit sú veľmi dobre optimalizované a dosiahnuť s nimi dobré výsledky nie je náročné. Pre porovnanie výkonu klasifikátorov použijeme test, kedy ich spustíme opakovane, pričom pri každom spustení bude inak rozdelený dataset na trénovaciu a testovaciu množinu. Rozdelenie datasetu zostáva rovnaké medzi použitými klasifikátormi. Následne porovnáme priemerné hodnoty presnosti klasifikátorov. Klasifikátor s rozsahovým dopytom z tohto porovnania vyradíme, kvôli nekonzistentným výsledkom. Parametre modelov sú zvolené tak, aby klasifikátor dosahoval najvyššiu presnosť. Pri klasifikácii sa zohľadňujú vlastnosti: trieda, pohlavie, vek a cena lístka.

Klasifikátor	Parametre klasifikátora	Presnosť modelu v %
SVMs s radiálnou funkciou jadra	γ=10, C =1	72,97
SVMs s polynomickým jadrom	d = 3, C = 2	72,41
Rozhodovací strom	gini index	71,89
SVMs s lineárnym jadrom	C=1	70,52
Klasifikátor k-najbližšieho suseda	k = 25, k-d tree, váhy = distance, Manhattanova vzialenosť	64,58
SVMs so sigmoid jadrom	γ=17, C =1	61,68
Metóda najbližších ťažísk	Euklidova vzdialenosť	60,00

Tabuľka 1 – Porovnanie výsledkov klasifikátorov

Z tabuľky je vidieť, že klasifikátory SVMs a rozhodovací strom dosahujú veľmi podobné priemerné hodnoty. SVMs s radiálnou (Gaussovou) funkciou jadra a s polynomickým jadrom obstáli najlepšie. Rozhodovací strom a lineárny SVMs klasifikátor neboli ďaleko za nimi. Treba však poznamenať, že klasifikátory Scikit sú vysoko optimalizované. Keby vytvárame klasifikátory od základov, mohli by sme čakať diametrálne odlišné výsledky.

Záver

V práci sme analyzovali klasifikáciu ako nástroj strojového učenia s učiteľom. Zoznámili sme sa s teoretickými a matematickými základmi vybraných algoritmov strojového učenia používaných pro klasifikácii. Objasnili sme si Bayesov teorém ako základ pre Bayesov naivný klasifikátor. Oboznámili sme sa s metrikami používanými pri hľadaní najbližšieho suseda pri rovnomernom algoritme. Ozrejmili sme si postupy pri tvorbe rozhodovacích stromov. V závere opisu súčasného stavu sme sa zaoberali teoretickými základmi pre support vector machines.

Následne sme tieto poznatky aplikovali pri tvorbe klasifikačných modelov v jazyku Python. Tieto modely demonštrovali, ako sa jednotlivé algoritmy dajú využiť pri klasifikácii a aké špecifiká so sebou prinášajú. Ukázali sme si, ako sa vykonávajú úpravy vstupných údajov v podobe normalizácie, škálovania. Zatiaľ čo v prvej kapitole sme sa oboznámili s pojmami trénovacia a testovacia množiny, v tejto časti sme si ukázali ako dataset rozdeliť na trénovaciu a testovaciu množinu. Následne sme ich použili na tvorbu rozhodovacieho stromu, ktorý sme pre lepšiu predstavu vizualizovali. Zhodnotili sme, že rozhodovací strom je relatívne jednoduchý, prehľadný a presný. Pri metóde najbližšieho suseda sme demonštrovali niekoľko možných prístupov. Pri klasifikátore *k*-najbližšieho suseda sme si ukázali, ako vybrať vhodný parameter *k*. Metódou najbližších ťažísk sme ukázali o niečo jednoduchší prístup k metóde najbližšieho suseda. Napriek jeho priamočiaremu prístupu priniesol dobré výsledky. Pri klasifikátore s rozsahovým dopytom sme si overili, že skúmané dáta majú veľký vplyv na výber klasifikátor, a že výber nevhodného klasifikátora môže viesť k neadekvátnym výsledkom. Z tohto dôvodu sme ho pri finálnom porovnaní klasifikátorov vynechali. Metóda SVMs nám pomohla objasniť si problémy spojené s lineárne neseparovateľným priestorom. Zmenou jadrovej funkcie sa nám však podarilo vytvoriť obstojné klasifikačné modely, pričom každý z týchto modelov so sebou priniesol určité špecifiká. Pri SVMs polynomickým jadrom sme hľadali optimálny stupeň polynómu, zatiaľ čo pri SVMs modeli s radiálnou funkciou jadra a jadrom sigmoid sme hľadali najlepšiu hodnotu parametra γ. Klasifikačné modely vytvorené pomocou SVMs sme si vizualizovali v dvojrozmernom euklidovskom priestore, aby bolo možné pozorovať zmeny v rozdelení priestoru v závislosti od použitej metódy.

Následne sme v diskusii zhrnuli podstatu vyššie spomenutých klasifikačných modelov, ich kladné a záporné stránky. Modely sme upravili, otestovali a výsledky porovnali. Použité modely dosahovali dobré a konzistentné výsledky. Podobnosť výslednej

presnosti klasifikátorov sme pripísali vysokej optimalizácii modelov, s ktorými sme pracovali. Tým sme dovŕšili splnenie hlavného cieľa diplomovej práce.

Zoznam použitej literatúry

AGGARWAL, Charu C. *Data Mining: The Textbook*. 1. vyd. Springer International Publishing Switzerland, 2015. 726 s. ISBN 978-3-319-14141-1

AMO, Del et al. *On the principles of fuzzy classification*. 18th North American Fuzzy Information Processing Society Annual Conf, (S. 675 – 679). 1999.

BISHOP, Christopher M. *Pattern Recognition and Machine Learning.* Springer Science+Business Media. 2006. 738 s. ISBN-10: 0-387-31073-8

BRAMER, Max. *Principles of Data Mining*. 4. vyd. Springer-Verlag London Ltd. 2020. 571 s. ISBN 978-1-4471-7492-9

BRÉDOVÁ, Lucia. *Naivný Bayesovský klasifikátor* [bakalárska práca]. In: Fakulta elektrotechniky a informatiky, Technická univerzita v Košiciach. 2011

DORADO, Hugo et al. *Wrapper for Building Classification Models Using Covering Arrays*. In IEEE Access (Volume: 7), Electronic ISSN: 2169-3536. september 2019

ENCYCLOPEDIA OF MATHEMATICS. *Bayes formula*. Dostupné na: http://encyclopediaofmath.org/index.php?title=Bayes_formula&oldid=45997

GRAY, Robert M. *Entropy and Information Theory*. 1. vyd. Springer-Verlag. 2013. 409 s. ISBN 978-1-4419-7970-4

HASPANÚR, Eirk. *Algoritmy pre vyhľadávanie najbližších susedov.* [bakalárska práca]. In: Fakulta informatiky, Masarykova univerzita. Brno. 2014

HSSINA, Badr et al. *A comparative study of decision tree ID3 and C4.5*. In: International Journal of Advanced Computer Science and Applications (IJACSA). Special Issue on Advances in Vehicular Ad Hoc Networking and Applications 2014. ISSN 2156-5570

HUSSEIN, Sajid. *Deep Learning for Sentiment Analysis*. In: Medium, 06. 06. 2019. Dostupné na: https://medium.com/@hussein.sajid7/deep-learning-for-sentiment-analysis-7da8006bf6c1

LAYTON, Robert. *Learning Data Mining with Python*. 1. vyd. Packt Publishing. 2015. 317 s. ISBN 978-1-78439-605-3

MCGRAYNE, Sharon Bertsch. *The Theory That Would Not Die*. Yale University Press, 2012. 360 s. ISBN 978-0300188226

PEDREGOSA, F et al. *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research (Volume 12), 2011

PYTHON SOFTWARE FOUNDATION. *Python 3.9.12 documentation.* Dostupné na: https://docs.python.org/3.9/

SEDDAWY, *Ahmed Bahgat El et al. Applying Classification Technique using DID3 Algorithm to improve Decision Support System under Uncertain Situations.* In International Journal of Modern Engineering Research (IJMER) (Vol. 3). Electronic ISSN: 2249-6645. august 2013

TANG, Jiliang et al. *Feature Selection for Classification: A Review*. In: Data Classification: Algorithms and Applications. 2014

ZHOU, Yu et al. *Machine Learning Based Embedded Code Multi-Label Classification*. In IEEE Access (Volume: 9), Electronic ISSN: 2169-3536. október 2021