VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

POROVNÁNÍ KLASIFIKAČNÍCH METOD

COMPARISON OF CLASSIFICATION METHODS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MARTIN DOČEKAL**
AUTHOR

VEDOUCÍ PRÁCE **Ing. IVANA BURGETOVÁ, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Dočekal Martin, Bc.**
Program: Informační technologie Obor: Inteligentní systémy
Název: **Porovnání klasifikačních metod**
Comparison of Classification Methods
Kategorie: Data mining

Zadání:

1. Prostudujte různé klasifikační metody.
2. Po dohodě s vedoucí připravte vhodné datové sady pro porovnání klasifikačních metod.
3. Navrhněte aplikaci, která umožní porovnání vybraných klasifikačních metod dle vybraných kritérií.
4. Navrženou aplikaci implementujte.
5. Otestujte vybrané klasifikační metody na zvolených datových sadách.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Third Edition. Morgan Kaufmann Publishers, 2012, 703 p., ISBN 978-0-12-381479-1

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Burgetová Ivana, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 9. května 2019

Abstrakt

Tato práce se zabývá porovnáním klasifikátorů. Nejprve jsou popsány klasifikační techniky založené na strojovém učení, poté je navržen a implementován systém pro porovnání klasifikátorů. Dále jsou popsány klasifikační úlohy a datové sady, na kterých je systém otestován. Vyhodnocení je prováděno pomocí standardních metrik.

V rámci práce je též navržen a implementován klasifikátor založený na principu evolučních algoritmů.

Abstract

This thesis deals with a comparison of classification methods. At first, these classification methods based on machine learning are described, then a classifier comparison system is designed and implemented. This thesis also describes some classification tasks and datasets on which the designed system will be tested. The evaluation of classification tasks is done according to standard metrics.

In this thesis is presented design and implementation of a classifier that is based on the principle of evolutionary algorithms.

Klíčová slova

strojové učení, extrakce příznaků, Bag-of-words, TF-IDF, hašování příznaků, Histogram orientovaných gradientů, HOG, SVM, Support Vector Machine, Rozhodovací strom, Umělá neuronová síť, perceptron, ANN, k-nejbližších sousedů, naivní Bayesův klasifikátor, CEEF, klasifikace pomocí evolučně odhadovaných funkcí, evoluční algoritmy, klasifikace, vyhodnocování klasifikátorů, selekce příznaků

Keywords

machine-learning, feature extraction, Bag-of-words, TF-IDF, feature hashing, Histogram of oriented gradients, HOG, SVM, Support Vector Machine, Decision Tree, Artificial neural network, perceptron, ANN, k-nearest neighbors, Naive Bayes classifier, CEEF, Classification by evolutionary estimated functions, evolutionary algorithms, classification, evaluation of classifiers, feature selection

Citace

DOČEKAL, Martin. *Porovnání klasifikačních metod*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Ivana Burgetová, Ph.D.

Porovnání klasifikačních metod

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením paní Ing. Ivany Burgetové, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Dočekal
20. května 2019

Poděkování

Rád bych poděkovat své vedoucí diplomové práce Ing. Ivaně Burgetové, Ph.D., za podnětné rady a konzultace.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Tato diplomová práce se zabývá automatickými klasifikačními metodami založenými na strojovém učení a jejich porovnáním. V dnešní době přinášející velké množství dat, které není člověk ani schopen zpracovávat, nalézá klasifikace velké uplatnění. Umožňuje totiž automaticky řešit například problémy jako jsou: detekce závadného obsahu (např. SPAM), identifikace objektů v obraze, roztřídění knih dle obsahu a další.

Rozebereme si zde klasifikaci jako takovou a zařadíme ji do oboru umělé inteligence a dolování znalostí. Čtenář tak získá širší povědomí o tom, kam zde popisované metody klasifikace zařadit. Uvedeme si několik dnes význačných klasifikátorů: SVM, neuronové sítě, rozhodovací stromy, naivní Bayesovi klasifikátory, k-nejbližších sousedů a klasifikátory založené na principu evolučních algoritmů.

Mimo samotné klasifikace tato práce popisuje i metody pro extrakci příznaků. Extrakce příznaků bývá nedílnou součástí procesu klasifikace, zvláště když pracujeme s daty jako jsou obrázky či běžné texty.

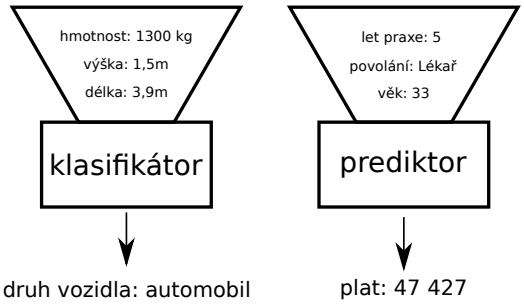
V neposlední řádě bude proveden návrh aplikace pro porovnání klasifikátorů a popis její implementace. Uvedeme druhy problémů, na kterých bude tato aplikace zkoušena a popíšeme si vybrané datové sady, které budou pro tyto problémy použity. Na těchto datových sadách bude zjišťováno jak dobře si dokáží klasifikátory poradit s předloženými problémy a jak jsou při řešení těchto problémů časově náročné. V rámci těchto úkolů bude sledován i vliv ostatních aspektů (ne jen problém sám) jako je velikost dat, které dáme k dispozici klasifikátoru, či například počet tříd, do kterých mají klasifikátory za úkol klasifikovat. Pro vyhodnocení těchto experimentů budou používány zde popsané standardní metriky.

Kapitola 2

Klasifikace

V této sekci budou představeny základní pojmy, které čtenáře uvedou do problematiky. Popíšeme si klasifikaci jako takovou. Ukážeme si, jaké metody lze pro klasifikaci použít a jak zjišťovat jejich kvalitu/úspěšnost. Sekce vychází z předchozí práce [25].

Klasifikace je forma datové analýzy, která vytváří modely popisující třídy dat. Takovéto modely nazýváme klasifikátory. Predikují kategorické (diskrétní, neuspořádané) značení daných tříd. Pokud bychom chtěli určovat spojité (uspořádané) atributy, pak bychom nemluvili o klasifikaci, ale o predikci. Model pro predikci je nazýván prediktor. [28]



Obrázek 2.1: Ilustrace rozdílu mezi klasifikací a predikcí.

Rozdíl mezi těmito přístupy ilustruje obrázek 2.1. Oba modely vyhodnocují svůj výsledek na základě hodnot zvolených atributů. Při klasifikaci model vybere jednu z n tříd, například: automobil, motocykl a nákladní automobil. Při predikci naopak dojde k vybrání hodnoty ze spojitého intervalu pro atribut plat.

Klasifikaci můžeme zařadit do dolování znalostí, anglicky Data Mining[1]. Data mining je proces objevování zajímavých vzorů a znalostí z velkého množství dat. [28]

2.1 Strojové učení a dolování znalostí

V rámci procesu dolování znalostí se používá vícero technik, jako je statistika či strojové učení. Tato práce se zabývá právě strojovým učením.

Strojové učení je podobor umělé inteligence. Samotný obor umělé inteligence lze charakterizovat jako snahu zautomatizovat intelektuální úlohy, které bývají běžně prováděny člověkem. V minulosti se věřilo, že umělou inteligenci (na úrovní srovnatelné s člověkem) lze získat tak, že naprogramujeme pevně danou sadu pravidel. Tento přístup nazýváme symbolickou umělou inteligencí. Symbolická umělá inteligence je sice vhodná pro některé druhy problémů, jako jsou dobře specifikované logické problémy, nicméně později se ukázalo, že tento přístup není příliš vhodný pro složitější úlohy jako je například klasifikace obrazu. Vznikl tedy nový přístup, a to právě strojové účení[2]. Namísto toho, aby programátor přímo

[1] Z angličtiny se tento pojem překládá různě: dolování z dat, vytěžování dat, dolování znalostí a další.
[2] V angličtině: Machine learning (ML).

určoval pravidla, tak se je program sám naučí (odvodí) na základě vstupních dat (případně i poskytnuté zpětné vazby). [23]

Ovšem co to vlastně znamená, že se program učí? Uveďme si zde příklady a definici: [35]

Definice 1. *Říkáme, že počítačový program se učí ze zkušenosti E, s ohledem na třídu úkolů T a měřením výkonu P, tehdy pokud se jeho výkon na úkolech z T, měřený pomocí P, zlepšuje se zkušeností E.*

Ukažme si, pro osvětlení, tuto definici na příkladu. Protože je tato práce zaměřena na klasifikaci, tak si rozebereme problém, který s ní souvisí, a to spam filtr[3] realizovaný pomocí strojového učení:

- **úloha T** - Klasifikuj nevyžádanou poštu do třídy SPAM, ostatní poštu do třídy HAM.
- **měření výkonu P** - Počet správně klasifikované pošty.
- **zkušenost E** - Dataset s poštou, kde každá pošta je správně označena jako SPAM/HAM.

Podobně by šlo rozebrat i jiné problémy jako je například hraní tenisu. Měření výkonu by mohl být počet vítězných utkání a zkušenost by mohly být hry odehrány proti soupeři, či proti sobě samému.

Samotné učení může probíhat vícero způsoby, proto ve strojovém učení rozlišujeme několik druhů učení: [28][23][35]

- **supervised learning (učení s učitelem)** - Učícímu algoritmu je poskytnuta datová sada společně s příslušnými značkami, které určují správný výsledek úlohy nad daty z datové sady. Pojmenování této skupiny učení s učitelem vyjadřuje právě tu skutečnost, že algoritmus má k dispozici anotovaná data.

  Jedná se o problém hledání funkce:

  $$f : D \to L. \tag{2.1}$$

  Tedy funkce f, která zobrazuje data z množiny dat D na značení z množiny značek L.

  Z této skupiny učících se algoritmů budou pocházet algoritmy pro klasifikaci, které si představíme dále v tomto textu. V kontextu klasifikační úlohy by datová sada obsahovala příznaky reprezentující entitu z dané domény (např. dokument či obrázek) a značení by odpovídalo příslušné třídě, do které entita patří.

- **unsupervised learning (učení bez učitele)** - Narozdíl od učení s učitelem neposkytujeme data společně s příslušným značením, ale pouze data samotná.

  Do této skupiny bychom mohli zařadit shlukování.

- **active learning (aktivní učení)** - Tento přístup zapojuje do procesu učení uživatele. Učící algoritmus může žádat uživatele, aby mu označil nějaký vzorek dat. Tento vzorek dat může být z datové sady neoznačených dat nebo vyprodukován algoritmem samotným.

[3] Spam filtr je program, který odfiltrovává nevyžádanou poštu, takzvaný spam.

Smyslem tohoto druhu učení je optimalizovat model tím, že získáváme znalosti od uživatele. Existují také omezení, která říkají kolik značení od uživatele můžeme maximálně požadovat.

- **reinforcement learning (zpětnovazební učení)** - Zpětnovazební učení, někdy též posilované učení, je druh učení, kdy autonomní agent[4] získává informace o svém prostředí a učí se vybírat akce, které maximalizují jeho odměnu.

  Agentovi dáváme odměnu nebo jej můžeme naopak trestat (negativní odměna) na základě jeho akcí a stavu, do kterého ho chceme dostat. Tímto mu poskytujeme zpětnou vazbu k jeho činům.

  Odměnu/trest nemusí agent získat hned po vykonání akce, ale můžeme ji odložit a dostane ji až po vykonání větší sekvence akcí. Například na konci odehrané hry, kdy jej můžeme odměnit kladnou odměnou za výhru, zápornou za prohru a nulovou za remízu.

  Příkladem algoritmu založeného na zpětnovazebním učení je Q learning.

- **semi-supervised learning** - V tomto případě se pracuje jak s označenými, tak neoznačenými daty. Příklad postupu může být následující. Označená data jsou použita pro naučení se modelů tříd a neoznačená data jsou použita k vylepšení hranice mezi těmito třídami.

- **self-supervised learning** - Jedná se o specifickou variantu učení s učitelem. Stejně jako učení s učitelem používá anotovanou sadu. Značení ovšem nepochází od vnějšího zdroje (člověka), ale generuje se pomocí heuristických algoritmů přímo z dat.

  Příkladem může být predikování následujícího slova z posloupnosti předcházejících slov v textu. Značením je zde odhadované slovo, které je součástí vstupních dat a tudíž se nejedná o explicitně přidanou značku.

2.2 Extrakce příznaků

Nežli si popíšeme samotnou extrakci příznaků, je vhodné si popsat co chápeme pod pojmem příznak (anglicky feature).

instance / vzorek / dokument	**třída**
... Česká republika, je stát ve střední Evropě ...	Zeměpis
... Slon je název pro druh chobotnatce ...	Biologie
... Obsah čtverce nad předponou je roven ...	Matematika

Tabulka 2.1: Příklad označených textových dokumentů v datové sadě.

Příznakem rozumíme nějakou měřitelnou charakteristiku či vlastnost entity/jevu, který zpracováváme. Pro ilustraci si představme, že máme datovou sadu s textovými dokumenty. Například takovou jako je v tabulce 2.1. Tuto sadu můžeme pro účely našeho dalšího zpracování reprezentovat pomocí množiny příznaků, jak ilustruje tabulka 2.2. Pokud si vezmeme libovolný atribut a k němu jednu z hodnot, která se pod daným atributem vyskytuje, pak dostáváme příznak. Příznakem tedy rozumíme kombinaci atributu a jeho hodnoty. Nyní se

[4] Agent je cokoliv co dokáže vnímat prostředí zkrze své senzory a působit na toto prostředí svými aktuátory. [38]

Atributy instance / vzorku / dokumentu					Třída
...	**republika**	**chobotnatec**	**čtverec**	**...**	
...	20	2	0	...	Zeměpis
...	1	10	0	...	Biologie
...	0	0	20	...	Matematika

Tabulka 2.2: Možná reprezentace dokumentů z tabulky 2.1 pomocí atributů představujících jednotlivá slova ve všech dokumentech. Hodnoty atributů udávájí počet výskytů daného slova v dokumentu/instanci.

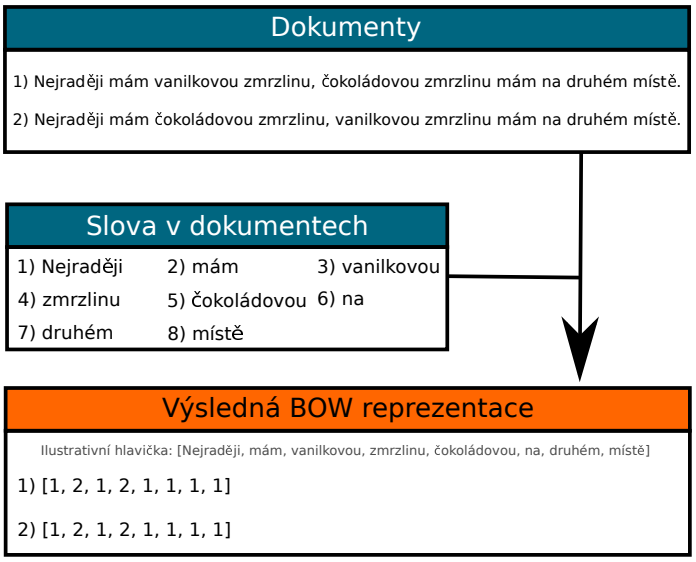
na věc podívejme znovu a více obecně. Příznak odpovídá predikátu F, pro který platí, že instance (dokument) x má daný příznak pouze v případě, kdy $F(x)$ je pravda. Dále mějme atribut odpovídající funkci h. Párování atributu h a hodnoty v odpovídá predikátu, který je pravdivý pouze v případě, kdy $h(x) = v$ a jinak je nepravdivý. [20] Přestože zde je příznak popsán jako kombinace atributu a hodnoty, v literatuře se také vyskytuje používání příznaku jako synonyma k atributu. [28]

Pod pojmem extrakce příznaků rozumíme proces redukce dimensionality, při kterém původní množinu prvotních dat redukujeme na více zvládnutelnou skupinu pro zpracování. Přičemž tato nová reprezentace stále přesně a úplně popisuje původní data (vzhledem k danému procesu). Tímto způsobem můžeme dostat kompaktnější reprezentaci snižující množství redundance v datech. [36] Hodnoty v tabulce 2.2 lze chápat jako výsledek extrakce příznaků provedené na instancích z tabulky 2.1.

V této části si představíme několik metod používaných pro extrakci příznaků z textu a obrázků.

2.2.1 Bag-of-words

První metoda, kterou si popíšeme, je Bag-of-words (zkráceně BOW). Jedná se o základní metodu pro extrakci příznaků z textu.



Obrázek 2.2: Ilustrace průběhu metody Bag-of-words.

Pracuje nad takzvanými termy (slovy) a zjišťuje u jednotlivých termů jejich počty výskytů v dokumentu. Počet výskytů daného slova představuje váhu slova, kterou označujeme anglickým názvem *term frequency*. Značím $tf_{t,d}$, kde t představuje daný term a d dokument, v němž se term vyskytuje v určeném počtu. [34]

Jak je možné vidět z příkladu na obrázku 2.2, tak tento přístup vůbec nehledí na pořadí slov a uvedené sémanticky odlišné věty mají ve výsledku stejnou reprezentaci.

Pokud chceme více zohlednit pořadí, v jakém se slova v dokumentu vyskytují, je možné použít n-gramy, jedná se o n-tice po sobě jdoucích n slov v textu. Namísto jednotlivých slov budeme například brát dvojice slov (bigramy). Pro uvedený první dokument v obrázku 2.2 dostáváme:

Nejraději mám, mám vanilkovou, vanilkovou zmrzlinu, ...

Nepočítali bychom výskyty samostatných slov, ale výskyty těchto dvojic v dokumentu.

2.2.2 TF-IDF

Podobně jako BOW (2.2.1) se i tato metoda používá pro extrakci příznaků z textu. Narozdíl od BOW nepoužívá pouze term frequency, ale pracuje i s pojmem inverse document frequency. Odtud také pochází název této metody: Term Frequency–Inverse Document Frequency (TF-IDF).

Inverse document frequency (IDF) představuje inverzní dokumentovou četnost, která má více vyjádření. Uveďme si zde alespoň jedno z nich: [34]

$$idf_t = log\frac{N}{df_t}. \tag{2.2}$$

Jedná se o zlogaritmovanou hodnotu podílu počtu všech dokumentů N a počtu dokumentů df_t, v nichž se vyskytuje alespoň jednou term (slovo) t. Nabývá vysokých hodnot pro vzácné termy (napříč dokumenty) a malé hodnoty pro běžné termy.

Term frequency $tf_{t,d}$, již zmiňované u BOW v 2.2.1, nemusí být pouze vyjádřeno jako počet výskytů daného slova. Můžeme například frekvence termu t podělit počtem slov v d, čímž získáváme normalizovanou variantu. [29] Existují však i další vyjádření jako je: [34]

$$wtf_{t,d} = \begin{cases} 1 + log(tf_{t,d}) & tf_{t,d} > 0 \\ 0 & jinak \end{cases}. \tag{2.3}$$

Konečně výslednou hodnotu tf-idf získáme:

$$tf\text{-}idf_{t,d} = tf_{t,d} \cdot idf_t. \tag{2.4}$$

Hodnota $tf\text{-}idf_{t,d}$ je vysoká v případě, když se term vyskytuje často v malém počtu dokumentů a naopak malá, když se term v dokumentu vyskytuje málo často nebo se vyskytuje v hodně dokumentech. Tedy například term, který se vyskytuje v jednom dokumentu frekventovaně a v ostatních se nevyskytuje, bude mít vysokou hodnout.

2.2.3 Hašování příznaků

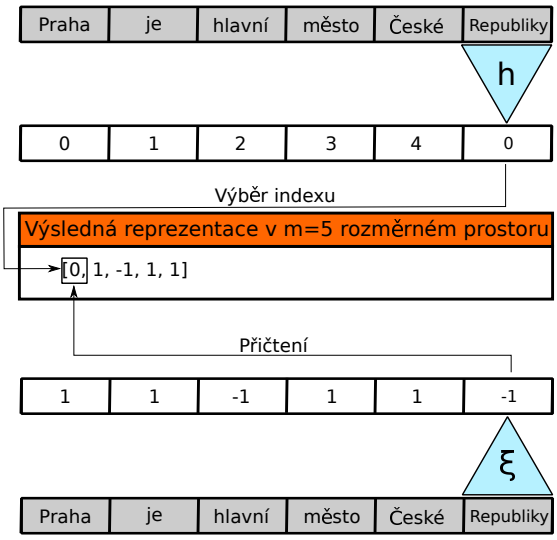
Podobně jako předchozí metody si i tuto představíme jako metodu pro extrakci příznaků z textu. Na rozdíl od předchozích přístupů zde nebudeme potřebovat vytvářet slovník slov,

které se v dokumentech vyskytují, a budeme mít možnost vytvářet příznakové vektory o velikosti, kterou si uživatel může sám zvolit. Hašování příznaků provádí zobrazení: [44]

$$f : X \to \mathbb{R}^m, \tag{2.5}$$

kde X je množina vstupních vektorů z prostoru R^d a R^m je výstupní prostor s tím, že předpokládáme, že $m \ll d$. Tedy provádíme převod z vysoce dimenzionálního prostoru do méně dimenzionálního.

Vzhledem ke zmiňovaným vlastnostem je Hašování příznaků, také známo jako hashing trick, vhodné pro velké objemy dat, které by byly náročné zpracovat metodami, jako je BOW a podobnými.



Obrázek 2.3: Hašování příznaků používající funkci h pro získání indexu i a funkci ξ udávající hodnotu, která bude přičtena k aktuální hodnotě na indexu i. Situace je zachycena po přičtení -1 k 1, která se nacházela na indexu 0 ve výsledném vektoru.

Hašování příznaků používá až dvě hashovací funkce. [44] [43] Jedna, která je použita vždy, transformuje term na vstupu na index do příslušného výsledného vektoru, kde na získaném indexu dojde k přičtení (případně odečtení) jedničky. Jedná se o funkci:

$$h : \mathbb{N} \to \{0, 1, ..., m\}. \tag{2.6}$$

Druhá (volitelná) funkce se používá pro určení znaménka hodnoty pro přičtení. Určuje tedy, zdali pro daný term bude jednička přičtena či odečtena. Tato hashovací funkce je:

$$\xi : \mathbb{N} \to \{-1, 1\}. \tag{2.7}$$

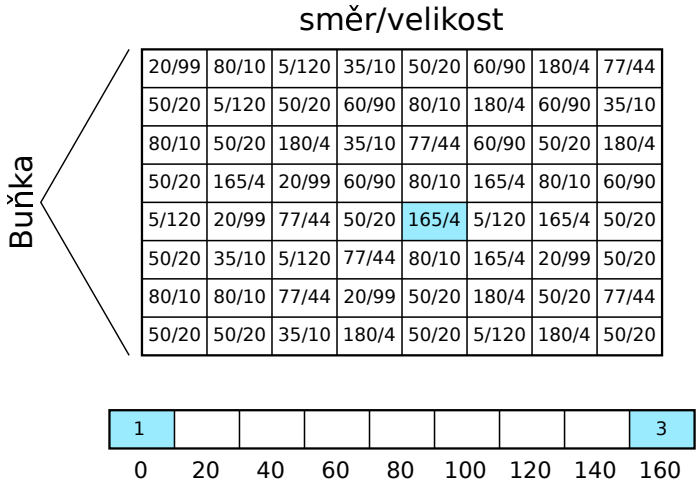
Zmiňovanou hašovací funkci je vhodné použít kvůli zmírnění vlivu kolizí. Příklad kolize je vidět v ilustrativním obrázku 2.3. Použitím hashovacích funkcí se totiž vystavujeme riziku, že se dvě rozdílná slova mohou namapovat na stejný index.

Přestože zde definujeme obě funkce nad přirozenými čísly, v praxi chceme pracovat hlavně s textovými řetězci. Nejedná se ovšem principiálně o problém, jelikož každý konečný řetězec je možné reprezentovat přirozeným číslem. [44]

2.2.4 Histogram orientovaných gradientů

Histogram orientovaných gradientů (v angličtině Histogram of Oriented Gradients), často označovaný jen jako HOG je metoda pro získání vektoru příznaků z obrazu. Skládá se z několika navazujících kroků: [24] [33]

- **Předzpracování** - V této fázi se jedná pouze o předupravení si vstupních dat. Provádí se zde operace, jako jsou například: ořez obrazu, škálování obrazu či gama korekce.
- **Výpočet gradientů** - Vypočítáme si gradient g_x v ose x a g_y v ose y. Toho lze docílit použitím obrazového filtru s maskou



Obrázek 2.4: Přičítání hodnot do košů histogramu gradientů. Modrou barvou je zvýrazněn gradient v buňce, který aktuálně zpracováváme. Tento gradient má směr s hodnotou 165°a velikost 4. Hodnotu 4 rozdělíme do koše označeného 0 a 160 (uvažujeme napojení konce a začátku). Rozdělení provedeme proporcionálně vzhledem ke vzdálenosti (160: 75%, 0: 25%).

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

pro osu x a s maskou

$$\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

pro osu y. Z vypočítaných hodnot získáme velikost

$$g = \sqrt{g_x^2 + g_y^2} \tag{2.8}$$

a směr gradientu:

$$\theta = \arctan \frac{g_y}{g_x}. \tag{2.9}$$

Pro obrázky s více barevnými kanály vypočítáme gradienty pro každý kanál zvlášť a nakonec vybereme kanál s největší velikostí gradientu, jehož velikost a směr přiřadíme danému pixelu. Výsledky jsou vizualizovány na obrázku 2.5.

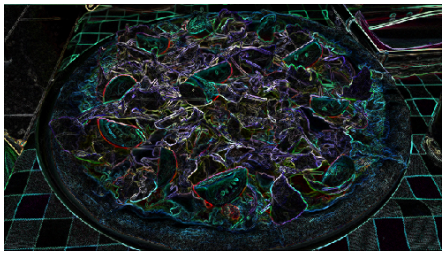
- **Prostorové/směrové plněni košů** - V tomto kroku vytvoříme histogram gradientů. Nejprve si rozdělíme vstupní obraz na buňky 8x8 pixelů, velikost lze nastavovat. Pro každou takovou buňku budeme tvořit histogram, jak je naznačeno na obrázku 2.4. V ilustraci je použito 9 košů na jeden histogramu, které rovnoměrně dělí interval stupňů určujících orientaci gradientu. Používáme interval od 0°-180°, což je interval pro takzvaný bez znaménkový gradient. Znaménkový gradient používá interval 0°-360°.

- **Normalizace bloků** - Normalizace se provádí na úrovni bloků. Bloky jsou tvořeny buňkami. Velikost bloku je volitelná. Můžeme například zvolit velikost 2x2 buňky (16x16 pixelů). S používanou konfigurací v příkladech pak normalizujeme vektor o velikosti 36x1 (9*4=košů*buněk v bloku). Samotný blok pak získáme tak, že použijeme posuvné okno, které se pro každý nový blok posune o jednu buňku.

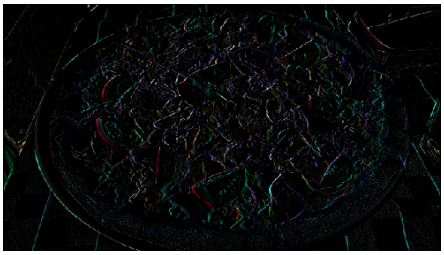
  Používaná je L2 (L2-Hys) normalizace.

- **Výpočet příznakového vektoru** - Výsledný příznakový vektor získáme konkatenací dílčích vektorů. Vizualizaci výsledku je možné si prohlédnout na obrázku 2.5e. Objekt z původního obrázku je zde stále patrný.

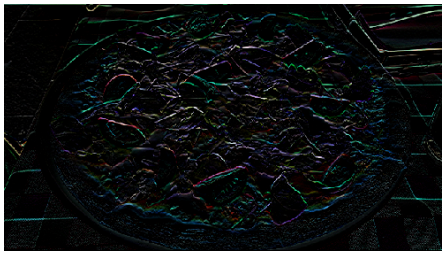
(a) původní obrázek



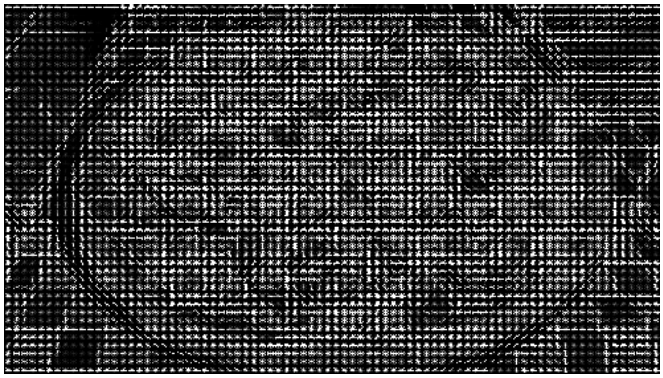
(b) Vizualizace velikosti gradientu.



(c) Vizualizace gradientu v ose x.



(d) Vizualizace gradientu v ose y.



(e) Vizualizace výsledku HOG.

Obrázek 2.5: Průběh výpočtu Histogramu orientovaných gradientů.

2.3 Selekce atributů

Máme-li větší množství atributů je vhodné předřadit krok selekce atributů před samotnou klasifikaci. Při selekci se snažíme odstranit atributy, které nevedou ke zkvalitnění výsledků klasifikace. Přitom se snažíme, aby rozdělení pravděpodobnosti nových dat bylo co nejpodobnější rozdělení původních data.

Díky tomu, že vynecháme některé atributy můžeme získat lepší výsledky a zkrátit čas, který je nutný pro natrénování klasifikátoru a i k samotné klasifikaci.

Pokud má náš příznakový vektor n dimenzí, tak všech různých podmnožin jeho atributů je 2^n. Procházení takového potencionálně velkého množství možností by bylo velmi výpočetně náročné, proto se volí strategie, které neprocházejí tyto možnosti všechny. Takovýchto strategií existuje několik a my si zde uvedeme alespoň některé reprezentanty. [28]

2.3.1 Postupná dopředná/zpětná selekce

Postupná dopředná selekce začíná s prázdnou množinou atributů. Nejprve vybere atribut, který je nejlepší a dále k němu přidá atribut, který je nejlepší ze zbylých a takto pokračuje.

Zpětná selekce pracuje opačným způsobem. Začíná se všemi atributy a nejprve odstraní atribut, který je ze všech pro klasifikaci nejhorší. V dalším kroku odstraní nejhorší ze zbylých.

Kritérium, které určuje jak je daný atribut dobrý (množina atributů) lze získat vícero způsoby. Například přímo natrénováním modelu danou podmnožinou příznaků a vyhodnocením úspěšnosti křížovou validací nebo vyhodnocovat každý atribut zvlášť, například pomocí metriky Information Gain, a vždy přidávat/ubírat atribut s nejlepší/nejhorší hodnotou.

Ukončit popsané algoritmy můžeme například pokud nepozorujeme v dalších krocích zlepšení. [28][42]

2.3.2 Selekce založená na varianci

Jedná se o poměrně přímočarou metodu, kdy si vypočítáme varianci každého atributu a na základě zvoleného prahu určíme, zdali jej vyhodíme nebo ponecháme.

Například tedy můžeme vyřadit všechny atributy jejichž příznaky mají nulovou varianci a tedy ty, které jsou pro všechny vzorky stejné. [5]

2.3.3 Selekce založená na rozhodovacím stromě

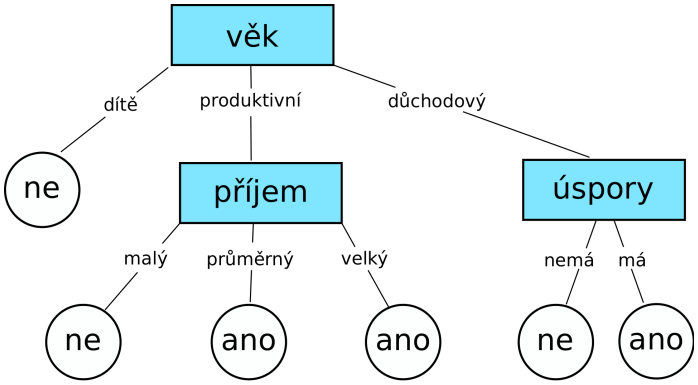
U tohoto přístupu můžeme provést indukci rozhodovacího stromu (více o indukci 2.4.3) a vybrat pouze ty atributy, které se ve stromě nacházejí. [28]

Dále je možné, pomocí používaných metrik při indukci stromu, měřit důležitost daného atributu a na základě takto získaných hodnot vyloučit atributy nesplňující zvolený práh. [5]

2.4 Rozhodovací strom

Klasifikaci pomocí rozhodovacího stromu, lze rozdělit na dvě části:

1. Indukce rozhodovacího stromu. Jedná se o vytvoření klasifikačního modelu.
2. Použití vytvořeného rozhodovacího stromu na vstupní data.



Obrázek 2.6: Příklad rozhodovacího stromu.

Indukce rozhodovacího stromu je náročnější proces nežli samotné použití. Z tohoto důvodu si zde nejprve popíšeme rozhodovací strom a jak jej použít ke klasifikací a až poté si ukážeme jakým způsobem je možné jej získat.

2.4.1 Klasifikace pomocí rozhodovacího stromu

Na obrázku 2.6 je znázorněna možná podoba rozhodovacího stromu. Z pohledu teorie grafu se jedná o strom. Každý vnitřní uzel (nelistový) představuje test nad daným atributem, jimž je konkrétní uzel pojmenován. Hrany připojující potomky daného uzlu pak znázorňují výstup tohoto testu. Konečně listové uzly představují výsledné třídy, do kterých vstupní vzorek klasifikujeme. [28]

Představme si, že obrázek 2.6 znázorňuje rozhodovací strom, který rozhoduje, zda si daný člověk pravděpodobně zaplatí dovolenou v zahraničí. Mějme člověka, kterého chceme klasifikovat, s těmito atributy na vstupu:

věk: produktivní, příjem: průměrný.

Kořenovým uzlem stromu je věk. První test tedy provedeme na věk, protože náš člověk je v produktivním věku, přesuneme se dle stromu na další test, který rozřazuje dle příjmu. Na základě toho, že máme člověka s průměrným příjmem, dostáváme výsledek v podobě třídy ano. V našem příkladu to tady znamená, že si dovolenou pravděpodobně koupí.

Jak tedy probíhá klasifikace obecně? Na vstupu máme n-tici X, která nemá přiřazenou značku, do které třídy patří. Hodnoty atributů n-tice X jsou otestovány vzhledem k rozhodovacímu stromu a to v pořadí od kořene až ke konkrétnímu listovému uzlu, který nese výslednou značku třídy. Hodnoty atributů mohou být jak diskrétní tak spojité. [28]

2.4.2 Vyjadřovací schopnost rozhodovacího stromu

Mějme binární rozhodovací strom, tedy strom, který nám na vstupní n-tici X dává binární odpověď (pravda/nepravda). Takovýto strom je logicky ekvivalentní tvrzení, že cílový atribut/značení ($Goal$) je pravda tehdy a jen tehdy, když vstupní n-tice X splňuje jednu z cest vedoucích od kořene k listu s hodnotou pravda. V jazyce výrokové logiky můžeme tuto skutečnost vyjádřit následovně:

$$Goal \iff (Path_1 \vee Path_2 \vee Path_3 \vee ...). \tag{2.10}$$

Kde každá cesta ($Path$) je konjunkcí testů hodnot atributů, které se nacházely na dané cestě. Vezměme si příklad cesty uvedené v textu výše pro strom na obrázku 2.6:

$$Path = (věk = produktivní \wedge příjem = průměrný). \tag{2.11}$$

Celkově tedy dostáváme výraz, který je ekvivalentní disjunktivní normální formě, což znamená, že libovolná funkce výrokové logiky může být vyjádřena pomocí rozhodovacího stromu. [38]

2.4.3 Indukce rozhodovacího stromu

Nyní si popíšeme jakým způsobem je možné vytvořit rozhodovací strom. Ukážeme si algoritmus, který patří do třídy algoritmů učení s učitelem a budeme tedy požadovat označenou trénovací sadu. Základní průběh je naznačen v pseudokódu Algoritmu 1.

Algoritmus 1 Generování rozhodovacího stromu z trénovacích dvojic, pro část dat D. [28][38]

Vstup:

- Část dat D, což je množina označených n-tic trénovacích dat.
- List kandidátních atributů: $attributeList$.
- Procedura pro výběr atributu $attributeSelectionMethod$ určující rozdělovací kritérium, které „nejlepším“ způsobem rozděluje vstupní data do jednotlivých tříd. Toto kritérium se skládá z rozdělovacího atributu ($splittingAttribute$) a případně $split-point$ nebo $splittingsubset$.

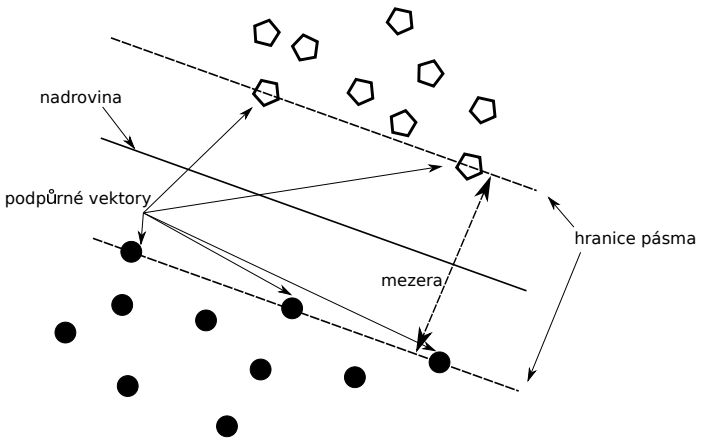
Výstup: rozhodovací strom

1: **Procedura:** GennerateDecisionTree
2: vytvoř uzel N;
3: **if** n-tice v D patří všechny do stejné třídy C **then**
4: **return** N jako listový uzel tříd C;
5: **if** $attributeList$ je prázdný **then**
6: **return** N jako listový uzel převažující třídy C v D;
7: aplikuj $attributeSelectionMethod(D, attributeList)$ pro nalezení rozdělovacícho kritéria;
8: označ uzel N rozdělovacím kritériem; //atributem
9: $attributeList \leftarrow attributeList - splittingAttribute$
10: **for all** i **in** rozdělovací kritérium **do**
11: nechť D_i je množina n-tice z D vyhovujících i;
12: **if** D_i je prázdné **then**
13: připoj list k uzlu N a označ ho převažující třídou C v D_i;
14: **else**
15: připoj uzel vrácený z $GennerateDecisionTree(D_i, attributeList)$ k uzlu N hranou označenou i;
16: **return** N

Procedura, v Algoritmu 1 označovaná jako $attributeSelectionMethod$, vybírá nejvhodnější atribut pro testování na základě výběrové metriky. V literatuře je popsáno více druhů takovýchto metrik: Information Gain, Gain Ratio či Gini Index. [28]

2.5 Support Vector Machines

Support Vector Machines, často označované pouze jen jako SVM, je klasifikační metoda pracující s nadrovinou[5], která rozděluje vzorky dvou tříd ve vektorovém prostoru. Tato metoda je použitelná jak na lineárně separovatelná (lze přímo rozdělit nadrovinou), tak nelineárně separovatelná data. Ukážeme si jakým způsobem funguje na datech, která jsou lineárně separovatelná.



Obrázek 2.7: Ilustrace Support Vector Machines.

Jednoduchý příklad je vidět na obrázku 2.7. Máme zde pouze dvě třídy, které chceme klasifikovat, a navíc jsou (jak požadujeme) lineárně separovatelné. Každý vzorek máme v podobě dvojice $(\vec{x}_i, y_i)$. Kde $\vec{x}_i$ představuje vektor reprezentující jeden bod v prostoru a y_i je označení třídy. Protože uvažujeme pouze dvě možné třídy, tak $y_i \in \{-1, 1\}$. Důvod proč jsme použili právě tato dvě čísla, bude zřejmý po představení podoby klasifikátoru, nežli však ukážeme jeho podobu je nutné si prvně uvést matematickou reprezentaci nadroviny:

$$\vec{w}^T \vec{x} + b = 0. \tag{2.12}$$

Kde $\vec{w}$ je váhový vektor, jedná se o normálu nadroviny. $\vec{x}$ je bod ležící přímo na nadrovině a b je konstanta. Samotný klasifikátor je pak vyjádřen: [34]

$$f(\vec{x}) = sgn(\vec{w}^T \vec{x} + b). \tag{2.13}$$

Díky funkci signum pak dostáváme -1 pro jednu a 1 pro druhou třídu. Pohleďme znovu na obrázek 2.7. Je vidět, že zde vystupují tři nadroviny. Dvě hraniční a samotná dělící nadrovina. Všechny nadroviny jsou navzájem rovnoběžné a dělící nadrovina je stejně vzdálená od obou hraničních. Vektory ležící na jedné z hraničních nadrovin nazýváme podpůrné vektory (support vectors). SVM se snaží hledat dělící nadrovinu, která by nejlépe generalizovala, tedy měla nejlepší výsledky na nových neznámých datech. Za takovou dělící nadrovinu považuje nadrovinu, která má maximální odstup od hraničních nadrovin. Na obrázku 2.7 bychom našli nekonečné množství nadrovin, které by mohly dělit prostor na dvě části, my však vybereme právě tu s největším odstupem. [28]

Hledáním takovéto nadroviny řešíme následující optimalizační problém [34]. Najdi $\vec{w}$ a b že:

[5] V euklidovském prostoru nadrovina dělí prostor na dva poloprostory. Ve 2D se jedná o přímku, ve 3D o rovinu, atd.

- $\frac{1}{2}\vec{w}^T\vec{w}$ je minimální
- a pro všechny $\{(x_i,y_i)\}$: $y_i(\vec{w}^T\vec{x}_i + b) \geq 1$

Jedná se o optimalizační problém z kvadratického programování.

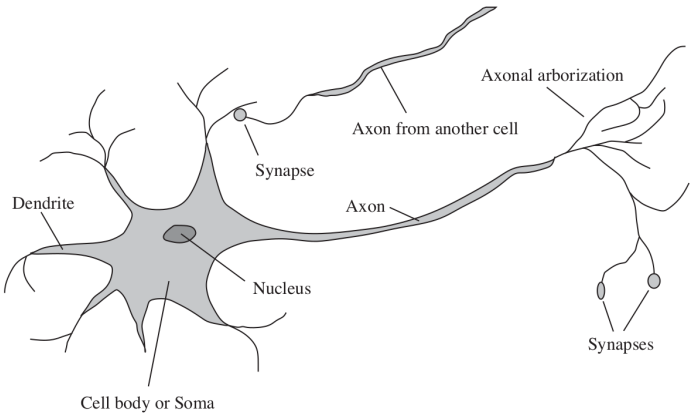
Řešení případu, kdy data nejsou lineárně separovatelná se provádí ve dvou krocích. Nejprve převedeme vstupní data do prostoru s vyšší dimenzí, kde jsou data lineárně separovatelná, toho lze vždy pro dvě třídy docílit. Po převodu, již stejně jako předtím, hledáme dělící nadrovinu v tomto novém prostoru. [28]

2.6 Neuronové sítě

Nyní se podíváme na biologií inspirovaný model, a to umělé neuronové sítě. Ukážeme si, jak se liší od svého biologického podkladu, jejich základní stavební prvek, použití neuronových sítí pro klasifikaci a nastíníme průběh učení tohoto modelu.

2.6.1 Biologická inspirace

Obor neuronových sítí má své počátky v neurobiologii a psychologii [28]. Poprvé v 18. století se mozek začal široce uznávat jako centrum vědomí. Předtím se jako kandidátní orgány uvažovaly srdce a slezina.



Obrázek 2.8: Ilustrace neuronu. Převzato z [38]

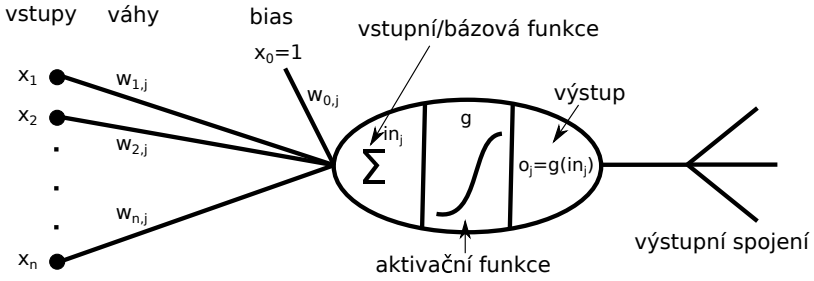
.

Mozek samotný se skládá z nervových buněk zvaných neurony. Neurony se navzájem spojují s 10 až 100 000 jinými neurony pomocí spojů, které nazýváme synapse. V síti těchto buněk dochází k přenosu signálu mezi neurony pomocí komplikované elektrochemické reakce. [38]

Příklad neuronu je vidět na obrázku 2.8.

2.6.2 Perceptron

Perceptron je základní jednotkou neuronové sítě (nejjednodušší síť). Jedná se o jednoduchý matematický model pro neuron. Jeho schémantické znázornění je možné vidět na obrázku 2.9.



Obrázek 2.9: Model neuronu: perceptron.

Do perceptronu vede na vstup až n spojení x_i plus bias, každé z těchto spojení má přiřazenou váhu $w_{i,j}$. Tato váha určuje pomocí číselné hodnoty váhu a znaménko (+/-) daného spojení. Perceptron pracuje tak, že nejprve spočítá vážený součet svých vstupů:

$$in_j = \sum_{i=0}^{n} w_{i,j} x_i, \tag{2.14}$$

poté aplikuje aktivační funkci g, na výsledek tohoto součtu, pro získání výstupu:

$$o_j = g(in_j) = g(\sum_{i=0}^{n} w_{i,j} x_i). \tag{2.15}$$

Aktivační funkce g může mít více podob. Typicky se může jednat o skokovou aktivační funkci s výstupem 0/1 (časté bývá i -1/1):

$$g(in_j) = \begin{cases} 1 & in_j = \sum_{i=0}^{n} w_{i,j} x_i > 0 \\ 0 & jinak \end{cases}, \tag{2.16}$$

pak se jedná o standardní perceptron či:

$$g(in_j) = \frac{1}{1 + e^{-in_j}}, \tag{2.17}$$

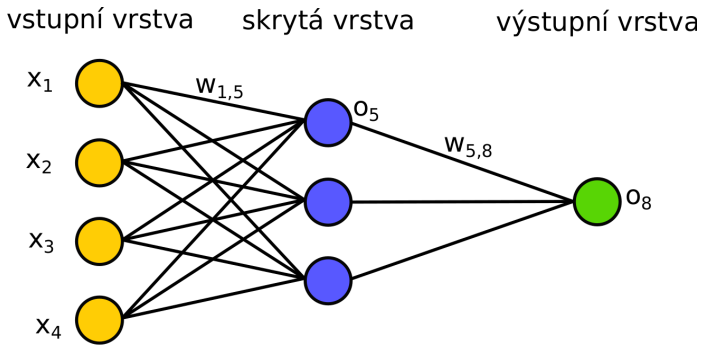
pak mluvíme o sigmoidovém perceptronu. Samotný perceptron pracuje jako lineární klasifikátor a není tedy schopen se naučit lineárně neseparovatelné problémy. Pro tyto druhy problémů se používají neuronové sítě s více jednotkami (perceptrony) ve více vrstvách. [38][28]

2.6.3 Klasifikace pomocí neuronových sítí

V této sekci si představíme jakým způsobem je možné použít umělou neuronovou síť pro klasifikační problémy a také si popíšeme samotnou síť.

Každá neuronová síť se skládá ze vstupní vrstvy, výstupní vrstvy a libovolně z jedné až několika skrytých vrstev. Vstupní vrstva pouze představuje vstupní data, která se předkládají síti. Nejedná se tedy o žádné výpočetní jednotky, jako je perceptron (na rozdíl od dalších vrstev). Výstupní vrstva poskytuje výsledek neuronové sítě.

Příklad neuronové sítě je možné vidět na obrázku 2.10. Na tomto obrázku je zobrazena dvouvrstvá (vstupní se nepočítá) dopředná neuronová síť. Dopředná síť znamená, že



Obrázek 2.10: Příklad dvouvrstvé plně propojené dopředné umělé neuronové sítě.

máme spojení mezi jednotkami pouze do další nejbližší vrstvy ve směru k výstupní vrstvě. Plně propojená znamená, že existují spojení od každé jednotky dané vrstvy do každé další jednotky v následující vrstvě. Existuje několik druhů architektur neuronových sítí, které kladou různá omezení na propojení mezi jednotkami v síti, tato práce se však zaměřuje na plně propojené dopředné neuronové sítě.

Jak tedy použít neuronovou síť pro klasifikaci? První věcí, kterou je nutné provést je stanovení topologie sítě, tedy počet vrstev a počet neuronů v jednotlivých vrstvách. Jednotlivá propojení mezi jednotkami máme pevně daná, protože používáme plně propojené dopředné neuronové sítě.

Kolik jednotek použijeme ve vstupní vrstvě se odvíjí od vstupních dat. Neuronová síť očekává na vstupu číselné hodnoty, které se navíc často normalizují do intervalu <0,1> pro urychlení učení. Pro spojité hodnoty můžeme přiřadit každému vstupnímu atributu jednu jednotku. Pokud pracujeme s diskrétními hodnotami atributu, je možné pro každou diskrétní hodnotu (např. jablko, hruška) přiřadit jednu jednotku. Pak pokud je vstup roven příslušné hodnotě, kterou jednotka reprezentuje, nastavíme ji na hodnotu 1 a ostatní na hodnotu 0. Tedy pro příklad jablko-hruška nastavíme jednotku příslušející jablku na hodnotu 1 a druhou jednotku na hodnotu 0, při hodnotě atributu rovnou jablko.

Počet skrytých vrstev a neuronů v nich se nastavuje různě dle řešeného problému, většinou metodou pokus-omyl.

Kolik jednotek použijeme ve výstupní vrstvě se odvíjí od počtu tříd, do kterých klasifikujeme. Pokud máme pouze dvě třídy, stačí nám pouze jeden výstupní neuron. Kdy hodnota 0 značí jednu třídu a hodnota 1 druhou. Máme-li však více tříd, můžeme přiřadit každé třídě jednu výstupní jednotku.

Po stanovení topologie sítě následuje její učení, získání hodnot vah. Standardně se jedná o učení algoritmem backpropagation. Tento algoritmus ve zkratce pracuje tak, že na začátku zvolí nějaké hodnoty vah sítě. Dále pracuje s trénovací sadou dat, kterou přivádí na vstupy sítě a sleduje výstupní odezvu sítě, kterou porovnává s referenčními hodnotami (značení daných tříd). Zjišťuje chybu, rozdíl mezi tím co má síť na výstupu a referenčními hodnotami, a tuto chybu dále zpětně šíří a na základě toho upravuje váhy mezi spojeními. Tento proces se opakuje v několika iteracích a může být završen v momentu, kdy je splněno alespoň jedno z následujících:

- Dosáhne maximálního počtu iterací.
- Počet špatně klasifikovaných označených vzorků klesne pod předem určený práh.

- Všechny změny vah klesnou pod hodnotu daného prahu.

Není nutné upravovat váhy pro každý vzorek trénovací množiny. Váhy můžeme upravit až po zpracování celé trénovací sady (či její části).

Natrénovaná síť je již připravena k použití. Platí, že trénování je značně složitější proces, nežli použití sítě na klasifikaci neoznačených dat. [28]

2.7 K-nejbližších sousedů

K-nejbližších sousedů, často označované pouze jen jako kNN (z anglického k nearest neighbor), patří mezi metody s takzvaným **líným učením** (lazy learning) To znamená, že ve své fázi trénování si nevytváří model dat, který je může nějakým způsobem generalizovat. Takovýto model si vytváří metody používající **dychtivé učení** (eager learning), mezi takovéto metody patří například: SVM, neuronové sítě, rozhodovací stromy a další.

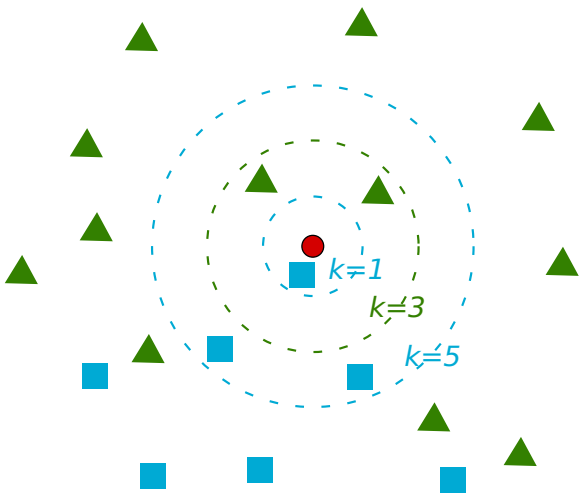
Metody s líným učením si pouze uloží trénovací data a ve fázi klasifikace provedou generalizaci. Vyhodnocení poté probíhá na základě podobnosti k trénovacím datům.

Pro kNN je podobnost měřena pomocí vzdálenosti. Vycházíme z hypotézy souvislosti, tedy že klasifikovaný vzorek patří do stejné třídy jako trénovací vzorky, které ho obklopují.

Každý příznakový vektor lze chápat jako bod v n rozměrném prostoru. Pro daný příznakový vektor, jehož třídu chceme určit, prohledáváme prostor a hledáme k nejbližších trénovacích vektorů. Vzdálenostní metriku můžeme zvolit například Euklidovu:

$$d(\boldsymbol{X_1}, \boldsymbol{X_2}) = \sqrt{\sum_{i=1}^{n}(x_{1i} - x_{2i})^2}. \tag{2.18}$$

Kde $\boldsymbol{X_1} = (x_{11}, x_{12}, \ldots, x_{1n})$ a $\boldsymbol{X_2} = (x_{21}, x_{22}, \ldots, x_{2n})$ jsou dva příznakové vektory, které porovnáváme.



Obrázek 2.11: Ilustrace chování kNN pro různé volby parametru k. Chceme klasifikovat příznakový vektor, který je ilustrován červeným kolečkem do jedné ze dvou tříd (zelená,modrá). Čárkované soustředné kružnice znázorňují svoji barvou, do které ze tříd by byl s daným parametrem vzorek zařazen.

Výsledné rozhodnutí je poté provedeno pomocí majority a vítězí tedy ta třída, do které patří nejvíce z k nejbližších vzorků. Volba k je závislá na problému a doporučuje se používat liché k větší než jedna. Liché z důvodu minimalizace kolize při výběru třídy na základě majority a větší než jedna, protože rozhodnutí pouze na základě nejbližšího souseda se nepovažuje za příliš robustní.

Také není dobré volit příliš velké k, protože může dojít vlivem majority k zastínění pravé třídy. Vezměme si například volbu $k = 5$ v ilustraci 2.11. Předpokládejme, že vzorky modré třídy spadající do příslušné kružnice představují pouze šum, který přesahuje do regionu druhé třídy a tedy vzorek bude nesprávně klasifikován. Optimální hodnotu k je možné zjišťovat experimentálně na dodané trénovací množině.

Je dobré zmínit, že trénování kNN je rychlejší než samotné použití ke klasifikaci, což je opačná situace než u klasifikátorů založených na dychtivém učení. V trénovací fázi kNN pouze uloží, případně předzpracuje, trénovací vzorky. V testovací fázi poté musí projít všechny uložené vzorky a změřit vzdálenost k testovacímu vzorku. Dostáváme tedy časovou složitost $\mathcal{O}(S)$, kde S je počet trénovacích vzorků. Dobré ovšem je, že čas potřebný pro testování není závislý na počtu tříd, jako u jiných klasifikátorů.

Existují různé modifikace tohoto algoritmu. Je například možné přiřazovat jednotlivým k nejbližším vzorkům váhu, která je odvozena od vzdálenosti k testovacímu vzorku a tuto váhu poté zohlednit při výběru třídy. [28] [34]

2.8 Naivní Bayesův klasifikátor

V této části si představíme statistický klasifikátor, který je schopen určovat pravděpodobnost příslušnosti příznakového vektoru (vzorku) do dané třídy. Rozeberme si zde název Naivní Bayesův klasifikátor, protože hodně vypovídá o tom, jak tento klasifikátor pracuje. V souvislosti s tímto klasifikátorem mluvíme o Bayesovské klasifikaci, která je založena na Byesově teorému[6], proto Bayesův klasifikátor. Přízvisko naivní dostal proto, že předpokládá nezávislost mezi jednotlivými atributy. Tato vlastnost jistě vždy neplatí. Nicméně toto zjednodušení usnadňuje výpočet.

Nežli si více rozepíšeme, jakým způsobem Naivní Bayesův klasifikátor pracuje, uvedeme si zde Bayesův teorém v kontextu klasifikace.

Nechť $\boldsymbol{X}$ je příznakový vektor, nazýváme jej evidence. Dále nechť C je hypotéza, že daný příznakový vektor patří do třídy c. Při klasifikaci, pak chceme určit podmíněnou pravděpodobnost, že příznakový vektor patří do dané třídy, tedy $P(C|\boldsymbol{X})$. Tuto pravděpodobnost nazýváme posteriorní. Obdobně mějme i posteriorní pravděpodobnost $P(\boldsymbol{X}|C)$, vyjadřující pravděpodobnost, že dostaneme daný příznakový vektor ve třídě c.

Dále pracujeme s pojmem priorní pravděpodobnost. Priorní pravděpodobnost nazýváme pravděpodobnost $P(C)$. Jedná se tedy o pravděpodobnost dané třídy bez ohledu na příznakový vektor.

Nyní, když si položíme otázku, jakým způsobem získáme zmiňované pravděpodobnosti z trénovacích dat, zjistíme, že všechny výše uvedené pravděpodobnosti, minimálně tedy ty, které budeme opravdu pro klasifikaci potřebovat, dokážeme přímo z trénovacích dat vypočítat (jakým způsobem si uvedeme dále). Pro posteriorní pravděpodobnost $P(C|\boldsymbol{X})$, tedy tu vyjadřující otázku, kterou si při klasifikaci pokládáme, použijeme Bayesův vzorec. Ten můžeme jednoduchým způsobem získat ze vzorce pro podmíněnou pravděpodobnost:

[6] Také se používá název Bayesova věta.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}. \tag{2.19}$$

Kde A a B jsou dva náhodné jevy. Z toho dále platí:

$$P(A \cap B) = P(A|B)P(B), \tag{2.20}$$

tedy:

$$P(A|B)P(B) = P(B|A)P(A). \tag{2.21}$$

Z tohoto vyjádření již snadno dostáváme Byesův vzorec, o kterém mluví následující Bayesův teorém. Nechť A a B jsou dva náhodné jevy a jejich pravděpodobnosti jsou P(A) a P(B), a pokud P(B) > 0 potom platí:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{2.22}$$

Přepíšeme si tento vzorec do podoby pro náš klasifikační problém a získáme:

$$P(C|\boldsymbol{X}) = \frac{P(\boldsymbol{X}|C)P(C)}{P(\boldsymbol{X})}. \tag{2.23}$$

Nyní se již můžeme podívat na to, jakým způsobem pracuje Naivní Bayesův klasifikátor.

Mějme trénovací sadu D skládající se z příznakových vektorů $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$. Dále nechť C je množina tříd $c_1, c_2, \ldots, c_m$, do kterých chceme klasifikovat. Klasifikátor pak určí, že vzorek s příznakovým vektorem $\boldsymbol{X}$ patří do třídy c_i tehdy a jen tehdy když:

$$\forall c_j \in C : P(c_i|\boldsymbol{X}) > P(c_j|\boldsymbol{X}) \wedge c_i \neq c_j \tag{2.24}$$

Tedy hledáme třídu s maximální posteriovou pravděpodobností:

$$c_i = \arg\max_{c \in C} P(c|\boldsymbol{X}) \tag{2.25}$$

Jak již bylo řečeno, tuto pravděpodobnost získáme za pomocí Bayesova vzorce. Nejprve však musíme vypočítat ostatní pravděpodobnosti, ze kterých tuto získáme. Protože $P(\boldsymbol{X})$ je konstantní pro všechny třídy a my hledáme maximum, tak si můžeme dovolit tuto pravděpodobnost zanedbat.

Pravděpodobnost $P(c_i)$ můžeme získat z trénovacích dat následovně:

$$P(c_i) = \frac{|D_{c_i}|}{|D|} \tag{2.26}$$

Kde:

- D_{c_i} je počet trénovacích vzorků patřících do třídy c_i.
- D je počet trénovacích vzorků celkem.

Pro získání pravděpodobnosti $P(\boldsymbol{X}|c_i)$ použijeme náš naivní předpoklad nezávislosti atributů a dostáváme:

$$P(\boldsymbol{X}|c_i) = \prod_{k=1}^{n} P(x_k|c_i) \tag{2.27}$$

Výpočet $P(x_k|c_i)$ se odvíjí od toho, zdali máme atribut s kategorickými hodnotami nebo spojitými.

Pokud je x_k hodnota kategorického atributu (např. paleta barev), pak:

$$P(x_k|c_i) = \frac{N_{vi}}{D_{c_i}}. \tag{2.28}$$

Kde N_{vi} značí počet vzorků ve třídě c_i s hodnotou x_k u příslušného atributu.

Pro atribut se spojitými hodnotami předpokládáme, že se řídím nějakým daným rozložením pravděpodobnosti typicky Gaussovým. Uvažujme tedy Gaussovo rozložení, to znamená, že budeme muset dopočítat střední hodnotu μ_{c_i} a směrodatnou odchylku σ_{c_i} hodnot uvažovaného atributu pro vzorky v dané třídě. Poté pravděpodobnost dostaneme pomocí:

$$P(x_k|c_i) = \frac{1}{\sigma_{c_i}\sqrt{2\pi}} e^{-\frac{(x_k-\mu_{c_i})^2}{2\sigma_{c_i}^2}} \tag{2.29}$$

Nyní jsme si popsali jakým způsobem pracuje Naivní Bayesův klasifikátor. [28][34][2]

Na závěr si však ještě uveďme dva praktické problémy. Pokud se podíváme na rovnici 2.27, tak uvidíme, že pro vektory s velkou dimenzí dochází k násobení velkého množství malých čísel, což může způsobit podtečení v plovoucí řádové čárce[7]. Je tedy vhodné při implementaci raději použít následující podobu s logaritmem: [34]

$$P(\boldsymbol{X}|c_i) = \sum_{k=1}^{n} \log P(x_k|c_i) \tag{2.30}$$

Další možnou obtíží jsou nulové hodnoty $P(x_k|c_i)$, které se vyskytují v případě, že daná hodnota (x_k) atributu se nevyskytuje ve vzorcích trénovací sady patřících pod c_i. Tedy vezmeme-li rovnici 2.28, tak N_{vi} bude nulové. Tím ovšem vzniká problém pro rovnice 2.27 a 2.30. U první dostaneme celkovou hodnotu 0 a u druhé počítáme logaritmus z 0. Možné řešení poskytuje Laplaceova korekce. Při použití Laplaceovi korekce přičteme ke každé N_{vi} hodnotu 1 a hodnotu odpovídající sumě všech přičtených jedniček nezapomeneme přičíst k hodnotě jmenovatele D_{c_i}.

Například mějme třídu jablka s $D_{c_i} = 100$ a hodnoty N_{vi} rovny 0, 60 a 40, pro možné hodnoty žlutá, červená a zelená u atributu barva. Kdybychom nepoužili zmiňovanou korekci a přišlo žluté jablko, tak pouze na základě tohoto jednoho atributu řekneme, že se o jablko nejedná, i kdyby ostatní atributy určovaly třídu jablek s velkou pravděpodobností. S použitím korekce však dostáváme $D_{c_i} = 103$ a hodnoty N_{vi}: 1, 61 a 41. Tímto způsobem jsme se vypořádali s nulovou hodnotou $P(x_k|c_i)$. [28]

2.9 Evoluční algoritmy

V této sekci si představíme evoluční algoritmy a především pak jejich zástupce v podobě genetických algoritmů. Tato sekce je zde zařazena z toho důvodu, že v rámci práce byl navržen a implementován klasifikátor, který je založen právě na principu evolučních algoritmů.

Jedná se o přístup, který je inspirován přírodou a to zejména Darwinovou evoluční teorií a genetikou. Nežli se pustíme přímo do evolučních algoritmů pojďme si tyto pojmy, z nichž evoluční algoritmy vychází, více přiblížit.

[7] Jedná se o problém reprezentace reálných čísel na omezeném počtu bitů v paměti počítače. Číslo se stane natolik malé, že již je nedokáže počítač reprezentovat.

2.9.1 Biologická inspirace

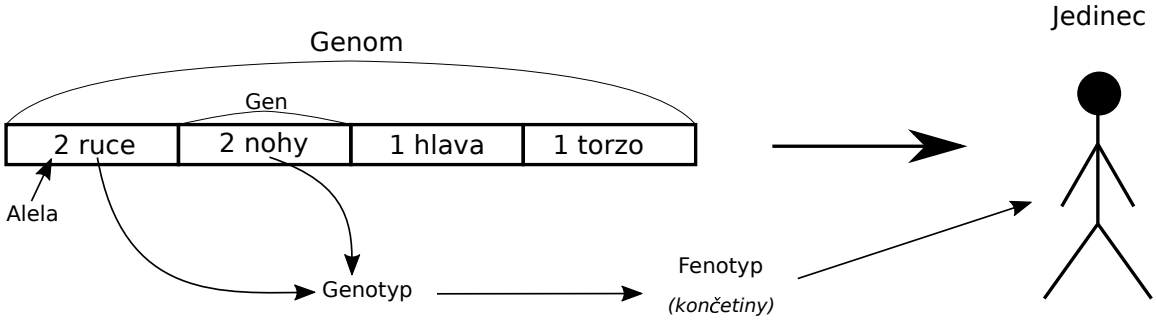
Darwinova evoluční teorie popisuje vývoj živých organismů, objasňuje jejich rozmanitost, mechanismus, který to zapříčiňuje, a poskytuje makroskopický pohled na evoluci. Ústřední roli, v této teorii, má **přirozený výběr**. K tomu dochází v nějakém prostředí obývaném omezeným množstvím jedinců, kteří se reprodukují. Přirozený výběr je příznivý k jedincům, kteří jsou schopni efektivněji soupeřit o zdroje, a tedy se dokáží lépe adaptovat na dané prostředí. Tato skutečnost je známa pod pojmem **přežití nejpřizpůsobivějších**.

Dalším důležitým pojmem v této teorii, mimo přirozeného výběru, jsou **fenotypové znaky**. Jedná se o behaviorální a fyzické znaky jedince, které ovlivňují interakci s prostředím a ostatními jedinci, čímž vlastně určují **fitness** (ohodnocení) daného jedince. Každý jedinec se skládá z kombinace těchto fenotypových znaků, kterou dané prostředí vyhodnocuje a pokud ji vyhodnotí ve prospěch jedince, zvýší se jeho pravděpodobnost na vytvoření potomka. Takovýto potomek poté může podědit fenotypové znaky rodiče, jsou-li dědičné.

Důležitou roli v procesu reprodukce (vytváření potomků) hraje pojem zvaný **mutace**. Mutací se rozumí malá náhodná změna ve fenotypovém znaku při reprodukci. Díky tomu mohou vznikat zcela nové kombinace znaků.

Celkem tedy pracujeme s určitým počtem jedinců zasazených ve společném prostředí, podléhajících přirozenému výběru závislému na ohodnocení fenotypových znaků daného jedince. Lépe přizpůsobený jedinec má větší pravděpodobnost se rozmnožit a tím i předat své fenotypové znaky do další generace. Při reprodukci může dojít k občasné mutaci a tak i vzniku zcela nových znaků.

V následujícím textu si nastíníme genetiku (molekulární), která nám poskytuje mikroskopický náhled na přírodní evoluci. V Darwinově evoluční teorii jsme mluvili o fenotypových znacích, nyní nahlédneme hlouběji a budeme se ptát na struktury, které je vytváří. Nově budeme chápat jedince jako duální entitu, která má své fenotypové vlastnosti (vnější pohled), které jsou reprezentovány pomocí **genotypů**[8] (vnitřní pohled).



Obrázek 2.12: Ilustrace souvislosti genetických pojmů. V obrázku je vyobrazen pouze jeden z možných fenotypů.

Genotyp kóduje **fenotyp** za pomocí **genů**. Geny jsou funkčními jednotkami dědičnosti kódující fenotypové vlastnosti jedince. Tedy vlastnosti jako je například počet končetin nebo očí. Dalším přímo souvisejícím pojmem jsou **alely**. Alela je konkrétní podoba genu. Lze ji v jistém smyslu připodobnit zvolené hodnotě proměnné, kde gen může být právě tou proměnnou. Ilustrativní příklad z biologického organismu by mohl například být gen,

[8] V literatuře je pojem genotypu dosti přetížen [7] a někdy se používá i ve významu celé množiny genů organismu. Nicméně my jej zde chápeme jako určitou kombinaci alel.

který je zodpovědný za počet očí. U člověka bychom pak očekávali alelu, která určuje, že daný organismus bude mít dvě oči. Naopak třeba pro pavouky by to mohla být alela určující například osm očí. Ovšem je dobré zmínit, že v přírodě nemusí ve skutečnosti být toto mapování takto jednoznačné, ale jeden gen může ve skutečnosti ovlivňovat více fenotypových znaků a stejně tak jeden znak může být ovlivňován více geny.

Změna fenotypu je vždy zapříčiněna změnou genotypu (případně vlivem prostředí [6]). Změny genotypu jsou způsobovány mutací nebo rekombinací genů při reprodukci. Celou genetickou informaci jedince označujeme jako **genom**. Genom je kompletním plánem pro vybudování jedince. Na obrázku 2.12 je vyobrazena souvislost zde uvedených pojmů. [26]

2.9.2 Základní struktura evolučních algoritmů

Nyní, když máme nastíněné biologické pozadí, se podíváme na základní společnou strukturu evolučních algoritmů. Nejprve si uvedeme schéma v podobě pseudokódu.

Algoritmus 2 Základní průběh evolučních algoritmů. [26]

```
1: inicializuj počáteční populaci s náhodnými kandidátními řešeními (jedinci)
2: ohodnoť každého kandidáta v populaci
3: while dokud neplatí ukončující podmínka do
4:     vyber rodiče
5:     proveď rekombinaci párů rodičů
6:     proveď mutace potomků
7:     ohodnoť každého kandidáta v populaci
8:     vyber jedince pro další generaci
```

Nežli si rozeberme jednotlivé kroky algoritmu 2, tak si prvně mírně ujasněme terminologii, kterou jsme zavedli v minulé sekci, pro tento kontext. Budeme se pohybovat ve dvou prostorech. V prostoru problému, takzvaném **fenotypovém prostoru** a v prostoru evolučního algoritmu, který nazýváme **genotypovým prostorem**. Fenotyp bude označovat řešení v podobě pro náš řešený problém. Může se také označovat synonymy jako je kandidátní řešení nebo jedinec. Genotypem budeme označovat zakódovaný fenotyp v nějakém vhodném kódu pro prohledávání prostoru možných řešení. Pro genotyp existuje také několik dalších synonym, a to chromozom a též, jako pro fenotyp, jedinec.

Evoluční algoritmus tedy pracuje v genotypovém prostoru, proto jedním z prvních kroků při konstrukci evolučního algoritmu je zvolení vhodného kódování fenotypů.

Algoritmus začíná získáváním počáteční populace, u které je uvedeno, že je náhodně vygenerována ovšem povšimněte si, že princip algoritmu nebude narušen, pokud vytvoříme například počáteční populaci s ohledem k nějakému známému řešení, které chceme pouze evolučním algoritmem zlepšit.

Dalším krokem je ohodnocení populace. K tomuto účelu se používá hodnotící nebo také **fitness funkce**. Fitness funkce určuje kvalitu genotypu vzhledem k řešenému problému. Často se realizuje tak, že nejprve převedeme genotyp na fenotyp, čímž se dostaneme do prostoru řešeného problému, kde ohodnotíme kvalitu dekódovaného fenotypu.

Následuje cyklus, ve kterém se tvoří a ohodnocují nové generace. Ukončovací podmínky mohou být různé. Uveďme si několik příkladů:

- Byla dosažena požadovaná hodnota fitness u nějakého jedince v populaci. Ne vždy ji však můžeme znát.

- Vypršel čas.
- Počet ohodnocení kandidátních řešení dosáhl limitu.
- Po danou dobu se hodnota udávající zlepšení fitness pohybuje pod danou hranicí.

První věc, která se v cyklu provádí je výběr rodiče. Jedná se o stochastický operátor, který většinou zohledňuje fitness rodičů, lepší mají větší šanci, že budou vybráni. Další dva kroky rekombinace a mutace slouží pro vytváření nových potomků, které vzápětí ohodnotíme pomocí fitness funkce. Tyto kroky je možné provádět s určitou pravděpodobností a nemusí být tedy provedeny vždy, nebo může být proveden pouze jeden z nich. Kroky rekombinace a mutace si více vysvětlíme na genetických algoritmech 2.9.5.

Nakonec cyklu je uveden výběr jedinců do další generace. Tento krok bývá spíše deterministický, narozdíl od výběru rodičů. Více je krok popsán v sekci 2.9.4. [26]

2.9.3 Metody pro výběr rodičů

Pro výběr rodičů existuje několik technik. Jejich výběr při implementaci bude závislý na konkrétním řešeném problému. Představme si zde následující často uváděné přístupy: [26] [45] [21]

- **Ruletová selekce (Roullete Wheel Selection)**

  Pravděpodobnost výběru jedince i je dána dle:

  $$P_i = \frac{f_i}{\sum_{j=1}^{n} f_j}, \tag{2.31}$$

  kde f_i je fitness daného jedince i a n je počet jedinců v populaci. Samotný výběr, pak uskutečníme tak, že si vygenerujeme náhodné číslo x v rozsahu intervalu $< 0, 1 >$. Poté procházíme populaci jedinců, v nějakém daném pořadí, a přičítáme do čítače c hodnotu pravděpodobnosti výběru daného jedince. V momentě, kdy $c >= x$, tak vybereme jedince, u kterého po příčtení jeho pravděpodobnosti došlo poprvé k vyhodnocení výrazu jako pravdivého.

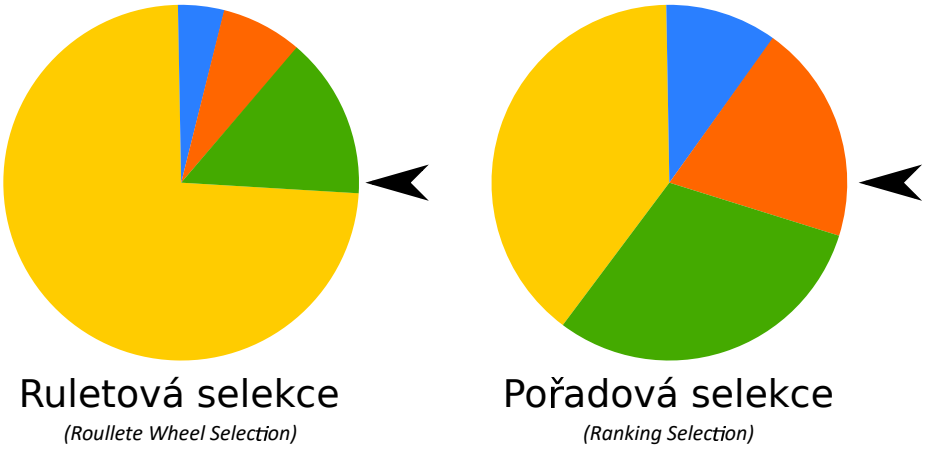
  Tento přístup získal své jméno na základě toho, že připomíná roztáčení rulety, jak je možné vidět na obrázku 2.13.

- **Pořadová selekce (Ranking Selection)**

  Pořadová selekce se snaží odstranit problém, který ruletová selekce přináší pro populace jedinců, kde je velká diverzita v hodnotách fitness. Ruletová selekce bude příliš upřednostňovat jedince, kteří mají o hodně větší fitness než ostatní (ilustrace problému na obrázku 2.13). Což může být problém (uváznutí v lokálním extrému).

  Tato metoda tedy, namísto počítání pravděpodobnosti výběru přímo z fitness, nejprve seřadí jedince (uvažujme vzestupně) dle jejich fitness a každému přiřadí pravděpodobnost danou následujícím výrazem:

  $$P_i = \frac{i}{S}, \tag{2.32}$$



Obrázek 2.13: Ilustrace rozdělení pravděpodobnosti u Ruletové selekce a u Pořadové selekce. Šipka znázorňuje výběr jedince po pomyslném roztočení rulety.

kde:

- i je pozice jedince v seřazené posloupnosti. První jedinec je na pozici $i = 1$ popřípadě někdy udáváno i $i = 0$. Udává takzvaný rank (odtud název). Ze seřazení vychází, že jedinec s největší fitness má i největší rank.
- P_i je pravděpodobnost výběru jedince na pozici i v seřazené posloupnosti.
- S je součet všech ranků.

Možná implementace této metody by byla stejná jako Ruletová selekce, ale s takto vypočítanými pravděpodobnostmi.

- **Turnaj (Tournament)**

  Selekce na základě turnaje probíhá ve dvou krocích. Nejprve vybere náhodně n jedinců z populace. Poté vybereme jedince, který je z n vybraných nejlepší.

  Povšimněte si, že nebylo nutné ohodnocovat celou populaci, a tedy tato metoda může být vhodná pro populace s velkým počtem jedinců.

- **Náhodný výběr (Uniform Selection)**

  Jedná se o selekci, který přiřazuje každému jedinci stejnou pravděpodobnost výběru. Odpovídá tedy ruletě, kde každý jedinec má stejnou fitness.

2.9.4 Nahrazení - metody pro výběr jedinců do další iterace

Nyní již víme, jakým způsobem lze získávat rodiče pro nové potomky, je však dobré si ještě zavést mechanizmus, který je schopen regulovat počet jedinců v populaci, tak abychom například pouze jen nepřidávali nové jedince a tím nám populace pouze rostla. Za tímto účelem by bylo jistě možné použít i techniky, které existují pro selekci rodičů, ale představme se zde i techniky, které jsou určeny přímo pro tento úkol.

Pro správu populace se v literatuře vyskytují dva modely: **generační model** a **steady-state model**. Generační model pracuje tak, že je vytvořena celá nová populace potomků,

která nahrazuje původní populaci. Svůj název generační, tedy získal zřejmě proto, že nová generace nahrazuje starší generaci. Steady-state model naopak s generacemi v tomto smyslu nepracuje. Zde se pouze nahrazuje část stávající populace novými potomky. Počet vygenerovaných potomků bývá malý, například pouze jeden potomek. Samotné nahrazování, pak může probíhat například tak, že x nejhorších jedinců z aktuální populace je nahrazeno x nejlepšími potomky. Další možností by bylo například vytvořit ke každým dvěma párům dva potomky a pak z rodičů a vytvořených potomků vybrat dva nejlepší, kteří budou nadále v populaci a ostatní dva budou vyjmuti [15].

Dále si zde uveďme často zmiňované strategie pro nahrazování jedinců. Svým přístupem k tvorbě nových generací by je bylo možné srovnávat se zmiňovanými modely, ovšem ukazují, jakým způsobem, lze vybírat jedince pro nahrazování. Pojďme si představit tyto následující:

- **Náhodné nahrazení (Random replacement)**

  Nová populace je vytvořena z náhodného výběru jedinců existující populace a jejich potomků.

- **Nahrazení na základě věku (Age-Based Replacement)**

  Nahrazení na základě věku, provádí výběr jedinců pro další generaci tak, že nechává dané jedince v populaci po nějaký stanovený počet iterací evolučního algoritmu. Nebere tedy v potaz fitness daných jedinců a je možné, že nová generace bude složena z jedinců, kteří budou mít horší fitness než jedinci původní generace. Tato vlastnost nemusí být nutně na škodu a může zabránit uváznutí v lokálním extrému. Je možné, že nějaký genotyp přežije déle, než je maximální věk. Jelikož se může stát, že vlivem provedení/neprovedení rekombinací či mutací bude vytvořen nebo zůstane nezměněn daný genotyp.

  Pokud každý jedinec žije pouze jeden cyklus, tedy se vždy zbavíme původní populace, a necháme pouze novou populaci vytvořenou z potomku, pak dostáváme generační nahrazení. Často se nechává počet jedinců v populaci napříč generacemi konstantní.

- **Elitismus (Elitism)**

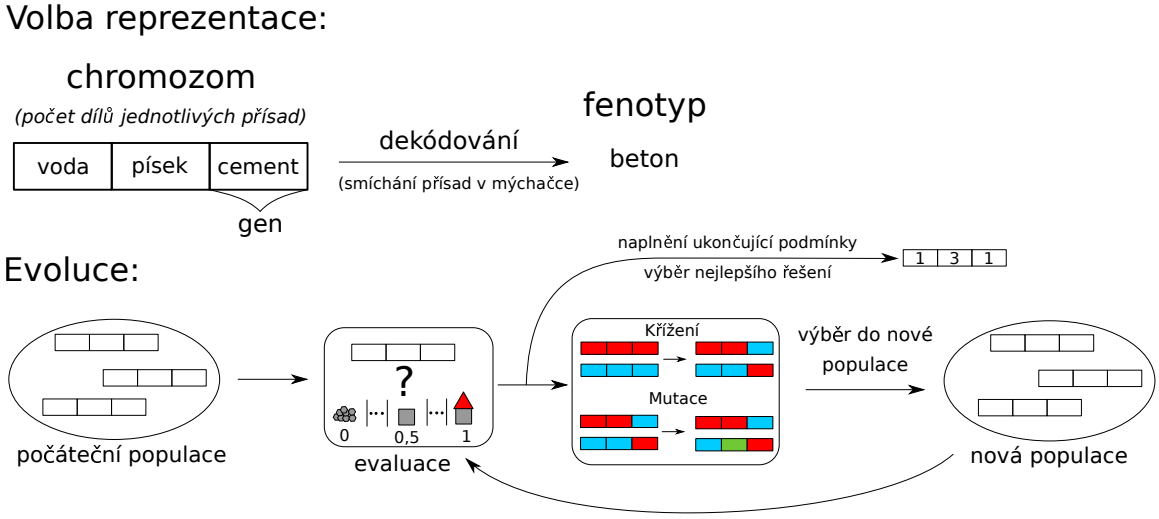
  Tento přístup zaručuje, že nejméně jedna kopie nejlepšího genotypu se dostane do další generace. Výhodou tohoto přístupu je, že pokud se při provádění algoritmu dostaneme do globálního optima, pak máme zaručeno, že o naše řešení nepřijdeme. Ovšem může se stát, že dokonvergujeme pouze do lokálního optima.

  Může se také použít v kombinaci s jinou technikou a pomocí elitismu si budeme pouze vytvářet archiv nejlepších řešení.

Závěrem k této sekci se hodí zmínit, že populace může mít proměnlivý počet jedinců napříč iteracemi evolučního algoritmu, ale stejně tak může mít počet jedinců pokaždé stejný. [26][45][21][41]

2.9.5 Genetické algoritmy

V této sekci se budeme bavit o pravděpodobně nejznámější variantě evolučních algoritmů, a to sice o genetických algoritmech. U genetických algoritmů kódujeme naše kandidátní řešení do chromozomů. Ukázku chromozomu je možné vidět na obrázku 2.14. Chromozomy mohou mít délku fixní, ale i proměnnou. Tedy pro nějaké druhy problémů se může hodit, že



Obrázek 2.14: Schématická ilustrace průběhu genetického algoritmu na řešení problémů hledání optimálního poměru surovin pro beton. Včetně návrhu chromozomu.

každý jedinec má stejný počet genů, ale pro jiné problémy je dobré mít pro každého jedince jiný počet genů.

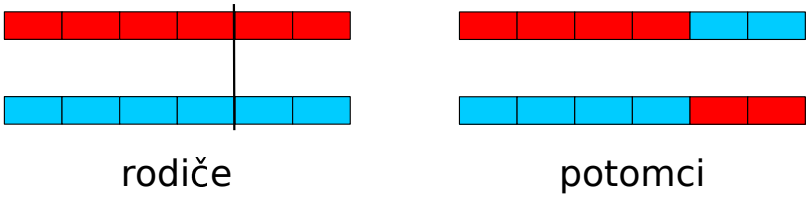
Takovéto chromozomy se pak stávají členy populace, kterou podrobíme evoluci. Obecná struktura evolučního algoritmu zůstává zachována, proto si zde pouze uvedeme, jakým způsobem dochází k rekombinacím (**křížení**) a mutacím nad chromozomy. Jak rekombinace, tak mutace jsou obecné operace pro evoluční algoritmy, ovšem zde využijeme právě genetické algoritmy pro jejich názorné odilustrování.

Křížení se tradičně aplikuje na dva rodiče, ale není to nezbytně nutné, nicméně tento text se bude zabývat pouze křížením dvou jedinců. Operátor křížení nemusí být aplikován pokaždé, ale může být použitý pouze s určitou pravděpodobností. Pokud nebude aplikován, tak se vybraní rodiče pouze okopírují v podobě v jaké jsou.

Uveďme si zde několik typu křížení:

- **Jednobodové křížení**

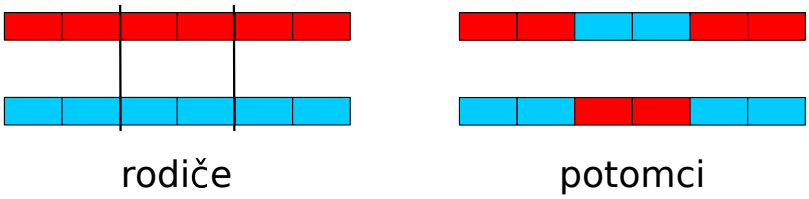
  Náhodně zvolíme bod křížení, který určuje, které části budou vyměněny. Toto křížení je ilustrováno na obrázku 2.15.



Obrázek 2.15: Jednobodové křížení.

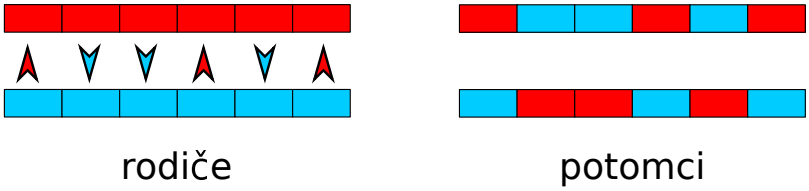
- **Dvoubodové křížení**

  Podobné jednobodovému křížení, ale používáme zde, jak název napovídá, dva body. Ilustrace na obrázku 2.16.



Obrázek 2.16: Dvoubodové křížení

- **Uniformní křížení**
  Existují dva přístupy. Můžeme prvně náhodně vybírat geny od jednoho či druhého rodiče pro jednoho potomka, poté proces opakovat pro druhého potomka. Další přístup pracuje tak, že pro každý gen prvního rodiče se náhodně rozhodneme, zdali bude v prvním potomkovi. Pokud ano, tak gen na stejné pozici u druhého rodiče bude ve druhém potomkovi. Pokud ne, tak aktuální gen prvního rodiče přiřadíme druhému potomkovi a prvnímu přiřadíme gen na stejné pozici u druhého rodiče. Tento druhý přístup je vyobrazen v obrázku 2.17.



Obrázek 2.17: Ilustrace uniformního křížení. Šipka ukazuje náhodný výběr genu pro prvního potomka. Pro druhého potomka je vždy vybrán gen na stejné pozici, ale od druhého rodiče.

Nyní jsme si ukazovali křížení pro chromozomy fixní délky. Pro chromozomy proměnné délky, lze použít stejné strategie. Pouze je nutné udělat menší úpravy. Například pro jednobodové či více bodové budou body křížení na jiných pozicích u každého z rodičů.

Posledním doposud nepopsaným operátorem je mutace. Jedná se o změnu (obvykle malou) chromozomu jedince, na který tuto operaci aplikujeme. Podobně jako křížení se i tento operátor může aplikovat pouze s určitou pravděpodobností, ovšem pokud jej aplikujeme s příliš velkou pravděpodobností proměníme algoritmus na náhodné prohledávání. Naopak, pokud jej neaplikujeme vůbec, tak se vystavujeme riziku, že nám populace, vlivem použití pouze křížení, zkonvertuje do stejného genotypu. V rámci mutací může docházet například k následujícím operacím:

- Změna alely na jinou alelu daného genu. Tedy například změna celočíselné hodnoty na jinou.
- Přidání/odstranění genu. Pro chromozomy proměnné délky.
- Permutace genů. Prohodíme náhodně vybrané geny mezi sebou (je-li to možné).

Při vytváření nových potomků mohou vznikat chromozomy, které po převedení na fenotyp nemusí dávat smysl v dané doméně problému, proto je dobré, jsme-li toho schopni, aplikovat operátory takovým způsobem, aby k tomu nedocházelo. [21][45]

2.9.6 Klasifikace pomocí genetických algoritmů

V této sekci si uvedeme, jakým způsobem je možné použít genetické algoritmy pro řešení problému klasifikace. Bude se jednat pouze o stručný popis vybraných metod, protože v rámci této práce byla implementována jiná metoda, která je popsána v sekci 4.5. Zde uvedené metody mají sloužit hlavně jako přehled dalších možných řešení.

Nejprve si zde uvedeme přístup, který je založen na pravidlech. Začíná tak, že si vytvoříme náhodnou počáteční populaci klasifikačních pravidel. Každé z těchto pravidel může být reprezentováno jako řetězec bitů. Například máme-li dva booleovské atributy A_1 a A_2 a klasifikujeme do dvou tříd C_1 a C_2. Pak můžeme pravidlo tvaru:

$$\text{If not } A_1 \text{ and } A_2 \text{ THEN } C_1,$$

zakódovat jako: 010. Dva nejlevější bity jsou geny pro pravidla a poslední je gen pro výslednou třídu. Pro více tříd či více hodnot u atributu může mít příslušný gen více bitů.

Dále pokračujeme způsobem, který je pro genetické algoritmy běžný a již byl popsán v předcházejícím textu. [28]

Další přístup je založený na symbolické regresi, ta striktně vzato spadá spíše do genetického programování, což je ovšem opět evoluční algoritmus. Jedná se o hledání matematického výrazu, který nejlépe sedí na poskytnutá data. Například si vezměme tyto body:

$$\text{(-2,4), (-1,1), (0,0), (1,1), (2,4).}$$

Kde první číslo ve dvojici odpovídá ose x a druhé ose y, pak by výsledkem symbolické regrese mohla být funkce:

$$f(x) = x^2. \tag{2.33}$$

Možné využití symbolická regrese pak spočívá v hledání modelu, který je schopen od sebe oddělovat jednotlivé třídy, tak jak je uvedeno v [39].

2.10 Metody vyhodnocování klasifikátorů

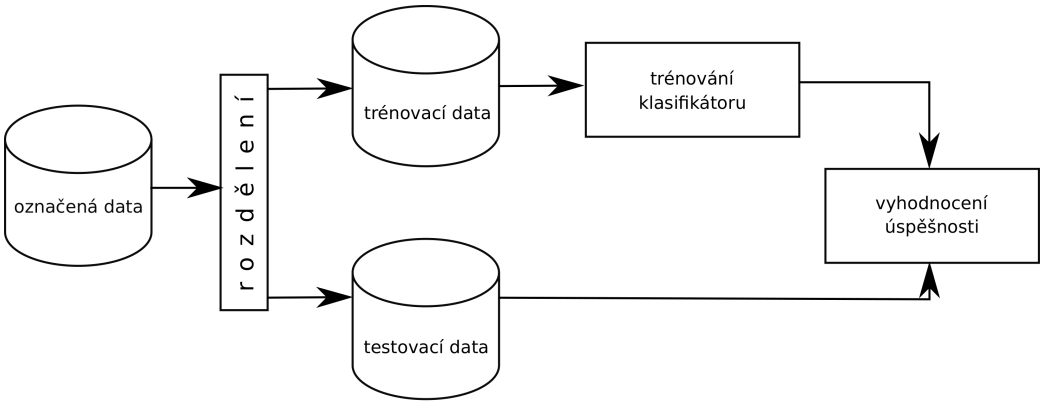
Nyní si uvedeme, jakým způsobem lze získat představu o tom, jak dobrý je náš klasifikátor. Typicky nás nejvíce zajímá, jak si povede na nových datech, které nezná (nebyly k dispozici při učení). Tomuto říkáme, že se ptáme jak klasifikátor dobře generalizuje. Ukážeme si tedy, jakým způsobem lze zjistit kvalitu klasifikátoru nad takovýmito daty, a představíme si zde několik metrik použitelných pro vyhodnocení.

Možné schéma průběhu testování klasifikátoru je vidět na obrázku 2.18. Máme k dispozici nějakou datovou sadu s označenými daty, kterou rozdělíme na dvě části: trénovací a testovací. Pomocí trénovací sady natrénujeme klasifikátor a natrénovaný klasifikátor použijeme k označení testovací sady. Testovací sada obsahuje referenční značení tříd, které použijeme k porovnání s výsledky, které nám dal náš natrénovaný klasifikátor.

Naším cílem je, aby testovací množina neobsahovala (nebo jen minimálně) vzorky z trénovací množiny, protože nás zajímá jak náš klasifikátor dobře generalizuje.

Můžeme také vícekrát různě dělit dostupnou označenou datovou sadu, agregovat výsledky z běhů nad různými rozděleními, a tím dosáhnout lepšího využití dostupných označených dat.[28]

Dále si popíšeme některé konkrétní metody použitelné pro rozdělování datové sady.



Obrázek 2.18: Možné schéma průběhu vyhodnocování klasifikátoru na trénovací/testovací sadě.

2.10.1 Křížová validace

Budeme zde mluvit konkrétně o k-fold křížové validaci (cross-validation). Tato metoda rozděluje původní datovou množinu na k přibližně stejně velkých množin, které mají navzájem prázdný průnik. Trénování/testování je poté provedeno přesně k krát, kdy vždy jedna množina je použita pro testování a zbytek pro trénování.

Například tedy pro $k = 4$ rozdělíme původní data na 4 množiny S_1, S_2, S_3 a S_4. V prvním ze čtyř kol natrénujeme klasifikátor na $S_2 \cup S_3 \cup S_4$ a otestujeme jej na S_1. Ve druhém kole klasifikátor natrénujeme na $S_1 \cup S_3 \cup S_4$ a otestujeme na S_2. Tímto způsobem pokračujeme ještě dvakrát. Nakonec vyhodnotíme získané výsledky ze všech kol pomocí metrik. Povšimněte si, že jsme dosáhly toho, že každý vzorek byl použit jak pro testování, tak pro trénování.

Speciálním případem k-fold křížové validace je metoda **leave-one-out**. Jedná se o k-fold křížovou validaci, kde k je nastaveno na počet vzorků v původní sadě. Velikost testovací sady v každém z k kol bude rovna jedné. Tímto přístupem se nejvíce přiblížíme situaci, kdy trénujeme klasifikátor na všech označených vzorcích, ale celý proces je výpočetně velmi náročný.

Další možnou variantou je **stratified cross-validation**, která zachovává původní poměr počtu vzorků pro každou ze tříd v získaných množinách po rozdělení. [28]

2.10.2 Holdout metoda a Random Subsampling

Holdout metoda vybírá náhodně vzorky z původní sady a dělí je do trénovací a testovací množiny, bez navracení[9].

Trénovací množina bývá typicky větší než testovací (2/3). Na rozdíl od k-fold křížové validace zde neprovádíme více kol, ale pouze jen jedno.

Random subsampling je variací holdout metody, kdy holdout metodu provádíme k krát a nakonec agregujeme výsledky ze všech kol. [28]

[9] Vzorek nemůže být vybrán dvakrát.

2.10.3 Metriky

Ukázali jsme si jakým způsobem lze testovat klasifikátory, nyní je ještě nutné si uvést, jaká kvantitativní vyjádření můžeme použít, abychom získali představu o tom, jak dobrý je náš klasifikátor.

		referenční	
		pozitivní	negativní
predikované	pozitivní	skutečně pozitivní (TP)	falešně pozitivní (FP)
	negativní	falešně negativní (FN)	skutečně negativní (TN)

Tabulka 2.3: Kontingenční tabulka či také matice záměn pro klasifikaci do dvou tříd.

Pro vyjádření zde uváděných metrik se budeme opírat o matici záměn 2.3. Tato matice uvádí výsledky pro klasifikace do dvou tříd, pozitivní a negativní. V tabulce vystupují následující čtyři hodnoty [28]:

- **TP (True positives):** Počet správně označených vzorků (ze třídy pozitivních) klasifikátorem.
- **FP (False positives):** Počet špatně označených vzorků (ze třídy negativních) klasifikátorem. Byly označeny za pozitivní, i když mají být ve třídě negativních.
- **FN (False negatives):** Počet špatně označených vzorků (ze třídy pozitivních) klasifikátorem. Byly označeny za negativní, i když mají být ve třídě pozitivní.
- **TN (True negatives):** Počet správně označených vzorků (ze třídy negativních) klasifikátorem.

Z těchto hodnot můžeme sestavovat různé metriky, jak je ukázáno v tabulce 2.4. [28]

metrika	**vzorec**
accuracy, recognition rate (správnost)	$\frac{TP+TN}{TP+TN+FP+FN}$
error rate, misclassification rate (chyba)	$\frac{FP+FN}{TP+TN+FP+FN}$
sensitivity, true positive rate, recall (úplnost)	$\frac{TP}{TP+FN}$
precision (přesnost)	$\frac{TP}{TP+FP}$
specificity, true negative rate (specificita)	$\frac{TN}{FP+TN}$
F	$\frac{(\beta^2+1)\cdot precision\cdot recall}{\beta^2\cdot precision+recall}$
F_1 - speciální případ F pro $\beta = 1$	$\frac{2\cdot precision\cdot recall}{precision+recall}$

Tabulka 2.4: Příklady používaných metrik při vyhodnocování klasifikátoru.

Kapitola 3

Návrh aplikace pro porovnání klasifikátorů

V této sekci si uvedeme podobu aplikace, která bude schopna porovnávat klasifikátory na poskytnutých datových sadách. Nejprve si popíšeme požadavky na navrhovanou aplikaci, poté bude proveden návrh grafického uživatelského rozhraní. V závěru kapitoly si popíšeme návrh aplikace na úrovni tříd, které mají pro implementaci stěžejní postavení. Sekce vychází z předchozí práce [25].

3.1 Požadavky na systém

Aplikace bude splňovat minimálně následující požadavky:

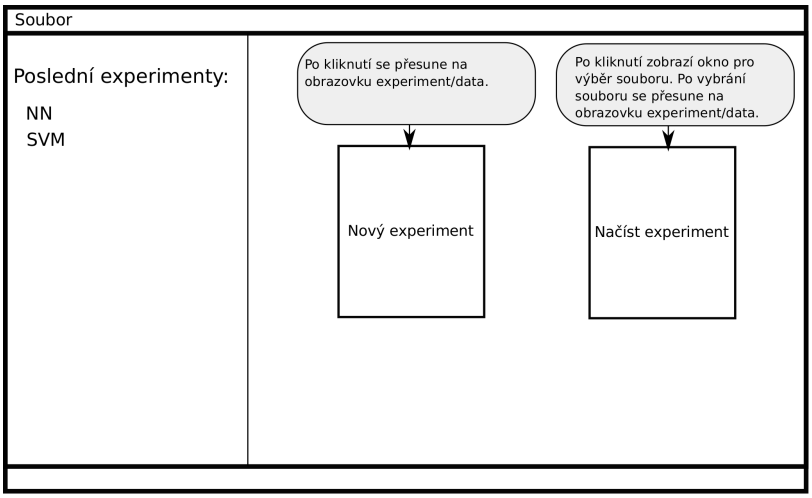
- Bude poskytovat grafické uživatelské rozhraní.
- Klasifikátory budou v aplikaci uvažovány jako zásuvné moduly, které by měly jít snadno doplňovat.
- Uživatel bude mít možnost spouštět zvolený klasifikátor na vlastní datové sadě, která bude v podporovaném formátu.
- Pro datovou sadu si bude možné zobrazit základní statistické údaje.
- Aplikace bude poskytovat metody pro extrakci příznaků z textu a obrázků, obecně se však bude jednat o zásuvné moduly, stejně jako u klasifikátorů.
- Uživatel bude mít možnost použít metody pro selekci příznaků a použít vybrané metody pro normalizaci příznaků.
- Bude možné spouštět nad danými klasifikátory a daty testy, které budou používat křížovou validaci.
- Po provedení testů se uživateli zobrazí vyhodnocení úspěšnosti pomocí standardních metrik. Zobrazí se i časová vyhodnocení jednotlivých klasifikátorů.
- Bude umožňovat ukládat šablony experimentů. Šablona nese informace o datové sadě a klasifikátorech, které se mají porovnávat.

Systém bude implementován v jazyce python [10]. Pro implementaci grafického uživatelského rozhraní bude použita knihovna PySide2 [11].

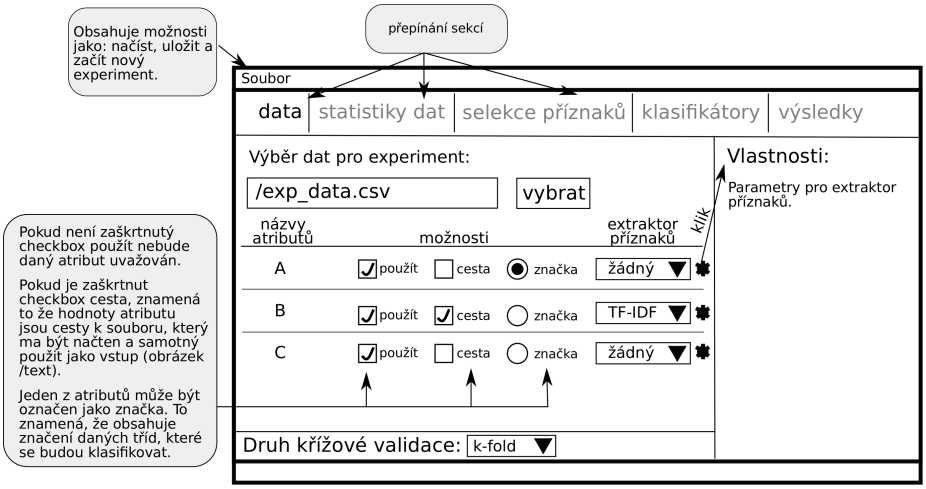
Klasifikátory, extrakce příznaků, počítání metrik a křížová validace budou buď přímo implementovány anebo bude použita knihovna/framework typu: scikit-learn [37], Keras [9] či TensorFlow [18].

3.2 Návrh grafického uživatelské rozhranní

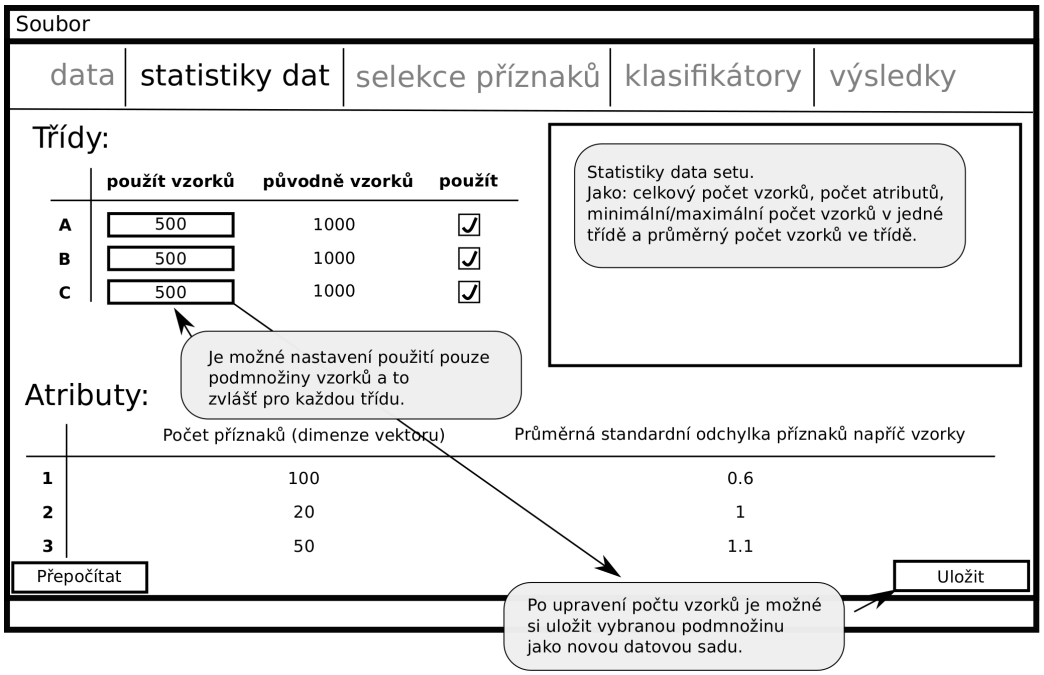
V této sekci si uvedeme návrh grafického uživatelského rozhraní aplikace. Z tohoto návrhu bude jasné, jaké funkce bude aplikace poskytovat a kde (na jakých obrazovkách) budou k dispozici. Návrh je zobrazen na obrázcích 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7 a 3.8, které zobrazují jednotlivé sekce aplikace a jejich prvky.



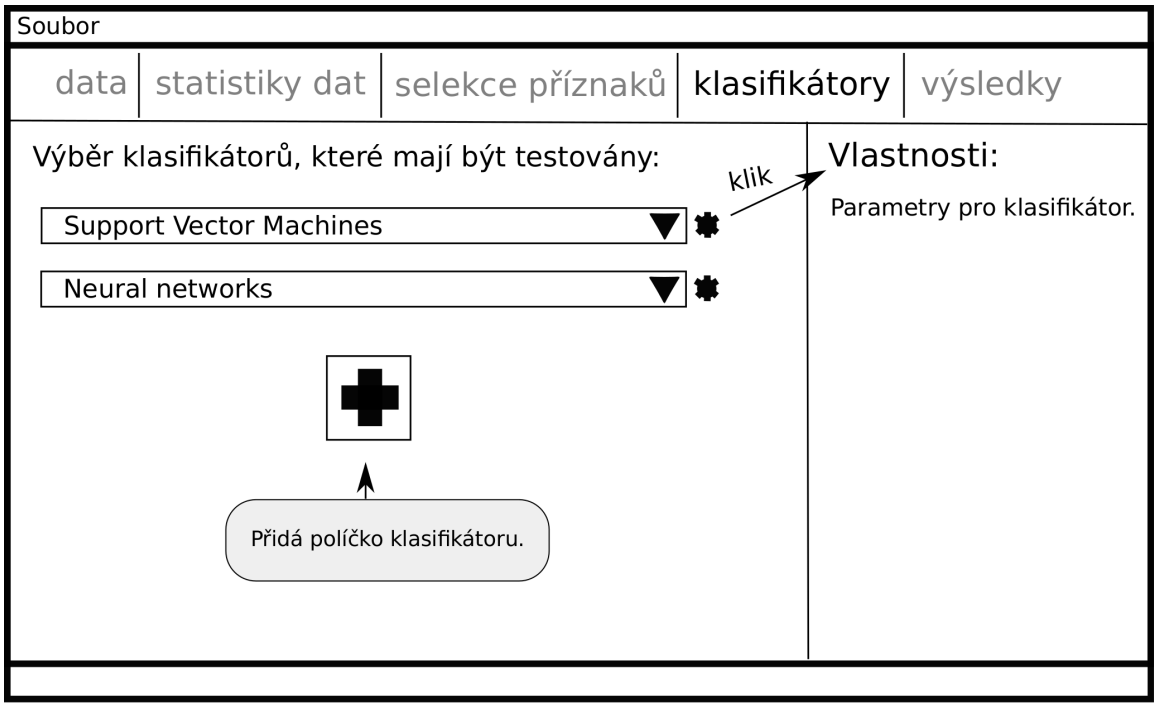
Obrázek 3.1: Návrh domovské obrazovky.



Obrázek 3.2: Návrh obrazovky experimentu, konkrétně sekce pro výběr datové sady.

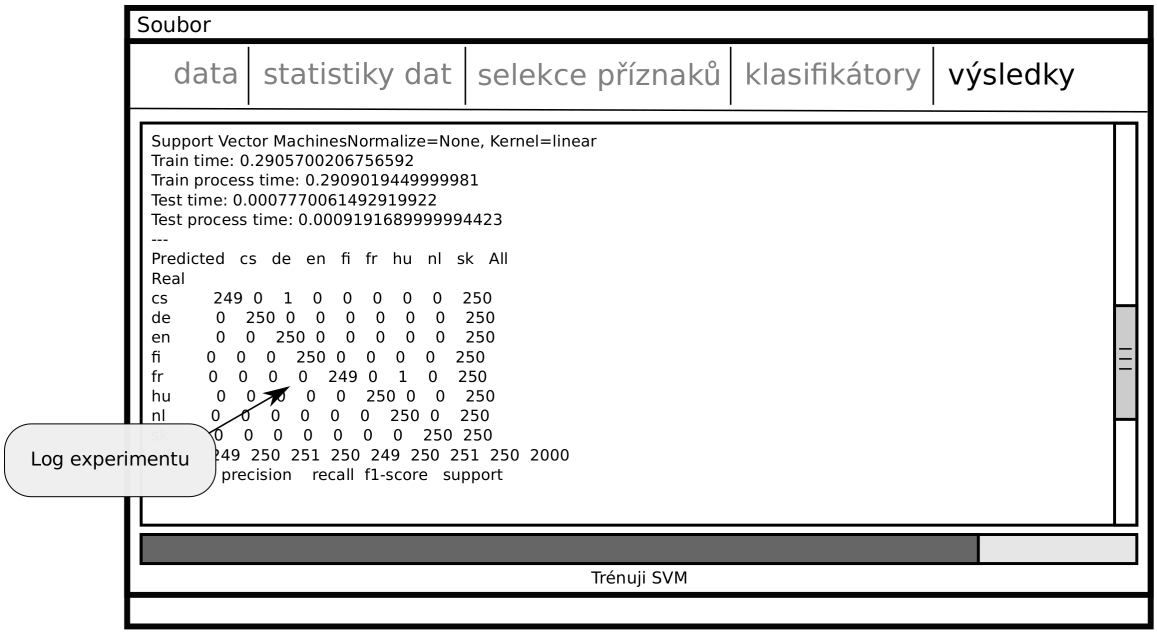


Obrázek 3.3: Návrh obrazovky zobrazující statistiky k datové sadě. Pro získání tohoto pohledu bude uživatel nejprve muset stisknout tlačítko, které provede nutné výpočty, podobně jako v sekci: výsledky.

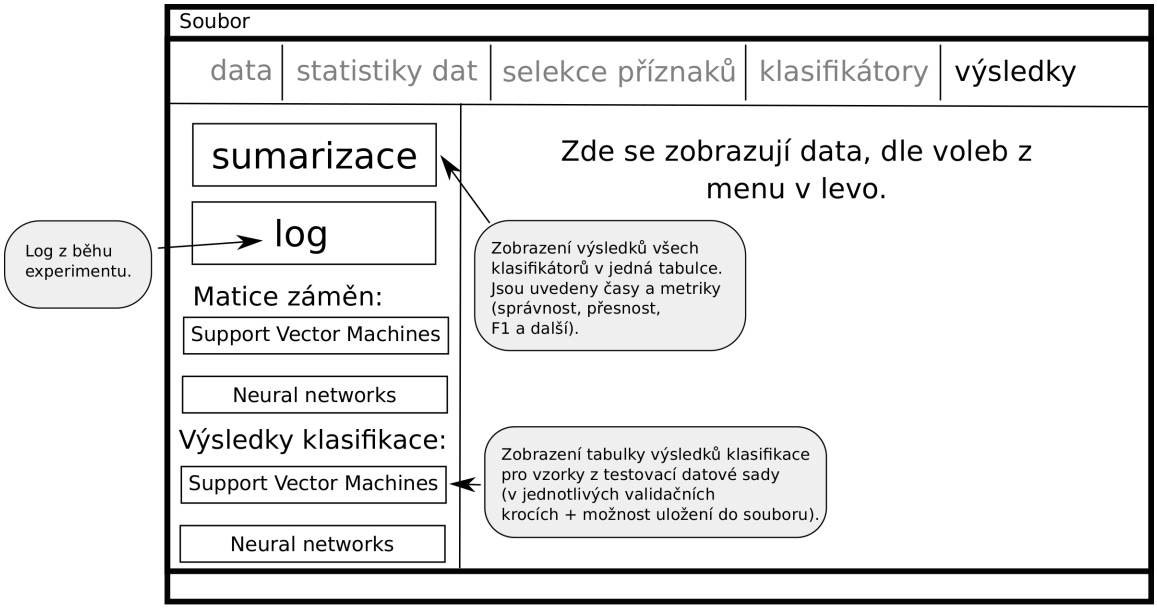


Obrázek 3.4: Návrh obrazovky experimentu, konkrétně sekce pro výběr klasifikátorů, které mají být otestovány.

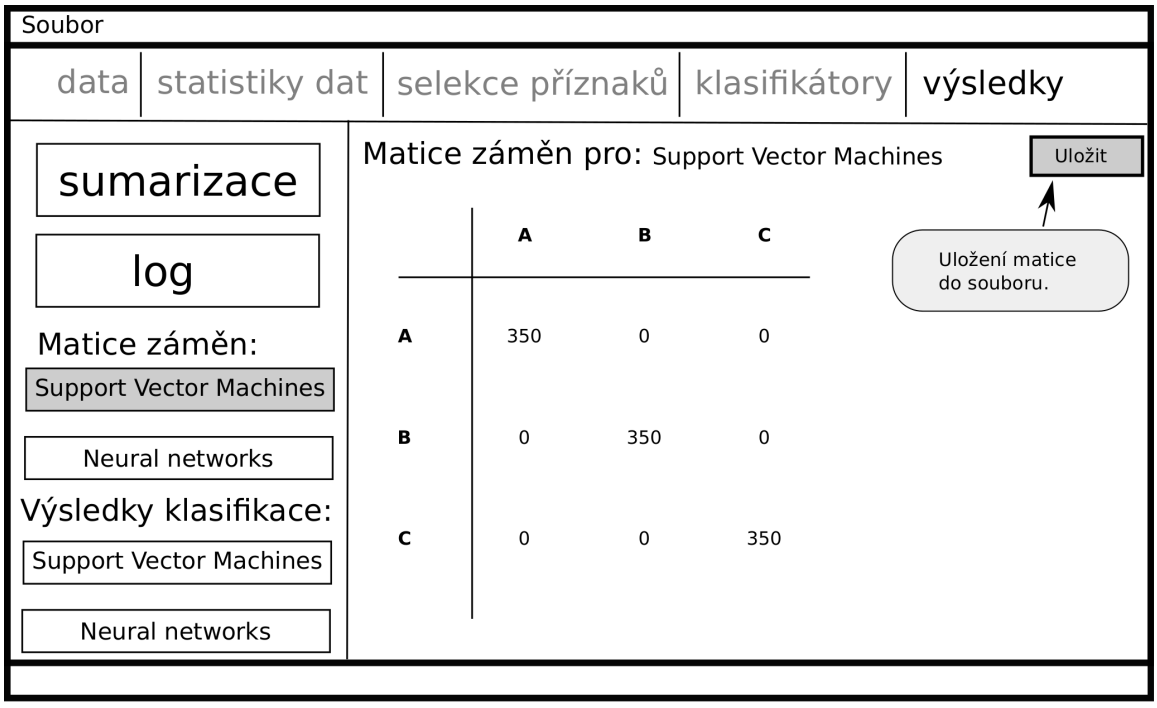
Obrázek 3.5: Návrh obrazovky experimentu, konkrétně sekce s výsledky pro případ, kdy ještě není proveden experiment.



Obrázek 3.6: Návrh obrazovky probíhajícího experimentu.



Obrázek 3.7: Návrh obrazovky experimentu, konkrétně sekce se získanými výsledky experimentu.



Obrázek 3.8: Návrh obrazovky pro matici záměn vybraného klasifikátoru. Uvádíme si zde pouze tento detailnější návrh pro matici záměn. Pro další volby je vzhled obdobný, hlavním zobrazovacím prvkem je opět tabulka.

3.3 Diagram tříd

Na obrázku 3.9 lze vidět diagram tříd navrženého systému. Tento digram se skládá ze čtyř balíčků:

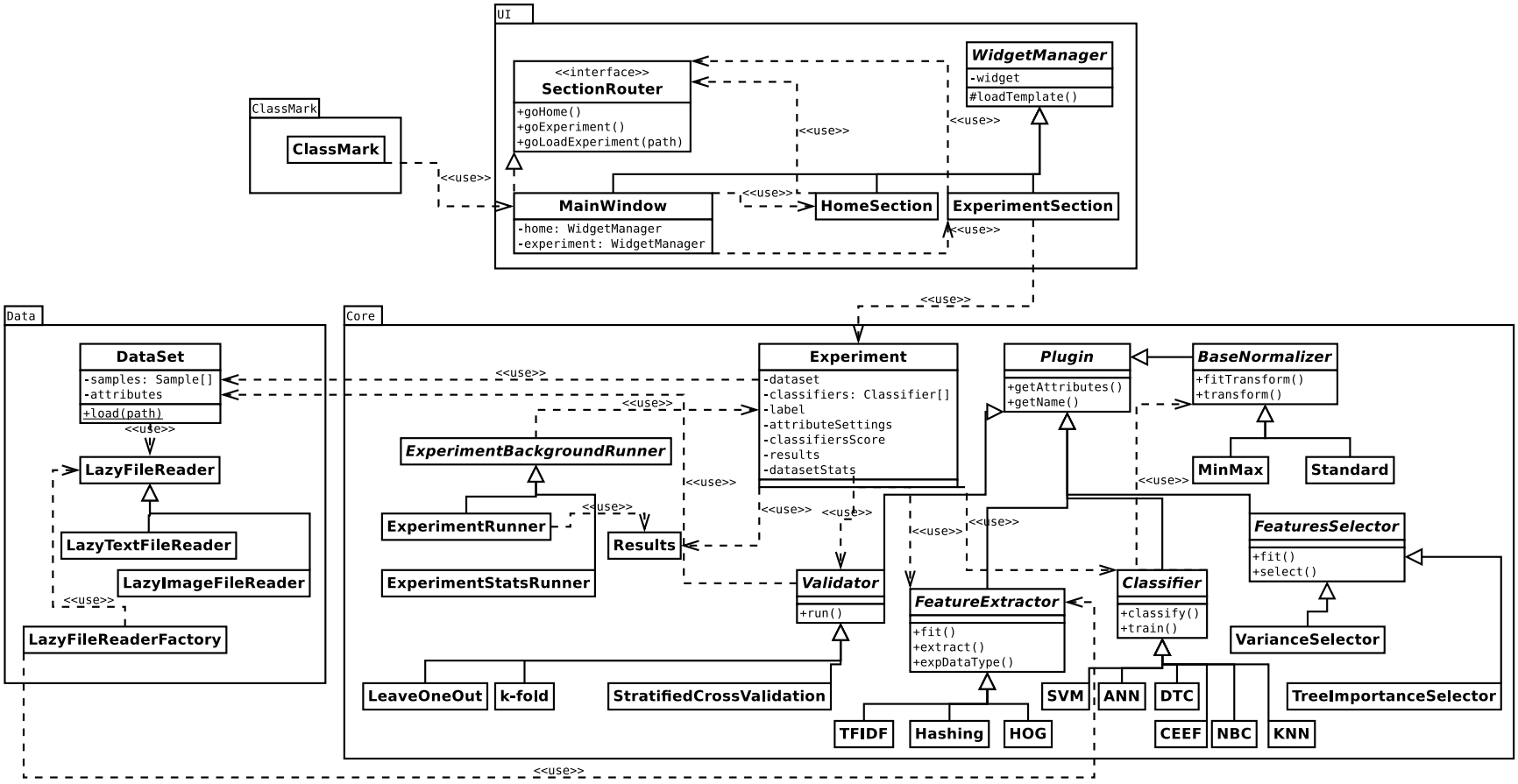
- **ClassMark**: Obsahuje třídu se vstupním bodem programu. Tato třída bude sloužit primárně pro start aplikace.
- **UI**: Tento balíček je sestaven ze tříd, které tvoří grafické uživatelské rozhraní a zpracovávají pokyny od uživatele. V digramu 3.9 sice nejsou naznačeny vazby na externí PySide2, ale je to právě tento balíček, který s PySide2 spolupracuje. Jiné třídy/balíčky vazbu na PySide2 nesmějí mít.
- **Data**: Obsahuje třídy pro nahrávání a reprezentaci datové sady. V 3.9 je naznačena závislost mezi extraktory příznaků a továrnou LazyFileReaderFactory. Tato závislost je zde proto, že se továrna rozhoduje jaký druh LazyFileReader vytvoří na základě druhu dat, který příslušný extraktor příznaků vyžaduje.
- **Core**: Tento balíček je složen ze tříd, které slouží pro: samotné testování klasifikátorů, reprezentaci experimentu, počítání metrik a získávání/ukládání výsledků. Je zde také obecně definován zásuvný modul a jeho druhy: klasifikátory, extraktory příznaků, validátory, normalizátory a selektory příznaků. Nejedná se o vyčerpávající výčet všech pluginů, ale pouze o ilustrativní výčet. Důraz je kladen na snadnou rozšiřitelnost, tedy aby společné rozhraní bylo co nejobecnější a jednoduché.

  Návrh počítá s během náročnějších operací v samostatném procesu či vlákně, proto je zde třída **ExperimentBackgroundRunner**. Třídy, které od této dědí, jsou třídy zapouzdřující běh experimentu a získání statistik k datové sadě.

Popišme si na tomto místě detailněji navržený systém zásuvných modulů, který je znázorněn v diagramu 3.9. V hierarchii dědičnosti u zásuvných modulů je abstraktní třída ***Plugin*** položena nejvýše. Každý zásuvný modul, bude tedy muset být schopen poskytnout své atributy (**getAttributes**), které bude uživatel nastavovat přes grafické uživatelské rozhraní. Dále je v diagramu znázorněno poskytnutí jména (**getNames**). Návrh nezobrazuje další možné metody jako by například mohlo být poskytnutí zkratky názvu zásuvného modulu. Každý klasifikátor musí umět klasifikovat a musí se umět natrénovat na poskytnutých datech, proto existují metody **classify** a **train** u abstraktní třídy ***Classifier***. Některé extraktory příznaků vyžadují podobně jako klasifikátory fázi trénování, proto abstraktní třída ***FeatureExtractor*** obsahuje metodu **fit**, pro samotnou extrakci pak existuje metoda **extract**. Obdobné platí i pro selektory příznaků, kde však pro fázi použití, u abstraktní třídy ***FeatureSelector***, slouží metoda **select**.

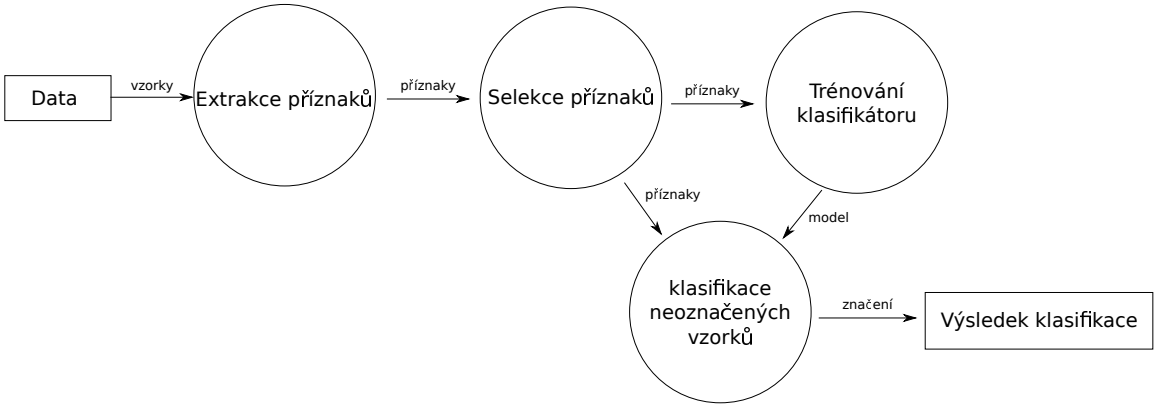
Velmi podobná filozofie platí i pro zásuvné moduly pro normalizaci. Pro normalizace definuje abstraktní třída ***BaseNormalizer*** dvě metody: **fitTransform** a **transform**. Metoda fitTransform „natrénuje" (získá statistiky z trénovacích dat) normalizátor a rovnou i na vstupních datech provede normalizaci, naopak metoda transform pouze provede normalizaci na daných datech .

Posledním druhem zásuvných modulů jsou validátory. Abstraktní třída ***Validator*** má v návrhu vyznačenou metodu **run**, která slouží pro řízení běhu experimentu (včetně měření času), jedná se tedy o provádění jednotlivých (křížově) validačních kroků. Každý takový krok se bude skládat s fáze trénování a testování. Tomu jsou i přizpůsobeny metody u daných zásuvných modulů. Například u klasifikátorů budeme volat metodu train ve fázi trénování a classify ve fázi testování.



Obrázek 3.9: Digram tříd zobrazující důležité třídy a závislosti mezi nimi. Jsou zde zahrnuty i stěžejní atributy a metody. Z digramu jsou vyřazeny metody, které nemají příliš informativní charakter z pohledu návrhu systému. Jako jsou metody pro získání hodnoty, nastavení hodnoty a další.

Každý validátor by měl, mimo výše zmíněného, zachovávat pořadí vykonávání jednotlivých operací v rámci jednoho validačního kroku, tak jak je znázorněno v diagramu toků dat 3.10. V diagramu je záměrně vynechána normalizace, jelikož tu si bude moci nastavit každý klasifikátor sám. Díky tomu bude možné spouštět jeden experiment, nad stejnými příznaky, pro více různých klasifikátorů s tím, že každý si bude moci zvolit, zdali chce příznaky normalizovat či ne.



Obrázek 3.10: Diagram toků dat pro jeden validační krok.

Kapitola 4

Implementace aplikace pro porovnání klasifikátorů

V této sekci si popíšeme implementaci jednotlivých částí aplikace pro porovnání klasifikátorů. Uvedeme si jakým způsobem je implementován systém zásuvných modulů, v jakém formátu jsou očekávana vstupní data, v jakém formátu jsou poskytovány výstupy, volání procesů/vláken na pozadí a další. Mimo popisu implementace aplikace samotné uvedeme i popis implementovaného klasifikátoru, který je založen na principu evolučních algoritmů.

Aplikace je implementována dle návrhu popsaného v kapitole 3, tedy implementačním jazykem je python konkrétně verze 3 (testováno na verzi 3.6.7).

4.1 Části aplikace

Aplikace je implementována jako pythonovský balík. Díky tomu je usnadněna její instalace a instalace potřebných závislostí. Všechna potřebná nastavení balíku se nachází v *setup.py* [11]. Tento soubor říká, že aplikace se skládá z balíku *classmark*, ve kterém se nachází vstupní bod aplikace, konkrétně se jedná o souboru *___main___.py*.

Soubor *___main___.py* má hlavní zodpovědnost spustit grafické uživatelské rozhraní realizované v *PySide2*. Jeho další zodpovědnost tkví v ukončení procesů na pozadí v případě ukončení aplikace v momentu, kdy se stále vykonává nějaká výpočetně náročnější činnost, pro které si aplikace vytváří pomocné procesy.

Nejprve je vytvořeno hlavní okno pomocí třídy *MainWindow*, tato třída dále sama vytváří dvě hlavní sekce aplikace a to sice domovskou sekci, kterou má na starost třída *HomeSection* a sekci experimentu, která je spravována třídou *ExperimentSection*.

Přestože jsou tyto dvě sekce na pozadí vytvořeny obě, pro uživatele je zobrazena prvně domovská sekce, odpovídající grafický návrh je možné vidět na obrázku 3.1. Uživatel má zde možnost vytvořit nový experiment, načíst již vytvořený nebo vybrat jeden z již v minulosti použitých. Načtení seznamu dříve spuštěných experimentů zajišťuje třída *ListLastExperiments*, která je modelem pro list v uživatelském rozhraní. Jedná se o jeden z několika používaných modelů, které aplikace implementuje z důvodů využívání Model/View architektury.

Sekce experimentu může být inicializována dvěma způsoby: jako prázdný experiment či jako načtený uložený experiment. Hlavní objekt, který reprezentuje celý experiment, je třídy *Experiment*. Tato třída obsahuje všechny potřebné informace o daném experimentu, stará se

o vytváření nového experimentu, ukládání experimentu do souboru a načtení experimentu ze souboru. Dále poskytuje rozhraní pro manipulaci vnitřních struktur jako jsou:

- klasifikátory
- extraktory příznaků
- selektory příznaků
- datová sada
- statistiky o datové sadě
- výsledky experimentu

Grafické uživatelské rozhraní pro sekci experimentu se při uživatelských akcích téměř výhradně odvolává na objekt třídy *Experiment*.

Jak je vidět v návrhu na obrázku 3.2, tak sekce experimentu se skládá z několika podsekcí (záložek). Úvodní záložkou je záložka pro data. Zde je možné vybrat datovou sadu, se kterou chceme experimentovat. Načítání datových sad má na starosti objekt třídy DataSet, který reprezentuje zvolenou datovou sadu. Dále zde vybíráme druh validace, atributy, které budou použity, extraktory příznaků pro jednotlivé atributy (včetně nastavení jejích parametrů) a určujeme, zdali se jedná o atribut jehož hodnoty představují cesty k souborům s obsahem. Tedy tak jak definuje návrh.

Další záložkou v pořadí je záložka pro získání statistik datové sady. Zobrazené statistiky se odvíjejí od aktuálního nastavení v záložce data, jelikož se zde objevují statistiky vztažené k třídám a příznakům. Nejprve je tedy nutné, aby uživatel vybral atribut, který má jako své hodnoty značky tříd. Dále, aby vybral extraktory příznaků, pro nimiž získané příznaky chce získat statistiky.

Výpočet statistik, včetně nezbytné extrakce příznaků, se děje paralelně v za tímto účelem vytvořeném procesu, který zapouzdřuje třída *ExperimentStatsRunner*. Samotné výsledky jsou poté uloženy a předány hlavnímu procesu v objektu třídy *ExperimentDataStatistics*.

V záložce pro výběr selektorů příznaků je uživateli poskytnuta možnost vybrat si několik selektorů a určit pořadí v jakém budou aplikovány. Nejvrchnější selektor v seznamu je aplikován na příznaky datové sady jako první, další je aplikován na výsledek prvního a tímto způsobem dále. Selektory příznaků jsou objekty jejichž třídy dědí od abstraktní třídy *FeaturesSelector*.

Předposlední záložkou je záložka pro výběr klasifikátorů. Zde si může uživatel vybrat jaké klasifikátory chce testovat a s jakým nastavením. Každý klasifikátor je objektem jehož třída dědí od abstraktní třídy *Classifier*. Pořadí, bráno od vrchu, pouze určuje jak budou po sobě vyhodnocovány klasifikátory v jednom kroku validace. Klasifikátory jsou, kromě zmíněného pořadí, na sobě nezávislé.

Poslední záložka je určena pro výsledky. V momentu, kdy má uživatel sestaven experiment, tak spustí jeho provedení pomocí tlačítka, které se nachází v této záložce. Akce, která je svázána s tlačítkem, zapříčiní vytvoření objektu třídy *ExperimentRunner*, který pro svoji činnost vytváří proces na pozadí. Uživateli je dovoleno si dále prohlížet a měnit nastavení experimentu, zatímco v procesu na pozadí probíhá trénování a testování zvolených klasifikátorů. Pokud uživatel provede nějaké změny v experimentu, když proces na pozadí stále běží, tak tyto změny nebudou do výsledků probíhajícího experimentu zahrnuty. Proces na pozadí s hlavním procesem v průběhu experimentu komunikuje a poskytuje aktuální informace o stavu experimentu.

Výsledky experimentu, v podobě časů a všech provedených klasifikací, jsou ukládány do objektu třídy *Results*, která je po skončení poslána hlavnímu procesu. Na základě získaných dat, v objektu třídy *Results*, jsou vypočítány metriky, které jsou posléze zobrazeny uživateli.

4.2 Manipulace s datovými sadami

Každý klasifikátor založený na strojovém učení se pravděpodobně neobejde bez dobré datové sady, na které by mohl být natrénován či použit. Aplikace očekává datovou sadu ve formátu *CSV* neboli *Comma-separated values*. Jedná se o textový formát, který se skládá z jednotlivých záznamů a jejich hodnot. Uveďme si základní podmínky, které jsou ve formátu definovány.

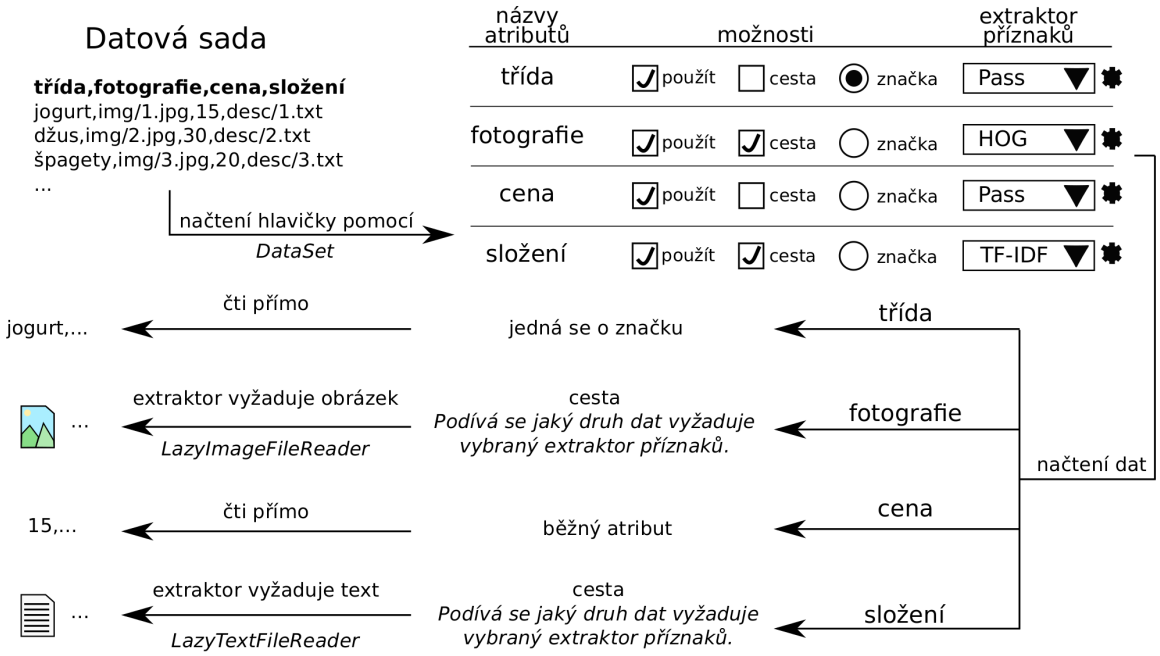
- Jednotlivé záznamy jsou od sebe odděleny koncem řádku. Poslední záznam může a nemusí být zakončen koncem řádku.
- Hodnoty jsou oddělené čárkami.
- Soubor může obsahovat volitelný záznam, který obsahuje hlavičku. Hlavička má stejný formát jako ostatní záznamy a vyskytuje se jako první v souboru. Hodnoty hlavičky jsou názvy polí, které se vyskytují v záznamech. Hlavička by měla mít stejný počet hodnot, jako všechny ostatní záznamy. Jinými slovy i všechny záznamy by měly mít stejný počet hodnot.
- Obsahuje-li hodnota čárku nebo znak konce řádku, tedy znaky pro formátování *CSV*, tak je hodnota uzavřena do uvozovek.
- Máme-li hodnotu co obsahuje znak uvozovky a kvůli výše popsaným důvodům je uzavřena do uvozovek, tak musíme použít escape sekvenci v podobě předřazení dalších uvozovek, abychom dali na vědomí, že uvozovky jsou součástí hodnoty.

Více detailní popis formátu lze nalézt v [40] . Aplikace vyžaduje, aby každý vstupní soubor datové sady měl hlavičku. Na základě této hlavičky zobrazuje uživateli názvy polí. Další záznamy jsou pak datové vzorky.

Jedno z polí musí obsahovat značení, které určuje do jaké třídy daný vzorek patří. Takovéto pole je pak uživatelem označeno v grafickém uživatelském rozhraní. Ostatní pole jsou pak automaticky, pokud nejsou označena pro vynechání, vybrána jako datová pole na nichž se klasifikátory mohu učit. Každé datové pole může obsahovat data přímo, nebo se odkazovat pomocí cesty na soubor jehož obsah má být použit. Jeden záznam může tedy obsahovat i více cest k několika různým souborům, jak je vidět v obrázku 4.1.

Jakým způsobem probíhá načtení datové sady, je možné též vidět na schématu 4.1. Poté co si uživatel zvolí svoji datovou sadu, je zobrazeno v grafickém uživatelském rozhraní nastavení pro jednotlivá pole. Mimo možností použít, cesta a značka, se objekt třídy *DataSet*, který načítá datovou sadu, rozhoduje o způsobu načtení daného pole také na základě vybraného extraktoru příznaků. Každý extraktor poskytuje informaci o tom, jaký druh dat očekává na svém vstupu. Aplikace dokáže pracovat s textem či obrázky, ale implementace umožňuje snadné rozšíření o další druhy dat jako by mohl být například zvuk. Továrna *LazyFileReaderFactory* vytvoří objekt třídy *LazyTextFileReader* nebo *LazyImageFileReader* na základě typu dat, který přijímá přiřazený extraktor příznaků.

Zmiňované třídy pro text a obrázky dědí od abstraktní třídy *LazyFileReader*. Slovo **lazy** v názvu tříd značí, že data nebudou načtena ihned, ale až v případě, kdy budou skutečně



Obrázek 4.1: Schéma načítání datové sady na základě uživatelského nastavení. Ikona obrázku a textového souboru značí vytvoření objektů tříd LazyImageFileReader a LazyTextFileReader.

požadována. Mimo to data, která jsou tímto způsobem načtena si objekt, který je načetl neuchovává a pouze je předá dál. Díky tomu mohou být po extrahování příznaků z paměti odstraněna a není nutné mít v paměti načtenou celou datovou sadu.

Mimo načítání datových sad umožňuje aplikace i vytváření nových datových sad z existujících. Jedná se o vytváření podmnožin sad, kdy můžeme vynechávat určité počty vzorků v jednotlivých třídách či vynechat třídu úplně. V návrhu je tato funkce, z pohledu uživatele, ilustrována na obrázku 3.3. O uložení nové datové sady se stará objekt třídy *DataSet*.

4.3 Úlohy vyžadující spuštění v separátním vlákně/procesu

Některé úlohy, které aplikace vykonává jsou výpočetně náročnější a jejich zpracování trvá delší dobu. Kdyby aplikace nespouštěla takovýto druh úloh paralelně k hlavnímu procesu, který má na starosti uživatelské rozhraní, tak by se uživateli mohlo jevit, že došlo k zaseknutí aplikace, protože při takovýchto úlohách přestane aplikace reagovat na uživatelské akce.

Pro tyto druhy úloh zde existuje abstraktní třída *ExperimentBackgroundRunner*. Tato třída dědí od třídy *Qthread* [12] z frameworku *PySide2*. *Qthread* slouží pro vytváření vláken. Pomocí signálů, definovaných v *PySide2*, lze vzájemně komunikovat s hlavním vláknem. Vlákno, které vykonává danou úlohu může posílat průběžné informace o pokračování úlohy a obdobně i hlavní vlákno může poslat například signál vyžadující předčasné ukončení. Základní sada signálů je implementována přímo v třídě *ExperimentBackgroundRunner*. Obsahuje následující signály:

- **numberOfSteps** - Odhadnutý počet kroků nutný k dokončení úlohy. Používá se pro inicializaci ukazatele průběhu.

- **step** - Signál, že došlo k dokončení kroku. Slouží pro posunutí ukazatele průběhu.
- **actInfo** - Aktuální informace o zpracovávání. Mělo by se jednat o krátkou výstižnou zprávu, která je zobrazena uživateli u ukazatele průběhu.
- **error** – Předání chybových zpráv. Zobrazuje se uživateli v dialogovém okně.
- **log** – Posílá informace, které by měly být ukládány do logu. Jedná se o detailnější informace než ty, které poskytuje signál actInfo.

Další signály si definují třídy, které od *ExperimentBackgroundRunner* dědí, sami. Jedná se o signály pro předání výsledků. Zahrnutí takovéhoto signálu již do základní třídy by nebylo příliš vhodné, protože se předání výsledků může lišit dle úlohy.

Uživatelské rozhraní umožňuje uživateli v jakýkoliv moment ukončit provádění úlohy na pozadí. Aby bylo toto možné, nestačí pouze použití vlákna *Qthread*. Z důvodu, že vlákno vyžaduje, pro řádné předčasné ukončení, odchycení signálu v rámci prováděného kódu. Mělo by se tedy například cyklicky dotazovat, zdali nebylo ukončeno. Problém je, že takovéto vlákno by nebylo schopno odchytit jím vyžadovaný signál o ukončení, protože by nastal problém, který ilustruje pseudokód 3. Dílčím krokem úlohy by mohlo být například trénování klasifikátoru, které může trvat potenciálně velice dlouho a proces by byl ukončen až poté, pak by se v očích uživatele mohlo zdát, že tlačítko pro ukončení úlohy nefunguje.

Algoritmus 3 Ilustrace problému odchytnutí signálu pro ukončení vlákna *Qthread*.

```
1: Inicializace vlákna.
2: while Nedostal jsem signál pro ukončení. do
3:     Provádění dílčího kroku úlohy.
```

Z výše zmíněných důvodů spustíme ve vytvořeném vlákně nový proces pomocí standardního pythonovského balíku *multiprocessing* [1]. Samotná úloha běží ve vytvořeném procesu a vlákno poté pouze kontroluje signál o ukončení. Dále slouží jako prostředník pro předání zpráv mezi procesem a grafickým uživatelským rozhraním. V případě přijmutí signálu o ukončení pošle procesu signál terminate (pro UNIX SIGTERM). Řešení problému je nastíněno pseudokódem 4.

Algoritmus 4 Ilustrace řešení problému odchytnutí signálu pro ukončení vlákna *Qthread*.

```
1: Inicializace vlákna.
2: Vytvoření procesu a komunikačního kanálu pro komunikaci s ním.
3: while Nedostal jsem signál pro ukončení a vytvořený proces stále pracuje. do
4:     Příjem a předání zprávy mezi procesem a grafickým uživatelským rozhraním, pokud
       nějaké zpráva existuje. Na zprávu čekáme maximálně po čas t.
5: Ukončení procesu, pokud stále běží.
6: Odeslání nevyřízených zpráv uživatelskému rozhraní.
```

Implementované úlohy, které budou popsány dále v této části, nemohou svým předčasným ukončením způsobit nestabilitu dat. Nicméně v případě rozšiřování aplikace o další úlohy je třeba na možnost předčasného ukončení pamatovat.

Použití samostatného procesu přináší ještě jednu výhodu. Interpret jazyka python *CPython*, používá *GIL* (global interpreter lock), který znemožňuje běh nově vytvořeného vlákna na jiném jádře procesoru, čímž nedostaneme sílu, kterou bychom dostali při použití více jader [8][19]. Naopak použití procesů tento nedostatek nemá.

4.3.1 Úloha získávající statistiky o datové sadě

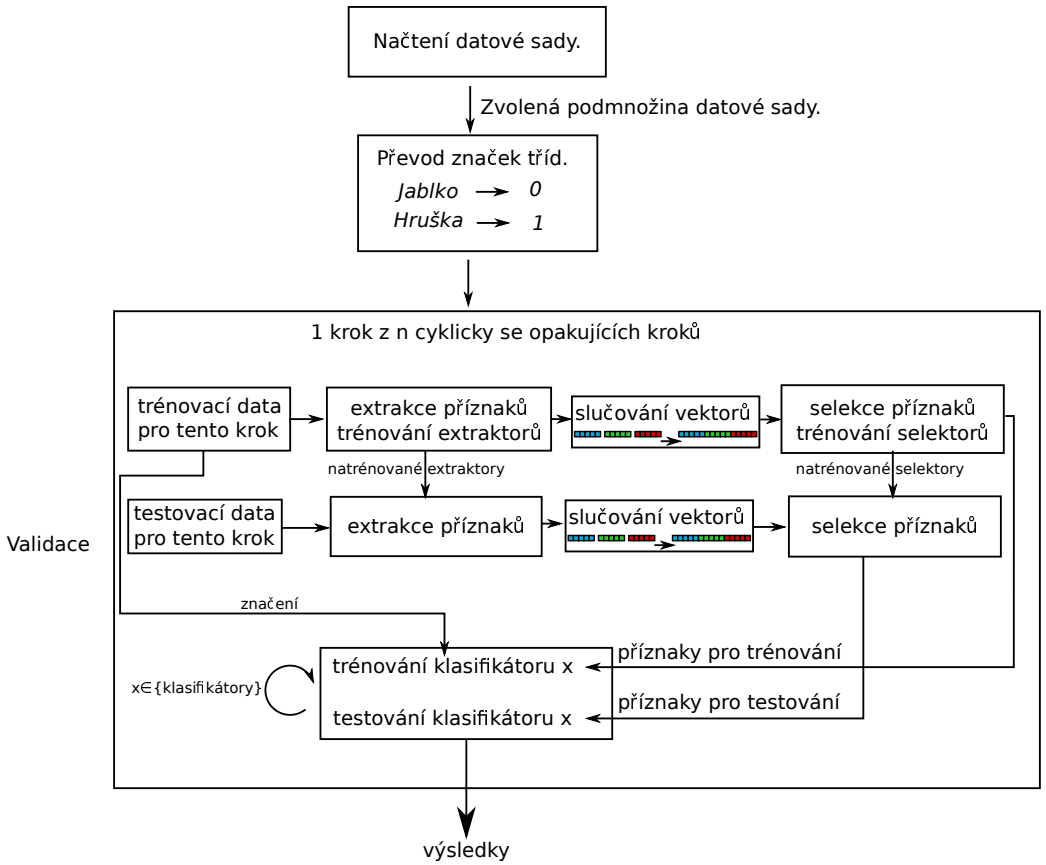
Aplikace dovoluje, aby si uživatel nechal vyhotovit statistiky o datové sadě, které zahrnují i statistiky o příznacích, které může být nutné nejprve extrahovat, což většinou bývá časově náročnější proces.

Z tohoto důvodu je v aplikaci implementována třída *ExperimentStatsRunner*, která dědí od abstraktní třídy *ExperimentBackgroundRunner*. Statistiky jsou vždy vyhodnocovány na aktuálně používané podmnožině vstupní datové sady. O průběžném pokroku je uživatel informován pomocí výše popsaných zpráv.

Tato úloha může být v rámci provádění přerušena aniž by tím poškodila vstupní datovou sadu, protože ji používá pouze pro čtení.

4.3.2 Úloha provádějící experimentování s klasifikátory

Jedná se o úlohu, která plní hlavní účel celé aplikace a to sice otestování klasifikátorů na dané datové sadě. Průběh experimentu znázorňuje schéma 4.2, o které se tento text dále bude opírat.



Obrázek 4.2: Schéma znázorňující průběh experimentování s klasifikátory.

Po vytvoření procesu, pomocí objektu třídy *ExperimentRunner*, je v novém procesu načtena datová sada. Po načtení se provede překódování značek tříd do číselné reprezentace. Díky převodu ušetříme nejen paměť počítače, ale dostaneme i vhodnější reprezentaci pro některé klasifikátory, jako jsou například neuronové sítě.

Jako další krok spustíme uživatelem vybraný druh validace. Jedná se o objekt třídy, která dědí od abstraktní třídy *Validator*. Samotná validace začíná voláním metody *run*, kterou implementuje zmiňovaný objekt. Spuštěním validace se dostáváme do hlavního (největšího) bloku ve schématu. Validace se typicky skládá z několika validačních kroků, které se liší výběrem dat použitých pro trénování a testování. Výběrem těchto dat validační krok začíná. Poté jsou nad trénovacími daty natrénovány extraktory příznaků a také pomocí těchto extraktorů jsou získány příznaky. Extraktory příznaků jsou objekty tříd, které dědí od abstraktní třídy *FeatureExtractor*. Každý extraktor má přiřazenou svoji část dat. Pro každou z nich dostáváme jeden příznakový vektor od daného extraktoru. My však chceme pro jeden vzorek datové sady pouze jeden vektor, který daný vzorek reprezentuje. Z toho důvodů jsou všechny získané vektory sloučeny do jednoho d dimenzionálního vektoru, kde d je dán součtem jednotlivých dimenzí vektorů. Dále je provedena selekce příznaků, pokud je zvolen alespoň jeden selektor. Selektory dědí od abstraktní třídy *FeaturesSelector*.

V této fázi máme již natrénovány všechny extraktory příznaků a případně i všechny selektory. Můžeme tedy provést extrakci, slučování a selekci pro testovací data s naučenými extraktory a selektory nad trénovacími daty. Pro testovací data musíme získávat příznaky až po získání trénovacích příznaků, protože předpokládáme, že si extraktor tvoří model ze vstupních dat, který používá i pro testovací data. Některé extraktory to sice dělat nemusí, ale je nutné předpokládat obecnější případ. Obdobné platí i pro fázi selekce příznaků.

Již se dostáváme k samotným klasifikátorům. Povšimněte si, že příznaky pro trénování a testování jsou pro všechny klasifikátory stejné, a tudíž jsme si mohli dovolit je nezískávat pro každý z klasifikátorů zvlášť. Tato skutečnost se vyplatí zvláště při experimentování s větším množstvím klasifikátorů zároveň. Nevýhoda tohoto přístupu od přístupu, kdy bychom získávali příznaky pro každý klasifikátor zvlášť je, že musíme uchovávat v paměti zároveň jak trénovací, tak testovací příznakové vektory.

Na konci validačního kroku jsou postupně jednotlivé klasifikátory natrénovány a vzápětí otestovány. Každý klasifikátor je objektem třídy dědící od abstraktní třídy *Classifier*. Výsledky (predikce a časy) jsou uloženy a pokračujeme dalším krokem. Pokud je již validace hotová, odešleme výsledky do procesu, ve kterém běží grafické uživatelské rozhraní.

V průběhu provádění experimentu proces informuje o provádění dílčích kroků a odesílá logovací informace. Stejně jako u *ExperimentStatsRunner* může i zde dojít k předčasnému ukončení na vyžádání uživatele. Proces opět nijak nemění datovou sadu, která je použita, a tedy při náhlém ukončení by nemělo dojít k jejímu poškození.

4.4 Systém zásuvných modulů

V této sekci si blíže vysvětlíme vytvořený systém zásuvných modulů, který byl vytvořen z důvodu co nejsnazší rozšiřitelnosti aplikace.

Všechny zásuvné moduly dědí od abstraktní třídy *Plugin*. Rozlišujeme však zásuvné moduly, které jsou instalovatelné a ty, které instalovány být nemohou. Přidání neinstalovatelných zásuvných modulů spočívá v úpravě zdrojových kódů aplikace. Ukažme si zde výčet používaných zásuvných modulů. V závorce jsou uváděny abstraktní třídy, od kterých musí daný druh zásuvného modulu dědit. Výčet je následující:

- instalovatelné
  - extraktory příznaků (*FeatureExtractor*)
  - klasifikátory (*Classifier*)

- neinstalovatelné
  - validátory (*Validator*)
  - selektory příznaků (*FeaturesSelector*)
  - zásuvné moduly provádějící normalizaci příznaků (*BaseNormalizer*)

Instalovatelné zásuvné moduly jsou tvořeny jako samostatné pythonovské balíky, které mají speciálně vytvořený soubor *setup.py*. Speciální nastavení tohoto souboru spočívá v udání vstupního bodu v definovaném formátu [4]. Pro klasifikátor, který je uložen v modulu m a název jeho třídy je C, by příslušný řádek *setup.py* mohl vypadat následovně:

```
entry_points={'classmark.plugins.classifiers': 'C = m:C'}
```

Pro extratkor příznaků E v modulu m:

```
entry_points={'classmark.plugins.features_extractors': 'E = m:E'}
```

Díky tomuto přístupu je možné zásuvné moduly instalovat a doinstalovávat pomocí nástroje *pip*[1] a to včetně jejich dodatečných závislostí.

Neinstalovatelné zásuvné moduly, jak již bylo řečeno, můžeme přidat pouze tak, že upravíme přímo zdrojové soubory aplikace. Tento druh zásuvných modulů byl vytvořen z důvodů, že instalovatelné balíky, přes zmiňované výhody, nejsou vždy zcela vhodné, protože je nutný přece jenom nějaký kód navíc. Mimo to u neinstalovatelných zásuvných modulů je kladen předpoklad, že jejich rozšiřování nebude tolik časté a aplikace by měla obsahovat všechny již od prvního spuštění.

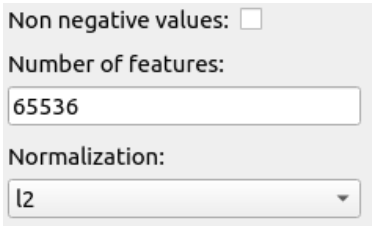
Abstraktní třída *Plugin*, za pomocí tříd *PluginAttribute* a *AttributesWidgetManager*, dovoluje snadnou implementaci atributů zásuvného modulu, které by mohly být nastavitelné z grafického uživatelského prostředí. Třída *PluginAttribute* definuje následující druhy atributů:

- **CHECKABLE**

  Tento druh je vhodný pro booleovské atributy. V grafického uživatelském rozhraní se objevuje jako zaškrtnutelné tlačítko (snímek 4.3).

  Příklad použití:

  ```
  self._nonNegativ=PluginAttribute("Non negative values",
     PluginAttribute.PluginAttributeType.CHECKABLE, bool)
  ```



Obrázek 4.3: Snímek obrazovky znázorňující podobu atributů: CHECKABLE, VALUE a SELECTABLE.

[1] Nástroj pro instalaci pythonovských balíků.

- **VALUE**
  Vhodný například pro atributy, které mohou nabývat číselné hodnoty či řetězce. Má podobu běžného pole pro zadávání hodnot (snímek 4.3).
  Příklad použití:

```
self._nFeatures=PluginAttribute("Number of features", PluginAttribute.
    PluginAttributeType.VALUE, int)
```

- **SELECTABLE**

  Pokud daný atribut má na výběr z několika málo hodnot, pak je vhodné použít tento druh. Vizuálně se zobrazuje jako pole s možností výběru (snímek 4.3).

  Příklad použití:

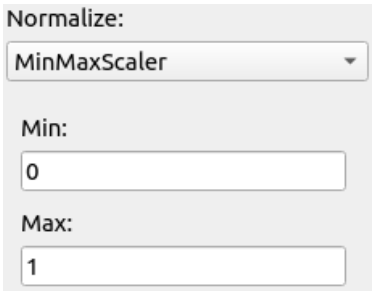
```
self._norm=PluginAttribute("Normalization", PluginAttribute.
    PluginAttributeType.SELECTABLE, str,[None,"l1","l2",])
```

- **SELECTABLE_PLUGIN**

  Chceme-li mít atribut, který umožňuje výběr jednoho z několika zásuvných modulů, tak použijeme tento druh atributu. Vzhledem k tomu, že vybíráme zásuvný modul, tak je tu možnost, že i ten bude mít vlastní atributy, které jsou nastavitelné. Aplikace pamatuje i na tento případ a takovéto atributy budou vloženy pod dané pole, tak jak je vidět na snímku obrazovky 4.4.

  Příklad použití:

```
self._normalizer=PluginAttribute("Normalize", PluginAttribute.
    PluginAttributeType.SELECTABLE_PLUGIN, None, [None,
    NormalizerPlugin, MinMaxScalerPlugin, StandardScalerPlugin,
    RobustScalerPlugin])
```



Obrázek 4.4: Snímek obrazovky s vizuální podobu atributu druhu SELECTABLE_PLUGIN.

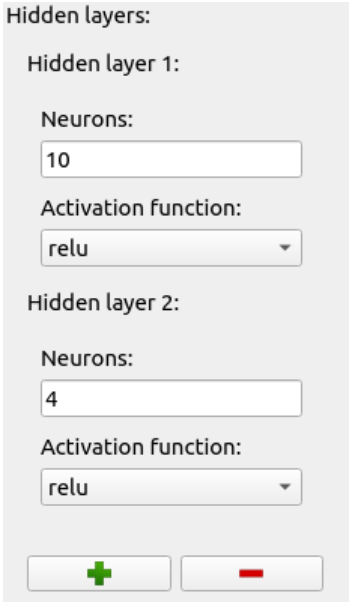
- **GROUP_PLUGINS**

  Jedná se o podobný druh atributu jako SELECTABLE_PLUGIN. Platí pro něj stejné pravidla co se týče vnořování atributů vybraných zásuvných modulů, ale na rozdíl od SELECTABLE_PLUGIN si uživatel nevybírá jeden zásuvný modul, nýbrž několik stejných zásuvných modulů. Vizuální podoba je zachycena na snímku obrazovky 4.5.

  Příklad použití:

```
self._hiddenLayers=PluginAttribute("Hidden layers", PluginAttribute.
    PluginAttributeType.GROUP_PLUGINS, Layer)
```

Třída *Layer* je zásuvný modul přímo vytvořený pouze pro účel získání nastavení pro skrytou vrstvu. Obsahuje dva atributy: počet neuronů a aktivační funkci.



Obrázek 4.5: Snímek obrazovky s vizuální podobu atributu druhu GROUP_PLUGIN.

Abstraktní třída *Plugin* v sobě přímo implementuje podporu logování. Což se hodí hlavně pro extraktory příznaků a zejména pak pro klasifikátory. Umožňuje totiž posílat logovací zprávy z běžícího procesu experimentu do procesu s grafickým uživatelským rozhraním a to pomocí jednoduchého příkazu:

```
self._logger.log("zprava")
```

Plugin má totiž ve své statické proměnné objekt třídy *Logger*, která implementuje dva návrhové vzory *Singleton* a *Observer*. Díky tomu, že se jedná o *Singleton*, máme jeden sdílený objekt, na kterém si pomocí návrhového vzoru *Observer* zaregistrujeme, že chceme přijímat dané logy. Následně se tyto zprávy předají do grafického uživatelského rozhranní.

Každý zásuvný modul je povinen implementovat metody:

- getName() - Vrací název zásuvného modulu.
- GetNameAbbreviation() - Vrací zkratku názvu modulu.

4.4.1 Extraktory příznaků

V této podsekci si uvedeme jaké extraktory příznaků jsou s aplikací v této práci dodávány. Popis jejich nastavitelných parametrů lze nalézt v příloze A. Výčet extraktorů je následující:

- **Pass**
  Ve své podstatě se nejedná o extraktor příznaků, ale jen o převodník uložených příznaků (číselných) do podoby pro další zpracování. Použijeme jej tedy v momentu, kdy máme již číselné příznaky přímo v datové sadě.

- **Hashing**

  Jedná se o extraktor příznaků z textu pomocí metody: Hašování příznaků (2.2.3). Implementován je jako obálka nad *HashingVectorizer* z *scikit-learn*.

- **Histogram of Oriented Gradients**

  Tento extraktor implementuje metodu: Histogram orientovaných gradientů (2.2.4). Tato metoda vyžaduje obrázek na vstupu. Implementován je jako obálka nad *hog* z *scikit-image*[13].

- **Term Frequency–Inverse Document Frequency**

  Extraktor příznaků z textu, pracující metodou TF-IDF (2.2.2). Implementován je opět formou obálky nad *TfidfVectorizer* z *scikit-learn*.

4.4.2 Klasifikátory

Podobně jako u extraktorů příznaků si i pro klasifikátory popíšeme přehled dostupných možností, které jsou dodány v rámci této práce. Popis jejich nastavitelných parametrů se nachází opět v příloze A. Jedná se o tento výčet klasifikátorů:

- **Support Vector Machines**

  Tento klasifikátor je obálkou nad dvěma klasifikátory z *scikit-learn* a to sice *LinearSVC* a *SVC*. Jak je dle názvu patrné jedná se o SVM klasifikátor, který je popsán v sekci 2.5.

- **Classification by evolutionary estimated functions**

  Jedná se o klasifikátor používající evoluci pro hledání funkcí, které jsou použity k určení míry příslušnosti vzorku k dané třídě. Tento klasifikátor byl celý implementován v rámci této práce a popis jeho implementace je možné nalézt v sekci 4.5.

- **k-Nearest Neighbor**

  Klasifikátor používající metodu K-nejbližších sousedů 2.7. Implementace je založena na třídě *KneighborsClassifier* z *scikit-learn*.

- **Decision Tree Classifier**

  Provádí klasifikaci pomocí rozhodovacího stromu 2.4. Realizován je pomocí *KneighborsClassifier* z *scikit-learn*.

- **Naive Bayes Classifier**

  Jedná se o použití Naivního Bayesova klasifikátoru 2.8. Implementace využívá třídy *MultinomialNB* a *GaussianNB* z *scikit-learn*.

- **Artificial Neural Networks**

  Klasifikace pomocí umělých neuronových sítí. Implementace je založena na frameworku *Keras* s *Tensorflow*.

4.5 Implementace klasifikátoru používajícího evolučně odhadované funkce

Při popisu tohoto klasifikátoru budou často používány pojmy, které byly uvedeny v sekci 2.9. Ve zmiňované sekci byl uveden způsob klasifikace (za použití symbolické regrese), který se tomuto vzdáleně blíží v tom smyslu, že používá evoluční algoritmus pro nalezení modelu (hranice) oddělujícího jednotlivé třídy. Zde uvedený algoritmus jde však na klasifikaci jinou cestou. Jak si uvedeme dále, oddělující hranici přímo nehledá a používá evoluci jiným způsobem.

Hlavní myšlenkou tohoto algoritmu je nalezení funkce f_c, pro každou třídu c z množiny tříd C, do kterých chceme klasifikovat. Tato funkce vrací hodnotu skóre udávající jak moc klasifikovaný vzorek do dané třídy patří. Klasifikace je pak provedena dle:

$$\arg\max_{c \in C} f_c(x), \tag{4.1}$$

kde x je klasifikovaný vzorek. Vybíráme tedy tu třídu, jejichž funkce vrací největší skóre. Pokud máme více stejných maxim, tak vybereme jedno z nich. Ovšem je nutné si položit otázku jak takovéto funkce získat.

Podívejme se na tento problém jako na optimalizační problém, kdy se budeme pokoušet najít funkce f_c, které nad trénovací datovou sadou D dávají největší hodnotu výrazu:

$$|\{x | x_c = \arg\max_{c \in C} f_c(x) \wedge x \in D\}|, \tag{4.2}$$

kde x_c značí třídu, do které patří vzorek x. Slovně tedy hledáme takovou sadu funkcí, která dává největší počet dobře klasifikovaných trénovacích vzorů. Tato úloha bude řešena pomocí genetických algoritmů, které v optimalizačních problémech nachází své uplatnění.

Samotný průběh algoritmu sleduje klasické schéma evolučních algoritmů, tak jak bylo popsáno v sekci 2.9. Způsob výběru rodičů pro křížení lze nastavit pomocí grafického uživatelského rozhraní. Na výběr je ruletová a pořadová selekce. Nahrazení je realizováno pomocí steady-state modelu. Konkrétně se jedná o zmiňovanou podobu, kdy vybereme dva rodiče, k nim vytvoříme dva potomky a nakonec vrátíme do populace dva nejlepší z množiny rodičů a jejich potomků. Popis křížení, mutací a fitness funkce si uvedeme dále v samostatných podsekcích.

4.5.1 Chromozom jedince

Jedním z prvních kroků při vytváření genetického algoritmu je návrh chromozomu, který si ponese každý jedinec v populaci a reprezentuje zmíněný problém v prostoru evolučního algoritmu. Navrženou podobu chromozomu ilustruje obrázek 4.6. Máme zde jeden chromozom pro klasifikaci do tří tříd A,B a C. Vizuálně jsou odděleny nezávislé části pro jednotlivé třídy. Nyní se budeme orientovat na libovolnou část přiřazenou jedné třídě.

Obrázek 4.6: Možná podoba jednoho chromozomu při hledání funkcí ke klasifikaci. Mezery pouze vizuálně oddělují části chromozomu od sebe.

V takovéto části jsou body p, které jsou v rámci jedné třídy různé, a metoda g (každá třída má svoji). Body p mají počet dimenzí odvíjející se od počtu dimenzí příznakového vektoru z trénovací sady, ale navíc mají ještě hodnotu skóre. Každý bod si tedy můžeme představit tak, že se skládá ze dvou částí: části p_x, která odpovádá možným hodnotám příznakového vektoru, a části p_s, která přiřazuje skóre na souřadnicích p_x. Metoda g dává pro přiřazenou množinu bodů P funkci g_f, která se snaží, aby danými přiřazenými body procházela. Tedy, aby platilo.

$$\forall p \in P : g_f(p_x) = p_s. \tag{4.3}$$

Teoreticky by se mohla pouze hodnotám skóre blížit. Je ovšem důležité, aby funkce byla definována i v jiných bodech, konkrétně pro všechny možné hodnoty, kterých může příznakový vektor nabývat. Pokud by tomu tak nebylo, tak by klasifikátor nebyl schopen generalizovat.

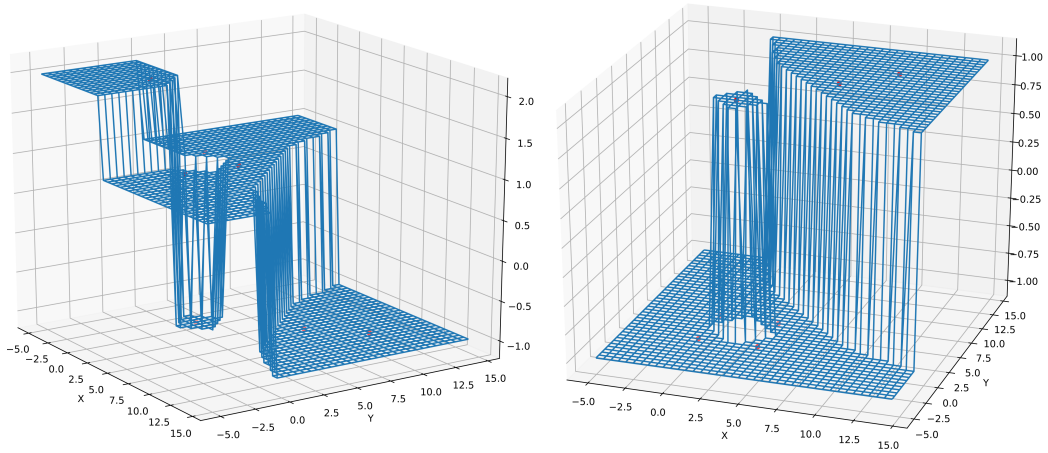
Funkci g_f, pro danou třídu c, bychom mohli ztotožnit s funkcí f_c. Tímto způsobem dostáváme transformaci do prostoru fenotypů.

Když tvoříme počáteční populaci, tak pro každou třídu vybereme minimálně jeden příznakový vektor, který patří do dané třídy, a minimálně jeden příznakový vektor, který do ní nepatří. Maxima si nastavuje uživatel. Tím získáme části p_x pro minimálně dva body p. Zbývající část skóre p_s určíme tak, že bodům patřících do dané třídy přiřadíme skóre 1 a ostatním, které do třídy nepatří, přiřadíme skóre -1.

V rámci provádění evolučního algoritmu, pak pouze měníme funkce g, skladbu vybraných bodů a skóre těchto bodů. Souřadnice bodů p_x jsou ve všech generacích vždy převzaté od nějakého vzorku z trénovací sady. V sekci o mutacích je diskutována i možnost vytváření p_x, které se v trénovací sadě nevyskytují, ovšem v zde uváděné implementaci se tento přístup nepoužívá.

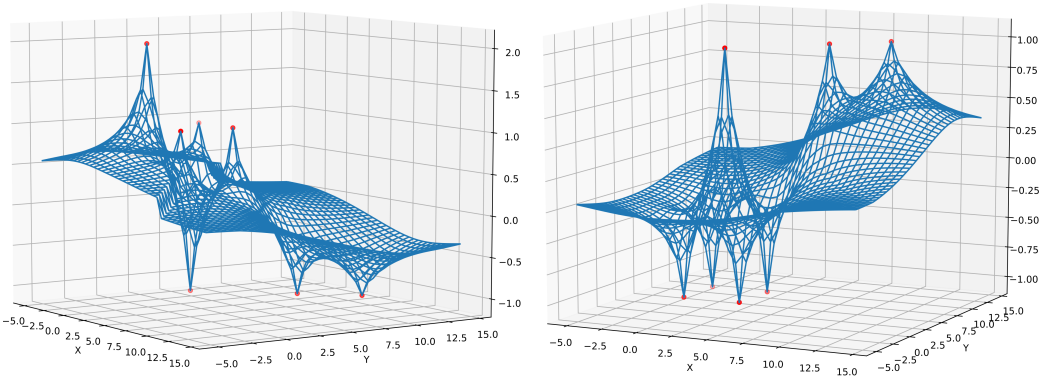
4.5.2 Implementované metody g

V této podsekci si uvedeme výčet implementovaných metod g, avšak teoreticky je možné použít i libovolné jiné (např. hledat funkce pomocí symbolické regrese).



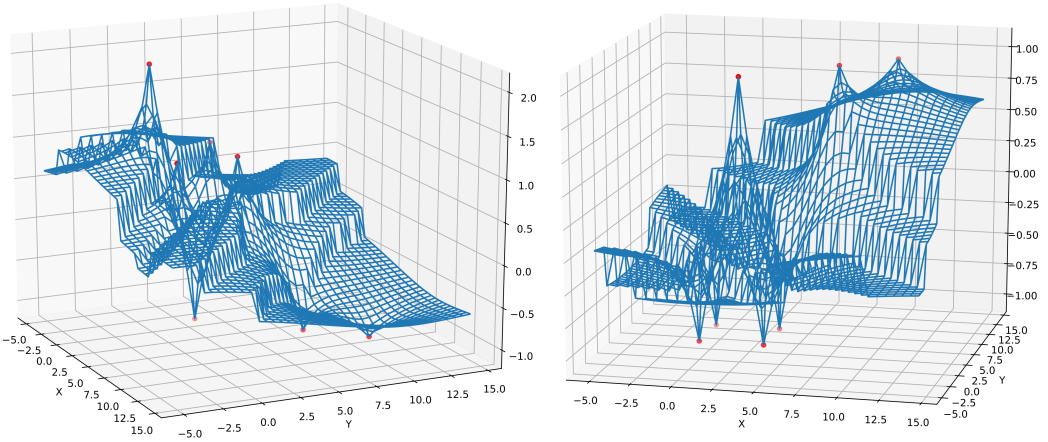
Obrázek 4.7: Ilustrace jakým způsobem pracuje 1. popsaná metoda g. Na obrázku jsou dvě funkce pro klasifikaci do dvou tříd. Červené tečky značí vybrané body. Používá 2D příznakové vektory se souřadnicemi na osách x a y. Skóre je na vertikální ose (z).

První použitou metodou, kterou si popíšeme, je jednoduchá metoda, která pro daný příznakový vektor x vždy přiřadí skóre p_s nejbližšího bodu $p \in P$. Vzdálenost je měřena euklidovsky na části p_x. Možná podoba výsledných funkcí je zachycena na obrázku 4.7.



Obrázek 4.8: Ilustrace jakým způsobem pracuje 2. popsaná metoda g. Na obrázku jsou dvě funkce pro klasifikaci do dvou tříd. Červené tečky značí vybrané body. Používá 2D příznakové vektory se souřadnicemi na osách x a y. Skóre je na vertikální ose (z).

Další metoda najde nejbližší dva body, které mají skóre větší než nula (spíše do třídy patří), a další nejbližší dva body, které mají skóre menší nebo rovné nule (spíše nepatří). Pokud množina P neobsahuje dostatečný počet bodů, klesneme na výběr méně bodů. Teoreticky může dojít k situaci, kdy nemáme například žádné body, které mají skóre s hodnotou menší nebo rovnou nule (či naopak), pak tyto body neuvažujeme.



Obrázek 4.9: Ilustrace jakým způsobem pracuje 3. popsaná metoda g. Na obrázku jsou dvě funkce pro klasifikaci do dvou tříd. Červené tečky značí vybrané body. Používá 2D příznakové vektory se souřadnicemi na osách x a y. Skóre je na vertikální ose (z).

Výsledná hodnota skóre pro příznakový vektor x je pak určena váženým průměrem skóre jednotlivých bodů a váha je dána na základě vzdálenosti, tedy:

$$\frac{\sum_{p \in P_s} p_s d(p_x, x)}{\sum_{p \in P_s} d(p_x, x)}, \tag{4.4}$$

kde P_s je množina vybraných bodů a d je vzdálenostní funkce, která určuje stejně jako v minulém případě euklidovskou vzdálenost bodu p (p_x) od příznakového vektoru x. Ilustrace funkcí je na obrázku 4.8.

Poslední implementovaná metoda pracuje stejným způsobem jako předešlá, ale přidává do váženého průměru až dva další body, které jsou nejblíže bez ohledu na to, jaké mají skóre. Možné podoby výsledných funkcí jsou zobrazeny na obrázku 4.9.

4.5.3 Mutace

Tato implementace genetického algoritmu nepoužívá pouze jeden operátor mutace, ale definuje jich hned několik. Větší množství operátorů je používáno především z důvodu, že i samotná velikost chromozomu je předmětem evoluce, abychom se nemuseli omezovat pouze na velikost fixní.

Počet mutací, respektive maximální počet mutací, v rámci fáze, kdy dochází k mutování jedince, může uživatel nastavovat. Nicméně skutečný počet je vždy náhodně před samotnou mutací zvolen a může dojít i k úplnému vynechání mutace.

Uveďme si zde všechny implementované druhy mutací:

- **mutace vzorku**

  Náhodně vybere gen, který reprezentuje vybraný bod (geny označeny písmenem p na obrázku 4.6), a provede změnu za nový bod p. Tomuto novému bodu bude přiřazen náhodný příznakový vektor, který zatím v množině bodů není, z trénovací množiny a jeho skóre bude 1, pokud patří do třídy jejíž přiřazenou část chromozomu právě mutujeme, jinak -1.

- **přidání/odstranění genů pro body**

  Náhodně vybereme část chromozomu přiřazenou určité třídě. Dále se náhodně rozhodneme, zdali budeme přidávat nebo ubírat gen. Možnost ubírání genu bude zakázána v případě, kdy daná část chromozomu má již pouze jeden bod. V případě odebírání pouze náhodně odstraníme z uvažované části chromozomu gen reprezentující bod. V případě přidání genu rozšíříme danou část chromozomu o jeden gen, který inicializujeme náhodně vytvořeným bodem. Inicializace bodu probíhá stejně jako při mutaci vzorku.

- **mutace skóre**

  Náhodně vybereme jeden z genů pro body. Poté si náhodně vygenerujeme číslo v intervalu <-1,1>, které přičteme k hodnotě skóre. Teoreticky hodnota skóre může nabývat libovolných číselných hodnot. Může i dojít k případu, kdy se vzorek původně prohlášený, že do dané třídy patří (v počáteční populaci měl skóre 1), změní na vzorek, který do třídy spíše nepatří (skóre menší nebo rovno nule). Díky tomu se může potlačit negativní vliv špatně označeného vzorku v trénovací sadě.

- **mutace metody pro tvorbu funkce**

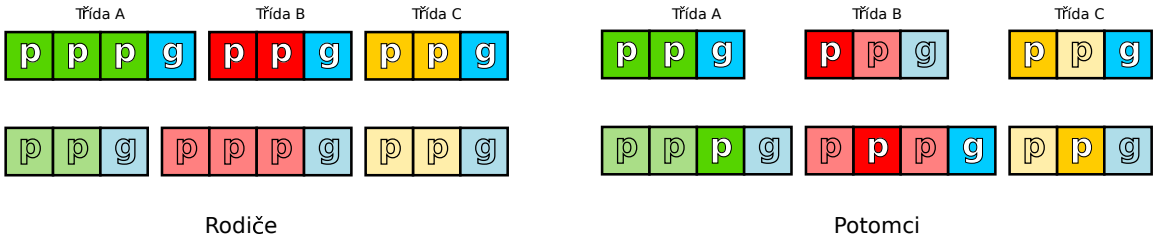
  Náhodně vybere gen, který reprezentuje metodu (geny označeny písmenem g na obrázku 4.6), a náhodně vybere jednu z implementovaných metod, kterou použije pro výměnu stávající.

Bylo by jistě možné používat i další druhy mutací. Například by šlo vytvářet zcela nové body p, které by nemusely mít část p_x zcela převzatou od vzorku z trénovací množiny,

ale jednalo by se o souřadnice p_x, které jsou například od těchto vzorků pouze odvozené. Jeden z takových přístupů by mohl fungovat tak, že náhodně vybereme n vzorků z trénovací množiny patřící k dané třídě a vytvoříme bod, který by měl část p_x rovnou průměru z vybraných bodů, tedy jednalo by se o bod, který by byl ve středu vybraných vzorků. Dále by bylo možné vybrat náhodně bod patřící do dané třídy a posunout se od něj ve směru zvoleného vektoru o délku danou tímto vektorem, čímž bychom získali nové souřadnice p_x.

4.5.4 Křížení

Křížení probíhá vždy mezi dvěma náhodně zvolenými rodiči a křížíme pouze části chromozomů, které odpovídají stejné třídě, tak jak ilustruje obrázek 4.10.



Obrázek 4.10: Ukázka použitého uniformního křížení v implementovaném klasifikátoru.

Dále budeme mít namysli vždy části chromozomu (z obou rodičů) spadající pod konkrétní stejnou třídu. Tedy při křížení neuvažujeme chromozom jako celek, ale postupujeme po částech. Na úrovni dílčích částí chromozomu se pak jedná o uniformní křížení, které probíhá následovně. Nejprve náhodně vybereme rodiče, ze kterého dostane první potomek gen typu g. Druhý potomek dostane gen g od druhého rodiče.

Následně iterujeme přes délku delší části genů p a vždy náhodně vybereme rodiče, ze kterého zvolíme gen dle aktuální pozice. Vybraný gen přidáme do příslušné části v prvním potomkovi. Gen nevybraného rodiče pak uložíme do druhého potomka. V případě, kdy je pro potomka vybrán rodič, který na aktuální pozici nemá gen, tak nový gen potomek nedostane a nový gen dostane pouze druhý potomek od druhého rodiče. V rámci křížení se dále hlídá, zdali vybraný příznakový vektor aktuálně tvořený potomek neobsahuje, pokud ano, tak se gen zahodí.

4.5.5 Fitness funkce

Hodnota fitness je určována na základě fenotypu (hotového klasifikátoru). Její hodnota je dána zlomkem dobře klasifikovaných vybraných vzorů z celkového počtu vybraných vzorů. Obyčejně se pro vyhodnocení fitness funkce používá celá trénovací množina. Uživatel má však možnost si nastavit použití pouze zlomku trénovací množiny (urychlí trénování).

Dále je možné nastavit, používáme-li pouze podmnožinu trénovací množiny, aby se tato podmnožina v každé generaci náhodně změnila. To ovšem zapříčiní znovu vyhodnocování fitness u již jednou vyhodnocených jedinců, protože jejich úspěšnost může být na této sadě jiná. Celková hodnota fitness je v tomto případě průměrem z jednotlivých ohodnocení.

4.6 Dokumentace zdrojového kódu programu

Tento text doplňuje programová dokumentace, která detailněji popisuje zdrojový kód programu. Dokumentace je psána přímo ve zdrojových souborech ve formě komentářů. Pro vygenerování byl použit nástroj Sphinx [14]. Nachází se na přiloženém elektronickém médiu.

Kapitola 5

Datové sady pro experimenty

Se systémem, jehož návrh je v kapitole 3, bude provedeno několik experimentů, pro které byly vybrány čtyři datové sady. Na každé z nich bude provedena praktická úloha jiného typu. Mimoto zmiňované datové sady obsahují různé druhy dat: text, biologická data a obrázky.

5.1 Určení jazyka textu

Pro úlohu určení jazyka z textu byla vybrána datová sada titulků[1] z webového portálu OpenSubtitles pro rok 2018. [31]

Datová sada obsahuje 3 735 070 souborů s titulky v 62 jazycích. Bohužel je však napříč jazyky velmi nevyvážená, co se týče počtů souborů s titulky. Rozpětí počtu souborů na jazyk je 4 – 446612. Z tohoto důvodu budou jazyky s méně soubory vyřazeny.

5.2 SPAM filtr

SPAM filtr lze realizovat jako binární klasifikátor textových dokumentů, který bude klasifikovat do tříd vyžádaná/nevyžádaná. Pro účely natrénování a otestování klasifikátoru se povedlo sestavit datovou sadu, která má dohromady 80 824 emailů z toho 40 063 SPAMů. Jedná se tedy o poměrně vyváženou sadu.

Tato datová sada se skládá z několika veřejně dostupných internetových zdrojů: enron[2], spamassassin publiccorpus[3] a TREC[4]. Většina e-mailů byla strojově přeložena do českého jazyka, což simuluje běžně přijímané SPAMy pro českého uživatele.

5.3 Zjištění biologické odezvy molekuly

Zde bude úkolem vytvořit a otestovat různé klasifikátory, které budou schopny zjistit biologickou odezvu molekuly z jejich chemických vlastností. Jelikož datová sada obsahuje odezvy označené jako 1 a 0, jedná se o binární klasifikaci.

[1] V době psaní práce dostupná na adrese: http://opus.nlpl.eu/OpenSubtitles2018.php
[2] V době psaní práce dostupné na adrese: http://www2.aueb.gr/users/ion/data/enron-spam/
[3] V době psaní práce dostupné na adrese: https://spamassassin.apache.org/old/publiccorpus/
[4] V době psaní práce dostupné na adrese: https://plg.uwaterloo.ca/~gvcormac/treccorpus07/

Každá molekula je reprezentována deskriptory d1-d1776. Dohromady datová sada obsahuje 3751 vzorků. Dataset pojmenovaný Bioresponse byl získán z webového portálu OpenML[5].

5.4 Rozpoznání objektu na obrázku

Pro tento úkol budeme testovat klasifikátory na sadě obrázků CIFAR-10[6]. [30] Tento dataset obsahuje 60000 barevných obrázků o rozměrech 32x32 pixelů. Každý z obrázků má přiřazenou jednu z 10 tříd (airplane, automobile, bird, ...). Pro každou třídu máme k dispozici 6000 obrázků.

5.5 Klasifikace květin

Experimentům podrobíme jednu často používanou malou datovou sadu květin. Nazývá se Iris dataset[7]. Obsahuje třídy: Iris-setosa (kosatec chlupatý), Iris-versicolor (kosatec pestrý) a Iris-virginica (kosatec virginský).

Pro klasifikaci lze použít 4 příznaky:

- délku a šířku kalichu
- délku a šířku okvětních lístků

[5] V době psaní práce dostupné na adrese: https://www.openml.org/d/4134
[6] V době psaní práce dostupné na adrese: https://www.cs.toronto.edu/~kriz/cifar.html.
[7] Datová sada byla pro účely této práce získána z adresy: https://www.kaggle.com/uciml/iris/version/2.

Kapitola 6

Experimenty

V této kapitole si uvedeme experimenty, které byly provedeny pomocí nástroje, který jsme si popsali v minulé kapitole. Experimenty mají za účel zjistit úspěšnosti (i doby trénování) jednotlivých klasifikátorů na různých problémech. Budeme se snažit zjišťovat vlivy jako jsou: velikost dat, normalizace příznaků, druh extrakce příznaků, velikost příznakového vektoru, počet tříd či selekce příznaků.

Pro testování klasifikátorů je ve většině případů použit jednotný přístup. Převážně bude použita křížová validace (*stratified*) nastavena na 10 kol. Tento přístup bude pouze pozměněn u datové sady *CIFAR-10*, která definuje vlastní rozdělení dat do trénovací a testovací množiny. Validace bude v tomto případě probíhat pouze na jedné definované trénovací a testovací množině.

Tabulky shrnující většinu sledovaných metrik k experimentům jsou v příloze B. Na tomto místě je také uveden seznam používaných zkratek a hodnoty parametrů klasifikátorů. Zde si budeme výsledky raději prezentovat přehledněji pomocí grafů. Vynecháme však některé sledované údaje. Například nebudeme uvádět časy pro klasifikaci dat naučeným klasifikátorem, ale pouze trénovací časy. Tyto časy jsou většinou velmi malé, až na *KNN*, u kterého trvá krátce trénování.

Vždy bude použit stejný druh selekce příznaků, který vybere příznaky na základě Gini importance [32] získané z rozhodovacího stromu. Vybrány jsou pak ty příznaky, které mají vyšší nebo stejné hodnoty než je medián těchto hodnot. Normalizace budou používány různé a jsou vždy uvedeny v příloze.

Ne u všech experimentů se budeme zaměřovat na to stejné. U některých budeme zjišťovat jak se mění vlastnosti v závislosti na počtu použitých vzorků dat, jindy se zase spíše zaměříme na měnící se velikost příznakového vektoru a tak podobně.

U experimentů bude uváděna doba trénování/testování, která vzhledem k implementaci zásuvných modulů může obsahovat i dobu potřebnou k normalizaci dat. Tato doba se do trénovací/testovací doby započítává v případě, kdy se normalizuje na úrovni zásuvného modulu klasifikátoru. Není však započítána v případech, kdy se bude v experimentech normalizovat již ve fázi extrakce příznaků. Tedy při použití HOG, TF-IDF a hašování příznaků.

6.1 Klasifikace květin

V tomto experimentu je sledován vliv velikosti podmnožin datové sady na úspěšnost a dobu trénování klasifikátorů. Je i sledován vliv použití normalizace a selekce příznaků. Je

použita nejmenší datová sada, která byla vybrána pro experimentování (5.5). Protože již přímo obsahuje příznaky pro klasifikaci, tak není nutné je extrahovat.

Z grafů v obrázku 6.1 je vidět, že všechny klasifikátory zvládaly tuto úlohu velice dobře. Dále je vidět, že příznaky v původním tvaru byly povětšinou dostačující a nebylo nutné je zlepšovat pomocí normalizace či selekce, až na *ANN*, kde se normalizace vyplatila. Nicméně kombinace selekce a normalizace neměla úspěch vůbec, opět až na *ANN*. Důvod nízké úspěšnosti při použití této techniky lze zjistit při detailnějším pohledu na data. Pro všechny klasifikátory (kromě *ANN*) byla použita normalizace, která dělí danou hodnotu příznaku velikostí celého příznakového vektoru. Selektor příznaků, v nejmenší podmnožině datové sady, vybral pouze jeden příznak, tedy se všechny hodnoty převedly na hodnotu 1. U dalších podmnožin datové sady vybral sice již 2 příznaky, které ovšem jsou stále málo a docházelo k následujícím problémům:

originál: (1,4; 0,2) | převedený: (0,99; 0,14)
originál: (4,7; 1,4) | převedený: (0,96; 0,29)

Tedy hodnota 4,7, která byla původně vyšší než 1,4, je v této reprezentaci nižší. U *ANN* k tomu nedocházelo, protože byla použita jiná normalizace.

Tento experiment také ukazuje, že u *CEEF* dochází u normalizovaných a vyselektovaných příznaků ke znatelnému zpomalení. Je to způsobeno tím, že v implementaci chceme přidávat do chromozomu pouze vzorky z trénovací množiny (část p_x nějakého bodu p), které v dané části chromozomu již nejsou. Musíme ovšem jejich unikátnost nejprve zkontrolovat a případně dohledat jiný vhodný vzorek, který se v dané části chromozomu již nenachází, což přidává čas navíc.

V tomto experimentu docházelo k navýšení kolizí při vybírání vzorku se vzorkem, který již v části chromozomu pro danou třídu byl a tím došlo ke zpomalení. Při odstranění této kontroly se takový vliv zpomalení, po použití selekce a normalizace dohromady, neprojevil (obrázek 6.2). Z těchto důvodů jsou zde uváděny zmíněné příčiny.

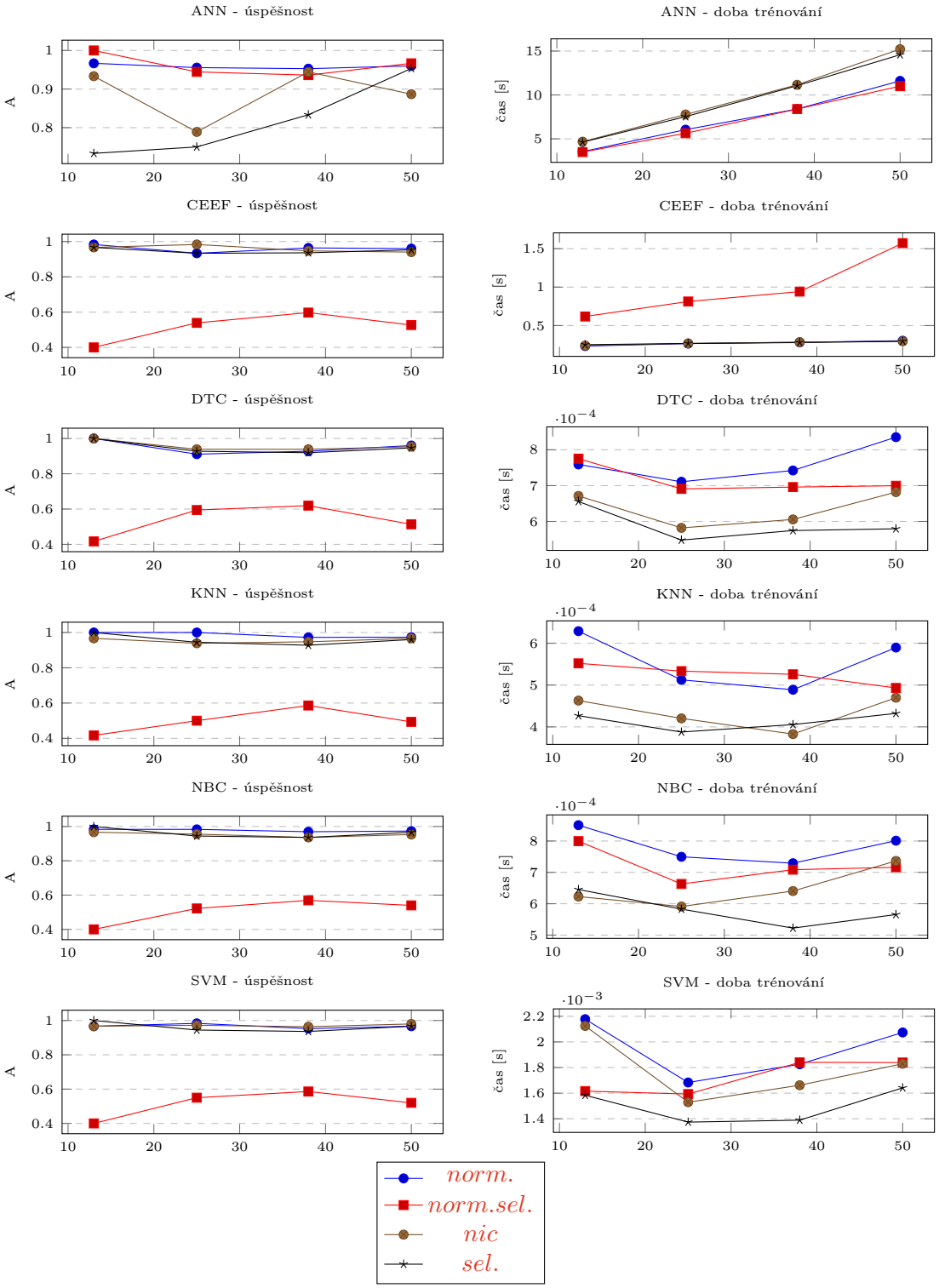
Výsledky všech klasifikátorů (s vhodným ne/použitím normalizace a selekce), které byly získány nad celou datovou sadu, jsou srovnatelné s nejlepšími prezentovanými v [17]. V době psaní práce nejlepší udávané výsledky ve zdroji dosahovaly něco málo přes 0,98 hodnoty správnosti.

6.2 Zjištění biologické odezvy molekuly

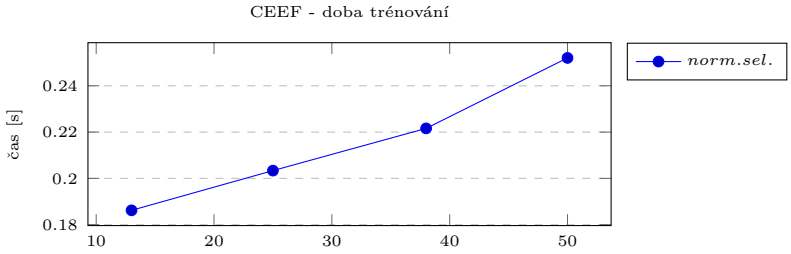
Tento experiment, se stejně jako minulý experiment 6.1, opět zaměřuje na sledování vlivu velikosti podmnožin datové sady na úspěšnost a dobu trénování klasifikátoru. Mimo to opět, stejně jako u předešlého experimentu, je sledován i vliv normalizace a selekce příznaků.

Dimenze příznakových vektorů byly následující:

- Podmnožina datové sady obsahující 400 vzorků na třídu (po selekci): **79**
- Podmnožina datové sady obsahující 800 vzorků na třídu (po selekci): **135**
- Podmnožina datové sady obsahující 1200 vzorků na třídu (po selekci): **183**
- Celá datová sada (po selekci): **242**
- Původní počet dimenzí bez selekce: **1776**



Obrázek 6.1: Výsledky jednotlivých klasifikátorů při klasifikaci květin. Osa x udává průměrný počet vzorků na jednu třídu. Použité zkratky: **ANN** — umělé neuronové sítě, **CEEF** — Classification by evolutionary estimated functions, **DTC** — rozhodovací strom, **KNN** — k-nejbližších sousedů, **NBC** — Naivní Bayesův klasifikátor a **SVM** — Support Vector Machines



Obrázek 6.2: Doba trénování klasifikátoru *CEEF* při odstranění kontroly unikátnosti vybraných vzorků datové sady v chromozomu. Osa x udává průměrný počet vzorků na jednu třídu.

Na této datové sadě se žádnému testovanému klasifikátoru nepovedlo dostat nad hodnotu správnosti 0,8, jak je možné vidět v grafech na obrázku 6.3. Metody aplikované za účelem zlepšení výsledku (selekce příznaků a normalizace) měly největší úspěch u klasifikátorů *SVM* a *NBC*. Pro *SVM* měly tyto metody pozitivní dopad i na trénovací časy. U *CEEF* se zejména uplatnila selekce příznaků.

Pro umělé neuronové sítě se doporučuje volit normalizaci hodnot do intervalů <-1,1> či <0,1>. Zde požitá normalizace, která převádí hodnoty do intervalu <0,1>, neměla pozitivní vliv na výsledek. Mohlo to být způsobeno tím, že již původní data se nachází v intervalu <0,1> a použitá normalizace pouze způsobila nevhodné změny hodnot. Podobně jako například u klasifikace květin.

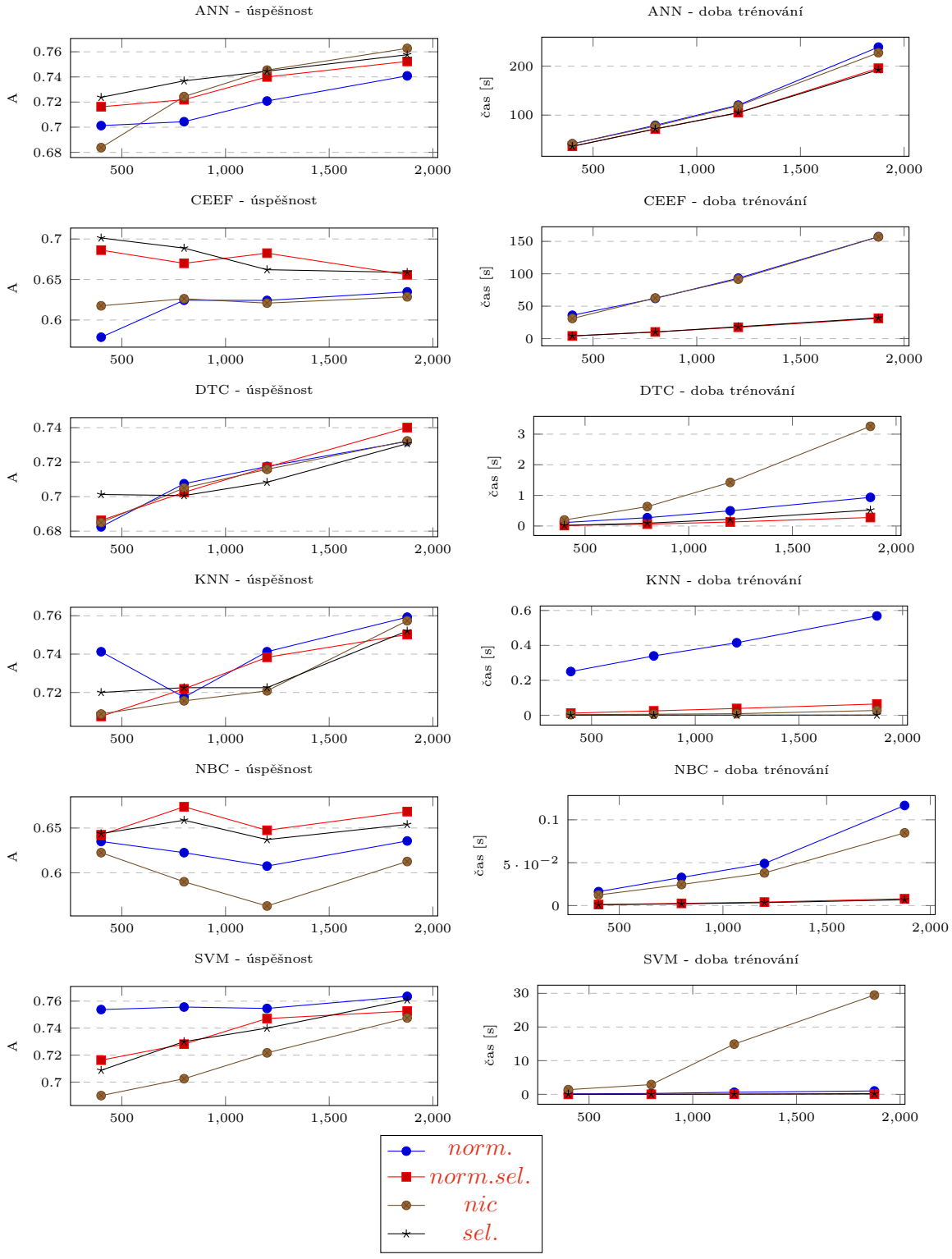
Nejlepší výsledky získané pomocí KNN, ANN a SVM nad plnou datovou sadu lze srovnávat s těmi lepšími, které jsou uváděné v [16]. V daném zdroji ty nejlepší výsledky, v době psaní této práce, dosahovaly okolo 0,81 správnosti. Nejlepší výsledky v této práci dosahují správnosti 0,76.

6.3 Rozpoznání objektu na obrázku

Pro tento experiment je použita sada obrázku *CIFAR-10* popsaná v sekci 5.4. Experiment zjišťuje, jak si klasifikátory poradí s touto datovou sadu nad *HOG* příznacích.

Tento experiment je odlišný od dvou předchozích hned v několika směrech. Nepoužívá křížovou validaci, jelikož datová sada definuje přímo rozdělení trénovací a testovací sady. Protože však experiment testuje některé klasifikátory, které nedávají pokaždé stejné výsledky i při stejných trénovacích/testovacích sadách (obsahují prvek náhody), tak tyto klasifikátory (*ANN*, *CEEF* a *DTC*) byly spuštěny 3krát a jednotlivé výsledky byly zprůměrovány. Z důvodu použití implementace extraktoru příznaků (*HOG*), který neumožňuje vypnutí normalizace, je sledován pouze vliv ne/použití selekce příznaků pro daný klasifikátor. Extrahované příznakové vektory měly dimenzi 1764, po selekci pak 687.

Výsledky shrnují grafy na obrázku 6.4. Nejlepších výsledků dosahovaly neuronové sítě. Nejhorších pak *CEEF*, které zde poprvé ukazují svůj problém se škálovatelností. Tento problém používá velké příznakové vektory, má desítky tisíc trénovacích vzorků a požaduje klasifikaci do 10 tříd. Což znamená, že *CEEF* se snaží vytvořit 10 funkcí a navíc fitness musí tyto funkce ohodnocovat pomocí mnoha velkých vektorů. Všechny tyto faktory zpomalují evoluci. Pro zrychlení běhu bylo parametrem nastaveno, aby se pro vyhodnocení fitness používala pouze desetina trénovací sady. Za účelem dosažení lepších výsledků by zřejmě bylo nutné zvětšit počet generací a další parametry.



Obrázek 6.3: Výsledky jednotlivých klasifikátorů při zjišťování biologické odezvy molekul. Osa x udává průměrný počet vzorků na jednu třídu. Použité zkratky: **ANN** — umělé neuronové sítě, **CEEF** — Classification by evolutionary estimated functions, **DTC** — rozhodovací strom, **KNN** — k-nejbližších sousedů, **NBC** — Naivní Bayesův klasifikátor a **SVM** — Support Vector Machines

Výsledky dosažené s těmito příznaky (*HOG*) jsou u některých klasifikátorů srovnatelné s těmi dohledatelnými v literatuře 0,46 [22] či 0,5 [3]. Pro neuronové sítě se výsledek nad těmito příznaky, v experimentech prováděných v této práci, pohybuje až okolo hodnoty správnosti 0,66. Takový výsledek se však nedá srovnat s výsledky, kterých dosahují hluboké neuronové sítě jako [27], tedy hodnoty správnosti až 0,99. Tento výsledek však nebyl dosažen na *HOG* příznacích.

6.4 SPAM filtr

V tomto experimentu se zaměřujeme, mimo samotné klasifikace nevyžádané pošty, na chování klasifikátorů při různých velikostech příznakových vektorů a různých použitých extraktorů pro jejich získání. Budeme používat hašování příznaků a *TF-IDF*. Nastavování velikosti u hašování příznaků je běžné. U *TF-IDF* si pomůžeme tím, že vybereme x příznaků s největší term frequency.

Výsledky experimentu shrnuje obrázek 6.5. Je vidět, že tentokrát jsou grafy vyneseny v závislosti na měnící se dimenzi příznakového vektoru. Lepším extraktorem, z hlediska úspěšnosti i času, bylo pro všechny klasifikátory bezesporu *TF-IDF*. Nejlépe si s určováním nevyžádané pošty poradily umělé neuronové sítě. Nejhůře si vedlo *CEEF*, u kterého lze vypozorovat výrazný propad při navýšení dimenzí příznakového vektoru, získaného pomocí *TF-IDF*, z 800 na 1200. Opět se zde objevil problém se škálovatelností, kdy velikost prohledávaného prostoru algoritmu uškodila. Přestože pro výpočet fitness byla použita pouze část trénovací množiny, tak trénovací časy *CEEF* byly mnohem větší než u jiných klasifikátorů.

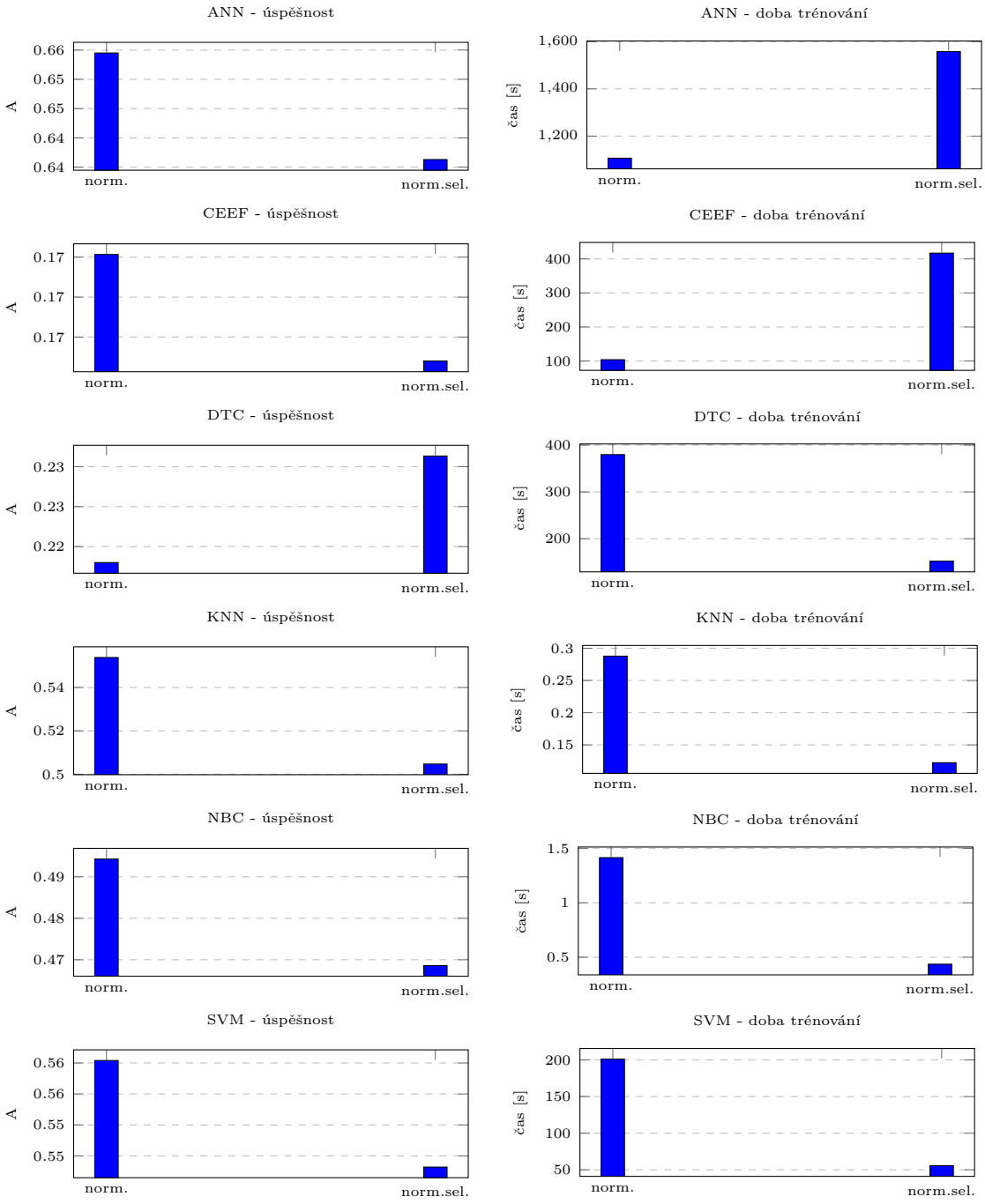
6.5 Určení jazyka textu

V tomto experimentu je použita podmnožina datové sady titulků, která byla popsána v sekci 5.1. Je vybráno pouze 32 jazyků a ke každému z nich je vybráno 2500 vzorků. Experimentování probíhá tak, že začneme s 8 jazyky, které chceme klasifikovat, a postupně navyšujeme počet jazyků o 8 až po finálních 32 jazyků ke klasifikaci. Jaké jazyky byly v jednotlivých podmnožinách vybrány lze nalézt v příloze B.

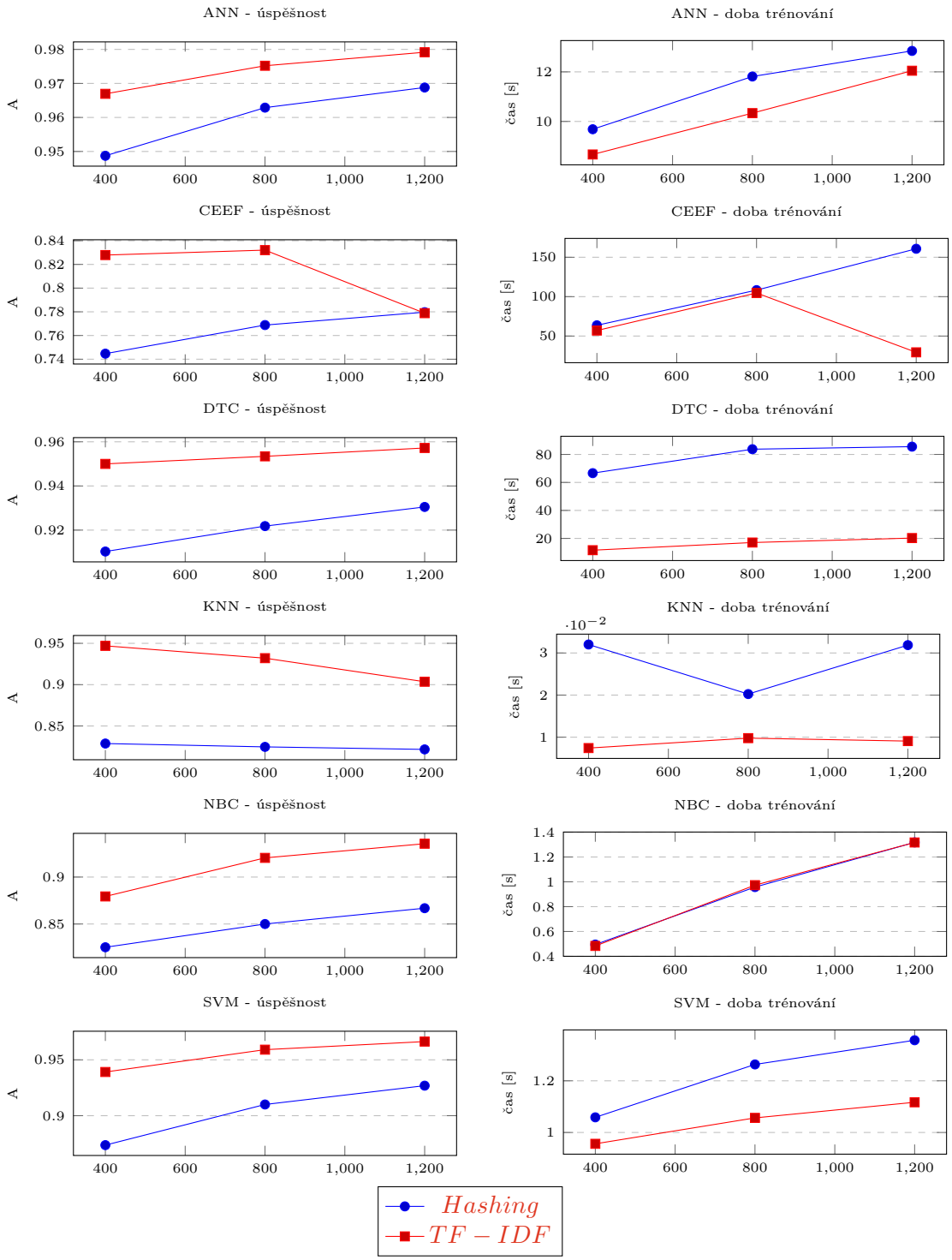
Experiment používá vždy stejnou velikost příznakového vektoru nastavenou na 40 dimenzí. Velikost je schválně nastavena na menší počet dimenzí, aby klasifikace byla těžší. Pro extrakci příznaků je použito hašování.

Výsledky experimentu jsou zachyceny v grafech na obrázku 6.6.

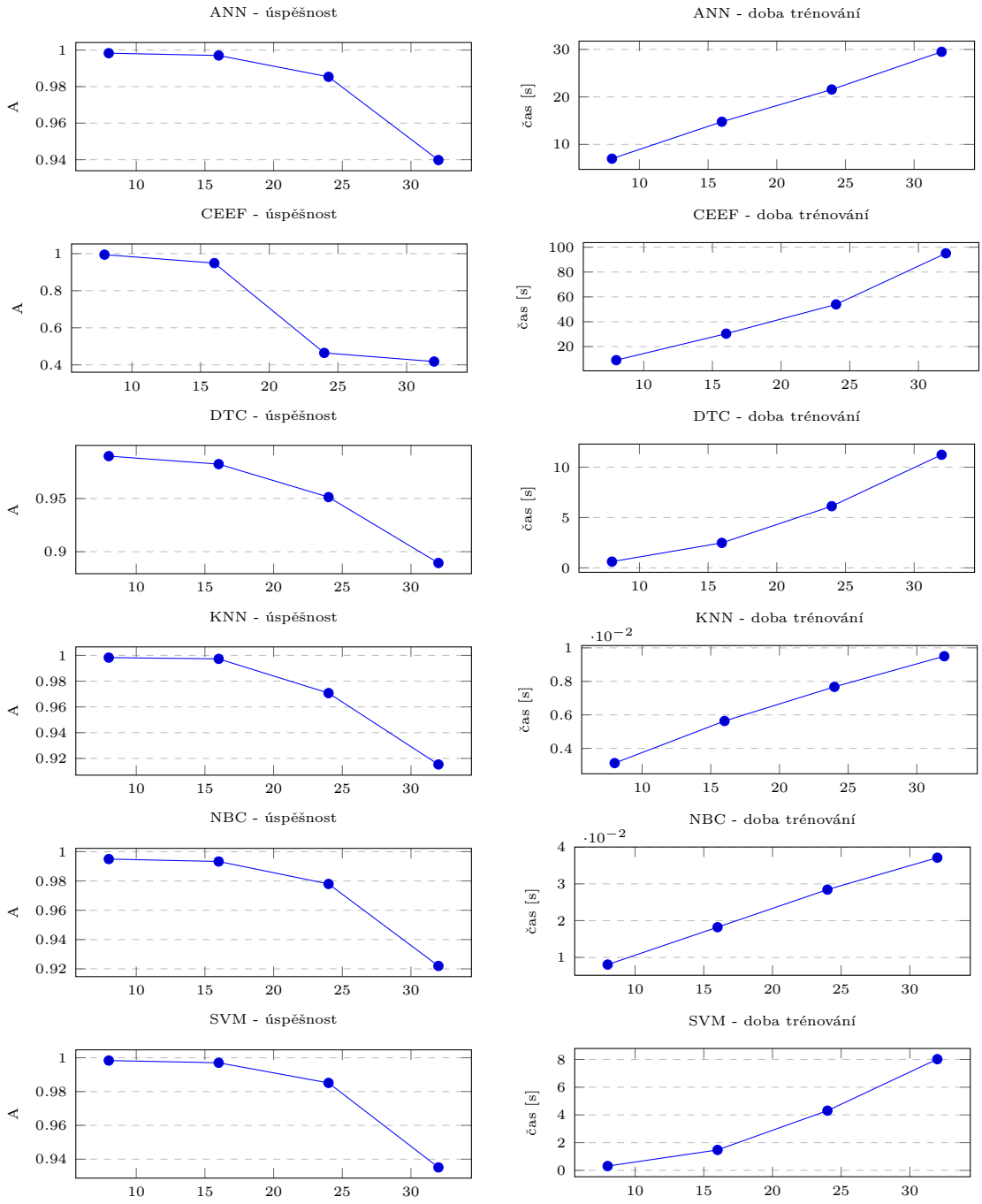
U všech klasifikátorů je zaznamenán pokles úspěšnosti s přibývajícím počtem tříd. Největší úpadek je u *CEEF*, což opět poukazuje na problém se škálovatelností. Všechny klasifikátory vykazují závislost trénovacího času na počtu tříd a s tím souvisejícím nárůstem vzorků. Nejmenší vliv na trénovací čas je zaznamenán u *KNN*. Vzhledem k faktu, že se jedná o metodu s líným učením, se nejedná o nic překvapivého.



Obrázek 6.4: Výsledky jednotlivých klasifikátorů při rozpoznávání objektu na obrázku. Použité zkratky: **ANN** — umělé neuronové sítě, **CEEF** — Classification by evolutionary estimated functions, **DTC** — rozhodovací strom, **KNN** — k-nejbližších sousedů, **NBC** — Naivní Bayesův klasifikátor a **SVM** — Support Vector Machines



Obrázek 6.5: Výsledky jednotlivých klasifikátorů při klasifikaci nevyžádané pošty. Osa x udává počet dimenzí příznakového vektoru. Použité zkratky: **ANN** — umělé neuronové sítě, **CEEF** — Classification by evolutionary estimated functions, **DTC** — rozhodovací strom, **KNN** — k-nejbližších sousedů, **NBC** — Naivní Bayesův klasifikátor a **SVM** — Support Vector Machines



Obrázek 6.6: Výsledky jednotlivých klasifikátorů při určování jazyka titulků k filmům. Osa x udává počet použitých jazyků. Použité zkratky: **ANN** — umělé neuronové sítě, **CEEF** — Classification by evolutionary estimated functions, **DTC** — rozhodovací strom, **KNN** — k-nejbližších sousedů, **NBC** — Naivní Bayesův klasifikátor a **SVM** — Support Vector Machines

Kapitola 7

Závěr

V této práci bylo uvedeno několik metod pro klasifikaci, které byly založeny na strojovém učení a jsou v dnešních dnech běžně užívané. Je zde popsáno, jak tyto metody vyhodnocovat za účelem jejich porovnání a obecně kvality výsledků, kterých dosahují. Mimo metod samotných bylo popsáno i obecně strojové učení jako takové.

Je zde uveden i popis metod pro extrakci příznaků z textu a obrázků. Tyto metody jsou nezbytné pro předzpracování surových dat pro klasifikátor.

Dále byl proveden návrh aplikace pro porovnání klasifikátorů. Aplikace byla implementována a zdokumentována. Jako implementační jazyk byl zvolen python. Implementace většiny klasifikátorů (i extraktorů příznaků a dalších) jsou získány z externích knihoven jako jsou scikit-learn a Keras s TensorFlow.

V rámci práce byl implementován a navržen klasifikátor založený na principu evolučních algoritmů, který byl společně s dalšími klasifikátory otestován na několika datových sadách. Experimenty ukázaly, že je schopen se učit a podávat dobré výsledky. Slabina klasifikátoru je ovšem škálovatelnost a pro těžší problémy (více tříd, více dimenzí příznakového vektoru atd.) je pravděpodobně nutné používat velký počet generací, větší populace či navyšovat jiné parametry, aby byla dosažena přesnost srovnatelná s ostatními algoritmy. Tento klasifikátor by bylo zřejmě výhodné paralelizovat, obdobně jako je výhodné paralelizovat neuronové sítě. Mimo to by bylo vhodné vyzkoušet i jiné metody pro tvorbu funkcí pocházejících vybranými body, než jsou uvedené v 4.5.2, a zvážit vynechání kontroly na unikátnost vzorků v rámci dané části chromozomu, tak jak bylo provedeno v experimentu 6.1.

Napříč experimenty výchazely většinou nejlépe neuronové sítě. Jejich trénovací časy byly sice oproti jiným klasifikátorům větší, ale za to dosahovaly lepších výsledků. Neuronové sítě byly pro větší hodnoty parametru *batch size* spouštěny na grafické kartě. Pro malé hodnoty, jako je například 1, došlo s použitím grafické karty naopak ke zpomalení. Pravděpodobně vlivem režie při přenosu dat na grafickou kartu.

Pro standardně používané datové sady bylo provedeno porovnání dosažených výsledků s výsledky z literatury.

V některých experimentech byly zkoumány vlivy metod pro zlepšení klasifikace. Konkrétně se jednalo o normalizaci a selekci příznaků. V rámci experimentů bylo i poukázáno na možná úskalí, kdy není jejich použití vhodné.

Literatura

[1] *17.2. multiprocessing — Process-based parallelism*. [Online; navštíveno 06.05.2019].
URL https://docs.python.org/3.6/library/multiprocessing.html

[2] *Bayesova věta*. [Online; navštíveno 28.04.2019].
URL https://www.wikiskripta.eu/w/Bayesova_v%C4%9Bta

[3] *cifar10-classification*. [Online; navštíveno 14.05.2019].
URL https://github.com/mpienkosz/cifar10-classification/wiki

[4] *Creating and discovering plugins*. [Online; navštíveno 06.05.2019].
URL
https://packaging.python.org/guides/creating-and-discovering-plugins/

[5] *Feature selection - scikit-learn*. [Online; navštíveno 28.04.2019].
URL https://scikit-learn.org/stable/modules/feature_selection.html

[6] *fenotype definition*. [Online; navštíveno 30.04.2019].
URL https://www.nature.com/scitable/definition/phenotype-phenotypes-35

[7] *genotype definition*. [Online; navštíveno 30.04.2019].
URL https://www.nature.com/scitable/definition/genotype-234

[8] *GlobalInterpreterLock*. [Online; navštíveno 06.05.2019].
URL https://wiki.python.org/moin/GlobalInterpreterLock

[9] *Keras*. [Online; navštíveno 16.01.2019].
URL https://keras.io/

[10] *Python, programovací jazyk*. [Online; navštíveno 16.01.2019].
URL https://python.cz/

[11] *Qt for Python*. [Online; navštíveno 16.01.2019].
URL https://wiki.qt.io/Qt_for_Python

[12] *QThread*. [Online; navštíveno 06.05.2019].
URL https://doc.qt.io/qtforpython/PySide2/QtCore/QThread.html

[13] *scikit-image: Image processing in Python — scikit-image*. [Online; navštíveno 07.05.2019].
URL https://scikit-image.org/

[14] *Sphinx*. [Online; navštíveno 07.05.2019].
URL http://www.sphinx-doc.org/en/stable/

[15] *Steady State Genetic Algorithm*. [Online; navštíveno 01.05.2019].
URL https://www.cs.unm.edu/~neal.holts/dga/optimizationAlgorithms/steadyStateGA.html

[16] *Supervised Classification on Bioresponse*. [Online; navštíveno 14.05.2019].
URL https://www.openml.org/t/14966

[17] *Supervised Classification on iris*. [Online; navštíveno 14.05.2019].
URL https://www.openml.org/t/59

[18] *TensorFlow*. [Online; navštíveno 16.01.2019].
URL https://www.tensorflow.org/

[19] *What is the Python Global Interpreter Lock (GIL)?* [Online; navštíveno 06.05.2019].
URL https://realpython.com/python-gil/

[20] Abney, S.: *Semisupervised Learning for Computational Linguistics*. CRC Press, 2007, ISBN 9781420010800.

[21] Brabazon, A.; O'Neill, M.; McGarraghy, S.: *Natural Computing Algorithms*. Springer, 2015, ISBN 978-3-662-43631-8.

[22] Cheng, L.; Leung, A.; Ozawa, S.: *Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13–16, 2018, Proceedings*. číslo díl 6 in Lecture Notes in Computer Science, Springer International Publishing, 2018, ISBN 9783030042240.
URL https://books.google.cz/books?id=IR19DwAAQBAJ

[23] CHOLLET, F.: *Deep Learning with Python*. Manning Publications Co., 2018, ISBN 9781617294433.

[24] Dalal, N.; Triggs, B.: Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, ročník 1, June 2005, ISSN 1063-6919, s. 886–893 vol. 1, doi:10.1109/CVPR.2005.177.

[25] Dočekal, M.: *Porovnání klasifikačních metod*, 2019, semestrální projekt. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Ivana Burgetová, Ph.D.

[26] Eiben, A.; Smith, J.: *Introduction to Evolutionary Computing*. Springer, 2015, ISBN 978-3-662-44874-8.

[27] Huang, Y.; Cheng, Y.; Chen, D.; aj.: GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. 2018, arXiv:1811.06965.

[28] Jiawei Han, J. P., Micheline Kamber: *Data Mining Concepts and Techniques Third Edition*. Elsevier Inc., 2012, ISBN 978-0-12-381479-1.

[29] Joshi, P.: *Artificial Intelligence with Python*. Packt Publishing Ltd., 2017, ISBN 978-1-78646-439-2.

[30] Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images. 2009.
URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[31] Lison, P.; Tiedemann, J.: OpenSubtitles2016: Extracting Large Parallel Corpora from Movie and TV Subtitles. 2016.
URL http://stp.lingfil.uu.se/~joerg/paper/opensubs2016.pdf

[32] Louppe, G.; Wehenkel, L.; Sutera, A.; aj.: Understanding variable importances in forests of randomized trees. In *Advances in Neural Information Processing Systems 26*, editace C. J. C. Burges; L. Bottou; M. Welling; Z. Ghahramani; K. Q. Weinberger, Curran Associates, Inc., 2013, s. 431–439.
URL http://papers.nips.cc/paper/4928-understanding-variable-importances-in-forests-of-randomized-trees.pdf

[33] Mallick, S.: *Histogram of Oriented Gradients*. [Online; navštíveno 04.01.2019].
URL https://www.learnopencv.com/histogram-of-oriented-gradients/

[34] Manning, C. D.; Raghavan, P.; Schütze, H.: *Introduction to information retrieval*. Cambridge University Press, 2008, ISBN 0521865719.

[35] Mitchell, T. M.: *Machine Learning*. Manning Publications Co., 1997, ISBN 0070428077.

[36] Murat Aslan, N. A. B.: *Feature Extraction Definition | DeepAI*. [Online; navštíveno 31.12.2018].
URL https://deepai.org/machine-learning-glossary-and-terms/feature-extraction

[37] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; aj.: Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, ročník 12, 2011: s. 2825–2830.

[38] Russell, S. J.; Norvig, P.: *Artificial Intelligence A Modern Approach Third Edition*. Pearson Education, Inc., 2010, ISBN 978-0-13-604259-4.

[39] Schwab, I.; Link, N.: Gaining Insights from Symbolic Regression Representations of Class Boundaries.

[40] Shafranovich, Y.: Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC 4180, RFC Editor, October 2005, http://www.rfc-editor.org/rfc/rfc4180.txt.
URL http://www.rfc-editor.org/rfc/rfc4180.txt

[41] Simon, D.: *Evolutionary Optimization Algorithms*. Wiley, 2013, ISBN 9781118659502.
URL https://books.google.cz/books?id=gwUwIEPqk30C

[42] Stańczyk, U.; Jain, L. C.: *Feature Selection for Data and Pattern Recognition*. Springer, 2015, ISBN 978-3-662-45620-0.

[43] Tellez, A.; Pumperla, M.; Malohlava, M.: *Mastering Machine Learning with Spark 2.x*. Packt Publishing Ltd., 2017, ISBN 978-1-78398-028-4.

[44] Weinberger, K. Q.; Dasgupta, A.; Attenberg, J.; aj.: Feature Hashing for Large Scale Multitask Learning. *CoRR*, ročník abs/0902.2206, 2009, 0902.2206.
URL http://arxiv.org/abs/0902.2206

[45] Weise, T.: *Global Optimization Algorithms – Theory and Application*. 2009.

Příloha A

Popis atributů instalovatelných zásuvných modulů

Jako součást aplikace je poskytnuto několik samostatně instalovatelných zásuvných modulů jejichž atributy jsou zde uvedeny a popsány.

A.1 Extraktory příznaků

- **Pass**
  bez atributů
- **Hashing**

  Atributy:

  - non negative values: Vypnutí/zapnutí generování negativních hodnot.
  - number of feature: Počet dimenzí příznakového vektoru.
  - normalization: Výběr z několika druhů normalizací. Možné je i normalizaci vypnout.
- **Histogram of Oriented Gradients**

  Atributy:

  - number of orientation bins: Počet košů.
  - width of a cell: Šířka buňky v pixelech.
  - height of a cell: Výška buňky v pixelech.
  - number of cells in each block (horizontal): Počet buněk v jednom bloku (horizontálně).
  - number of cells in each block (vertical): Počet buněk v jednom bloku (vertikálně).
  - normalization: Výběr z několika druhů normalizací.
- **Term Frequency–Inverse Document Frequency**

  Atributy:

  - max number of features: Maximální velikost příznakového vektoru.

- case sensitive: Zapnutí/Vypnutí rozeznávání mezi malými a velkými písmeny. Pokud je zapnuto, pak jsou například slova auto a Auto brána jako totožná.
- normalization: Výběr z několika druhů normalizací. Možné je i normalizaci vypnout.

A.2 Klasifikátory

- **Support Vector Machines**

  Atributy:

  - normalize: Umožňuje vybrat normalizaci dat, pouze pro konkrétní jeden klasifikátor. Možné je i normalizaci vypnout.
  - kernel: Umožňuje výběr z více možných druhů funkcí mapujících původní prostor do více dimenzionálního, které umožňují klasifikovat i lineárně neseparovatelné problémy.

- **Classification by evolutionary estimated functions**

  Atributy:

  - normalize: Umožňuje vybrat normalizaci dat, pouze pro konkrétní jeden klasifikátor. Možné je i normalizaci vypnout.
  - number of generations: Počet generací evolučního algoritmu.
  - stop accuracy: Hodnota fitness při niž se evoluce ukončí.
  - population size: Počet jedinců v populaci.
  - selection method: Výběr metody pro výběr rodičů ke křížení.
  - random seed: Nastavení semínka náhodného generátoru. Vhodné pro zajištění stejného výsledku i při vícenásobném spuštění.
  - max crossovers in generation: Maximální počet křížení v jedné generaci. Validní hodnoty jsou kladná celá čísla.
  - max mutations: Maximální počet mutací provedených při mutování jedince.
  - max start slots: Maximální počet slotů pro uchovávání vzorových bodů v daném prostoru pro jednu třídu, při prvotním vytvoření jedince. Minimum jsou dva sloty.
  - crossover probability: Pravděpodobnost provedení křížení.
  - test set size: Udává část trénovacích dat, která bude použita pro fitness funkci (testování fenotypu). Validní hodnoty jsou čísla v intervalu (0,1>.
  - change test set for each generation: Zapne náhodnou změnu množiny dat, použité pro fitness funkci, v každé generaci. Dává smysl pokud nebereme celou trénovací množinu (test set size není 1).
  - Log generation fitness: Zapne/Vypne logování aktuální nejlepší hodnoty fitness.

- **k-Nearest Neighbor**

  Atributy:

  - normalize: Umožňuje vybrat normalizaci dat, pouze pro konkrétní jeden klasifikátor. Možné je i normalizaci vypnout.
  - neighbors: Udává kolik nejbližších sousedů bude bráno v potaz.

- **Decision Tree Classifier**

  Atributy:

  - normalize: Umožňuje vybrat normalizaci dat, pouze pro konkrétní jeden klasifikátor. Možné je i normalizaci vypnout.
  - random seed: Nastavení semínka náhodného generátoru. Vhodné pro zajištění stejného výsledku i při vícenásobném spuštění.
  - split criterion: Volba výběrové metriky pro volbu nejlepšího rozdělovacího atributu.

- **Naive Bayes Classifier**

  Atributy:

  - normalize: Umožňuje vybrat normalizaci dat, pouze pro konkrétní jeden klasifikátor. Možné je i normalizaci vypnout.
  - type: Výběr jakým způsobem bude počítat $P(x_k|c_i)$. Jedná se o výběr jedné z implementací MultinomialNB (kladná celá čísla) nebo GaussianNB (pro spojité hodnoty).

- **Artificial Neural Networks**

  Atributy:

  - normalize: Umožňuje vybrat normalizaci dat, pouze pro konkrétní jeden klasifikátor. Možné je i normalizaci vypnout.
  - random seed: Nastavení semínka náhodného generátoru. Vhodné pro zajištění stejného výsledku i při vícenásobném spuštění.
  - epochs: Počet trénovacích epoch.
  - batch size: Počet vzorků, po kterých dojde k aktualizaci vah (počet vzorků použitých pro jeden průchod).
  - learning rate: Určuje velikost posuvu dle gradientu.
  - GPU: Vypnutí/Zapnutí použití grafické karty.
  - output layer activation function: Jakou aktivační funkci má použít výstupní vrstva.
  - log epoch: Vypnutí/Zapnutí logování aktuálních informací o epochách (aktuální úspěšnost).
  - hidden layers: Nastavení počtu skrytých vrstev plně propojené dopředné sítě, počtu neuronů v jednotlivých skrytých vrstvách a použitých aktivačních funkcí.

Příloha B

Tabulky výsledků experimentů

Vysvětlivky ke zkratkám:

- **SVM**: Označuje klasifikátor Support Vector Machines.
- **CEEF**: Označuje klasifikátor používající evolučně odhadnuté funkce (Classification by evolutionary estimated functions).
- **KNN**: Označuje klasifikátor k-nejbližších sousedů.
- **DTC**: Označuje klasifikátor používající rozhodovací strom.
- **NBC**: Označuje Naivní Bayesův klasifikátor.
- **ANN**: Označuje klasifikátor používající neuronovou síť.
- **A**: Označuje průměrnou správnost (accuracy).
- **F1**: Jedná se o průměr z váženého průměru F1 mír počítaných zvlášť pro každou třídu. Váha je dána počtem vzorků ve třídě.
- **P**: Jedná se o průměr z váženého průměru přesností (precision) počítaných zvlášť pro každou třídu. Váha je dána počtem vzorků ve třídě.
- **R**: Jedná se o průměr z váženého průměru úplností (recall) počítaných zvlášť pro každou třídu. Váha je dána počtem vzorků ve třídě.
- ***-*** (např. nic-13)

  Část za pomlčkou udává průměrný počet použitých vzorků na jednu třídu či počet dimenzí příznakového vektoru. Část před pomlčkou může znamenat jedno z následujících:

  - **nic**: Na data nebyla použita ani normalizace a ani selekce.
  - **norm.**: Na data byla použita pouze normalizace.
  - **sel.**: Na data byla použita pouze selekce.
  - **norm.sel.**: Na data byla použita normalizace a selekce.
  - **TF-IDF**: Pro extrakci příznaků byla použita metoda TF-IDF.
  - **Hashing**: Pro extrakci příznaků byla použita metoda hašování příznaků.

B.1 Klasifikace květin

Parametry jednotlivých klasifikátorů:

- **SVM**
  - Normalize=None | Normalizer:Norm->l2
  - Kernel=rbf
- **CEEF**
  - Normalize=None | Normalizer:Norm->l2
  - Number of generations=100
  - Stop accuracy=None
  - Population size=20
  - Selection method=RANK
  - Random seed=None
  - Max crossovers in generation=1
  - Max mutations=5
  - Max start slots=2
  - Crossover probability=0.75
  - Test set size=1.0
  - Change test set for each generation=False
  - Log generation fitness=True
- **KNN**
  - Normalize=None | Normalizer:Norm->l2
  - Neighbors=3
- **DTC**
  - Normalize=None | Normalizer:Norm->l2
  - Random seed=None
  - Split criterion=Gini Impurity
- **NBC**
  - Normalize=None | Normalizer:Norm->l2
  - Type=gaussian
- **ANN**
  - Normalize=None | MinMaxScaler:Min->-1, Max->1
  - Random seed=None
  - Epochs=100
  - Batch size=1

- Learning rate=0.001
- GPU=False
- Output layer activation function=softmax
- Log epoch=True
- Hidden layers=[LayerNeurons=4, Activation function=relu]

klasifikátor	**data**	**A**	**F1**	**P**	**R**	**trénování[s]**	**testování[s]**
SVM	nic-13	0.97	0.96	0.95	0.97	0.00129	0.00012
CEEF	nic-13	0.97	0.96	0.95	0.97	0.25073	0.00065
KNN	nic-13	0.97	0.96	0.95	0.97	0.00038	0.00087
DTC	nic-13	0.97	0.96	0.95	0.97	0.00048	0.00018
NBC	nic-13	0.97	0.96	0.95	0.97	0.00061	0.00022
ANN	nic-13	0.7	0.62	0.6	0.7	4.18564	0.09154
SVM	norm.-13	0.57	0.48	0.47	0.57	0.00146	0.00025
CEEF	norm.-13	0.53	0.42	0.39	0.53	0.45145	0.00072
KNN	norm.-13	0.67	0.59	0.57	0.67	0.0005	0.00091
DTC	norm.-13	0.63	0.54	0.52	0.63	0.00062	0.00032
NBC	norm.-13	0.62	0.53	0.52	0.62	0.00071	0.00041
ANN	norm.-13	0.97	0.96	0.95	0.97	3.23815	0.10229
SVM	nic-25	0.97	0.97	0.98	0.97	0.00153	0.00014
CEEF	nic-25	0.98	0.98	0.99	0.98	0.2676	0.00062
KNN	nic-25	0.94	0.94	0.96	0.94	0.00042	0.00102
DTC	nic-25	0.94	0.94	0.95	0.94	0.00058	0.00019
NBC	nic-25	0.96	0.96	0.96	0.96	0.00059	0.00019
ANN	nic-25	0.79	0.74	0.74	0.79	7.77442	0.08947
SVM	norm.-25	0.98	0.98	0.99	0.98	0.00168	0.00031
CEEF	norm.-25	0.93	0.93	0.96	0.93	0.26514	0.00076
KNN	norm.-25	1.0	1.0	1.0	1.0	0.00051	0.00111
DTC	norm.-25	0.91	0.91	0.94	0.91	0.00071	0.00031
NBC	norm.-25	0.98	0.98	0.99	0.98	0.00075	0.00037
ANN	norm.-25	0.96	0.96	0.96	0.96	6.04717	0.09827
SVM	nic-38	0.96	0.96	0.97	0.96	0.00166	0.00015
CEEF	nic-38	0.95	0.95	0.96	0.95	0.28393	0.00065
KNN	nic-38	0.95	0.95	0.96	0.95	0.00038	0.00114
DTC	nic-38	0.94	0.94	0.95	0.94	0.00061	0.00019
NBC	nic-38	0.94	0.93	0.95	0.94	0.00064	0.0002
ANN	nic-38	0.94	0.94	0.95	0.94	11.15485	0.08916
SVM	norm.-38	0.95	0.95	0.97	0.95	0.00183	0.0003
CEEF	norm.-38	0.96	0.96	0.97	0.96	0.28248	0.00076
KNN	norm.-38	0.97	0.97	0.98	0.97	0.00049	0.00112
DTC	norm.-38	0.93	0.92	0.95	0.93	0.00074	0.00033
NBC	norm.-38	0.97	0.97	0.98	0.97	0.00073	0.00032
ANN	norm.-38	0.95	0.95	0.96	0.95	8.39582	0.09849
SVM	nic-50	0.98	0.98	0.98	0.98	0.00183	0.00016
CEEF	nic-50	0.94	0.94	0.95	0.94	0.29532	0.00072

KNN	nic-50	0.97	0.97	0.97	0.97	0.00047	0.00119
DTC	nic-50	0.95	0.95	0.96	0.95	0.00068	0.0002
NBC	nic-50	0.95	0.95	0.96	0.95	0.00074	0.00018
ANN	nic-50	0.89	0.87	0.87	0.89	15.21784	0.0898
SVM	norm.-50	0.97	0.97	0.98	0.97	0.00207	0.00035
$CEEF$	norm.-50	0.96	0.96	0.96	0.96	0.30259	0.00084
KNN	norm.-50	0.97	0.97	0.98	0.97	0.00059	0.00118
DTC	norm.-50	0.96	0.96	0.96	0.96	0.00084	0.00032
NBC	norm.-50	0.97	0.97	0.98	0.97	0.0008	0.00036
ANN	norm.-50	0.96	0.96	0.96	0.96	11.60181	0.0987
SVM	sel.-13	1.0	1.0	1.0	1.0	0.00144	0.00015
$CEEF$	sel.-13	1.0	1.0	1.0	1.0	0.19148	0.00056
KNN	sel.-13	1.0	1.0	1.0	1.0	0.00037	0.0009
DTC	sel.-13	1.0	1.0	1.0	1.0	0.00058	0.00018
NBC	sel.-13	1.0	1.0	1.0	1.0	0.00056	0.00015
ANN	sel.-13	0.83	0.78	0.75	0.83	4.25508	0.08291
SVM	norm.sel.-13	0.55	0.44	0.43	0.55	0.00144	0.00028
$CEEF$	norm.sel.-13	0.53	0.42	0.39	0.53	0.2185	0.00061
KNN	norm.sel.-13	0.57	0.46	0.43	0.57	0.00052	0.00096
DTC	norm.sel.-13	0.53	0.41	0.38	0.53	0.00071	0.00028
NBC	norm.sel.-13	0.55	0.44	0.43	0.55	0.00075	0.00041
ANN	norm.sel.-13	1.0	1.0	1.0	1.0	3.43482	0.09036
SVM	sel.-25	0.94	0.94	0.95	0.94	0.00137	0.00018
$CEEF$	sel.-25	0.93	0.93	0.95	0.93	0.26681	0.00065
KNN	sel.-25	0.94	0.94	0.95	0.94	0.00039	0.00103
DTC	sel.-25	0.93	0.93	0.94	0.93	0.00055	0.00021
NBC	sel.-25	0.94	0.94	0.95	0.94	0.00058	0.00019
ANN	sel.-25	0.75	0.7	0.72	0.75	7.53059	0.09551
SVM	norm.sel.-25	0.55	0.46	0.47	0.55	0.00159	0.00032
$CEEF$	norm.sel.-25	0.54	0.45	0.46	0.54	0.81269	0.00074
KNN	norm.sel.-25	0.5	0.41	0.41	0.5	0.00053	0.00116
DTC	norm.sel.-25	0.59	0.51	0.51	0.59	0.00069	0.00032
NBC	norm.sel.-25	0.52	0.43	0.44	0.52	0.00066	0.00042
ANN	norm.sel.-25	0.94	0.94	0.95	0.94	5.63642	0.1035
SVM	sel.-38	0.94	0.93	0.95	0.94	0.00139	0.00018
$CEEF$	sel.-38	0.94	0.93	0.95	0.94	0.27813	0.00068
KNN	sel.-38	0.93	0.92	0.94	0.93	0.00041	0.00108
DTC	sel.-38	0.92	0.92	0.94	0.92	0.00057	0.00019
NBC	sel.-38	0.94	0.93	0.95	0.94	0.00052	0.00021
ANN	sel.-38	0.83	0.81	0.83	0.83	11.05699	0.08851
SVM	norm.sel.-38	0.59	0.51	0.51	0.59	0.00184	0.00031
$CEEF$	norm.sel.-38	0.6	0.52	0.52	0.6	0.9414	0.00074
KNN	norm.sel.-38	0.59	0.51	0.53	0.59	0.00053	0.00116
DTC	norm.sel.-38	0.62	0.54	0.56	0.62	0.0007	0.00032
NBC	norm.sel.-38	0.57	0.49	0.49	0.57	0.00071	0.00041

ANN	norm.sel.-38	0.94	0.93	0.95	0.94	8.39407	0.09729
SVM	sel.-50	0.97	0.97	0.97	0.97	0.00164	0.00015
CEEF	sel.-50	0.95	0.95	0.96	0.95	0.29522	0.0007
KNN	sel.-50	0.96	0.96	0.97	0.96	0.00043	0.00126
DTC	sel.-50	0.95	0.95	0.95	0.95	0.00058	0.00019
NBC	sel.-50	0.97	0.97	0.97	0.97	0.00057	0.00019
ANN	sel.-50	0.95	0.95	0.96	0.95	14.57642	0.0851
SVM	norm.sel.-50	0.52	0.42	0.4	0.52	0.00184	0.0003
CEEF	norm.sel.-50	0.53	0.43	0.41	0.53	1.57155	0.00061
KNN	norm.sel.-50	0.49	0.39	0.37	0.49	0.00049	0.0012
DTC	norm.sel.-50	0.51	0.41	0.39	0.51	0.0007	0.00036
NBC	norm.sel.-50	0.54	0.44	0.41	0.54	0.00072	0.00037
ANN	norm.sel.-50	0.97	0.97	0.97	0.97	10.99785	0.09303

Tabulka B.1: Výsledky jednotlivých klasifikátorů při klasifikace květin.

B.2 Zjištění biologické odezvy molekuly

Parametry jednotlivých klasifikátorů:

- **SVM**
  - Normalize=None | Normalizer:Norm->l2
  - Kernel=linear
- **CEEF**
  - Normalize=None | Normalizer:Norm->l2
  - Number of generations=500
  - Stop accuracy=None
  - Population size=50
  - Selection method=RANK
  - Random seed=None
  - Max crossovers in generation=1
  - Max mutations=20
  - Max start slots=2
  - Crossover probability=0.75
  - Test set size=1.0
  - Change test set for each generation=False
  - Log generation fitness=True
- **KNN**
  - Normalize=None | RobustScaler:centering->True
  - Neighbors=3

- **DTC**
  - Normalize=None | StandardScaler:centering->True
  - Random seed=None
  - Split criterion=Gini Impurity
- **NBC**
  - Normalize=None | Normalizer:Norm->l2
  - Type=gaussian
- **ANN**
  - Normalize=None | Normalizer:Norm->l2
  - Random seed=None
  - Epochs=300
  - Batch size=16
  - Learning rate=0.001
  - GPU=True
  - Output layer activation function=softmax
  - Log epoch=True
  - Hidden layers=[LayerNeurons=100, Activation function=sigmoid]

klasifikátor	**data**	**A**	**F1**	**P**	**R**	**trénování**[s]	**testování**[s]
SVM	nic-1876	0.75	0.75	0.75	0.75	29.48807	0.00129
$CEEF$	nic-1876	0.63	0.63	0.63	0.63	157.05186	0.02404
KNN	nic-1876	0.76	0.76	0.76	0.76	0.02739	3.52228
DTC	nic-1876	0.73	0.73	0.73	0.73	3.25586	0.00203
NBC	nic-1876	0.61	0.56	0.65	0.61	0.08476	0.005
ANN	nic-1876	0.76	0.76	0.76	0.76	227.22438	0.10854
SVM	norm.-1876	0.76	0.76	0.76	0.76	1.0137	0.00387
$CEEF$	norm.-1876	0.63	0.63	0.64	0.63	157.22364	0.02548
KNN	norm.-1876	0.76	0.76	0.76	0.76	0.56877	2.5735
DTC	norm.-1876	0.73	0.73	0.73	0.73	0.93493	0.00307
NBC	norm.-1876	0.64	0.6	0.66	0.64	0.11666	0.00713
ANN	norm.-1876	0.74	0.74	0.74	0.74	238.74556	0.11822
SVM	nic-800	0.7	0.7	0.7	0.7	2.90042	0.0007
$CEEF$	nic-800	0.63	0.62	0.63	0.63	62.66633	0.00808
KNN	nic-800	0.72	0.72	0.72	0.72	0.00575	0.67245
DTC	nic-800	0.71	0.7	0.71	0.71	0.63401	0.00119
NBC	nic-800	0.59	0.55	0.64	0.59	0.02457	0.00242
ANN	nic-800	0.72	0.72	0.72	0.72	77.55879	0.09756
SVM	norm.-800	0.76	0.76	0.76	0.76	0.25653	0.00202
$CEEF$	norm.-800	0.62	0.62	0.63	0.62	62.00415	0.00877
KNN	norm.-800	0.72	0.72	0.72	0.72	0.33965	0.46862

DTC	norm.-800	0.71	0.71	0.71	0.71	0.27191	0.00155
NBC	norm.-800	0.62	0.6	0.66	0.62	0.03284	0.00344
ANN	norm.-800	0.7	0.7	0.71	0.7	79.14158	0.1097
SVM	nic-1200	0.72	0.72	0.72	0.72	14.92305	0.00091
$CEEF$	nic-1200	0.62	0.62	0.63	0.62	91.65218	0.01109
KNN	nic-1200	0.72	0.72	0.72	0.72	0.00889	1.4775
DTC	nic-1200	0.72	0.72	0.72	0.72	1.42333	0.00152
NBC	nic-1200	0.56	0.51	0.61	0.56	0.0381	0.00333
ANN	nic-1200	0.75	0.75	0.75	0.75	118.75872	0.09861
SVM	norm.-1200	0.75	0.75	0.76	0.75	0.61808	0.0027
$CEEF$	norm.-1200	0.62	0.62	0.63	0.62	93.19398	0.01299
KNN	norm.-1200	0.74	0.74	0.74	0.74	0.4151	1.14478
DTC	norm.-1200	0.72	0.72	0.72	0.72	0.49688	0.0022
NBC	norm.-1200	0.61	0.57	0.65	0.61	0.04902	0.00492
ANN	norm.-1200	0.72	0.72	0.72	0.72	120.32169	0.10979
SVM	nic-400	0.69	0.69	0.69	0.69	1.38952	0.00053
$CEEF$	nic-400	0.62	0.61	0.63	0.62	30.999	0.00438
KNN	nic-400	0.71	0.71	0.71	0.71	0.00341	0.18197
DTC	nic-400	0.68	0.68	0.69	0.68	0.19574	0.00088
NBC	nic-400	0.62	0.61	0.64	0.62	0.01242	0.00146
ANN	nic-400	0.68	0.68	0.68	0.68	41.41883	0.10062
SVM	norm.-400	0.75	0.75	0.75	0.75	0.13251	0.00117
$CEEF$	norm.-400	0.58	0.58	0.58	0.58	35.97321	0.00492
KNN	norm.-400	0.74	0.74	0.74	0.74	0.25021	0.14602
DTC	norm.-400	0.68	0.68	0.68	0.68	0.11327	0.00094
NBC	norm.-400	0.64	0.63	0.64	0.64	0.01626	0.00196
ANN	norm.-400	0.7	0.7	0.7	0.7	41.40033	0.10981
SVM	sel.-1876	0.76	0.76	0.76	0.76	0.1963	0.00038
$CEEF$	sel.-1876	0.66	0.66	0.66	0.66	32.04572	0.00451
KNN	sel.-1876	0.75	0.75	0.75	0.75	0.00115	0.1886
DTC	sel.-1876	0.73	0.73	0.73	0.73	0.52233	0.00075
NBC	sel.-1876	0.65	0.64	0.66	0.65	0.00677	0.00084
ANN	sel.-1876	0.76	0.76	0.76	0.76	192.42829	0.10114
SVM	norm.sel.-1876	0.75	0.75	0.75	0.75	0.06671	0.00067
$CEEF$	norm.sel.-1876	0.66	0.65	0.66	0.66	31.14468	0.00484
KNN	norm.sel.-1876	0.75	0.75	0.75	0.75	0.06434	0.31374
DTC	norm.sel.-1876	0.74	0.74	0.74	0.74	0.27767	0.00078
NBC	norm.sel.-1876	0.67	0.66	0.67	0.67	0.00798	0.00117
ANN	norm.sel.-1876	0.75	0.75	0.75	0.75	195.73221	0.11046
SVM	sel.-800	0.73	0.73	0.73	0.73	0.03156	0.00032
$CEEF$	sel.-800	0.69	0.69	0.69	0.69	10.14099	0.00177
KNN	sel.-800	0.72	0.72	0.72	0.72	0.00074	0.02817
DTC	sel.-800	0.7	0.7	0.7	0.7	0.09234	0.00054
NBC	sel.-800	0.66	0.65	0.67	0.66	0.00207	0.00047
ANN	sel.-800	0.74	0.74	0.74	0.74	71.66657	0.09721

SVM	norm.sel.-800	0.73	0.73	0.73	0.73	0.014	0.00055
CEEF	norm.sel.-800	0.67	0.67	0.67	0.67	10.25722	0.002
KNN	norm.sel.-800	0.72	0.72	0.72	0.72	0.02493	0.03531
DTC	norm.sel.-800	0.7	0.7	0.7	0.7	0.06069	0.0006
NBC	norm.sel.-800	0.67	0.67	0.68	0.67	0.00258	0.00064
ANN	norm.sel.-800	0.72	0.72	0.73	0.72	71.87925	0.10601
SVM	sel.-1200	0.74	0.74	0.74	0.74	0.07149	0.00033
CEEF	sel.-1200	0.66	0.66	0.66	0.66	18.31524	0.0026
KNN	sel.-1200	0.72	0.72	0.72	0.72	0.0009	0.0696
DTC	sel.-1200	0.71	0.71	0.71	0.71	0.21992	0.00062
NBC	sel.-1200	0.64	0.63	0.65	0.64	0.00343	0.00066
ANN	sel.-1200	0.74	0.74	0.75	0.74	104.7257	0.09523
SVM	norm.sel.-1200	0.75	0.75	0.75	0.75	0.02916	0.00059
CEEF	norm.sel.-1200	0.68	0.68	0.68	0.68	17.48059	0.00284
KNN	norm.sel.-1200	0.74	0.74	0.74	0.74	0.0387	0.10646
DTC	norm.sel.-1200	0.72	0.72	0.72	0.72	0.13066	0.00072
NBC	norm.sel.-1200	0.65	0.64	0.66	0.65	0.00408	0.00083
ANN	norm.sel.-1200	0.74	0.74	0.74	0.74	105.05628	0.10217
SVM	sel.-400	0.71	0.71	0.71	0.71	0.00827	0.00017
CEEF	sel.-400	0.7	0.7	0.71	0.7	4.08961	0.00096
KNN	sel.-400	0.72	0.72	0.72	0.72	0.0006	0.00658
DTC	sel.-400	0.7	0.7	0.7	0.7	0.02039	0.00047
NBC	sel.-400	0.64	0.62	0.67	0.64	0.00101	0.00033
ANN	sel.-400	0.72	0.72	0.73	0.72	36.27646	0.09427
SVM	norm.sel.-400	0.72	0.72	0.72	0.72	0.00443	0.00039
CEEF	norm.sel.-400	0.69	0.69	0.69	0.69	4.11804	0.00122
KNN	norm.sel.-400	0.71	0.71	0.71	0.71	0.01202	0.00738
DTC	norm.sel.-400	0.69	0.69	0.69	0.69	0.01525	0.00049
NBC	norm.sel.-400	0.64	0.63	0.66	0.64	0.00123	0.00042
ANN	norm.sel.-400	0.72	0.72	0.72	0.72	36.3481	0.10659

Tabulka B.2: Výsledky jednotlivých klasifikátorů při zjišťování biologické odezvy molekul.

B.3 Rozpoznání objektu na obrázku

Parametry jednotlivých klasifikátorů:

- **SVM**
  - Normalize=None
  - Kernel=linear
- **CEEF**
  - Normalize=None
  - Number of generations=10
  - Stop accuracy=None

  - Population size=10
  - Selection method=RANK
  - Random seed=None
  - Max crossovers in generation=5
  - Max mutations=5
  - Max start slots=2
  - Crossover probability=0.75
  - Test set size=0.1
  - Change test set for each generation=False
  - Log generation fitness=True
- **KNN**
  - Normalize=None
  - Neighbors=5
- **DTC**
  - Normalize=None
  - Random seed=None
  - Split criterion=Gini Impurity
- **NBC**
  - Normalize=None
  - Type=gaussian
- **ANN**
  - Normalize=None
  - Random seed=None
  - Epochs=400
  - Batch size=128
  - Learning rate=0.001
  - GPU=True
  - Output layer activation function=softmax
  - Log epoch=True
  - Hidden layers=[LayerNeurons=1700, Activation function=relu, LayerNeurons=1000, Activation function=relu, LayerNeurons=100, Activation function=relu]

Parametry extraktoru příznaků:

- **Histogram of Oriented Gradients**
  - Number of orientation bins=9
  - Width of a cell [px]=4

  - Height of a cell [px]=4
  - Number of cells in each block (horizontal)=2
  - Number of cells in each block (vertical)=2
  - Normalization=L2

klasifikátor	data	A	F1	P	R	trénování[s]	testování[s]
CEEF	norm.-60000	0.18	0.1	0.16	0.18	119.86746	3.11791
CEEF	norm.-60000	0.18	0.14	0.17	0.18	121.28863	3.64454
CEEF	norm.-60000	0.16	0.12	0.18	0.16	70.79269	2.90746
SVM	norm.-60000	0.57	0.56	0.56	0.57	201.14854	0.07511
KNN	norm.-60000	0.55	0.54	0.59	0.55	0.28778	681.8729
DTC	norm.-60000	0.22	0.22	0.22	0.22	386.0924	0.03193
DTC	norm.-60000	0.22	0.22	0.22	0.22	375.80902	0.02675
DTC	norm.-60000	0.23	0.23	0.23	0.23	377.1665	0.03035
NBC	norm.-60000	0.49	0.49	0.49	0.49	1.41521	1.25182
ANN	norm.-60000	0.66	0.66	0.66	0.66	1075.53658	0.43159
ANN	norm.-60000	0.65	0.65	0.65	0.65	1098.97474	0.49079
ANN	norm.-60000	0.66	0.66	0.66	0.66	1145.34787	0.49766
CEEF	norm.sel.-60000	0.17	0.11	0.14	0.17	421.49483	1.23132
CEEF	norm.sel.-60000	0.16	0.12	0.21	0.16	379.17875	1.51927
CEEF	norm.sel.-60000	0.18	0.11	0.11	0.18	450.08568	1.38801
SVM	norm.sel.-60000	0.55	0.54	0.54	0.55	55.64052	0.03405
KNN	norm.sel.-60000	0.5	0.5	0.54	0.5	0.1224	291.95802
DTC	norm.sel.-60000	0.23	0.23	0.23	0.23	151.53814	0.04719
DTC	norm.sel.-60000	0.23	0.23	0.23	0.23	152.14762	0.01133
DTC	norm.sel.-60000	0.22	0.22	0.22	0.22	152.56599	0.01131
NBC	norm.sel.-60000	0.47	0.46	0.47	0.47	0.43604	0.20056
ANN	norm.sel.-60000	0.63	0.63	0.63	0.63	1640.83341	0.74795
ANN	norm.sel.-60000	0.64	0.64	0.64	0.64	1604.58876	0.79121
ANN	norm.sel.-60000	0.64	0.63	0.64	0.64	1427.28541	0.87737

Tabulka B.3: Výsledky jednotlivých klasifikátorů při rozpoznávání objektu na obrázku.

B.4 SPAM filtr

Parametry jednotlivých klasifikátorů:

- **SVM**
  - Normalize=None
  - Kernel=linear
- **CEEF**
  - Normalize=None
  - Number of generations=300
  - Stop accuracy=None

  - Population size=10
  - Selection method=RANK
  - Random seed=None
  - Max crossovers in generation=2
  - Max mutations=10
  - Max start slots=10
  - Crossover probability=0.75
  - Test set size=0.05
  - Change test set for each generation=False
  - Log generation fitness=True
- **KNN**
  - Normalize=None
  - Neighbors=3
- **DTC**
  - Normalize=None
  - Random seed=None
  - Split criterion=Gini Impurity
- **NBC**
  - Normalize=None
  - Type=gaussian
- **ANN**
  - Normalize=None
  - Random seed=None
  - Epochs=25
  - Batch size=1024
  - Learning rate=0.001
  - GPU=True
  - Output layer activation function=softmax
  - Log epoch=True
  - Hidden layers=[LayerNeurons=50, Activation function=relu, LayerNeurons=20, Activation function=relu]

Parametry extraktorů příznaků:

- **Hashing**
  - Non negative values=False
  - Number of features=1200 | Number of features=800 | Number of features=400

  - Normalization=l2

- **Term Frequency–Inverse Document Frequency**
  - Number of features=1200 | Number of features=800 | Number of features=400
  - Normalization=l2

klasifikátor	**data**	**A**	**F1**	**P**	**R**	**trénování[s]**	**testování[s]**
SVM	TF-IDF-800	0.96	0.96	0.96	0.96	1.05646	0.00088
ANN	TF-IDF-800	0.98	0.98	0.98	0.98	10.33485	0.32087
KNN	TF-IDF-800	0.93	0.93	0.93	0.93	0.00977	20.21107
DTC	TF-IDF-800	0.95	0.95	0.95	0.95	17.05717	0.00503
NBC	TF-IDF-800	0.92	0.92	0.92	0.92	0.97317	0.11636
CEEF	TF-IDF-800	0.83	0.83	0.84	0.83	104.65061	0.30171
SVM	TF-IDF-1200	0.97	0.97	0.97	0.97	1.11686	0.00087
ANN	TF-IDF-1200	0.98	0.98	0.98	0.98	12.0463	0.35778
KNN	TF-IDF-1200	0.9	0.9	0.91	0.9	0.00907	19.82348
DTC	TF-IDF-1200	0.96	0.96	0.96	0.96	20.24272	0.00552
NBC	TF-IDF-1200	0.94	0.94	0.94	0.94	1.3169	0.1583
CEEF	TF-IDF-1200	0.78	0.78	0.78	0.78	29.30433	0.07609
SVM	TF-IDF-400	0.94	0.94	0.94	0.94	0.95614	0.00073
ANN	TF-IDF-400	0.97	0.97	0.97	0.97	8.66692	0.3048
KNN	TF-IDF-400	0.95	0.95	0.95	0.95	0.00742	19.65375
DTC	TF-IDF-400	0.95	0.95	0.95	0.95	11.57934	0.00408
NBC	TF-IDF-400	0.88	0.88	0.89	0.88	0.48359	0.04646
CEEF	TF-IDF-400	0.83	0.83	0.83	0.83	56.98078	0.15805
SVM	Hashing-800	0.91	0.91	0.91	0.91	1.26334	0.00168
ANN	Hashing-800	0.96	0.96	0.96	0.96	11.81278	0.3605
KNN	Hashing-800	0.82	0.82	0.85	0.82	0.02024	41.82038
DTC	Hashing-800	0.92	0.92	0.92	0.92	83.74027	0.00613
NBC	Hashing-800	0.85	0.85	0.85	0.85	0.95767	0.1139
CEEF	Hashing-800	0.77	0.77	0.77	0.77	108.27582	0.31964
SVM	Hashing-1200	0.93	0.93	0.93	0.93	1.35697	0.0017
ANN	Hashing-1200	0.97	0.97	0.97	0.97	12.84572	0.36876
KNN	Hashing-1200	0.82	0.82	0.85	0.82	0.03187	37.56615
DTC	Hashing-1200	0.93	0.93	0.93	0.93	85.56342	0.0067
NBC	Hashing-1200	0.87	0.87	0.87	0.87	1.31761	0.15854
CEEF	Hashing-1200	0.78	0.78	0.79	0.78	160.76016	0.43341
SVM	Hashing-400	0.87	0.87	0.87	0.87	1.05863	0.00151
ANN	Hashing-400	0.95	0.95	0.95	0.95	9.68477	0.31511
KNN	Hashing-400	0.83	0.83	0.85	0.83	0.03201	46.2488
DTC	Hashing-400	0.91	0.91	0.91	0.91	66.62696	0.00565
NBC	Hashing-400	0.83	0.82	0.83	0.83	0.49481	0.03235
CEEF	Hashing-400	0.74	0.74	0.75	0.74	63.62628	0.17866

Tabulka B.4: Výsledky jednotlivých klasifikátorů při klasifikaci nevyžádané pošty.

B.5 Určení jazyka textu

Použité podmnožiny jazyků:

- {ar, bg, bs, cs, da, de, el, en, es, et, fa, fi, fr, he, hr, hu, id, it, nl, no, pl, pt, pt_br, ro, ru, sk, sl, sr, sv, th, tr, zh_cn}
- {ar, bg, cs, da, de, el, en, es, fa, fi, fr, he, hu, id, it, nl, pt, pt_br, ru, sk, sl, sv, tr, zh_cn}
- {bg, cs, de, el, en, es, fi, fr, he, hu, id, nl, pt, ru, sk, zh_cn}
- {cs, de, en, fi, fr, hu, nl, sk}

Parametry jednotlivých klasifikátorů:

- **SVM**
  - Normalize=None
  - Kernel=linear
- **CEEF**
  - Normalize=None
  - Number of generations=200
  - Stop accuracy=None
  - Population size=10
  - Selection method=RANK
  - Random seed=None
  - Max crossovers in generation=3
  - Max mutations=5
  - Max start slots=2
  - Crossover probability=0.75
  - Test set size=0.05
  - Change test set for each generation=False
  - Log generation fitness=True
- **KNN**
  - Normalize=None
  - Neighbors=3
- **DTC**
  - Normalize=None
  - Random seed=None
  - Split criterion=Gini Impurity

- **NBC**
  - Normalize=None
  - Type=gaussian
- **ANN**
  - Normalize=None
  - Random seed=None
  - Epochs=100
  - Batch size=1024
  - Learning rate=0.001
  - GPU=True
  - Output layer activation function=softmax
  - Log epoch=True
  - Hidden layers=[LayerNeurons=40, Activation function=relu]

Parametry extraktoru příznaků:

- **Hashing**
  - Non negative values=False
  - Number of features=40
  - Normalization=l2

klasifikátor	**použitých jazyků**	**A**	**F1**	**P**	**R**	**trénování[s]**	**testování[s]**
SVM	16	1.0	1.0	1.0	1.0	1.47201	0.00187
CEEF	16	0.95	0.94	0.94	0.95	30.38833	0.07948
KNN	16	1.0	1.0	1.0	1.0	0.00563	11.39346
DTC	16	0.98	0.98	0.98	0.98	2.48319	0.00094
NBC	16	0.99	0.99	0.99	0.99	0.01822	0.00705
ANN	16	1.0	1.0	1.0	1.0	14.77019	0.17477
SVM	32	0.94	0.93	0.93	0.94	8.02137	0.0056
CEEF	32	0.42	0.36	0.41	0.42	95.08527	0.26002
KNN	32	0.92	0.91	0.92	0.92	0.0095	45.49166
DTC	32	0.89	0.89	0.89	0.89	11.23328	0.00204
NBC	32	0.92	0.92	0.93	0.92	0.03711	0.02761
ANN	32	0.94	0.94	0.94	0.94	29.49535	0.37417
SVM	24	0.99	0.99	0.99	0.99	4.30693	0.00347
CEEF	24	0.46	0.39	0.42	0.46	53.94054	0.14049
KNN	24	0.97	0.97	0.97	0.97	0.00767	25.74985
DTC	24	0.95	0.95	0.95	0.95	6.12581	0.00142
NBC	24	0.98	0.98	0.98	0.98	0.02843	0.01499
ANN	24	0.99	0.99	0.99	0.99	21.55876	0.23778
SVM	8	1.0	1.0	1.0	1.0	0.31128	0.00085
CEEF	8	0.99	0.99	0.99	0.99	9.08848	0.02204

KNN	8	1.0	1.0	1.0	1.0	0.00312	2.855
DTC	8	0.99	0.99	0.99	0.99	0.62133	0.00063
NBC	8	0.99	0.99	1.0	0.99	0.00806	0.00214
ANN	8	1.0	1.0	1.0	1.0	6.99132	0.10555

Tabulka B.5: Výsledky jednotlivých klasifikátorů při určování jazyka titulků k filmům.