FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Zpracování akustických signálů z optických senzorů
Student:	Bc. Jakub Šedý
Vedoucí:	Ing. Radek Mařík, CSc.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Cílem práce bude analýza, návrh a tvorba aplikace umožňující třídění událostí a jejich lokalizaci v prostoru zaznamenaných optickými senzory.
1) Seznamte se se strukturou dat z optických vláknových mikrofonů.
2) Navrhněte a ověřte identifikaci vzniku události na vybraných signálech datových vzorků.
3) V mikrofonní síti navrhněte a implementujte metodu lokalizace místa vzniku události v prostoru.
4) Vhodnou metodou navrhněte klasifikaci identifikovaných událostí do tříd.
5) Na základě těchto znalostí vytvořte aplikaci, jejíž vstupem bude datový soubor a výstupem klasifikace případných událostí do předem připravených tříd. Proveďte řádný a zdůvodněný návrh architektury aplikace a její implementace.
6) Výsledky získané implementovanými metodami diskutujte.
7) Výslednou aplikaci řádným způsobem zdokumentujte a otestujte. Proveďte diskusi toho, do jaké míry byly dosaženy vytčené výsledky.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 1. února 2019

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Diplomová práce

Zpracování akustických signálů z optických senzorů

Bc. Jakub Šedý

Katedra softwarového inženýrství
Vedoucí práce: Ing. Radek Mařík, CSc.

9. května 2019

Poděkování

Chtěl bych poděkovat svojí rodině za podporu při studiu a při tvorbě diplomové práce. Dále bych chtěl poděkovat vedoucímu práce Radku Maříkovi za pevné nervy, trpělivost, cenné rady a spolupráci při psaní.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 9. května 2019 .....

Odkaz na tuto práci

Šedý, Jakub. *Zpracování akustických signálů z optických senzorů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Diplomová práce se zabývá klasifikací zvukových signálů získaných z optických vláken. Cíle dosahuje pomocí analýzy problému, návrhu architektury, implementace klientské a serverové aplikace. Dále se práce zabývá testováním klientské aplikace. Výsledná aplikace je schopná zpracovávat různé typy signálů v předem stanoveném formátu a klasifikovat je. Poslední část se věnuje hledání té nejvhodnější klasifikace pro experimentální data. Z výsledků experimentální části je patrné, že úspěšnost klasifikace u experimentálních dat ovlivňuje více předzpracování než samotná klasifikace.

Klíčová slova Strojové učení, Django, Angular, Python, Scikit-learn, Optická vlákna, Náhodný les, Rozhodovací strom, Podpůrné vektorové stroje, Logistická regrese, Naivní Bayes, Shlukování, Klasifikace, Deep learning, Umělé neuronové sítě, Rekurentní neuronové sítě, K-nejbližších sousedů

Abstract

The aim of the master thesis is to classify acoustic signals acquired from optical fibres. Main goal is achieved by analyzing the problem, designing the architecture of client and server applications and their implementation followed by a test of the client application. The final version of the application is able to

process different types of signals in predetermined format and classify them. Last part of this thesis focuses on finding the ideal classification for experimental data. Based on the results, it is apparent that success rate of classification for the experimental data is affected way more by preprocessing than the classification itself.

Keywords Machine learning, Django, Angular, Python, Scikit-learn, Optical fiber, Random forest, Decision tree, Support vector machine, Logistic regression, Naive Bayes, Clustering, Classification, Deep learning, Artificial neural network, Recurrent neural network, K-nearest neighbors

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V posledních letech zažívá strojové učení a práce s daty veliký rozkvět. V roce 2009 byla suma všech existujících dat 0,8 zettabytů, dnes je to již 33 zettabytů a v roce 2025 předpokládá agentura IDC sumu všech dat na 175 zettabytů [14]. S takovým nárůstem se objevují nové možnosti zpracování dat.

Optická vlákna se používají hlavně pro přenos dat, ale lze je využít i jiným způsobem - pro detekci událostí. Pokud se totiž světlo optického vlákna naruší, lze pomocí narušení lomu světla sledovat charakteristiku této události.

Má práce se zabývá právě tímto problémy - klasifikací zvukových signálů zaznamenaných pomocí optických vláken. Hlavním cílem práce je seznámit se s metodami pro klasifikaci, porozumět datům z optických vláken a následně se pokusit nalézt nejvhodnější metodu pro klasifikaci těchto dat a vytvořit aplikaci, která bude schopna klasifikovat události automaticky.

Strojové učení a práce s daty mne zajímají a vnímám je jako velice perspektivní. Možnost vyzkoušet si během diplomové práce řešení reálného problému, vidět jeho skutečné využití v praxi a zároveň proniknout pod povrch výše zmíněných oborů je hlavní důvod, proč jsem si právě toto téma diplomové práce vybral.

Ve své diplomové práci se zabývám popisem nejdůležitějších disciplín strojového učení, analýzou, návrhem, implementací, experimenty s daty a testováním. Aplikace se skládá z serverové a klientské části. Klientská část byla vytvořena s využitím frameworku Angular, zatímco serverová část byla vytvořena s využítím frameworku Django a knihovny pro strojové učení Scikit-learn.

V oboru strojového učení nejsou pro některé termíny definovány české překlady a zároveň v práci používám mnoho zkratek, k jejichž vysvětlení slouží příloha A.

Cíl práce

Cílem práce je analýza, návrh a tvorba aplikace umožnující třídění událostí a jejich lokalizaci v prostoru. Tento cíl se dále rozděluje na teoretickou, praktickou a experimentální část.

Teoretická část práce slouží k seznámení se s disciplínami strojového učení.

Praktická část práce slouží k popsání analýzy, návrhu, tvorbě a testování výsledné aplikace, která bude implementovat metody popsané v teoretické části.

Experimentální část slouží k nalezení nejvhodnějšího zpracování testovacích dat pro co nejúspěšnější klasifikaci.

Kapitola 1

Teoretická část

Následující kapitola se věnuje popisem esenciálních kroků strojového učení.

1.1 Porozumění datům

Před řešením problematiky strojového učení je důležité porozumět datům. Tomu se věnuje následující sekce.

1.1.1 Atribut

Atribut je datové pole, které reprezentuje vlastnost nebo charakteristiku datového objektu [15]. V literatuře se mohou také označovat jako dimenze, vlastnost nebo proměnná. Označení se většinou liší podle oboru. V DWH se používá dimenze, ve strojovém učení vlastnost, ve statistice proměnná a v oboru data miningu atribut. V tabulce 1.1 jsou ukázány atributy - každý řádek je jeden atribut. Množina atributů pro jeden objekt se nazývá příznakový vektor.

Index	0	1	2	3	4
ID	1	2	3	4	5
Jméno	Jan	Jana	Jakub	Jana	Jana
Příjmení	Novák	Nováková	Surový	Nevrlá	Surová
Velikost	L	M	XL	S	S
Typ karty	Classic	Premium		Classic	Simple
Nákup poslední měsíc	Ano	Ne	Ne	Ne	Ano
Pohlaví	Muž	Žena	Muž	Žena	Žena
Zaměstnání	Učitel	Sekretářka	Vývojář		
Útrata	10000	5000	100	9000	50000

Tabulka 1.1: Ukázka typu atributů.

1.1.2 Nominální atribut

Nominální nebo kategorický atribut obsahuje typy kategorií, stavů a nebo kódů [15]. Tyto hodnoty se nedají řadit a žádná z nich není nadřazená nad jinou. Hodnoty mohou být i číselné, ale je potřeba myslet na to, že ani tyto číselné hodnoty se nedají řadit (kategorie 1 není nadřazená kategorii 0). V tabulce 1.1 je tento atribut zastoupen v řádku zaměstnání.

1.1.3 Binární atribut

Binární atribut je velice podobný nominálnímu - také popisuje stav nebo kategorii, ovšem jeho hodnoty mohou být pouze binární a může tedy popisovat pouze dvě kategorie nebo dva stavy [15]. V tabulce 1.1 je tento atribut zastoupen v řádku pohlaví.

1.1.4 Pořadový atribut

Pořadový atribut je opět podobný nominálnímu s tím rozdílem, že se hodnoty dají porovnávat - neznáme ovšem, jak moc velký rozdíl mezi nimi je [15]. Tento atribut V tabulce 1.1 popisuje řádek velikost. Neznáme přesné rozdíly mezi hodnotami, ale víme, že S < M < L < XL.

1.1.5 Číselný atribut

Číselný atribut je kvantitativní - to znamená, že lze měřit jeho kvantita, která je reprezentována v celých nebo desetinných číslech [15]. Tyto atributy se dále rozdělují na intervalově škálované a poměrově škálované. Intervalově škálované atributy jsou měřeny ve fixních a stejných jednotkách. Poměrově škálované atributy mají vlastní nultý bod. V tabulce 1.1 je poměrově škálovaný atribut zastoupec v řádku útrata.

1.2 Předzpracování dat

V reálném světě jsou data často nekvalitní a nekonzistentní, a to negativně ovlivňuje strojové učení [15]. Většina algoritmů strojového učení je na tato data velice citlivá, a proto je potřeba je nejprve správně předzpracovat. Nejčastější problémy a jejich řešení je popsáno v této sekci.

1.2.1 Rozdělení dat

Při řešení problému strojového učení se ve většině případech rozdělují atributy na 2 typy - nezávislé a závislé. Je tedy potřeba se nejprve podívat na vstupní data a rozhodnout, které atributy jsou nezávislé (a ovlivňují závislé) a které jsou závislé [15]. V tomto kroku je důležité se také zbavit neužitečných atributů. Na obrázku je takový atribut například ID, Jméno a Příjmení.

Pokud bychom chtěli například vytvořit model pro předpokládanou útratu, tak by právě útrata byla závislý atribut. Mezi nezávislé atributy by se poté zařadily ostatní sloupce (velikost, typ karty, nákup poslední měsíc, pohlaví a zaměstnání).

1.2.2 Chybějící data

Může se stát, že v datech bude některým objektům chybět atribut [15]. Je několik možností, jak si s tímto problémem poradit. Každé řešení se může hodit na něco jiného a je tedy dobré o nich mít povědomí.

1.2.2.1 Ignorovat vstup

Jednoduchá metoda, která pouze ignoruje takové vstupy, které mají nevyplněné atributy [15]. Při této metodě můžeme přijít o důležitá data, protože budou z množiny kompletně odebrána. Pokud ovšem k objektu chybí několik atributů, může se tato metoda hodit.

1.2.2.2 Doplnit hodnoty manuálně

Další jednoduchá metoda, ale pro velké množiny dat velice neefektivní [15].

1.2.2.3 Doplnit chybějící atributy konstantou

Všechny atributy doplníme stejnou hodnotou [15]. Například nějakou, která jinak nelze dosáhnout. Taková hodnota může být třeba „UNKNOWN“. Při tomto řešení ovšem může aplikace zařadit objekty do stejné kategorie na základě této hodnoty.

1.2.2.4 Doplnit atribut průměrem nebo mediánem ze všech objektů

Tato metoda spočítá průměr nebo medián ostatních hodnot stejného atributu a všude, kde atribut není vyplněný, doplní stejnou hodnotu [15].

1.2.2.5 Doplnit atribut průměrem nebo mediánem z objektů, které se řadí do stejné třídy

Tato metoda spočítá průměr nebo medián hodnot stejného atributu pouze u objektů, které jsou zařazené do stejné třídy [15]. Například neznáme příjem člověka, ale víme, že je student a u ostatních studentů atribut příjem vyplněný máme. Je možné tedy všem objektům, kterým chybí atribut, ze stejné třídy (např. student) doplnit příjem.

1.2.2.6 Vypočítáním nejpravděpodobnější hodnoty

Dopočítáním hodnoty podle metod strojového učení [15]. Pokud data obsahují hodně atributů, z kterých lze chybějící atribut vypočítat, pak je tato metoda nejpřesnější. Pokud například data obsahují atributy věk, dosažené vzdělání, typ zaměstnání a počet let praxe, tak lze na základě těchto údajů pomocí regrese vypočítat mzdu.

1.2.3 Kategorizační data

Kategorizační data je třeba také předzpracovat. [5] Nejprve je nutné rozhodnout, jestli jsou data pořadová nebo nominální. Pokud jsou data pořadová a obsahují nečíselné hodnoty, je nezbytné je zakódovat do číselných hodnot. Pokud jsou ovšem data nominální, tak je nelze pouze zakódovat do hodnot, protože algoritmy pro strojové učení tyto hodnoty porovnávají. Pokud by se tedy hodnoty zakódovaly následovně:

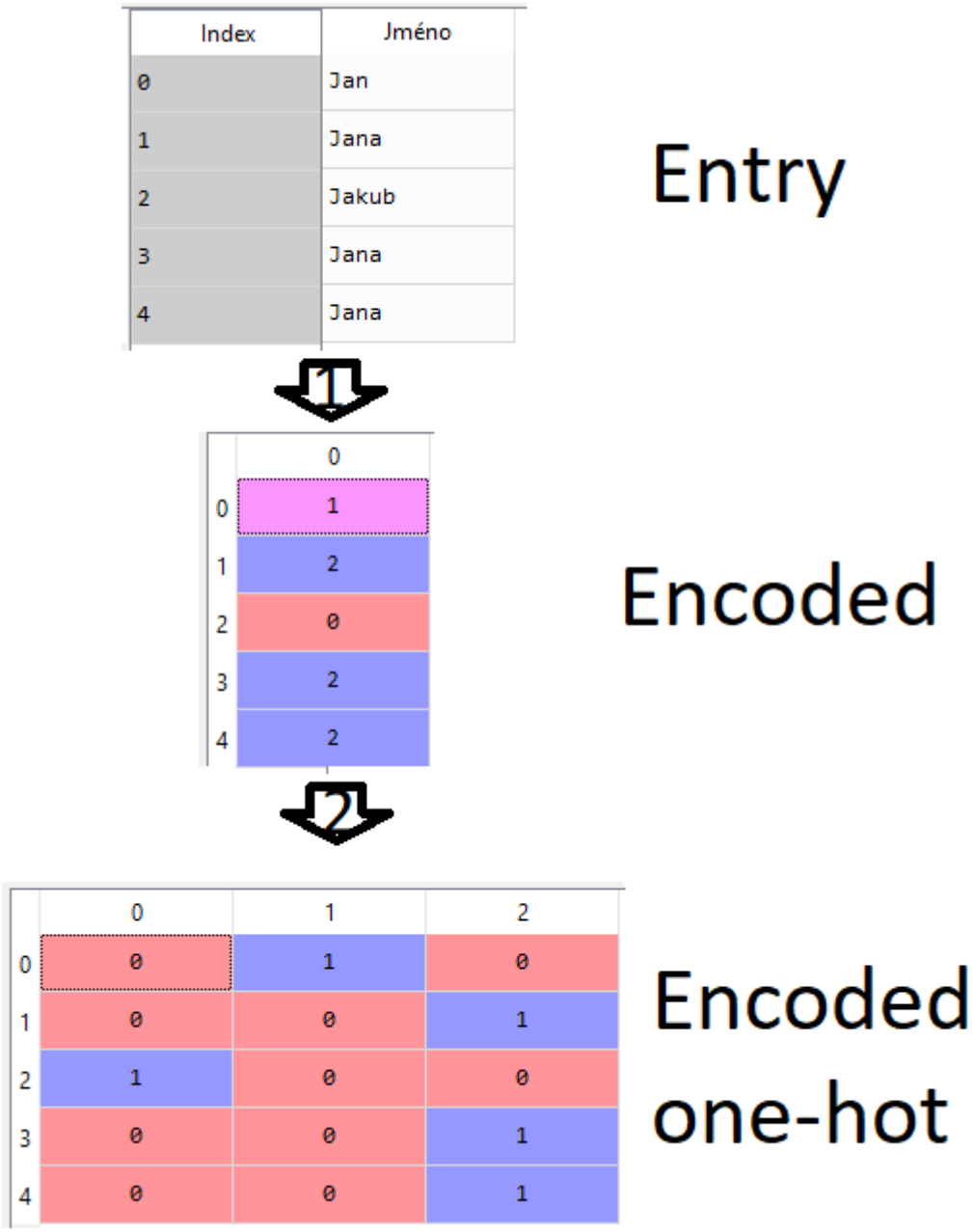
- Jan do hodnoty 0,
- Jakub do hodnoty 1,
- Jana do hodnoty 2.

Pak by algoritmus usoudil, že Jan < Jakub < Jana, přitom mezi atributy není žádný vztah a nelze je porovnávat. Takový problém se řeší vytvořením dummy sloupců pro každou kategorii a zakódováním do kódu 1 z n. To znamená, že ve sloupcích jsou pouze hodnoty 0 (pokud do kategorie nepatří) a 1 (pokud do kategorie patří). Příklad zakódování nominálních atributů je vidět na obrázku 1.1.

1.2.4 Testovací a trénovací množina

Pro většinu algoritmů strojového učení je potřeba dvou typů dat [5]. Ta, na kterých se algoritmus učí (trénovací), a ta, na kterých algoritmus zkouší, co se naučil (testovací). Obsah dat je v obou množinách stejný (u obou typů dat je potřeba znát závislé atributy), a proto se tato data vytváří tak, že se rozdělí naměřené hodnoty na dvě množiny. Na otázku, v jakém poměru neexistuje obecná odpověď. Pokud uřízneme moc velkou část z trénovací množiny, může algoritmus přijít o důležité informace. Pokud ale bude testovací množina zas moc malá, nebude hodnocení výsledků výstižné. Pro menší množiny dat jsou nejčastejší poměry trénovací a testovací množiny 60:40,70:30 a 80:20. Pokud je objem dat opravdu velký, poté je možné si dovolit testovat na menším procentu dat, používá se tedy poměr 90:10 nebo dokonce 99:1.

Tato rozdělení množin se používají hlavně pro vyhodnocení přesnosti a vybrání modelu. Po vybrání modelu je nejlepší daný model naučit na celé množině pro optimální výsledky.



Obrázek 1.1: Ukázka kódování 1 z n.

1.2.5 Škálování vlastností

Mezi další kritický krok předzpracování dat patří škálování vlastností [5]. Uvažujme dva atributy - věk a mzda. Jejich očekávané hodnoty jsou v jiných řádech. Věk v hodnotách od 0 do 100 a mzda v hodnotách od 0 do 1 000 000. Při většině algoritmů strojového učení se měří vzdálenost, což je pro rozdílné hodnoty velký problém. Při měření Eukleidovi vzdálenosti mezi hodnotami [20, 1 000 000] a [80, 120 000] je velikost druhého prvku o tolik větší, že bude věk úplně ignorován.

Proto je třeba hodnoty dostat do stejného řádu. Dva nejčastější přístupy převádění hodnot jsou normalizace a standardizace. Tyto dva pojmy jsou často používány dost volně v různých odvětvích.

1.2.5.1 Normalizace

Nejčastěji je jako normalizace bráno přeškálování hodnot atributů do intervalu mezi 0 a 1, což je speciální případ min-max škálování [5]. Tato metoda je nejvíce užitečná, když potřebujeme hodnoty v nějakém intervalu. Vzoreček pro takovou normalizaci je:

$$x_{norm}^{(i)} = \frac{x^i - x_{min}}{x_{max} - x_{min}}$$

kde $x^{(i)}$ je hodnota atributu, x_{max}, je maximální hodnota atributu a x_{min} je minimální hodnota atributu.

1.2.5.2 Standardizace

Některé algoritmy inicializují své váhy blízko k 0 [5]. Při standardizaci směrujeme hodnoty tak, aby byla střední hodnota 0 s rozptylem 1 a aby hodnoty měly formu normálního rozdělení, což usnadňuje správné určení vah algoritmů. Standardizace také udržuje informaci o vyčnívajících hodnotách a dělá tak algoritmy vůči takovým hodnotám méně náchylné. Standarizace je popsána následným vzorečkem:

$$x_{std}^{(i)} = \frac{x^i - \mu_x}{\sigma_x}$$

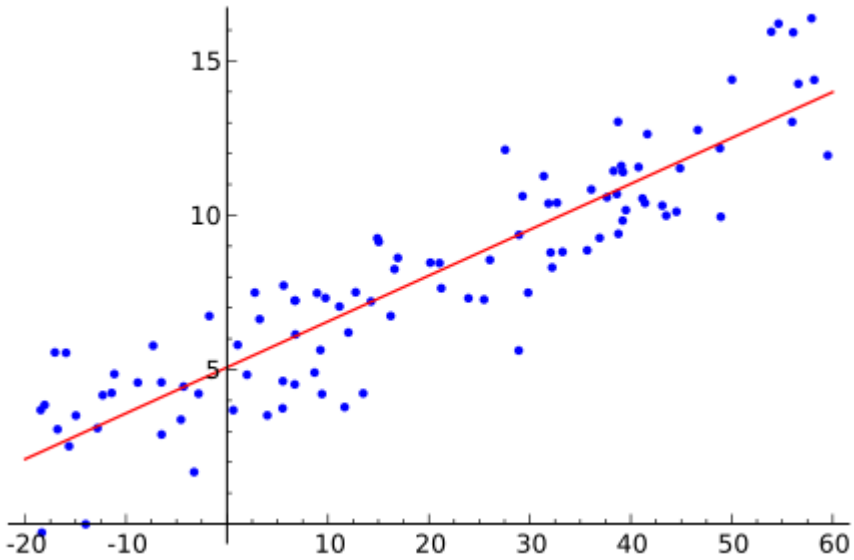
kde μx je střední hodnota atributu a σ_x je rozptyl.

1.3 Regrese

Regrese je jedním z oborů strojového učení [16]. Tento obor predikuje závislý atribut na základě nezávislých atributů pomocí funkce a vrací číslo. Hodí se například pro předpovídání mzdy na základě věku, let praxe a dosaženého vzdělání.

input	standardized	normalized
0.0	-1.336306	0.0
1.0	-0.801764	0.2
2.0	-0.267261	0.4
3.0	0.267261	0.6
4.0	0..801764	0.8
5.0	1.336306	1.0

Tabulka 1.2: Rozdíl mezi normalizací a standardizací na číslech od 0 do 5, převzato z [5].



Obrázek 1.2: Příklad lineární regrese, převzato z [1].

1.3.1 Jednoduchá lineární regrese

„Lineární regrese je metoda, při které je soubor bodů v grafu proložen přímkou, neboť předpokládáme, že závislost y na x lze graficky vyjádřit přímkou (viz. obrázek). Pokud měřené body proložíme přímkou, lze vypočítat odchylku a to tak, že odečteme od y hodnoty měřeného bodu y hodnotu vytvořené přímky. Podstatou lineární regrese je nalézt právě takovou přímku, aby součet druhých mocnin zmíněných odchylek byl co nejmenší, jedná se tedy o aproximaci daných hodnot přímkou a to metodou nejmenších čtverců.“[1]

Předpis funkcelinéarní regrese je následující [16]:

$$f(x) = a + bx$$

kde x je hodnota atributu a a a b jsou váhy, které se získají pomocí metody nejmenších čtverců z trénovacích dat. Kritérium pro optimalizaci metody

nejmenších čtverců vypadá následovně:

$$\sum_{i=1}^{n} \left(x^{(i)} - (a + bx^{(i)}) \right)^2$$

kde $x^{(i)}$ je naměřená hodnota. Při hledání optimálních hodnot a a b se v algoritmu hledá nejmenší suma.

1.3.2 Mnohočetná lineární regrese

Mnohočetná lineární regrese je velice podobná té jednoduché [7]. Hlavní rozdíl, jak už z názvu vyplývá, je v počtu nezávislých atributů. Předpis funkce pro mnohočetnou lineární regresi vypadá následovně:

$$f(x) = a + \sum_{i=1}^{k} b_i x_i$$

kde x_i je jeden z nezávislých atributů a b_i je váha nezávislého atributu a k je počet nezávislých atributů. Kritérium pro optimalizaci metody nejmenších čtverců je také velmi podobný:

$$\sum_{i=1}^{n} \left(x^{(i)} - \sum_{j=1}^{k} b_j x_j^{(i)} \right)^2$$

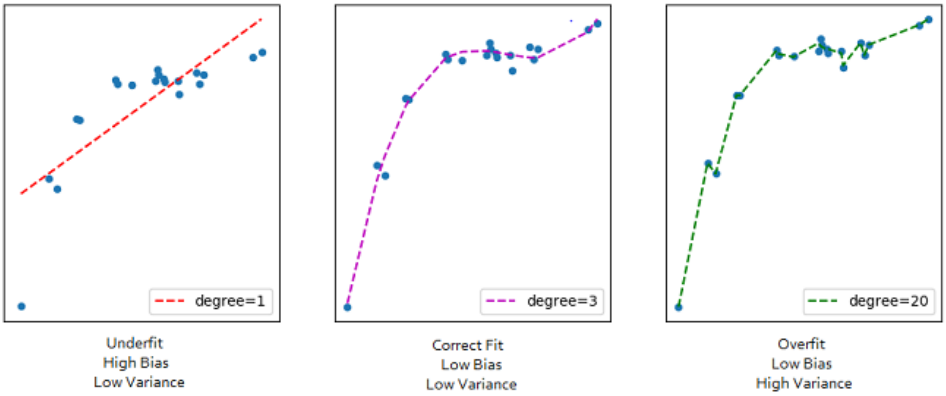
1.3.3 Polynomiální regrese

Lineární regrese vyžaduje, aby relace mezi nezávislými a závislými atributy byly lineární, ale mohou nastat i jiné případy, kdy relace nejsou lineární [2]. Pokud relace nejsou lineární, jednoduchá ani mnohočetná lineární regrese nejsou k řešení problému vhodné.

K takovému problému je vhodné využít polynomiální regresi. Predpis funkce pro pro polynomiální regresi je:

$$f(x) = a + \sum_{i=1}^{k} b_i x^i$$

kde k je stupeň polynomu a b_i jsou váhy jednotlivých polynomů. Polynomiální regrese je brána jako lineární, protože koeficienty a váhy jsou stále lineární. x^i je brána pouze jako další vlastnost. Při výběru stupně k je důležité vědět, že větší stupeň nemusí znamenat lepší výsledky. Může dojít k takzvanému overfittingu. Ilustrace je vidět na obrázku 1.3.

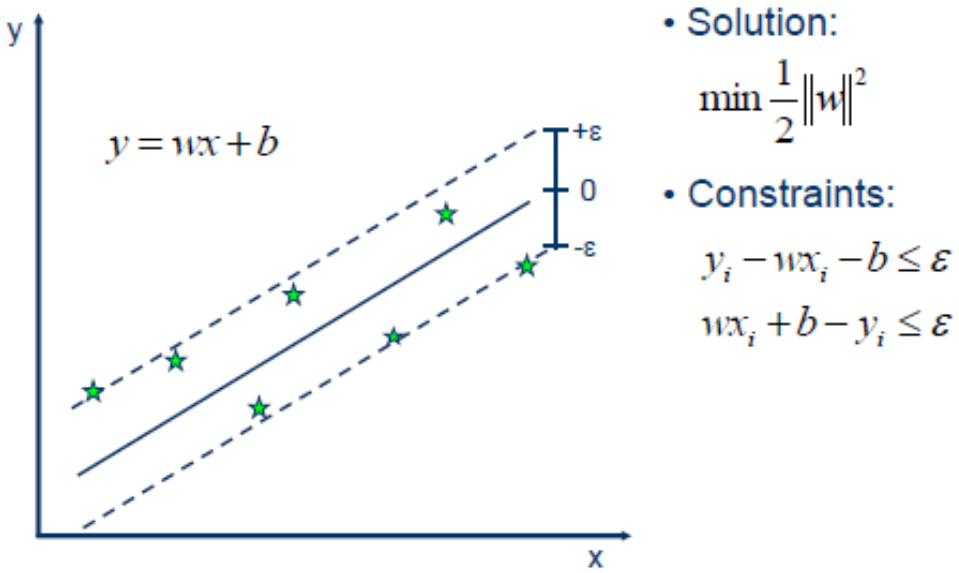


Obrázek 1.3: Příklad overfitingu, převzato z [2].

1.3.4 SVR

Support vector regression funguje na stejném principu jako SVM 1.4.3, s vyjímkou toho, že regresní model předpovídá reálná čísla.

Algoritmus pracuje s tolerancí ϵ a tím vyznačuje okraje hodnot, se kterými bude pracovat [3]. Pokud jsou hodnoty za těmito okraji, tak jsou algoritmem ignorovány.



Obrázek 1.4: Příklad SVR, převzato z [3].

1.3.5 Decision tree regrese

Rozhodovací stromy tvoří regresní modely ve formě stromové struktury [17]. Rozděluje dataset na podmnožiny. Výsledek je strom s rozhodovacími uzly a listy. Každý rozhodovací uzel má alespoň 2 potomky a list obsahuje předpovídanou hodnotu. Rozhodovací stromy umí pracovat i s kategorickými daty (nominální a ordinální atributy).

1.3.6 Random forest regrese

Random forest model se stal velmi populární ve strojovém učení díky dobré přesnosti, škálování a jednoduchosti použití [5]. Funguje stejně jako Decision tree model 1.3.5, ovšem random forest vytváří n stromů místo jednoho a pro většinu případů je tedy přesnější. V regresním modelu se hodnoty z každého stromu zprůměrují. Rostoucí počet stromů by měl do určité hodnoty zlepšovat přesnost na úkor ceny výpočtu.

1.3.7 Hodnocení regrese

K vybrání vhodného modelu je velice důležité zvolit správná kritéria pro měření. Tomu se věnuje tato sekce.

1.3.7.1 R-Square

R-Square je statistické měřítko, které popisuje, jak blízko jsou hodnoty k regresní křivce [7]. Této hodnotě se také říká koeficient determinace. Vzoreček pro výpočet je:

$$R^2 = 1 - \frac{\sum_{i=0}^{k} (y_i - \hat{y}_i)}{\sum_{i=0}^{k} (y_i - \bar{y}_i)}$$

kde $\bar{y}_i$ je průměrná hodnota všech hodnot y_i a $\hat{y}_i$ je hodnota předvídaná regresním modelem. Zpravidla platí, že čím větší hodnota, tím lépe, ale existují i výjimky.

1.3.7.2 Adjusted R-Square

Hlavním problém u R-square je ten, že má tendenci růst s počtem nezávislých vlastností [7]. Pokud máme mnohočetný linéarní model:

$$y = a + b_1x_1 + b_2x_2$$

a jeho R-Square pojmenujeme R_a^2 a zkusíme přidat třetí vlastnost:

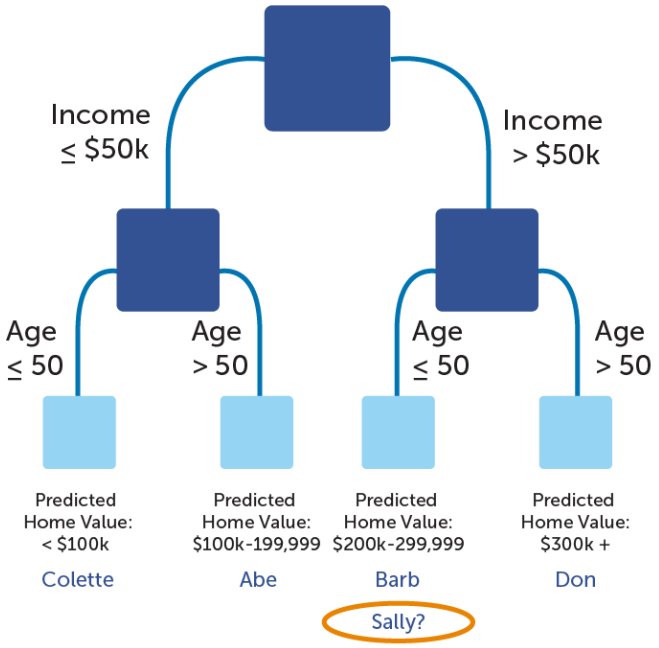
$$y = a + b_1x_1 + b_2x_2 + b_3x_3$$

a R-Square pojmenujeme R_b^2, tak platí, že $R_b^2 \geq R_a^2$ - to znamená, že by každá přidaná vlastnost měla pouze pozitivní vliv na R-Square, i kdyby neměla žádný vztah s závislou vlastností.

target indicator

	Age	Income	Home Value
Abe	70	$30,000	$150,000
Barb	30	$70,000	$250,000
Colette	25	$45,000	$98,000
Don	57	$90,000	$300,000
Sally	33	$55,000	$175,000

training data



Obrázek 1.5: Příklad rozhodovacího stromu, převzato z [4].

Tento problém řeší Adjusted R-Square, který penalizuje každou přidanou vlastnost do modelu:

$$\bar{R}^2 = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

kde n je počet pozorování a p je počet vlastností v regresním modelu.

1.4 Klasifikace

Klasifikace je dalším z oborů strojového učení [7]. Na rozdíl od regrese má pevně danou konečnou množinu hodnot, kterých můžou závislé vlastnosti dosahovat, a slouží k určování tříd na základě nezávislých vlastností.

1.4.1 Logistická regrese

Logistická regrese je jednoduchá na implementaci a podobná ostatním regresním modelům [5]. Jedná se klasifikační model. Pokud jsou data lineárně oddělitelná, je logistická regrese vhodným a velmi účinným modelem.

Jedná se o pravděpodobnostní model, který využívá *logit* funkci. Pro její pochopení je potřeba si definovat poměr pravděpodobností, který je popsán vzorcem:

$$\frac{P}{(1-P)}$$

kde P znamená pravděpodobnost, že událost nastane. Dále je potřeba definovat logit funkci, která je logaritmem poměru pravděpodobností, tedy:

$$logit\,(P) = log\frac{P}{(1-P)}$$

Tato funkce přijímá pravděpodobnost P, že jev nastane, takže $D_{logit} = (0, 1)$.Z Předpisu je zřejmé, že tento obor hodnot je bez krajních hodnot. Funkce vyjadřuje logartimický poměr pravděpodobností, proto $H_{logit} = \mathbf{R}$. Díky tomu můžeme vyjádřit lineární vztah mezi vlastnostmi a poměrem pravděpodobností:

$$logit\,(P\,(y = k \mid x)) = a + b_1x_1 + b_2x_2 + ... + b_nx_n$$

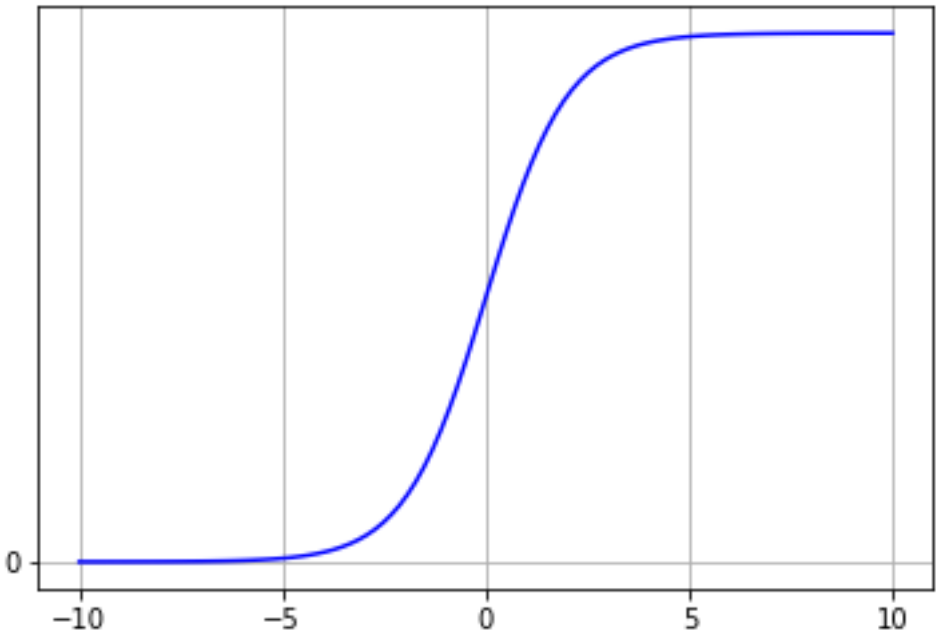
kde $P\,(y = k \mid x)$ je podmíněná pravděpodobnost, že objekt x patří do třídy k. K předpovídání pravděpodobnosti, že objekt patří do třídy je potřeba inverzní funkce. Tato funkce se nazývá logistická nebo také sigmoida, díky svému tvaru do písmene S. Je definována následovně:

$$\phi\,(z) = \frac{1}{1+e^{-z}}$$

kde z je lineární kombinace vah a vlastností konkrétního objektu, tedy:

$$z = a + b_1x_1 + b_2x_2 + ... + b_nx_n$$

Na obrázku 1.6 je vidět jednoduchý příklad sigmoidy. Hodnota $\phi\,(z)$ uvádí pravděpodobnost, s jakou patří tato lineární kombinace do třídy. Většinou se tyto hodnoty ještě převádí na binární, tedy $y = 1$ pokud $\phi\,(z) \geq 0.5$ a $y = 0$ pokud $\phi\,(z) < 0.5$. [5].



Obrázek 1.6: Ukázka sigmoidy.

1.4.2 K-nejbližších sousedů

Dalším klasifikačním algoritmem je K-NN algoritmus, který je velice odlišný od logistické regrese [5]. Je to příklad algoritmu „lazy learner“, což znamená, že si nevytváří žádnou vlastní funkci, pouze si pamatuje trénovací data. Cena při učení je tedy na rozdíl od ostatních algoritmů malá.

Postup pro klasifikaci je u tohoto algoritmu velice jednoduchý a je ukázán na obrázku. Jeho kroky jsou:

1. Vyber k (počet sousedů) a metodu pro měření vzdálenosti.

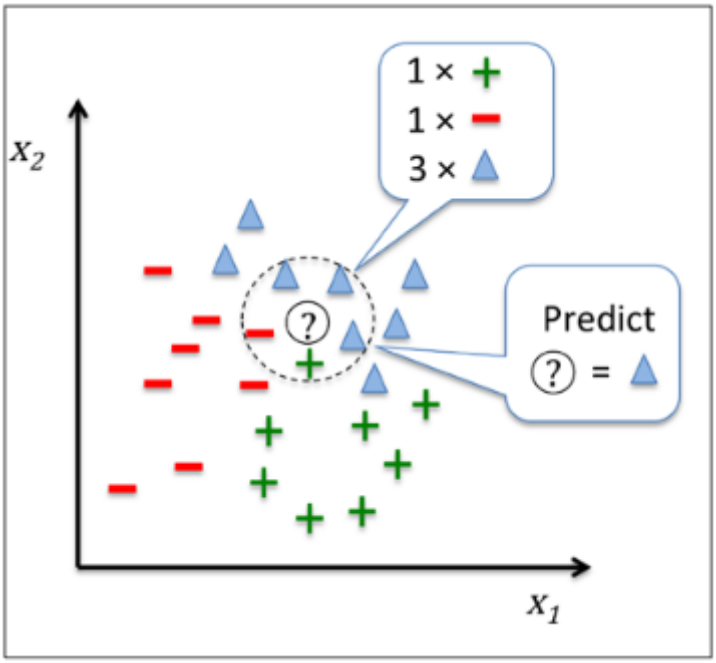
2. Najdi k nejbližších sousedů k vzorku, který chceme klasifikovat.

3. Přiřad vzorku třídu, na základě majoritního hlasování (třída, do které patří nejvíce sousedů).

1.4.3 SVM

Na rozdíl od jiných klasifikačních algoritmů jsou pro SVM nejdůležitější takové body(vektory), které jsou si nejvíce podobné a určují hranice mezi typy tříd. Těmto bodům se říká podpůrné vektory [6].

Tyto hranice nemusí být jednoduché najít. Je možné, že data nepůjdou takto rozdělit. V tom případě se za každý špatně určený vektor určuje penali-



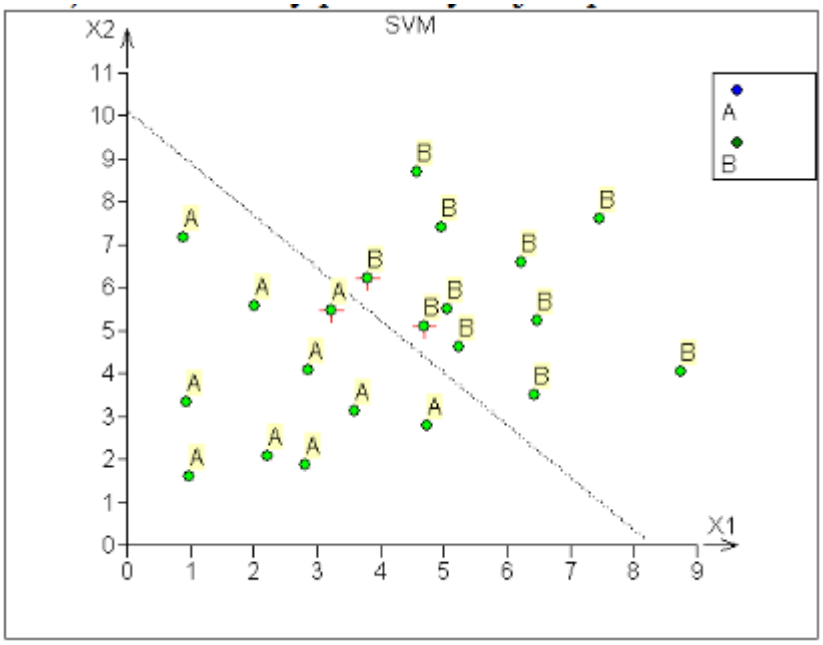
Obrázek 1.7: Ukázka K-NN algoritmu, převzato z [5].

zace. Poté se hledá taková hranice, která má co nejmenší penalizaci (nesprávně klasifikovaných hodnot).

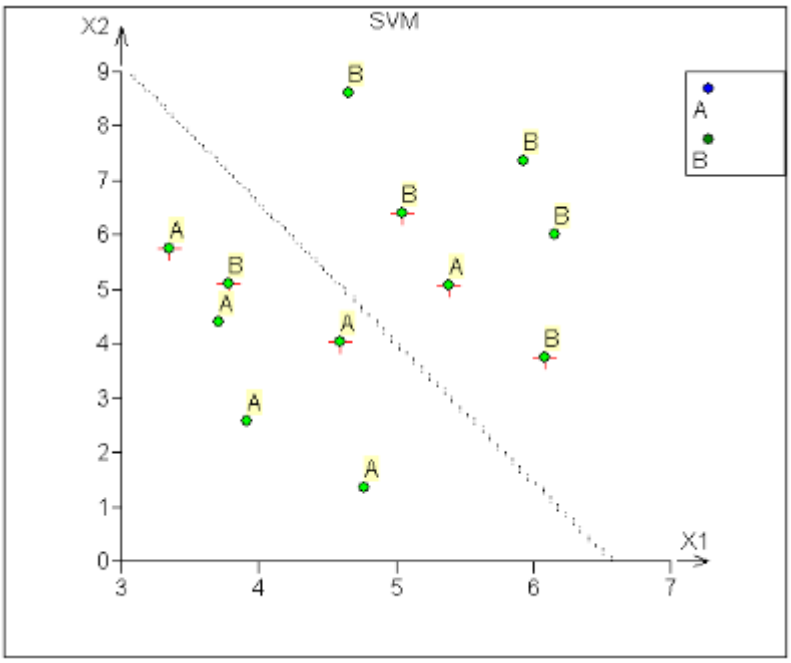
1.4.4 Kernel SVM

SVM popsané v svm nelze jednoduše použít na nelinéarní problémy. Většinou nelze data oddělit jednoduše a je tedy třeba transformovat jádro (angl. kernel) tak, aby to bylo možné [6]. Nejjednodušším příkladem je 1D jádro ukázané na obrázku 1.10. Nelze ho rozdělit přímkou, pokud ovšem převedeme hodnoty do 2D jádra, lze již třídy přímkou oddělit 1.11.

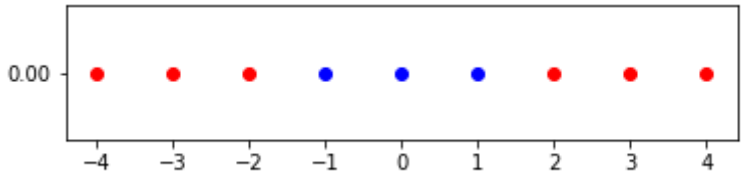
Ve většině případů se ovšem transformuje do složitějších jader [18]. Pro zpracování obrazu je to polynomiální jádro. Dále se poté nejčastěji používá Gaussovo jádro a Laplaceovo jádro.



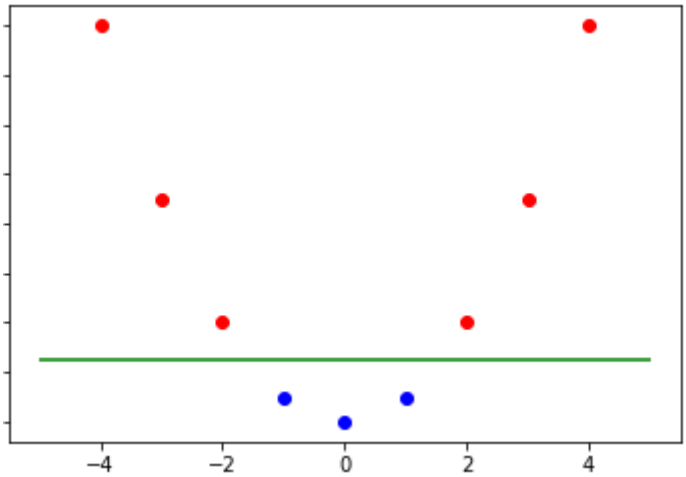
Obrázek 1.8: Ukázka SVM separovatelných dat, převzato z [6].



Obrázek 1.9: Ukázka SVM neseparovatelných dat, převzato z [6].



Obrázek 1.10: Ukázka dat v 1D, které lze oddělit pouze pomocí změny jádra.



Obrázek 1.11: Ukázka transformace z 1D jádra na 2D jádro.

1.4.5 Naivní Bayes

Další pravděpodobnostní klasifikátor je Naivní Bayes [7]. Hledá nejpravděpodobnější třídu popsanou atributy. Naivní se nazývá proto, že předpokládá, že všechny atributy jsou mezi sebou nezávislé, to znamená, že:

$$P\left(x \mid y=k\right)=\prod_{i=1}^{n} P\left(x_i \mid y=k\right)$$

I přesto, že tento předpoklad není v praxi téměř nikdy splněn, má tento algoritmus velké využití a velmi dobré předvídací výsledky. Pravděpodobnost je popsána následujícím vzorcem:

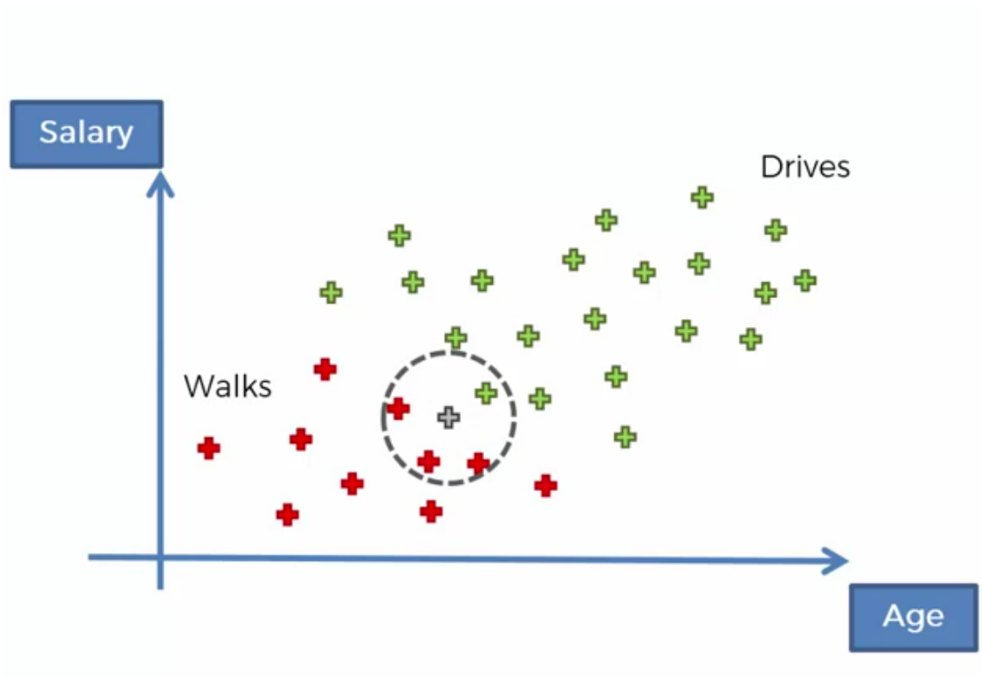
$$P\left(y=k \mid x\right)=\frac{P\left(x \mid y=k\right) * P\left(y=k\right)}{P\left(x\right)}$$

kde:

- $P\left(x \mid y=k\right)$ je pravděpodobnost, že třída k bude mít atributy dostatečně podobné x ,

- $P(y = k)$ je pravděpodobnost, že třída bude právě k - tedy počet objektů třídy k vydělený počtem všech objektů,

- $P(x)$ je pravděpodobnost, že atributy budou dostatečně podobné x podle algoritmem určeného rádiusu (jako na obrázku 1.12).



Obrázek 1.12: Ukázka atributů podobných x, převzato z [7].

1.4.6 Decision tree klasifikace

Klasifikace pomocí algoritmu Decision tree je velice podobná regresní [5]. Jediný rozdíl je ve výstupu algoritmu, který vrací identifikátor třídy.

1.4.7 Random forest klasifikace

Stejně jako u decision tree klasifikace je random forest klasifikace velmi podobná regresní [5]. Každý strom vrací identifikátor třídy a random forest klasifikace tedy počítá pravděpodobnost, s jakou klasifikovaná entita patří do určitého typu třídy (každý strom může vrátit jiný typ třídy).

Mezi hlavní výhody této klasifikace patří odolnost vůči odchýleným hodnotám a jednoduchý výběr parametrů - jediným parametr, který je potřeba vybrat, je počet stromů.

1.4.8 Hodnocení klasifikace

Podobně jako u regrese je potřeba vyhodnotit kvalitu klasifikaci. Tomu se věnuje následující sekce.

1.4.8.1 Matice záměn

Matice záměn (angl. Confusion matrix) je jednoduchá matice, která má v každém ze svých prvků přirozená čísla [5]. Pro každý prvek z matice $a_{i,j}$ je hodnota vyplněna následovně: počet událostí, které patří do i, ale algoritmus vyhodnotil, že patří do j. Na diagonále jsou tedy správně predikované hodnoty a všechny okolní hodnoty značí počet chyb.

Nejjednodušší matice záměn je taková, která vyhodnocuje pouze ANO/NE. poté každý prvek má vlastní název:

- prvek $a_{0,0}$ je true positive (TP) - správně vyhodnocené ANO,
- prvek $a_{0,1}$ je false negative (FN) - špatně vyhodnocené NE,
- prvek $a_{1,0}$ je true positive (FP) - špatně vyhodnocené ANO,
- prvek $a_{1,1}$ je true negative (TN) - správně vyhodnocené NE.

Z této matice se následně počítají další hodnoty.

		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Obrázek 1.13: Matice záměn, převzato z [5].

Error (ERR) Popisuje chybovost algoritmu - poměr špatně vyhodnocených klasifikací oproti všem klasifikacím, tedy:

$$ERR = \frac{FP + FN}{FP + FN + TP + TN}$$

Accuracy (ACC) Popisuje přesnost algoritmu - poměr správně vyhodnocených klasifikací oproti všem klasifikacím, tedy:

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR$$

.

Precision (PRE) Popisuje počet správně pozitivně ohodnocených, oproti všem, které algoritmus vyhodnotil jako pozitivní, tedy:

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

Recall (REC) Někde uváděno také jako true positive rate (TPR), popisuje počet správně pozitivně ohodnocených oproti všem, které měly být pozitivně ohodnocené, tedy:

$$REC = \frac{TP}{FN + TP}$$

.

False positive rate (FPR) Popisuje počet špatně pozitivně ohodnocených oproti všem, které měly být negativně ohodnocené, tedy:

$$FPR = \frac{FN}{FN + TN}$$

.

F1 score ($F1$) Nejčastěji používané je takzvané F1 skóre, které je kombinací PRE a REC. K hodnotě se dostaneme pomocí vzorce:

$$F1 = 2\frac{PRE * REC}{PRE + REC}$$

1.4.8.2 Paradox přesnosti

Uvažujme algoritmus, který vyhodnotil hodnoty jako na obrázku 1.14. Jeho přesnost je 98%.

	0	1
0	970	15
1	5	10

Obrázek 1.14: Matice záměn sofistikovanějšího algoritmu.

	0	1
0	985	0
1	15	0

Obrázek 1.15: Matice záměn hloupého algoritmu.

Nyní uvažujme jiný hloupý algoritmus, který vždy bude klasifikovat pozitivně jako na obrázku 1.15. Tento algoritmus má přesnost $98,5\%$.

Ne vždy musí být vyšší přesnost lepší, a proto se používají i jiná hodnocení. Pokud bychom chtěli detekovat převážně negativní hodnocení, tak by nám druhý (přesnější) algoritmus moc dobře neposloužil. Dalším problémem je nepoměr pozitivních a negativních událostí, což je v praxi také běžné. Je tedy potřeba se u každého problému zamyslet nad tím, co je pro nás správné hodnocení.

1.5 Shlukování

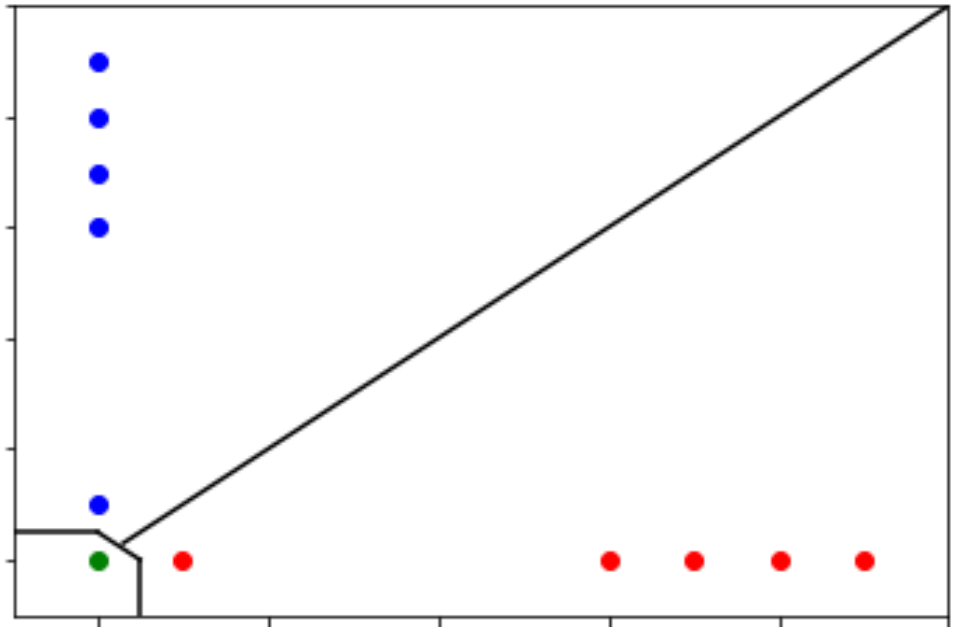
Předchozí dvě sekce byly věnovány takzvanému učení s učitelem - tedy učení s daty, u kterých algoritmus ví, jak mají dopadnout [5]. Shlukování (angl. Clustering) je učení bez učitele - tedy nezná správné hodnoty vázaných atributů. Funguje tak, že hledá nejpodobnější objekty mezi sebou a ty dává do jedné skupiny, která se nazývá shluk.

1.5.1 K-means

K-means je jeden z nejvíce populárních algoritmů, který je velice používaný v akademické a průmyslové sféře [5]. K v názvu algoritmu znamená počet shluků a je třeba zvolit správnou hodnotu počtu shluků, kterou volí sám uživatel, a je to hlavní nevýhoda tohoto algoritmu. Algoritmus pracuje s takzvanými centroidy, což je střed shluku. Postup algoritmu je následovný:

1. Náhodně vyber K centroidů.
2. Každému bodu přiřaď nejbližší centroid.
3. Posuň centroid tak, aby byl uprostřed všech bodů, které byly přiřazeny k tomuto centroidu.
4. Opakuj body 2 a 3, pokud se alespoň jednomu bodu změnil nejbližší centroid a nebo dokud nenastal maximální počet iterací.

V algoritmu se pracuje s počítáním vzdálenosti. Metod pro počítání vzdálenosti je mnoho a je tedy potřeba vybrat tu správnou pro daný problém. Nejčasteji se používá Eukleidova vzdálenost.



Obrázek 1.16: Ukázka problému při algoritmu K-means.

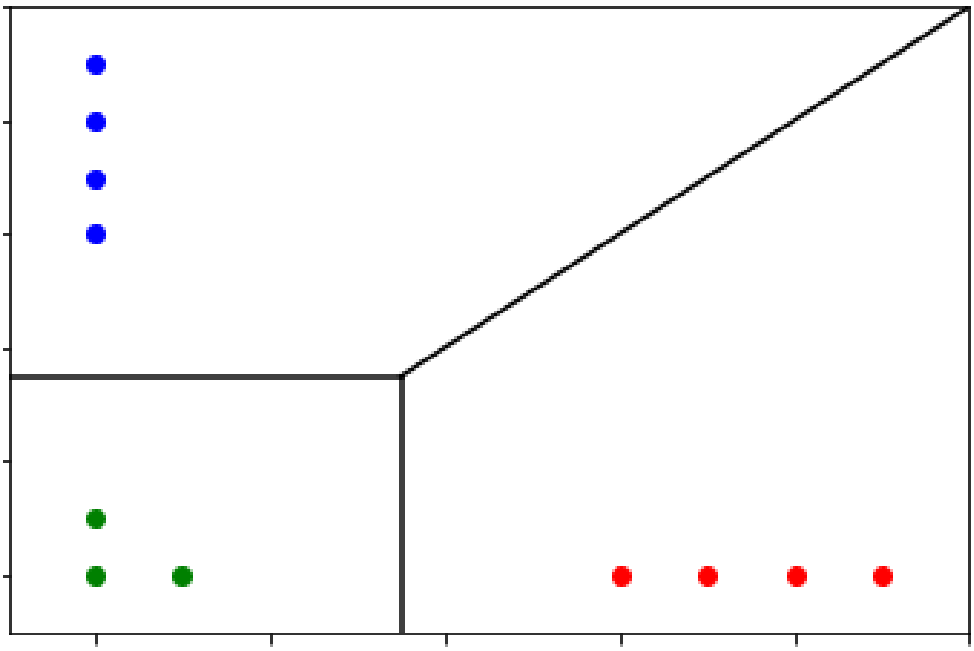
V algoritmu může nastat problém při prvním kroku. Je možné dosáhnout takového stavu, který je konečný, ale není správným řešením. Pokud bychom vybrali 3 blízké body jako na obrázku 1.16, tak by byl stav konečný, ale špatný. Výsledné rozdělení clusterů by mělo vypadat jako na obrázku 1.17. Tento problém řeší algoritmus K-means++, který má pokročilejší vybírání centroidů v začátečním kroku.

1.5.1.1 Elbow metoda

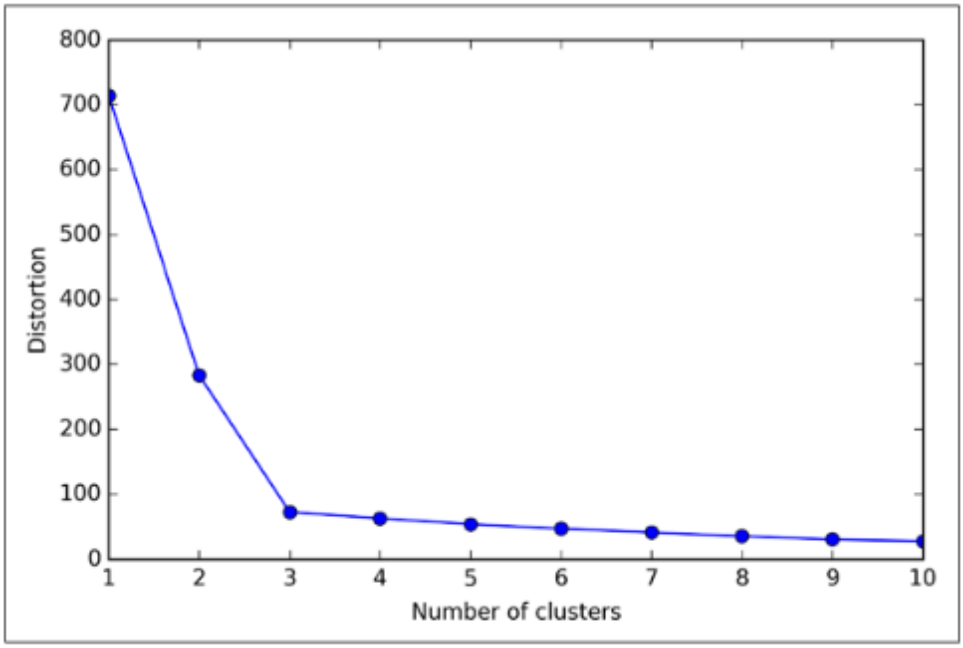
Pro výběr správného K existují metody, které výběr usnadňují [5]. Jednou z metod je elbow metoda, která pracuje s hodnotou $WCSEE$ - suma kvadratických odchylek v jednotlivých shlucích. Tato suma postupně klesá do hodnoty 0 (pokud má každý bod vlastní shluk). Ovšem při jednotlivých iteracích klesá stále méně a lze najít hodnotu (elbow), kde přestává klesat. Na obrázku 1.18. je tato hodnota 3.

1.5.2 Hierarchické

Dalším algoritmem je hierarchické shlukování [5]. Tento algoritmus má výhodu v tom, že není třeba předem definovat počet shluků, a umožnuje zobrazit dendogramy (viz. 1.5.2.1). Algoritmus má dva přístupy - aglomerační a rozdělovací. Algoritmy se liší v tom, že aglomerační začíná s n shluky, kde n je počet bodů a počet shluků snižuje jejich spojováním, zatímco rozdělovací začíná s

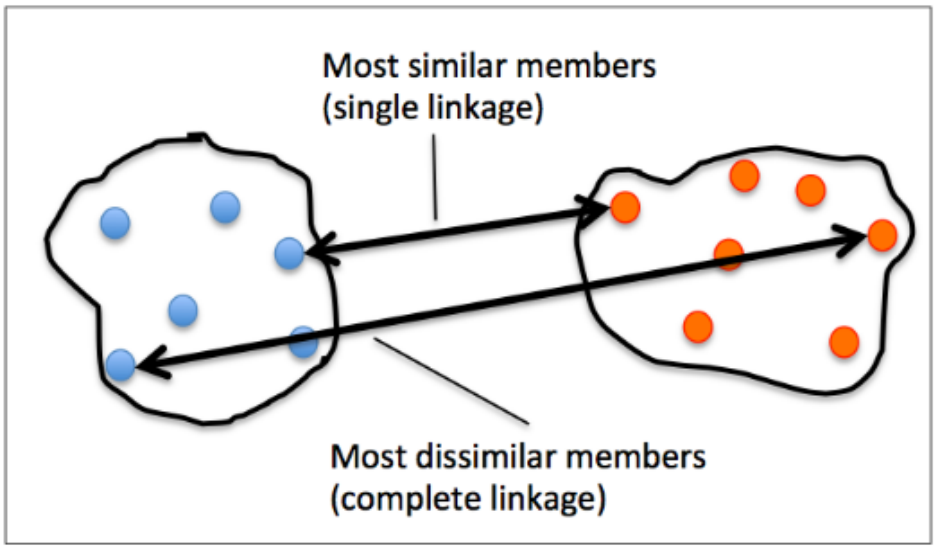


Obrázek 1.17: Ukázka správného řešení K-means.



.

Obrázek 1.18: Ukázka elbow metody, převzato z [5]



Obrázek 1.19: Ukázka různých metod pro měření vzdálenosti shluků, převzato z [5].

jedním shlukem a postupně počet shluků zvyšuje rozdělováním shluků. Tato sekce se bude věnovat pouze aglomeračnímu, který pro pochopení stačí.

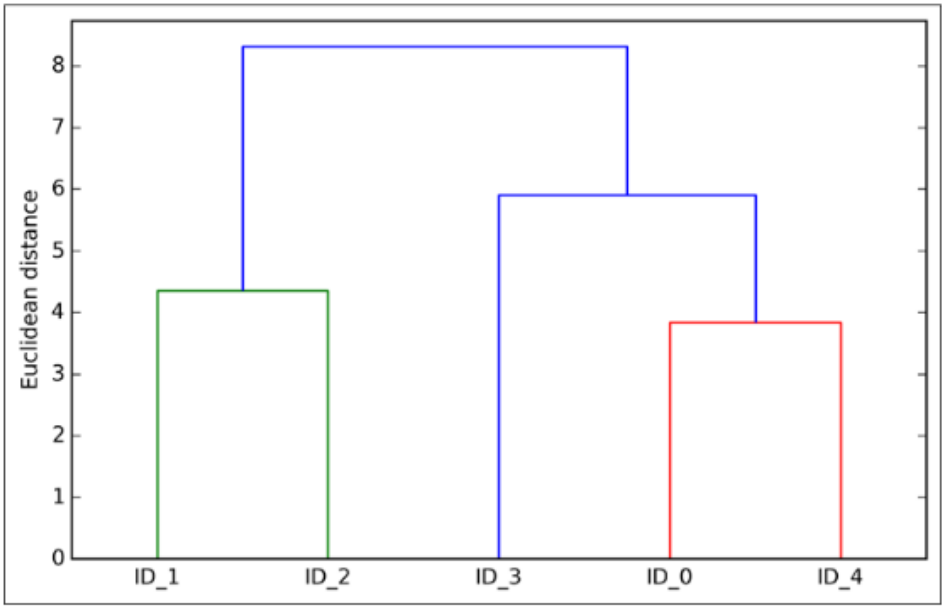
Při aglomeračním algoritmu tedy v každé iteraci hledáme dva nejpodobnější shluky. Podobnost opět počítáme pomocí vzdálenosti, ovšem u shluků je to trochu složitější. Lze počítat různými metodami, nejznámnější je vzdálenost centroidů, nejbližších bodů(single linkage) nebo nejvzdálenějších bodů (complete linkage). Na obrázku 1.19 jsou ukázány dvě nejčastější pro aglomerativní shlukování.

Algoritmus je následující:

1. Spočítej vzájemnou vzdálenost všech bodů.
2. Reprezentuj každý bod jako shluk.
3. Spoj dva nejbližší shluky (podle vybrané vzdálenostní metody).
4. Aktualizuj podobnost shluků.
5. Opakuj body 2 do 4, dokud nezůstane jeden shluk.

1.5.2.1 Dendogramy

K vyhodnocení vhodného počtu shluků slouží dendogramy [5]. Tyto dendogramy ukazují, jak hodně si jsou jednotlivé shluky podobné. Místo počtu shluků se tedy algoritmu zadává maximální limit na podobnost. Pro dendogram z obrázku 1.20 a limit vzdálenosti 5 by to byly 3 shluky (shluk ID_1



Obrázek 1.20: Ukázka dendogramu, převzato z [5].

a ID_2, shluk ID_3, shluk ID_0 a ID_4), pokud bychom zvolili hodnotu 1, byl by počet shluků 5.

1.5.3 Měření výsledků

Shlukování, podobně jako klasifikace, shlukuje data do tříd a výsledky se tedy dají měřit stejně jako u klasifikace. Problém je ale ten, že shlukování si udělá vlastní identifikátory tříd a je potřeba je správně namapovat na třídy testovací množiny. K tomu slouží Jaccardův a Randův index.

1.5.3.1 Jaccardův index

„Jaccardův index je podobnostní míra, známá také jako relativní překryv. Nabývá hodnot mezi 0 a 1 a je dán poměrem počtu stejných prvků dvou množiny vůči celkovému počtu prvků v obou množinách.“ [19]

Pokud máme množinu C_i (množina identifikátorů entit, které byly klasifikovány do třídy i ze shlukování) a T_i (množina identifikátorů entit, které byly klasifikovány do třídy i z testovacích dat), pak

$$J(C_i, T_i) = \frac{\mid C_i \cap T_i \mid}{\mid C_i \cup T_i \mid}$$

je Jaccardovým indexem.

1.5.3.2 Randův index

„Jestliže Jaccardův index lze chápat jako míru relativního překryvu, pak Randův index je míra absolutního překryvu. Nabývá hodnot mezi 0 a 1 a je dán poměrem počtu stejných prvků dvou podmnožin nějakého celku vůči celkovému počtu prvků v tomto celku.“[19]

Pokud máme množinu C_i (množina identifikátorů entit, které byly klasifikovány do třídy i ze shlukování) a T_i (množina identifikátorů entit, které byly klasifikovány do třídy i z testovacích dat), které jsou podmnožinou X (množina identifikátorů všech entit), pak

$$R(C_i, T_i) = \frac{\mid C_i \cap T_i \mid}{\mid X \mid}$$

je Randovým indexem.

1.6 Deep learning

Deep learning je jedním z oborů strojového učení inspirovaný strukturou a funkcí mozku zvanou umělá neuronová síť [8]. Neuronové sítě zažívají v posledních letech velký převrat a to hlavně díky tomu, že jsou počítače čím dál výkonější a objem dat pro učení neuronových sítí se také neustále zvyšuje.

Oproti jiným metodám má jednu velkou výhodu a to takovou, že deep learning je škálovatelný - počtem dat (čím více má neuronová síť trénovaćich dat, tím se zlepšuje její výkon) a výpočetní silou (pro hledání správných parametrů složitých struktur).

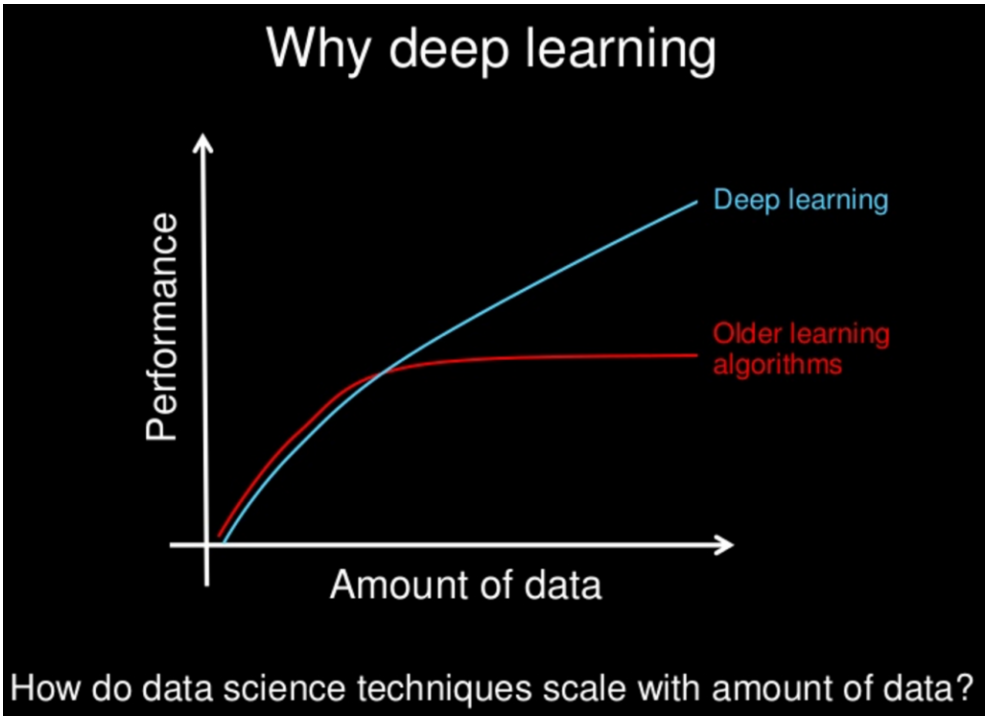
1.6.1 Neuronové sítě

Neuronová síť je velmi silný mechanismus strojového učení, který napodobuje chování lidského mozku [9]. Mozek získá podnět z vnějšího světa, zprocesuje vstup a vygeneruje výstup. Při složitějších úkolech vytvářejí neurony komplexní síť, ve které si mezi sebou posílají informace.

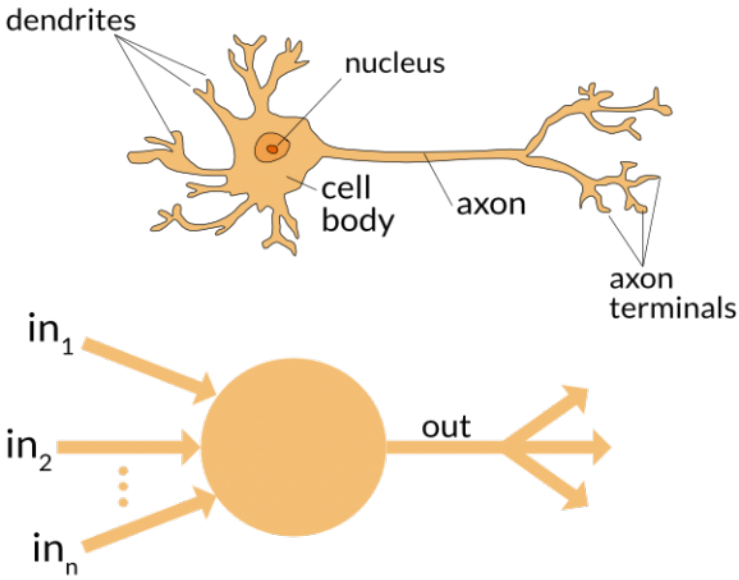
1.6.2 Umělá neuronová síť

Při tvoření neuronové sítě se snažíme napodobit podobné chování jako v lidské neuronové síti.

Na obrázku 1.23 je příklad umělé neuronové sítě. Každý černý kruh je neuron a každý neuron je charakterizován svojí váhou, aktivační funkcí. Vstupní data jsou vložena do první vrstvy, která se nazývá vstupní. Tato vstupní vrstva dále lineárně transformuje data pomocí aktivační funkce a svých vah. Dále se transformují data nelineárně pomocí aktivační funkce. Po transformaci se přesouvají do skryté vrstvy, kde se celý proces opakuje tolikrát, dokud se data nedostanou do výstupní vrstvy. Tento proces se nazývá forward-propagation.

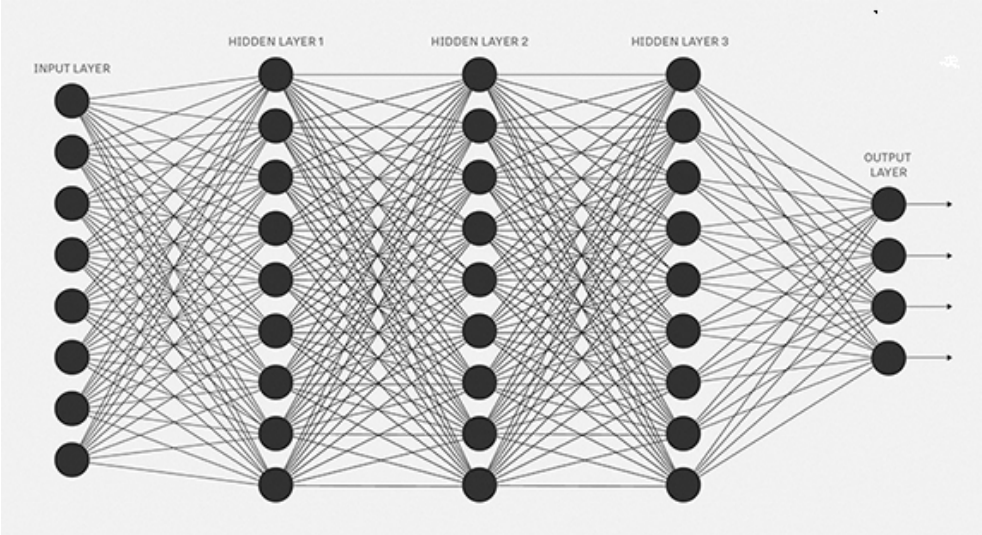


Obrázek 1.21: Ukázka vztahu na výkon a počet dat, převzato z [8].



Obrázek 1.22: Ukázka neuronu, převzato z [9].

Neuronová síť se učí pomocí takzvaných epoch. Tato epocha slouží k nalezení optimálních vah a v každé epoše se ohodnotí všechna trénovací data pomocí této neuronové sítě s vybranými váhami. V každé epoše se zkouší jiné váhy. U složitých neuronových sítí je kombinace možností vah mnoho a proto je potřeba výběr vah optimalizovat pomocí procesu back-propagation..

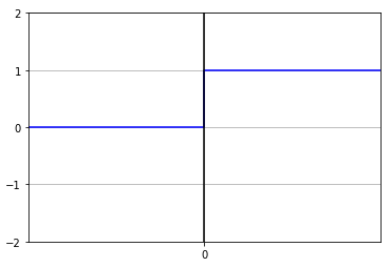


Obrázek 1.23: Ukázka umělé neuronové sítě, převzato z [9].

1.6.2.1 Aktivační funkce

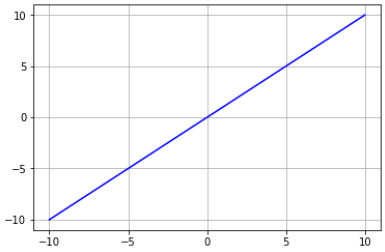
Aktivační funkce rozhoduje o tom, jestli má být neuron aktivovaný nebo ne. Aktivační funkce dostává jako vstup sumu vah, vstupů a nákloností a nelineárně je transformuje. Tyto transformované hodnoty jsou následně poslány do dalších neuronů jako vstup. Bez aktivační funkce by umělá neuronová síť byla pouze další lineárně regresní model.

Heavisideova funkce je nejjednodušší aktivační funkci jednoduše vrací 0, pokud je $x < 0$ a 1, pokud $x >= 0$.



Obrázek 1.24: Heavisideova funkce.

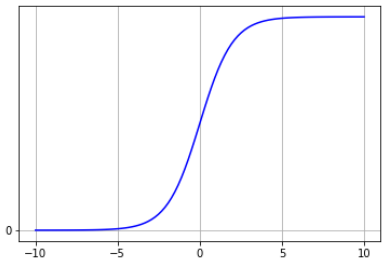
Lineární funkce je další možnou aktivační funkcí. Je jednoduchá, lehká na pochopení a použití.



Obrázek 1.25: Lineární funkce.

Sigmoida je velmi používaná jako aktivační funkce. Už o ní byla zmíňka v kapitole 1.4.1. Problémem je, že obor hodnoty sigmoidy je pouze mezi 0 a 1. Je definována jako:

$$f(x) = \frac{1}{1 + e^{-x}}$$



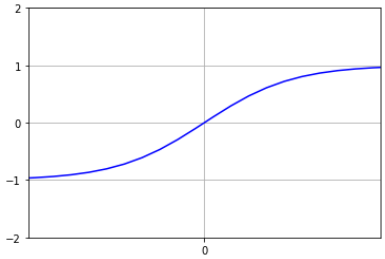
Obrázek 1.26: Sigmoida.

Tanh je velmi podobná sigmoidě. Je to pouze škálovaná verze sigmoid funkce. Obor hodnot je od -1 do 1. Je definována jako:

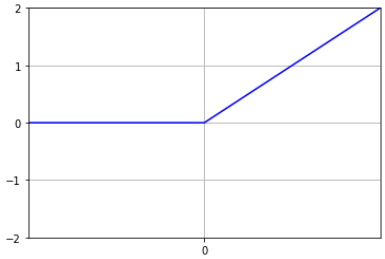
$$tanh(x) = 2sigmoid(2x) - 1$$

ReLU (rectified linear unit) je nejvíce používaná aktivační funkce pro neuronové sítě. Tato funkce není lineární, což podporuje back-propagation. Hlavní výhodou této funkce je, že nemusí aktivovat všechny neurony. Pokud má hodnoty menší než 0, tak se neurony vůbec neaktivují. Je definována jako:

$$f(x) = max(0, x)$$



Obrázek 1.27: Tanh funkce.



Obrázek 1.28: ReLU funkce.

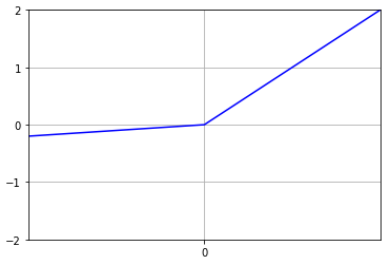
Leaky ReLU je vylepšená verze ReLU funkce. Na rozdíl od ReLU funkce vrací nenulové hodnoty pro $x < 0$, její předpis je:

$$f(x) = x$$

pro $x >= 0$ a

$$f(x) = ax$$

pro $x < 0$, kde a je velmi malá hodnota (např. 0.01).



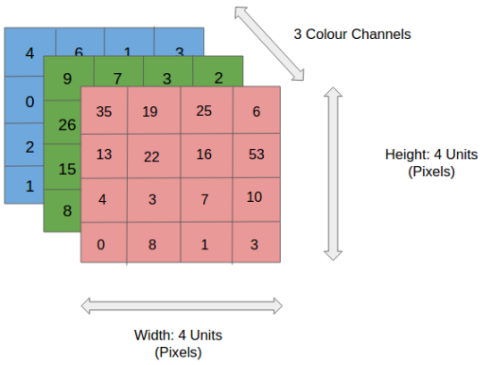
Obrázek 1.29: Leaky ReLU funkce.

Softmax funkce je dalším typem sigmoidy, ale hodí se při klasifikačních problémech. Klasická sigmoida si dokáže poradit pouze s dvěmi třidami. Softmax funkce dokáže klasifikovat n tříd. Softmax funkce se používá ve výstupní vrstvě pro zjištění, s jakou pravděpodobností patří vstup do třídy.

1.6.3 Konvoluční neuronová síť

Konvoluční neuronová síť je algoritmus, který má na vstupu obrázek, rozezná aspekty a objekty na obrázku a každému z nich přiřadí důležitost [10]. Na rozdíl od ostatních algoritmů sloužících k rozpoznání obrazu má konvoluční neuronová síť malé požadavky na předzpracování. Hlavní výhodou je, že CNN umí najít charakteristiky obrazu. Architektura CNN byla inspirovaná opět funkcí lidského mozku, konkrétně tou částí, která se stará o vizuální zpracování.

Každý obraz se dá převést do matice, která obsahuje intezitu jednotlivých barev. Černobílý obrázek lze tedy reprezentovat jako jednu matici o rozměrech rozlišení. U RGB obrázků jsou tyto matice potřeba 3. Zpracování se skládá z několika vrstev.

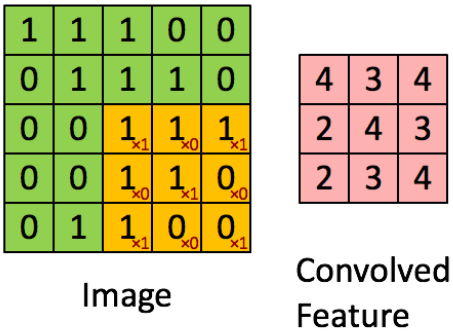


Obrázek 1.30: Příklad 4x4 RGB obrázku reprezentovaného maticí, převzato z [10].

1.6.3.1 Konvoluční vrstva

Konvoluční vrstva je první vrstva předzpracování. V této vrstvě se algoritmus snaží najít vlastnosti. Jako vstup je obraz, ve kterém se hledají vlastnosti pomocí násobením podmatic s filtrem (matice, nejčastěji se používá rozměr 3x3). Jako výstup je matice konvolovaných vlastností. Na obrázku 1.31 je zelenou barvou vyznačená reprezentace vstupního obrazu, žlutou barvou (s červenými čisly) filtr a červenou barvou matice konvolovaných vlastností.

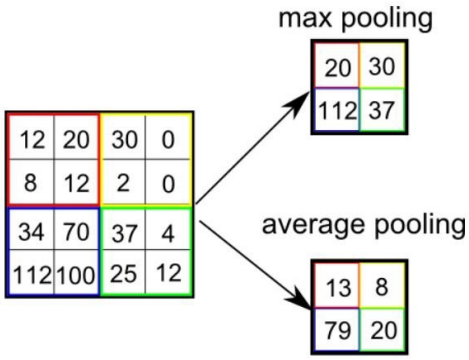
V případě, že máme víc vstupních matic (u RGB 3), je počet matic konvolovaných vlastností třikrát větší. Každá matice konvolovaných vlastností reprezentuje pouze výskyt jedné specifické vlastnosti (např. psí čumák). Proto je potřeba mít několik různých konvolučních matic (různé filtry). Konvolučních vrstev může být více za sebou.



Obrázek 1.31: Příklad konvoluční vrstvy, převzato z [10].

1.6.3.2 Sdružovací vrstva

Sdružovací vrstva slouží, podobně jako konvoluční vrstva, k redukci velikosti matice, což snižuje výpočetní dobu pro zpracování dat. Kromě toho také slouží k extrakci dominantních vlastností. Ke sdružování jsou nejčastěji používány dva typy sdružování - sdružování podle největší hodnoty a sdružování podle průměrné hodnoty. Rozdíl mezi nimi je znázorněn na obrázku 1.32.



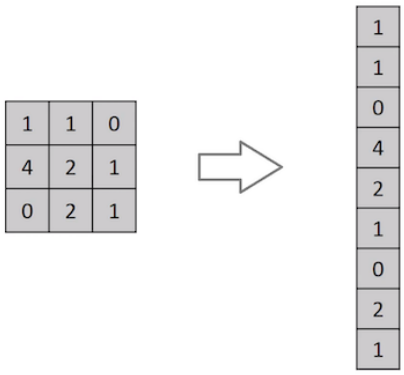
Obrázek 1.32: Příklad typů sdružování u sdružovací vrstvy, převzato z [10].

1.6.3.3 Zploštění (angl. Flattening)

Následný krok slouží k zploštění 2D matice do 1D pole. Příklad je ukázán na obrázku 1.33.

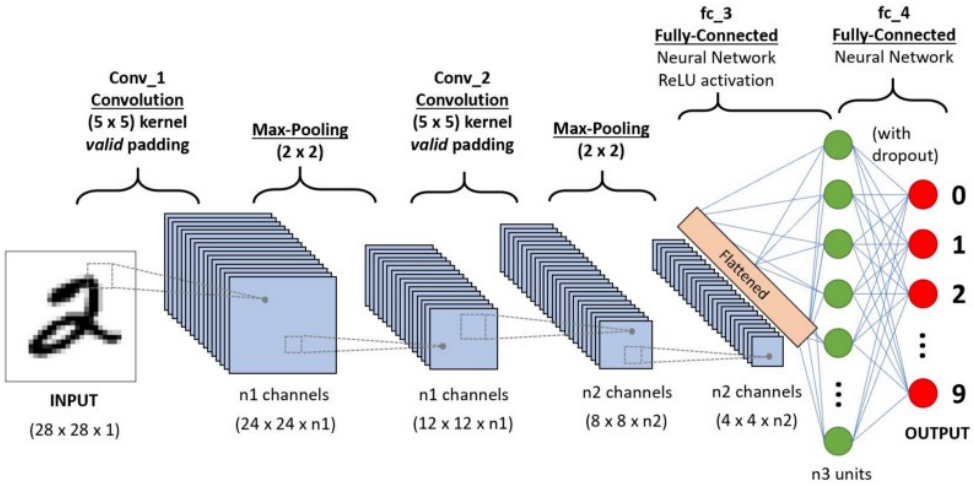
1.6.3.4 Plně spojená vrstva

Poslední krok je napojení zploštěného pole na plně propojené ANN. Od klasického ANN se liší v tom, že každý neuron v jedné vrstvě má stejný počet



Obrázek 1.33: Příklad zploštění, převzato z [10].

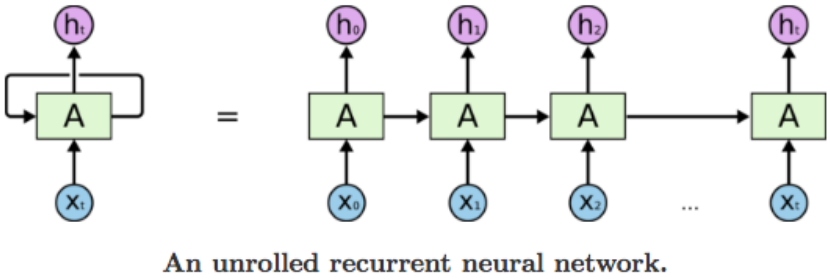
vstupů a výstupů. Celá CNN včetně plně spojené vrstvy je vidět na obrázku 1.34.



Obrázek 1.34: Příklad CNN ke klasifikaci ručně napsaných čísel, převzato z [10].

1.6.4 Rekurentní neuronové sítě

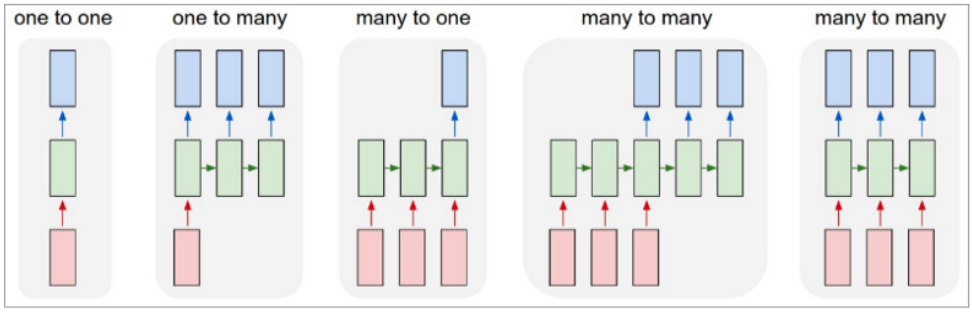
Při rozhodování a rozpoznávání lidé málokdy rozpoznávají události bez kontextu [11]. Pokud bychom například chtěli klasifikovat události ve filmu, je dobré vědět, co se stalo v celém filmu před danou událostí. S tím mají umělé neuronové sítě, na rozdíl od rekurentních, problém. Rekurentní sítě se zabývají právě tímto problémem. Jejich rozhodování se tedy odvíjí na základě událostí v čase. Příklad je ukázán na obrázku 1.35.



Obrázek 1.35: Příklad rekurentní neuronové sítě převzato z [11].

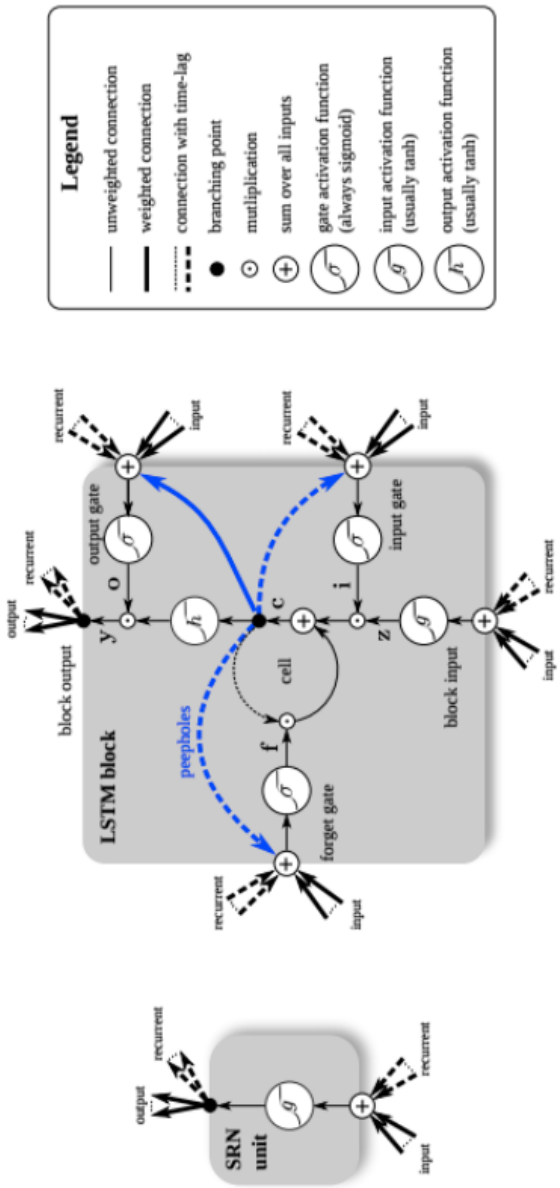
RNN může mít mnoho podob vstupů a výstupů. Nejčastější podoby jsou ukázány na obrázku 1.36. Typy vstupů a výstupů mohou reprezentovat:

1. Z fixované délky vstupu do fixované délky výstupu (klasifikace obrazu).
2. Z fixované délky vstupu do sekvence výstupů (popis obrazu do věty).
3. Z sekvence vstupů do fixované délky výstupu (analýza nálady člověka podle slov ve větě).
4. Z sekvence vstupů do sekvence výstupů jiné délky (překlad jazyka).
5. Z sekvence vstupů do sekvence výstupů stejné délky (klasifikace videa).



Obrázek 1.36: Příklad rekurentní neuronové sítě převzato z [11].

Hlavní rozdíl mezi ostatními neuronovými sítěmi je tedy v skrytých vrstvách - v RNN se používá stav, který je závislý na předchozích událostech. Na výpočet těchto stavů se používají různé funkce. Nejčastěji používaná je LSTM (Long-short-term memory). Její princip je ukázán na obrázku 1.37.



Obrázek 1.37: Princip fungování LSTM ukázaný na jednoduché rekurentní síti, převzato z [12].

1.7 Python

Python je interpretovaný, objektově-orientovaný, high-level programovací jazyk s dynamickou semantikou [20]. Má mnoho knihoven zabývajících se problematikou strojového učení a deep learningu, a proto je vhodnou volbou pro tuto problematiku. Následující sekce se věnuje nejpoužívanějším knihovnám v tomto odvětví.

1.7.1 Scikit-learn

Nejrozsáhlejší knihovna v Pythonu pro strojové učení [21]. Obsahuje implementaci většiny algoritmů strojového učení (regrese, klasifikace, shlukování), včetně předzpracování dat, redukce dimenze a metody pro ladění parametrů. Na obrázku 1.38 je ukázaná funkce, která rozdělí data na testovací a trénovací, vytvoří SVM klasifikátor, vytvoří matici záměn a vyhodnotí přesnost pomocí metody křížové validace.

```
1  def classification_svm(X, y):
2      #splitting into test and train sets
3      from sklearn.model_selection import train_test_split
4      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.10, random_state = 0)
5      # SVM
6      from sklearn.svm import SVC
7      classifier = SVC(kernel = 'linear')
8      classifier.fit(X_train, y_train)
9      # Predicting results for test
10     y_pred = classifier.predict(X_test)
11     # Confusion Matrix
12     from sklearn.metrics import confusion_matrix
13     cm = confusion_matrix(y_test, y_pred)   # 3 mistakes
14     #k-fold cross validation
15     from sklearn.model_selection import cross_val_score
16     accuracies = cross_val_score(estimator = classifier, X= X_train, y = y_train, cv = 10)
17     accuracies.mean()
```

Obrázek 1.38: Ukázka využití scikit-learn.

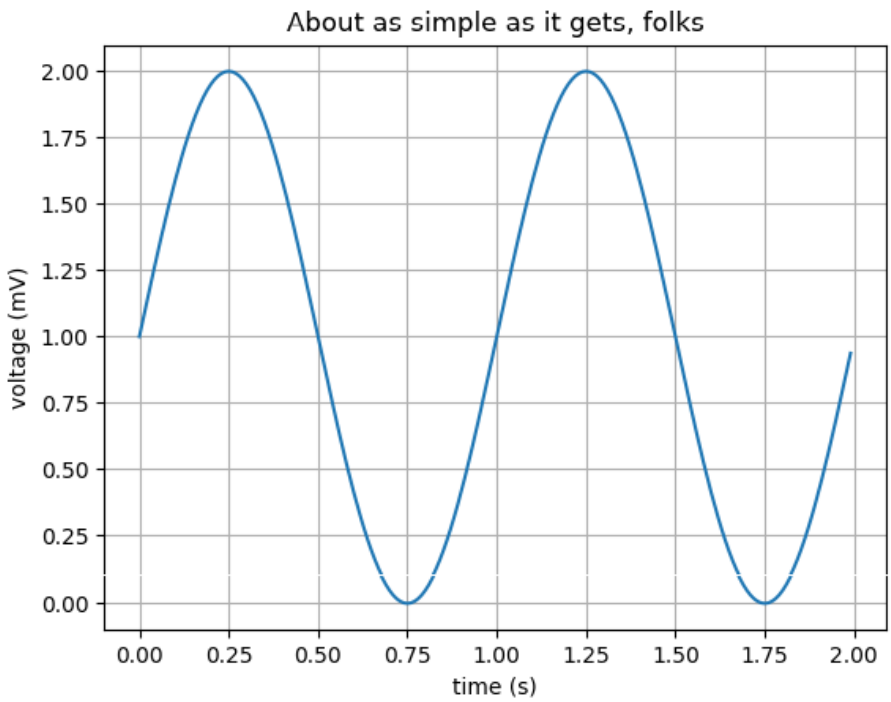
1.7.2 Pandas

Pandas je open source knihovna, která umožňuje s vysokým výkonem práci s datovými strukturami a datovou analýzu v Pythonu. Nejvíce používané funkčnosti jsou:

- rychlý a účinný DataFrame objekt, sloužící k manipulaci s daty,
- nástroje pro čtení a zápis mezi daty v paměti a různými formáty (CSV, SQL databáze, atd.),
- inteligentní sladění dat a integrované zacházení s chybějícími daty,
- vysoce výkonné slučování datových množin,
- vysoce optimalizované pro výkon.

1.7.3 Matplotlib

Knihovna v Pythonu sloužící pro vytváření 2D diagramů. Umožňuje generování grafů, histogramů a mnohých dalších. Na obrázku 1.39 je ukázán graf vygenerovaný pomocí matplotlib.



Obrázek 1.39: Ukázka jednoduchého matplotlib grafu, převzato z [13].

1.7.4 NumPy

Základní knihovna pro vědecké výpočty [22]. Nejpoužívanější funkcionality jsou:

- mocné N-dimenzionální pole objektů,
- sofistikované funkce,
- užitečná lineární algebra, Fourierova transformace a práce s náhodnými čísly.

1.7.5 TensorFlow

„TensorFlow^{TM} je open source softwarová knihovna, sloužící k vysoce výkonným numerickým výpočtům. Její flexibilní architektura dovoluje jednoduché

nasazení výpočtů přes různé platformy (CPU,GPU,TPU) a od stolních počítačů přes shluky serverů až k mobilním zařízením. Originálně vytvořená vědci a inženýry z Google Brain teamu, patřící pod Google AI organizaci. Obsahuje velkou podporu pro machine learning, deep learning a díky flexibilnímu výpočetnímu jádru je používána skrze mnoha vědecké domény.“ [23]

1.7.6 Keras

API pro práci s neuronovými sítěmi napsané v Pythonu. Běží nad TensorFlow(Případně CNTK, Theano)[24]. Byl vyvinut za účelem rychlého experimetování, aby bylo možné přejít z nápadu k výsledku co nejrychleji. Keras umožňuje:

- dovoluje jednoduché a rychlé prototypování,
- podporuje ANN, CNN i RNN,
- umožňuje běh na CPU i GPU.

Na obrázku 1.40 je vidět použití keras - vytvoření ANN se třemi skrytými vrstvami.

```
def create_ann():
    from keras.models import Sequential
    from keras.layers import Dense
    # Initialising the ANN
    classifier = Sequential()
    # Adding the input layer and the first hidden layer
    classifier.add(Dense(output_dim = 50, init = 'uniform', activation = 'relu', input_dim = 512))
    # Adding the second hidden layer
    classifier.add(Dense(output_dim = 15, init = 'uniform', activation = 'relu'))
    # Adding the third hidden layer
    classifier.add(Dense(output_dim = 10, init = 'uniform', activation = 'relu'))
    # Adding the output layer
    classifier.add(Dense(output_dim = 5, init = 'uniform', activation = 'softmax'))
    # Compiling ANN
    classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
    return classifier
```

Obrázek 1.40: Ukázka použití Keras.

Kapitola 2

Analýza

Signál z vláken optických mikrofonů vykazuje jiné příznaky než běžný zvukový signál. To umožňuje nové možnosti zpracování strojovým učením. Zpracování signálů se liší od standardního zpracování klasických signálů. Je tedy potřeba prozkoumat chování signálů a experimentovat s klasifikačními metodami pro rozpoznání typu signálu. Na tuto problematiku je užitečné vyvinout speciální aplikaci, která odráží potřebý na optické straně (nestaciárnost a jiné speciálně dané vlastnosti). Pro další experimentování s jinými vstupními daty je vhodné mít uživatelské rozhraní.

2.1 Vstup

Vstupní data do aplikace bude složka obsahující CSV soubory s předzpracovanými signály jednotlivých událostí. Pro každý typ události bude existovat příslušný soubor. Ukázka obsahu tohoto souboru je vidět na obrázku 2.1 Pokud u dat nebude známý typ, tak bude složka obsahovat pouze jeden soubor. Typy událostí se dají změnit i po nahrání souboru. CSV soubor bude obsahovat následující sloupce:

1. rIndex - index řádku,
2. ID - identifikátor události,
3. timeStr - časová značka signálu,
4. timeOff - čas v mikrosekundách vůči začátku sběru dat,
5. rawValue - hodnota signálu,
6. diffValue - rozdíl hodnoty signálu od předchozího řádku,
7. trendLevel - úroveň trendu.

rIndex	ID	timeStr	timeOff	rawValue	diffValue	trendLevel
136259	0	15-04-2019 02:55:59.496…	8719474	2.19184e+06	1520	-3740
136260	0	15-04-2019 02:55:59.496…	8719538	2.19591e+06	4070	-5910
136261	0	15-04-2019 02:55:59.496…	8719602	2.19971e+06	3800	-5910
136262	0	15-04-2019 02:55:59.496…	8719667	2.20002e+06	310	-5910
136263	0	15-04-2019 02:55:59.496…	8719731	2.20097e+06	950	-5910
136264	0	15-04-2019 02:55:59.496…	8719795	2.20272e+06	1750	-5910
136265	0	15-04-2019 02:55:59.496…	8719859	2.20271e+06	-10	-5910
136266	0	15-04-2019 02:55:59.496…	8719923	2.20656e+06	3850	-5910
136267	0	15-04-2019 02:55:59.496…	8719987	2.20927e+06	2710	-5910
136268	0	15-04-2019 02:55:59.496…	8720051	2.20684e+06	-2430	-5910
136269	0	15-04-2019 02:55:59.497…	8720115	2.20093e+06	-5910	-5910
136270	0	15-04-2019 02:55:59.497…	8720178	2.20022e+06	-710	-5910
136271	0	15-04-2019 02:55:59.497…	8720242	2.19993e+06	-290	-5910
136272	0	15-04-2019 02:55:59.497…	8720306	2.20062e+06	690	-5910

Obrázek 2.1: Ukázka struktury vstupních dat.

2.2 Výstup

Aplikace bude mít 2 funkčnosti a tedy i typy výstupů - klasifikované události a lokalizaci.

2.2.1 Klasifikované události

Výstupem aplikace bude klasifikace jednotlivých událostí včetně statistik, sloužících pro pororovnání algoritmů.

2.2.2 Lokalizace

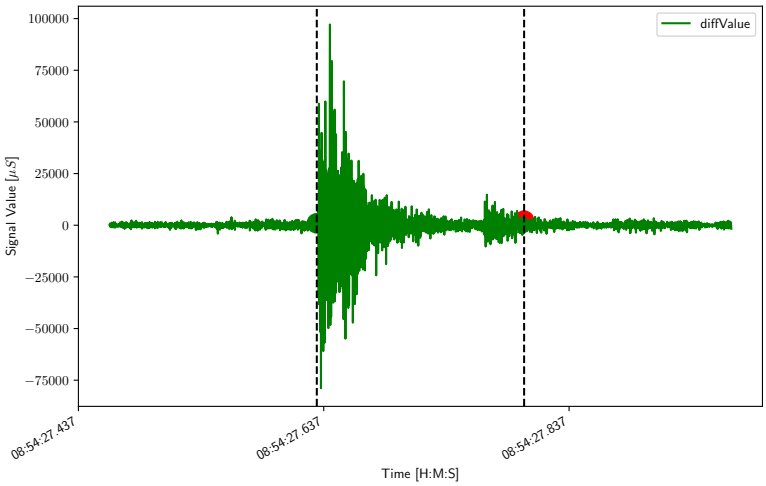
Dalším možným výstupem budou souřadnice události. Uživatel vybere časový úsek, pro který se mají brát všechny události jedné třídy jako stejná událost z jiných zdrojů. Každý zdroj bude mít souřadnice, podle kterých bude možné událost lokalizovat.

2.3 Typy událostí

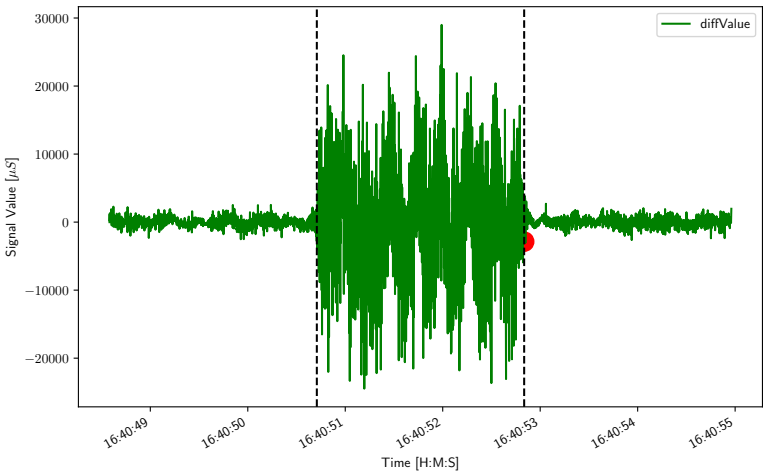
Aplikace bude pracovat s libovolným počtem událostí. Základní vstupní události pro vývoj jsou tyto typy událostí:

- pád kuličky na koš,

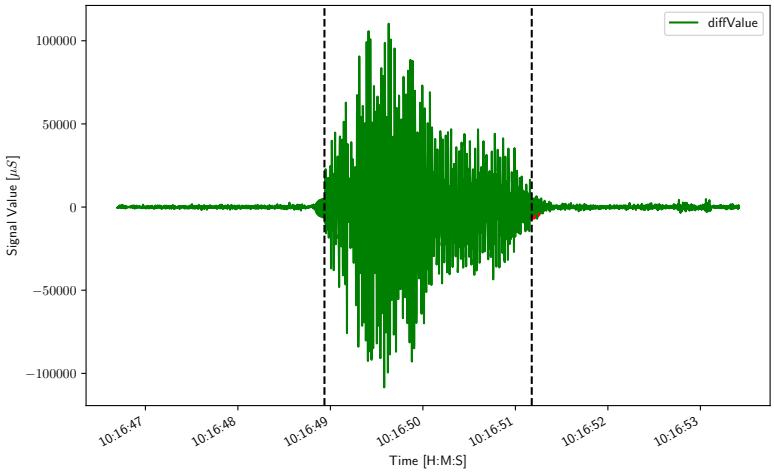
- vrtání vrtačkou,
- spuštení kompresoru,
- mluvení.



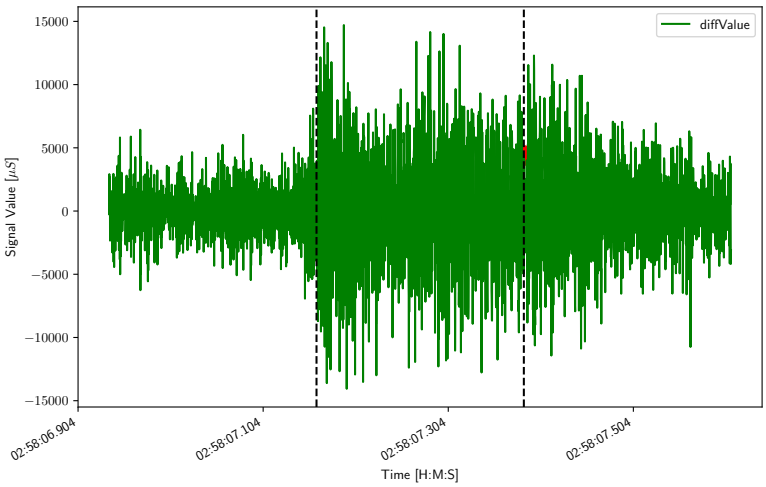
Obrázek 2.2: Ukázka signálu pádu kuličky na koš.



Obrázek 2.3: Ukázka signálu vrtání vrtačkou.



Obrázek 2.4: Ukázka signálu spuštění kompresoru.



Obrázek 2.5: Ukázka signálu mluvení.

2.4 Rozložení mikrofonní sítě

V některých řešeních bude události detekovat hned několik mikrofonů, což umožní lokalizaci. Rozložení mikrofonů se v zádání projektu předpokládá v jedné ose, jak je vidět na obrázku 2.6.

Obrázek 2.6: Rozložení mikrofonní sítě.

2.5 Požadavky

Následující sekce se věnuje funkčním a nefunkčním požadavkům na aplikaci.

2.5.1 Funkční požadavky

F_1 **Změna metody klasifikace**

Je žádoucí umožnit uživateli změnit metodu klasifikace, pomocí které se budou třídy klasifikovat.

F_2 **Změna trénovací množiny**

Aplikace se bude používat v různých prostředích, kde budou nastávat jiné události. Je tedy nutné uživateli umožnit vkládat trénovací množiny.

F_3 **Výběr testovací množiny**

Vytvořené klasifikátory bude možno použít opakovaně na klasifikování ostatních neoklasifikovaných množin.

F_4 **Redukce dimenze**

Bude možné redukovat dimenze u testovacích a trénovacích množin.

F_5 **Změny typu události**

Bude umožněno měnit načteným událostem očekávané výsledky.

F_6 **Změny názvu třídy**

Bude umožněno měnit názvy typů událostí.

F_7 **Zobrazení klasifikačních vlastností**

Je potřeba zobrazit základní údaje o klasifikaci.

F_8 **Určení pozice události**

Pokud bude možné, je potřeba určit souřadnice události v linéarním prostoru.

F_9 **Uložení dat**

Data budou uložena v databázovém úložišti pro jednoduché dohledávání.

F_{10} **Zobrazení událostí v grafu**

Události budou možné zobrazit také v grafu.

2.5.2 Nefunkční požadavky

N_1 **Možnost spuštění na Linux i Windows**

Všechny vytvořené aplikace je potřeba umožnit spustit na obou operačních systémech.

N_2 **Rozšírení metod klasifikace**

Bude jednoduché přidat metodu klasifikace. Jak přidat metodu bude popsáno.

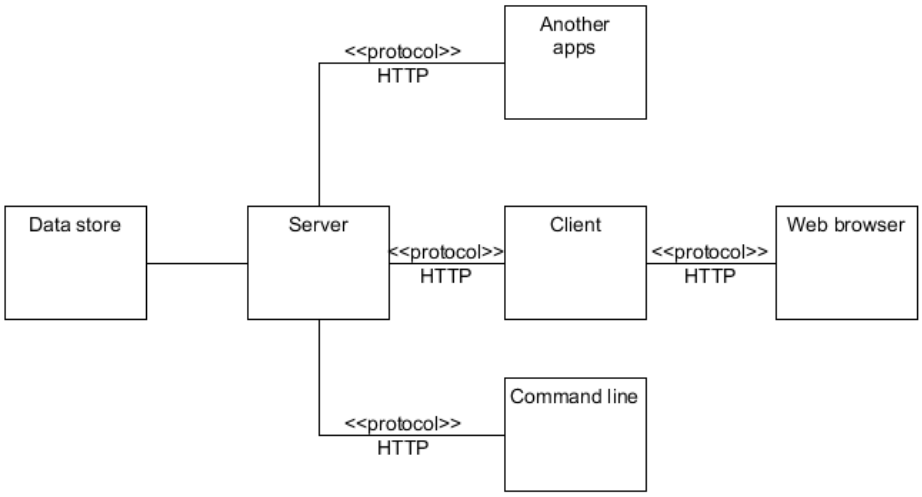
N_3 **Modul pro klasifikaci bude stejný pro CLI i GUI**

Klasifikace bude prováděna stejnými metodami v obou typech uživatelského rozhraní.

Kapitola 3

Návrh

Následující část se věnuje návrhu aplikace. Architektura aplikace je klient-server, jak je vidět na obrázku 3.1. V této kapitole jsou použité poslední verze návrhů. Všechny verze jsou na přiloženém médiu.



Obrázek 3.1: Architektura aplikace.

3.1 Server

Serverová část aplikace má tři samostatně funkční celky - trénování 3.1.1, testování 3.1.2 a lokalizaci 3.1.3. Serverová část bude vystavovat HTTP metody, které budou volány klientskou částí aplikace.

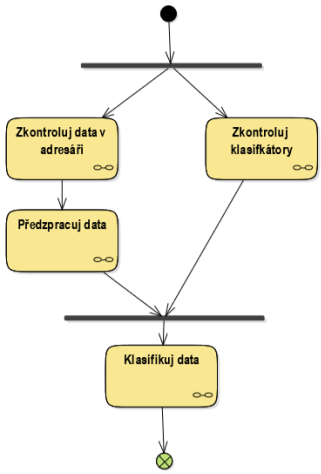
3.1.1 Životní cyklus trénování

Tento funkční celek se věnuje vytvoření klasifikátoru na základě vstupních ohodnocených dat, na kterých se klasifikátor trénuje. Pro vytvoření klasifikátoru je potřeba dodržet posloupnost popsanou v aktivity diagramu 3.2. Nejprve je tedy potřeba načíst vstupní data, předzpracovat je, a až poté lze data trénovat a vytvořit klasifikátor. Jednotlivé aktivity jsou popsány níže v sekcích kontrola dat v adresáři 3.1.5, předzpracování dat 3.1.6 a trénování klasifikace 3.1.7. Jednotlivé aktivity lze opakovat - na stejných předzpracovaných datech lze natrénovat neomezený počet klasifikací. Stejně tak lze předzpracovat stejná data několikrát (například s jinou redukcí dimenze).

Obrázek 3.2: Aktivity diagram životního cyklu trénování klasifikace.

3.1.2 Životní cyklus testování

Tento funkční celek se věnuje ohodnocení vstupních dat pomocí klasifikátoru, který byl vytvořen pomocí procesu trénování klasifikace 3.1.1 nebo importem klasifikátoru. Pro vytvoření klasifikátoru je potřeba dodržet posloupnost popsanou v aktivity diagramu 3.3. Nejprve je tedy potřeba načíst klasifikátor 3.1.4, vstupní data 3.1.5, předzpracovat 3.1.6 je a až poté lze data ohodnotit příslušnými třídami. Stejně jako v trénování 3.1.1 je možné aktivity neomezeně opakovat, pokud jim předcházely příslušné aktivity - lze tedy pomocí jednoho načteného klasifikátoru otestovat více typů dat. Stejně tak lze jedna data otestovat několika klasifikátory.



Obrázek 3.3: Aktivity diagram životního cyklu testování klasifikace.

3.1.3 Lokalizace

Tento funkční celek se stará o lokalizaci událostí, pokud mikrofonní síť obsahuje alespoň 2 mikrofony. Lokalizace se vždy počítá pouze pomocí 2 mikrofonů, a pokud je v síti více mikrofonů, které událost zaznamenaly, tak se pro každou dvojici odhadne pozice události zvlášt, a poté se finální pozice vyhodnotí pomocí průměru těchto pozic. Vzhledem ke vstupním parametrům a rozložení mikrofonů lze pozice vypočítat pouze tehdy, když událost vznikla mezi mikrofony.

Pro výpočet známe souřadnice mikrofonů (vzdálenost jednoho od druhého) a rozdíl v čase, v kterém událost detekovaly, lze tedy využít následující vzorec:

$$s = v_1 t_1 + v_2 t_2$$

kde s je vzdálenost mikrofonů, v_1, v_2 rychlost zvuku k prvnímu, resp. druhému mikrofonu a t_1, t_2 doba, za kterou se zvuk dostal k prvnímu, resp. druhému mikrofonu. Pro řešení bude počítáno s tím, že $v_1 = v_2$, tedy:

$$s = v(t_1 + t_2)$$

t_2 lze následně vyjádřit jako $t_1 + \Delta$, můžeme tedy zapsat:

$$s = 2vt_1 + v\Delta$$

pomocí tohoto vzorce můžeme tedy vypočítat neznámou t_1:

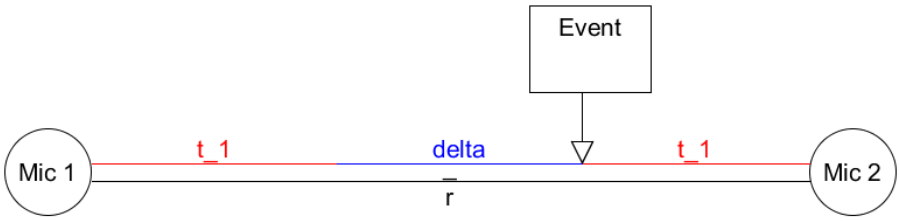
$$t_1 = \frac{(s - v\Delta)}{2v}$$

s vypočítanou $t1$ už pouze vypočítáme vzdálenost:

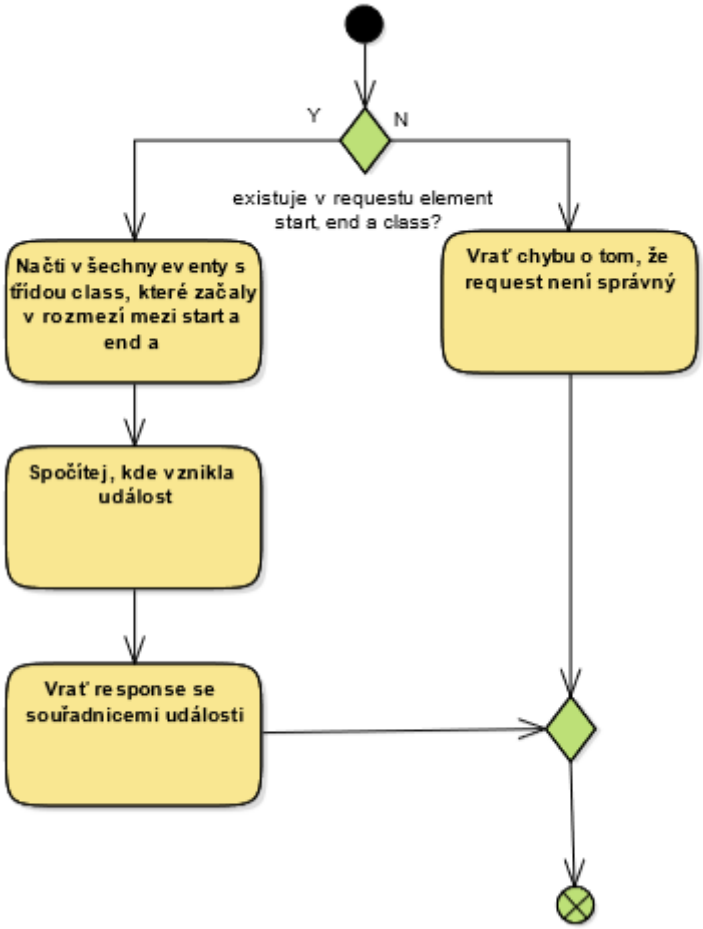
$$s_1 = t_1 v_1$$

,

kde s_1 je vzdálenost od prvního mikrofonu.



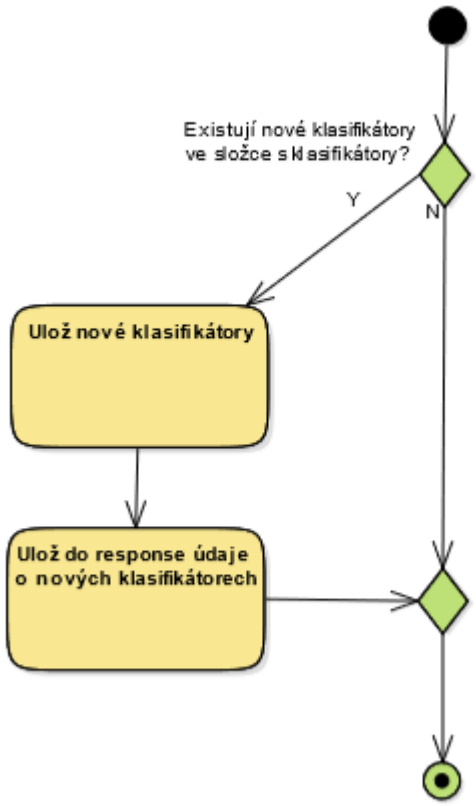
Obrázek 3.4: Grafické znázornění výpočtu.



Obrázek 3.5: Aktivity diagram lokalizace.

3.1.4 Kontrola klasifikátorů

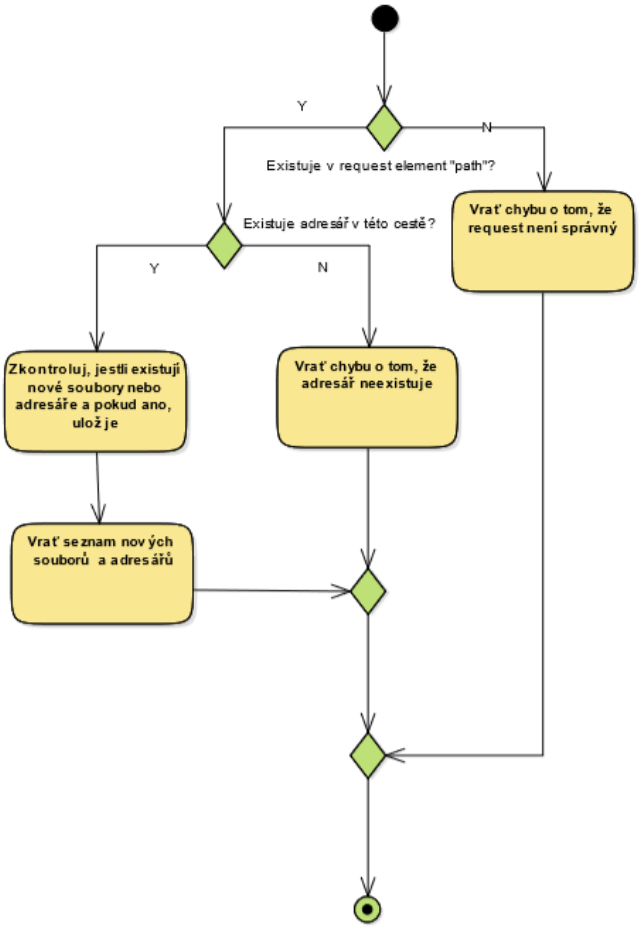
Tato aktivita slouží k identifikaci nových klasifikátorů. Jak je vidět podle aktivity diagramu 3.6, není vůbec složitá. Jediné, co dělá, je, že se podívá do složky z klasifikátory, jestli nepřibyly nové klasifikátory, a pokud ano, tak je uloží do databáze a vrátí je v response.



Obrázek 3.6: Aktivity diagram kontroly klasifikátorů.

3.1.5 Kontrola dat v adresáři

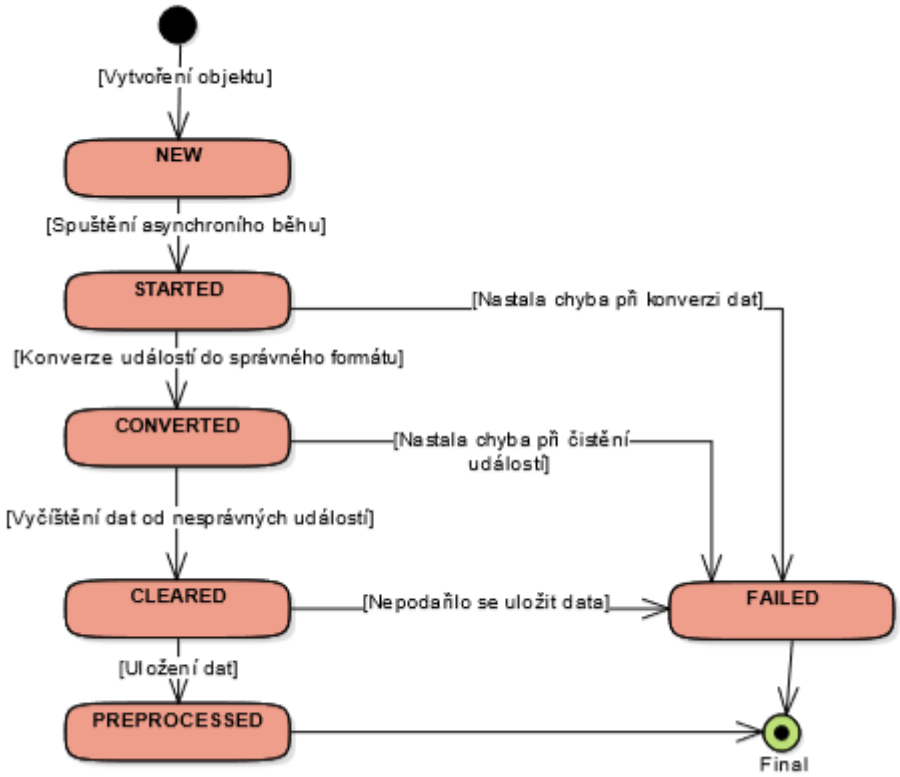
Tato aktivita je velice podobná kontrole klasifikátorů 3.1.4. Slouží k identifikaci nových vstupních dat. Podobně jako v kontrole klasifikátorů kontroluje nová data, ale kontroluje je v adresáři, který dostane v elementu path, a je potřeba kontrolovat validní vstupy.



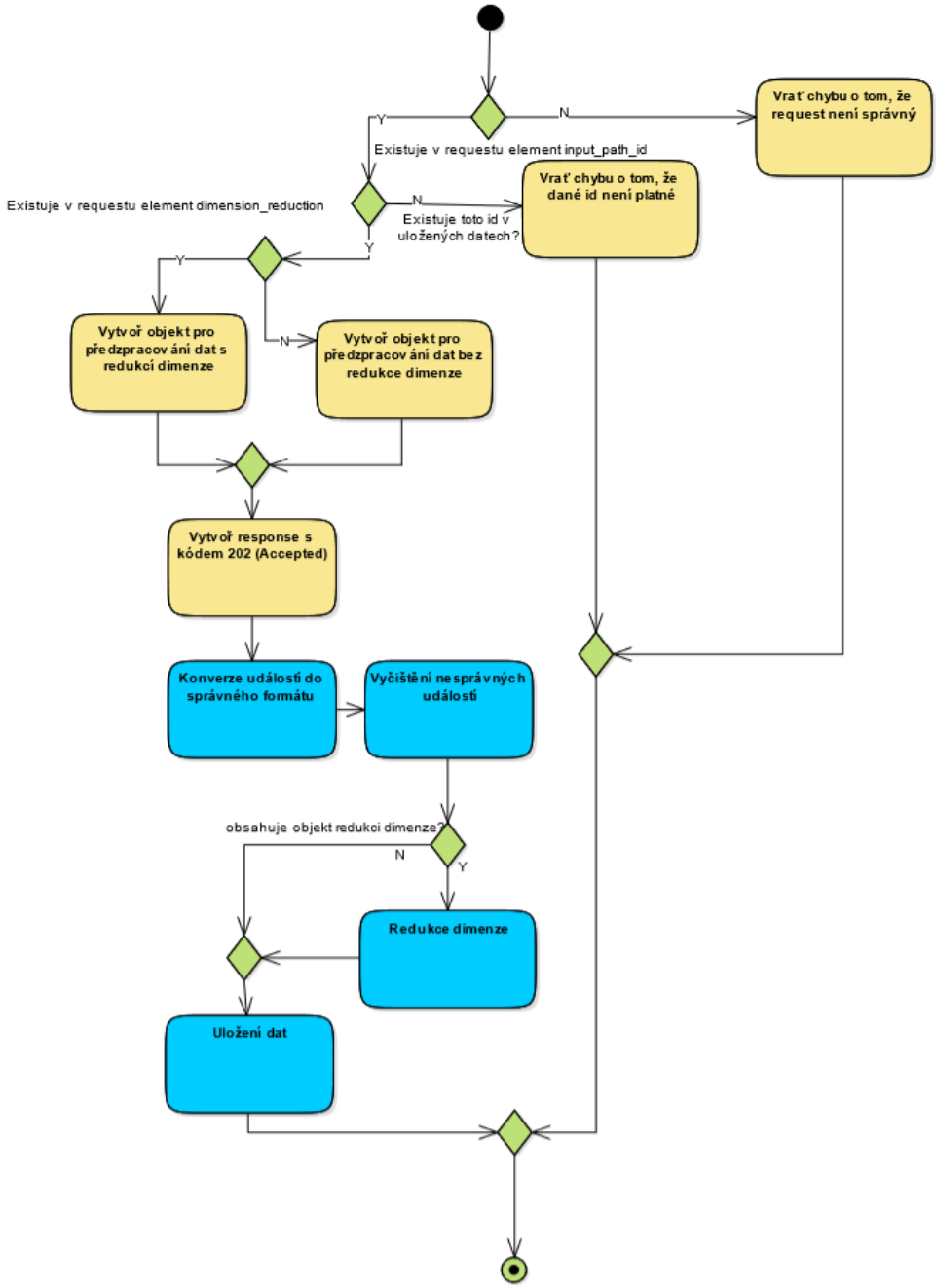
Obrázek 3.7: Aktivity diagram kontroly dat v adresáři.

3.1.6 Předzpracování dat

Tato aktivita je následníkem kontroly dat v adresáři 3.1.5. K předzpracování dat je nutné mít identifikovaná data v adresáři. Následně tato metoda zjišťuje, jestli je v requestu žádost o redukci dimenze a data uloží do objektu a vrátí response o úspěšném přijetí předzpracování a id, pod kterým bylo předzpracování vytvořeno. Následná logika již běží ve vlákně na serveru. O průběhu předzpracování se poté může uživatel dozvědět pomocí dotazování na obdržené id, kde po každém kroku změní předzpracování stav. Tyto stavy jsou popsány ve stavovém diagramu 3.8. Bližší logika je popsána v aktivity diagramu předzpracování 3.9.



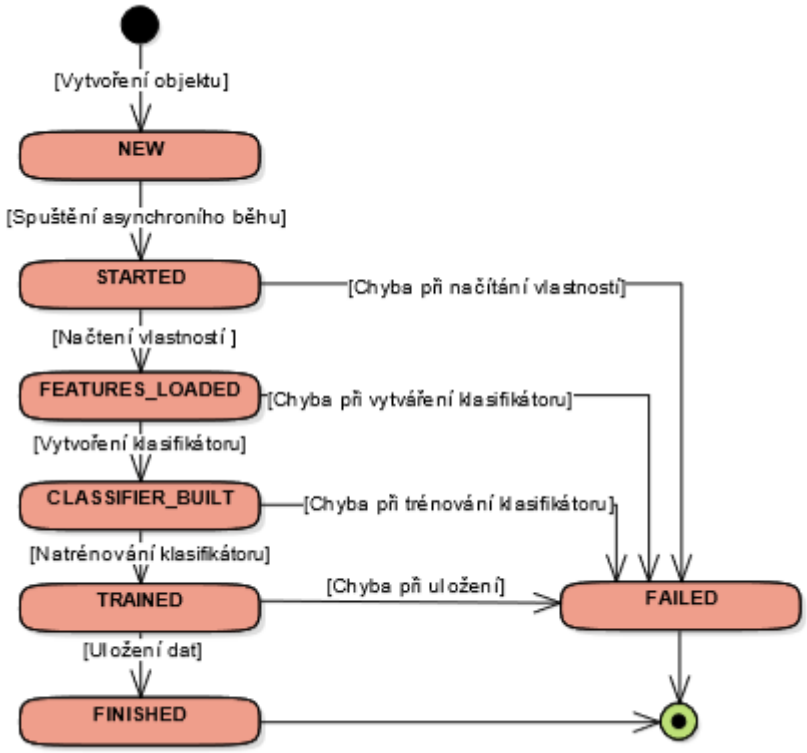
Obrázek 3.8: Stavový diagram předzpracování.



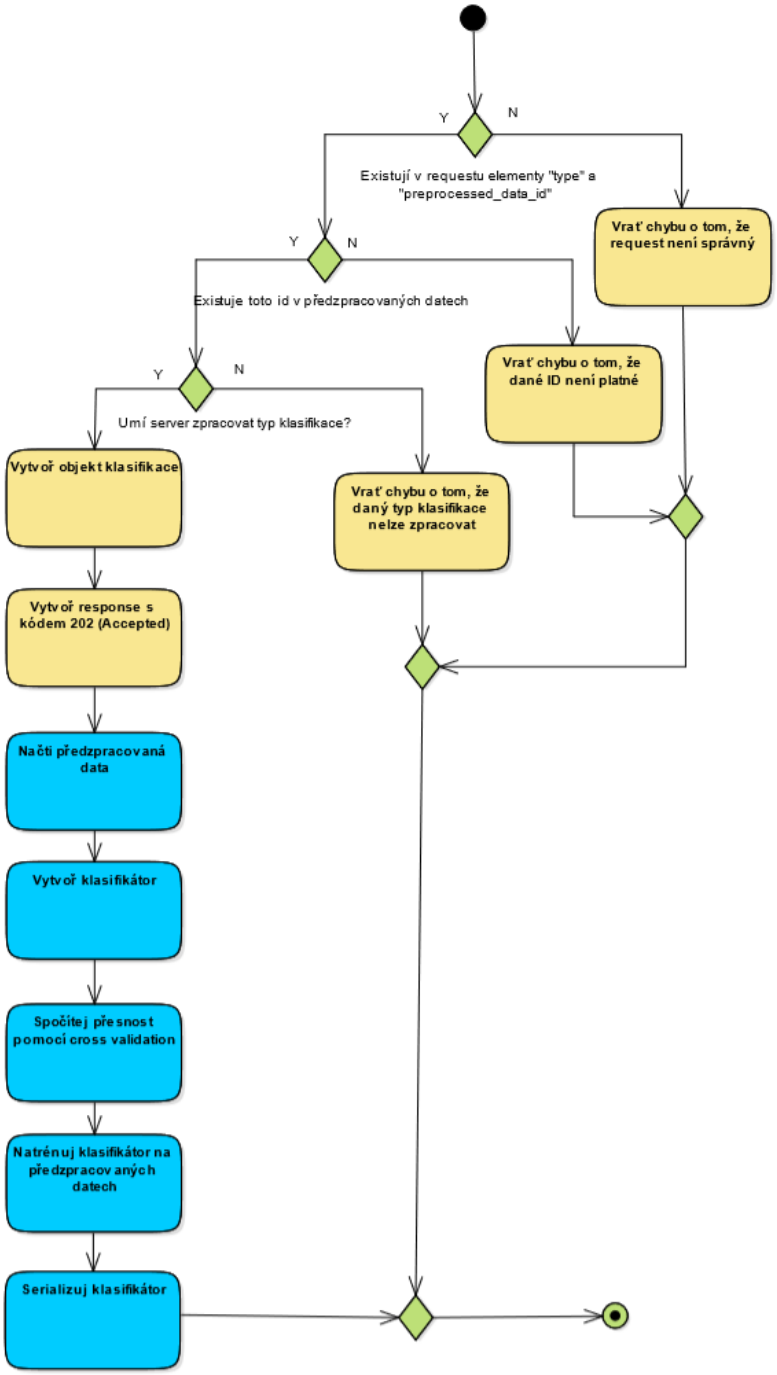
Obrázek 3.9: Aktivity diagram předzpracování (modré aktivity znamenají, že běží na pozadí).

3.1.7 Trénování klasifikace

Tato aktivita je následníkem předzpracování 3.1.6. K natrénování klasifikace je nutné mít předzpracovaná data a zároveň vybrat správnou klasifikační metodu - takovou, kterou umí server zpracovat. Následně získaná data uloží do objektu a vrátí response o úspěšném vytvoření klasifikace s identifikátorem, pod kterým byla klasifikace vytvořena. Podobně jako v předzpracování je následná logika zpracována na pozadí a uživatel může sledovat průběh pomocí dotazování se na konkrétní identifikátor, kde je průběh určený stavy. Stavy jsou popsány ve stavovém diagramu 3.10. Bližší logika je popsána v aktivity diagramu předzpracování 3.11.



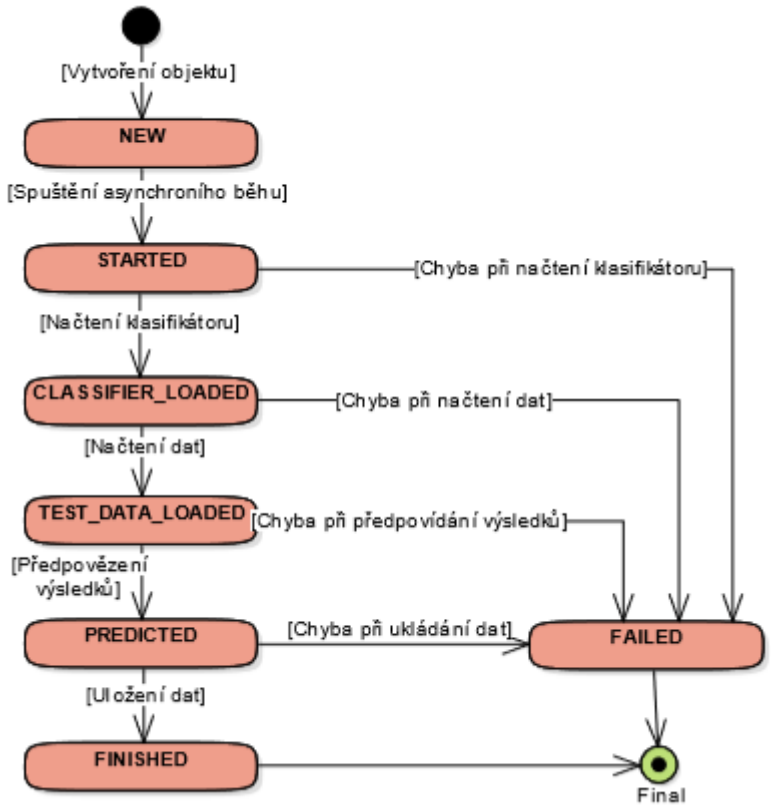
Obrázek 3.10: Stavový diagram trénování klasifikace.



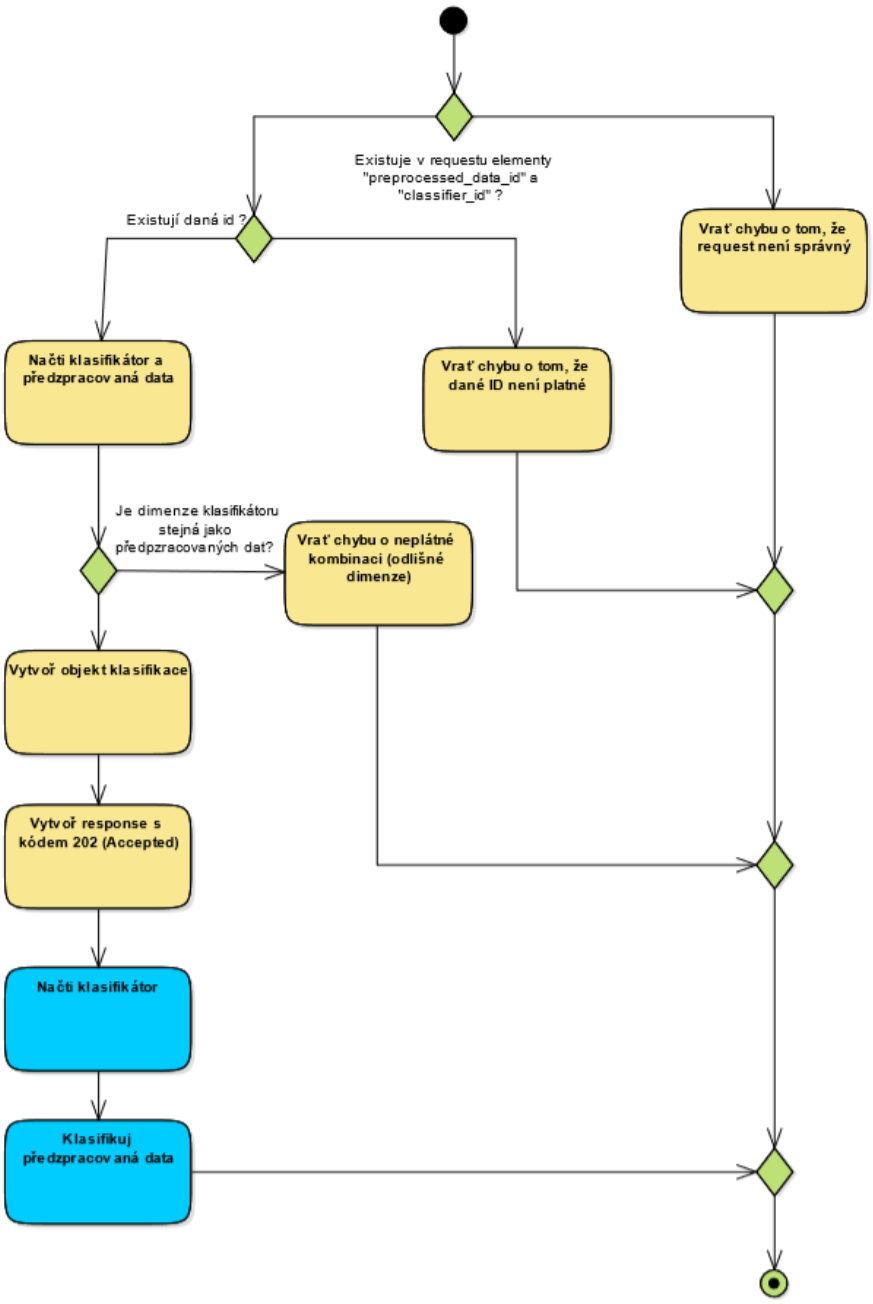
Obrázek 3.11: Aktivity diagram trénování klasifikace (modré aktivity znamenají, že běží na pozadí).

3.1.8 Testování klasifikace

Tato aktivita je také následníkem předzpracování 3.1.6 a zároveň musí existovat klasifikátor, pomocí kterého se budou předzpracovaná data testovat. Pro otestování je tedy potřeba tato data načíst, vytvořit objekt testovací klasifikace a vrátit odpověď o úspěšném přijetí s id testovací klasifikace. Následně, stejně jako v sekcích předzpracování 3.1.6 a trénování klasifikace 3.1.7, se testování vykonává na pozadí, přičemž uživatel může sledovat průběh pomocí dotazování se na konkrétní testovací klasifikaci. Průběh testování je opět určen stavy popsanými v stavovém diagramu 3.12. Bližší logika je popsána v aktivity diagramu 3.13.



Obrázek 3.12: Stavový diagram trénování klasifikace

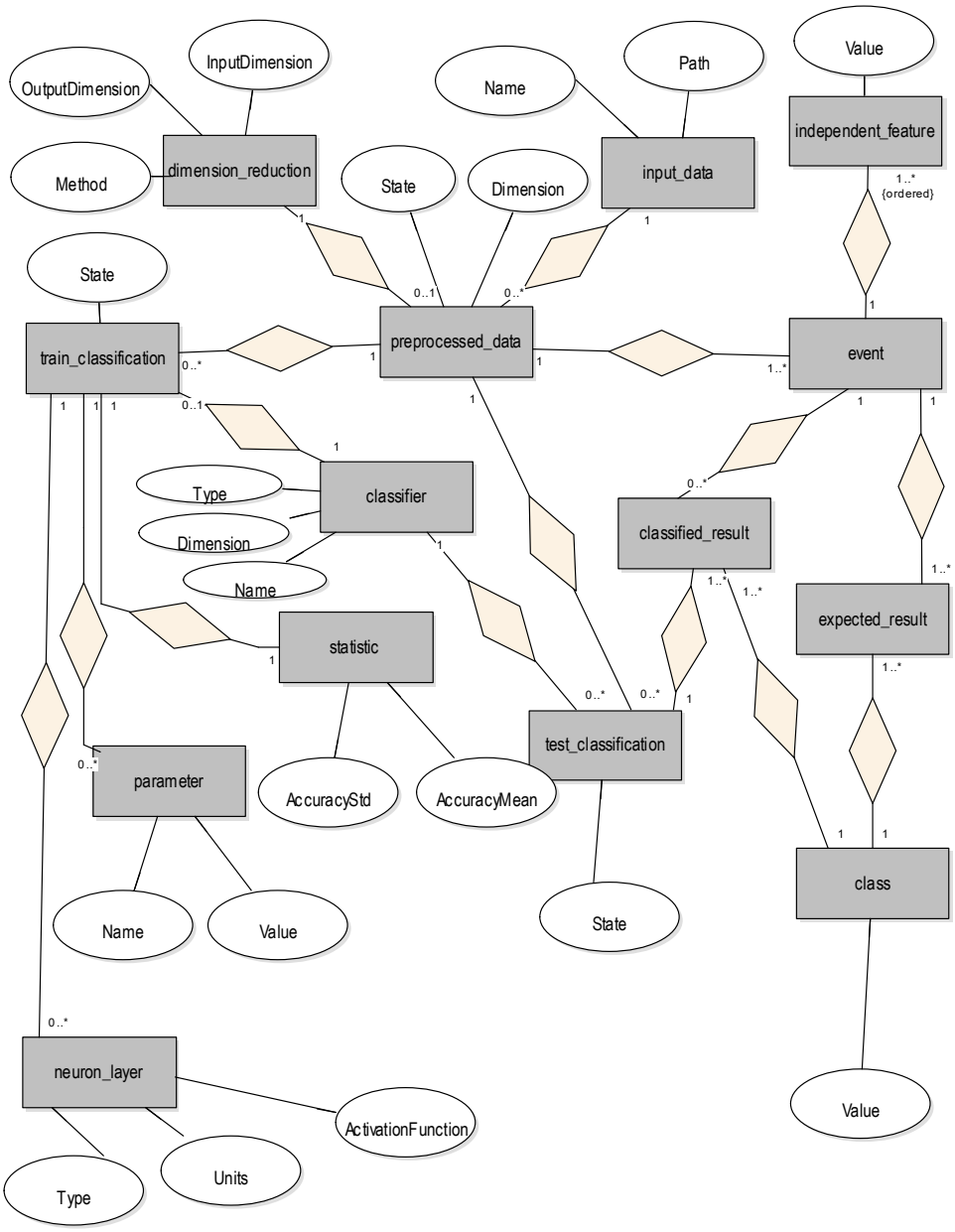


Obrázek 3.13: Aktivity diagram trénování klasifikace (modré aktivity znamenají, že běží na pozadí).

3.1.9 Struktura dat

Všechna data budou ukládána do databáze ve struktuře popsané pomoci diagramu vztahů entit 3.14. Jednotlivé entity jsou:

- input_data - vstupní data, obsahují pouze cestu k adresáři a název souboru.
- preprocessed_data - předzpracovaná data, vychází ze vstupních dat a obsahují informaci o běhu předzpracování (stav) a o typu předzpracování (dimenze). Dále mají nutné napojení na události a volitelné napojení na trénovací, testovací klasifikaci a redukci dimenze.
- dimension_reduction - entita obsahující informace o redukci dimenze. Každá entita se vždy vztahuje pouze na jednu entitu předzpracování. Obsahuje informace o metodě, vstupní a výstupní dimenzi.
- event - entita spojující nezávislé a závislé vlastnosti.
- independent_feature - entita obsahující informace o nezávislých vlastnostech. Obsahuje hodnotu, je seřazená a vždy se vztahuje pouze k jedné konkrétní události.
- expected_result - entita obsahující informace o třídě události. Vždy se vztahuje pouze ke konkrétní události.
- train_classification - entita reprezentující proces trénování klasifikace. Obsahuje pouze stav a má relace na parametry, statistiky a neuronové vrstvy.
- parameter - entita reprezentující parametry klasifikace pomocí názvu parametru a hodnoty parametru.
- neuron_layer - entita reprezentující skrytou vrstvu neuronové sítě. Obsahuje parametry pro vytvoření jednotlivých neuronových vrstev - jediná povinná hodnota.
- statistic - entita reprezentující statistiku klasifikace.
- classifier - entita obsahuje název klasifikátoru a základní údaje pro vytvoření testovací klasifikace (dimenze a typ).
- test_classification entita reprezentující proces testování klasifikace. Podobně jako trénovací klasifikace obsahuje pouze stav a vztah ke klasifikovaným hodnotám.
- classified_result - klasifikované hodnoty. Obsahuje pouze id třídy, na kterou je klasifikována.
- class - třída, obsahuje pouze název třídy klasifikace.



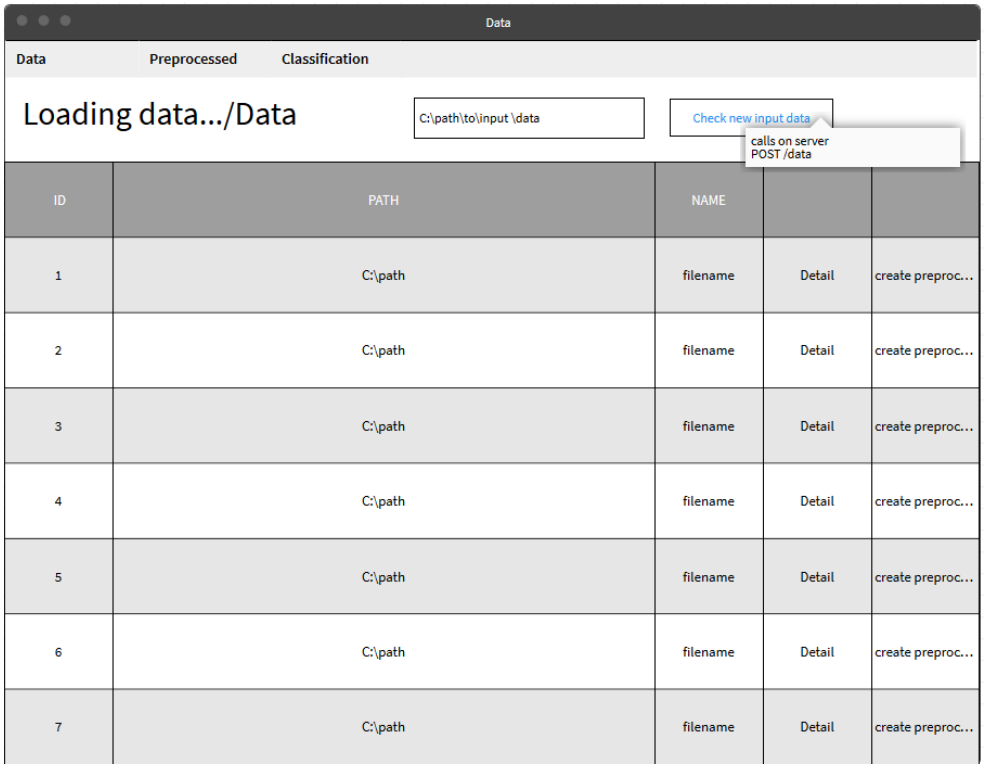
Obrázek 3.14: Diagram vztahů entit.

3.2 Klient

Klientská část aplikace bude sloužit jako grafické rozhraní pro volání serverových metod a sama nebude obsahovat žádnou logiku. Tato část se věnuje popsání jednotlivých obrazovek.

3.2.1 Zobrazení existujících dat

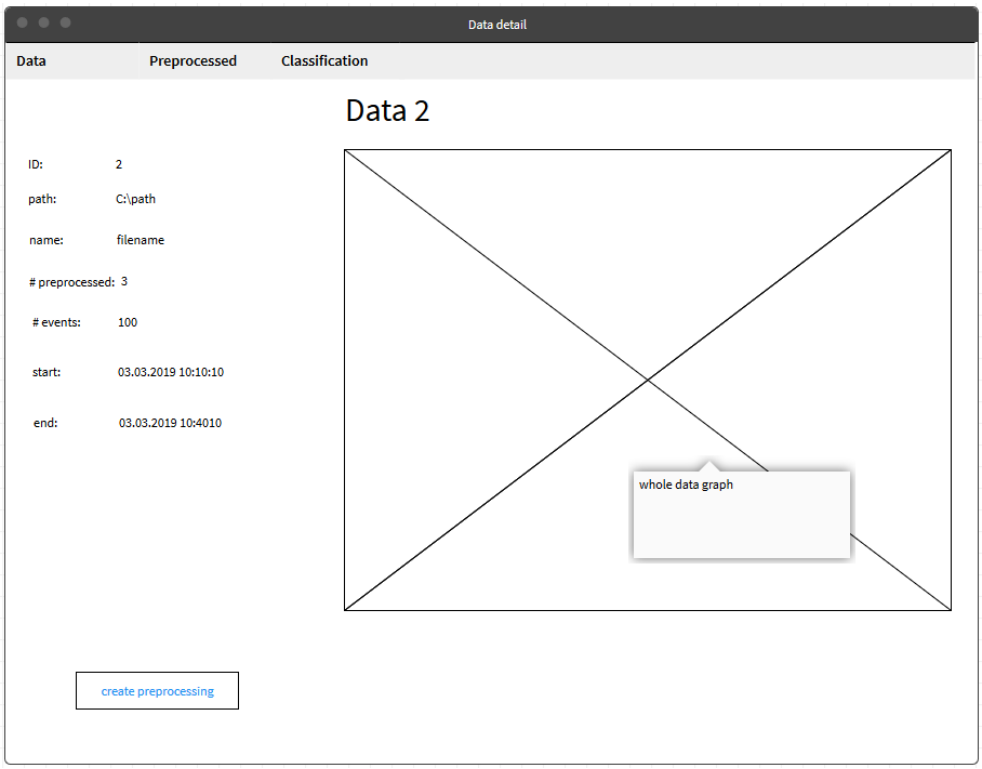
Obrazovka slouží k zobrazení všech zaregistrovaných existujících dat. Na této obrazovce je možné zaregistrovaná data smazat, zkontrolovat nová data 3.1.5, zobrazit jejich detail 3.2.2 a vytvořit předzpracování 3.2.3.



Obrázek 3.15: Wireframe zobrazení existujících dat.

3.2.2 Zobrazení detailu existujících dat

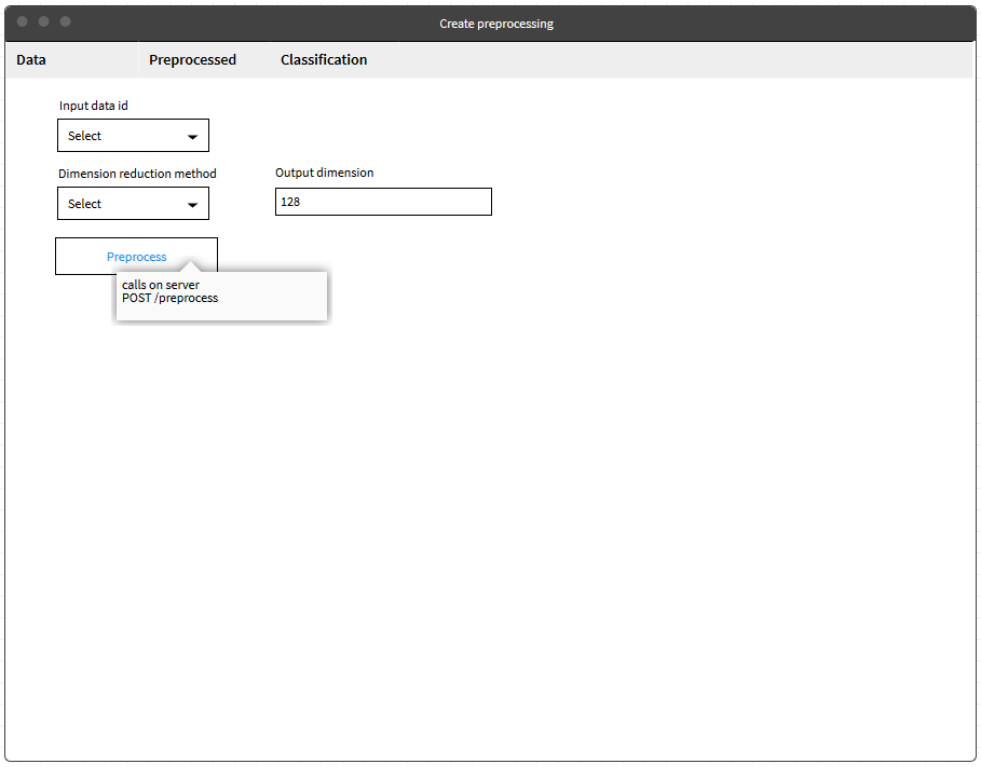
Slouží k zobrazení bližších informací o konkrétních datech včetně grafu. Z této obrazovky je možné vytvořit předzpracování 3.2.3.



Obrázek 3.16: Wireframe zobrazení detailu existujících dat.

3.2.3 Vytvoření předzpracování

Obrazovka slouží k vytvoření nových předzpracovaných dat. Obsahuje jednoduchý formulář, kde uživatel může zvolit, jaká data chce předzpracovat a zda chce redukovat dimenzi. Při potvrzení formuláře se vše posílá na server, který vykoná předzpracování, jak je popsáno v 3.1.6.



Obrázek 3.17: Wireframe vytvoření předzpracování.

3.2.4 Zobrazení všech předzpracování

Obrazovka slouží k zobrazení všech existujících předzpracovaných dat. Na této obrazovce je možné data aktualizovat (načte nová data a aktualizuje stavy), zobrazit detail 3.2.5, a pokud budou data ve stavu PREPROCESSED, tak bude možné vytvořit trénování klasifikace 3.2.6.

Preprocessed

Data | Preprocessed | Classification

Loading preprocessed data.../Preprocessed

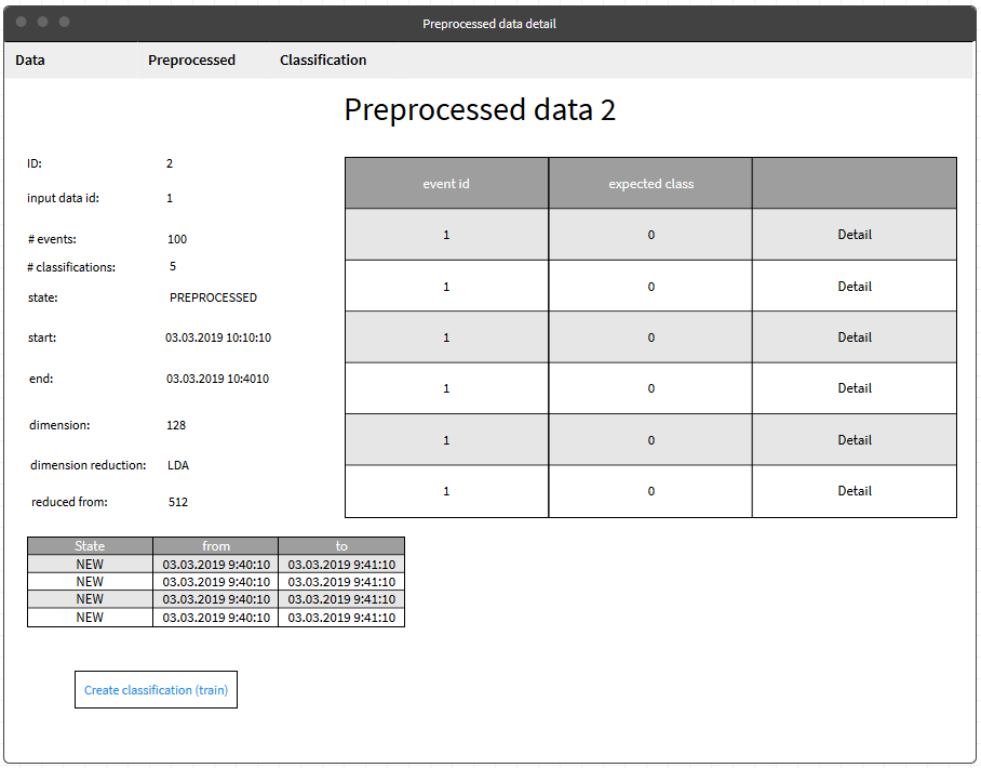
update

ID	input data id	dimension reduction	#events	STATE		
1	1	LDA(128)	100	PREPROCESSED	Detail	Create classific...
2	3		50	PREPROCESSED	Detail	Create classific...
3	1	PCA(128)	100	PREPROCESSED	Detail	Create classific...
4	1		100	PREPROCESSED	Detail	Create classific...
5	4	LDA(2)	100	PREPROCESSED	Detail	Create classific...
6	4	LDA(8)	100	PREPROCESSED	Detail	Create classific...
7	4	LDA(128)	100	PREPROCESSED	Detail	Create classific...

Obrázek 3.18: Wireframe zobrazení všech předzpracování.

3.2.5 Zobrazení detailu předzpracování

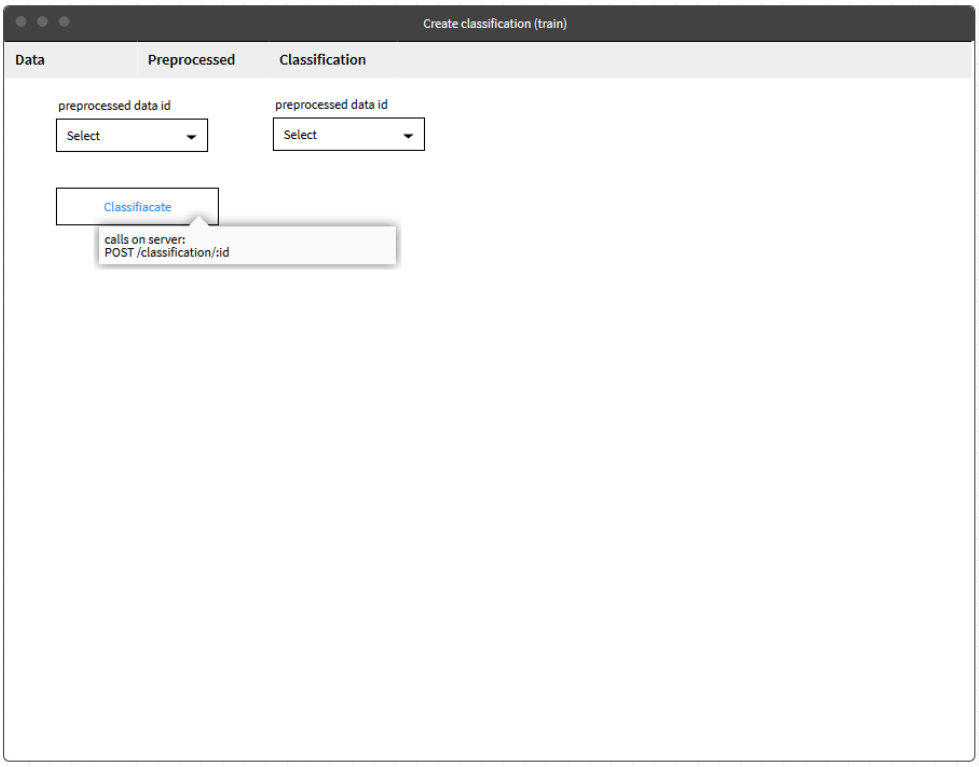
Obrazovka slouží k zobrazení bližších informací konkrétních předzpracovaných dat. Z této obrazovky je možné vytvořit trénování klasifikace. 3.2.6.



Obrázek 3.19: Wireframe zobrazení detailu předzpracování.

3.2.6 Vytvoření trénovací klasifikace

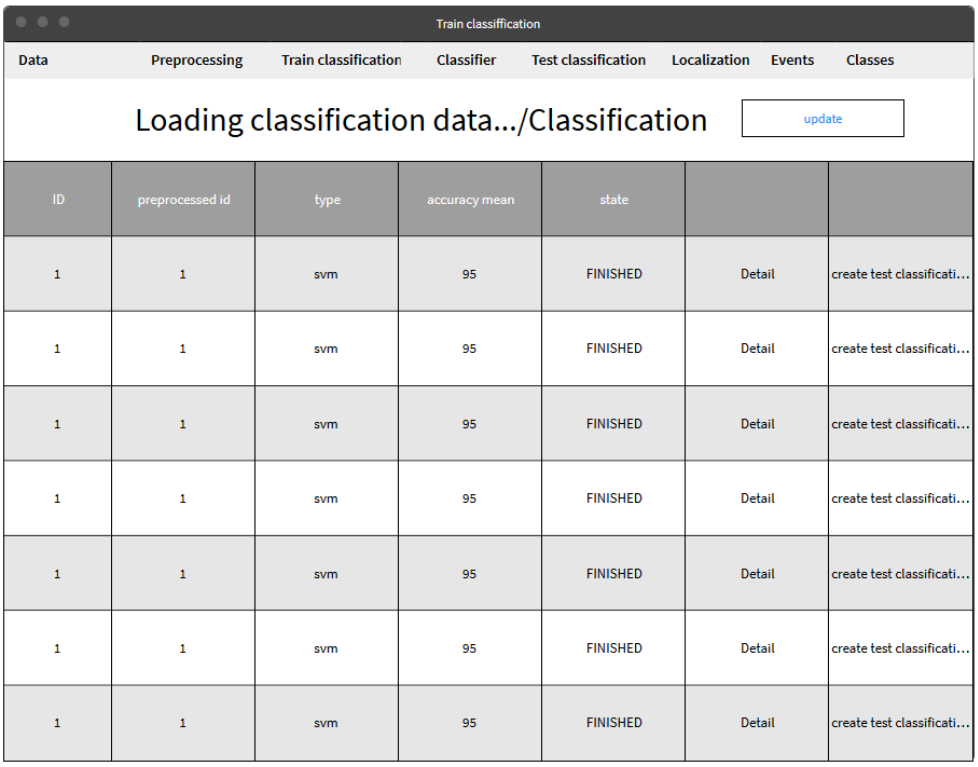
Obrazovka slouží k vytvoření nové trénovací klasifikace. Obsahuje formulář, kde uživatel zvolí typ předzpracovaných dat, klasifikační metodu a volitelně její parametry. V případě neuronových sítí je možné vytvořit libovolnou strukturu neuronové sítě. Při potvrzení formuláře se vše posílá na server, který vytvoří trénovací klasifikaci, jak je popsáno v 3.1.7.



Obrázek 3.20: Wireframe vytvoření trénovací klasifikace.

3.2.7 Zobrazení trénovacích klasifikací

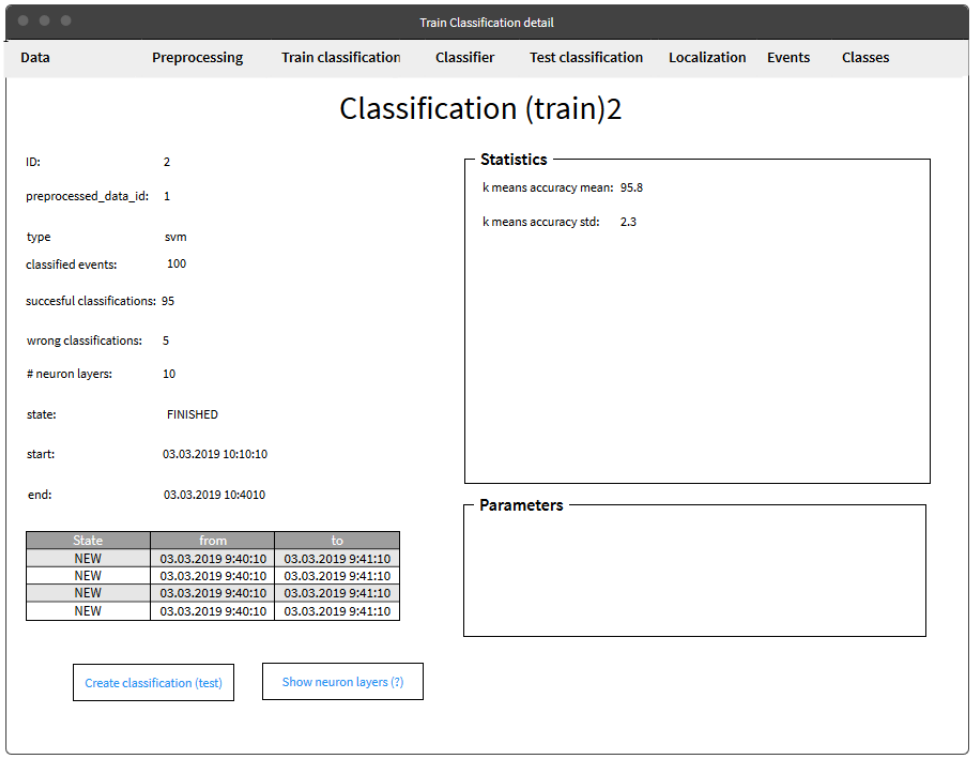
Obrazovka slouží k zobrazení všech existujících klasifikací. Na této obrazovce je možné data aktualizovat (načte nové trénovací klasifikace a aktualizuje stavy), zobrazit detail 3.2.8, a pokud je klasifikace ve stavu FINISHED se zaregistrovaným klasifikátorem, tak je možné vytvořit testovací klasifikaci 3.2.10.



Obrázek 3.21: Wireframe zobrazení trénovacích klasifikací.

3.2.8 Zobrazení detailu trénovací klasifikace

Obrazovka slouží k zobrazení bližších informací konkrétní trénovací klasifikace. Z této obrazovky je možné vytvořit testovací klasifikaci 3.2.10.



Obrázek 3.22: Wireframe zobrazení detailu trénovací klasifikace.

3.2.9 Zobrazení existujících klasifikátorů

Obrazovka slouží ke zobrazení všech existujících klasifikátorů. Z této obrazovky je možné zkontrolovat klasifikátory 3.1.4 a vytvořit testovací klasifikaci 3.2.10.

Classifier

Data | Preprocessing | Train classification | Classifier | Test classification | Localization | Events | Classes

Loading classifiers.../Classifiers

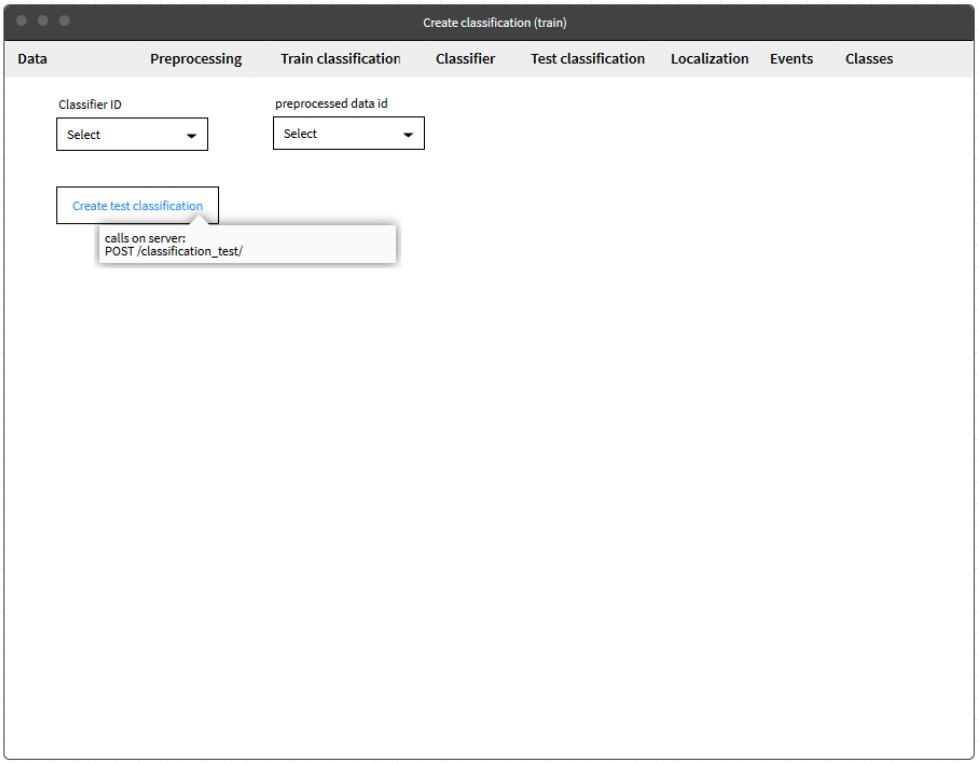
check classifiers

ID	path	name	dimension	type	
1	classifiers/	18.joblib	2	SVM	create test classification
1	classifiers/	18.joblib	2	SVM	create test classification
1	classifiers/	18.joblib	2	SVM	create test classification
1	classifiers/	18.joblib	2	SVM	create test classification
1	classifiers/	18.joblib	2	SVM	create test classification
1	classifiers/	18.joblib	2	SVM	create test classification
1	classifiers/	18.joblib	2	SVM	create test classification
1	classifiers/	18.joblib	2	SVM	create test classification

Obrázek 3.23: Wireframe zobrazení existujících klasifikátorů.

3.2.10 Vytvoření testovací klasifikace

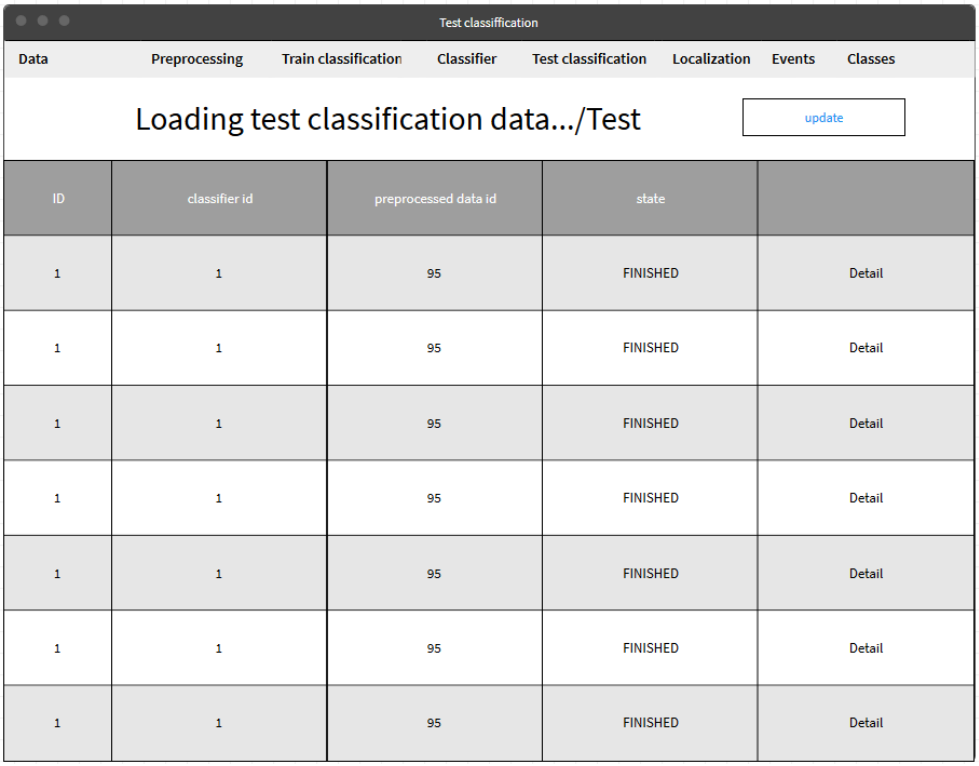
Obrazovka slouží k vytvoření nové testovací klasifikace. Obsahuje formulář, kde uživatel zvolí typ předzpracovaných dat a klasifikátor. Při potvrzení formuláře se vše posílá na server, který vytvoří testovací klasifikaci, jak je popsáno v 3.1.8.



Obrázek 3.24: Wireframe vytvoření testovací klasifikace.

3.2.11 Zobrazení testovacích klasifikací

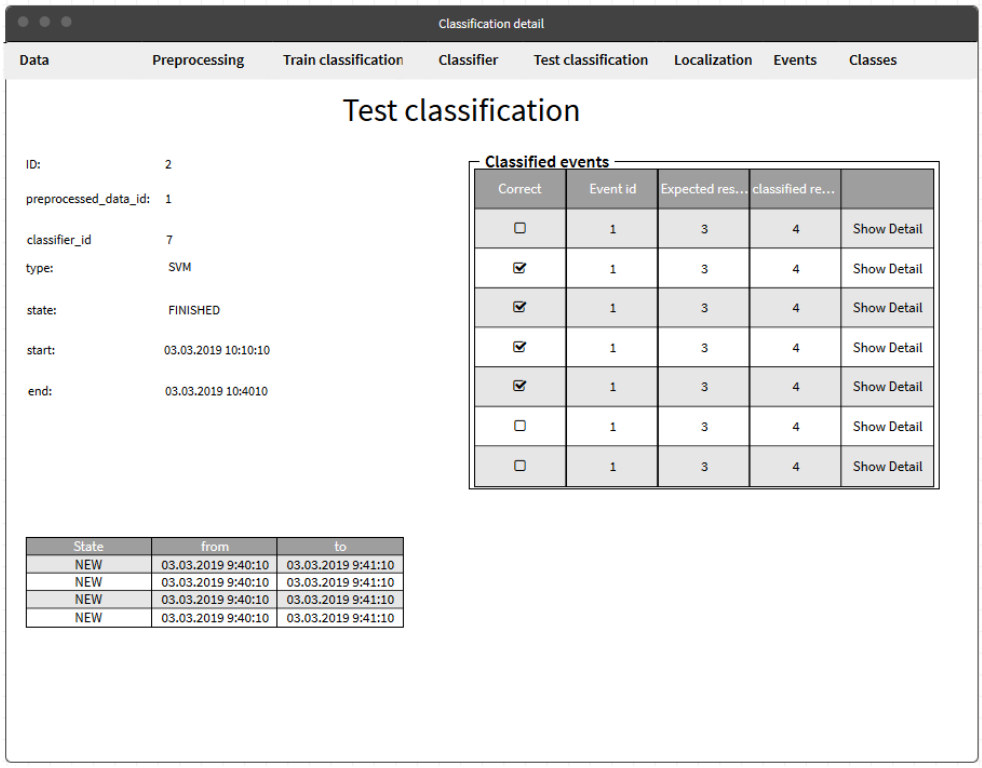
Obrazovka slouží k zobrazení všech existujících testovací klasifikací. Na této obrazovce je možné data aktualizovat (načte nové testovací klasifikace a aktualizuje stavy) a zobrazit detail testovací klasifikace 3.2.12.



Obrázek 3.25: Wireframe zobrazení testovacích klasifikací.

3.2.12 Zobrazení detailu testovací klasifikace

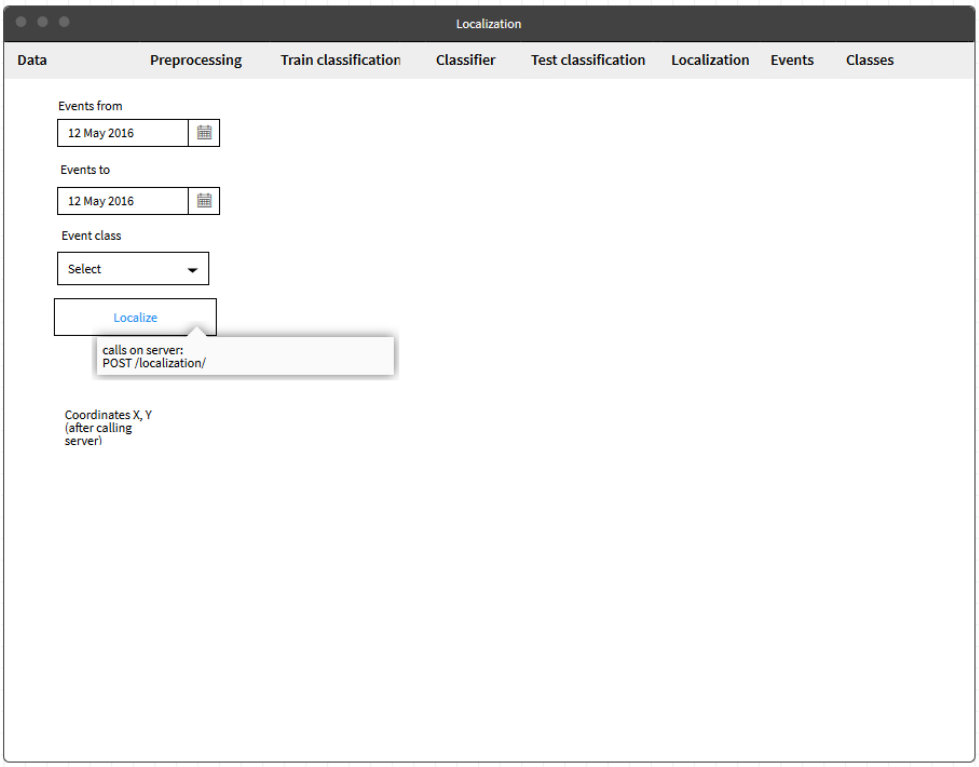
Obrazovka slouží k zobrazení bližších informací konkrétní testovací klasifikace a všech událostí, které otestovala. Z této obrazovky bude možné se podívat na detail jednotlivých událostí 3.2.15.



Obrázek 3.26: Wireframe zobrazení detailu testovací klasifikace.

3.2.13 Lokalizace

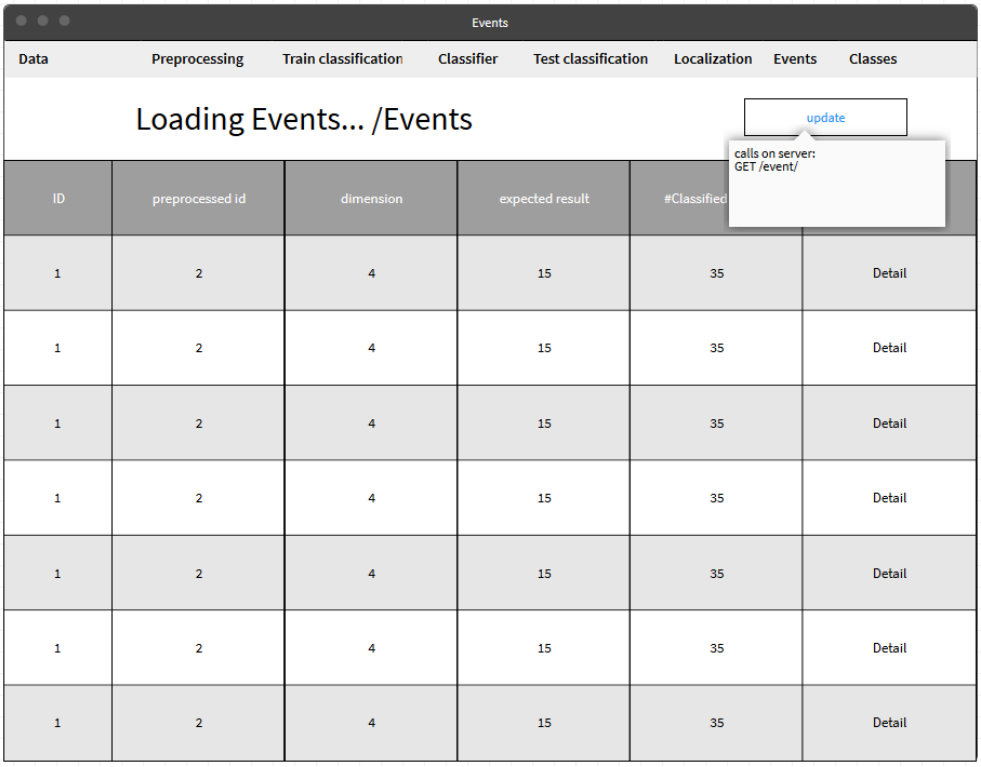
Obrazovka slouží k lokalizaci událostí. Obsahuje formulář, kde uživatel zvolí čas začátku a konce, v kterém se mají události zvoleného typu hledat. Při potvrzení formuláře se vše posílá na server, který události lokalizuje, jak je popsáno v 3.1.3. Vrácené souřadnice se zobrazí na této obrazovce.



Obrázek 3.27: Wireframe lokalizace.

3.2.14 Zobrazení všech událostí

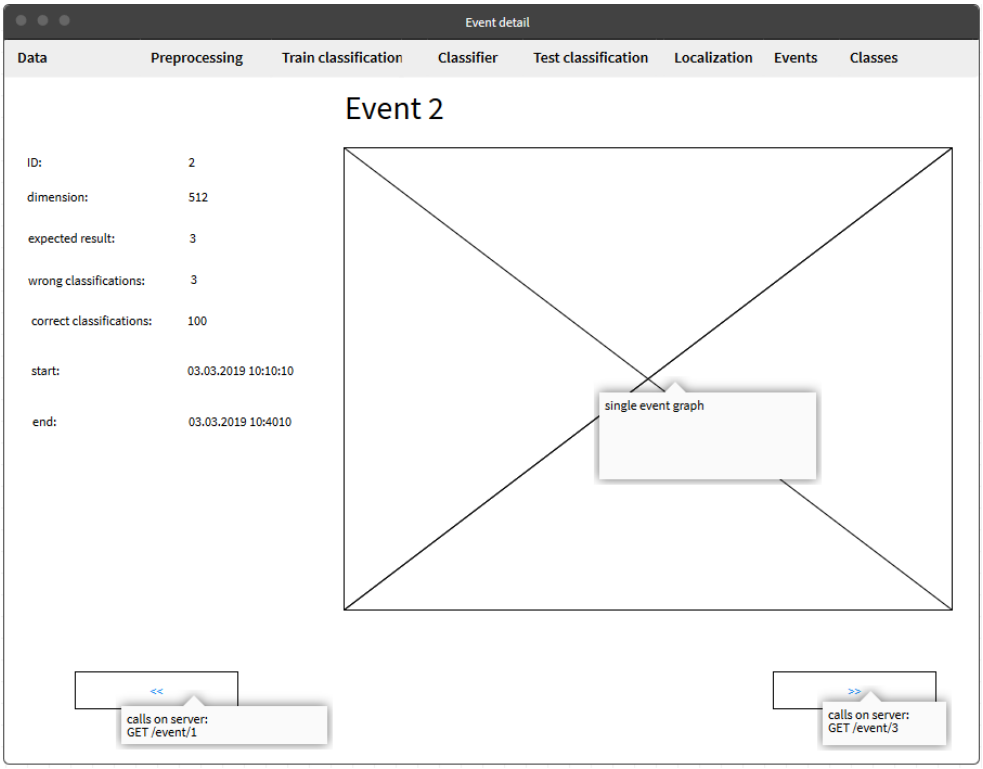
Obrazovka slouží k zobrazení všech existujících událostí. Na této obrazovce bude možné události aktualizovat a zobrazit detail události 3.2.15.



Obrázek 3.28: Wireframe zobrazení všech událostí.

3.2.15 Zobrazení detailu události

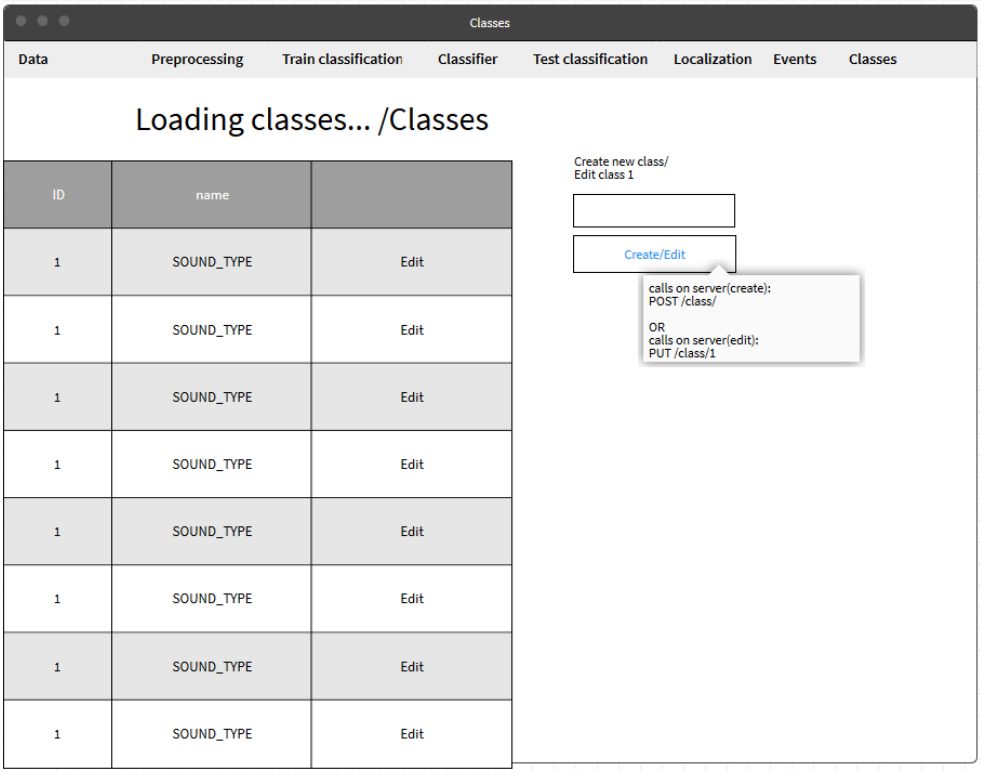
Obrazovka slouží k zobrazení bližších informací konkrétní události. Na této obrazovce bude možné změnit typ třídy události a hodnotu uložit. Dále bude obrazovka obsahovat odkazy na předchozí a nadcházející událost, pokud budou existovat.



Obrázek 3.29: Wireframe zobrazení detailu události.

3.2.16 Zobrazení všech tříd

Obrazovka slouží k zobrazení všech existujících tříd. Na této obrazovce bude možné vytvořit novou třídu v jednoduchém formuláři s jedním polem - názvem události. Dále bude možné upravit název existujících tříd.



Obrázek 3.30: Wireframe zobrazení všech tříd.

Kapitola 4

Implementace

4.1 Server

Serverová část je napsána ve frameworku django, kde se o jednotlivé URL starají konkrétní podcelky nazývané aplikace. Následující sekce je rozdělená do podsekcí podle těchto aplikací. složitější logika se nachází v hlavní aplikaci ossp v adresáři businesslogic. Příklady volání všech implementovaných metod jsou dostupné v projektu ossp.postman aplikace Postman.

4.1.1 Data

O metody související s daty se stará aplikace check_data, která implementuje metody popsané v této podsekci. Tyto metody slouží pro práci se vstupními daty.

4.1.1.1 Zkontroluj data

Popis: Zkontroluje data v daném adresáři a pokud existují nová, tak je uloží.
URL: /check_data/
Metoda: POST
Parametry dat:

```
{
    path: String
}
```

4.1.1.2 Zobraz data

Popis: Zobrazí všechna uložená data.
URL: /data/

Metoda: GET

4.1.1.3 Smaž data

Popis: Smaže všechna uložená data.
URL: /check_data/
Metoda: DELETE

4.1.1.4 Zobraz detail dat

Popis: Zobrazí detail konkrétních uložených dat.
URL: /data/:id
Metoda: GET

4.1.1.5 Zobraz graf

Popis: Zobrazí graf konkrétních dat.
URL: /data/:id/graph
Metoda: GET

4.1.2 Předzpracování

O metody předzpracování se stará aplikace preprocess, která implementuje metody popsané v této podsekci. Tyto metody slouží pro práci s předzpracovanými daty.

4.1.2.1 Vytvoř předzpracování

Popis: Vytvoří nová předzpracovaná data na základě vstupních parametrů.
URL: /preprocess/
Metoda: POST
Parametry dat:

```
{
    input_data_id: Number,
    dimension_reduction: {
        method: String,
        output_dimension: Number
    }
}
```

4.1.2.2 Zobraz všechna předzpracování

Popis: Zobrazí všechna předzpracovaná data.
URL: /preprocess/
Metoda: GET

4.1.2.3 Zobraz detail předzpracování

Popis: Zobrazí detail předzpracovaných dat
URL: /preprocess/:id
Metoda: GET

4.1.3 Trénování klasifikace

O metody trénování klasifikace se stará aplikace classification, která implementuje metody popsané v této podsekci. Tyto metody slouží pro práci s trénovacími klasifikacemi.

4.1.3.1 Vytvoř trénovací klasifikaci

Popis: Vytvoří trénovací klasifikaci na základě vstupních parametrů.
URL: /classification_train/
Metoda: POST
Parametry dat:

```
{
    preprocessed_data_id: Number,
    type: String,
    parameters: [
    ]
}
```

4.1.3.2 Zobraz všechny trénovací klasifikace

Popis: Zobrazí všechny trénovací klasifikace.
URL: /classification_train/
Metoda: GET

4.1.3.3 Zobraz detail trénovací klasifikace

Popis: Zobrazí detail trénovací klasifikace
URL: /classification_train/:id

Metoda: GET

4.1.4 Klasifikátory

O metody klasifikátorů se stará aplikace classifier, která implementuje metody popsané v této podsekci. Tyto metody slouží pro práci s klasifikátory.

4.1.4.1 Zkontroluj nové klasifikátory

Popis: Zkontroluje existenci nových klasifikátorů v adresáři a uloží je.
URL: /classifier/
Metoda: POST

4.1.4.2 Zobraz všechny klasifikátory

Popis: Zobrazí všechny existující klasifikátory.
URL: /classifier/
Metoda: GET

4.1.5 Testování klasifikace

O metody testovací klasifikace se stará aplikace classification_test, která implementuje metody popsané v této podsekci. Tyto metody slouží pro práci s testovací klasifikacemi.

4.1.5.1 Vytvoř testovací klasifikaci

Popis: Vytvoří novou testovací klasifikaci na základě vstupních parametrů.
URL: /classification_test/
Metoda: POST
Parametry dat:

```
{
    preprocessed_data_id: Number,
    classifier_id: Number
}
```

4.1.5.2 Zobraz všechny testovací klasifikace

Popis: Zobrazí všechny existující testovací klasifikace.
URL: /classification_test/

Metoda: GET

4.1.5.3 Zobraz detail testovací klasifikace

Popis: Zobrazí detail testovací klasifikace.
URL: /classification_test/
Metoda: GET

4.1.6 Lokalizace

O metody lokalizace se stará aplikace localization, která implementuje metody popsané v této podsekci. Tyto metody slouží pro práci s lokalizací.

4.1.6.1 Lokalizuj událost

Popis: Lokalizuje události na základě vstupních dat.
URL: /localization/
Metoda: POST
Parametry dat:

```
{
    start: Date,
    end: Date,
    class_id: Number
}
```

4.1.7 Události

O metody událostí se stará aplikace event, která implementuje metody popsané v této podsekci. Tyto metody slouží pro práci s událostmi.

4.1.7.1 Zobraz všechny události

Popis: Zobrazí všechny existující události.
URL: /event/
Metoda: GET

4.1.7.2 Zobraz detail události

Popis: Zobrazí detail existující události
URL: /event/:id

Metoda: GET

4.1.7.3 Zobraz graf události

Popis: Zobrazí graf existující události
URL: /event/:id/graph
Metoda: GET

4.1.7.4 Uprav událost

Popis: Upraví událost (změní třídu)
URL: /event/:id
Metoda: PUT
Parametry dat:

```
{
    expected_result: {
        value: Number
    }
}
```

4.1.8 Třídy

O metody tříd se stará aplikace class_type, která implementuje metody popsané v této podsekci. Tyto metody slouží pro práci s třídami.

4.1.8.1 Zobraz všechny třídy

Popis: Zobrazí všechny existující třídy.
URL: /class/
Metoda: GET

4.1.8.2 Vytvoř třídu

Popis: Vytvoří novou třídu.
URL: /class/
Metoda: POST
Parametry dat:

```
{
    class_name: String
}
```

4.1.8.3 Uprav třídu

Popis: Upraví třídu (změní název).
URL: /class/:id
Metoda: PUT
Parametry dat:

```
{
    class_name: String
}
```

4.2 Klient

Klientská část je napsána ve webovém frameworku Angular. Jednotlivé moduly, starající se o konkrétní URL, se zde nazývají komponenty. Následující sekce je rozdělená do podsekcí korespondujících s hierarchíi těchto komponent. O komunikaci se serverem se starají služby.

4.2.1 ClassComponent

Tato komponenta slouží pouze k seskupení komponent.

4.2.1.1 ClassEditComponent

Komponenta slouží k změně nebo vytvoření nové třídy popsané v návrhu 3.2.16.

4.2.1.2 ClassListComponent

Komponenta slouží k zobrazení seznamu popsané v návrhu 3.2.16.

4.2.1.3 ClassService

Služba slouží ke komunikaci se serverem týkající se tříd 4.1.8.

4.2.2 ClassificationComponent

Tato komponenta slouží pouze k seskupení komponent.

4.2.2.1 ClassificationCreateComponent

Komponenta slouží k vytvoření trénovací klasifikace popsané v návrhu 3.2.6. Obsahuje nejsložitější formulář, kde definuje logiku pro zobrazení všech různých kombinací.

4.2.2.2 ClassificationDetailComponent

Komponenta slouží k zobrazení detailu trénovací klasifikace popsané v návrhu 3.2.8.

4.2.2.3 ClassificationListComponent

Komponenta slouží k zobrazení seznamu trénovací klasifikace popsané v návrhu 3.2.7.

4.2.2.4 ClassificationService

Služba slouží ke komunikaci se serverem týkající se trénovací klasifikace 4.1.3.

4.2.3 ClassifierComponent

Tato komponenta slouží pouze k seskupení komponent.

4.2.3.1 ClassifierListComponent

Komponenta slouží k zobrazení seznamu klasifikátorů popsané v návrhu 3.2.9.

4.2.3.2 ClassifierService

Služba slouží ke komunikaci se serverem týkající se klasifikátorů 4.1.4.

4.2.4 DataComponent

Tato komponenta slouží pouze k seskupení komponent.

4.2.4.1 DataDetailComponent

Komponenta slouží k zobrazení detailu o datech popsané v návrhu 3.2.2.

4.2.4.2 DataListComponent

Komponenta slouží k zobrazení seznamu dat popsané v návrhu 3.2.1.

4.2.4.3 DataService

Služba slouží ke komunikaci se serverem týkající se vstupních dat 4.1.1.

4.2.5 EventComponent

Tato komponenta slouží pouze k seskupení komponent.

4.2.5.1 EventDetailComponent

Komponenta slouží k zobrazení detailu o události popsané v návrhu 3.2.15.

4.2.5.2 EventListComponent

Komponenta slouží k zobrazení seznamu událostí popsané v návrhu 3.2.14.

4.2.5.3 EventService

Služba slouží ke komunikaci se serverem týkající se událostí 4.1.7.

4.2.6 LocalizationComponent

Tato komponenta slouží k lokalizaci popsané v návrhu 3.2.13.

4.2.6.1 LocalizationService

Služba slouží ke komunikaci se serverem týkající se lokalizace 4.1.6.

4.2.7 PreprocessingComponent

Tato komponenta slouží pouze k seskupení komponent.

4.2.7.1 PreprocessingCreateComponent

Komponenta slouží k vytvoření předzpracování popsané v návrhu 3.2.3.

4.2.7.2 PreprocessingDetailComponent

Komponenta slouží k zobrazení detailu předzpracování popsané v návrhu 3.2.5.

4.2.7.3 PreprocessingListComponent

Komponenta slouží k zobrazení seznamu předzpracování popsané v návrhu 3.2.4.

4.2.7.4 PreprocessingService

Služba slouží ke komunikaci se serverem týkající se předzpracování dat 4.1.2.

4.2.8 TestClassificationComponent

Tato komponenta slouží pouze k seskupení komponent.

4.2.8.1 TestClassificationCreateComponent

Komponenta slouží k vytvoření trénovací klasifikace popsané v návrhu 3.2.10.

4.2.8.2 TestClassificationDetailComponent

Komponenta slouží k zobrazení detailu trénovací klasifikace popsané v návrhu 3.2.12.

4.2.8.3 TestClassificationListComponent

Komponenta slouží k zobrazení trénovacích klasifikací popsané v návrhu 3.2.11.

4.2.8.4 TestClassificationService

Služba slouží ke komunikaci se serverem týkající se testovací klasifikace 4.1.5.

4.3 Přidání metody klasifikace

Pro přidání nové metody klasifikace z knihovny sklearn je třeba několik zásahů na serverové a klientské části. Postup je následující:

1. Na serverové části vytvořit novou třídu, která bude dědit z abstraktní třídy AbstractClassifier, a naimplementovat metodu build_classifier a konstruktor tak, aby ukládal všechna důležitá data o klasifikaci.

2. Na serverové části v aplikaci classification v souboru views.py přidat podmínku na nový název klasifikace.

3. V klientské části upravit komponentu ClassificationCreateComponent tak, aby uměla pracovat s novou klasifikací (přidat název, pod kterým je identifikována na serveru, popřípadě přidat parametry).

Kapitola 5

Testování

5.1 Testovací případy

Následující sekce se věnuje popsání testovacích případů klientské aplikace a vzhledem k tomu, že klientská aplikace pouze přepoužívá logiku serveru, lze předpokládat, že tím bude otestována i serverová část aplikace.

T_1 **Vytvoření trénovací klasifikace typu SVM**

Účel: Ověření, zda lze provést funkční požadavek F_1 a požadavek F_2 a F_9.

Čas: 2 min.

Podmínky: Existující předzpracovaná data.

Kroky:

1. U vybraných předzpracovaných dat na stránce D.4 kliknu na „Create classification“.
2. Vyberu SVM.
3. Kliknu na „Create Classification“.

Očekávané výsledky:

a) Proběhné natrénování klasifikace.
b) Na stránce D.7 bude nově vytvořená klasifikace.

Provedení testu: OK.

T_2 **Vytvoření testování klasifikace pro daný klasifikátor pro více předzpracovaných dat**

Účel: Ověření, zda lze provést funkční požadavek F_3 a F_9.

Čas: 2 min.

Podmínky: Existující předzpracovaná data a klasifikátor.

Kroky:

1. U vybraného klasifikátoru na stránce D.9 kliknu na „Test classification“.
2. Vyberu předzpracovaná data.
3. Kliknu na „Create Classification“.
4. Zopakuji od kroku 2 pro všechna předzpracovaná data.

Očekávané výsledky:

a) Proběhné testování klasifikace pro každá předzpracovaná data.
b) Na stránce D.11 budou nově vytvořené testované klasifikace.

Provedení testu: OK.

T_3 **Vytvoření předzpracování dat s redukcí dimenze pomocí LDA na 2**

Účel: Ověření, zda lze provést funkční požadavek F_4 a F_9.

Čas: 5 min.

Podmínky: Existující vstupní data.

Kroky:

1. U vybrých vstupních dat na stránce D.4 kliknu na „Preprocess“.
2. Vyberu metodu redukce dimenze LDA a nastavím „output_dimension“ na 2.
3. Kliknu na „Preprocess“.

Očekávané výsledky:

a) Proběhne předzpracování dat s redukcí dimenze LDA do dimenze 2.
b) Na stránce D.4 budou nově vytvořená předzpracovaná data.

Provedení testu: OK.

T_4 **Změna typu události na „compressor“**

Účel: Ověření, zda lze provést funkční požadavek F_5.

Čas: 1 min.

Podmínky: Existující událost pro předzpracovaná data a existující třída compressor.

Kroky:

1. U vybrané události na stránce D.14 kliknu na „Show Detail“.
2. Změním „expected result“ na „compressor“.
3. Kliknu na „Save“.

Očekávané výsledky:

a) U události se změní expected result na „compressor“.

Provedení testu: OK.

T_5 **Změna názvu třídy z „ball_drop“ na „walk“**

Účel: Ověření, zda lze provést funkční požadavek F_6.

Čas: 2 min.

Podmínky: Existující třída „ball_drop“.

Kroky:

1. U třídy „ball_drop“ na stránce D.16 kliknu na „Edit“.
2. Změním název z „ball_drop“ na „walk“.
3. Kliknu na „Edit“.

Očekávané výsledky:

a) U třídy se změní název na „walk“.

Provedení testu: OK.

Poznámky Název se nezmění na stránce ihned, ale až po aktualizaci.

T_6 **Zobrazení parametrů trénovací klasifikace s ID 2**

Účel: Ověření, zda lze provést funkční požadavek F_7.

Čas: 2 min.

Podmínky: Existující trénovací klasifikace s ID 2.

Kroky:

1. U trénovací klasifikace s ID 2 na stránce D.11 kliknu na „Show Detail“.

Očekávané výsledky:

a) Na stránce D.12 bude informace o parametrech klasifikace.

Provedení testu: OK.

Poznámky Klasifikace musí být ve stavu „FINISHED“.

T_7 **Lokalizace události typu „compressor“ v mezi časy 12:53:27 a 12:53:28**

Účel: Ověření, zda lze provést funkční požadavek F_8.

Čas: 2 min.

Podmínky: Existující alespoň 2 události typu „compressor“ mezi časy 12:53:27 a 12:53:28.

Kroky:

1. Na stránce D.13 vyberu událost „compressor“, do hodnoty start vyberu 12:53:27 a do hodnoty end vyberu 12:53:28.
2. Kliknu na „Localize“.

Očekávané výsledky:

a) Zobrazí se vypočítané souřadnice události.

Provedení testu: OK.

Poznámky Nelze nastavovat milisekundy.

T_8 **Zobrazení detailu události**

Účel: Ověření, zda lze provést funkční požadavek F_10.

Čas: 1 min.

Podmínky: Existence události.

Kroky:

1. Na stránce D.14 vyberu událost.
2. Zkontroluji, že se mi zobrazí detail události včetně grafu.

Očekávané výsledky:

a) Zobrazí se detail o událostech včetně grafu.

Provedení testu: OK.

5.2 Možné úpravy

Z testů, ale také během ukazování funkčnosti prototypu bylo uvedeno několik návrhů na úpravy, konkrétně to je:

- umožnit nastavování startu a konce lokalizace na milisekundy,
- umožnit zobrazování detailu i mimo koncové stavy,
- při vytvoření nebo změně názvu třídy ihned aktualizovat seznam v D.16,

- přidat filtrace pro všechny seznamy,
- přidat shlukování pro detekci tříd u neklasifikovaných dat,
- přidat historii stavů,
- přidat další možnost klasifikace pomocí neuronových sítí,
- upravit vzhled klientské aplikace,
- přidat stránkování do seznamů,
- upravit generování grafu vstupních dat tak, aby šlo zobrazovat i objemná data.

5.3 Závěr testování

Testováno bylo pouze na dvou vstupních množinách odlišných dat a je možné, že u nových množin dat bude nutné udělat další úpravy, které umožní snadnější užívání a zároveň zamezí uživateli zadávat nesmyslné vstupy.

Při práci s druhou množinou dat, která byla velice objemná a žádala si redukci dimenze bylo dočasně vypnuto zobrazování grafu vstupních dat, protože bylo velice náročné graf vygenerovat.

Kapitola 6

Experimentální část

Následující kapitola se věnuje experimentování s daty. Část experimentů je realizována mimo naimplementované serverové metody a slouží pouze k nalezení nejvhodnějšího předzpracování a klasifikace pro experimentální data.

6.1 Předzpracování

Vstupní signály událostí jsou různě dlouhé a je tedy důležité tato data správně předzpracovat. Nejdelší data obsahují 39281 dimenzí a všechna data se tedy musí této dimenzi přizpůsobit. Samotná klasifikace a přezpracování bez redukce dimenze trvala přes 3 hodiny, zatímco předzpracování dat do dimenze menší než 512 trvala méně než minutu. Na tabulce jsou vidět výsledky. Při testování bylo pracováno s redukcemi dimenze PCA a LDA a se třemi typy dat - raw, diff a jejich kombinace, která je v tabulce 6.1 reprezentována jako both. Jako nejvhodnější předzpracovaná data se jasně jeví data redukovaná pomocí LDA s dimenzí 3, která obsahují všechny informace o dimenzích a lze z nich zobrazit data zpět do všech 39281 dimenzí. Pro další experimenty se tedy bude pracovat s těmito předzpracovanými daty, konkrétně hodnotou diff.

6.2 Klasifikace

Částečně je testování klasifikací se základními parametry obsazeno v tabulce 6.1 a následující kapitola se věnuje takovým metodám, u kterých lze definovat parametry. Cílem je najít nejvhodnější metodu a parametry, respektivě strukturu neuronových sítí pro klasifikaci trénovacích dat.

Název metody	AVG	nb	dt	knn	lr	rf	svm
LDA(3) diff	99.49	99.79	99.38	99.58	99.28	99.38	99.49
LDA(3) raw	99.42	98.97	99.69	99.79	99.58	99.59	98.87
LDA(2) raw	97.28	98.46	98.57	99.17	90.30	98.87	98.26
LDA(1) diff	96.4	99.59	99.89	99.59	79.49	99.89	99.9
LDA(2) diff	96.38	99.59	99.89	99.59	79.49	99.79	99.9
LDA(1) both	96.33	99.38	99.79	99.58	79.49	99.89	99.79
LDA(2) both	96.33	99.38	99.79	99.58	79.49	99.89	99.79
LDA(3) both	96.29	99.38	99.79	99.58	79.49	99.69	99.79
LDA(1) raw	88.39	87.12	90.28	93.36	79.69	90.99	88.87
PCA(128) diff	78.63	65.58	85.13	85.82	71.54	94.48	69.19
PCA(16) diff	78.2	66.18	81.54	86.83	74.70	90.73	69.19
PCA(32) diff	78.18	63.33	84.51	85.42	74.28	92.34	69.19
PCA(8) diff	78.1	64.64	81.76	88.26	74.59	90.12	69.19
PCA(4) diff	77.45	65.88	82.04	86.24	74.28	87.06	69.19
PCA(64) diff	76.95	58.05	85.32	82.86	73.18	93.09	69.19
PCA(8) raw	76.49	68.65	82.17	82.28	74.78	81.88	69.19
PCA(8) both	76.48	68.75	81.46	82.28	74.78	82.38	69.19
bez redukce raw	76.08	63.80	78.70	83.50	79.10	82.19	69.19
PCA(4) raw	75.74	72.21	79.21	82.28	71.60	79.94	69.19
PCA(4) both	75.62	72.21	77.67	82.28	71.60	80.76	69.19
PCA(1) diff	74.78	78.89	73.03	80.23	74.59	73.34	68.57
PCA(128) raw	74.54	57.20	77.36	83.91	75.16	84.40	69.19
PCA(128) both	74.07	55.65	78.06	83.71	75.48	82.28	69.19
PCA(64) raw	73.68	54.12	79.41	83.81	71.52	83.98	69.19
PCA(64) both	73.51	52.79	80.12	83.71	71.11	84.12	69.19
PCA(16) both	73.31	46.54	81.56	83.10	74.37	85.05	69.19
PCA(32) raw	73.2	46.95	81.87	83.70	72.63	84.82	69.19
PCA(16) raw	73.13	46.64	81.56	83.10	74.37	83.93	69.19
PCA(32) both	73.03	45.63	82.49	83.60	72.63	84.63	69.19
PCA(2) diff	73.01	56.40	76.52	81.95	73.67	80.32	69.19
PCA(2) both	71.88	56.32	75.66	78.40	74.40	77.29	69.19
PCA(2) raw	71.76	56.32	75.24	78.40	74.40	76.98	69.19
PCA(1) both	68.21	68.43	63.56	70.81	73.08	64.17	69.19
PCA(1) raw	68.14	68.43	63.56	70.81	73.08	63.76	69.19

Tabulka 6.1: Měření výsledků předzpracovaní seřazené podle největší průměrné přesnosti klasifikací. Hodnoty jsou uvedeny v procentech.

6.2.1 SVM

Tato klasifikace obsahuje nejvíce parametrů a je tedy vhodné pokusit se u ní najít takové parametry, které jsou pro experimentální data nejlepší. Jako parametry, které nejvíce ovlivňují klasifikaci, byly zvoleny:

- C - hodnota trestu za špatnou klasifikaci,
- kernel - specifikuje typ jádra,
- degree - stupeň polynomiální funkce (pouze při polynomiálním typu jádra).

Jak ukazuje tabulka 6.3, parametry SVM nemají moc velký vliv na výsledek pro tato data. Podařilo se ovšem najít lepší parametry než jsou základní, která jsou skoro až na konci tabulky (C = 1, kernel = linear). Zároveň tyto ideální parametry mají úspěšnost stejně velkou jako dosud nejúspěšnější metoda nb v tabulce 6.1.

C	Kernel	Degree	Úspěšnost (%)
1	poly	4	99.7959
10	poly	4	99.7959
100	poly	4	99.7959
1000	poly	4	99.7959
1	poly	3	99.6938
1	poly	5	99.6938
10	poly	3	99.6938
10	poly	5	99.6938
100	poly	3	99.6938
100	poly	5	99.6938
1000	poly	3	99.6938
1000	poly	5	99.6938
1	poly	2	99.5918
10	poly	2	99.5918
100	poly	2	99.5918
1000	poly	2	99.5918
10	rbf	-	99.5918
100	rbf	-	99.5918
1000	rbf	-	99.5918
1	rbf	-	99.4897
1	linear	-	99.3877
10	linear	-	99.3877
100	linear	-	99.3877
1000	linear	-	99.3877
1	poly	1	99.3877
10	poly	1	99.3877
100	poly	1	99.3877
1000	poly	1	99.3877
1	poly	10	88.9795
10	poly	10	88.9795
100	poly	10	88.9795
1000	poly	10	88.9795

Tabulka 6.2: Měření průměrné úspěšnosti klasifikací SVM s různými parametry seřazené podle největší úspěšnosti klasifikace.

6.2.2 Random forrest

Tato klasifikace obsahuje pouze dva parametry, ale mohou mít velký vliv na klasifikaci:

- n_estimators - počet stromů,

- criterion - funkce, která vyhodnocuje kvalitu rozdělení.

Podobně jako u SVM se ukazuje, že parametry nemají příliš velký vliv na vyhodnocení. Při volbě nejlepších parametrů vychází opět úspěšnost, které dosáhl i nb a svm s nejvhodnějšími parametry.

n_estimators	criterion	Úspěšnost (%)
12	gini	99.7959
14	gini	99.7959
6	gini	99.6938
7	gini	99.6938
8	gini	99.6938
60	gini	99.6938
15	entropy	99.6938
17	entropy	99.6938
19	entropy	99.6938
16	gini	99.5918
17	gini	99.5918
18	gini	99.5918
20	gini	99.5918
40	gini	99.5918
50	gini	99.5918
100	gini	99.5918
13	gini	99.4897
13	entropy	99.4897
7	entropy	98.8775

Tabulka 6.3: Měření průměrné úspěšnosti klasifikací random forrest s různými parametry seřazené podle největší úspěšnosti klasifikace.

6.2.3 Neuronové sítě

Jako poslední metoda k experimentování jsou neuronové sítě. U této metody jsou očekávány výrazně jiné výsledky. U neuronových sítí ovšem budeme zkoumat jejich strukturu na rozdíl od parametrů. Popsány budou pouze skryté vrstvy a to následovně:

- Dense(x) - klasická skrytá vrstva umělé neuronové sítě, kde x symbolizuje počet neuronů,
- Dropout(x) - vrstva, která v poměru x dat nastaví hodnoty na 0, pomáhá pro redukci overfittingu,
- LSTM(x) - vrsta LSTM, která vrací x výstupů.

Pro testování byly zvolené následující struktury (pořadí je zachováno):

- ann1 - Dense(50), Dropout(0,2), Dense(15), Dropout(0,2), Dense(10), Dropout(0,2).
- ann2 - Dense(50), Dense(15), Dense(10).
- ann3 - Dense(5), Dropout(0,2), Dense(3), Dropout(0,2), Dense(8), Dropout(0,2), Dense(12), Dropout(0,2), Dense(3), Dropout(0,2), Dense(6), Dropout(0,2), Dense(7), Dropout(0,2), Dense(4), Dropout(0,2), Dense(8), Dropout(0,2), Dense(6), Dropout(0,2).
- ann4 - Dense(1000), Dropout(0,2).
- ann5 - Dense(5), Dropout(0,2).
- ann6 - Dense(10), Dropout(0,2).
- ann7 - Dense(5), Dropout(0,2), Dense(5), Dropout(0,2), Dense(5), Dropout(0,2).
- rnn1 - LSTM(50), Dropout(0,2), LSTM(50), Dropout(0,2), LSTM(50), Dropout(0,2).
- rnn2 - LSTM(50), LSTM(50), LSTM(50).
- rnn3 - LSTM(8), Dropout(0,2).
- rnn4 - LSTM(5), Dropout(0,2), LSTM(7), Dropout(0,2), LSTM(4), Dropout(0,2), LSTM(2), Dropout(0,2), LSTM(6), Dropout(0,2), LSTM(8), Dropout(0,2), LSTM(3), Dropout(0,2).
- rnn5 - LSTM(1000), Dropout(0,2).
- rnn6 - LSTM(10), Dropout(0,2), Dense(3), Dropout(0,2), LSTM(3), Dropout(0,2).

- rnn7 - LSTM(10), Dropout(0,2), Dense(3), Dropout(0,2).
- rnn8 - LSTM(3), Dropout(0,2), LSTM(2), Dropout(0,2), LSTM(50), Dropout(0,2).

Typ NN	Úspěšnost (%)
ann1	99.3877
ann2	98.7755
ann3	76.3265
ann4	99.3877
ann5	45.4081
ann6	99.7959
ann7	86.9387
rnn1	99.3877
rnn2	99.5918
rnn3	99.3877
rnn4	92.1428
rnn5	99.3877
rnn6	98.8775
rnn7	99.5918
rnn8	99.2857

Tabulka 6.4: Měření průměrné úspěšnosti neuronových sítí různých struktur.

V experimentu s neuronovými sítěmi se ukázalo, že struktura hraje větší roli než parametry u běžných klasifikací. Jako nejúspěšnější se ukázala jednoduchá umělá neuronová síť s jednou skrytou vrstvou o 10 neuronech. Jako tři nejméně úspěšné neuronové sítě byly takové, které obsahovaly v první skryté vrstvě pouze 5 neuronů. Je tedy pravděpodobné, že takto malý počet neuronů není pro tento problém vhodný. Rekurentní sítě už takový rozdíl ve výsledcích neměly a jediná síť, která se dostala pod úspěšnost 99% byla rnn4, která obsahovala 7 LSTM vrstev. Je tedy zřejmé, že takto velký počet vrstev není vhodný.

6.3 Lokalizace

K lokalizaci byla využita jiná experimentální data, která obsahovala pouze jeden typ události a byla měřena v 75 metrů dlouhé chodbě s 5 mikrofony. Data bohužel neobsahovala souřadnice vzniku události a tak bylo možné pouze porovnat výsledky, které byly vyhodnoceny naimplementovanou lokalizační metodou. Vzhledem k tomu, že se pro lokalizaci vždy používá pouze dvojice mikrofonů, tak bylo možné porovnat výsledky těchto dvojic. K identifikaci stejné události slouží informace o startu události - mezi jednotlivými událostmi

je minimálně 5 sekundový interval. Pozice mikrofonů byly následující (použito číslování stejné jako v tabulce 6.5):

- 1 - x = 0,
- 2 - x = 0,
- 3 - x = 25,
- 4 - x = 50,
- 5 - x = 73.

Při lokalizaci je možné lokalizovat pouze událost, která nastala mezi mikrofóny.

Jak je vidět v tabulce 6.5, tak lokalizace má problémy s krajními hodnotami jednotlivých dvojic mikrofonů. Mimo tyto problematické hodnoty má lokalizace u jednotlivých událostí odchylky v jednotkách metrů. Problém s lokalizací také nastává tehdy, když vznikla událost dále od mikrofónu. Na obrázku 6.2 je vidět velká odchylka v začátku události. Tato odchylka může být zapříčeněna detekci událostí nebo kvalitou mikrofonu. Dále je také nutné podotknout, že doba trvání události je často odlišná (viz. 6.1). Tento experiment byl omezen vstupními daty, která neobsahovala očekávanou hodnotu.

	id integer	event_start timestamp with time zone	event_end timestamp with time zone
1	17098	2017-10-12 17:49:07.962108+02	2017-10-12 17:49:09.398101+02
2	17069	2017-10-12 17:49:08.008179+02	2017-10-12 17:49:09.39682+02
3	17128	2017-10-12 17:49:08.031212+02	2017-10-12 17:49:09.903642+02
4	17027	2017-10-12 17:49:08.091369+02	2017-10-12 17:49:08.970628+02
5	17026	2017-10-12 17:49:08.10033+02	2017-10-12 17:49:09.625915+02

Obrázek 6.1: Ukázka vymezení začátku a konce jedné události na více mikrofonech.

	microphone_id integer	event_start timestamp with time zone	event_end timestamp with time zone
1	1	2017-10-12 17:48:54.445579+02	2017-10-12 17:48:55.486102+02
2	2	2017-10-12 17:48:54.627318+02	2017-10-12 17:48:55.020235+02
3	2	2017-10-12 17:49:08.091369+02	2017-10-12 17:49:08.970628+02
4	1	2017-10-12 17:49:08.10033+02	2017-10-12 17:49:09.625915+02
5	1	2017-10-12 17:50:43.963852+02	2017-10-12 17:50:44.968535+02
6	2	2017-10-12 17:50:44.13151+02	2017-10-12 17:50:44.471948+02

Obrázek 6.2: Ukázka rozdílu začátku u vzdálených událostí pro mikrofonů 1 a 2.

ID	1,3	1,4	1,5	2,3	2,4	2,5	3,4	3,5	4,5
1	2.5	0.4	N/A	2.5	0.4	N/A	N/A	N/A	N/A
2	2.1	2.0	2.3	2.5	2.5	2.7	N/A	25.2	50.2
3	5.9	6.5	3.8	6.5	7.1	4.4	25.6	N/A	N/A
4	5.7	5.9	5.5	5.9	6.1	5.7	25.2	N/A	N/A
5	10.3	11.0	10.8	10.8	11.4	11.2	25.6	25.4	N/A
6	10.1	10.5	10.4	11.2	11.6	11.4	25.4	25.2	N/A
7	15.2	12.4	15.9	15.4	12.7	16.1	N/A	25.6	53.4
8	15.2	15.0	15.0	16.9	16.7	16.7	N/A	N/A	50.0
9	20.1	20.3	20.5	21.6	21.8	22.0	25.1	25.4	50.2
10	21.6	21.8	22.0	22.4	22.6	22.9	25.1	25.4	50.2
11	N/A	27.9	28.2	N/A	26.0	26.3	26.0	26.3	50.2
12	24.8	25.6	25.6	N/A	27.5	27.5	25.8	25.8	50.0
13	N/A	31.1	29.7	N/A	30.7	29.2	30.2	28.8	N/A
14	N/A	29.6	31.1	N/A	30.9	32.4	29.0	30.5	51.5
15	24.8	34.9	35.2	N/A	46.8	47.1	35.1	35.4	50.2
16	24.8	40.0	40.3	N/A	43.8	44.1	40.2	40.5	50.2
17	N/A	43.4	43.7	N/A	N/A	N/A	40.2	40.5	50.2
18	N/A	47.9	47.9	N/A	46.4	46.4	45.1	45.1	50.0
19	N/A	45.3	45.6	N/A	45.5	45.8	44.9	45.1	50.2
20	N/A	N/A	51.5	N/A	N/A	51.1	49.8	49.6	N/A
21	N/A	48.7	49.6	N/A	N/A	66.4	48.7	49.6	50.8
22	N/A	N/A	55.3	N/A	N/A	58.7	49.3	54.7	55.3
23	24.8	49.3	54.7	N/A	N/A	72.7	49.5	54.9	55.3
24	N/A	N/A	60.4	N/A	N/A	65.9	N/A	60.0	59.8
25	24.7	49.1	59.4	N/A	N/A	61.0	49.3	59.6	60.2
26	N/A	N/A	65.5	N/A	N/A	65.5	49.5	64.4	64.8
27	23.7	49.3	64.7	N/A	N/A	71.7	N/A	65.9	65.3
28	N/A	N/A	N/A	N/A	N/A	N/A	49.5	69.3	69.7
29	N/A	49.8	70.0	N/A	N/A	70.2	49.3	69.5	70.1

Tabulka 6.5: Měření Lokalizací, v prvním řádku je uvedena kombinace mikrofónů oddělená čárkou

6.4 Závěr experimentální části

K experimentu s klasifikacemi byla použita a poskytnuta pouze jedna sada dat, u které se podařilo najít velmi úspěšné klasifikace. Bylo překvapivé, jak moc předzpracování ovlivní celý proces klasifikace. Zároveň se ukázalo, že neuronové sítě dosahují stejných výsledků jako klasické metody strojového učení. Lokalizační metoda bohužel tak úspěšná jako klasifikace nebyla. U této metody dělaly největší problémy krajní hodnoty jednotlivých mikrofonů a také vstupní data, která byla velice omezená.

Závěr

Cílem diplomové práce byly analýza, návrh a tvorba aplikace umožnující třídění událostí a jejich lokalizaci v prostoru zaznamenaných optickými senzory.

V teoretické části, která sloužila k seznámení se s disciplínami strojového učení, jsem mimo strojové učení popsal práci s daty a jejich předzpracování. Ze strojového učení jsem se seznámil s regresními, klasifikačními a shlukovacími metodami a celou kapitolu jsem uzavřel neuronovými sítěmi a popisem nejdůležitějších knihoven pro strojové učení v jazyce Python.

V praktické části jsem se seznámil s problémem, zanalyzoval požadavky, formát vstupu a výstupu. Poté jsem navrhnul, naimplementoval a otestoval klientskou a serverovou aplikaci, která umožňuje zpracování a následnou klasifikaci vstupních dat.

V experimentální části jsem poté hledal optimální konfiguraci klasifikace a předzpracování pro experimentální data. V této části jsem byl nejvíce překvapen tím, že největší rozdíly nastávaly již v typu předzpracování a se správným předzpracováním se hodnoty úspěšnosti klasifikace již moc nelišily. K největší úspěšnosti pro testovací data rozhodně doporučuji u předzpracování redukovat dimenzi dat na dimenzi 3 pomocí LDA a následně pro klasifikaci použít metodu naivního Bayese, K-nejbližších sousedů, náhodný les s 12 nebo 14 stromy a kritériem „gini“ nebo SVM s polynomiálním jádrem a stupněm 4. S těmito metodami se podařilo oklasifikovat testovací data s 99,79% úspěšností.

K návrhu architektury a aktivity, entity a stavových diagramů byl využit Enterprise Architect K vývoji klientské aplikace bylo využito IDE WebStorm a framework Angular. K vývoji serverové části aplikaci bylo využito IDE PyCharm a framework Django s knihovnami scikit-learn, TensorFlow a Keras. Jako úložiště bylo využito databáze PostgreSQL. K Testování naimplementovaných serverových HTTP metod byla využita aplikace Postman. V experimentální části jsem poté využil platformu Anaconda.

Literatura

[1] Korelační a regresní analýza [online]. [Citováno 20. února 2019]. Dostupné z: https://wikisofia.cz/wiki/Korelační_a_regresní_analýza

[2] Polynomial Regression - Towards Data Science [online]. [Citováno 20. února 2019]. Dostupné z: https://towardsdatascience.com/polynomial-regression-bbe8b9d97491

[3] Support Vector Regression [online]. [Citováno 2. března 2019]. Dostupné z: http://www.saedsayad.com/support_vector_machine_reg.htm

[4] Decision Trees: An Overwier| Aunalytics [online]. [Citováno 5. května 2019]. Dostupné z: https://www.aunalytics.com/2015/01/30/decision-trees-an-overview/

[5] Raschka, S.: *Python Machine Learning*. Livery Place, 35 Livery Street, Birmingham B3 2PB, UK: Packt Publishing Ltd., 2015, ISBN 978-1-78355-513-0.

[6] svm.pdf [online]. [Citováno 2. března 2019]. Dostupné z: http://www.trilobyte.cz/downloadfree/qcemanual/svm.pdf

[7] Hadelin de Ponteves, K. E.: Machine Learning A-Z: Hands-On Python R In Data Science | Udemy [online]. [Citováno 20. února 2019]. Dostupné z: https://www.udemy.com/machinelearning/learn/v4/content

[8] What is Deep Learning? [online]. [Citováno 2. března 2019]. Dostupné z: https://machinelearningmastery.com/what-is-deep-learning/

[9] Fundamentals of Deep Learning - Activation Functions and their use [online]. [Citováno 2. března 2019]. Dostupné z: https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/

[10] A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way [online]. prosinec 2018, [Citováno 3. března 2019]. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[11] An Introduction to Recurrent Neural Networks - Explore Artificial Intelligence - Medium [online]. květen 2018, [Citováno 3. března 2019]. Dostupné z: https://medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912

[12] A Beginner's Guide to LSTMs and Recurrent Neural Networks | Skyrmind [online]. [Citováno 3. března 2019]. Dostupné z: https://skymind.ai/wiki/lstm

[13] Matplotlib: Python plotting[online]. [Citováno 3. března 2019]. Dostupné z: https://matplotlib.org/

[14] IDC: Expect 175 zettabytes of data worldwide by 2025 | Network World [online]. [Citováno 8. května 2019]. Dostupné z: https://www.networkworld.com/article/3325397/idc-expect-175-zettabytes-of-data-worldwide-by-2025.html

[15] HAN, J. a. M. K.: *Data mining: concepts and techniques. 3rd ed.* Englewood Cliffs: Elsevier, 2011, ISBN 978-0-12-381479-1.

[16] WITTEN, I. H. a. E. F.: *Data mining: practical machine learning tools and techniques. 2nd ed.* Boston, MA: Morgan Kaufman, 2015, ISBN 0-12-088407-0.

[17] Decision Tree Regression [online]. [Citováno 20. února 2019]. Dostupné z: https://www.saedsayad.com/decision_tree_reg.htm

[18] Kernel Functions-Introduction to SVM Kernel Examples - DataFlair[online]. [Citováno 8. května 2019]. Dostupné z: https://data-flair.training/blogs/svm-kernel-functions/

[19] Zikmund, B. T.: Afiliace uzlu na bipartitnıch grafec. *Problems Information Transmission*, 2018: str. 20.

[20] What is Python? Executive Summary | Python.org [online]. [Citováno 3. března 2019]. Dostupné z: https://www.python.org/doc/essays/blurb/

[21] scikit-learn: machine learning in Python[online]. [Citováno 3. března 2019]. Dostupné z: https://scikit-learn.org/

[22] NumPy [online]. [Citováno 3. března 2019]. Dostupné z: http://www.numpy.org/

[23] TensorFlow [online]. [Citováno 3. března 2019]. Dostupné z: https://www.tensorflow.org/

[24] Home - Keras Documentation [online]. [Citováno 3. března 2019]. Dostupné z: https://keras.io/

Příloha **A**

Seznam použitých zkratek

NB Naive Bayes (Naivní Bayes)

SVM Support vector machine

SVR Support vector regression

RF Random forrest (Náhodný les)

DT Decision tree (Rozhodovací strom)

KNN K-nearest neighbor (K-nejbližších sousedů)

DWH Data warehouse (Datový sklad)

TP True positive (Správně pozitivní)

FN False negative (Špatně negativní)

TN True negative (Správně negativní)

FP False positive (Špatně pozitivní)

ACC Accuracy (Přesnost)

ERR Error (Chyba)

PRE Precision (Určitost)

REC Recall

FPR False positive rate

NN Neural network (Neuronová síť)

ANN Artificial neural network (Umělá neuronová síť)

CNN Convolutional neural network (Konvoluční neuronová síť)

RNN Recurent neural network (Rekurentní neuronová síť)

LSTM Long-short-term memory (Dlouhá krátkodobá paměť)

CSV Comma separated value (Čárkou oddělené hodnoty)

CLI Command line interface

GUI Graphical user interface (Grafické rozhraní)

Příloha B

Instalační příručka pro serverovou část aplikace

Tato příloha popisuje, jak vytvořit prostředí pro spuštění serverové aplikace.

B.1 Prostředí pro server

Pro spuštění serverové části je nutné mít prostředí se všemi knihovnami. Toho se dá dosáhnout pomocí následujících kroků:

1. Stáhnout a nainstalovat platformu Anaconda (https://www.anaconda.com/).
2. Otevřít Anaconda Prompt.
3. Otevřít složku impl/server na médiu.
4. Spustit příkaz pro nainstalování prostředí django_project:

```
conda env create -f environment.yml
```

B.2 Databáze

Pro běh aplikace je nutné připojení do databáze. Databázi lze lokálně nainstalovat z https://www.postgresql.org/. Pro napojení serveru na databázi je potřeba změnit databázové nastavení v souboru:

```
impl/server/ossp/ossp/settings.py.
```

Poté už je jen potřeba vytvořit datovou strukturu:

1. Otevřít Anaconda Prompt.
2. Spustit příkaz pro přepnutí prostředí na django_project:

```
activate django_project
```

3. Otevřít složku impl/server/ossp.
4. Spustit příkaz pro vytvoření skriptů pro migraci struktury dat:

```
python manage.py makemigrations
```

5. Spustit příkaz pro spuštění skriptů pro migraci struktury dat:

```
python manage.py migrate
```

B.3 Spuštění aplikace

Po nainstalování prostředí a vytvoření datové struktury v databázi lze spustit aplikaci pomocí následujících kroků:

1. Otevřít Anaconda Prompt.
2. Spustit příkaz pro přepnutí prostředí na django_project:

```
activate django_project
```

3. Otevřít složku impl/server/ossp.
4. Spustit příkaz pro spuštění serveru:

```
python manage.py runserver
```

Příloha C

Instalační příručka pro klientskou část aplikace

Tato příloha popisuje, jak vytvořit prostředí pro spuštění klientské aplikace.
Pro spuštění je potřeba mít nainstalované node.js (https://nodejs.org/en/download/).
Poté stačí následovat tyto kroky:

1. Otevřít si příkazovou řádku.
2. Spustit příkaz pro nainstalování Angularu:

```
npm install -g @angular/cli
```

3. Otevřít složku s klientskou aplikaci:

```
impl/client/ossp/
```

4. Spustit příkaz pro nainstalování všech dependencies:

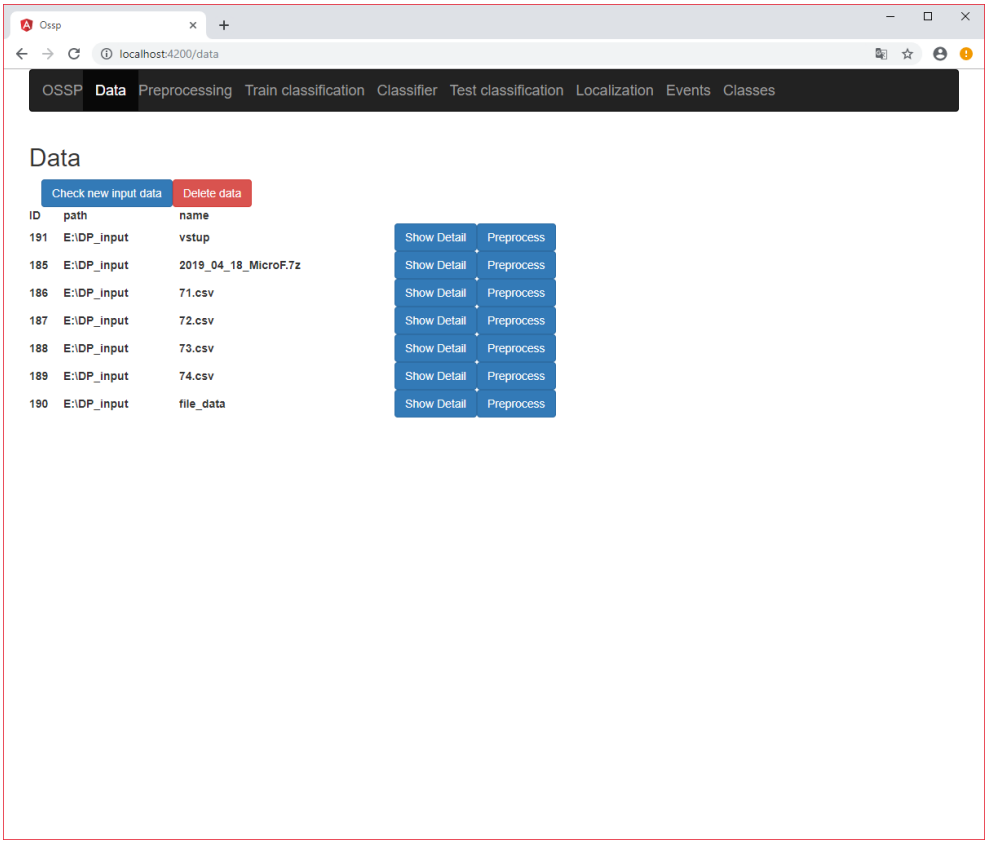
```
npm-install-all
```

5. Spustit aplikaci:

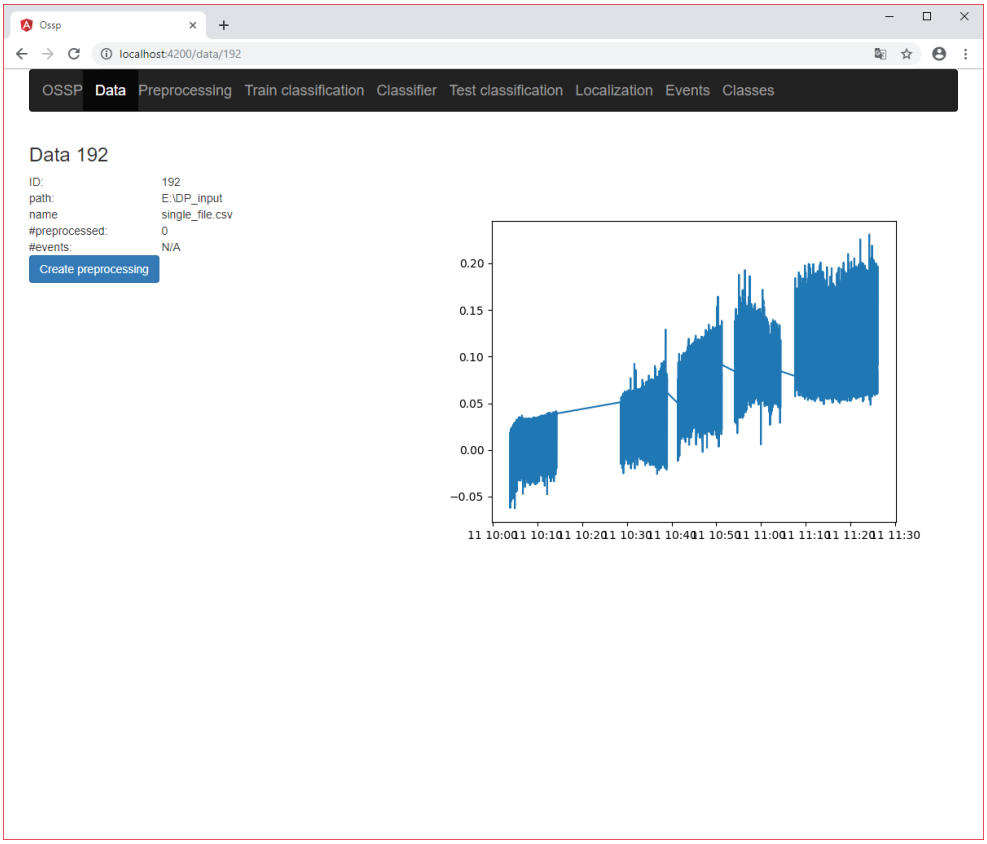
```
ng serve
```

Příloha D

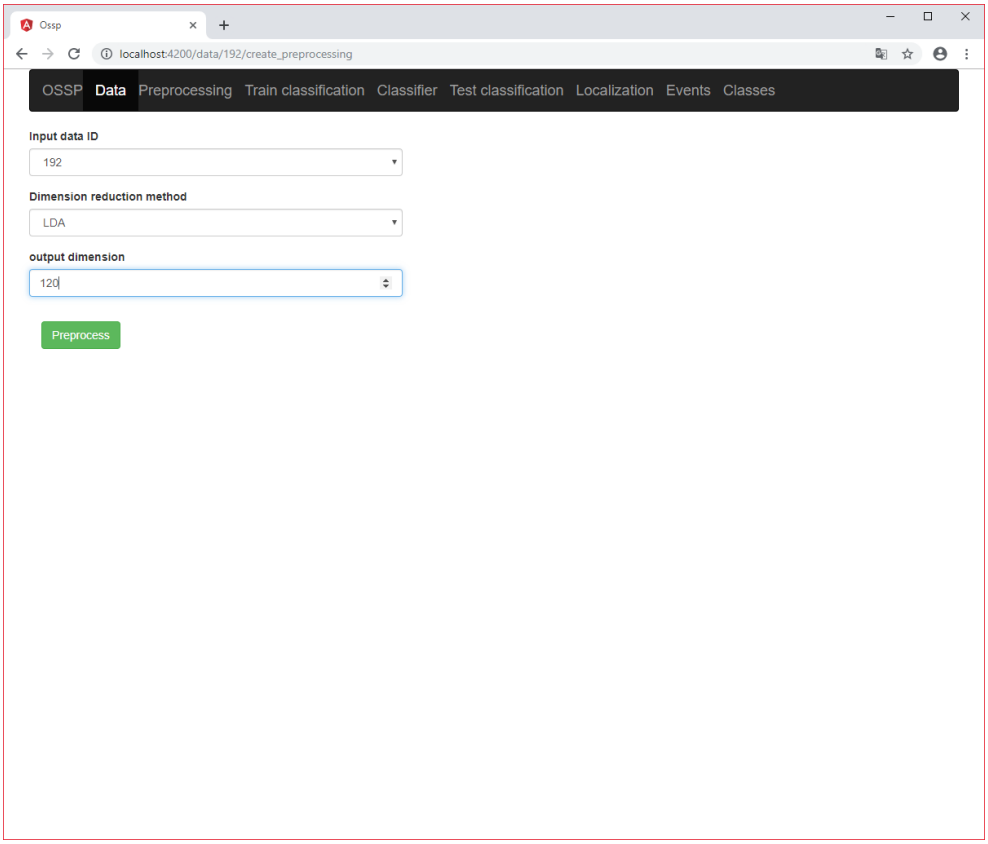
Klientské GUI



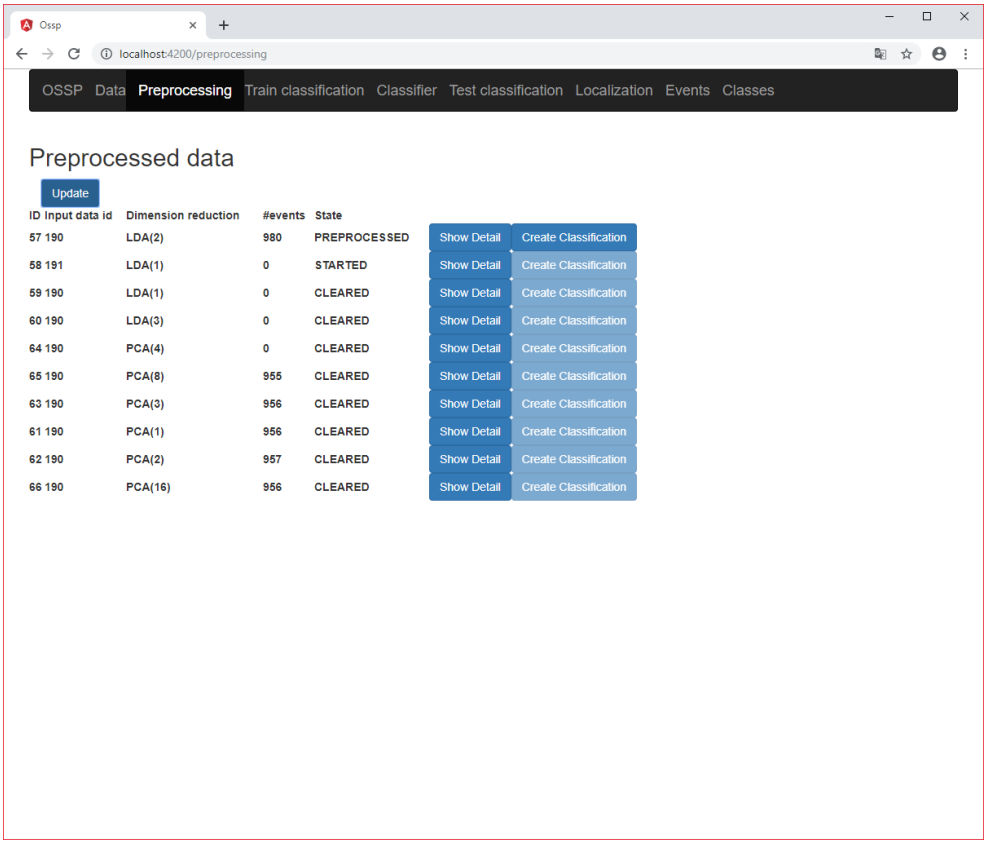
Obrázek D.1: Ukázka klientské stránky pro zobrazení všech existujících vstupních dat.



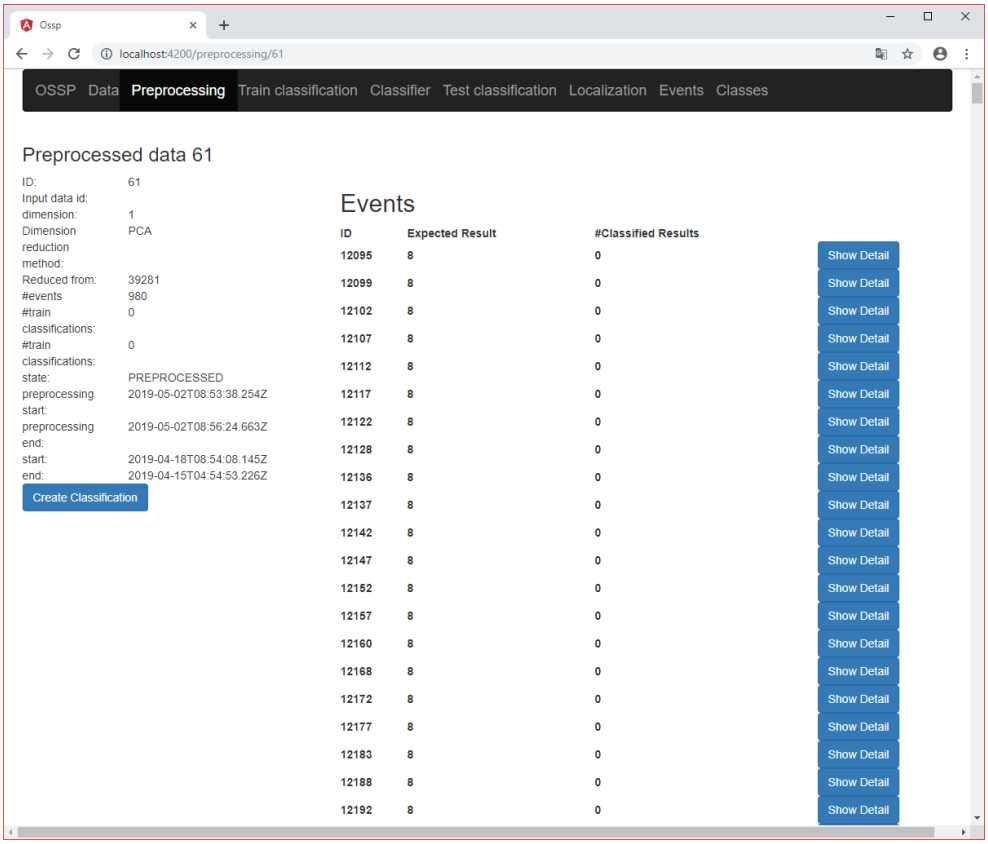
Obrázek D.2: Ukázka klientské stránky pro zobrazení detailu vstupních dat.



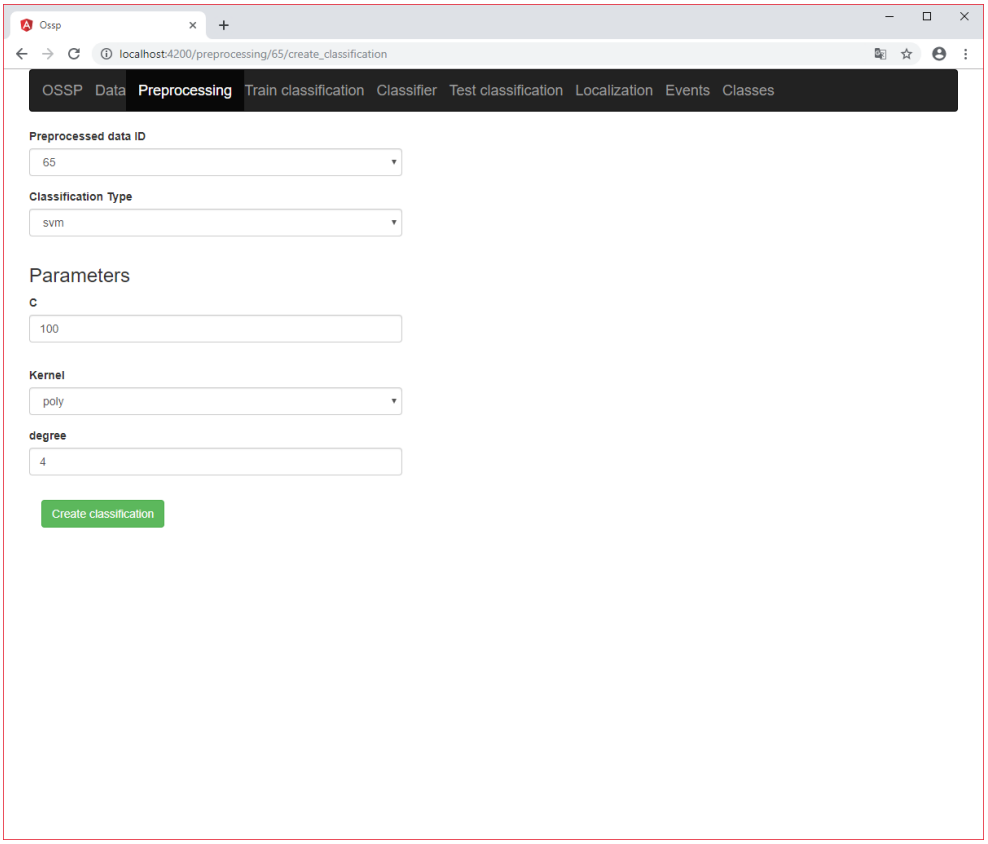
Obrázek D.3: Ukázka klientské stránky pro vytvoření předzpracování dat.



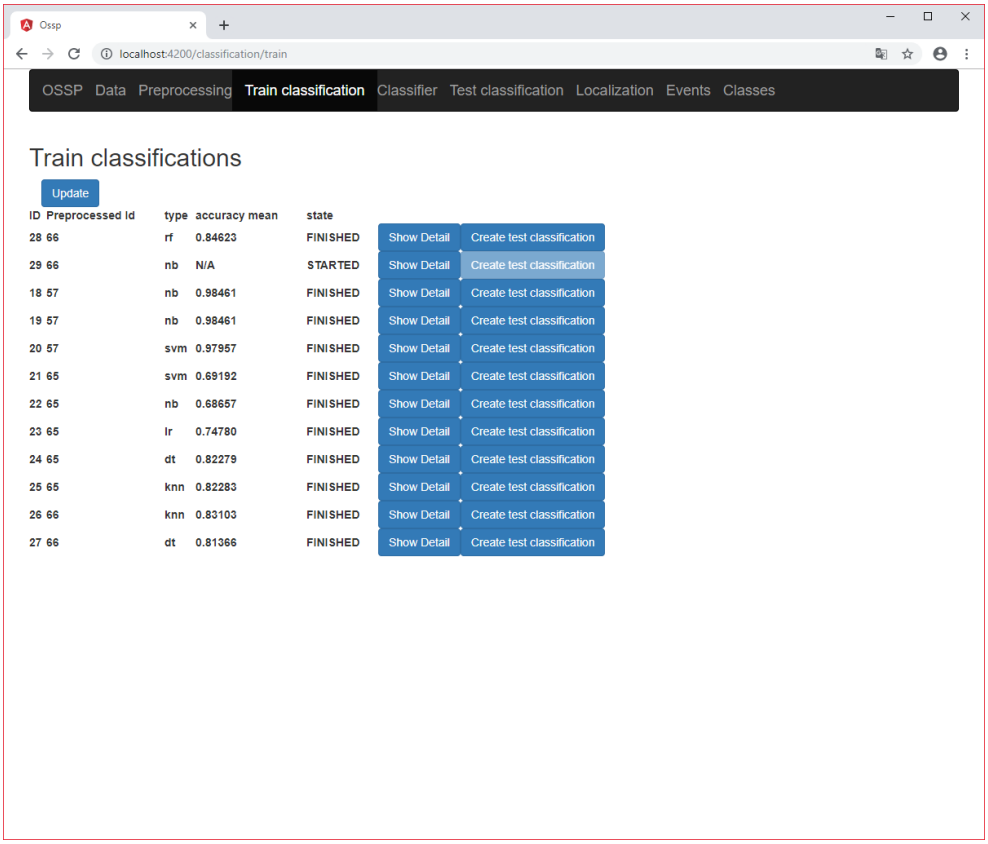
Obrázek D.4: Ukázka klientské stránky pro zobrazení všech existujících předzpracovaných dat.



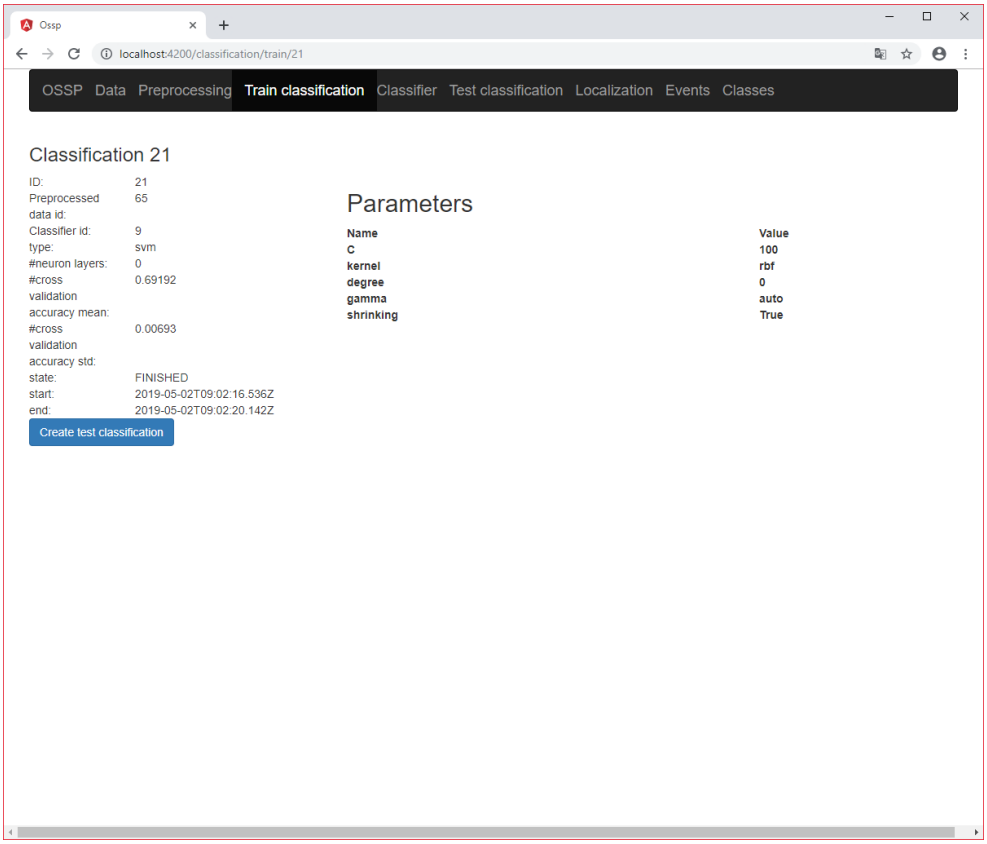
Obrázek D.5: Ukázka klientské stránky pro zobrazení detailu předzpracovaných dat.



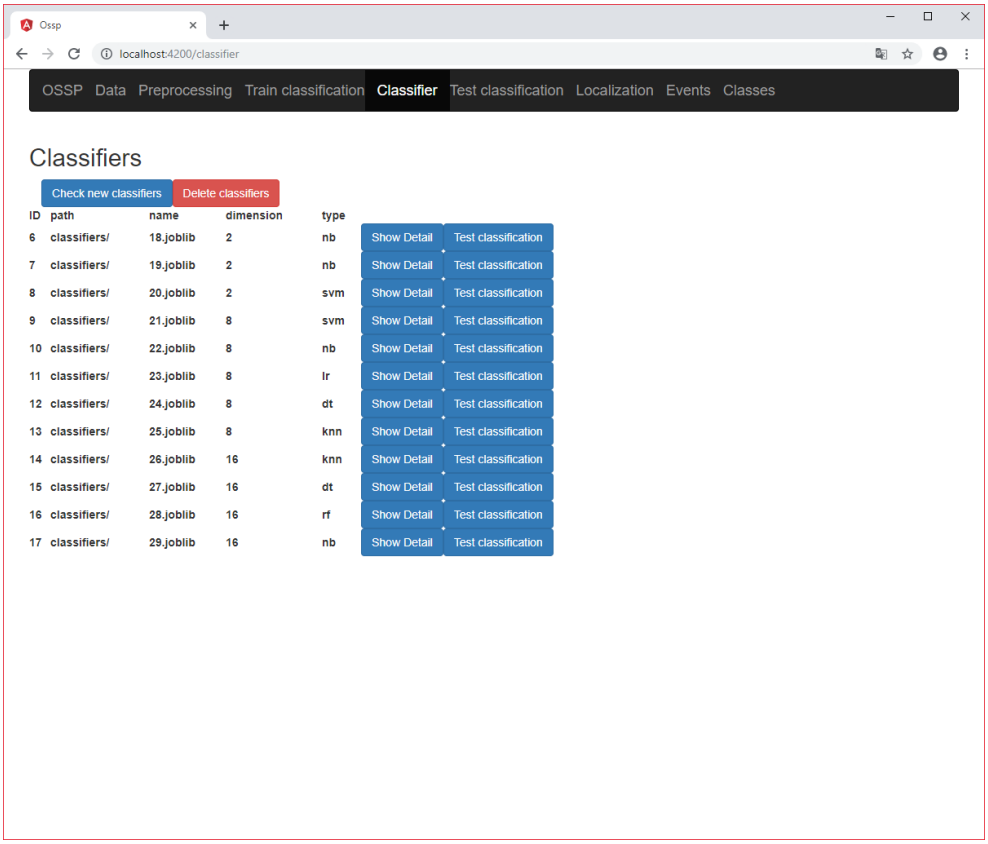
Obrázek D.6: Ukázka klientské stránky pro vytvoření trénovací klasifikace.



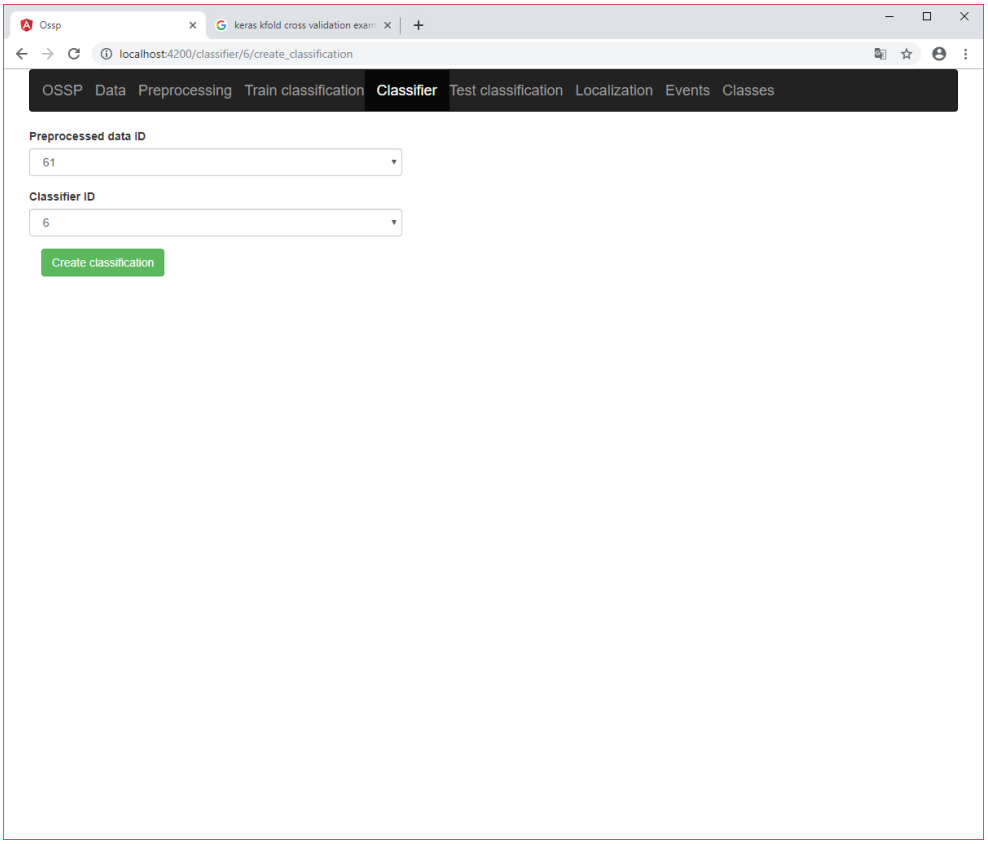
Obrázek D.7: Ukázka klientské stránky pro zobrazení všech existujících trénovacích klasifikací.



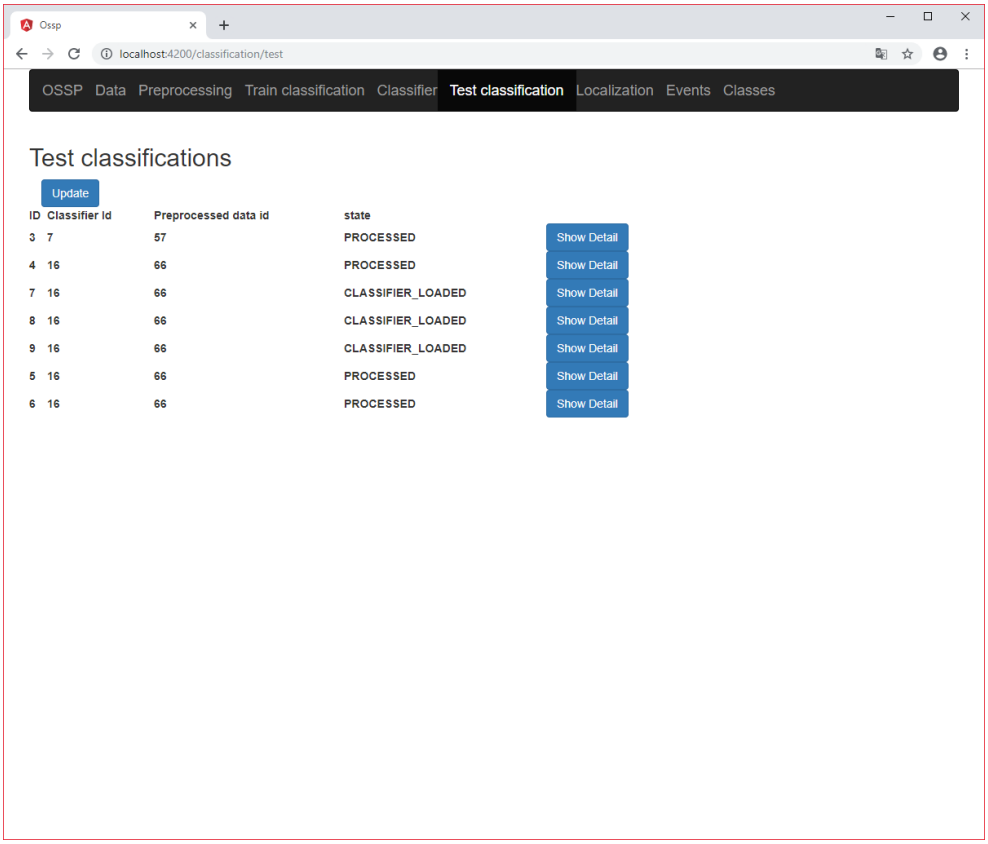
Obrázek D.8: Ukázka klientské stránky pro zobrazení detailu trénovací klasifikace.



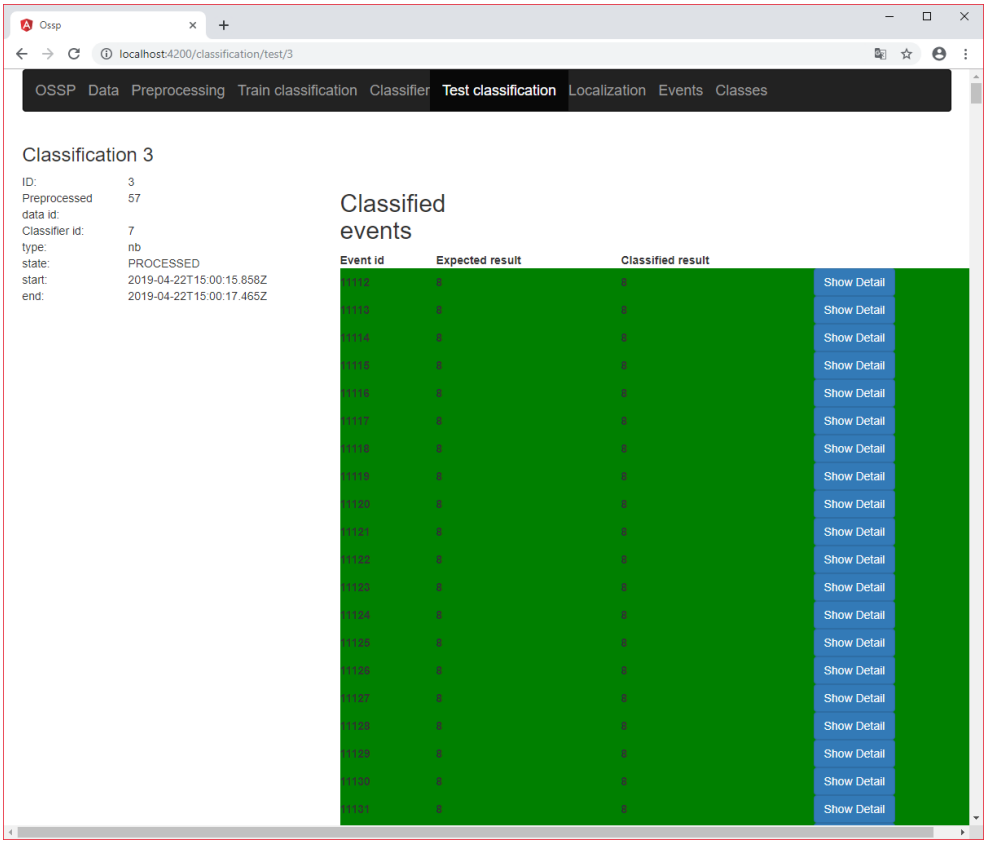
Obrázek D.9: Ukázka zobrazení všech klasifikátorů.



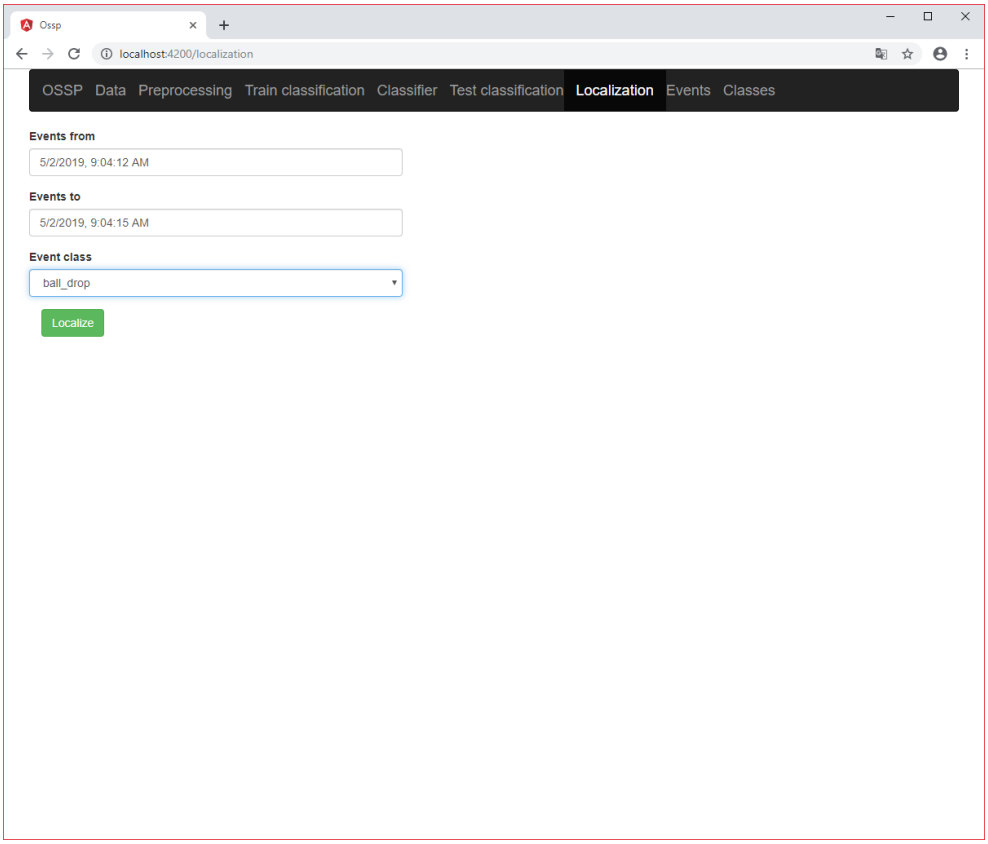
Obrázek D.10: Ukázka klientské stránky pro vytvoření testovací klasifikace.



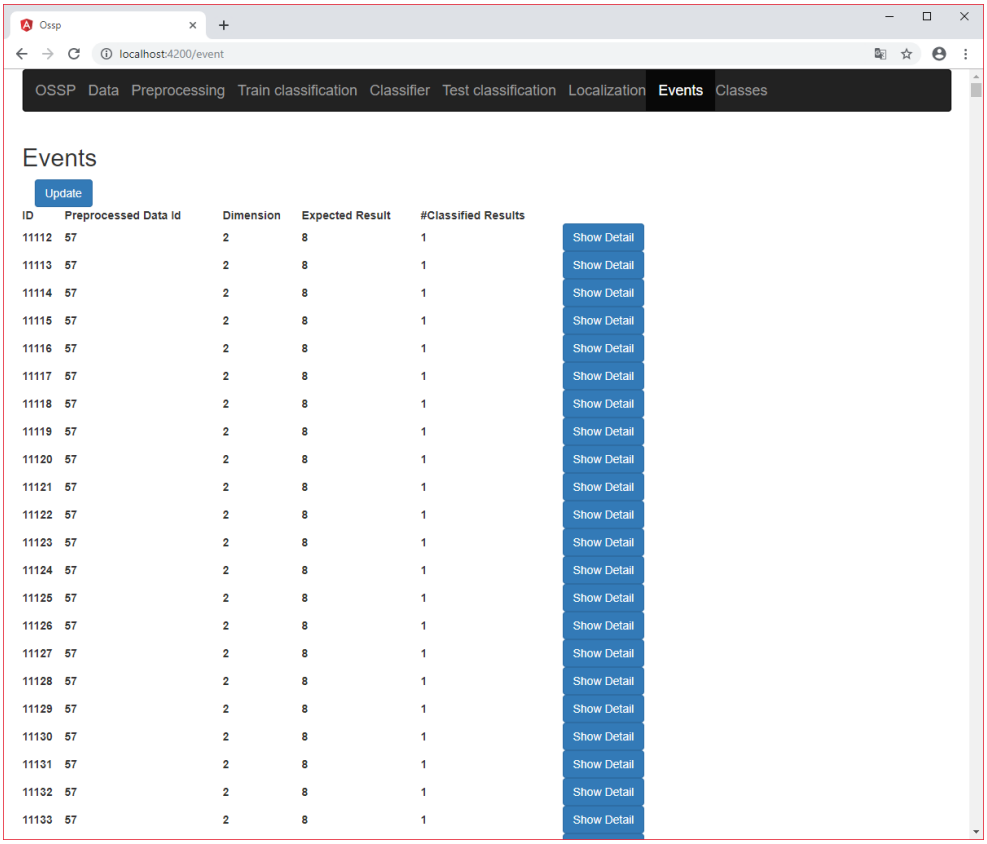
Obrázek D.11: Ukázka klientské stránky pro zobrazení všech existujících testovacích klasifikací.



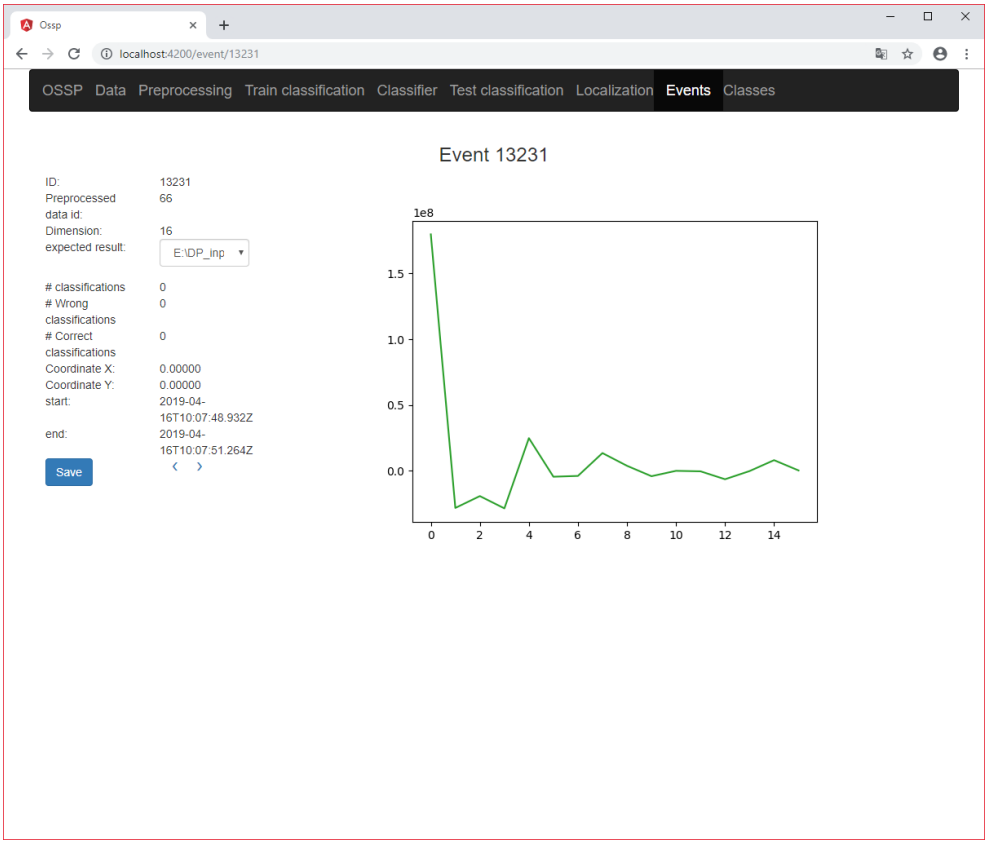
Obrázek D.12: Ukázka klientské stránky pro zobrazení detailu testovací klasifikace.



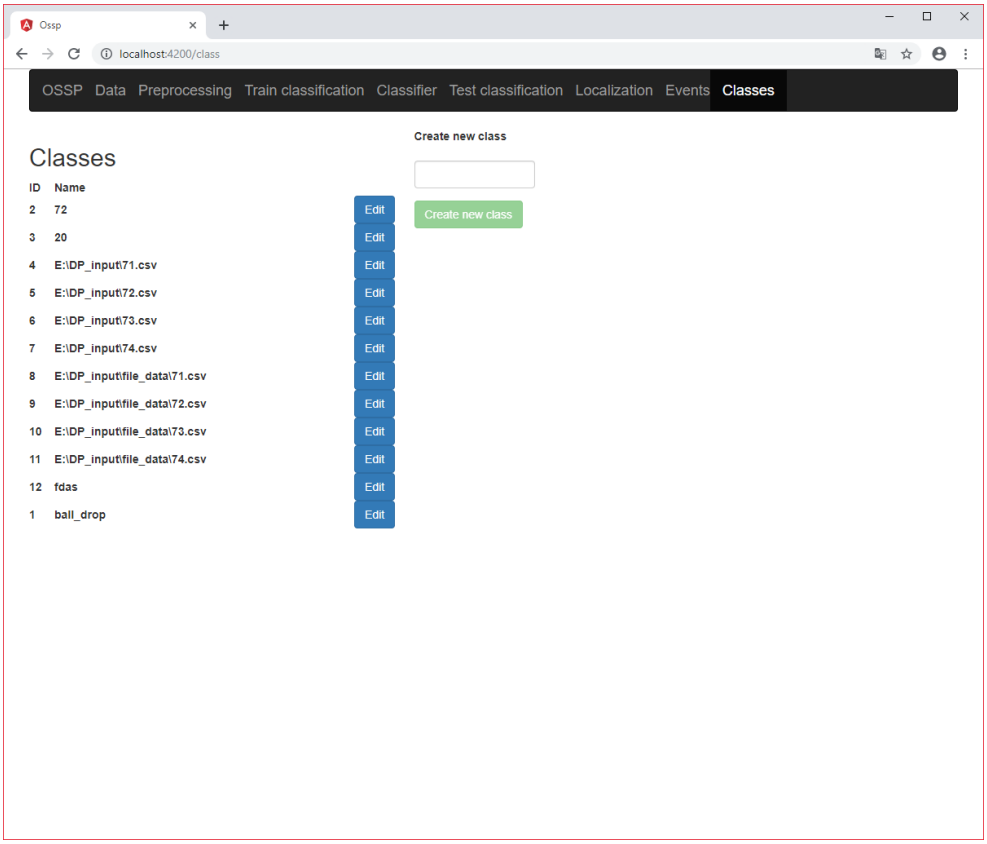
Obrázek D.13: Ukázka klientské stránky pro lokalizaci.



Obrázek D.14: Ukázka klientské stránky pro zobrazení všech existujících událostí.



Obrázek D.15: Ukázka klientské stránky pro zobrazení detailu události.



Obrázek D.16: Ukázka klientské stránky pro zobrazení všech existujících tříd.

Příloha **E**

Obsah přiložené SD karty

```
readme.txt ........................ stručný popis obsahu SD karty
design ........................................ adresář s návrhem
  ossp_first.EAP ................ EA projekt pro první verzi návrhu
  ossp_final.EAP ................ EA projekt pro finální verzi návrhu
  wireframes ................................ adresář s wireframy
impl ........................................ adresář s implementací
  client ................... zdrojové kódy implementace klientské části
  server ................... zdrojové kódy implementace serverové části
thesis ...................................... text práce
  thesis.pdf ........................... text práce ve formátu PDF
  thesis ..................... zdrojová forma práce ve formátu LaTeX
ossp.postman_collection.json postman projekt s voláním serverových
                                                              metod
```