VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

SUPPORT VECTOR MACHINES: TEORIE, APLIKACE A SOFTWAROVÉ IMPLEMENTACE

SUPPORT VECTOR MACHINES: THEORY, APPLICATIONS AND SOFTWARE IMPLEMENTATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Daniil Podtesov
AUTHOR

VEDOUCÍ PRÁCE — Ing. Jakub Kůdela, Ph.D.
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Daniil Podtesov**
Studijní program:	Strojírenství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**Ing. Jakub Kůdela, Ph.D.**
Akademický rok:	2022/23

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Support vector machines: teorie, aplikace a softwarové implementace

Stručná charakteristika problematiky úkolu:

Support vector machines (občas překládaná jako metoda podpůrných vektorů) je jednou z metod strojového učení s učitelem, která se používá převážně pro klasifikaci. Student se bude zabývat studiem potřebné teorie k popsání této metody, řešerší jejich aplikací a analýzou/srovnáním různých software implementací.

Cíle bakalářské práce:

Popis teorie Support vector machines.
Rešerše aplikací.

Analýza a srovnání různých software implementací na vhodném datasetu.

Seznam doporučené literatury:

VAPNIK, V. Statistical Learning Theory. Wiley, 1998.

HASTIE, T., TIBSHIRANI, R., FRIEDMAN, J.H. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2009.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2022/23

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRAKT

Tato bakalářská práce se zabývá problematikou algoritmu strojového učení metody podpůrných vektorů. Práce se zaměřuje na teoretický základ nutný pro pochopení mechanismu fungování této metody, aplikace této metody v reálném životě a implementace této metody v různých softwarech. Jsou vysvětleny pojmy jako jsou optimální separační nadrovina, lineární rozhodovací hranice a podpůrné vektory. Dále jsou popsány různé varianty této metody při použití různých jádrových funkcí. V třetí části práce bylo nejen použito metody pro ukázku toho, jak to funguje, ale i využito několika postupů pro zlepšení výsledků aplikace dané metody.

ABSTRACT

This bachelor thesis deals with the problem of machine learning algorithm called support vector machine learning method. The thesis focuses on the theoretical basis required to understand the mechanism of operation of this method, the application of this method in real life and the implementation of this method in various software. Concepts such as optimal separation superplane, linear decision boundary and support vectors are explained. Furthermore, different variants of this method using different kernel functions are described. In the third part of the paper, the method was not only used to show how it works, but also several techniques were used to improve the results of the application of the method.

KLÍČOVÁ SLOVA

Metoda podpůrných vektorů, separační nadrovina, strojové učení, nadrovina s maximální šírkou, hyperparametry, klasifikace, bias-variance tradeoff, overfitting a underfitting.

KEYWORDS

Support vector machines, separating hyperplane, machine learning, maximal margin hyperplane, hyperparameters, classification, bias-variance tradeoff, overfitting and underfitting.

2023

BIBLIOGRAFICKÁ CITACE

PODTESOV, Daniil. Support vector machines: teorie, aplikace a softwarové implementace. Brno, 2023. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/149570. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Jakub Kůdela.

PODĚKOVÁNÍ

Na tomto místě bych rád poděkoval Ing. Jakubu Kůdelovi, Ph.D. za odborné vedení bakalářské práce, za podnětné podmínky, věnovaný čas, za jeho vstřícnost a pomoc.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následku porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestně právních důsledků.

V Brně dne 20. 5. 2023 ……

Podtesov Daniil

OBSAH

ÚVOD

Je těžko si představit život bez umělé inteligence (angl. *Artificial intelligence*, zkr. AI). Cílem umělé inteligence je automatizovat intelektuální činnost člověka. Face ID v mobilu, online překlad textu, filtrace spamu – jsou to všechno příklady použití umělé inteligence. Čím dál, tím je „počítač“ chytřejší a dnes dokáže detekovat oční onemocnění nebo řídit auto.

Součástí umělé inteligence je strojové učení (angl. *Machine learning*, zkr. ML). Strojové učení je obecný název pro velkou řádu metod, které se používají pro různé aplikace. Rozhodujícím parametrem pro volbu metody strojového učení je typ úlohy, o čemž budeme diskutovat dále v této práci. Strojové učení se zabývá řešením úloh nepřímým způsobem, tzn. „učením“ z velkého množství vyřešených podobných úloh. Velkou část úspěchu při učení tvoří data. Jejich množství, kvalita a reprezentativita jsou zodpovědné za přesný a kvalitní výsledek. Teoretická část metod strojového učení je postavena na základě tří věd: statistika, teorie pravděpodobnosti a lineární algebra.

Strojové učení, jakožto věda ne zcela nová, se začalo rozvíjet ještě ve 20. století, ale zájem o umělou inteligenci roste a v dnešní době efektivita různých modelů je otázkou výkonnosti počítačů, na kterých ty modely jsou naprogramovány a spouštěny.

Tématem práce je metoda podpůrných vektorů (angl. *Support Vector Machines*, dále jen SVM). Bakalářská práce je rozdělena na tři části: teoretickou část, rešerše aplikací a implementace různých softwarů na vhodném datasetu (datová sada nebo shluk/kolekce dat).

Teoretická část bakalářské práce je věnována popisu hlavních typů úloh strojového učení, typů metod a obsahuje také matematické a statistické teorie metody SVM.

V druhé části práce je stručný přehled možných aplikací této metody v reálném životě.

V třetí části bakalářské práce se bude řešit implementace SVM v různých softwarech. Cílem této části je ukázat reálný příklad použití SVM a porovnat výkonnost různých softwarů.

1 TEORETICKÁ ČÁST

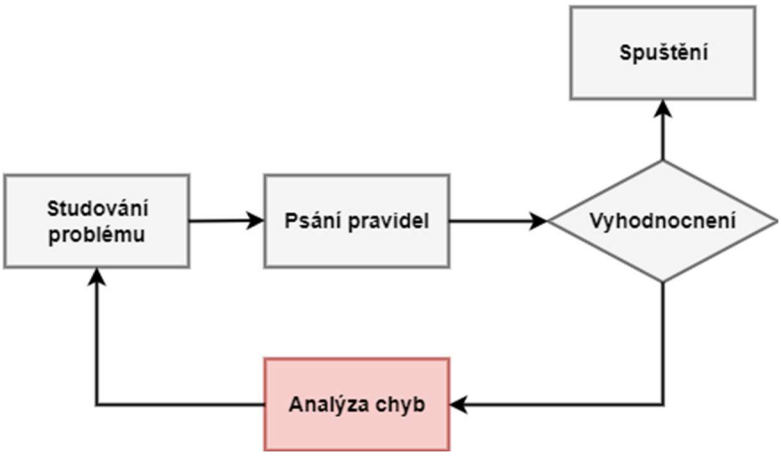
1.1 Základy strojového učení

Člověk již od útlého věku dělá „složité“ logické závěry, protože má přirozenou schopnost se učit. Děti se rychle učí rozeznávat věci a vědí, že pokud dáte lžičku cukru do čaje, čaj bude sladký. Co když ale dáte dítěti za úkol filtrovat spam vašeho e-mailu nebo predikovat cenu akcií na burze? To sotva dokáže dospělý. Strojové učení (dále jen angl. zkratka ML) řeší problémy, které člověk buď nedokáže vyřešit, nebo mu to zabere víc času než počítači. ML metody se dělí na tři hlavní typy:

- Učení s učitelem (angl. *Supervised learning*) – u tohoto typu metod počítač má k dispozici případy, ve kterých jsou dány dvojice „situace, přijaté řešení“ (vstup a požadovaný výstup).
- Učení bez učitele (angl. *Unsupervised learning*) – algoritmus má jenom vstupy a cílem je shlukování těchto vstupů a hledání skrytých závislostí.
- Zpětnovazebné učení (angl. *Reinforcement learning*) – počítač interaguje s dynamicky měnícím se prostředím, ve kterém musí splnit požadovaný úkol.

Každý typ metody je určen pro různé situace. Tyto situace lze také rozdělit na tři hlavní druhy úloh: *klasifikace*, *regrese*, *shlukování*. Tato práce bude zaměřena především na problematiku klasifikace, ale budou tady představeny i základní myšlenky ohledně regrese a shlukování. Klasickým klasifikačním příkladem je detekování spamu. Před použitím ML detekování spamu probíhalo následovně:

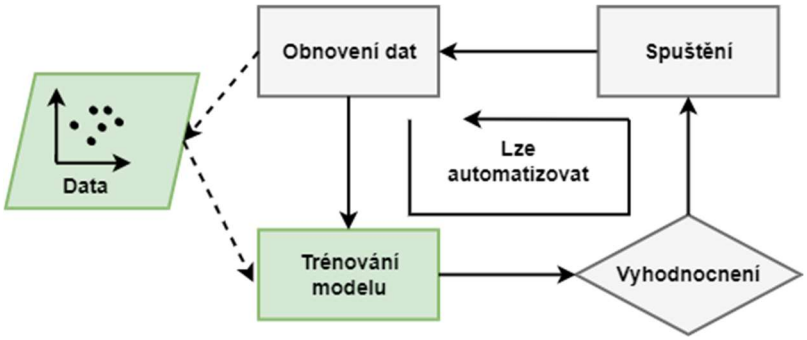
1. V prvním kroku se podíváme na to, jak vypadá obyčejná spam zpráva. Pravděpodobně si všimneme, že takovéto zprávy budou mít nějaké vzorové výrazy nebo slova. Je možné, že jména odesilatelů připadají podobně a jejich emailové adresy jsou vytvořené podle určité šablony.
2. Napíšeme detekční algoritmus pro všechny vzory, kterých jsme si všimli a program bude detekovat zprávu jako „spam“ podle počtu vyskytujících se vzorů.
3. Otestujeme program a zopakujeme kroky 1 a 2 až program je splňuje požadovanou přesnost detekování. [1]



Obr. 1: Detekování spamu tradičním způsobem. [1]

Jelikož problém není triviální, pravděpodobně program bude vypadat jako dlouhý seznam komplexních pravidel, což se těžko udržuje.

Na druhou stranu, detekování spamu založené na ML principech, se bude automaticky učit detekovat neobvyklé vzory ve slovech a výrazech. Takovýto program bude o mnoho řádku kratší, jednodušší na údržbu a s největší pravděpodobností přesnější.



Obr. 2: Detekování spamu pomocí ML modelu. [1]

Takovýto model bude potřeba „naučit“ na dostatečně velkém *datasetu* pro to, aby mohl nejenom efektivně detekovat co je spam, ale i sestavit seznam slov nebo frází, které se nejčastěji používají zejména ve spamech. Použití ML algoritmu na velkém počtu dat pro hledání skrytých závislostí se říká *data mining* (dolování z dat).

Jak už bylo zmíněno, existují 3 hlavní typy metod ML, a sice: *učení s učitelem, učení bez učitele, zpětnovazebné učení.* Rozebereme každou z nich detailněji. [1]

1.2 Metoda učení s učitelem

Nejpoužívanější metoda, která je definovaná použitím označených („označení“ angl. *label)* datasetů pro učení algoritmu určeného pro klasifikaci dat nebo přesný odhad výstupu.

1.2.1 Klasifikace

Výše zmíněna detekce spamu je typický příklad klasifikace. Cílem této úlohy je přiřadit vstup x výstupu y, kde $y \in \{1, \ldots, C\}$, C je počet tříd. Pokud $C = 2$, této úloze se říká *binární klasifikace* (obecně předpokládáme, že $y \in \{0,1\}$); Pokud $C > 2$, mluvíme o *vícetřídní klasifikaci* (angl. *Multiclass classification).* Existuje taky *multi-label klasifikace***,** ve které labely tříd nejsou vzájemně se vylučující (např., někdo je **vysoký** nebo **silný**).

Jeden ze způsobů definování tohoto problému je *funkční aproximace*. Hledáme $y = f(x)$ pro neznámou funkci f. Cílem je odhadnout funkci f pomocí daného trénovacího setu (dataset určený pro trénování modelu) a predikovat $\hat{y} = \hat{f}(x)$ (symbol střechy znamená odhad). Naším hlavním cílem je udělat predikce pomocí nových vstupů, tzn. těch, které nebyly použité při trénování (tomu se říká *generalizace*) [2].

1.2.2 Pravděpodobnostní odhad

Pomocí pravděpodobnostní predikce se řeší nejasné případy. Označíme rozdělení pravděpodobnosti přes všechna možná označení pro daný vektor vstupů x a trénovací set D jako $p(y|x,\ D)$. Toto reprezentuje vektor o délce C. V případě, kdy máme jenom dvě třídy, je postačující zavést jenom jedno číslo $p(y = 1|x,\ D)$, jelikož $p(y = 1|x,\ D) + p(y = 0|x,\ D) = 1$. Zavádíme také podmínku použitého modelu tím, že přidáme do pravděpodobností další člen $p(y = 1|x,\ D,\ M)$, kde M znamená model.

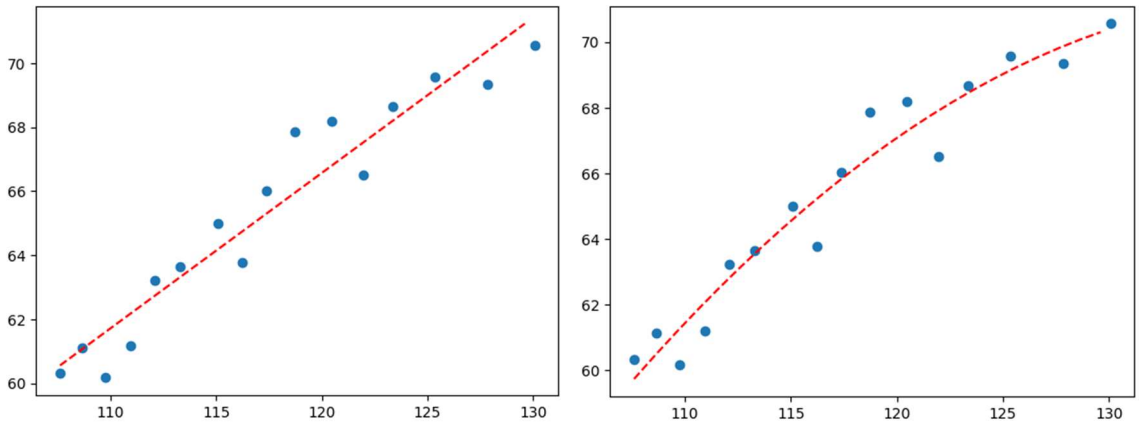
V našem případě „nejlepší odhad“ označujeme za „pravdivé označení“, což je výstup s největší pravděpodobností výskytu, použitím

$$\hat{y} = \hat{f} = \arg\max_{c=1,\ldots,C} p(y = c|x, D)\,. \tag{1}$$

Toto odpovídá nejpravděpodobnějšímu označení třídy a nazývá se to *modus* rozdělení $p(y|x, D)$; je také známý jako odhad MAP (MAP znamená **maximum a posteriori**). [2]

1.2.3 Regrese

Regrese je podobná klasifikaci, s výjimkou toho, že výstupní proměnná (response variable) je spojitá. Obr. 3 ukazuje jednoduchý příklad: máme jeden vstup $x_i \in \mathbb{R}$ a jeden vystup $y_i \in \mathbb{R}$. Přes daná data proložíme dvě čáry: přímku a parabolu (této formě regrese se říká *curve fitting*). Při řešení této úlohy mohou vzniknout různé problémy, jako například práce s vícerozměrnými vstupy, odlehlými hodnotami (angl. *outliers)*, nehladkými výstupy atd.



(a) (b)

Obr. 3: Curve fitting náhodně generovaných dat. (a) Přímka. (b) Parabola.

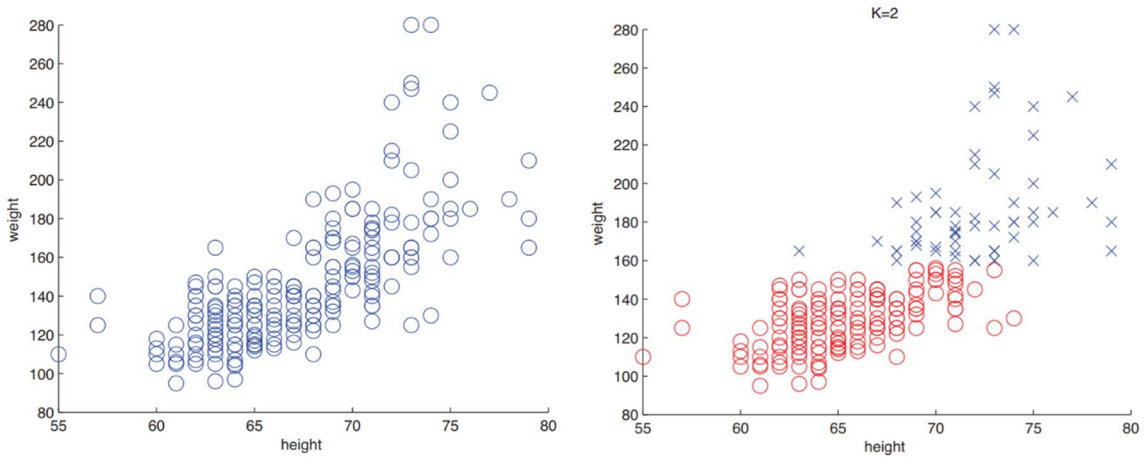
Tady je několik příkladu použití této metody v běžném životě:

- Odhad ceny akcí na burze.
- Odhad věku diváků daného videa na YouTube (online video platforma)
- Odhad polohy robotického ramena v prostoru pro dané signály poslané na různé motory
- Odhad teploty v jakémkoliv místě uvnitř budovy pomocí dat o počasí, času, senzoru dveří, atd. [2]

1.3 Metoda učení bez učitele

Teď se podíváme na metodu učení bez učitele, kdy máme pouze vstupy bez označených výstupů. Cílem je objevit skryté závislosti v datech. Na rozdíl od učení s učitelem, nemáme požadované výstupy, proto úlohu definujeme jako odhad hustoty (angl. *Density estimation*). To znamená, že budeme vytvářet model ve formě $p(x_i|\theta)$ (na rozdíl od učení s učitelem, kde jsme uvažovali $p(y|x,\ D)$). Další rozdíl je v tom, že x_i je vektor příznaků (angl. features), z toho plyne, že musíme používat pravděpodobnostní modely s více proměnnými (angl. multivariate). Toto neplatí pro učení s učitelem, kde y_i je většinou jediná proměnná, jejíž hodnotu se snažíme odhadnout. Tyto situace jsou o něco jednodušší, jelikož stačí použití pravděpodobnostních modelů s jednorozměrnou proměnnou (angl. *univariate*).

Tento typ učení se nejvíc podobá tomu, jak se učí lidé a zvířata. Je to také víc používané než učení s učitelem, protože nevyžaduje lidskou expertízu pro manuální označení dat, což je drahé. Typickou úlohou pro učení bez učitele je shlukování dat. [2]



(a) (b)

Obr. 4: (a) Výška a váha lidí. (b) Možné shlukování pomocí dvou K = 2 shluků.[2]

Obr. 4 ukazuje 2d (dvou rozměrová) data, která reprezentují výšku a váhu skupiny 210 lidí. Může se zdát, že ta data mohou být rozdělena do shluků (podskupin), ale není jasno kolik těch podskupin musí být. Označíme K za počet shluků. Naším prvním cílem je určit rozdělení pravděpodobnosti vůči počtu shluků., $p(y|x, D)$. Pro jednoduchost aproximujeme rozdělení $p(y|x, D)$ jejím modem $K^* = \arg\max p(K|D)$. V případě učení s učitelem bychom věděli, že existují dvě třídy, ale v tomto příkladu přesný počet shluků zadán nemáme a můžeme si zvolit jakékoliv číslo.

Druhým cílem bude odhadnout do jakého shluků patří jednotlivé data pointy (jednotlivé body na grafu). Řekneme $z_i \in \{1, \dots, K\}$ reprezentuje shluk do kterého patří data point i. (z_i je příklad *hidden* nebo *latent* proměnné, protože se nevyskytuje v trénovacím setu.) Můžeme přiřadit jednotlivé data pointy do patřičných shluků spočtením $z_i^* = \arg\max p(z_i = k|x_i, D)$. Toto je ilustrováno na Obr. 4 (b), kde autor znázornil odlišné shluky různými barvami. Shlukování se používá v různých disciplínách:

- V astrofyzice systém **autoclass** objevil nový druh hvězd pomocí shlukování
- V **e-komerci** (internetové obchody) pomocí shlukování se vytváří skupiny lidí se shodným chováním na internetu a shodnými nákupy. Dělá se to pro to, aby internetové obchody mohly zasílat reklamu svých produktů pouze jejich cílové skupině
- V biologii se shlukování používá v analýze dat z průtokové cytometrie pro objevení různých subpopulací buněk. [2]

1.4 Metoda zpětnovazebného učení

Základní myšlenkou zpětnovazebného učení je *agent* (počítač, program), který se učí pomocí interagování s jeho *prostředím*. Tento agent umí „cítit“ stav jeho prostředí a mít na tento stav vliv. Agent také musí mít zadány cíl, jakého stavu prostředí musí dosáhnout. Může se zdát, že tato metoda je velice podobná metodě učení bez učitele, ale není tomu tak. Pomocí metod učení bez učitele se hledají skryté závislosti a struktury v datech, kdežto metody zpětnovazebného učení řeší problém maximizace *odměny*.

Dalším problémem, který se nevyskytuje u jiných metod než u zpětnovazebného učení, je trade-off (česky *kompromis*) mezi průzkumem (*exploration*) a exploatací Agent musí aplikovat kroky, které mu předtím zaručily velkou odměnu, ale, na druhou stranu, musí prozkoumat nové, které mu mohou zajistit větší odměnu. Dilema spočívá v tom, že tato metoda bude mít zaručený neúspěch při prvních pokusech.

Tato práce je zaměřená především na problematiku klasifikace, proto se detailním rozborem metody učení bez učitele a zpětnovazebné metody nebudeme zabývat. Avšak pro lepší představu jsou níže uvedené příklady použití metody zpětnovazebného učení:

- Umělý šachový hráč. Volba tahu je zdůvodněna plánováním – uvažování různých tahů a protitahů – a pomocí tohoto počítač může usoudit, který tah je nejvíc efektivní.
- Mobilní robot uvažuje, zda musí začít uklízet další místnost nebo se vrátit k nabíjecí stanici. Robot rozhoduje podle aktuálního stavu baterie a jak rychle dokázal najít nabíjecí stanici v minulosti. [2]

1.5 Maximal Margin Classifier

V této sekce definujeme nadrovinu (angl. *hyperplane*) a řekneme si co je optimal separating hyperplane. Toto je základní myšlenka, kterou pak použijeme při definování metody podpůrných vektorů.

V p-rozměrném prostoru, nadrovina je afinní podprostor o rozměru $p-1$. Například, ve dvourozměrném prostoru (plocha) nadrovinou bude jednorozměrný podprostor, jinými slovy přímka. Matematická definice nadroviny je docela jednoduchá. Ve dvourozměrném prostoru je nadrovina definována následující rovnicí [3]:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0, \tag{2}$$

kde β_0, β_1 a β_2 jsou parametry, a $X = (X_1, X_2)^T$ je bodem nadroviny. Můžeme si všimnout, že rovnice (2) není nic jiného než rovnice přímky, protože nadrovina dvourozměrného prostoru je přímka. Obecný tvar této rovnice vypadá následovně, [3]

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0, \tag{3}$$

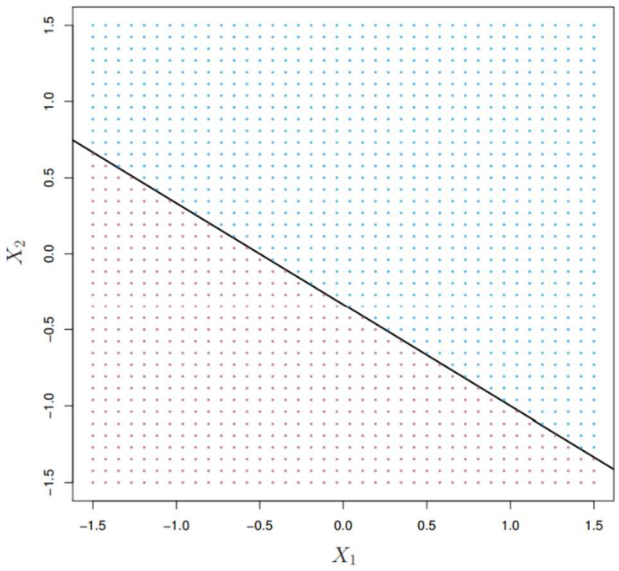
kde $X = (X_1, X_2, \ldots, X_p)^T$ v p-rozměrném prostoru (jinými slovy vektor o délce p) musí ležet na nadrovině.

Řekneme, že X je větší nebo menší než nula.

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p > 0 \tag{4}$$

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p < 0 \tag{5}$$

V tomto případě (4) a (5) leží na opačných stranách nadroviny, což znamená, že tato nadrovina dělí p-rozměrný prostor na dvě části. Dvourozměrný příklad této situace je znázorněn na Obr. 5. [3]



Obr. 5: Rozdělení dvourozměrného prostoru nadrovinou $1 + 2X_1 + 3X_2 = 0$. Pro modrou oblast platí $1 + 2X_1 + 3X_2 > 0$, pro fialovou oblast tedy platí $1 + 2X_1 + 3X_2 < 0$. [3]

1.5.1 Klasifikace pomocí nadroviny

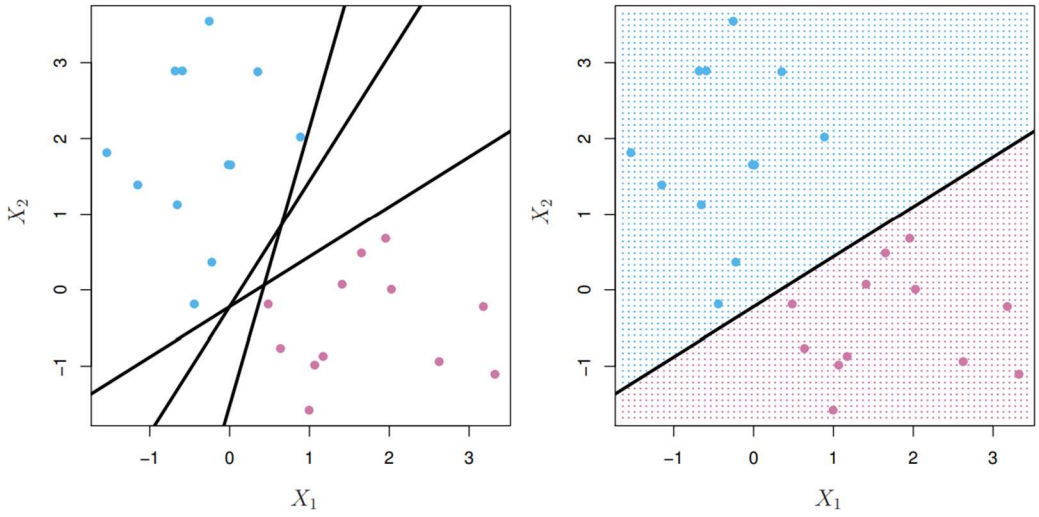
Máme $n \times p$ datovou matici $X = (x_1, \ldots, x_n)$, kde n je počet trénovacích pozorování a p je dimenze a

$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \ldots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}. \tag{6}$$

Tato pozorování spadají do dvou tříd, a sice $y_1, \ldots, y_n \in \{-1,1\}$, kde -1 reprezentuje jednu třídu a 1 druhou. Máme také testovací pozorování, p-vektor pozorovaných příznaků $x^* = (x_1^* \cdots x_p^*)^T$.

Naším cílem je vývoj klasifikačního algoritmu, který pomocí trénovacích dat dokáže správně klasifikovat testovací pozorování. Tento algoritmus bude založen na konceptu *separační nadroviny* (angl. *separating hyperplane*). [3, 4]

Předpokládáme, že existuje taková nadrovina, která bude separovat trénovací pozorování přesně do partičních tříd. Příklad tří takovýchto separačních nadrovin je znázorněn na Obr. 6 (a).



(a) (b)

Obr. 6: (a) Jsou zobrazeny dvě třídy pozorování (modrá a fialová). Na obrázku jsou zobrazeny tři, z mnoha možných, separační nadroviny. (b) Separační nadrovina rozděluje pozorování do dvou tříd (modrá mřížka a fialová mřížka). [3]

Označíme pozorování z modré třídy jako $y_i = 1$ a pozorování z fialové $y_i = -1$. Pak má separační nadrovina následující vlastnosti:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p > 0 \; if \; y_i = 1 \tag{7}$$

a

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p < 0 \; if \; y_i = -1 \tag{8}$$

nebo

$$y_i(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p) > 0 \tag{9}$$

pro všechna $i = 1, \ldots, n$.

Pokud takováto nadrovina existuje, můžeme pomocí ní vytvořit velice přirozený klasifikační model: testovací pozorování je přiřazeno do třídy podle toho na jaké straně nadroviny je rozmístěno. Obr. 6 (b) ukazuje příklad podobného klasifikačního modelu. Tzn., že klasifikujeme testovací pozorování x^* v závislosti na znaménku $f(x^*) = \beta_0 + \beta_1 x^* + \beta_2 x_2^* + \cdots + \beta_p x_p^*$. Pokud $f(x^*)$ je kladné, hovoříme o třídě 1 (modrá třída). Na druhou stranu, pokud $f(x^*)$ je záporné, přiřazujeme pozorování do třídy -1 (fialová třída). Můžeme také využít *magnitudy* $f(x^*)$ (angl. *magnitude*). V případě, že je $f(x^*)$ daleko od nuly, $f(x^*)$ je daleko od nadroviny, což má za následek to, že volba modelu je přesná. V opačném případě, kdy $f(x^*)$ je blízko nuly, si nemůžeme být naprosto jistí, že volba modelu je přesná. Je dobré zmínit, že klasifikační model založený na separační nadrovině vede k *linear desicion boundary* (česky *lineární rozhodovací hranice*). [3]

1.5.2 Maximal Margin Classifier

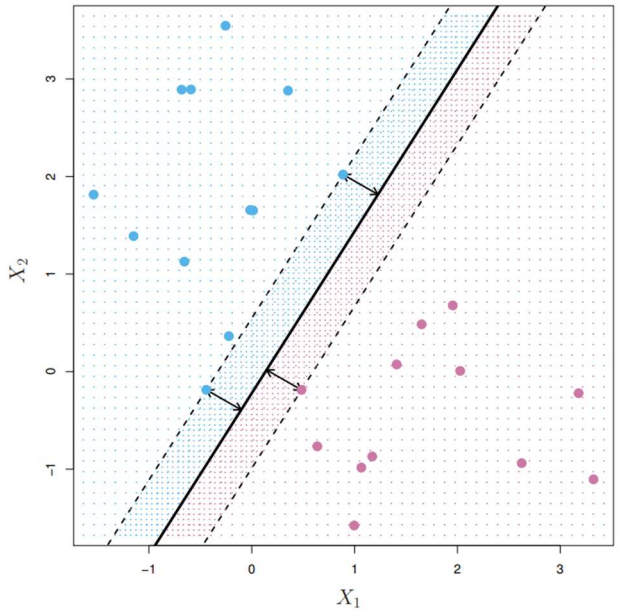
Takže, pokud naše data jsou dokonale separovatelná pomocí nadroviny, v tom případě bude existovat nekonečně mnoho vhodných nadrovin. To je dáno tím, že nadroviny obecně můžeme posouvat a rotovat, aniž by přestaly separovat data. Pro to, abychom vytvořili klasifikační model, měli bychom mít zdůvodněný postup pro volbu jedné nadroviny.

Přirozenou volbou je *maximal margin hyperplane (nadrovina s maximálním odstupem,* dále maximal margin nadrovina*)*, což je separační nadrovina, která je rozmístěna nejdále od trénovacích pozorování. *Margin* je vzdálenost nadroviny od trénovacích pozorování. Čím je tato vzdálenost menší, tím je menší margin. Můžeme tedy klasifikovat jednotlivá pozorování v závislosti na tom, na jaké straně maximal margin nadroviny jsou. Říká se tomu *maximal margin classifier*. Naší snahou je vytvořit model tak, aby měl velký margin jak u trénovacích dat, tak u testovacích dat. Přestože ve většině případů je maximal margin classifier je úspěšná volba, může to také vést k *overfittingu* (angl. *overfitting*) když p je velké. O problematice overfittingu a underfittingu budeme hovořit dále v této práci.

Řekneme, že $\beta_0, \beta_1, \ldots, \beta_p$ jsou koeficienty maximal margin nadroviny, tehdy maximal margin classifier klasifikuje testovací pozorování x^* v závislosti na znaménku $f(x^*) = \beta_0 + \beta_1 x^* + \beta_2 x_2^* + \cdots + \beta_p x_p^*$.

Obr. 7: 7 ukazuje maximal margin nadrovinu na datasetu z Obr. 6. Po porovnání Obr. 6 (b) a Obr. 7: vidíme, že maximal margin nadrovina z Obr. 7 má větší vzdálenost mezi dvěma třídami a nadrovinou, má tedy větší margin. Jednoduše řečeno, maximal margin nadrovina představuje středovou přímku nejširší „desky“, kterou můžeme vložit mezi dvěma třídami.

Po zkoumání Obr. 7 vidíme, že tři trénovací pozorování, které leží na čárkované přímce, jsou ekvidistantní od nadroviny a reprezentují šířku marginu. Tyto tři pozorování jsou známe jako *podpůrné vektory* (angl. *support vectors)*, jelikož jsou vektory v p-rozměrném prostoru a „podpírají“ maximal margin nadrovinu ve smyslu, že když si dovolíme trochu posunout tyto body, posune se i celá nadrovina. Zajímavé je to, že maximal margin nadrovina závisí přímo na podpůrných vektorech, ale ne na ostatních pozorováních [3].



Obr. 7: Jsou zobrazeny dvě třídy (modrá a fialová). Maximal margin nadrovina je plná přímka. Margin je vzdálenost od nadroviny k jedné z čárkovaných přímek. Dva modré body a jeden fialový bod jsou podpůrné vektory, vzdálenost mezi nimi a nadrovinou je vyznačena šipkami.[3]

To, že maximal margin nadrovina záleží přímo pouze na několika pozorováních je důležitá vlastnost, která se vyskytne u metody podpůrných vektorů.

1.5.3 Konstrukce maximal margin klasifikátoru

Podíváme se na konstrukci separační nadroviny. Máme N párů $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, kde $x_i \in \mathbb{R}$ a $y_i \in \{-1,1\}$. Nadefinujeme si nadrovinu

$$\{x: f(x) = x^T\beta + \beta_0 = 0\} \tag{10}$$

kde β je jednotkový vektor: $\|\beta\| = 1$. Klasifikační pravidlo je

$$G(x) = \text{sign}[x^T\beta + \beta_0]. \tag{11}$$

Našim cílem je maximalizovat margin.

$$\max_{\beta,\beta_0\|\beta\|=1} M$$
$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq M, i = 1, \dots, N. \tag{12}$$

Můžeme se zbavit omezení $\|\beta\| = 1$ a dostaneme následující výraz:

$$\frac{1}{\|\beta\|} y_i(x_i^T\beta + \beta_0) \geq M \tag{13}$$

Nebo

$$y_i(x_i^T\beta + \beta_0) \geq M\|\beta\| \tag{14}$$

Jelikož pro každou β a β_0 splňující tyto nerovnosti, jakýkoliv kladný násobek je splňuje také, můžeme bez újmy na obecnosti zavést $\|\beta\| = 1 / M$. Tím pádem (15) je ekvivalentní [3]

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 \quad \text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1, i = 1, \ldots, N. \tag{15}$$

Výraz (15) je obvyklý způsob zapisování support vector machine pro separovatelná data. Toto je konvexní optimalizační problém (kvadratická kriteriální funkce, lineární omezení). Lagrangeova funkce (primární), která má být minimalizována vzhledem k β a β_0 vypadá následovně: [5]

$$L_p = \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{N} \alpha_i[y_i(x_i^T\beta + \beta^0) - 1] \tag{16}$$

Dosazením nuly do derivací obdržíme

$$\beta = \sum_{i=1}^{N} \alpha_i\, y_i x_i, \tag{17}$$

$$0 = \sum_{i=1}^{N} \alpha_i\, y_i, \tag{18}$$

Dosazením tohoto do (16) dostaneme tzv. Wolfeho duální problém

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N} \alpha_i y_i x_i^T \sum_{k=1}^{N} \alpha_k\, y_k x_k, \tag{19}$$

kde $\alpha_i \geq 0$.

Řešením je maximalizace L_D v kladném orthantu[1]. Mimo jiné, řešení musí splňovat Karush-Kuhn-Tucker omezení, které zahrnuje (17), (18), (19) a

$$\alpha_i[y_i(x_i^T\beta + \beta_0) - 1] \geq 0\ \forall i. \tag{20}$$

Z toho plyne, že

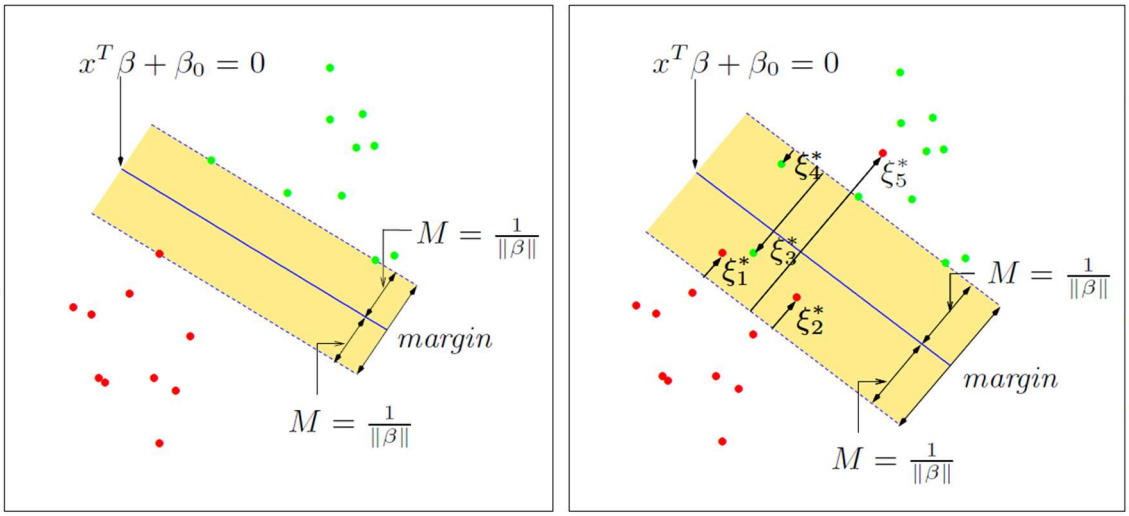
- pokud $\alpha_i > 0$, $y_i(x_i^T\beta + \beta_0) = 1$, jinými slovy x_i je na hranici marginu;
- pokud $y_i(x_i^T\beta + \beta_0) > 1$, x_i na hranici marginu není a $\alpha_i = 0$. [5]

[1] Kladný orthant je jedna ze čtyř částí libovolného prostoru, která obsahuje body, jejichž všechny souřadnice jsou kladné (tj. x > 0, y > 0 a z > 0). Tuto oblast lze také označit jako první kvadrant nebo jednoduše jako kladná polovina prostoru.[6]

1.6 The Support Vector Classifier

1.6.1 Neseparovatelný případ

Maximal margin je velmi přirozeným způsobem jak provádět klasifikace, pokud separační nadrovina existuje. Ne vždy však tomu tak je, a tudíž ne vždycky jde použít maximal margin classifier. V tomto případě optimalizační problém (10) – (12) nemá žádné řešení při $M > 0$. Takový případ je ukázán na Obr. 8 (b), kde nemůžeme jednoznačně rozdělit daná data na dvě třídy. Jak uvidíme dále, můžeme rozšířit koncept separační nadroviny pro vývoj nadroviny, která téměř separuje dané třídy pomocí tzv. *soft margin* (dovoluje některým pozorováním ležet uvnitř marginu nebo na špatné straně nadroviny). Generalizace maximal margin klasifikátoru pro neseparovatelný případ je známá jako ***support vector classifier*** (nebo *soft margin classifier*). [3]



(a) (b)

Obr. 8: Jsou zobrazeny dva případy. (a) Separovatelný případ. Nadrovina je plná modra čára, čárkované přímky omezují margin o šířce $2M = 2 / \|\beta\|$. (b) Neseparovatelný případ. Body označené ξ_i^* jsou na špatné straně marginu o částku $\xi_i^* = M\xi_i$; body na správné straně mají $\xi_i^* = 0$. Margin je maximizován za podmínky na celkový „rozpočet“ $\Sigma\xi_i \leq const$. Z toho plyne, že $\Sigma\xi_i^*$ je celková vzdálenost všech bodů na špatné straně marginu. [5]

Máme situaci jako na Obr. 8 (b). V tomto případě, třídy se překrývají a nejdou jednoznačně rozdělit pomocí separační nadroviny. Možným řešením by v tomto případě bylo pokračovat v maximalizaci marginu a povolit několika bodům zůstat na špatně straně marginu. Nadefinujeme slack proměnné $\xi = (\xi_1, \xi_2, \dots, \xi_N)$. Jsou dva možné způsoby modifikace omezení v (12):

$$y_i(x_i^T\beta + \beta_0) \geq M - \xi_i \tag{22}$$

nebo

$$y_i(x_i^T\beta + \beta_0) \geq M(1 - \xi_i), \tag{23}$$

$\forall i, \xi_i \geq 0, \sum_{i=1}^{N} \xi_i = const$. Oba způsoby vedou k různým řešením. První způsob vypadá přirozeněji, poněvadž vzdálenost je měřena přímo od marginu; v druhém způsobu je měřena relativní vzdálenost, která se mění se změnou šířky marginu M. Hlavní rozdíl je v tom, že první způsob vede k nekonvexní optimalizační úloze, kdežto druhý způsob vede ke konvexní optimalizační úloze, což znamená, že (23) vede k „standardnímu“ SVM.

Myšlenka je následující – hodnota ξ_i v omezení (23) je proporcionální tomu, jak moc odhad $f(x_i) = x_i^T \beta + \beta_0$ leží na špatné straně marginu. Misklasifikace (chybné zařazení) vznikají, když $\xi_i > 1$, takže omezení $\Sigma \xi_i$ na hodnotě K omezuje všechny trénovací klasifikace na K. Stejně jako v (15) v Kapitole 2.2.3, můžeme se zbavit normovaného omezení na β, nadefinovat $M = 1 / \beta$ a zapsat optimalizační problém v ekvivalentním formátu

$$\min \|\beta\| \quad \text{subject to } \begin{cases} y_i(x_i^T \beta + \beta_0) \geq 1 - \xi_i \, \forall i, \\ \xi_i \geq 0, \sum \xi_i \leq const. \end{cases} \tag{24}$$

Toto je tradiční způsob definování support vector classifier pro neseparovatelný případ.

Z rovnice (24) vidíme, že body daleko uvnitř hranice třídy nehrají velkou roli v budování této hranice. [5]

1.6.2 Výpočet support vector classifier

Problém (24) je kvadratický problém s lineárními omezeními, tudíž je to konvexní problém. Výpočetně je vhodné přepsat výraz (24) do následujícího tvaru

$$\begin{gathered} \min_{\beta, \beta_0} \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^{N} \xi_i \\ \text{subject to } \xi_i \geq 0, y_i(x_i^T \beta + \beta_0) \geq 1 - \xi_i \, \forall i, \end{gathered} \tag{25}$$

kde parametr „ceny“ C nahrazuje konstantu v (24); separační případ vede k $C = \infty$. Lagrangeova funkce (primární) vypadá následovně:

$$L_P = \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i [y_i(x_i^T \beta + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^{n} \mu_i \xi_i, \tag{26}$$

kterou minimalizujeme vzhledem k β, β_0 a ξ_i. Položíme příslušné derivace rovny nule a dostaneme

$$\beta = \sum_{i=1}^{N} \alpha_i y_i x_i, \tag{27}$$

$$0 = \sum_{i=1}^{N} \alpha_i y_i, \tag{28}$$

$$\alpha_i = C - \mu_1, \forall i. \tag{29}$$

Dosažením (27) – (29) do (26) obdržíme Wolfeho duální problém

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \, \alpha_{i'} y_i y_{i'} x_i^T x_{i'} \, , \tag{30}$$

který udává spodní hranici kriteriální funkce (25) pro každý přípustný bod. Maximalizujeme L_D s ohledem na $0 \leq \alpha_i \leq C$ a $\sum_{i=1}^{N} \alpha_i y_i = 0$. Navíc Karush-Kuhn-Tucker podmínky doplňují (27) – (29)

$$\alpha_i [y_i (x_i^T \beta + \beta_0) - (1 - \xi_i)] = 0, \tag{31}$$

$$\mu_i \xi_i = 0, \tag{32}$$

$$y_i (x_i^T \beta + \beta_0) - (1 - \xi_i) \geq 0, \tag{33}$$

pro $i = 1, \ldots, N$. Spolu rovnice (27) – (33) popisují řešení primární a duální úlohy. Z (27) vidíme, že řešení pro β vypadá následovně:

$$\hat{\beta} = \sum_{i=1}^{N} \hat{\alpha}_i y_i x_i, \tag{34}$$

pro nenulové koeficienty pouze pro takové pozorování i, které splňují omezení (33). Takovým pozorováním říkáme podpůrné vektory. Některé body leží na hranici marginu $\left(\hat{\xi}_i = 0\right)$ a jsou charakterizovány $0 < \hat{\alpha}_i < C$.

Maximalizace duálního problému (30) je výpočetně jednodušší než u primárního problému (26). Vzhledem k řešením pro $\hat{\beta}_0$ a β, funkce rozhodnutí má následující tvar

$$\begin{aligned} \hat{G}(x) &= \text{sign}\left[\hat{f}(x)\right] \\ &= \text{sign}\left[x^T \hat{\beta} + \hat{\beta}_0\right]. \end{aligned} \tag{35}$$

V této formulaci máme volnost pro volbu jednoho parametru (ceny C). Identifikaci vhodné volby takového parametru se říká tuning [5].

1.7 Metoda podpůrných vektorů

Support vector classifier, který jsme si vysvětlili v předchozí kapitole, hledá lineární hranice ve vstupním prostoru příznaků. Pro lepší výsledky můžeme rozšířit prostor příznaků pomocí tzv. bázových funkcí (angl. *basis functions*) (o těchto funkcích budeme hovořit dále v této práci). Obecně lineární hranice v rozšířeném prostoru dosahují lepších výsledků. Jakmile vybereme bázové funkce je procedura stejná jako v předchozím

případě. Určíme SV klasifikátor pomocí vstupů $h(x_i) = \big(h_1(x_i), h_2(x_i), \dots, h_m(x_i)\big), n = 1, \dots, N$, a tím vytvoříme nelineární funkci $\hat{f}(x) = h(x)^T\hat{\beta} + \hat{\beta}_0$. Klasifikátor je stejný jako předtím $\hat{G}(x) = \text{sign}\left(\hat{f}(x)\right)$.

SVM je rozšíření této myšlenky, kde rozměrnost prostoru příznaku může v některých případech dosahovat nekonečna. Může se zdát, že v tomto případě výpočty budou nepřístupné. Také se může zdát, že použitím vhodné basis funkce data budou separovatelná a vznikne overfitting. Nejdřív si ukážeme, jak SVM zachází s problémy výpočetní náročností. Potom uvidíme, že SVM klasifikátor opravdu řeší function-fitting problém pomocí konkrétního kritéria a tvaru regularizace. [5]

1.7.1 Výpočet SVM pro klasifikaci

Můžeme reprezentovat optimalizační problém (26) a jeho řešení speciálním způsobem, který pouze zahrnuje vstupní příznaky přes skalární součin. Děláme to přímo pro transformované vektory příznaků $h(x_i)$. Potom můžeme vidět, že pro některá zvolena h, tyto skalární součiny můžou být výpočetně levná.

Lagrangeova dualní funkce (30) vypadá následovně:

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N} \alpha_i\, \alpha_{i'} y_i y_{i'} \langle h(x_i), h(x_{i'})\rangle. \tag{36}$$

Z (29) vidíme, že funkce řešení $f(x)$ může být zapsána jako

$$f(x) = h(x)^T\beta + \beta_0 \; = \sum_{i=1}^{N} \alpha_i y_i \langle h(x), h(x_i)\rangle + \beta_0. \tag{37}$$

Jako předtím, α_i, β_0 můžou být určena pomocí vyřešení $y_i f(x_i) = 1$ v (37) pro všechna x_i, pro která platí $0 < \alpha_i < C$.

Takže, $h(x)$ je zapojeno do (36) a (37) pouze přes skalární součiny. Vlastně, nemusíme specifikovat transformaci $h(x)$ vůbec, stačí jenom znát jádrovou funkci (angl. kernel function).

$$K(x, x') = \langle h(x), h(x')\rangle, \tag{38}$$

která spočítá inner products v transformovaném prostoru. K má být symetrická pozitivní (semi-) definitní funkce. Tři běžné volby pro K v literatuře jsou

$$K(x, x') = (1 + \langle x, x'\rangle)^d, \tag{39}$$

$$K(x, x') = \exp(-\gamma\|x - x'\|^2), \tag{40}$$

$$K(x, x') = \tanh(\kappa_1\langle x, x'\rangle + \kappa_2). \tag{41}$$

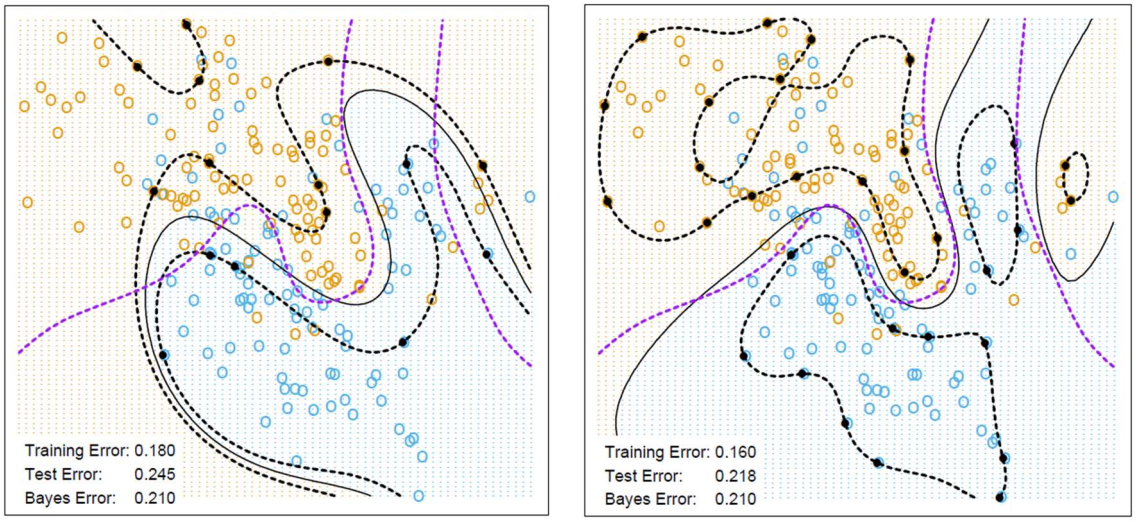
(39) – Polynom stupně d.; (40) – Gaussova (Radial basis); (41) – Hyperbolický tangens
Například, máme prostor příznaků se dvěma vstupy X_1 a X_2 a použijeme polynom 2. stupně jako jádro funkci.

$$K(X,X') = (1 + \langle X, X' \rangle)^2 = (1 + X_1X_1' + X_2X_2')^2 \\ = 1 + 2X_1X_1' + 2X_2X_2' + (X_1X_1')^2 + (X_2X_2')^2 \\ + 2X_1X_1'X_2X_2' \tag{42}$$

Tehdy $M = 6$ a pokud zvolíme $h_1(X) = 1, h_2(X) = \sqrt{2}X_1, h_3(X) = \sqrt{2}X_2, h_4(X) = X_1^2, h_5(X) = X_2^2, h_6(X) = \sqrt{2}X_1X_2$, pak $K(x,x') = \langle h(x), h(x') \rangle$. Z (37) vidíme, že řešení může být zapsáno jako [5]

$$\hat{f}(x) = \sum_{i=1}^{N} \hat{\alpha}_i y_i K(x, x_i) + \hat{\beta}_0. \tag{43}$$

Role parametru C je jasnější v rozšířeném prostoru příznaků, poněvadž v něm je často dosažitelná úplná separace. Velká hodnota C odradí jakékoliv pozitivní ξ_i a vede k overfittingu v původním prostoru příznaků; malá hodnota C prospívá k malé hodnotě $\|\beta\|$, což vede k vyhlazení hranice. Obr. 9 ukazuje dva nelineární případy použití SVM. Regularizační parametr byl zvolen v obou případech pro dosažení dobrých výsledků pro testovací chybu.



(a) (b)

Obr. 9: Dva nelineární případy SVM. (a) Použit polynom 4. stupně jako jádrové funkce. (b) Případ s použitím radial basis jádrové funkce. V obou případech parametr C byl navržen tak, aby zajistil nejlepší výsledky, což pro tyto případy je $C = 1$. Radial basis jádrová funkce v tomto případě bude nejlepší volbou. [5]

1.8 Bias-Variance tradeoff

Nyní rozebereme klasickou situaci ve statistické inferenci. Naším cílem ve statické inferenci je vytvořit odhad pro neznamý parametr θ^* pro daný dataset S.

Označíme náš estimator jako $\hat{\theta}_n$, kde n je rozměr (velikost) datasetu použitého pro model. Všimneme si, že $\hat{\theta}_n$ je náhodná proměnná, přestože θ^* není. Za nic nestojí, že náhodnost v $\hat{\theta}_n$ tedy nepřímo závisí na θ^*, a to díky tomuto šumu. Rozdělení $\hat{\theta}_n$ se často nazývá *Výběrové rozdělení*. Bias (odchylka) a Variance (rozptyl) estimatoru $\hat{\theta}_n$ jsou první a druhý moment jeho sampling rozdělení.

Nazýváme $\mathrm{Bias}(\hat{\theta}_n) \equiv \mathrm{E}[\hat{\theta}_n - \theta^*]$ Bias estimatoru $\hat{\theta}_n$. Estimator $\hat{\theta}_n$ se nazývá Unbiased (nevychýlený), pokud $\mathrm{E}[\hat{\theta}_n - \theta^*] = 0$, jinými slovy $E[\hat{\theta}_n] = \theta^*$ pro všechny hodnoty θ^*.

Obdobně nazýváme $\mathrm{Var}(\hat{\theta}_n) \equiv \mathrm{Cov}[\hat{\theta}_n]$ variančně-kovarianční matice odhadu. Všimněte si, že na rozdíl od Biasu, Variance estimatoru nezáleží přímo na parametru θ^*. V rámci kvadratické chyby, the Odchylka a Rozptyl odhadů jsou vztaženy jako:

$$(\hat{\theta}_n) = \mathrm{E}\left[\left\|\hat{\theta}_n - \theta^*\right\|^2\right] = \mathrm{tr}[\mathrm{Var}(\hat{\theta}_n)] + \left\|\mathrm{Bias}(\hat{\theta}_n)\right\|^2 \tag{44}$$

, kde tr je stopa matice.

Použití technik pro snížení rozptylu má často za následek zvýšení odchylky. Tomuto fenoménu se říká bias-variance tradeoff. [7]

1.9 Overfitting a Underfitting nebo Bias a Variance v praxi

Pro pochopení proč pojmy Bias a Variance jsou důležité, rozebereme si co je overfitting a underfitting:

- Overfitting znamená, že model má high variance. Abychom se zbavili overfittingu, musíme zmenšit variance estimatoru, což je: zvětšit zobecnění, použít větší dataset, zmenšit počet příznaků, atd.
- Underfitting vyjadřuje high bias modelu. Underfitting čelíme zmenšením biasu estimatoru, takže musíme: zmenšit regularization, zvětšit počet příznaků, atd. [7]

2 APLIKACE SVM V REÁLNÉM SVĚTĚ

V této části bakalářské práce se podíváme na několik aplikací SVM v reálném světě. Hlavně se zaměříme na kategorizaci textu, bioinformatiku a rozpoznání obrázků (image recognition). Také si něco povíme o příkladu ze začátku práce (detekce spamu). (jednotlivé výsledky mohou být zastaralé, nicméně tyto příklady jsou krásnou ukázkou použití SVM).

2.1 Kategorizace textu

Kategorizace textu spočívá v klasifikaci dokumentů do předem nadefinovaných kategorií. Například, rozebereme si sbírku dokumentů složenou z Usenet News zpráv. Jsou organizovány do nadefinovaných skupin, jakou jsou informatika, náboženství, statistika atd. Cílem je automaticky přiřadit každý nový dokument do správné skupiny. Dalším běžným příkladem je velká řáda internetových vyhledávačů, které ukazují webové stránky související s dotazem uživatele. Dokumenty jsou reprezentovány vektorovým prostorem, jehož rozměrnost je rovná počtu slov ve slovníku. To znamená, že problém kategorizace textu je problém ve vysoké dimenzi. Efektivita SVM v této úloze bude ilustrována na databázi Reuters [8] Toto je soubor textů obsahující 21576 dokumentů a 9947 kategorií. Rozměrnost datového prostoru v tomto případě je 9947, což je velikost slovníku. Výsledky dosažené při použití klasického SVM s lineární jádrovou funkcí jsou lepší než při použití jiných velice používaných klasifikačních metod: naive Bayes, Bayesian networks, classification trees, k-nearest neighbors. Průměrná úspěšnost pro SVM je 87%, kdežto pro výše zmíněné metody výsledky jsou 72%, 80%, 79% a 82%. Není to ale jediný důvod proč SVM je lepší než ostatní metody. SVM je 4-krát rychlejší než naive Bayes klasifikátor (nejrychlejší ze všech čtyř metod) a 35-krát rychlejší než classification trees. [9]

2.2 Použití v bioinformatice

Dalším velice používaným oborem SVM je bioinformatika. Dnes je velký zájem o analýzu microarray dat, což je analýza biologických vzorků pomocí využití profilů expresi genů. SVM se hlavně používal při klasifikaci tkání, predikci funkce genů, predikci subcelulární lokalizace proteinů, predikci sekundární struktury proteinů a predikci proteinových záhybů. Ve většině případů SVM dosáhl lepších výsledků jiné klasifikační metody a v nejhorším případě výkon SVM byl podobný výkonu nejlepší metody bez použití SVM. Například, v predikci subcelulární polohy proteinů se snažíme odhadnout subcelulární polohy proteinů z prokaryotických sekvencí. Jsou tři možné polohy: cytoplazmatická, periplazmatická a extracelulární. Z hlediska klasifikace problém se mění z klasifikace dvacetirozměrného vektoru do tří (velice nevyvážených) tříd.

Přesnost odhadu pro SVM (s použitím Gaussova jádra) dosahuje 91,4%, zatímco neuronové sítě a Markovské řetězce mají přesnost 81% a 89,1%. Při porovnání těchto metod na jiných příkladech bylo dosaženo stejných výsledků. Je důležité zmínit, že v tomto oboru je ještě prostor pro zlepšení. [9]

2.3 Rozpoznávání obrázků

V této sekci si rozebereme problém rozpoznávání obrázků. Podíváme se na dvě známé úlohy: psané rukou číslice a rozpoznávání obličeje. Data pro první problém byla získána z U.S. Postal Service databáze obsahující 9298 vzorků číslic obdržené z reálných PSČ (rozdělené do 7291 trénovacích vzorků a 2007 vzorků pro testování). Každá číslice je reprezentována černobílou maticí 16×16, tudíž každý datapoint je reprezentován vektorem $\mathbb{R}^{256}$. Lidské oko dokáže klasifikovat s nejvyšší přesností 97.5%. Nejvyšší dosažena přesnost SVM s jádrem polynomem 3.stupně je 96%. Dnes se na to ale používá zvláštní typ neuronových sítí, tzv. *konvoluční neuronové sítě* (angl. *convolutional neural network,* zkr. CNN), který dosahuje přesnosti víc než 99.9%. Použitím speciálního typu SVM s jádroem polynomem 3.stupně se dá dosáhnout přesnosti 96.8%. Princip specializace v tomto případě se skládá ze tří fází: v první fázi se model určí podpůrné vektory; v druhé fázi se generují nové datapointy pomocí různých uprav podpůrných vektorů (translace, rotace, transformace). Ve finální fázi se vytváří separační nadrovina na základě nově vytvořených datapointů.

Takže, teď se podíváme na problém rozpoznávání obličeje, a sice klasifikace pohlaví člověka. Data obsahují 1755 obrázků obličejů (1044 mužů a 711 žen). Celková přesnost dosažena použitím SVM s Gaussovým jádrem je 96.8% (97.9% pro muže a 95.2% pro ženy). Jako u předchozího problému, dnes se dospělo k modelům s větší přesností, ale tento výsledek můžeme stálé považovat za velice přesný. SVM také ukazuje dobré výsledky při detekci lidského obličeje v černobílém formátu. Hlavním problémem je zjistit připadnou polohu lidského obličeje na obrázku, a pokud nějaký člověk na obrázku je, algoritmus vrací kódovánou hodnotu jeho polohy. Přesnost dosažená použitím SVM s polynomiálním jádrem 2.stupně je 97.1%. [9]

2.4 Detekce spamu

Jak už jsem zmiňoval na začátku práce, detekce spamu je klasický příklad klasifikace a je to i dodnes velice častý problém, se kterým bojují největší firmy po celém světě. Populární klasifikační algoritmus Decision Tree Classifier se v tomto případě nedá použít, protože potřebuje hodně pamětí, což není přípustné pro systémy filtrace spamu. SVM je v tomto případě také vhodnou volbou i proto, že počet příznaků/atributů pro filtraci spamu je zhruba 7000, což je velké číslo. Dataset pro danou úlohu obsahuje 1897 spam zpráv, 250 těžce rozlišitelných dobrých zpráv, a 3900 jednoduchých dobrých zpráv.

Na tento problém byl použit SVM s dvěma různými jádrovými funkcemi: lineární a Gaussovo. Po kontrole výsledků vyšlo [10]

Tab. 1: Výsledky použití různých jádrových funkcí [10]

Typ funkce	Trénovací přesnost (%)	Testovací přesnost (%)	Trénovací čas (sekundy)	Testovací čas (sekundy)
Lineární	99.8	98.5	134	1.5
Gaussova	99.9	97.1	190	1.5

3 IMPLEMENTACE SVM

Tato část bakalářské práce bude věnována implementaci algoritmu SVM v různých softwarech na vhodném datasetu. Hlavním cílem bude porovnat výkonnost jednotlivých softwarů z hlediska rychlosti a přesnosti.

Volba vhodného softwaru v sobě zahrnuje několik částí: volbu vhodného programovacího jazyka a volbu *knihovny* (kolekce modulů pro konkrétní účely, v našem případě SVM knihovna nebo obecně knihovna s algoritmy strojového učení). Je také možnost napsat algoritmus SVM vlastnoručně bez použití knihoven, ale bez pokročilých znalostí programování takovýto algoritmus nebude dobře optimalizován, a tudíž bude pomalý. Po rešerši možných implementací mnou bylo zvoleno tří programovacích jazyků: Python, Matlab, R. Výhody použití těchto jazyků spočívají v tom, že tyto jazyky mají podobnou syntaxi, mají velice optimalizované knihovny pro implementaci SVM a jsou celosvětově použitelné pro tyto účely.

Velkou roli v dosažení dobrých výsledků také hraje vhodný dataset. V našem případě byla potřeba vyhledat dataset určený pro klasifikační účely, tzn. musí mít atribut, který jasně udává klasifikaci do dvou nebo více skupin. Každý dataset představuje shluk dat, roztříděných do atributů a hodnot těchto atributů. Data lze dále rozdělit do několika hlavních typů: [11]

1. Nominální data – typ dat, který reprezentuje jistou kvalitativní charakteristiku objektu z konečné množiny možností, proto se někdy hovoří o kvalitativních datech. Např. barva auta, kde množina možností je („modrá“, „černá“, „zelená“).
2. Ordinální data – typ dat, který jako nominální data reprezentuje kvalitativní charakteristiku z konečné množiny možností, s tou výjimkou, že ordinální data mají hierarchické uspořádání (tj. lze říct, která hodnota je větší nebo menší). Např. akademický titul.
3. Intervalová data – jsou typem dat, které jsou popsány jako intervaly mezi dvěma hodnotami. Tyto hodnoty označují meze, uvnitř kterých se může skutečná hodnota nacházet. Např. rozdíl teplot je vždy stejný bez ohledu na výchozí hodnotu.
4. Poměrová data – jsou typem dat, které představují poměr mezi dvěma hodnotami. Tyto hodnoty mohou být jakýmkoliv množstvím, například počtem, výškou, hmotností atd. Rozdílem od intervalových dat je, že poměrová data umožňují porovnávání mezi různými hodnotami v rámci stejné jednotky. Příkladem poměrových dat může být poměr mezi výškou a hmotností člověka, kde lze snadno porovnávat poměry mezi různými lidmi.

Intervalová a poměrová data jsou obvykle spojitá, kdežto ordinální a poměrová data jsou obvykle diskrétní, proto se zavádí obecnější rozdělení na *continuous data* (česky *spojitá*) a *categorical data* (česky *kategoriální*).

3.1 Stroke dataset

V této části práce bude implementace SVM na *stroke* (česky *mrtvice*) datasetu [12]. Podle WHO (Světová zdravotnická organizace) mrtvice je druhou nejčastější příčinou smrti (cca 11%). Tato data jsou použita k predikci toho, zda pacient dostane mrtvici v závislosti na vstupních parametrech. Rozebereme si podrobněji atributy a jejich možné vstupy:

1. Id – identifikační číslo pacienta
2. Gender – pohlaví pacienta (muž/žena)
3. Age – věk pacienta
4. Hypertension – vysoký krevní tlak (0 nemá, 1 má)
5. Heart disease – onemocnění srdce (0 nemá, 1 má)
6. Ever married – byl/a ženatý/vdaná (ano/ne)
7. Work type – povolání pacienta (dítě/práce pro stát/nikdy nepracoval/privatní info/self-employed)
8. Residence type – městský nebo vesnický typ pobytu
9. Avg glucose level – průměrná úroveň glukózy v krve
10. BMI – index tělesné váhy
11. Smoking status – kuřák/nekuřák/bývalý kuřák/nevíme
12. Stroke – pacient měl nebo neměl mrtvici (0 nebo 1)

Úloha klasifikace spočívá v tom, aby pomocí prvních jedenácti atributů se určila výsledná hodnota dvanáctého atributu, tj. zda došlo k mrtvici nebo ne. Každý atribut přidává další dimenzi a tím výpočetní složitost. V našem případě je dáno 12 atributu, lze tedy říci, že hovoříme o úloze ve jedenácti rozměrovém prostoru.

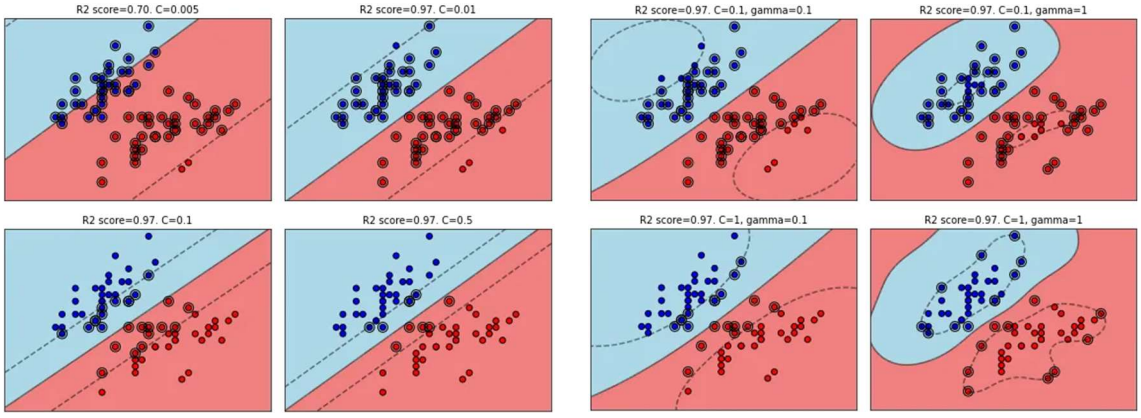
3.1.1 Hyperparametry

Toto je binární klasifikační úloha, která je dobře použitelná pro SVM. Maximal margin classifier, jádra a další vlastnosti SVM pomáhají řešit binární klasifikační úlohy s velkou přesností a stabilitou. Avšak pro dosažení dobrých výsledků nestačí jenom dobrý dataset a použití vhodné metody. Jako většina metod, SVM má tzv. *hyperparametry* (parametry nastavení modelu), které určují jeho chování. Mezi hlavní hyperparametry SVM patří:

1. Typ jádrové funkce: SVM má schopnost používat různé typy jádrových funkcí k transformaci dat do vyšších dimenzí, což umožňuje lepší oddělení dat v případě, že nejsou lineárně oddělitelná. Tyto funkce mohou být lineární, polynomiální, radiální atd.

2. Parametr C: Tento hyperparametr kontroluje míru penalizace chybných klasifikací. Vysoké hodnoty C vedou k modelu s větším množstvím podpůrných vektorů, což má za následek vyšší přesnost, ale také větší riziko overfittingu. Tento parametr byl podrobně rozebrán v kapitole 2.3.2.
3. Gamma (někdy sigma): Tento hyperparametr určuje, jak silně se budou body blízko rozdělovacího hyperplánu brát v úvahu při trénování modelu. Vysoké hodnoty Gamma vedou k modelům s vysokou variabilitou, kde malé změny v datech mohou mít velký vliv na výsledky.

Na Obr. 10: je vidět, jak se mění chování SVM při volbě různých hyperparametrů.



(a) (b)

Obr. 10: Vliv změny hyperparametrů na chování modelu. (a) Lineární jádro funkce. (b) Radiální jádro funkce. R2-score reprezentuje přesnost modelu. [13]

3.2 Implementace v Pythonu

Jazyk Python je dnes jedním z nejpoužívanějších programovacím jazykem a to z důvodu, že má jednoduchou syntaxi a obrovské množství optimalizovaných modulů. Při tvorbě modelu mnou byla použita knihovna SciKit Learn. Realizace SVM v Pythonu proběhla následovně:

3.2.1 Příprava a očištění dat

Před dosažením do modelu data musí být předem připravena. Na začátku stojí za zmínku to, že atribut Id (číslo pacienta) nemá žádný vliv na jeho zdravotní stav, proto tento atribut z datasetu bude vyhozen.

Po nahrávání dat do Pythonu a krátkému vyšetření bylo zjištěno, že data mají dva podstatné problémy:

1. Dataset má prázdné hodnoty v některých řádcích (NULL hodnoty)
2. Dataset má nesymetrické rozložení (počet pacientů bez mrtvice je skoro 20krát větší, než počet pacientů s mrtvicí)

První problém může být vyřešen následujícími způsoby:

- Smazat řádky, které obsahují NULL hodnoty.
- Nahradit NULL hodnoty aritmetickým průměrem pro tento atribut (v případě spojitých dat) anebo odhadnout hodnotu atributu pomocí regresních metod.
- V případě kategoriálních dat, lze odhadnout chybějící hodnotu pomocí nějaké klasifikační metody.

Pro dosažení nejlepšího výsledku mnou byla použita metoda rozhodovacích stromů (teoretický rozbor této metody je mimo rozsah práce). Na Obr 11. je ukázka implementace kódu v Pythonu.

```
# Replacing NaN values with Decision Tree model
DT_nan_pipe = make_pipeline(StandardScaler(), DecisionTreeRegressor(random_state=42))
X = dataset[['age', 'gender', 'bmi']].copy() # Atributy s NULL hodnotami
X.gender = X.gender.replace({'Male': 0, 'Female': 1, 'Other': -1}).astype(np.uint8) # Nahrazeni kategorialnich dat
nan_values = X[X.bmi.isna()] # Hledani NULL hodnot
X = X[~X.bmi.isna()]
Y = X.pop('bmi')
DT_nan_pipe.fit(X, Y)
predicted_bmi = pd.Series(DT_nan_pipe.predict(nan_values[['age', 'gender']]), index=nan_values.index)
dataset.loc[nan_values.index, 'bmi'] = predicted_bmi
```

Obr. 11: Implementace nahrazení NULL hodnot v Pythonu pomocí metody Decision Tree.

Problém nesymetrického rozložení datasetu se dá vyřešit pomocí algoritmu oversamplingu (česky *převzorkování*). Náhodné převzorkování (Random oversampling) zahrnuje náhodné duplikování příkladů z menšinové třídy a jejich přidání do trénovacího datasetu. Příklady z trénovacího datasetu jsou vybírány náhodně s opakováním. To znamená, že příklady z menšinové třídy mohou být vybrány a přidány do nového "více vyváženého" trénovacího datasetu vícekrát; jsou vybrány z původního trénovacího datasetu, přidány do nového trénovacího datasetu a poté vráceny nebo "nahrazeny" v původním datasetu, což umožňuje jejich opakované výběry. Tato technika může být účinná pro strojové učení algoritmů, které jsou ovlivněny nesymetrickým rozložením a kde více duplicitních příkladů pro danou třídu může ovlivnit přizpůsobení modelu. To může zahrnovat algoritmy, které iterativně učí koeficienty, jako jsou umělé neuronové sítě, které používají stochastickou gradientní metodu. Může to také ovlivnit modely, které hledají dobré rozdělení dat, jako jsou podpůrné vektorové stroje a rozhodovací stromy. Může být užitečné ladit cílové rozdělení tříd. V některých případech hledání vyváženého rozdělení pro silně nesymetrický dataset může způsobit, že ovlivněné algoritmy přeučí menšinovou třídu, což vede k zvýšené obecné chybě generalizace. Účinek může být lepší výkon na trénovacím datasetu, ale horší výkon na testovacím datasetu. Na Obr. 12 je ukázka implementace kódu v Pythonu. [14]

```
# Our data is biased and should be oversampled
oversample = RandomOverSampler(sampling_strategy='minority')
X_oversampled, y_oversampled = oversample.fit_resample(dataset.drop(columns='stroke'), dataset['stroke'])
oversampled_dataset = X_oversampled
oversampled_dataset['stroke'] = y_oversampled
```

Obr. 12: Implementace oversamplingu v Pythonu.

Parametr **sampling_strategy** funkce **RandomOverSampler** určuje to, co se má převzorkovat. Po převzorkování vyšlo, že počet pacientů s mrtvicí je roven počtu pacientů bez mrtvice. Teď, když data jsou připravená a očištěná, nastává rozdělení datasetu na trénovací a testovací části.

3.2.2 Train test split a standardizace

Train test split je funkce, která rozděluje dataset na trénovací a testovací části. Dělá se to hlavně pro to, abychom mohli zjistit přesnost modelu na neznámých pro ten model datech.

```
# Create training and testing set
X_train, X_test, y_train, y_test = train_test_split(X_oversampled.drop(columns='stroke'), y_oversampled,
                                                    test_size=0.3, random_state=42)
```

Obr. 13: Train test split. X_train a X_test je vstup (atributy, podle kterých se určuje výsledná třída y_train a y_test)

Standardizace pomáhá modelu efektivněji zpracovávát data tím, že se mění rozsah, ale nemění se rozložení. Podíváme se na příklad rozložení spojitých atributů před a po standardizaci na Obr. 14. Standard scaler funguje následovně: funkce upraví každý atribut tak, aby měl směrodatnou odchylku rovnou 1 a střední hodnotu rovnou nule.

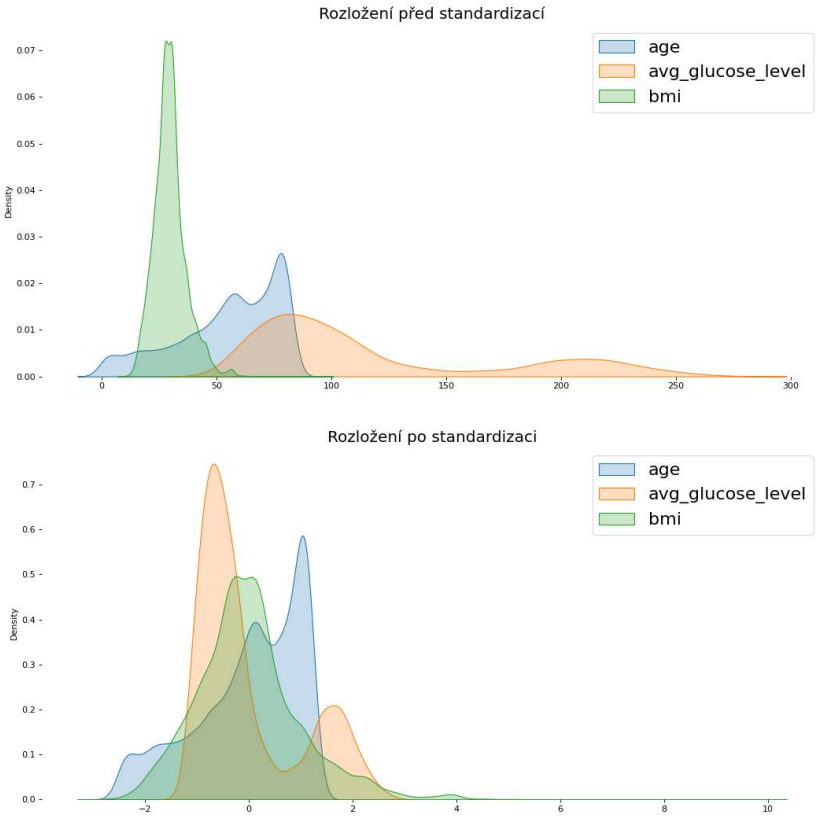
$$z = \frac{x - \mu}{\sigma}, \tag{45}$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i) \tag{46}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{47}$$

kde σ je směrodatná odchylka a μ je střední hodnota.

Teď, když máme očištěná, připravená a rozdělená data, nastává krok vytvoření modelu. Pro zajištění neměnností dat, upravený dataset mnou byl uložen pro následující požití v jazycích R a Matlab.



Obr. 14: Rozložení stejného datasetu před a po standardizaci. Z obrázku je vidět, že se nemění rozložení, ale pouze jeho rozsah.

3.2.3 Tvorba modelu

Samotná tvorba modelu se skládá z několika částí v závislosti na tom, co všechno je potřeba u modelu nadefinovat. Každý modul má předem nastavené hyperparametry a typ jádrové funkce, a proto výchozí výsledek nemusí být nejlepší. Pro dosažení nejlepších výsledků se zkouší různé hyperparametry a jádrové funkce, ale jelikož manuální nastavení všech parametrů je časově náročné, vědci byly vymyšlené algoritmy pro vyhledávání nejlepších hyperparametrů. Při tvorbě modelu mnou byl použit algoritmus RandomSearchCV, který náhodně dosazuje hyperparametry do modelu a vyhledává nejvíc optimální řešení. Funkce RandomSearchCV vyžaduje rozsah hyperparametrů, do modelu mnou byly dosaženy následující hyperparametry: $C \in (0{,}1; 1; 10; 100; 1000)$; $\gamma \in (1; 0{,}1; 0{,}01; 0{,}001; 0{,}0001)$. Jako jádrové funkce mnou byly zvolené radiální, lineární a polynomiální funkce.

```
# SVM model
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf', 'linear', 'poly']}

grid = RandomizedSearchCV(SVC(random_state=42), param_grid, refit=True, verbose=3, n_iter=150)
```

Obr. 15: Tvorba modelu v Pythonu.

Argument SVC funkce RandomizedSearchCV určuje použitý klasifikační algoritmus, argument refit má hodnotu True, tzn. že po dosažení nejlepšího výsledků funkce dosadí do modelu příslušné hyperparametry. Argument verbose stanoví počet informačních zpráv o stavu modelu a jeho výsledcích. Poslední argument n_iter určuje maximální možný počet iterací.

3.2.4 Výsledky

Pro lepší pochopení dosažených výsledků je potřeba definovat metriky, podle kterých bylo posouzeno o výkonnosti modelu. Hlavní metrikou je přesnost, která je spočítaná z *confusion matrix* (česky matice záměn). Matice záměn je tabulka používaná k popisu výkonnosti klasifikačních modelů strojového učení. Matice zobrazuje počty správně a nesprávně klasifikovaných příkladů v jednotlivých třídách. V Tab. 2 je zobrazena matice záměn, která reprezentuje přesnost vytvořeného modelu.

Tab. 2: Matice záměn

	Odhad «0»	Odhad «1»
Skutečnost «0»	1367	90
Skutečnost «1»	0	1460

Matice záměn je tvořena čtyřmi hodnotami, které popisují počty příkladů patřících do různých kategorií. Tyto hodnoty jsou:

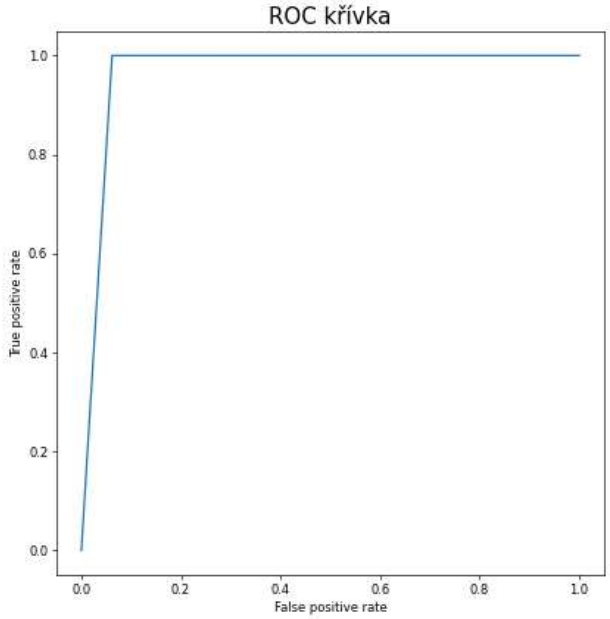
- True Positive (TP) - počet správně klasifikovaných příkladů jako pozitiva (1367)
- False Positive (FP) - počet nesprávně klasifikovaných příkladů jako pozitiva (90)
- True Negative (TN) - počet správně klasifikovaných příkladů jako negativa (1460)
- False Negative (FN) - počet nesprávně klasifikovaných příkladů jako negativa (0)

Z těchto hodnot se vypočítají různé metriky, v našem případě použijeme přesnost.

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN} \sim 0.97 = 97\% \tag{48}$$

Dalším velice reprezentativím způsobem ukázat vykonnost modelu je ROC-křívka. Dále v této práci bude použita jenom přesnost a ROC křívka. ROC křivka zobrazuje vztah mezi citlivostí a specifičností klasifikátoru při různých prahových hodnotách pro rozhodování. Citlivost (též true positive rate) udává podíl správně klasifikovaných pozitivních případů na celkový počet pozitivních případů, zatímco specifičnost (též true negative rate) udává podíl správně klasifikovaných negativních případů na celkový počet negativních případů. Ideální klasifikátor by měl mít ROC křivku, která prochází levým horním rohem grafu, což by znamenalo maximální citlivost a specifičnost. Na Obr. 16 je zobrazena ROC

křívka pro model vytvořený v Pythonu. Dále v této práci bude použita jenom přesnost a ROC křívka. [15]



Obr. 16: ROC křívka pro Python model.

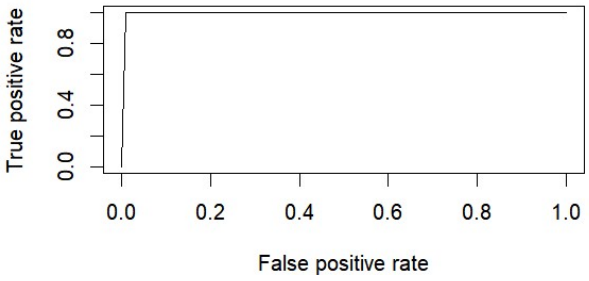
Tab. 3: Výsledky modelu v Pythonu

Přesnost	Rychlost	Gamma	C	Jádrová funkce
96.8%	52.5s	1	10	Radial basis

Je důležité zmínit, že vysoká rychlost v Pythonu je dana funkcí RandomSearchCV, která nedosazuje všechny možné hyperparametry, ale vybírá náhodně, tím pádem probíhá menší počet iterací a program je rychlejší. Pro dosažení ale dobrého výsledku mnou bylo vyzkoušeno několik *random seedů*.

3.3 Implementace v R

Z hlediska programování implementace v R byla jednodušší než v Pythonu, protože všechny potřebné úpravy dat byly provedeny v Pythonu. Výkonnost modelu vytvořeného v R je uvedena na Obr. 17 a výsledky jsou vypsány do Tab. 4.



Obr. 17: ROC křívka pro R model.

Tab. 4: Výsledky modelu v R

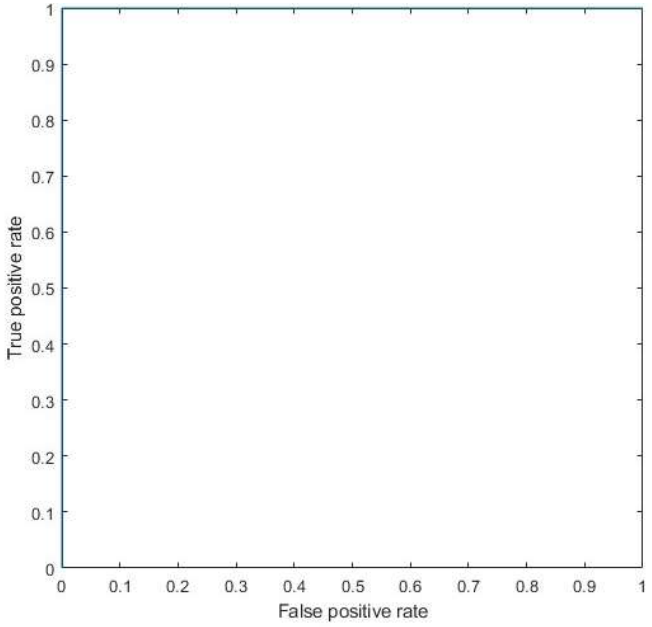
Přesnost	Rychlost	Gamma	C	Jádrová funkce
99.2%	1028s	10	1	Radial basis

Jak můžeme vidět, model v R dosahuje velice vysoké přesnosti, ale je výrazně pomalejší než Python. Je to dáno tím, že model používá jinou metodu validace různých hyperparametrů. Například, kdyby mnou byla použita podobná metoda validace v Pythonu, spouštění programu by trvalo řádově déle než v R.

Při porovnání těchto dvou výsledků je důležité se zamyslit nad tím, co v dané úloze má větší význam – rychlost nebo přesnost? V našem případě přesnost hraje větší roli, poněvadž účelem našeho modelu je odhadnout zda dojde k mrtvici nebo nedojde.

3.4 Implementace v Matlab

Po porovnání tří různých softwarů lze stanovit, že implementace v Matlabu je programatorsky nejméně náročná. V R a v Pythonu neexistují žádné nativní knihovny, kdežto Matlab obsahuje velkou řádu knihoven, jedna ze kterých je nacílená na vytvoření metod strojového učení. Podíváme se na výsledky implementace v Matlabu:



Obr. 18: ROC křívka pro Matlab model.

Tab. 5: Výsledky modelu v Matlab

Přesnost	Rychlost	Gamma	C	Jádrová funkce
100%	434s	0.0307	59.7168	Radial basis

Na první pohled model dává pochybný výsledek 100%, což znamená, že klasifikoval všechna testovací pozorování bez chyb. Pro lepší pochopení takového výsledku, podíváme se na varianty, kdy je ten výsledek chybný:

1. Overfitting – model se dokonale naučil na trénovácím datasetu, ale nedokáže zpracovat nová data, což není zrovna naším případem, protože přesnost kontrolujeme na datasetu, na kterém model nebyl postaven.
2. Struktura dat – pokud data v datasetu jsou uspořádaná podle příznaku, který máme odhadnout (první polovina má mrtvici, druhá polovina nemá) pak model se nebude učit tvořit nadrovinu, ale bude hledat právě ten přechodný datapoint, po kterém dochází ke změně hodnoty příznaku. V našem případě jsme tomu předešli tak, že dataset byl náhodně zamíchán před použitím ve všech softwarech.

3. Malý počet pozorování vůči počtu příznaků (taky dochází k overfittingu)

Je zřejmé, že žádný z těchto důvodu nemohl způsobit přesnost 100%, proto můžeme předpokládat, že výsledek je legitimní. Je to dáno vhodným rozložením dat, velkým počtem pozorování a dalšími faktory, které ovlivňují přesnost modelu.

4 ZÁVĚR

Metoda SVM je důležitou částí strojového učení a umělé inteligence jako takové. SVM je velice flexibilní metoda díky možnosti použití různých jádrových funkcí pro práci s různými typy dat a účinnost této metody umožňuje rychle a přesně trénovat klasifikační modely i na malém počtu dat.

Na začátku teoretické části byly vysvětleny rozdíly mezi jednotlivými typy úloh strojového učení a jaké existují typy metod. Dále jsme se podívali na to, jakým způsobem je tvořen primitivní klasifikátor pomocí separační nadroviny. Vysvětlili jsme si, že šířka marginu nezávisí na všech bodech, ale pouze na bodech, které leží na hranici toho marginu, což jsou podpůrné vektory. K tvorbě obecnějšího klasifikátoru bylo využito tzv. soft marginu, který dovoluje některým pozorováním ležet na špatné stráně nadroviny. Nakonec, ukázali jsme si, jak jádrové funkce řeší problém neseparovatelnosti dat.

Praktická část se skládala z implementace SVM ve tří různých softwarech a porovnání výsledků těchto implementací na úloze klasifikace. Implementace v Pythonu byla z hlediska programování a samotné práce nejtěžší, poněvadž v sobě zahrnovala víc kroků. Pro práci s daty zaprvé je bylo potřeba prozkoumat a zjistit případné problémy. Po kontrole dat se zjistilo, že data měla dva zásadní problémy: nevyváženost jednotlivých tříd a chybějící hodnoty. Tyto problémy mohly značně působit na přesnost výsledků, proto je vždycky potřeba mít „očištěný“ dataset. Nevyváženost datasetu mnou byla vyřešena tzv. náhodným převzorkováním, kde jsme do datasetu přidali ta samá data několikrát, abychom vyrovnali jednotlivé třídy. Problém chybějících řádků šlo vyřešit několika způsoby, ale nakonec mnou bylo rozhodnuto, že nahradit chybějící hodnoty odhadem pomocí metody rozhodovacích stromů je nejefektivnější řešení. Tím jsme zachovali původní velikost datasetu. Po očištění dat bylo potřeba data připravit pro použití v modelu. Příprava dat se skládala ze standardizace a rozdělení dat na testovací a trénovací datasety. Standardizací jsme zajistili to, že všechny příznaky mají stejné měřítko a rozptyl. Bylo to potřeba provést proto, že SVM se přizpůsobuje vzdálenosti mezi jednotlivými body v prostoru příznaků. Rozdělením dat na dva datasety jsme zajistili, že výsledná přesnost modelu byla kontrolována na odlišném datasetu než na kterém dataset byl naučen. Dále následovalo samotné budování modelu a jeho spuštění.

Předem připravený v Pythonu dataset mnou byl následovně nahrán do R a do Matlabu. Budování modelu v R a v Matlabu je velice podobné budování modelu v Pythonu, proto v práci byla jenom ukázka kódu v Pythonu. Implementace jako taková nebyla obtížná, poněvadž předem připravený dataset je kompatibilní s jakýmkoliv programem, který umožňuje zpracovávat data.

Po porovnání výsledků z tabulek Tab. 3, Tab. 4 a Tab. 5 bylo patrné, že model v Matlabu dosahuje největší přesnosti. Je zajímavé, že všechny modely použily stejnou jádrovou funkci, ale různé hodnoty hyperparametrů gamma a C. Model v Matlabu je sice pomalejší než Python, ale je to dáno typem použité validační funkce nikoliv špatnou

optimalizací algoritmu v Matlabu. Z hlediska přesnosti je R na druhém místě a je také výrazně přesnější než Python.

Na první pohled se může zdát, že dosažena přesnost 100% v Matlabu je pochybná, ale je to vysvětlitelné. Na začátku práce jsme ukazovali primitivní klasifikátor pomocí nadroviny a ideálně separovatelný případ. Když se nad tím zamyslíme, tak z velké části separovatelnost je dána rozložením dat v prostoru. Pro ukázku toho, jaký vliv mají data na výslednou efektivitu modelu mnou byl schválně vyhledán dataset, který má natolik vhodné rozložení v prostoru. Velkou roli v dosaženém výsledku také hrála úprava a očištění dat.

Na výsledek měly vliv dva hlavní faktory: výkonnost zařízení, na kterém model byl spouštěn a kvalita datasetu. Všechny implementace mnou byly provedeny na stejném počítači, při stejných podmínkách, což ale znamená, že na silnějším zařízení by spouštění modelu mohlo trvat řádově méně času. Práci bychom mohli také obohatit o implementaci jiného datasetu ve stejných softwarech. Je taky důležité zmínit, že poněvadž výsledek se může lišit v závislosti na úpravě a očištění dat, je možné dosáhnout lepších výsledků při detailnějších úpravách, například použitím metod jako Feature Engineering (český *inženýrství příznaků*) a Feature Selection (český *výběr příznaků*).

SEZNAM POUŽITÉ LITERATURY

[1] GÉRON, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems. Second edition. Beijing: O'Reilly, 2019. ISBN 978-149-2032-649.

[2] MURPHY, Kevin P. Probabilistic machine learning: an introduction. Cambridge, Massachusetts: The MIT Press, [2022]. Adaptive computation and machine learning (The MIT Press). ISBN 978-026-2046-824.

[3] JAMES, Gareth, Daniela WITTEN, Trevor HASTIE a Robert TIBSHIRANI. An introduction to statistical learning: with applications in R. New York: Springer, [2013]. Springer texts in statistics. ISBN 978-146-1471-370.

[4] VAPNIK, Vladimir N. Statistical Learning Theory. Wiley-Interscience, 1998. ISBN 0471030031.

[5] HASTIE, Trevor, Robert TIBSHIRANI a Jerome FRIEDMAN. The elements of statistical learning: data mining, inference, and prediction. 2nd ed. New York: Springer, c2009. Springer series in statistics. ISBN 978-038-7848-570.

[6] A. GORINI, Catherine. The Facts On File Geometry Handbook (Facts on File Science Handbooks). Revised edition. New York: Facts on File, (July 1, 2009). ISBN 978-081-6073-894.

[7] AVATI, Anand. Bias-Variance Analysis: Theory and Practice [online]. California, Stanford, 2019 [cit. 2023-05-01]. Dostupné z: http://cs229.stanford.edu/summer2019/BiasVarianceAnalysis.pdf. Studjiní opory. Stanford.

[8] Reuters-21578, Distribution 1.0 [online]. AT&T Labs - Research, 1987 [cit. 2023-05-04]. Dostupné z: https://statics.teams.cdn.office.net/evergreen-assets/safelinks/1/atp-safelinks.html

[9] MOGUERZA, Javier M. a Alberto MUÑOZ. Support Vector Machines with Applications. Statistical Science [online]. 2006, 21(3) [cit. 2023-05-01]. ISSN 0883-4237. Dostupné z: doi:10.1214/088342306000000493

[10] SINGH, Manmohan, Rajendra PAMULA a Shudhanshu kumar SHEKHAR. Email Spam Classification by Support Vector Machine. 2018 International Conference on Computing, Power and Communication Technologies (GUCON). IEEE, 2018, 2018, 878-882. ISBN 978-1-5386-4491-1. Dostupné z: doi:10.1109/GUCON.2018.8674973

[11] CYHELSKÝ, Lubomír a Vladimíra VALENTOVÁ. Importance of the basic classification of indicators for the correct interpretation of the mutual differences between their values. Politická ekonomie. 2006, 54(4), 542-548. ISSN 00323233. Dostupné z: doi:10.18267/j.polek.573

[12] Stroke Prediction Dataset [online]. Kaggle, 2021 [cit. 2023-05-04]. Dostupné z: https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset

[13] GARCÍA, Carlos Domínguez. Visualizing the effect of hyperparameters on Support Vector Machines [online]. 2021 [cit. 2023-05-01]. Dostupné z: https://towardsdatascience.com/visualizing-the-effect-of-hyperparameters-on-support-vector-machines-b9eef6f7357b

[14] BROWNLEE, Jason. Random Oversampling and Undersampling for Imbalanced Classification [online]. 2019 [cit. 2023-05-01]. Dostupné z: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/

[15] GOOGLE. Classification: ROC Curve and AUC [online]. [cit. 2023-05-01]. Dostupné z: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc