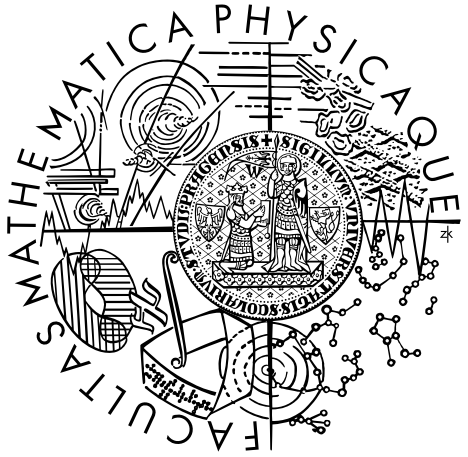
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Zdeněk Tesař

Klasifikace dat z posturografických měření

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Tomáš Bílý
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2015

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Klasifikace dat z posturografických měření

Autor: Zdeněk Tesař

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Tomáš Bílý, Katedra aplikované matematiky

Abstrakt: Tato bakalářská práce popisuje koncept a implementaci aplikace v jazyce C# sloužící pro klasifikaci poruch stabilizačního systému pacientů na základě dat, která byla získána z posturografických měření. Tato aplikace byla autorem práce vytvořena na základě spolupráce s odborníky z 2. lékařské fakulty Univerzity Karlovy v Praze. Zde také proběhlo její první experimentální nasazení pro testovací a výzkumné účely.

Klíčová slova: posturografie, klasifikace dat, umělé neuronové sítě, strojové učení

Title: Data classification from posturographic measurements

Author: Zdeněk Tesař

Department: Department of Applied Mathematics

Supervisor: Mgr. Tomáš Bílý, Department of Applied Mathematics

Abstract: This bachelor's thesis describes the design and implementation of a C# application for classifying disorders of patients' balance systems using data acquired from posturographic measurements. The application was created by the paper's author on the basis of a collaborative effort with experts from the 2nd Faculty of Medicine at Charles University in Prague, which was also the site of its first experimental use for testing and research purposes.

Keywords: posturography, data classification, artificial neural networks, machine learning

Rád bych poděkoval Mgr. Tomáši Bílému za vedení bakalářské práce a cenné rady a připomínky při její tvorbě. Dále PhDr. Ondřeji Čakrtovi, Ph.D. z 2 lékařské fakulty Univerzity Karlovy v Praze za možnost spolupráce na projektu klasifikace pacientů s poruchami stabilizačního systému, jehož výstupem je tato bakalářská práce a související odborný článek v časopise Česká a slovenská neurologie a neurochirurgie. V neposlední řadě bych chtěl poděkovat Ing. Tomáši Fundovi za poskytnutí MATLABových skriptů a Ing. Václavovi Dedkovi za poskytnutí MATLABu pro převod skriptů do dll knihoven. Na konec děkuji i své rodině a přítelkyni za stálou podporu při studiu a tvorbě této práce.

Obsah

Úvod

Lékaři mají mnoho různých přístrojů, které jim pomáhají s analýzou zdravotního stavu pacientů nebo jejich klasifikací. S tím souvisí i rozdílná míra složitosti jejich výstupů. Může se jednat o jednoduché výstupy, například hodnotu saturace (okysličení) krve z pulsního oxymetru či hodnotu systolického[1] a diastolického[2] krevního tlaku z tonometru (tlakoměru). Vzhledem k rychlému vývoji techniky jsou to však o výjimky a většinou má výstup komplexní charakter – obraz z rentgenového vyšetření, křivka z elektrokardiografu (EKG), obrazová informace ze sonografie (ultrazvukové vyšetření) a další.

U většiny výstupů je zatím člověk se specializací na danou problematiku nenahraditelný. Je to buď z důvodu, že pro kvalitní analýzu či klasifikaci pomocí počítače by bylo potřeba stovek tisíc kvalitně označkovaných dat z daného přístroje, na nichž by se mohl algoritmus učit. S tím ovšem souvisí i potřeba občas data nashromáždit znovu a provést opětovné učení, protože v medicíně stále dochází k objevům nových nemocí a poruch, anebo problém nemusí být natolik složitý po stránce vstupních dat, ale kvůli potřebě analýzy v reálném čase (např. EKG). Zde by se mohl nasadit nějaký algoritmus, který by se implementoval do elektrokardiografů, ale došlo by k jejich výraznému prodražení a výstupy by mohly být vzhledem k neznalosti dalšího stavu pacienta kontraproduktivní.

Existují ovšem i výjimky, kde je výstup z přístroje složitý a i pro specialistu obtížně čitelný, ale počítač by si mohl dle nashromážděných dat po naučení s problémem snadno poradit. Jedním z takových příkladů může být i klasifikace pacientů na základě posturografického vyšetření do předem určených kategorií, které jsou omezeny pouze na ty, jež se týkají problematiky daného specialisty.

Cílů této bakalářské práce je několik. Prvotním je vytvoření funkčního konceptu aplikace, který bude umět spravovat pacienty a data z jejich jednotlivých měření a na jejich základě bude v omezené míře schopen tyto pacienty klasifikovat pomocí umělých neuronových sítí, dále jen NS.

Druhým cílem této práce je rozšíření spolupráce s fyzioterapeuty z 2. lékařské fakulty Univerzity Karlovy v Praze, dále jen 2. LF UK. Hlavním impulsem je právě tato práce a s ní související funkční koncept aplikace.

V první kapitole bakalářské práce je popsána posturografie – o jakou vyšetřovací metodu se jedná, jaké přístroje využívá, popis vlastního měření, výstupních dat i jejich další zpracování. V druhé až šesté kapitole je popsána vlastní implementace konceptu aplikace, použité algoritmy, postupy, ukázky kódu i možnosti jeho dalšího rozšíření. Součástí těchto kapitol je vždy i zmínka o problémech, které při implementaci mohou nastat, jejich řešení a poznámky, čemu se při rozšiřování této aplikace vyhnout, či co již neopakovat. Předposlední kapitola popisuje uživatelskou dokumentaci a součástí je i demonstrace ukázkového použití aplikace v praxi. Poslední kapitola již jen shrnuje výsledky z experimentu, který sloužil jako hlavní výchozí bod pro implementaci konceptu aplikace z hlediska potřebných součástí.

[1] Nejvyšší tlak krve, kterého je dosaženo během stahu srdeční svaloviny (systoly). [1]

[2] Nejnižší tlak krve, kterého je dosaženo během období srdečního klidu mezi dvěma systolami. [1]

Koncept aplikace vznikal ve spolupráci s PhDr. Ondřejem Čakrtem, Ph.D., z jeho návrhů a požadavků.

Jako příloha této práce je vlastní kód aplikace v jazyce C#, úplná programátorská dokumentace k tomuto kódu, anonymní naměřená data z posturografické plošiny a další pomocné programy a skripty převážně v jazycích C#, Mathematica a MATLAB. Na cizí kód a skripty použité v aplikaci se vztahují licenční a autorská práva u nich uvedená. Pro jejich další využití ve vlastních programech se řiďte podmínkami uvedenými v licenčních dokumentech, nebo kontaktujte jejich autory. Ke spuštění kódu vlastní aplikace je vyžadována licence k Extended WPF Toolkit™Community Edition. Tato licence nemůže být v rámci zdrojového kódu zprostředkována, neboť je vázána na jedince a došlo by k porušení práv na její užívání.

1. Posturografie

Jedná se o klinické vyšetření pohybového ústrojí, které umožňuje objektivní zachycení pohybových dějů pomocí přesných číselných údajů. Posturografie (kinetická analýza) je měření, při němž pacient stojí na tenzometrické plošině (měřící tlak), dále posturografická plošina. Ta měří rozklad reakční síly pacienta na pohyb plošiny. Ten se měří ve třech vzájemně kolmých rovinách – anterioposteriorní (předozadní), mediolaterální (boční) a vertikální. Primární akční silou působící na plošinu je tíhová síla pacienta, sekundárními silami považujeme reakční sílu svalů udržující rovnováhu a vyrovnávající pohyb plošiny. Na tyto akční síly reaguje plošiny dle zákona akce a reakce. [2]

1.1 Posturografická plošina

Posturografická plošina je analyzační přístroj skládající se z několika částí. První z nich je pevná platforma (deska) s bezpečnostními madly, na níž pacient během vyšetření stojí na přesně určeném místě vyhrazeném obrázky plosek nohou. V rozích této platformy se standardně nacházejí piezoelektrické tenzometry snímající momenty reakčních sil. [2] Naměřené hodnoty jsou poté zpracovány analogovým převodníkem signálu a poslány k dalšímu zpracování do PC. Dalšími součástmi bývají i elektromotory pro pohyb platformy, či například měkká podložka pro změnu proprioceptivních[1] informací pro vyšetření pacienta s rovnováhou narušenou vnějším podnětem.

Všechna data v této práci pocházejí z posturografické plošiny Synapsys Posturography System, dále jen SPS, s následujícími specifikacemi.

Technické vlastnosti:

- Pevná platforma se třemi piezoelektrickými tenzory.
- 16bitový analogový převodník.
- Vzorkování: 100 Hz.
- Váhový limit: 130 Kg.
- Součástí kompletního systému jsou: statická platforma, nestabilní platforma, pohyblivá platforma, pozicovač plosek nohou, bezpečností madla a dřevěný stojan.

Doporučené vybavení PC & OS:

- Windows XP nebo Vista.
- Pentium 4.
- 512 MB RAM.
- Grafická karta ATI X300 nebo lepší.
- VNG Ulmer kompatibilní.
- CE Class I.

[1] Propriocepce je pojem pro hlubokou citlivost tvořenou mechanoreceptory Golgiho šlachových tělísek a svalových vřetének. [3]

- Vyhovující EN60601-1.
- Není schváleno FDA. [4]

1.2 Průběh vyšetření a zaznamenaná data

Posturografická vyšetření se dělí na dva základní typy – statická posturografie, dále jen stabilometrie, a dynamická posturografie. V našem případě se budeme zabývat pouze prvním typem, tedy měřením, kde nedochází k pohybu plošiny.

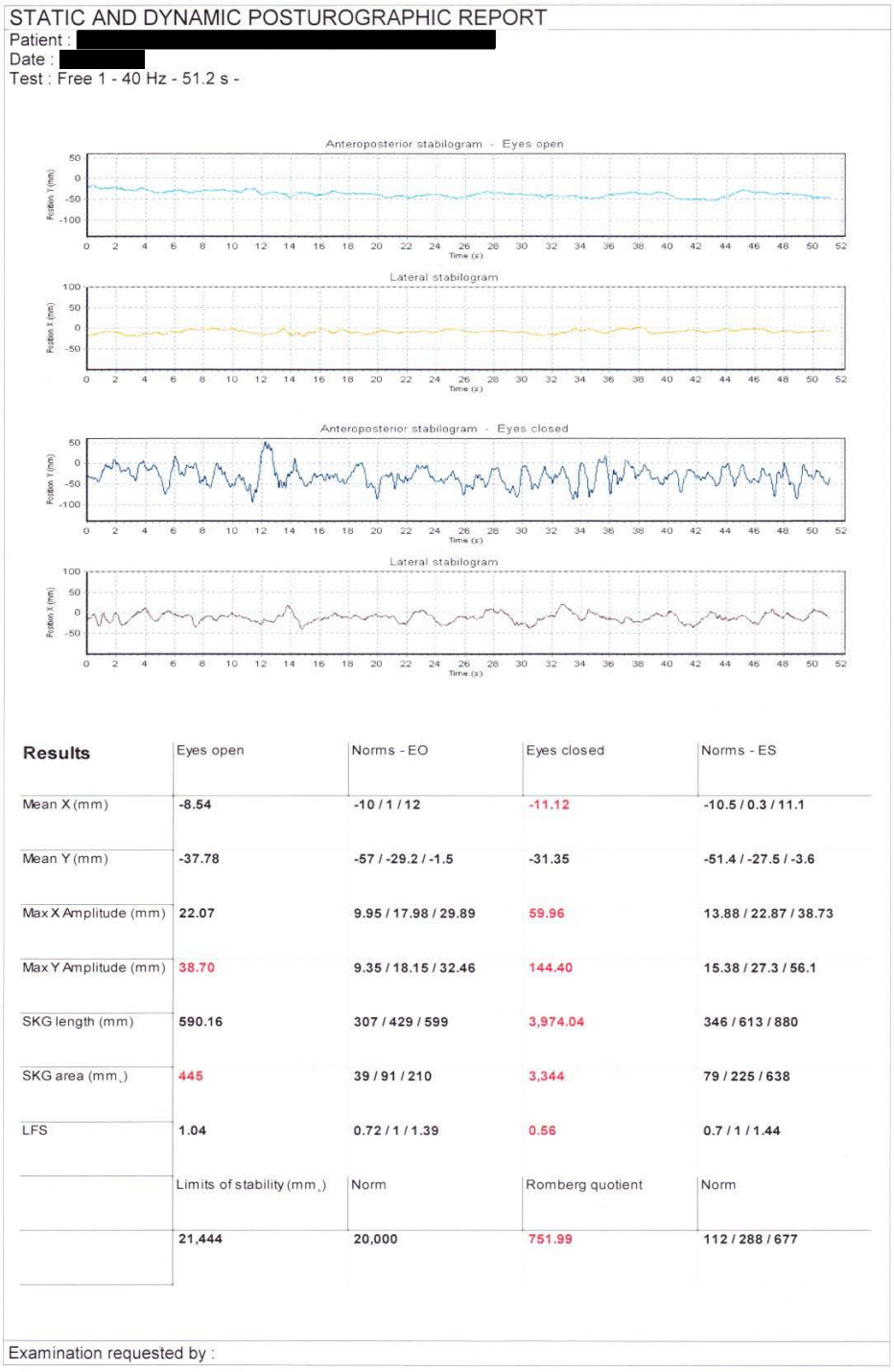
Pro potřeby naší práce se budeme zabývat stabilometrickým měřením, které se skládá ze čtyř částí – měření na pevné podložce s otevřenýma očima, měření na pevné podložce se zavřenýma očima, měření na měkké podložce s otevřenýma očima a měření na měkké podložce se zavřenýma očima. Každá z těchto čtyř částí probíhá po dobu 51,2 sekundy se vzorkovací frekvencí 40 Hz. Pacient při nich klidně stojí se současnou fixací pohledu do jednoho bodu. Toto vyšetření nám může umožnit diferenciaci oblastí centrální nervové soustavy, dále jen CNS, podílejících se na udržování stability, například zdali je porucha způsobena vestibulárním centrem ve středním uchu, či centry začleněnými do udržování rovnováhy v mozečku a v mozkovém kmeni.

Z každé části dostaneme celkem šest výsledných kanálů. Nás budou zajímat pouze první dva – odchylka „center of pressure“ (působiště reakční síly), dále jen COP, v anterioposteriorní rovině a odchylka COP v mediolaterální rovině. Každý kanál obsahuje na základě výše uvedeného času měření a vzorkovací frekvence celkem 2 048 vzorků. Z takto nasnímaných hodnot lze snadno matematickou úpravou vypočítat COP. COP reprezentuje vážený průměr všech tlakových sil působících na posturografickou plošinu. Plošina tedy registruje polohu COP a její změnu v čase. [2] Z těchto hodnot poté specializovaný program vypočítá výstupní parametry tenzomotorické plošiny, které zmíníme posléze.

Obr. 1.1, který zachycuje první stranu standardního výstupu ze SPS. Tento výstup je vždy společný pro dvě části (otevřené a zavřené oči při stejné podložce). V první polovině stránky lze vidět právě grafické znázornění hodnot z prvních dvou kanálů – grafy odchylky COP v čase pro anterioposteriorní a mediolaterální rovinu při otevřených a při zavřených očích. Tyto grafy budou dále označovány jako stabilogramy. Ve spodní části stránky jsou vypočtené parametry, vizme dále.

1.3 Výstupní parametry

SPS v rámci posturografického reportu dává i vypočtené hodnoty výstupních parametrů, vizme spodní polovinu obrázku 1.1. Pro potřeby další analýzy a klasifikace pacientů jsou ovšem tyto parametry nedostačující z důvodu malého počtu a odchylek jejich hodnot oproti standardizovaným výpočtům těchto parametrů uvedeným například v článku Thomase E. Prieta et al [5], dále jen Prieto, a v článku Christophera Maurera a Roberta J. Peterky [6], dále jen Maurer. Dále si představíme všechny parametry, ale pro další potřeby této práce ty ze SPS zanedbáme. U specialistů se spíše než s klasifikací pomocí výstupních parametrů setkáváme s klasifikací pomocí vlastností křivky signálu zpracovaného pomocí rychlé Fourierovy transformace, dále jen FFT, vizme horní stranu obrázku 1.2.


STATIC AND DYNAMIC POSTUROGRAPHIC REPORT

Patient : [redacted]

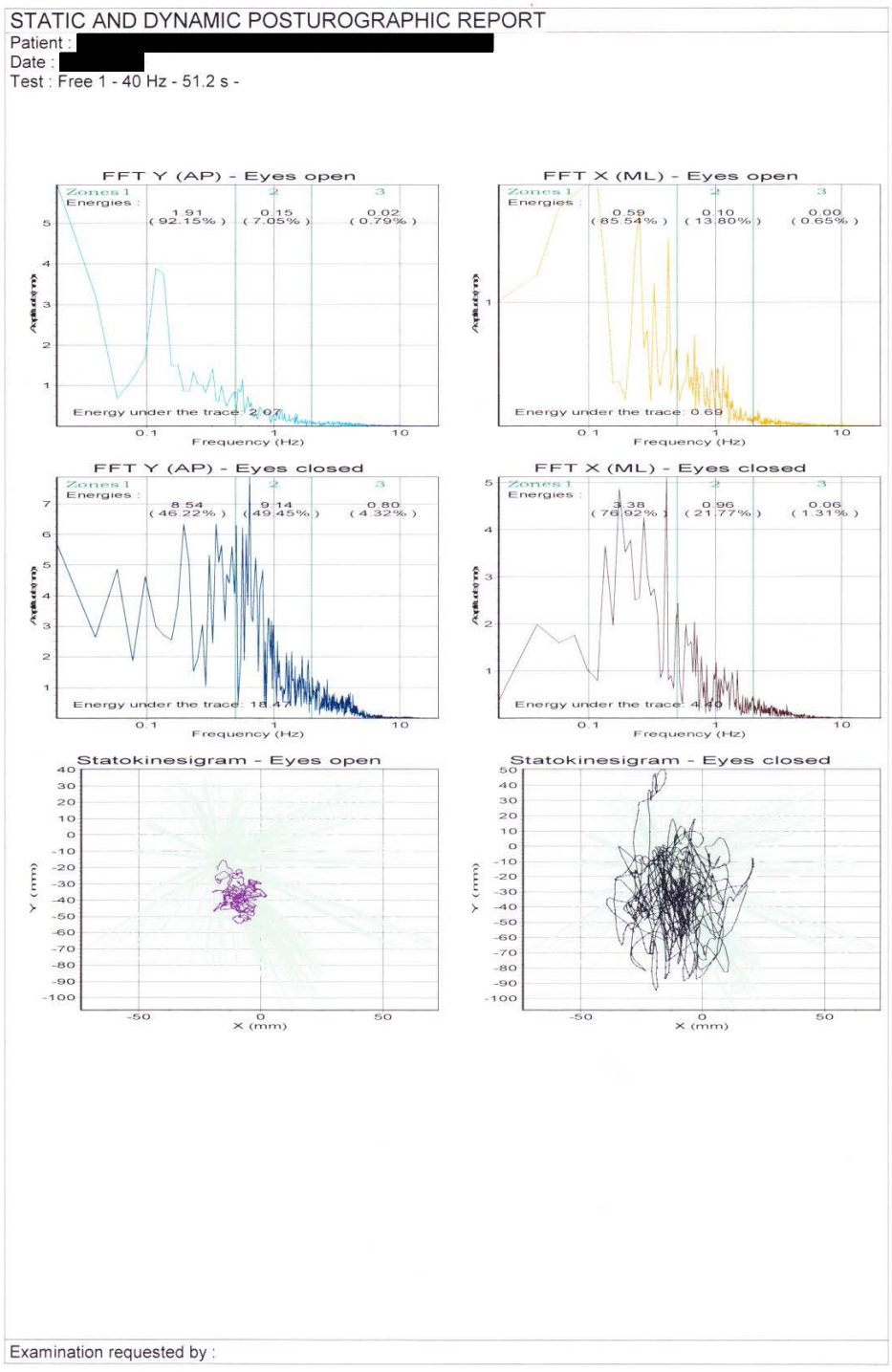
Date : [redacted]

Test : Free 1 - 40 Hz - 51.2 s -

Results	Eyes open	Norms - EO	Eyes closed	Norms - ES
Mean X (mm)	-8.54	-10 / 1 / 12	-11.12	-10.5 / 0.3 / 11.1
Mean Y (mm)	-37.78	-57 / -29.2 / -1.5	-31.35	-51.4 / -27.5 / -3.6
Max X Amplitude (mm)	22.07	9.95 / 17.98 / 29.89	59.96	13.88 / 22.87 / 38.73
Max Y Amplitude (mm)	38.70	9.35 / 18.15 / 32.46	144.40	15.38 / 27.3 / 56.1
SKG length (mm)	590.16	307 / 429 / 599	3,974.04	346 / 613 / 880
SKG area (mm²)	445	39 / 91 / 210	3,344	79 / 225 / 638
LFS	1.04	0.72 / 1 / 1.39	0.56	0.7 / 1 / 1.44
	Limits of stability (mm²)	Norm	Romberg quotient	Norm
	21,444	20,000	751.99	112 / 288 / 677

Examination requested by :

Obrázek 1.1: STATIC AND DYNAMIC POSTUROGRAPHIC REPORT, str. 1



Obrázek 1.2: STATIC AND DYNAMIC POSTUROGRAPHIC REPORT, str. 2

1.3.1 Parametry vypočtené SPS [7]

Statokinesigram, dále jen SKG, je grafické vektorové znázornění pohybu COP během měření. Ukázku SKG si lze prohlédnout v dolní části obrázku 1.2.

Mean X (mm) Průměrná pozice COP v mediolaterální rovině v mm.

Mean Y (mm) Průměrná pozice COP v anterioposteriorní rovině v mm.

SKG length (mm) Délka trajektorie, kterou urazil COP v průběhu měření, v mm.

SKG area (mm²) Plocha konfidenční elipsy[2] (90 %) v mm².

LFS Vyjadřuje množství vydané energie vůči referenční hodnotě, aby se člověk udržel zpříma.

Romberg quotient Poměr dráhy a plochy COP při otevřených i zavřených očích. [8]

Max X Amplitude (mm) Maximální amplituda posturální oscilace v mediolaterální rovině (maximální odchylka COP ve vztahu k základní hodnotě) v mm.

Max Y Amplitude (mm) Maximální amplituda posturální oscilace v anterioposteriorní rovině (maximální odchylka COP ve vztahu k základní hodnotě) v mm.

1.3.2 Parametry dle Prieta [5] a Maurera [6]

Následující parametry budeme považovat pro práci za výchozí. Ve zbytku práce budeme jako *COPx* označovat vektor všech hodnot z prvního kanálu SPS – odchylky COP v mediolaterální rovině, a jako *COPy* vektor všech hodnot z druhého kanálu SPS – odchylky COP v anterioposteriorní rovině. N je počet prvků ve vektoru.

Pro snadnější matematický zápis následujících parametrů si nejprve definujme následující operace:

Definice. *Buďte* $(u_1, u_2, \ldots, u_n) \in \mathcal{R}^n$ *a* $(v_1, v_2, \ldots, v_n) \in \mathcal{R}^n$ *dva vektory o stejné délce a* $M \in \mathcal{R}^{n \times n}$ *čtvercová matice. Pak máme operace*

- *Prvkové odčítání:*

$$(u_1, u_2, \ldots, u_n) - (v_1, v_2, \ldots, v_n) = (u_1 - v_1, u_2 - v_2, \ldots, u_n - v_n)$$

- *Prvkové umocňování:*

$$(u_1, u_2, \ldots, u_n) .^\wedge (v_1, v_2, \ldots, v_n) = (u_1^{v_1}, u_2^{v_2}, \ldots, u_n^{v_n})$$
$$(u_1, u_2, \ldots, u_n) .^\wedge j = (u_1^j, u_2^j, \ldots, u_n^j)$$

- *Prvkové odmocňování:*

$$\sqrt{(u_1, u_2, \ldots, u_n)} = (\sqrt{u_1}, \sqrt{u_2}, \ldots, \sqrt{u_n})$$

- *Prvkové násobení:*

$$(u_1, u_2, \ldots, u_n).*(v_1, v_2, \ldots, v_n) = (u_1 v_1, u_2 v_2, \ldots, u_n v_n)$$

[2] „Konfidenční elipsa je plocha zahrnující největší soustředění změn polohy COP při měření. V praxi se nejčastěji používá plocha 90 % či 95 % z celkové plochy všech COP.“ [2]

- *A je kovarianční maticí vektorů u a v:*

$$A = \mathrm{cov}(u, v)$$

- *w je vektor vlastních čísel matice M:*

$$w = \mathrm{eig}(M)$$

, kde priorita prvkových operací je totožná s neprvkovými variantami.

Před samotným výpočtem jednotlivých parametrů je třeba od jednotlivých vektorů odečíst jejich střední hodnotu.

$$COPx = COPx - \frac{1}{N}\sum_{i=1}^{N} COPx_i$$

$$COPy = COPy - \frac{1}{N}\sum_{i=1}^{N} COPy_i$$

Vzorce pro výpočty parametrů *POWERml* až *Fdispap* nebudou v textu uvedeny z důvodu použití metod pro frekvenční analýzu. Jejich definici je možno nalézt v přiložených skriptech Ing. Tomáše Fundy.

MD Průměrná pozice COP.

$$MD = \frac{1}{N}\sum_{i=1}^{N} \sqrt{COPy \mathbin{.\hat{}} COPy + COPx \mathbin{.\hat{}} COPx}_i$$

MDML Průměrná pozice COP v mediolaterální rovině.

$$MDML = \frac{1}{N}\sum_{i=1}^{N} |COPx_i|$$

MDAP Průměrná pozice COP v anterioposteriorní rovině.

$$MDAP = \frac{1}{N}\sum_{i=1}^{N} |COPy_i|$$

MV Průměrná rychlost pohybu COP.

$$\begin{aligned} line = {} & ((COPy_2, \ldots, COPy_N) - (COPy_1, \ldots, COPy_{N-1})) \mathbin{.\hat{}} 2 \\ & + ((COPx_2, \ldots, COPx_N) - (COPx_1, \ldots, COPx_{N-1})) \mathbin{.\hat{}} 2 \\ MV = {} & \frac{1}{N}\sum_{i=1}^{N} \frac{1}{T} line_i \end{aligned}$$

MVML Průměrná rychlost pohybu COP v mediolaterální rovině.

$$MVML = \frac{1}{N}\sum_{i=1}^{N} \left| \frac{(COPx_2, \ldots, COPx_N) - (COPx_1, \ldots, COPx_{N-1})}{T}_i \right|$$

MVAP Průměrná rychlost pohybu COP v anterioposteriorní rovině.

$$MVAP = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{(COPy_2,\ldots,COPy_N)-(COPy_1,\ldots,COPy_{N-1})}{T}_i\right|$$

RMSdis Průměrná kvadratická vzdálenost COP.

$$RMSdis = \frac{\|\sqrt{COPy\ .\hat{}\ 2 + COPx\ .\hat{}\ 2}\|}{\sqrt{N}}$$

RMSMLdis Průměrná kvadratická vzdálenost COP v mediolaterální rovině.

$$RMSMLdis = \frac{\|COPx\|}{\sqrt{N}}$$

RMSAPdis Průměrná kvadratická vzdálenost COP v anterioposteriorní rovině.

$$RMSAPdis = \frac{\|COPx\|}{\sqrt{N}}$$

RMSvel Průměrná kvadratická rychlost pohybu COP.

$$RMSvel = \frac{\|\frac{line}{T}\|}{\sqrt{N}}$$

RMSMLvel Průměrná kvadratická rychlost pohybu COP v mediolaterální rovině.

$$RMSMLvel = \frac{\|\frac{(COPx_2,\ldots,COPx_N)-(COPx_1,\ldots,COPx_{N-1})}{T}\|}{\sqrt{N}}$$

RMSAPvel Průměrná kvadratická rychlost pohybu COP v anterioposteriorní rovině.

$$RMSAPvel = \frac{\|\frac{(COPy_2,\ldots,COPy_N)-(COPy_1,\ldots,COPy_{N-1})}{T}\|}{\sqrt{N}}$$

MFREQ Průměrná frekvence sinusoidální oscilace COP.

$$MFREQ = \frac{MV}{2\pi MD}$$

MFREQML Průměrná frekvence sinusoidální oscilace COP v mediolaterální rovině.

$$MFREQML = \frac{MVML}{4\sqrt{\pi}MDML}$$

MFREQAP Průměrná frekvence sinusoidální oscilace COP v anterioposteriorní rovině.

$$MFREQAP = \frac{MVAP}{4\sqrt{\pi}MDAP}$$

AreaCC Plocha konfidenčního kruhu COP (95 %).

$$
\begin{aligned}
F05 &= 3 \\
Z05 &= 1,96 \\
RD &= \sqrt{COPx\,.\hat{}\,2 + COPy\,.\hat{}\,2} \\
MDIST &= \frac{1}{N}\sum_{i=1}^{N} RD_i \\
RDIST &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(RD\,.\hat{}\,2)_i} \\
SDRD &= \sqrt{RDIST\,.\hat{}\,2 - MDIST\,.\hat{}\,2} \\
AreaCC &= \pi(MDIST + Z05\,SDRD)
\end{aligned}
$$

AreaCE Plocha konfidenční elipsy COP (95 %).

$$
\begin{aligned}
meanML &= \frac{1}{N}\sum_{i=1}^{N} COPx_i \\
meanAP &= \frac{1}{N}\sum_{i=1}^{N} COPy_i \\
SDml &= \sqrt{\frac{1}{N-1}\sum_{i=1}^{N} |COPx - meanML|^2} \\
SDap &= \sqrt{\frac{1}{N-1}\sum_{i=1}^{N} |COPy - meanAP|^2} \\
SDapml &= \frac{1}{N}\sum_{i=1}^{N}((COPy - meanAP)\,.\hat{}\,(COPx - meanML))_i \\
AreaCE &= 2\pi F05\sqrt{SDap^2\,SDml^2 - SDapml^2}
\end{aligned}
$$

AreaSW Plocha výkyvů COP.

$$
\begin{aligned}
AreaCE = \frac{1}{2T(N-1)} & \\
\sum_{i=1}^{N-1}((&COPy_2, \ldots, COPy_N)\,.\hat{}\,(COPx_1, \ldots, COPx_{N-1}) \\
-\,(&COPy_1, \ldots, COPy_{N-1})\,.\hat{}\,(COPx_2, \ldots, COPx_N))_i
\end{aligned}
$$

Area Plocha výkyvů COP (kovarianční).

$$
\begin{aligned}
V &= \mathrm{cov}(COPy, COPx) \\
val &= \mathrm{eig}(V) \\
ax &= 2,4478\sqrt{\sqrt{\frac{1}{N-1}\sum |val - \frac{1}{N}\sum val_i|_j^2} \\
Area &= \pi\prod_{i=1}^{M} ax_i
\end{aligned}
$$

POWERml Celková síla COP v mediolaterální rovině.

POWERap Celková síla COP v anterioposteriorní rovině.

F50ml 50 % silové frekvence v mediolaterální rovině.

F50ap 50 % silové frekvence v anterioposteriorní rovině.

F95ml 95 % silové frekvence v mediolaterální rovině.

F95ap 95 % silové frekvence v anterioposteriorní rovině.

Fpeakml Vrchol silové frekvence v mediolaterální rovině.

Fpeakap Vrchol silové frekvence v anterioposteriorní rovině.

Fmeanml Průměrná silová frekvence v mediolaterální rovině.

Fmeanap Průměrná silová frekvence v anterioposteriorní rovině.

Fcentrml Těžištní frekvence v mediolaterální rovině.

Fcentrap Těžištní frekvence v anterioposteriorní rovině.

Fdispml Frekvenční rozptyl v mediolaterální rovině.

Fdispap Frekvenční rozptyl v anterioposteriorní rovině.

SP Trasa výkyvů COP.

$$\begin{aligned} line2 &= ((COPx_2, \ldots, COPx_N) - (COPx_1, \ldots, COPx_{N-1})) .\hat{} \, 2 \\ &\quad + ((COPy_2, \ldots, COPy_N) - (COPy_1, \ldots, COPy_{N-1})) .\hat{} \, 2 \\ SP &= \sum \sqrt{line2_i} \end{aligned}$$

2. Implementace aplikace – obecná část

Aplikace je psána v jazyce C# pro .NET Framework 4.0. Výběr jazyka a cílené verze .NET Frameworku je dán primárním strojem, na kterém se bude aplikace používat. Jelikož software pro SPS vyžaduje PC s Windows XP nebo Vista, netřeba vybírat multiplatformní jazyk, protože předpokládáme, že tento nástroj se bude instalovat na totožný stroj. Navíc je na MFF UK C# primárním jazykem pro výuku programování. Verze .NET Frameworku vyplývá z toho, že Windows XP vyšší verzi, než 4.0, nepodporují. V případě budoucího vylepšení softwaru pro SPS lze snadno povýšit i verzi .NET Frameworku v naší aplikaci a využívat novější koncepty a možnosti. S tím se pojí i nový kompilátor Roslyn, který v budoucnu umožní multiplatformní běh.

2.1 Motivace

Z každého posturografického měření získáme o pacientovi několik údajů – vizuální data (křivka pohybu COP po aplikaci FFT, SKG a další) a parametry, které nám matematicky popisují chování pacienta na posturografické plošině v průběhu měření. Při pohledu na tyto parametry u různých pacientů lze snadno vypozorovat, že hodnota některých z nich se v závislosti na diagnóze výrazně liší. Je možné předpokládat existenci matematického modelu, pomocí něhož jde pacienty na základě těchto výsledků klasifikovat.

V současnosti navíc dochází k velkému rozmachu „machine learningu“ – strojového učení, které poskytuje pokročilé metody pro klasifikování a regresi dat. Ze stejných předpokladů vychází i Krafczyk et al ve svém článku [9], dále jen Krafczyk. Ten v něm klasifikuje pacienty pomocí NS, jejímiž vstupy jsou vybrané parametry (převážně určité integrály z hodnot FFT ve specifickém frekvenčním spektru) z celkem deseti měření. Jejich výběr probíhal na základě násobku směrodatné odchylky hodnoty parametru od jeho referenční hodnoty od zdravého pacienta. V závěru došel ke zhodnocení, že lze vytvořit „pěknou“ – vysoká procentuální úspěšnost klasifikace pacienta – NS, již poté může snadno distribuovat jako „černou skříňku“ i dalším specialistům z oboru.

2.2 Stručná specifikace

Databázová část

- Zobrazování přehledné tabulky pacientů.
- Seskupování pacientů dle specifických sloupců.
- Vzestupné / sestupné řazení pacientů dle hodnoty ve sloupci.
- Zobrazování, přidávání, editování a odebírání pacientů v databázi.
- Zobrazování přehledné tabulky měření.

Klasifikační část

- Načíst pacientova data z měření.
- Vypočítat parametry uvedené v kapitole 1.
- Klasifikovat pacienta na základě parametrů pomocí předem vytvořených a naučených NS.

Kompletní zdrojové kódy včetně programátorské dokumentace je možno nalézt v příloze A.

2.3 Použité knihovny a nástroje

Jedná se o knihovny a nástroje specifické pro obecnou část, nebo týkající se celé aplikace. Doplňky, které byly instalovány jako součást kompletní instalace VS nebudou dále uváděny. Všechny jsou v aktuální verzi k datu odevzdání této bakalářské práce, pouze u doplňku ReSharper používáme verzi 8, nikoliv 9, s nejnovějšími aktualizacemi.

Visual Studio Ultimate [10] Oficiální vývojové prostředí od společnosti Microsoft, dále jen IDE, pro vývoj v jazyce C#/.NET v nejvyšší edici, zdarma dostupné pro studenty/ky MFF UK v rámci programu DreamSpark Premium. Tato edice byla zvolena kvůli dispozici nejpokročilejších nástrojů pro vývoj, správu a ladění kódu – převážně z důvodu nejrozsáhlejších možností vizuálního znázornění struktury aplikace, propojení jednotlivých projektů, tříd, metod...

Team Foundation Server [11] „Team Foundation Server tvoří centrum správy životního cyklu aplikací sady Visual Studio a jeho účelem je efektivnější využití vývojových týmů. Umožňuje účastníkům zapojit se do vývojového procesu pomocí jediného řešení. Dá se použít také ke správě heterogenních projektů a týmů. Team Foundation Server 2013 vám umožní rychlejší tvorbu softwaru.“ [11] Team Foundation Server, dále jen TFS, v naší prácí používáme převážně ve spojení s Visual Studiem Online, dále jen VSO, pro správu verzí kódu a výše uvedené agilní metodiky vývoje software.

Visual Studio Online [12] „Visual Studio Online vychází z možností Team Foundation Serveru a nabízí další cloudové služby. Je to online umístění pro vaše vývojové projekty.“ [12] VSO v naší práci využíváme převážně ve spojitosti s TFS pro správu kódu a SCRUM. V případě budoucího rozšiřování aplikace o CI či vývoje ve větším týmu lze za nízký měsíční poplatek povýšit verzi VSO a získat potřebné nástroje bez nutnosti výměny použitých technologií a kombinací produktů od několika různých společností.

Editor Guidelines [13] Doplněk pro VS umožňující přidání vertikálních vodítek do editoru kódu. Využívá se hlavně pro dodržování stylu kódu naší aplikace, viz dále.

Productivity Power Tools [14] Doplněk pro VS usnadňující vývoj. Využíváme jej převážně kvůli možnostem náhledu nápovědy přímo v kódu aplikace,

vizuálním znázorněním chyb přímo v Solution Exploreru, filtrování, značkám ve scrollbaru, snadnou maximalizaci/obnovení oken s kódem pomocí dvojkliku a další.

ResXManager [15] Doplněk pro VS, který je vhodný pro snadnější práci s jazykovými zdroji aplikace, jejich překladem, správou a dalším.

ReSharper [16] Jedná se o pokročilý nástroj pro usnadnění analýzy kódu, refaktorace, pohybu v kódu a další – například převod „foreach“ cyklů na LINQ metody. Popis úplných možnosti a ukázky použití jsou k nalezení na webu doplňku.

2.3.1 Alternativy

Jako alternativu ke službám společnosti Microsoft lze pro správu verzí kódu využít například GIT, pro který umožňuje VS přímé propojení bez dalších doplňků. Pro rozšíření o možnosti CI lze použít například nástroj TeamCity od společnosti JetBrains. Vše záleží jen na preferencích jednotlivých vývojářů. Pro snadnější vývoj a návaznost na tuto práci ovšem doporučuji zůstat u použitých nástrojů.

2.4 Struktura aplikace

Pro dodržování vysoké kvality kódu je v aplikaci zachována struktura a hierarchie kódu. Ta je znázorněna dělením kódu do jednotlivých solution – celá aplikace, ty obsahují dílčí projekty – databázová část, propojení databáze s UI, testy výpočtů posturografických parametrů... Projekty jsou poté děleny na části kódu přímo v kořeni a v samostatných adresářích – entity v databázové části, komplexní typy v databázové části a další.

Naše práce je strukturována následovně:

StabilometricDataAnalyzerAndVisualizer Jde o solution vlastního finálního konceptu aplikace. Můžeme ji rozdělit na tři hlavní části – projekty související s databází pacientů a NS, projekty pro zpracování parametrů z posturografických měření a práci s NS a jako poslední projekty související s GUI.

StabilometricDataAnalyzerAndVisualizer.Analyzers

StabilometricDataAnalyzerAndVisualizer.Data

StabilometricDataAnalyzerAndVisualizer.DataService

StabilometricDataAnalyzerAndVisualizer.DataServiceHelper

StabilometricDataAnalyzerAndVisualizer.DataServiceTests

StabilometricDataAnalyzerAndVisualizer.DataTests

StabilometricDataAnalyzerAndVisualizer.Posturography32bit

StabilometricDataAnalyzerAndVisualizer.Posturography32Tests

StabilometricDataAnalyzerAndVisualizer.Posturography64bit

StabilometricDataAnalyzerAndVisualizer.Posturography64Tests

StabilometricDataAnalyzerAndVisualizer.UI

PosturographyMeasurementsUtilities Solution s projekty sloužícími pro předzpracování dat a prezentaci možné funkcionality.

ANNPatientsClassifier Jedná se o projekt pro testování jednotlivých NS v praxi před jejich nasazením do hlavní aplikace. Je také možno jej využívat pouze pro výpočet jednotlivých parametrů pro dané měření.

ANNPatientsClassifiersCreator Projekt, který automatickou generuje a učí NS na základě výsledků z experimentu popsaného v kapitole 7.

DBFMeasurementsToCSharpCode Jde o projekt, jenž pomáhá připravovat naměřená data do formy C# kódu pro testování výpočtů parametrů apod.

Podrobnější popisy jednotlivých projektů z první solution budou v příslušných kapitolách. Situace je vizualizována na obrázku 2.1, kde máme tuto solution vizuálně znázorněnou včetně závislostí jí příslušejících projektů.

Obrázek 2.1: Struktura aplikace.

Při naší práci jsme se snažili dodržovat určitá paradigmata a styl kódu, která jsou uvedena v příloze B.

Kompletní kód obou dvou solution včetně veškerých podkladových materiálů nalezneme v příloze A.

3. Implementace aplikace – databázová část

Abychom mohli v naší aplikaci pracovat s pacienty a měřeními, vytvářet neuronové sítě, učit je a testovat, potřebujeme nejprve databázi, kde si budeme potřebná data ukládat.

Hlavním projektem obsahujícím vlastní databázi, na které se převážně zaměříme, je `StabilometricDataAnalyzerAndVisualizer.Data`. K němu se pojí pomocný projekt `StabilometricDataAnalyzerAndVisualizer.DataTests` s jednotkovými testy.

3.1 Entity

První entitou, kterou musíme zahrnout, je pacient. U něj nás budou zajímat převážně identifikační údaje. Jako povinné, pro jednoznačné určení pacienta, máme databází generované ID, které slouží jako primární klíč. Dále se jedná o jméno a příjmení pacienta, spolu s datem jeho narození a pohlavím. Výběr těchto údajů byl založen na znalosti faktu, že lékaři se často snaží množství zapsaných informací minimalizovat, což by mohlo při dlouhodobém používání aplikace vést k nepořádku a duplicitě dat. Lékař by například mohl zapomenout, jak si kterého ze dvou pacientů se stejným příjmením označil a poté je vzájemně zamění, čemuž se snažíme touto cestou alespoň částečně předejít. U věku je povinnost dána tím, jak vyplývá ze shrnutí v podkapitole 7.1.4, že výsledky měření na posturografické plošině jsou ovlivněny i stářím pacienta. Pokud bychom chtěli tento parametr v budoucnu zahrnout jako vstup do NS, nebude již třeba jej zpětně dohledávat a doplňovat. U pohlaví platí totéž co u věku.

Jako volitelné údaje u entity pacienta jsme dali adresu a poznámky. Pokud u pacienta známe výše uvedené povinné údaje, lze si většinou adresu najít v centrální databázi pacientů dané nemocnice či oddělení. Tímto se snažíme pouze v případě zájmu vyjít lékaři vstříc a udržovat veškerá data pohromadě. Význam poznámek netřeba zmiňovat.

Posledním povinným údajem je klasifikace daného pacienta, kterou poté budeme hojně využívat při tvorbě a učení NS. Více o jednotlivých klasifikačních třídách se můžeme dozvědět v podkapitole 4.2.

Druhou entitou je měření. Její položky si můžeme rozdělit na dvě části – ty, jež souvisí s daty z vlastního měření, a identifikační údaje. Nejprve se zmíníme o položkách z druhé části. Zde spadají databázi generované ID, které slouží jako primární klíč, spolu s datem a časem měření, umožňujícím sledovat pacientův vývoj po případné terapii apod. Položky související s daty z měření zde podrobně popisovat nebudeme. Jedná se o výstup z SPS a námi vypočítané parametry. Pro bližší popis vizme kapitolu 1.

Třetí entitou je klasifikátor. V kódu je tato entita pojmenována jako analyzér, protože se obecně nemusí jednat pouze o klasifikátory, ale i prediktory či klastry. Pro rozsah a účely této práce však vždy půjde o klasifikátor. Položkami jsou databázi generované ID, sloužící jako primární klíč, dále povinný typ klasifikátoru (NS, Deep learning NS...) včetně jména, které z důvodu jejich ukládání na disk

vyžadujeme unikátní. Tento požadavek je dán tím, že v prvotní fázi budeme často tyto klasifikátory vytvářet a učit my, a následně bude nutné je přenášet mezi jednotlivými stroji, na kterých naše aplikace poběží. Zbývají nám už jen povinná cesta k jeho umístění na disku a informace, zdali je již naučený. Volitelně je možno přidat popis.

Od obecného klasifikátoru máme poté děděný NS klasifikátor. Jeho položky přímo vyplývají z popisu NS v podkapitole 4.1.2.

Předposlední entitou je funkce. Položkami jsou databázi generované ID, sloužící jako primární klíč. Dále povinný typ funkce (NS funkce, Deep learning NS funkce...) včetně jména. To vyžadujeme unikátní především pro případnou budoucí možnost ukládání na disk, ale i z důvodu jiného typu zobrazování klasifikátorů, funkcí a učitelů v GUI. Dále se jedná o uživatelský a matematický popis funkce. V matematickém je zkopírována informace o dané funkci z frameworku, který pro tento účel využíváme, vizme dále.

Stejně jako u klasifikátoru, tak i u funkce máme děděnou entitu NS funkce. Ta obsahuje položky pro parametry alfa a rozsah hodnot pro jednotlivé funkce, včetně možných kombinací těchto parametrů pro určení, které se mají při jejich tvorbě použít. S tím se pojí i položka o použitých parametrech při tvorbě funkce.

Od této entity dědí již finální entity pro vlastní funkce. V těch je vše z výše uvedené entity s případně specifikovanými informacemi.

Poslední entitou je učitel. U něj jsou položky shodné s entitou funkce. I od této entity se dědí další pro specifické učitele.

3.2 Struktura databáze

Výše uvedené entity nám spolu tvoří objektově-relační databázi s následujícími obecnými vztahy, které neřeší specifické dědičnosti:

- Pacient 1 ... 1..n Měření
- Pacient 1..m ... 1..n Analyzér
- Pacient 1..m ... 1..n Učitel
- Měření 1..m ... 1..n Analyzér
- Měření 1..m ... 1..n Učitel
- Analyzér 1..n ... 1 Učitel
- Analyzér 1..m ... 1..n Funkce

Pro lepší představu vizme vztahové diagramy mezi třídami a kódové mapy v příloze C.

Ve vlastním kódu aplikace se navíc vyskytuje ještě entita publikace. Vizme dále.

3.3 Použité knihovny

Jedná se o knihovny specifické pro databázovou část aplikace. Knihovny a nástroje, které se týkají všech projektů, včetně těch z databázové části, nalezneme v podkapitole 2.3. Všechny jsou v aktuální verzi k datu odevzdání této bakalářské práce. Výjimkou je Entity Framework, který používáme ve verzi 6, nikoliv 7, s nejnovějšími aktualizacemi. Důvodem je, že verze 7 již nepodporuje aplikace psané pro .NET Framework 4.0.

AForge.NET Framework [17] Jedná se o framework speciálně určený pro strojové učení, počítačové vidění a další, včetně s tím spojených matematických knihoven s potřebnými metodami. V databázové části jej využíváme pouze pro položku rozsah u entity funkce. S tou pracují třídy a metody z klasifikační části aplikace.

Entity Framework [18] Specializovaný framework pro práci a tvorbu objektově-relačních databází.

log4net [19] Knihovna, kterou používáme pro logování některých událostí v databázi. Umožňuje pokročilé nastavení logování včetně možnosti rotačního logování a zasílání logů pomoci mailu.

3.3.1 Alternativy

Pro první jmenovaný framework příliš volně dostupných alternativ není. Pokud bychom požadovali i stejnou míru funkcionality, s největší pravděpodobností žádný další nenalezneme. Výjimkou může být například Accord.NET Framework. Ten ale z +AForge.NET Frameworku [17] vychází a požaduje jej jako prerekvizitu.

U Entity Frameworku máme mnoho možných alternativ. Příkladem může být NHibernate, který je taktéž volně dostupný, má velkou komunitu uživatelů, a vychází z projektu Hibernate pro programovací jazyk JAVA.

To samé co u Entity Frameworku [18] platí i pro log4net [19]. Vše záleží jen na preferencích vývojářů. Pro snadnější rozvoj bych ovšem doporučoval zůstat u výše uvedených nástrojů z důvodů velké časové náročnosti pro přepis veškeré funkcionality.

3.4 Tvorba databáze

Entity Framework nám pro tvorbu databáze nabízí celkem čtyři možnosti:

Code-First do existující databáze Jde o postup, při kterém již máme k dispozici databázi a pomocí reverzního inženýrství vytvoříme kód pro objektové mapování do této databáze. Tu můžeme dále upravovat a migrovat na nové verze se zachováním zpětné kompatibility.

Code-First do nové databáze Jedná se o postup, při kterém vytváříme úplně novou databázi čistě pomocí tvorby kódu s jednotlivými entitami apod. To nám umožňuje vysokou kontrolu nad výstupem včetně všech omezení, triggerů...

Design-First do existující databáze Další z postupů, při němž máme k dispozici databázi, ale reverzním inženýrstvím vytvoříme „UML“ model, který posléze přímo upravujeme.

Design-First do nové databáze Poslední nabízenou možností, při níž vytváříme databázi od začátku formou „UML“ modelu, do nějž zanášíme všechny potřebné údaje. Věci jako triggery je nutno napsat manuálně.

Z výše nabízených možností jsme si vybrali možnost druhou – Code-First do nové databáze. Lze při ní snadno ovlivnit nejvíce věci a pro potřeby složitějších konstrukcí je nenahraditelná. Příkladem špatně řešitelného problému v Design-First návrhu je například implementace kolekce držící skalární typy, tvářící se jako jediný sloupec v databázi, vizme dále.

3.4.1 Možné problémy

Při tvorbě databáze jsme se setkali s několika následně uvedenými problematickými body. Ty si ve stručnosti popíšeme, abychom se jim mohli příště vyvarovat.

Prvním z nich je chyba „The Entity Framework provider type 'System.Data .Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer' for the 'System.Data.SqlClient' ADO.NET provider could not be loaded. Make sure the provider assembly is available to the running application. See http://go.microsoft.com /fwlink/?LinkId=260882 for more information“. S tou jsme se setkávali při prvních pokusech o jednotkové testování přístupu k datům v databázi. Řešením je vytvoření statického konstruktoru s následujícím kódem:

```
static StabilometricDataContext()
{
    // ReSharper disable UnusedVariable
    var _ = typeof(
        System.Data.Entity.SqlServer.SqlProviderServices);
    var __ = System
        .Data.Entity.SqlServer.SqlProviderServices.Instance;
    // ReSharper restore UnusedVariable
}
```

Ukázka kódu 3.1: Statický konstruktor databázového kontextu.

Druhým problematickým bodem bylo ukládání seznamu skalárních hodnot do jednoho sloupce v databázi. Entity Framework tuto možnost defaultně nenabízí a je nutno si ji doprogramovat. Pro naše potřeby jsme tedy doprogramovali abstraktní generickou třídu `PersistableScalarCollection<T>`. Následuje krátká ukázka kódu této třídy:

```
/// <summary>
/// Trida, ktera umoznuje udrzovani zakladnich typu v
   ↪ kolekci (neni podporovano v EF 6.1.3 a~nizsi).
/// </summary>
/// <typeparam name="T">Typ prvku, ktery bude v kolekci
   ↪ prechovavan.</typeparam>
[ComplexType]
```

```
public abstract class PersistableScalarCollection<T> :
   ↪ ICollection<T>
{
        private const string DefaultValueSeperator = "|";
        private readonly string[] defaultValueSeperators =
           ↪ { DefaultValueSeperator };

        // ReSharper disable once
           ↪ PublicConstructorInAbstractClass
        /// <summary>
        /// Vychozi konstruktor pro inicializaci nove
           ↪ prazdne kolekce.
        /// </summary>
        public PersistableScalarCollection()
        {
                Data = new List<T>();
        }

...
```

Ukázka kódu 3.2: Abstraktní třída pro kolekci skalárních dat.

Třetím problematickým bodem je nemožnost mít v databázi `[NotMapped]` property, tj. property, pro kterou se nevytváří nový sloupec, ke které je možno přistupovat. Řešením je `[NotMapped]` atribut nepoužívat a smířit se se sloupci v databázi navíc.

Posledním bodem, který zmíníme, je nastavení atributu `[Index(IsUnique = true)]`. Pokud property s tímto atributem současně nenastavíme i `[MaxLength(450)]`, dostaneme chybu „column in table is of a type that is invalid for use as a key column in an index“.

3.5 Možná rozšíření

V případě další práce na této části aplikace je možno rozšíření rozdělit na dvě části.

První částí jsou rozšíření zaměřující se na údaje uložené v databázi. Toto se pojí převážně s druhou — klasifikační — částí práce, kdy se na jejím základě budou doplňovat potřebné entity. Případně, pokud bude 2. LF UK investovat peníze do rozvoje a inovace měřících přístrojů, připadá v úvahu i možnost rozšíření o naměřená data z jiných posturografických plošin. Posledním příkladem je i možnost rozšíření aplikace o export výsledků v rámci různých šablon. K tomuto účelu je již v konceptu nachystána základní kostra entity publikace.

Druhou částí jsou možné úpravy typu databáze a jejího umístění. V případě rozšíření aplikace mezi více pracovišť by se mohlo hodit ve sdíleném výzkumu pracovat na databázi umístěné v „cloudu“, kdy by se dalo k databázi přistupovat odkukoliv a měli bychom výhodu většího množství učících a testovacích dat.

3.6 Testovací data

Pro naše potřeby jsou v kódu programu zanesena i testovací data, na kterých náš koncept aplikace zkoušíme. Kvůli udržení lékařského tajemství jsou všechny údaje, kromě vlastních naměřených dat, náhodně vygenerovány, aby nebylo možno přiřadit jednotlivá data k reálným pacientům. Pokud by se tímto generováním povedlo vytvořit údaje o skutečné existující osobě, pak je tato spojitost čistě náhodná.

3.7 Vkládání měření do databáze

Jelikož množství dat souvisejících s jedním měřením je příliš mnoho, vždy budeme měření do databáze vkládat pomocí automatického načtení a parsování souborů z adresáře s měřením pro daného pacienta. Z toho důvodu je součástí projektu `StabilometricDataAnylyzerAndVisualizer.Data` i třída `MeasurementsDataParser` sloužící pro tento účel. Abychom usnadnili práci co nejvíce a nebylo nutno po načtení dat ručně pouštět výpočet našich parametrů, děje se tak automaticky. K tomu máme třídu `MeasurementParametersCalculator`.

Řídíme se postupem TDD. Proto jsou obě výše uvedené třídy testovány v rámci jednotkových testů projektu `Stabilometric DataAnylyzerAndVisualizer.Data` ↪ `Tests`.

3.7.1 Možné problémy

Pro parsování a načítání dat z měření jsme chtěli využít faktu, že jsou soubory ve formátu .dbf. Příslušejí nějaké databázi, mají hlavičku, a bude možno je načíst elegantněji pomocí `OleDbConnection` do `DataSetu`. Bohužel hlavička souborům chybí, a proto musela být tato možnost zamítnuta.

Výše uvedený problém také brání v tom, abychom mohli využít zálohu databáze z SPS pro hromadné načtení. Při zálohování databáze z SPS navíc dochází k jejímu šifrování. To nám brání i v možnosti vytáhnout alespoň data pro jednotlivá měření, bez nutnosti ručního výběru a exportu dat, která trvá příliš dlouho a znesnadňuje práci.

Možným řešením a urychlením by mohl být skript. Na tom se v době odevzdání této práce pracuje. Problém je ovšem v mísení dat z měření od různých lékařů, která potřebujeme filtrovat apod.

3.8 Přístup k datům v databázi

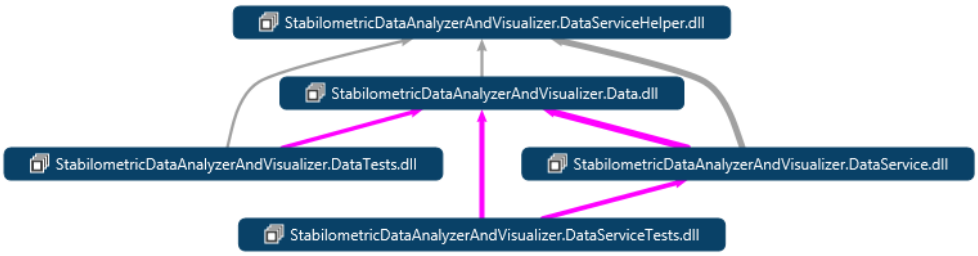
Posledními projekty, které spadají do databázové části aplikace jsou `Stabilometric` ↪ `DataAnylyzerAndVisualizer.DataService`, spolu s `StabilometricDataAnylyzer` ↪ `AndVisualizer.DataServiceHelper` a `StabilometricDataAnylyzerAndVisuali` ↪ `zer.DataServiceTests`.

První jmenovaný je Windows Communication Foundation projekt, dále jen WCF, pro komunikaci s databází, zpřístupňující data pomocí příslušných metod. Tento přístup nám umožňuje v dalších částech programu opomenout vlastní

implementaci databáze. Dále také můžeme nastavit omezení, jak a k jakým datům se dostaneme. Popis vlastní implementace přeskočíme. Zdrojové kódy jsou přístupné v příloze A.

Druhý projekt slouží pouze pro zjednodušenní některých operací při přístupu do databáze, vizme vlastní kód. Třetí projekt již opět obsahuje jen jednotkové testy pro kontrolu funkčnosti dané části aplikace.

Strukturu celé databázové části si můžeme prohlédnout na obrázku 3.1.

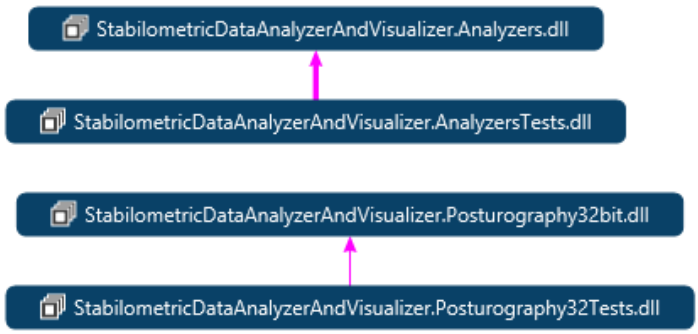


Obrázek 3.1: Struktura databázové části aplikace.

4. Implementace aplikace – klasifikační část

Všechny následující informace vycházejí z předešlé kapitoly. Popisy jednotlivých přenosových funkcí, učitelů pro NS apod. se nebudeme zabývat. V případě zájmu je možno nahlédnout do vlastního zdrojového kódu v příloze A.

Projekty související s klasifikační částí jsou `StabilometricDataAnalyzerAnd` ↪ `Visualizer.Analyzers`, v němž nalezneme metody pro tvorbu NS, jejich přechodových funkcí a učitelů. Dále jsou to jen projekty `StabilometricDataAnalyzer` ↪ `AndVisualizer.Posturography32bit` a `StabilometricDataAnalyzerAndVisua` ↪ `lizer.Posturography64bit`, které obsahují vlastní implementaci výpočtu jednotlivých parametrů, vizme podkapitolu 1.3.2. K výše uvedeným třem projektům připadají `StabilometricDataAnalyzerAndVisualizer.AnalyzersTests`, `Stabilo` ↪ `metricDataAnalyzerAndVisualizer.Posturography32Tests` a `Stabilometric` ↪ `DataAnalyzerAndVisualizer.Posturography64Tests` s jednotkovými testy. Jejich strukturu si můžeme prohlédnout na obrázku 4.1.



Obrázek 4.1: Struktura klasifikační části aplikace.

S klasifikační částí ještě přímo souvisí projekty `PosturographyMeasurements` ↪ `Utilities.ANNPatientsClassifier` a `PosturographyMeasurementsUtilities` ↪ `.ANNPatientsClassifiersCreator`. Ty umožňují tvorbu NS i jejich použití v experimentálním nasazení, více vizme kapitolu 6.

Abychom mohli pacienty klasifikovat, musíme si nejprve vybrat vhodnou metodu, se kterou budeme pracovat.

4.1 Použitá klasifikační metoda

4.1.1 Výběr metody

Při výběru jsme vycházeli z knihy Hastieho, Tibshiraniho a Friedmana The Elements of Statistical Learning [20].

Strojové učení nabízí rozsáhlé možnosti algoritmů, které lze použít. Je možné si je rozdělit podle typu úloh, které budou provádět na:

Klasifikační Rozdělují vstupní data do dvou či více tříd (je mail spam či nikoliv).

Regresní Odhadují výstupní hodnotu na základě vstupů (cena domu dle počtu pokojů, velikosti v m^2...).

Klastrovací Jsou schopné zařadit vstupní objekty do skupin s podobnými vlastnostmi, typicky bez znalosti jejich opravdového zařazení (třídění rostlin do tříd na základě jejich genomu).

Pro práci lze vyřadit algoritmy regresní a klastrovací a zaměřit se pouze na klasifikační. Ty jde rozdělit dle způsobu učení na:

Učení s učitelem Zde předem známe pro jednotlivé vstupy jejich výstupní třídy.

Učení bez učitele Zde výstupní třídy neznáme.

Kombinované učení Klasifikační třídy pro část vstupu (většinou menší část) známe a pro část nikoliv.

Zpětnovazebné učení

Z těchto čtyř typů nám nejvíce vyhovuje první – učení s učitelem.

Nyní jsme vybrali určitou množinu algoritmů, které vyhovují typu naší úlohy, kterou potřebujeme zredukovat pouze na jediný vybraný algoritmus. Při výběru jsme se rozhodovali mezi těmito čtyřmi – lineární klasifikace, support vector machines (hledání nadroviny optimálně rozdělující vstupní data), dále jen SVM, NS, a rozhodovacími stromy.

Lineární klasifikaci jsme vyřadili z důvodu, že množina vstupních dat je rozdělena do jednotlivých tříd přímkami, tj. nelze oddělit dvě třídy, kdy jedna tvoří v dvourozměrném zobrazení kruh uprostřed druhé apod. Toto negativum lze kompenzovat například umocňováním parametrů, jejich vzájemným násobením a dalšími matematickými úpravami. Vzhledem k tomu, že jich máme celkem 136 (vizme podkapitolu 1.3.2), bylo by hledání optimálních parametrů pro klasifikaci neúměrně časově náročné vzhledem k počtu jejich možných kombinací.

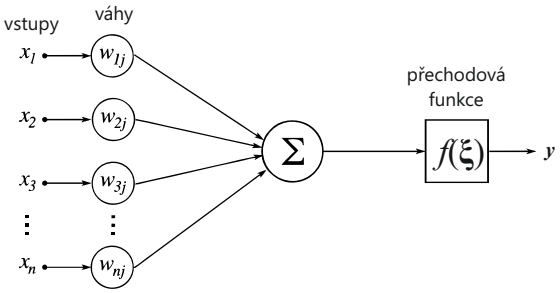
Rozhodovací stromy byly vyřazeny z důvodu autorových malých znalostí těchto algoritmů.

Mezi SVM a NS bylo rozhodnuto pro NS. A to na základě toho, že NS jsou založeny na biologickém podkladu fungování mozku. Navíc v oblasti NS probíhá dynamický výzkum s prezentací vysoce kvalitních výsledků, například Google DeepMind [21] založený na NS s možností přístupu k externí paměti jako konvenční Turingův stroj [22] či výzkum Tomáše Straky. Dále je to z odborných článků vyplývající časté využití NS v medicinském výzkumu, vizme diplomovou práci Lucie Görlichové [23], dále jen Görlichová. Posledním důvodem pro výběr NS byl právě Krafczyk [9].

4.1.2 Popis metody

Abychom mohli dále popisovat použité NS, musíme nadefinovat potřebné pojmy. Při jejich vymezení a použití v následujícím textu vycházíme z knihy Neural networks: a systematic introduction [24], dále jen Rojas, a z pdf slidů k přednášce Neuronové sítě doc. RNDr. Ivety Mrázové, CSc. [25], dále jen Mrázová.

Definice. *Neuron (perceptron) s vahami* $(w_1, \ldots, w_n) \in \mathcal{R}^n$*, prahem* $\vartheta \in \mathcal{R}$ *a přenosovou funkcí* $f : \mathcal{R}^{n+1} \times \mathcal{R}^n \to \mathcal{R}$ *počítá pro libovolný vstup* $\overrightarrow{z} \in \mathcal{R}$ *svůj výstup* $y \in \mathcal{R}$ *jako hodnotu přenosové (aktivační) funkce v* $\overrightarrow{z}$*,* $f[\overrightarrow{w}\vartheta](\overrightarrow{z})$*, kde* $\mathcal{R}$ *označuje množinu reálných čísel.*



Obrázek 4.2: Model umělého neuronu.

Definice. *Nejčastěji používáné přenosové funkce:*

- *Skoková:*

$$f(y) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i + \vartheta \geq 0 \\ 0 & \text{if } \sum_{i=1}^{n} w_i x_i + \vartheta < 0 \end{cases}$$

- *Signum:*

$$f(y) = \begin{cases} \text{sign}(x) = 1 & \text{if } \sum_{i=1}^{n} w_i x_i + \vartheta > 0 \\ \text{sign}(x) = 0 & \text{if } \sum_{i=1}^{n} w_i x_i + \vartheta = 0 \\ \text{sign}(x) = -1 & \text{if } \sum_{i=1}^{n} w_i x_i + \vartheta < 0 \end{cases}$$

- *Lineární:*

$$f(y) = y$$

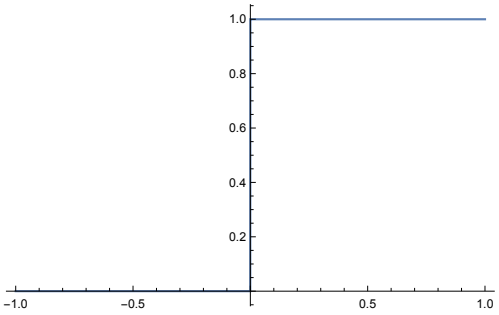
- *Sigmoidální:*

$$y = f[\overrightarrow{w}\vartheta](\overrightarrow{z}) = f(\xi) = \frac{1}{1 + e^{-\xi}}$$
$$\xi = \sum_{i=1}^{n} w_i x_i + \vartheta$$

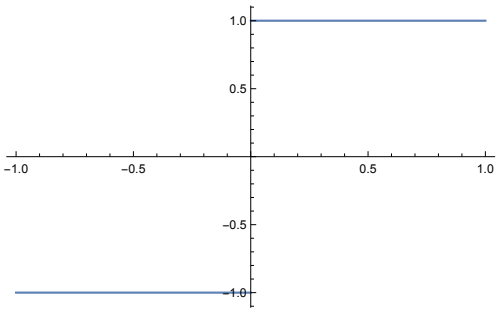
- *Hyperbolický tangent:*

$$y = f[\overrightarrow{w}\vartheta](\overrightarrow{z}) = f(\xi) = \frac{e^{\xi} - e^{-\xi}}{e^{\xi} + e^{-\xi}}$$
$$\xi = \sum_{i=1}^{n} w_i x_i + \vartheta$$

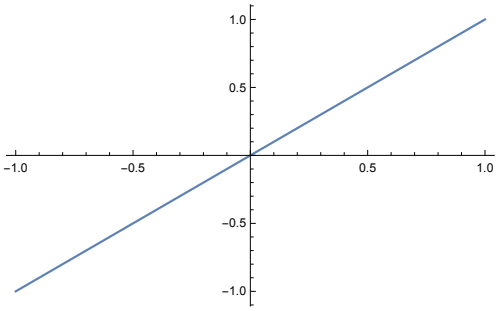
, kde ξ označuje tzv. potenciál neuronu.



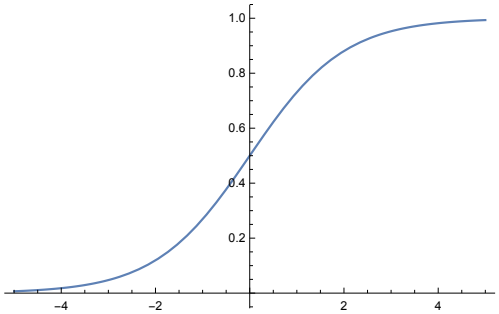
(a) Skoková.



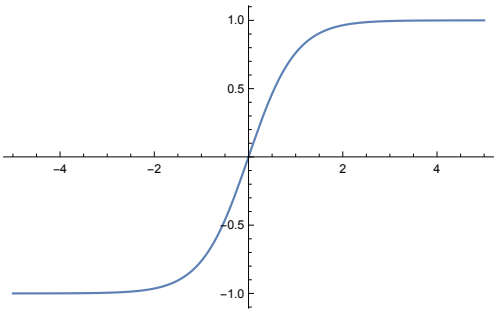
(b) Signum.



(c) Lineární.



(d) Sigmoidální.



(e) Hyperbolický tangent.

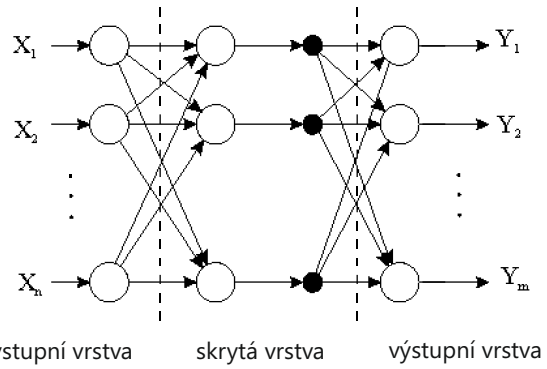
Obrázek 4.3: Přenosové funkce.

Pro další definice a popisy budeme vždy uvažovat přenosovou funkci za hyperbolický tangent.

Definice. *Nechť $\overrightarrow{z}$ označuje vstup neuronu.*

- *Jestliže $f[\overrightarrow{w}\vartheta](\overrightarrow{z}) = 1$, říkáme, že je neuron aktivní.*
- *Jestliže $f[\overrightarrow{w}\vartheta](\overrightarrow{z}) = \frac{1}{2}$, říkáme, že je neuron tichý;*
  Tato skutečnost znamená, že příslušný vstup leží v dělící nadrovině určené tímto neuronem.
- *Jestliže $f[\overrightarrow{w}\vartheta](\overrightarrow{z}) = -1$, říkáme, že je neuron pasivní.*

Definice. *Vícevrstvá NS je taková NS, kde jsou neurony sdruženy do vrstev. Výstup každého neuronu v n-té vrstvě je přiveden na vstup každého neuronu v $n+1$ vrstvě. První vrstvu nazýváme vstupní, poslední vrstvu výstupní. Vrstvy mezi vstupní a výstupní vrstvou se nazývají skryté.*



Obrázek 4.4: Model vícevrstvé umělé neuronové sítě.

Definice. *Pro NS B s n vstupními vektory a m výstupními neurony:*

- *Vstupní vzor označuje vstupní vektor $\overrightarrow{x} \in \mathcal{R}^n$ zpracovávaný sítí.*
- *Požadovaný výstup $\overrightarrow{d} = (d_1, \ldots, d_m)$ tvoří požadované výstupy neuronů výstupní vrstvy.*
- *Pro daný vstupní vzor představuje skutečný výstup B vektor $\overrightarrow{y} = (y_1, \ldots, y_m)$ tvořený skutečnými výstupy neuronů výstupní vrstvy.*

Definice. *Trénovací množina T je množina p uspořádaných dvojic tvaru vstupní vzor/požadovaný výstup:*

$$T = \{[\overrightarrow{x}_1, \overrightarrow{d}_1], \ldots, [\overrightarrow{x}_p, \overrightarrow{d}_p]\}$$

V obecném případě může být přenosová funkce u každého neuronu v NS různá. Pro dále uváděné použité NS ovšem uvažujeme pouze jednu přenosovou funkci pro všechny neurony v této NS.

NS jsou v naší práci učeny převážně algoritmem zpětné propagace chyby.

4.2 Klasifikační třídy

V naší práci jsme analyzovali posturografický signál získaný při vyšetření stoje na posturografické plošině SPS ve čtyřech senzorických situacích dle testu mCTSIB, vizme kapitolu 1. Vyšetření pacientů bylo provedeno v neurootologické laboratoři Neurologické kliniky 2. LF UK a Fakultní nemocnice v Motole.

Pacienti patřili do následujících klasifikačních tříd:

Zdraví pacienti „Skupina zdravých jedinců byla tvořena dobrovolníky z řad studentů 2. LF UK. U všech jedinců byla anamnesticky vyloučena onemocnění, která mohou ovlivňovat stabilitu stoje.“ [26]

Pacienti s vestibulární poruchou „Jedinci s poruchou vestibulární funkce byli tvořeni homogenní skupinou pacientů v raném pooperačním období po neurochirurgickém odstranění vestibulárního schwannomu[1]. U všech pacientů byl v době vyšetření klinicky přítomen jednostranný periferní vestibulární syndrom.“ [26]

[1]Nádor vycházející z buněk Schwannovy pochvy. [27]

Pacienti s poruchou mozečku „Jedinci s poruchou funkce mozečku byly pacienti s degenerativním onemocněním z okruhu spinocerebelárních ataxií[2]. V době vyšetření byla u všech pacientů přítomná mozečková symptomatika (ataxie stoje a chůze).“ [26]

Jak už bylo zmíněno dříve, všechny informace o pacientech, kromě dat z měření, jsou smyšlené. Jakákoliv podobnost s reálnou osobou je čistě náhodná. Všichni vyšetření jedinci dali souhlas s účastí ve studii.

4.3 Použité knihovny a nástroje

Jedná se o knihovny a nástroje specifické pro klasifikační část aplikace. V podkapitole 2.3 najdeme části týkající se projektů a klasifikační části. Všechny jsou v aktuální verzi k datu odevzdání této bakalářské práce.

Accord.NET Framework [29] Jde o framework specializovaný na strojové učení, počítačové vidění, zpracování obrazu, signálů a další. V klasifikační části jej využíváme jak pro tvorbu a učení NS, tak i pro některé matematické operace využité ve výpočtech parametrů.

Math.NET Numerics [30] Framework, který se specializuje na metody a algoritmy pro numerické výpočty v různých vědních odvětvích. Pro naše potřeby se bude jednat především o aplikace těchto metod v lineární algebře.

MATLAB [31] IDE a vysokoúrovňový programovací jazyk s rozsáhlými možnostmi v oblasti všemožných výpočtů. V našem konceptu aplikace jsme jej využili pro export skriptů Ing. Tomáše Fundy do dll knihoven kompilovaných pro .NET 4.0.

4.3.1 Alternativy

Pro první jmenovaný framework příliš volně dostupných alternativ není. Vizme podkapitolu 3.3.1. To samé platí i pro druhý uvedený framework.

U MATLABu [31] máme k dispozici volně dostupnou náhradu ve formě programu Octave. Ten je ovšem pro naše účely nevhodný, neboť v základu obsahuje pouze omezenou množinu MATLABovských toolboxů. Část je možno doinstalovat v rámci balíku Octave-Forge. V tom je obsažen i námi vyžadovaný Signal processing toolbox. Bohužel spousta funkcí je zatím bez vlastní implementace.

4.4 Implementační problémy

Při implementaci klasifikační části nastal pouze jeden závažný problém. Tím bylo převedení výpočtu parametrů přímo do C# kódu. K tomu nás vedla potřeba, že na stroji, kde bude aplikace nasazena, nebude nainstalován MATLAB [31] a jeho dokoupení není realizovatelné v rámci našich peněžních možností.

[2]Neurodegenerativní onemocnění. Pro bližší popis vizme například stránky Medicabaze.cz [28].

Parametry, které ve výpočtu využívají funkcí ze Signal processing toolboxu, nešlo přímo přepsat. Námi využívané frameworky (ani jiné volně dostupné) neobsahují potřebné metody. Bylo možné využít pouze implementaci pro jiné programovací jazyky. Při pokusech o kompilaci cizích kódů však docházelo k vážným chybám, které ji znemožňovaly.

Nakonec jsme tedy skončili u přímého exportu funkcí z MATLABU [31] do dll knihoven. Fakultní licence MATLABU [31] ovšem toto rozšíření neobsahuje. Zde patří díky Ing. Václavovi Dedkovi, který nám tuto možnost zprostředkoval a skripty pro nás zkompiloval.

Kód z problematického skriptu si můžeme prohlédnout na následující ukázce:

```
function [POWER, F50, F95, Fpeak, Fmean, Fcentr, Fdisp] =
    ↪ cpower (data, fs, tapers)
% [POWER, F50, F95, Fpeak, Fmean, Fcentr, Fdisp] = cpower
    ↪ (data, fs, tapers) calculates the total power, 50%
    ↪ and 90% of power frequency, peak frequency, mean
    ↪ frequency, centroidal
...

n = 2^nextpow2(size(data,1));
[p,f] = pmtm(data,tapers,n,fs);

[~,peak] = max(p);
area = cumtrapz(f,p);
F50 = find(area >= 0.50*area(end));
F95 = find(area >= 0.95*area(end));
u0 = trapz(f,p);
u1 = trapz(f,f.*p);
u2 = trapz(f,f.^2.*p);

...
```

Ukázka kódu 4.1: Problematický MATLAB skript pro výpočet POWER a F parametrů.

4.5 Možná rozšíření

V případě další práce na této části bude rozšíření spjato především s rozšiřováním databázové části o nové údaje použitelné při vlastní klasifikaci. Je také možné objevení nových zajímavých parametrů v odborném článku, který by bylo možné implementovat.

5. Implementace aplikace – GUI

Poslední nepopsanou částí našeho konceptu aplikace je GUI. V následujícím textu si pouze stručně popíšeme použité knihovny a nástroje, strukturu této části a problémy při implementaci. Vzhledem k aktuálnímu dynamickému vývoji nebudeme zacházet do hlubších detailů a pokusíme se shrnout pouze informace, které by se časem neměly výrazněji měnit.

S GUI části aplikace přímo souvisí pouze projekt `StabilometricDataAnalyzer ↪ AndVisualizer.UI`. Ten ale přímo či nepřímo využívá všech dosud popsaných částí aplikace. Strukturu aplikace si můžeme prohlédnout na obrázku 2.1.

5.1 Použité knihovny

Jedná se pouze o knihovny specifické pro GUI část aplikace. Ostatní použité knihovny a nástroje nalezneme v příslušných kapitolách.

Fluent Ribbon Control Suite [32] Jedná se o knihovnu implementující ribbon pro uživatelské rozhrání tvořené ve WPF. Dodává se spolu s Microsoft Office 2010, Microsoft Office 2013 a Windows 8 tématy pro dotvoření jednotného dojmu při používání na platformě Windows.

MVVM Light Toolkit [33] Jde o toolkit umožňující snadnou aplikaci MVVM patternu do naší aplikace. Pro jeho bližší popis vizme kapitolu B.

Xceed Extended WPF Toolkit Plus [34] Placený toolkit, který nám poskytuje spoustu nových či vylepšených prvků pro GUI. Využíváme z něj převážně prvek `DataGrid` a `ChildWindow`. V budoucnu je plánován přechod na nejvyšší placenou verzi pro zpřístupnění všech prvků a bez nutnosti každoroční placené aktualizace.

5.1.1 Alternativy

Pokud se zaměříme na knihovny s prvky pro GUI, bude hledání alternativ velmi složité. Příkladem mohou být tabulky. U nich vyžadujeme funkcionalitu pro seskupování prvků podle sloupců, řazení položek vzestupně / sestupně dle hodnoty a další. V budoucnu je plánováno i zařazení filtrování. Když budeme hledat knihovnu, která umí vše uvedené, narazíme pouze na ty placené. U těch, jež jsou zdarma, vždy něco chybí a je nutno danou funkčnost doplnit alternativně. To může být ovšem problém z důvodu nekompatibility jednotlivých prvků.

Toolkitů pro snadnou aplikaci MVVM patternu nalezneme více. Hlavním důvodem pro výběr MVVM Light Toolkitu [33] byla především zkušenost z praxe. Vizme zmínky v kapitole B.

5.2 Struktura projektu

Service References V této složce nalezneme servicy pro práci s daty v databázi. V aktuálním stádiu se jedná o servicy pro práci s pacienty, měřeními a možnost načítat a parsovat naměřená data.

Converters Zde nalezneme konvertory pro převody mezi různými typy dat využívané převážně v .xaml souborech pro převod `Enum` hodnot na lokalizované řetězce, invertování booleovských hodnot a další.

Interfaces Nachází se zde interfacy, převážně pro ulehčení komunikace mezi View a ViewModelem.

Localization Resources V tomto umístění se nachází veškeré zdroje pro lokalizaci GUI naší aplikace.

Messages V této části nalezneme třídy implementující zprávy pro komunikaci mezi různými View a ViewModely. Například zpráva se žádostí o otevření okna s editací pacienta po zmáčknutí příslušného tlačítka.

Utils Zde najdeme pomocné třídy a interfacy pro tabulky s pacienty, měřeními apod. Například třídu pro generickou asynchronní verzi `ObservableColle` ↪ `ction`.

View Models V tomto umístění se nachází View Modely pro jednotlivá okna GUI.

Views V této části nalezneme View pro jednotlivá okna GUI.

XAML Resources Poslední důležitá část, kde se nachází styly, templaty a další zdroje pro .xaml soubory.

5.3 Implementační problémy

Při implementaci GUI části aplikace jsme se pravidelně setkávali a stále setkáváme s různými implementačními problémy. Drtivá většina z nich se týká Xceed Extended WPF Toolkitu Plus [34].

V souhrnu jde převážně o problémy s přizpůsobováním velikosti okna, které v sobě obsahuje `DataGrid`. S ním se pojí i další problémy ohledně přizpůsobování jeho vzhledu a chování. Tyto problémy jsou aktuálně v řešení. Většinu je možno snadno vyřešit nasazením nejvyšší placene verze tohoto toolkitu. Zakoupení této verze je v plánu. Aktuálně závisíme na možnosti požádat v rámci výzkumu na 2. LF UK o grant na jeho zaplacení.

Jeden z již vyřešených problémů si můžeme prohlédnout v ukázkách kódu 5.1 a 5.2. Jednalo se zde o problém s bindováním lokalizovaných názvů sloupců v tabulce s pacienty.

```
...

<utils:SDAaVDataGridControl x:Name="Patients"
   ↪ ItemsSource="{Binding Patients}"
   ↪ Style="{StaticResource {x:Type
   ↪ utils:SDAaVDataGridControl}}" SelectedItem="{Binding
   ↪ Path=SelectedPatient}">
  <xcdg:DataGridControl.Columns>
```

```
    <xcdg:Column FieldName="LastName" AllowGroup="False"
       ↪ Visible="{Binding Source={x:Static
       ↪ settings:DatabaseSettings.Default},
       ↪ Path=LastNameColumnVisible}" />
    <xcdg:Column FieldName="FirstMiddleName"
       ↪ AllowGroup="False" Visible="{Binding
       ↪ Source={x:Static
       ↪ settings:DatabaseSettings.Default},
       ↪ Path=FirstMiddleNameColumnVisible}" />
    <xcdg:Column FieldName="FullName" AllowGroup="False"
       ↪ Visible="{Binding Source={x:Static
       ↪ settings:DatabaseSettings.Default},
       ↪ Path=FullNameColumnVisible}" />
    <xcdg:Column FieldName="DateOfBirth"
       ↪ AllowGroup="False" DisplayMemberBinding="{Binding
       ↪ Path=., Converter={StaticResource
       ↪ DateOfBirthConverter}}" Visible="{Binding
       ↪ Source={x:Static
       ↪ settings:DatabaseSettings.Default},
       ↪ Path=DateOfBirthColumnVisible}" />
    <xcdg:Column FieldName="Age" AllowGroup="True"
       ↪ Visible="{Binding Source={x:Static
       ↪ settings:DatabaseSettings.Default},
       ↪ Path=AgeColumnVisible}" />

...
```

Ukázka kódu 5.1: Ukázka XAML kódu s tabulkou pro zobrazení pacientů z databáze.

```
/// <summary>
/// Inicializuje bindingy pro View.
/// </summary>
private void initializeBindings()
{
    #region Column titles' bindings
    // ReSharper disable AccessToStaticMemberViaDerivedType
    Binding binding =
        new Binding("LastNameColumnName")
        {
            Source = viewModel,
            Mode = BindingMode.OneWay
        };
    BindingOperations.SetBinding(Patients.Columns[0],
       ↪ Column.TitleProperty, binding);

    binding =
        new Binding("FirstMiddleNameColumnName")
        {
            Source = viewModel,
            Mode = BindingMode.OneWay
```

```
        };
    BindingOperations.SetBinding(Patients.Columns[1],
       ↪ Column.TitleProperty, binding);

...
```

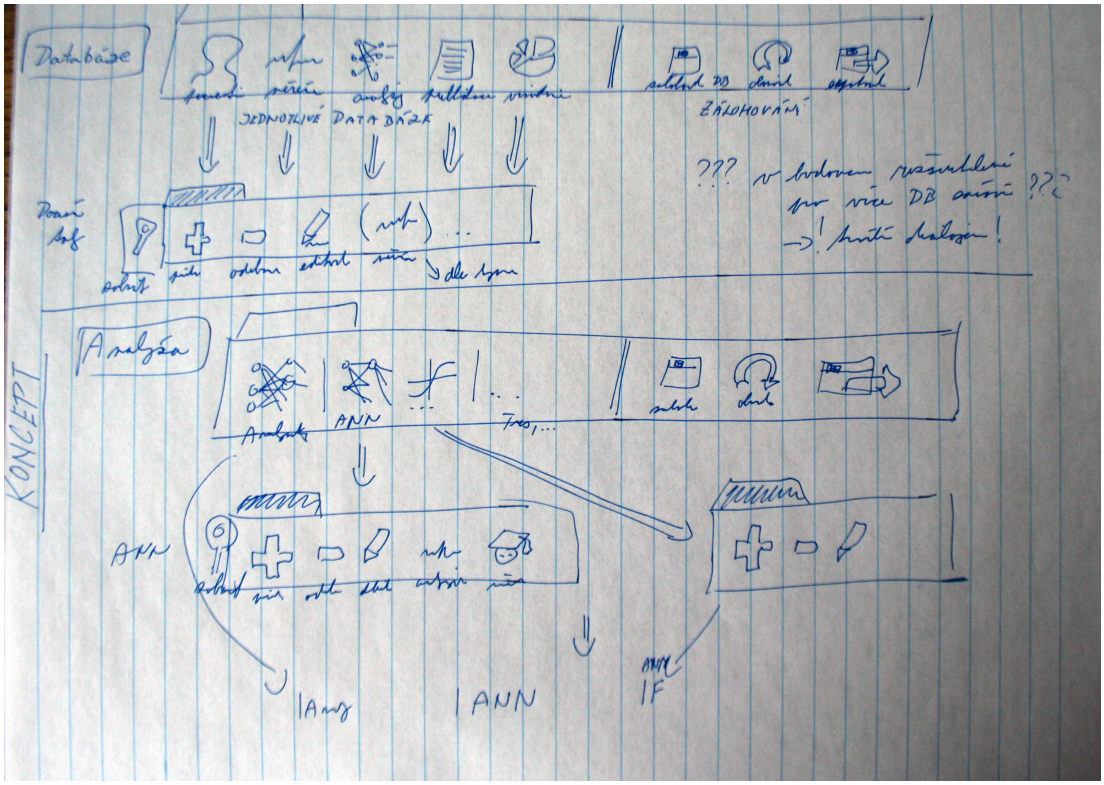
Ukázka kódu 5.2: Ukázka bindování názvů sloupců z code behind.

5.4 Další rozšiřování

Aktuálně máme v GUI část aplikace zprovozněnou tabulku s pacienty z databáze s možností zobrazování, přidávání, upravování a odebírání pacientů. Tabulka s měřeními z databáze je aktuálně ve fázi vývoje, kdy máme nachystanou základní kostru, ale probíhají diskuze nad prvky a možnostmi, které má tato tabulka nabízet. Tato diskuze souvisí i se snahou snížit množství přenášených údajů z databáze pomocí servicy při úvodním načítání.

Pro klasifikační část bude GUI dotvořeno až po vyřešení části s měřeními. Zatím slouží jako funkční náhrada využívaná v rámci experimentálního nasazení aplikace z projektu `PosturographyMeasurementsUtilities.ANNPatientsClassi` ↪ `fier`. Pro více informací vizme kapitoly 4 a 6.

Na obrázku 5.1 si můžeme prohlédnout jeden z pracovních konceptů GUI.



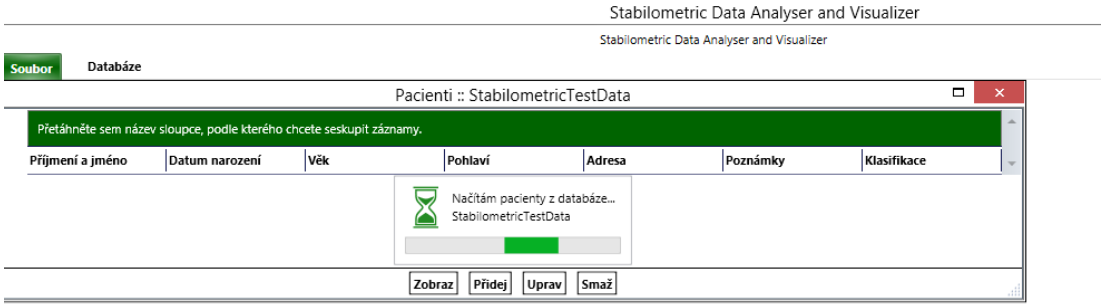
Obrázek 5.1: Koncept GUI naší aplikace.

6. Uživatelská dokumentace a experimentální nasazeni aplikace

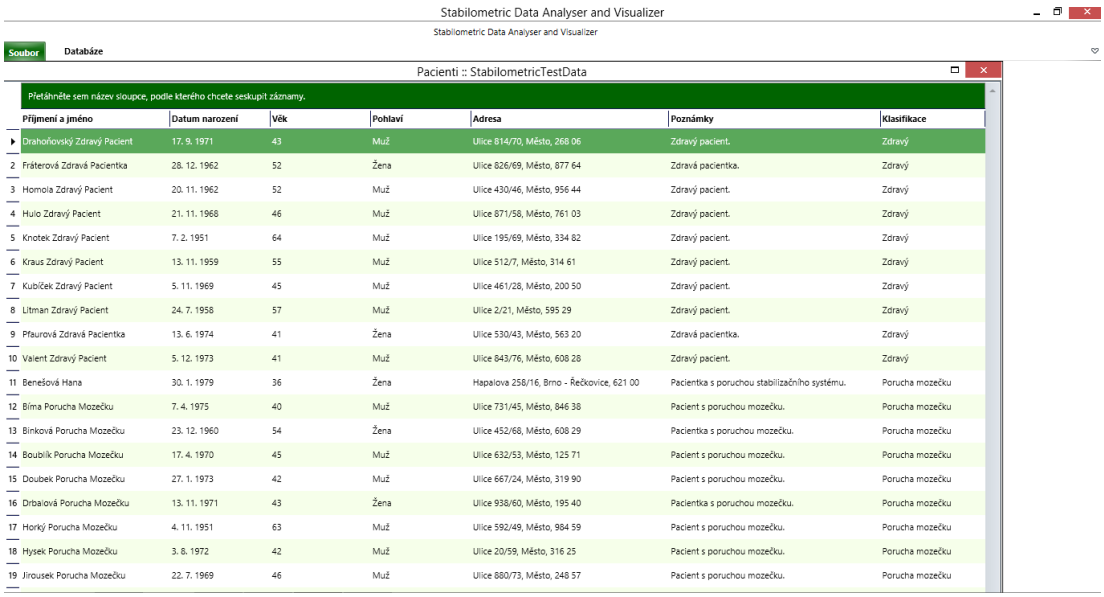
6.1 Uživatelská dokumentace

Vzhledem k intuitivnímu uživatelskému rozhraní bude uživatelská dokumentace zobrazena pouze formou obrázků se stručnými popisky. Pokud nastane problém s pochopením některého prvku z GUI, je možné shlédnout i ukázkové video, vizme podkapitolu 6.2.

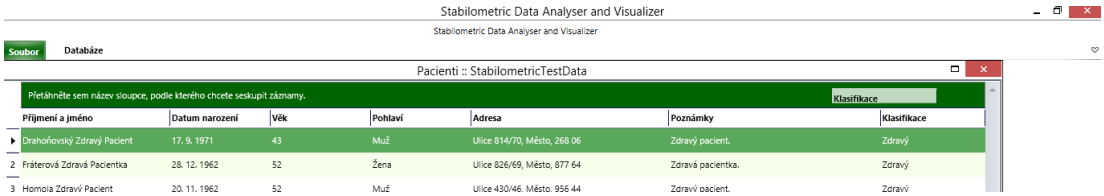
6.1.1 Stabilometric Data Analyzer and Visualizer



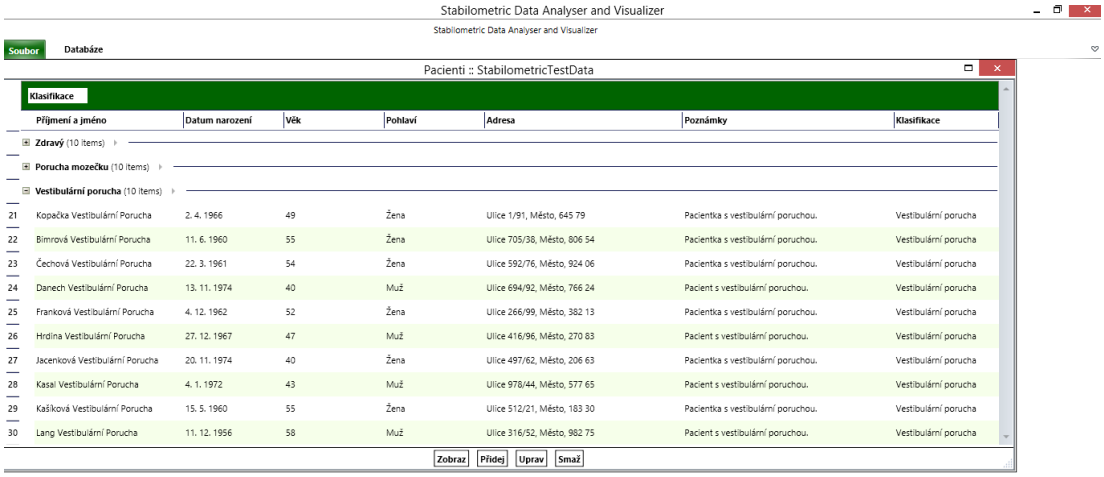
Obrázek 6.1: Spuštěný koncept aplikace při načítání pacientů z databáze.



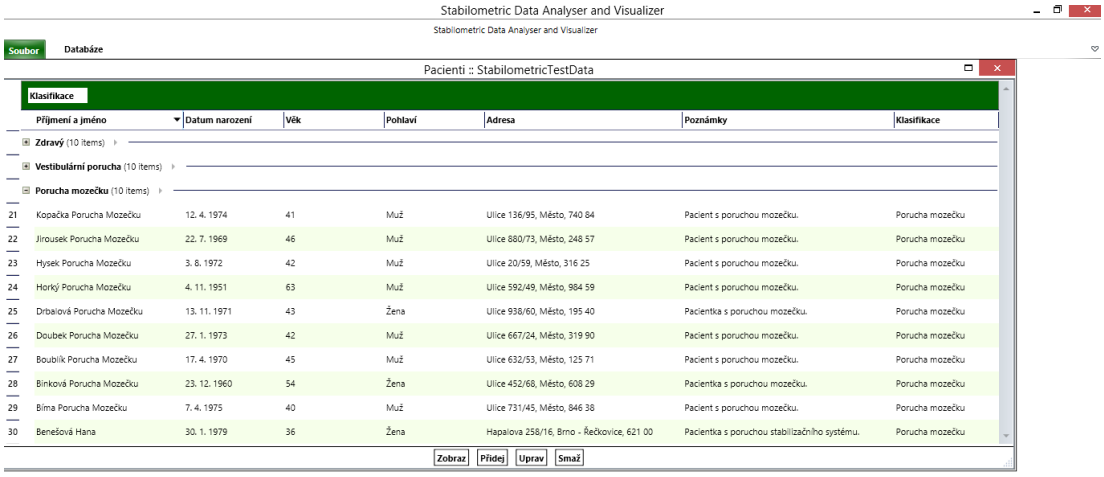
Obrázek 6.2: Spuštěný koncept aplikace s načtenými pacienty z databáze.



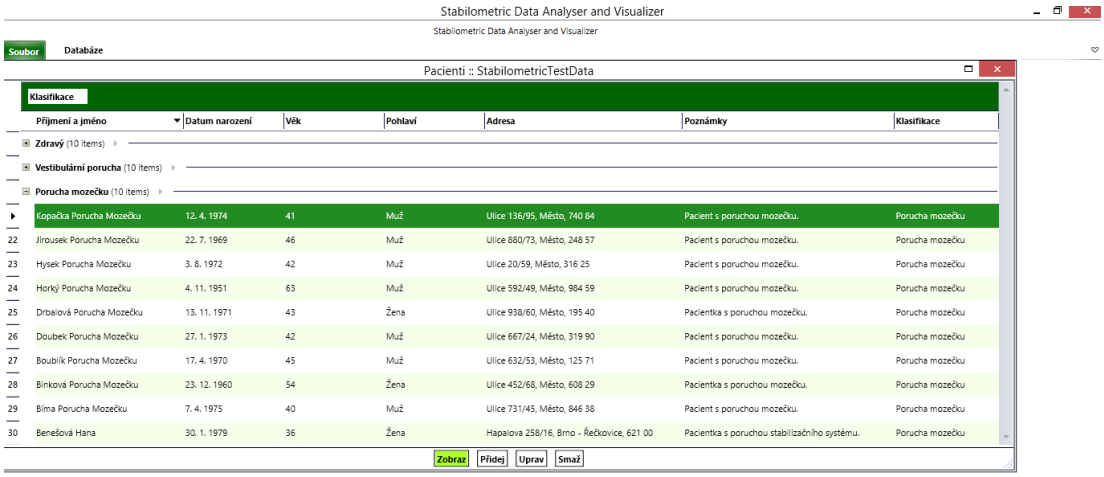
Obrázek 6.3: Seskupování pacientů v databázi dle hodnot ve sloupci.



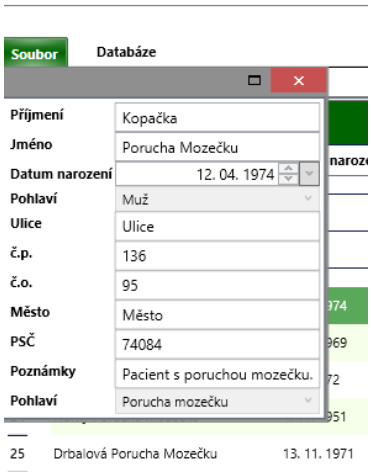
Obrázek 6.4: Skrývání nepotřebných skupin pacientů.



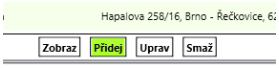
Obrázek 6.5: Úprava řazení pacientů v databázi sestupně dle sloupce „Příjmení a jméno“.



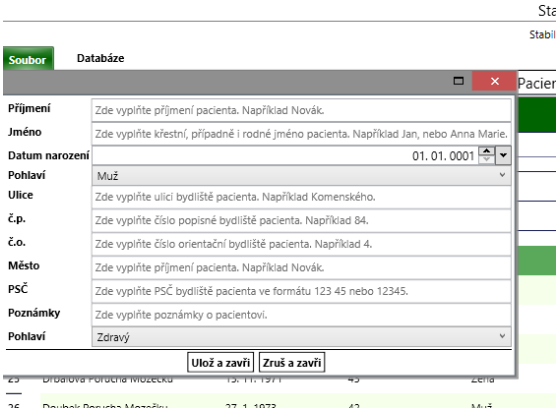
Obrázek 6.6: Výběr pacienta a zobrazení jeho detailu se všemi informacemi.



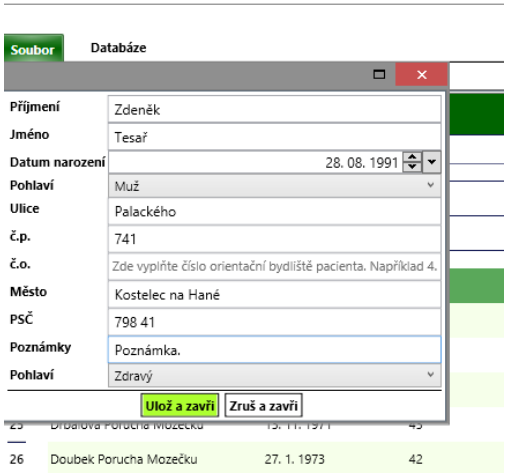
Obrázek 6.7: Zobrazený detail pacienta se všemi informacemi.



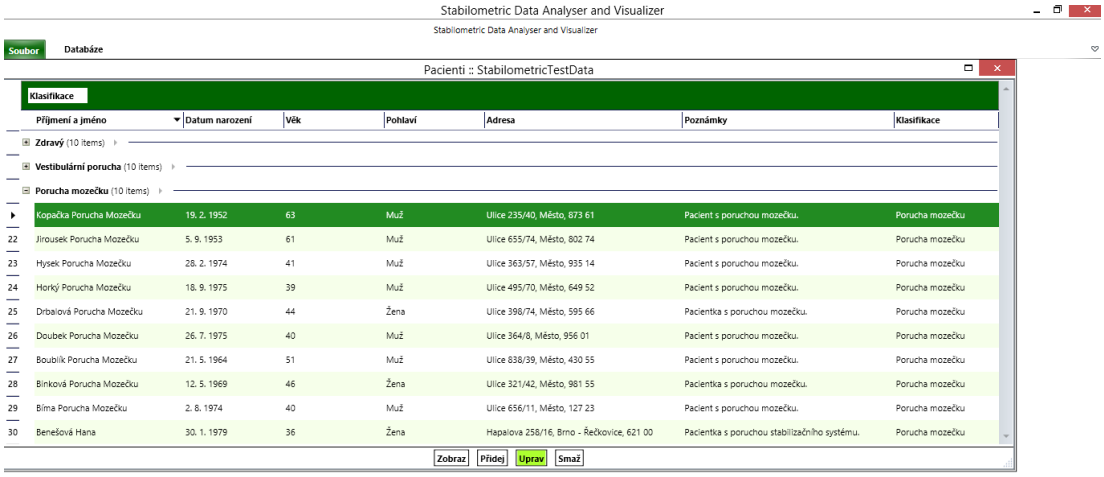
Obrázek 6.8: Tlačítko pro přidání nového pacienta do databáze.



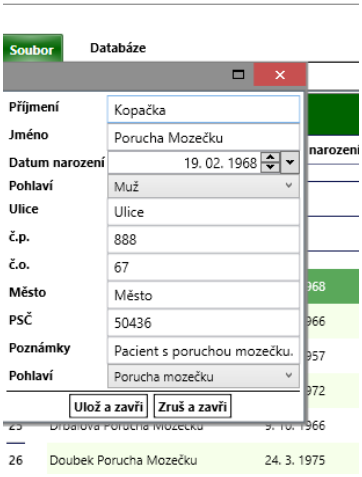
Obrázek 6.9: Zobrazené okno s položkami pro přidání nového pacienta do databáze.



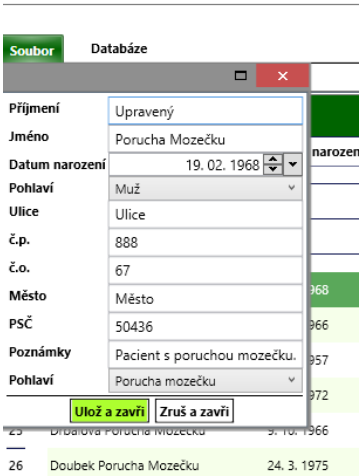
Obrázek 6.10: Vyplněné okno s položkami pro přidání nového pacienta do databáze s možností jej uložit, nebo zahodit povedené změny.



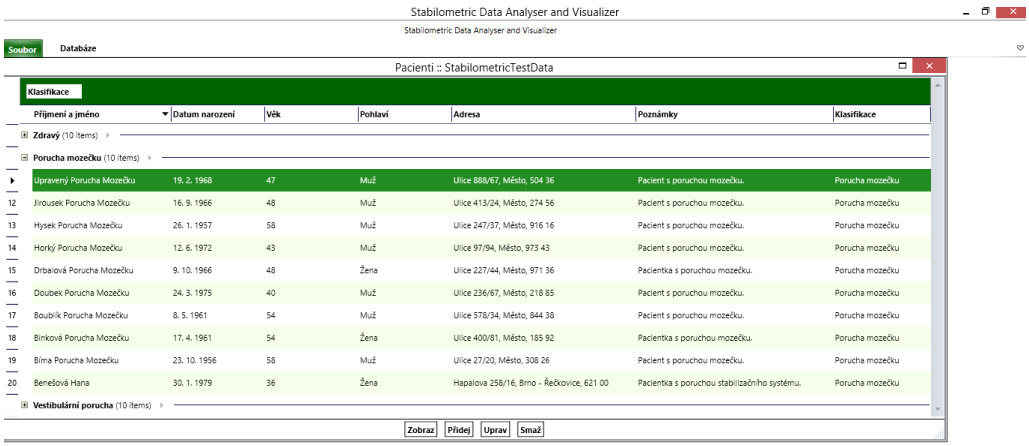
Obrázek 6.11: Výběr pacienta a zobrazení okna s možnostmi úprav jeho detailu.



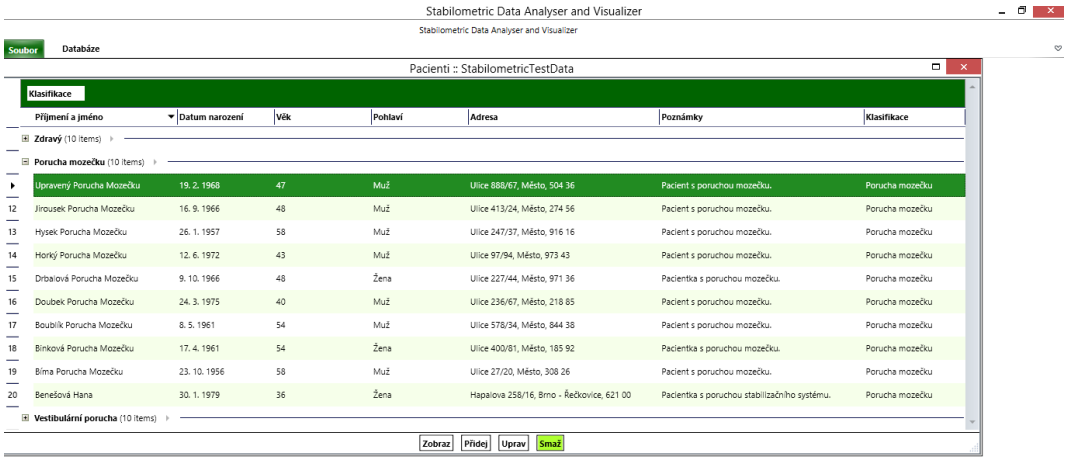
Obrázek 6.12: Zobrazené okno s vyplněnými položkami existujícího pacienta připravené k editování.



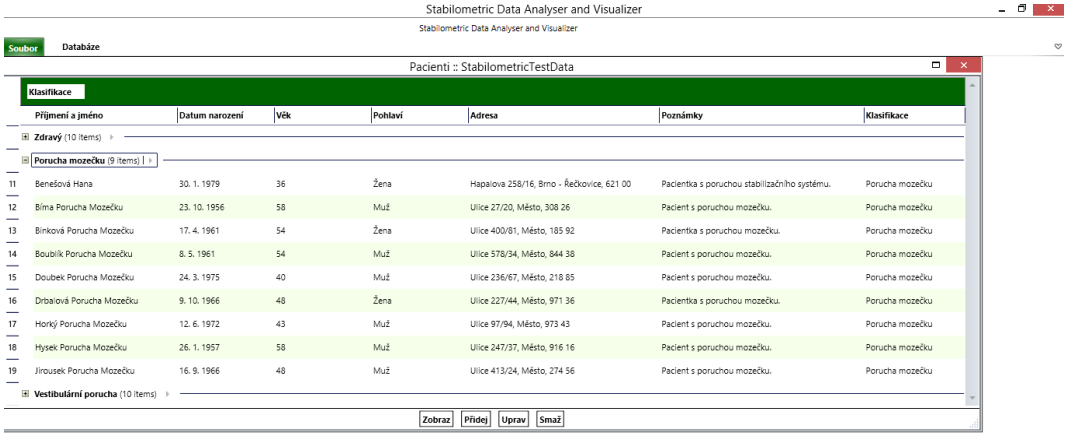
Obrázek 6.13: Okno s upraveným detailem pacienta s možností jej uložit, nebo zahodit povedené změny.



Obrázek 6.14: Pacienti z databáze včetně námi upraveného pacienta z předchozího kroku.

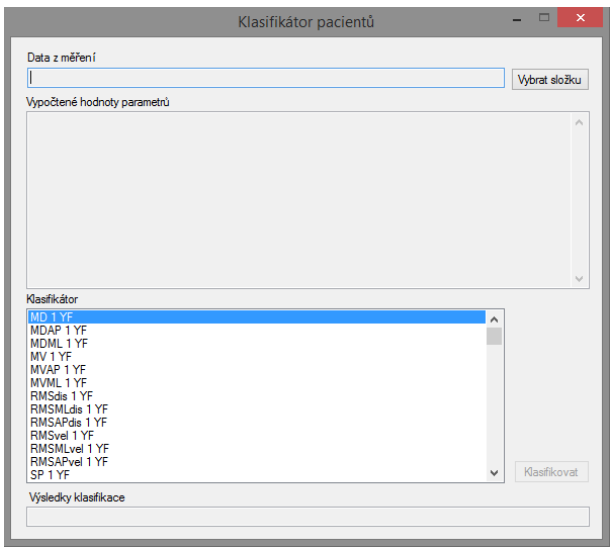


Obrázek 6.15: Výběr pacienta a tlačítko pro jeho smazání z databáze.

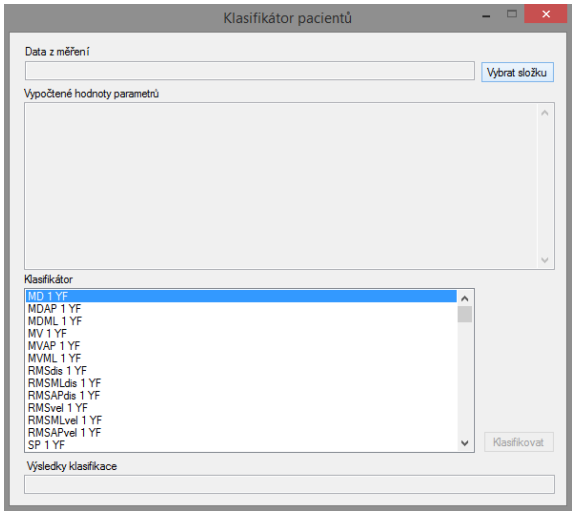


Obrázek 6.16: Pacienti z databáze včetně námi smazaného pacienta z předchozího kroku.

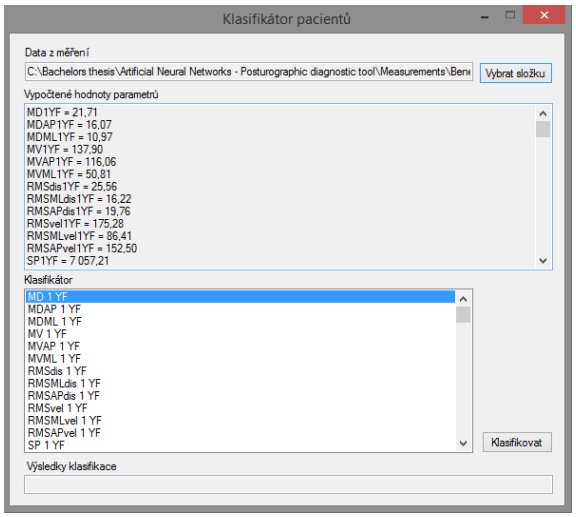
6.1.2 ANN Patients Classifier



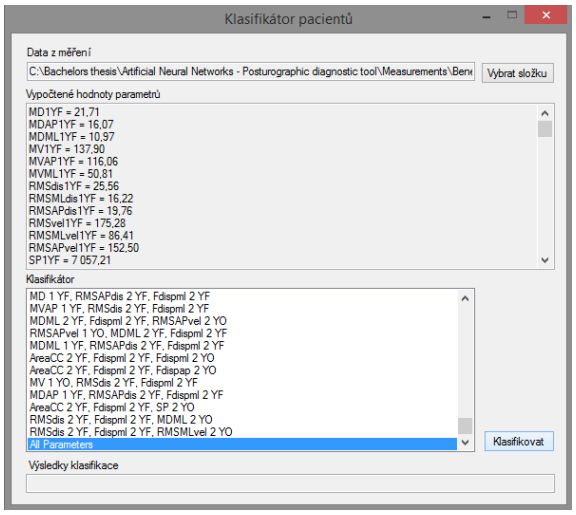
Obrázek 6.17: Spuštěný klasifikátor pacientů pomocí předpřipravených NS.



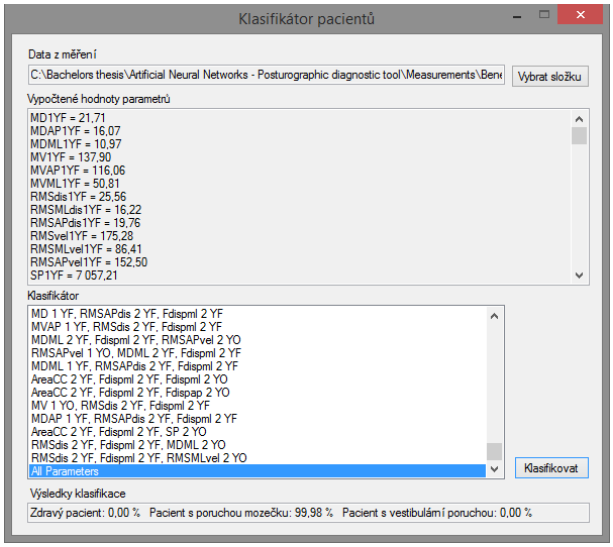
Obrázek 6.18: Tlačítko pro výběr adresáře s měřením pro daného pacienta.



Obrázek 6.19: Vypočtené parametry pro měření pro vybraného pacienta.



Obrázek 6.20: Výběr NS pro klasifikaci pacienta dle vypočtených parametrů.



Obrázek 6.21: Výsledky klasifikace pacienta dle vybrané NS.

6.2 Experimentální nasazení

Nasazení konceptu aplikace probíhá ve dvou fázích. První se týká databázové části pro správu pacientů a měření. Ta zatím stále probíhá na stejném stroji, jako samotné programování z důvodu testování na generovaných datech, které se při každém spuštění vrátí do původního stavu. Tato část aplikace bude do reálného provozu a testování na cizím stroji nasazena až po finálním dokončení obsahu databáze a její správy. O aktuálním stavu této části a důvodech pro něj je možno se dočíst v jednotlivých kapitolách této bakalářské práce.

Pro potřeby testování na cizím stroji je nutno ve zdrojových kódech upravit cesty k jednotlivým datům z měření, která se používají při generování databáze. Pokud bude při spouštění docházet k problémům souvisejícím z výpočty parametrů, pravděpodobně bude nutno doinstalovat MATLAB Runtime knihovny. Všechna potřebná data, zdrojové kódy a instalační soubory pro MATLAB Runtime knihovny nalezneme v příloze A.

Druhá fáze týkající se vlastní klasifikace pacientů pomocí NS je již v pokročilejší fázi. Aktuálně se testuje aplikace ANN Patients Classifier přímo na PC PhDr. Ondřeje Čarta, Ph.D. Ta již umí vše potřebné pro další průběžný výzkum, přičemž ten souběžně probíhá ještě za pomocí dalších nástrojů, vizme kapitolu 7.

Jako v předchozím případě jsou všechny potřebné soubory dostupné v příloze A. Zde již není potřeba spouštět aplikací pomocí kompilace zdrojových kódů, ale máme k dispozici přímo instalační soubory, které nevyžadují žádné další zásahy. Aplikaci je možno vyzkoušet na testovacích datech dostupných v téže příloze.

Součástí této kapitoly jsou i přiložená videa v příloze A s ukázkou běžících aplikací.

7. Experimentální výsledky

7.1 Výběr použitých vstupních parametrů

Pro správnou funkci NS je třeba ji učit na vhodných vstupních datech, která jsou dostatečně signifikantní pro správné rozdělení vstupů do jednotlivých výstupních tříd. Vhodnost jednotlivých parametrů jako vstupu do NS je možné testovat buď experimentálně, nebo ji určit na základě velikosti směrodatných odchylek.

7.1.1 Vhodné parametry dle Rocchiové [35]

V článku L. Rocchiové, L. Chiariho a A. Cappella [35], dále jen Rocchi, byla experimentálně testována vhodnost jednotlivých parametrů pro diagnostiku pacientů. Množina testovaných parametrů byla velmi podobná těm, které používáme, proto by bylo možné vycházet z jejich výsledků. Jelikož se ovšem jednalo o obecnější klasifikaci pacientů, než děláme my, můžeme k těmto datům pouze přihlížet.

Dle Rocchiové [35] jsou vhodné parametry pro klasifikaci následující:

- $\mathrm{RMS_{AP}}$
- $\mathrm{MV_{ML}}$
- $\mathrm{RMS_{ML}}$
- $\mathrm{F95_{AP}}$
- $\mathrm{FD_{AP}}$
- $\mathrm{MV_{AP}}$

7.1.2 Vhodné parametry dle směrodatných odchylek

Další možností, jak určit vhodné vstupní parametry, je metodika použitá v Krafczykovi [9].

Definice. *Pro vektor A o délce N je směrodatná odchylka definována jako*

$$S = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}|A_i - \mu|^2}$$
$$\mu = \frac{1}{N}\sum_{i=1}^{N} A_i$$

Nejprve změříme a vypočteme hodnoty všech parametrů u zdravých pacientů. Poté tato čísla zprůměrujeme a vypočítáme směrodatnou odchylku. Obdobně změříme a vypočteme hodnoty všech parametrů i u pacientů s jednotlivými poruchami, jež také zprůměrujeme. Následně vytvoříme tabulku pro zdravé pacienty, ve které do polí k jednotlivým parametrům zapíšeme hodnotu 0. V tabulkách u pacientů s poruchami do polí zaneseme hodnoty:

$$\frac{|prumer\_parametru_{pacient\ s\ poruchou}| - |prumer\_parametru_{zdravy\ pacient}|}{smerodatna\_odchylka\_parametru_{zdravy\ pacient}}$$

Z nich bychom poté vybrali parametry, které mají v jednotlivých tabulkách u pacientů s poruchami nejvyšší hodnoty. Nepoužijeme pouze tuto metodu, protože chceme zjistit, jak spolu korelují tyto hodnoty a výsledky z experimentu.

Důvodem je možný výběr navzájem se vylučujících parametrů, nebo opomenutí parametru s nižší hodnotou v tabulce, který může zbylé vhodně doplňovat.

Dle směrodatných odchylek by byly nejvhodnější parametry následující:

Měření na pevné podložce se zavřenýma očima

- MD
- MDML
- MV
- MVAP
- MVML
- RMSdis
- RMSMLdis
- RMSvel
- RMSMLvel
- RMSAPvel
- SP
- AreaCC
- AreaCE
- AreaSW
- Area
- POWERml
- POWERap

Měření na pevné podložce s otevřenýma očima

- AreaSW

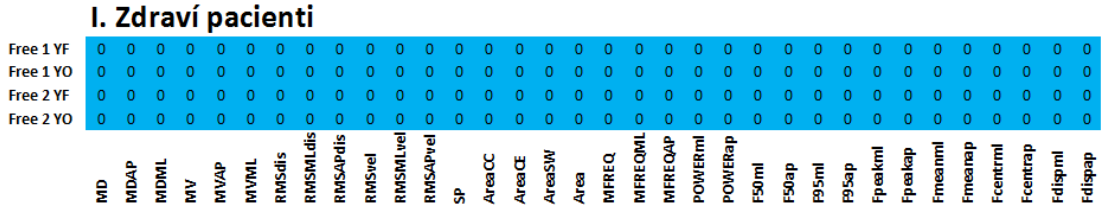
Měření na měkké podložce se zavřenýma očima

- MD
- MDML
- MV
- MVAP
- MVML
- RMSdis
- RMSMLdis
- RMSvel
- RMSMLvel
- RMSAPvel
- SP
- AreaCC
- AreaCE
- AreaSW
- Area
- POWERml
- POWERap

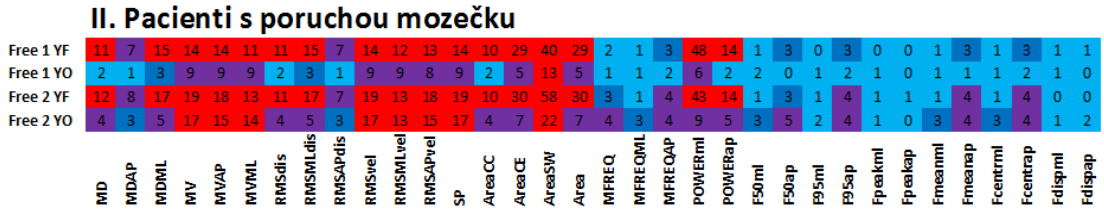
Měření na měkké podložce s otevřenýma očima

- AreaCE
- AreaSW
- Area
- POWERml
- POWERap

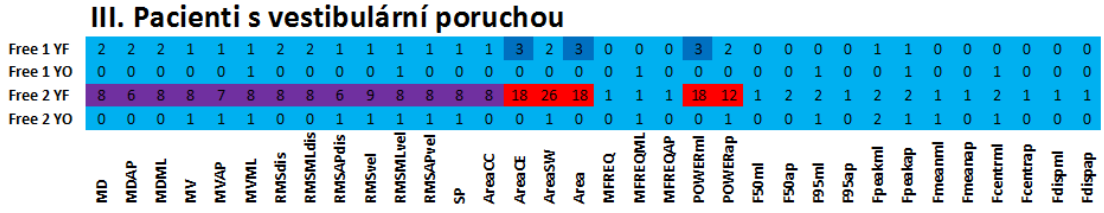
Tabulky s jednotlivými hodnotami si můžeme prohlédnout na obrázku 7.1. Jak si všimneme, všechny parametry z Rocchiové [35] jsou i mezi námi vybranými parametry. Jejich počet by se dal ještě zredukovat tím, že bychom nevybrali všechny, jejichž hodnota v tabulce je > 10, ale například > 20. Tento postup jsme ovšem zvolili na základě Krafczyka [9]. Popis parametrů a vzorce pro jejich výpočet nalezneme v podkapitole 1.3.2.



(a) Zdraví pacient.



(b) Pacienti s poruchou mozečku.



(c) Pacienti s vestibulární poruchou.

Obrázek 7.1: Tabulky směrodatných odchylek jednotlivých parametrů.

7.1.3 Experimentální výběr

Poslední možností je výběr parametrů na základě experimentu. Pouze tento postup také nebyl zvolen z důvodu vysoké časové a výpočetní náročnosti pro simulaci všech možných kombinací.

7.1.4 Shrnutí

Vzhledem k tomu, že jsme neměli k dispozici tak velkou množinu pacientů, nebylo možné jakýmkoliv z výše uvedených způsobů dosáhnout opravdu exaktních výsledků. Naším cílem bylo pouze ukázat, že metoda klasifikace pacientů pomocí NS je možná a dosahuje uspokojivých výsledků.

Dopředu však již můžeme říct, že kvalita a výsledky NS naučených na vybraných parametrech nebudou plně relevantní, neboť jak vyplývá z článků Maurizio Schmida [36], dále jen Schmid, a Prieta [5] – výsledky z posturografické plošiny ovlivňuje navíc ještě věk a zrakové schopnosti daného jedince.

7.2 Experimentální testování NS

7.2.1 Popis experimentu

Abychom mohli zhodnotit funkčnost NS a podívat se na korelaci parametrů s těmi, jež jsme vybrali na základě směrodatné odchylky, museli jsme nejprve vytvořit jednoduchý experiment. K tomuto účelu jsme použili software Wolfram

Mathematica 10.0.1+ včetně propojení s MATLABem. Kompletní zdrojový kód je k nalezení v příloze A, kde je i důkladně okomentován a připraven k ukázce ve formě prezentace.

Wolfram Mathematica [37] Jedná se o profesionální výpočetní software s možností využití v mnoha oblastech – matematických, vědeckých, inženýrských, výpočetních a dalších. Od verze 10.0.1 podporuje klasifikaci dat pomocí NS.

Stejně jako u testovacích dat zmíněných v podkapitole 3.6, jsou všechny osobní údaje u naměřených dat náhodně vygenerovány pro zachování lékařského tajemství.

Průběh experimentu:

1. Shromáždění naměřených testovacích dat (pro jejich popis vizme kapitolu 1).
   - 10 zdravých pacientů
   - 10 pacientů s poruchou mozečku
   - 10 pacientů s vestibulární poruchou
2. Import veškerých dat pro každého pacienta.
3. Extrakce potřebných dat pro výpočet všech parametrů.
   - $COPx$ a $COPy$ z měření na pevné podložce se zavřenýma očima.
   - $COPx$ a $COPy$ z měření na pevné podložce s otevřenýma očima.
   - $COPx$ a $COPy$ z měření na měkké podložce se zavřenýma očima.
   - $COPx$ a $COPy$ z měření na měkké podložce s otevřenýma očima.
4. Výpočet parametrů, vizme podkapitolu 1.3.2.
5. Rozdělení pacientů (vypočtených parametrů) do klasifikačních tříd a tvorba trénovacích/testovacích dat pro učení a testování NS.
6. Vytvoření a trénink NS s jedním vstupnim parametrem.
7. Testování NS na vstupních datech.
8. Zobrazení výsledků z testu NS s jedním vstupním parametrem.
   - Použité parametry.
   - Přesnost NS ($\frac{pocet\_spravne\_klasifikovanych\_pacientu}{pocet\_vsech\_pacientu}$).
   - Chybová matice NS.
   - Přesnost určení jednotlivých klasifikačních tříd ($\frac{pocet\_spravne\_klasifikovanych\_pacientu\_z\_dane\_tridy}{pocet\_vsech\_pacientu\_v\_dane\_tride}$).
   - Počet skrytých vrstev v NS.
   - Počet skrytých neuronů v jednotlivých skrytých vrstvách NS.
   - Typy aktivačních funkcí v neuronech v jednotlivých skrytých vrstvách.
9. Zhodnocení výsledků pro NS s jedním vstupním parametrem.
10. Opakování kroků 6. až 9. pro NS se dvěma vstupními parametry (všechny možné kombinace), třemi vstupními parametry (všechny možné kombinace NS se dvěma vstupními parametry s přesností = 90 %) a všemi parametry použitými jako vstupní.

Při experimentu byla použita funkce `Classify` [38] spolu s asociativními poli. Tyto funkce jsou v programu Wolfram Mathematica dostupné až od verze 10, přičemž možnost použití NS ke klasifikaci dat je dostupná až od verze 10.0.1. Použití funkce `Classify` k tvorbě NS pro klasifikaci pacientů si můžeme prohlédnout v ukázce kódu 7.1.

```
GetClassifierFunctions[trainingSets_List] :=
    Monitor[
        Table[
            Classify[trainingSets[[i]], Method ->
               ↪ "NeuralNetwork",  PerformanceGoal ->
               ↪ "Quality"],
            {i, Length[trainingSets]}],
        ProgressIndicator[i, {1, Length[trainingSets]}]];
GetClassifierFunctions::usage =
    "GetClassifierFunctions[trainingSets_List] vytvori
      ↪ pole klasifikatoru (umelych neuronovych siti) pro
      ↪ jednotliva testovaci data z \"trainingSets\".";
```

Ukázka kódu 7.1: Použití funkce `Classify` [38] k tvorbě NS pro klasifikaci pacientů.

Přepínače (options) funkce `Classify` [38] a jejich nastavení:

ClassPriors (Nepoužito) Určuje explicitní předpokládané pravděpodobnosti pro jednotlivé výstupní třídy. Pokud není použito, pravděpodobnosti se dedukují z trénovacích dat.

FeatureNames (Nepoužito) Specifikuje jména jednotlivých parametrů. Jelikož pro učení používáme asociativní pole, ve kterém máme jména jednotlivých parametrů explicitně uvedena, nemusíme specifikovat.

FeatureTypes (Nepoužito) Specifikuje, jak mají být jednotlivé parametry interpretovány. Pokud nespecifikujeme, interpretace se provede automaticky na základě učících dat.

IndeterminateThreshold (Nepoužito) Určuje, pod jakou pravděpodobností by měla být klasifikovaná data považována za neurčitelná. Výchozí hodnota je 0.

Method Metoda použítá pro klasifikaci dat. Nastaveno na `"NeuralNetwork"` – NS.

NominalVariables (Nepoužito) Určuje, se kterými parametry je třeba zacházet jako s diskrétními hodnotami. Pokud není použito, dedukuje se z trénovacích dat.

PerformanceGoal Určuje, který aspekt výkonu se algoritmus pokusí optimalizovat. Nastaveno na `"Quality"` – upřednostňuje kvalitu NS, před rychlostí získání výsledku klasifikace.

UtilityFunction (Nepoužito) Určuje hodnotu přiřazenou ke každému možnému párování skutečných a predikovaných hodnot. Pokud není použito, dedukuje se z trénovacích dat.

ValidationSet (Nepoužito) Specifikuje validační sadu dat, která mají být použita během učící fáze. Pokud není specifikování, bude použita křížová validace ze vstupních dat.

7.2.2 Shrnutí experimentu

Ve shrnutí experimentu opomineme část, kdy jsme zkoušeli NS se všemi parametry použitými jako vstupní.V té jsme dosáhli 100% celkové úspěšnosti klasifikace. Pro zjištění informací o NS jsme použili funkci `ClassifierMeasurements` [39]. Použití funkce `ClassifierMeasurements` [39] si můžeme prohlédnout v ukázce kódu 7.2.

Ukázky výsledného zobrazení informací o jednotlivých NS si můžeme prohlédnout na obrázcích u příslušných částí experimentu.

```
ClassifierMeasurementsProperties =
    {"Accuracy", "ConfusionMatrixPlot", "Precision"};

GetClassifierMeasurementsProperties[
   ↪ classifiersMeasurements_List] :=
    (#1 /@ ClassifierMeasurementsProperties &) /@
       ↪ classifiersMeasurements;
GetClassifierMeasurementsProperties::usage =
    "GetClassifierMeasurementsProperties[
       ↪ classifiersMeasurements_List] vrati seznam
       ↪ vlastnosti ClassifierMeasurements klasifikatoru
       ↪ pro kazdy ClassifierMeasurements z
       ↪ \"classifiersMeasurements_List\".";

GetClassifierInformationsProperties[
   ↪ classifierFunctions_List] :=
    ({ClassifierInformation[#1, "NumberHiddenLayers"],
       ↪ ClassifierInformation[#1, "NumberHiddenNodes"],
       ↪ ClassifierInformation[#1,
       ↪ "ActivationFunctionTypes"]} &) /@
       ↪ classifierFunctions;
GetClassifierInformationsProperties::usage =
    "GetClassifierInformationsProperties[
       ↪ classifierFunctions_List] vrati hodnoty parametru
       ↪ \"NumberHiddenLayers\", \"NumberHiddenNodes\"
       ↪ a~\"ActivationFunctionTypes\" pro kazdy
       ↪ klasifikator ze seznamu klasifikatoru
       ↪ \"classifierFunctions\".";
```

Ukázka kódu 7.2: Použití funkce `ClassifierMeasurements` [39] pro zjištění informací o jednotlivých NS.

Pro pochopení hodnot parametru *prop* funkce `ClassifierMeasurements` [39] si definujme následující pojmy. Při jejich definici vycházíme z článku Mariny Sokolove a Guye Lapalmeho [40].

Definice. *Jako chybovou matici označíme čtvercovou matici $A \in \mathcal{Z}^{n\times n}$, kde i-tý řádek značí skutečnou klasifikační třídu j-tý sloupec predikovanou klasifikační třídu. Potom prvek $A_{i,j}$ obsahuje počet příkladů třídy i klasifikovaných jako třída j.*

Definice. *Mějme chybovou matici $A \in \mathcal{Z}^{n\times n}$, pak*

$$Precision_i = \frac{A_{i,i}}{\sum_j A_{j,i}}$$
$$Recall_i = \frac{A_{i,i}}{\sum_j A_{j,i}}$$
$$Fscore = 2\frac{Precision_i Recall_i}{Precision_i + Recall_i}$$

Možné hodnoty parametru *prop* funkce `ClassifierMeasurements` [39]:

{"Examples",$i \to j$} (Nepoužito) Jsou to všechny příklady třídy i klasifikované jako j.

"Accuracy" Podíl správně klasifikovaných příkladů ke všem příkladům.

"AccuracyRejectionPlot" Graf přesnosti jako funkce míry odmítnutí.

"ConfusionFunction" Funkce vracející hodnoty chybové matice.

"ConfusionMatrix" Chybová matice.

"ConfusionMatrixPlot" Grafické znázornění chybové matice.

"Error" Podíl špatně klasifikovaných příkladů ku všem příkladům.

"FScore" Fscore pro každou klasifikační třídu.

"LogLikelihood" Logaritmická pravděpodobnost modelu vůči testovacím datům.

"LogLikelihoodRate" Průměrná logaritmická pravděpodobnost testovacího příkladu.

"Precision" Presicion klasifikace pro každou klasifikační třídu.

"Properties" Seznam možných hodnot parametru *prop* této funkce.

"Recall" Recal klasifikace pro každou klasifikační třídu.

"RejectionRate" Podíl příkladů klasifikovaných jako `Indeterminate` – neurčitý.

7.2.2.1 První část experimentu

V první části experimentu, při kterém měly použité NS jeden vstupní parametr, jsme došli k následujícím výsledkům:

Nejvhodnější parametry pro klasifikaci pacientů pomocí umělé neuronové sítě s jedním vstupním parametrem.

- AreaSW 2 YO s 83,3333% úspěšností klasifikace.
- MVAP 2 YO s 80% úspěšností klasifikace.
- SP 2 YF s 80% úspěšností klasifikace.

- AreaSW 2 YF s 80% úspěšností klasifikace.
- RMSvel 2 YF s 80% úspěšností klasifikace.
- MV 2 YF s 80% úspěšností klasifikace.

Při porovnání nejúspěšnějších NS s vhodnými parametry dle Rocchiové [35] vidíme, že úplná shoda nastala pouze v jednom doporučeném parametru. Jako částečnou shodu poté můžeme brát ještě parametr RMSvel s nabízenými $\mathrm{RMS_{AP}}$ a $\mathrm{RMS_{ML}}$.

Pokud nahlédneme do obrázku 7.1, vychází nám parametr AreaSW jako nejlepší. Výsledek je z měření na měkké podložce s otevřenýma očima, nikoliv zavřenýma. Stejný závěr bychom předpokládali i u parametru MVAP. U zbylých čtyř se již shodujeme i s obrázkem.

U výše uvedených NS je jejich celková úspěšnost způsobena vysokou chybovostí v klasifikaci pacientů s vestibulární poruchou. U těchto pacientů se klasifikace ukázala jako největší problém.

Můžeme zhodnotit, že nejlepších výsledků dosahovaly NS se vstupními parametry z měření na měkké podložce.

Z počtu 136 umělých neuronových sítí s jedním vstupním parametrem celkově 13 dosáhlo na celkovou úspěšnost 75 % a více, z toho 6 na 80 % a více, což můžeme brát za velmi uspokojivý výsledek.

Pro ukázku jsme vyzkoušeli u 6 nejlepších parametrů i klasifikaci pomocí lineární regrese. Výsledky dosahovaly obdobnou celkovou úspěšnost klasifikace jako NS se stejným vstupním parametrem. Při předběžné klasifikaci bychom tedy mohli použít následující vzorce.

AreaSW 2 YO

$$\begin{aligned} mozecek &= -3,00332 + 0,0204787 AreaSW\_2\_YO \\ vestibular &= 1,0569 - 0,00325922 AreaSW\_2\_YO \\ zdravy &= 1,94642 - 0,0172195 AreaSW\_2\_YO \\ klasifikacni\_trida &= \max(mozecek, vestibular, zdravy) \end{aligned}$$

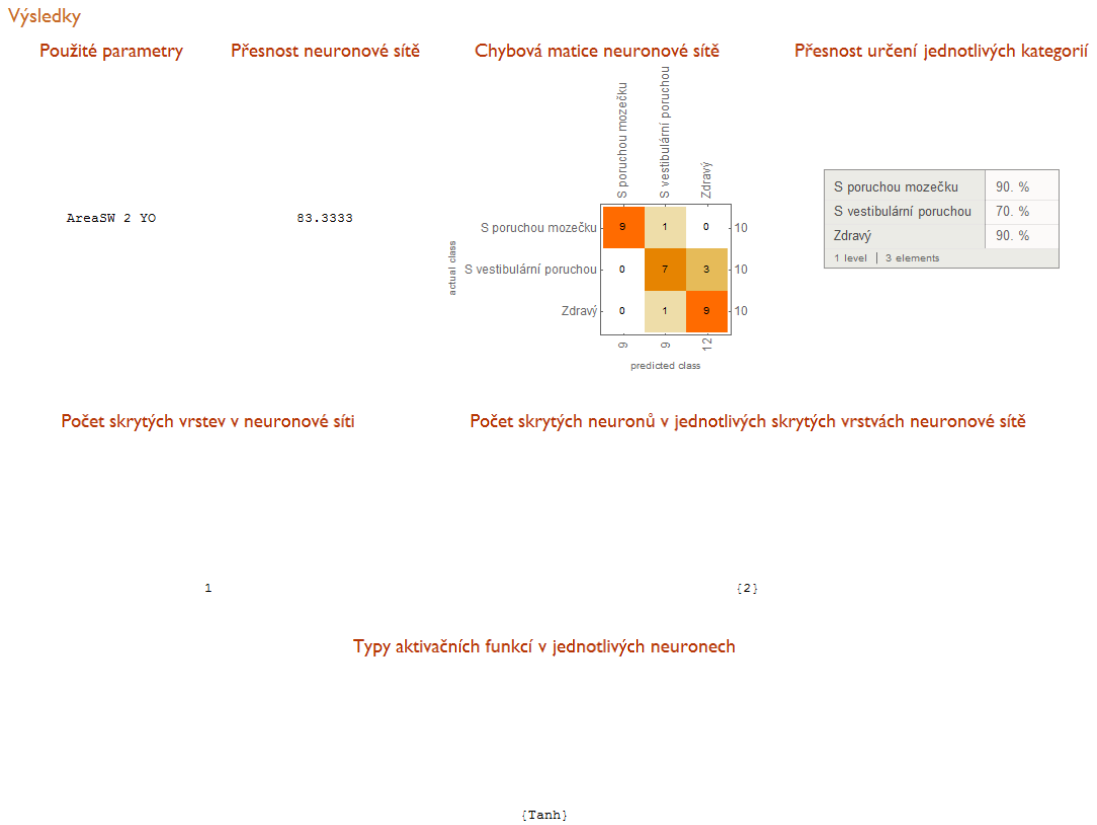
MVAP 2 YO

$$\begin{aligned} mozecek &= -3,96842 + 0,220087 MVAP\_2\_YO \\ vestibular &= 1,25565 - 0,04230742 MVAP\_2\_YO \\ zdravy &= 2,71277 - 0,17778 MVAP\_2\_YO \\ klasifikacni\_trida &= \max(mozecek, vestibular, zdravy) \end{aligned}$$

SP 2 YF

$$\begin{aligned} mozecek &= -2,65987 + 0,000594471 SP\_2\_YF \\ vestibular &= -0,338649 + 0,000263387 SP\_2\_YF \\ zdravy &= 2,99852 - 0,000857858 SP\_2\_YF \\ klasifikacni\_trida &= \max(mozecek, vestibular, zdravy) \end{aligned}$$

Ukázku zobrazení NS s celkovou nejlepší úspěšností klasifikace z první části experimentu si můžeme prohlédnout na obrázku 7.2.



Obrázek 7.2: Ukázku zobrazení NS s celkovou nejlepší úspěšností klasifikace z první části experimentu.

7.2.2.2 Druhá část experimentu

V druhé části experimentu, kdy měly použité NS dva vstupní parametry, jsme došli k následujícím výsledkům:

Nejvhodnější parametry pro klasifikaci pacientů pomocí umělé neuronové sítě se dvěma vstupními parametry.

- AreaCC 2 YF, Fdispml 2 YF se 100% úspěšností klasifikace.
- RMSAPdis 2 YF, Fdispml 2 YF se 100% úspěšností klasifikace.
- RMSdis 2 YF, Fdispml 2 YF se 100% úspěšností klasifikace
- MDML 2 YF, Fdispml 2 YF se 100% úspěšností klasifikace.

Opět můžeme konstatovat, že nejvhodnější parametry pro klasifikaci pacientů pomocí NS se dvěma vstupními parametry vycházejí z měření na měkké podložce se zavřenýma očima.

Výše uvedené NS mají společný parametr Fdispml 2 YF, který v části experimentu s NS s jedním vstupním parametrem zvládal klasifikovat správně 9 z 10 pacientů s vestibulární poruchou, ale dosahoval špatných výsledku u zdravých pacientů a pacientů s poruchou mozečku. Naopak zbylé čtyři parametry měly

velmi dobrou klasifikaci zdravých pacientů a pacientů s vestibulární poruchou, a to následovně:

- NS se vstupním parametrem AreaCC 2 YF dosahovala 73,3333% úspěšnosti klasifikace. Správně určila 8 z 10 pacientů s poruchou mozečku a 10 z 10 zdravých pacientů.
- NS se vstupním parametrem RMSdis 2 YF dosahovala 73,333% úspěšnosti klasifikace. Správně určila 8 z 10 pacientů s poruchou mozečku a 10 z 10 zdravých pacientů.
- NS síť se vstupním parametrem RMSAPdis 2 YF dosahovala 70% úspěšnosti klasifikace. Správně určila 8 z 10 pacientů s poruchou mozečku a 10 z 10 zdravých pacientů, ale oproti dvěma výše uvedeným NS správně určila pouze 3 z 10 pacientů s vestibulární poruchou (výše uvedené NS správně klasifikovaly 4 z 10 pacientů s vestibulární poruchou).
- NS se vstupním parametrem MDML 2 YF dosahovala 70% úspěšností klasifikace. Správně určila 7 z 10 pacientů s poruchou mozečku, 5 z 10 pacientů s vestibulární poruchou a 9 z 10 zdravých pacientů.

Z počtu $\binom{136}{2} = 9180$ NS se dvěma vstupními parametry celkově čtyři dosáhly na celkovou úspěšnost 100 %.

Pokud bychom chtěli zjistit, jak spolu korelují NS s jedním a dvěma vstupními parametry, tak valná většina NS se dvěma vstupními parametry s celkovou úspěšností 96,6667 % měla společný vstupní parametr s nejlepšími NSi z části experimentu s jedním vstupním parametrem.

Ukázku zobrazení NS s celkovou nejlepší úspěšností klasifikace pro každou ze čtyř částí výpočtu z druhé části experimentu si můžeme prohlédnout na obrázcích 7.3, 7.4, 7.5 a 7.6.

7.2.2.3 Třetí část experimentu

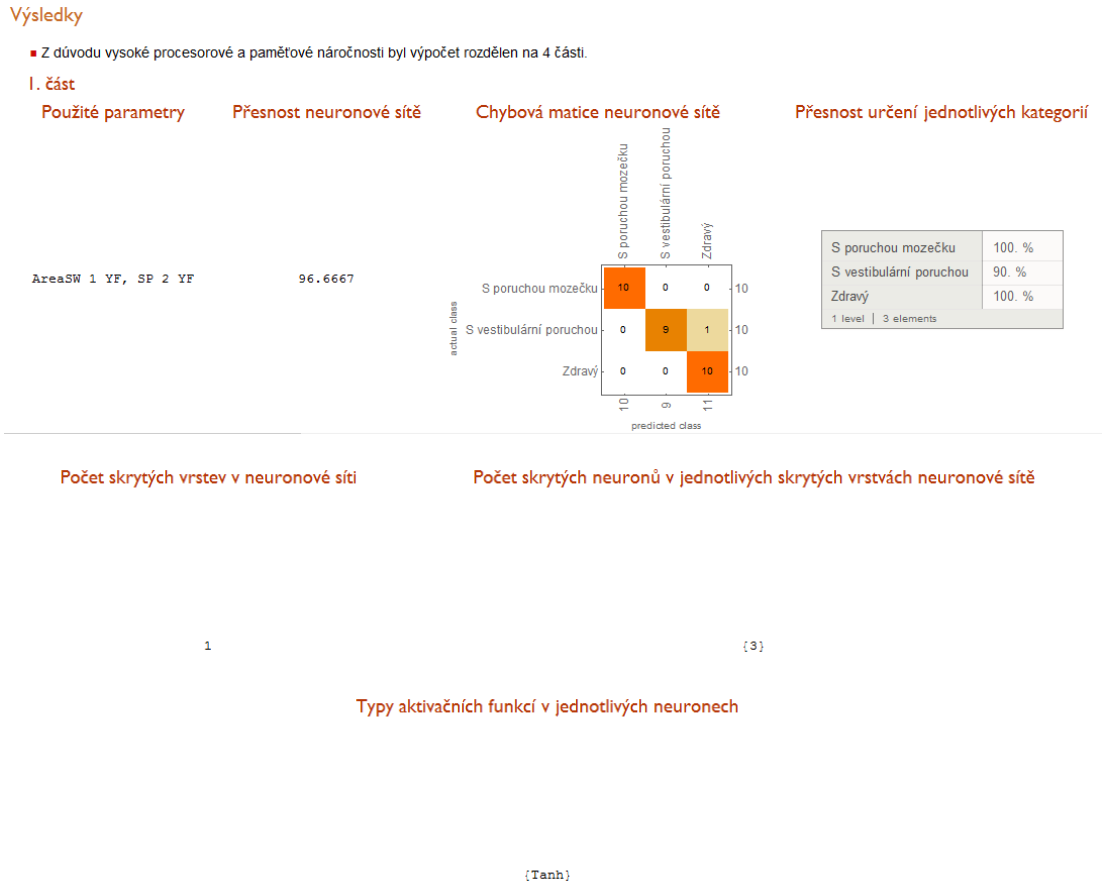
Ve třetí části experimentu, kdy měly použité NS tři vstupní parametry, jsme se zaměřili na to, zda se kvalita NS zachová, nebo dojde k jejímu zhoršení. Při tom jsme došli k následujícímu:

Ve většině případů došlo k zachování 100% celkové úspěšnosti klasifikace pacientů.

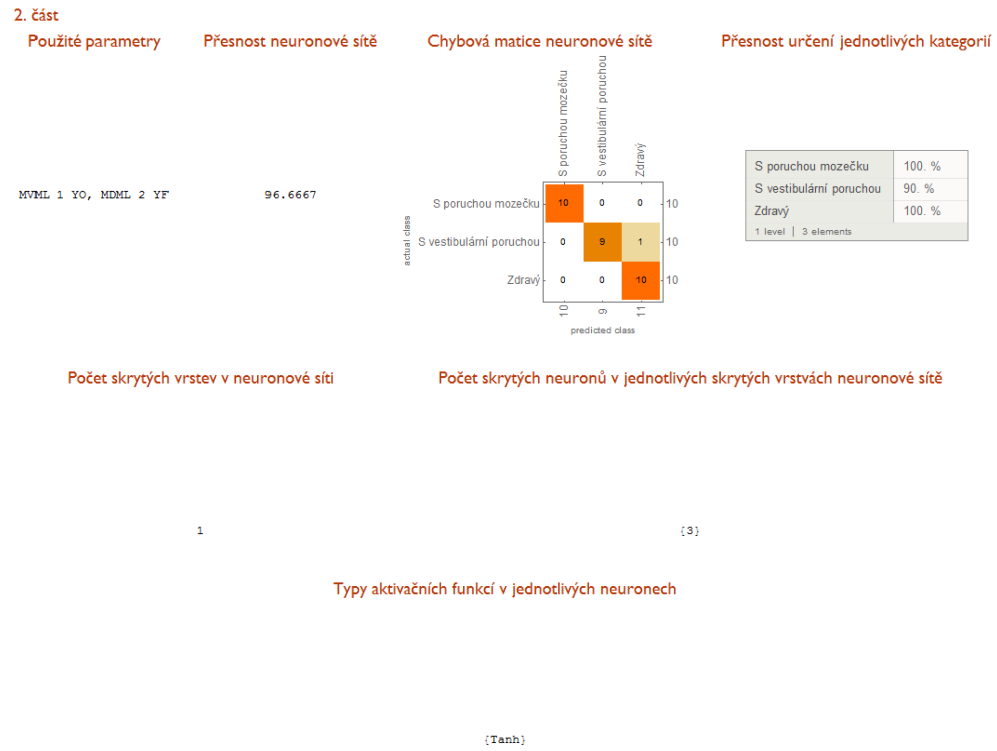
- NS se vstupními parametry AreaCC 2 YF, Fdispml 2 YF + 3. vstupní parametr měly pouze u 11 případů horší než 100% celkovou úspěšnost klasifikace. U případů s 96,6667% celkovou úspěšností docházelo nejčastěji k chybám klasifikace pacientů s vestibulární poruchou.
- NS se vstupními parametry RMSAPdis 2 YF, Fdispml 2 YF + 3. vstupní parametr měly u 19 případů horší než 100% celkovou úspěšnost klasifikace. U případů s 96,6667% úspěšností docházelo nejčastěji k chybám klasifikace pacientů s vestibulární poruchou.

- NS se vstupními parametry RMSdis 2 YF, Fdispml 2 YF + 3. vstupní parametr měly u 15 případů horší než 100% celkovou úspěšnost klasifikace. U případů s 96,6667% úspěšností docházelo nejčastěji k chybám klasifikace pacientů s vestibulární poruchou.
- NS se vstupními parametry MDML 2 YF, Fdispml 2 YF + 3. vstupní parametr měly u 17 případů horší než 100% úspěšnost klasifikace. U případů s 96,6667% úspěšností docházelo oproti jiným umělým neuronovým sítím přibližně stejně k chybným klasifikacím ve všech třech třídách.

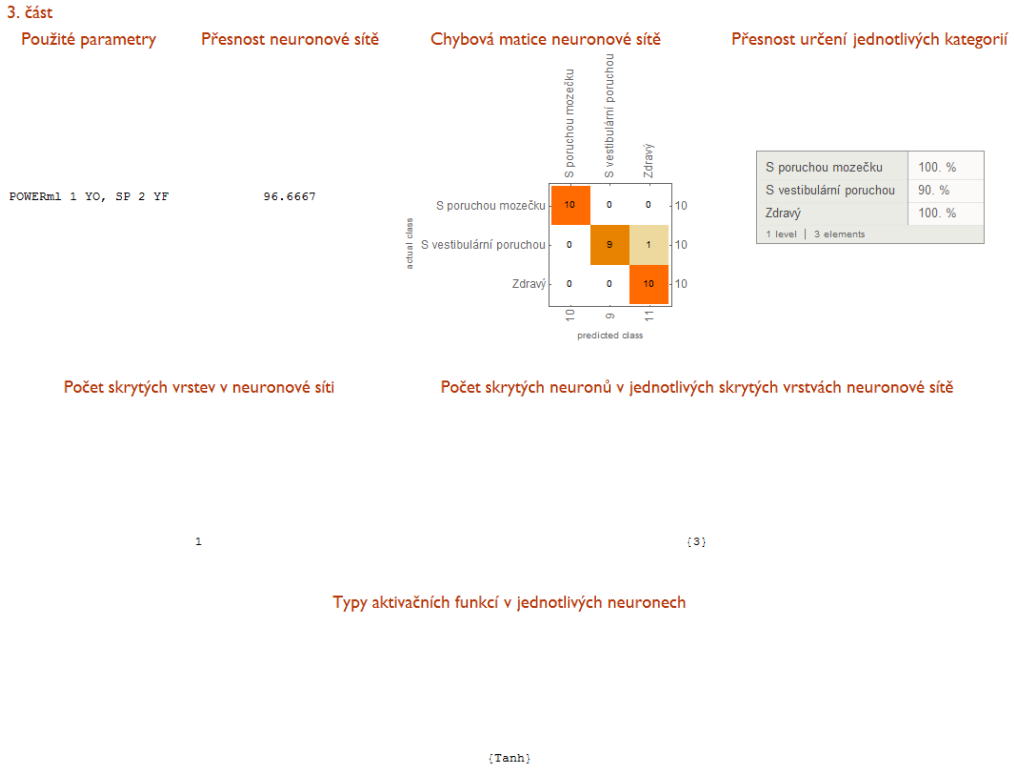
Parametry, které snížily úspěšnost klasifikace u jednotlivých NS, jsou následující:



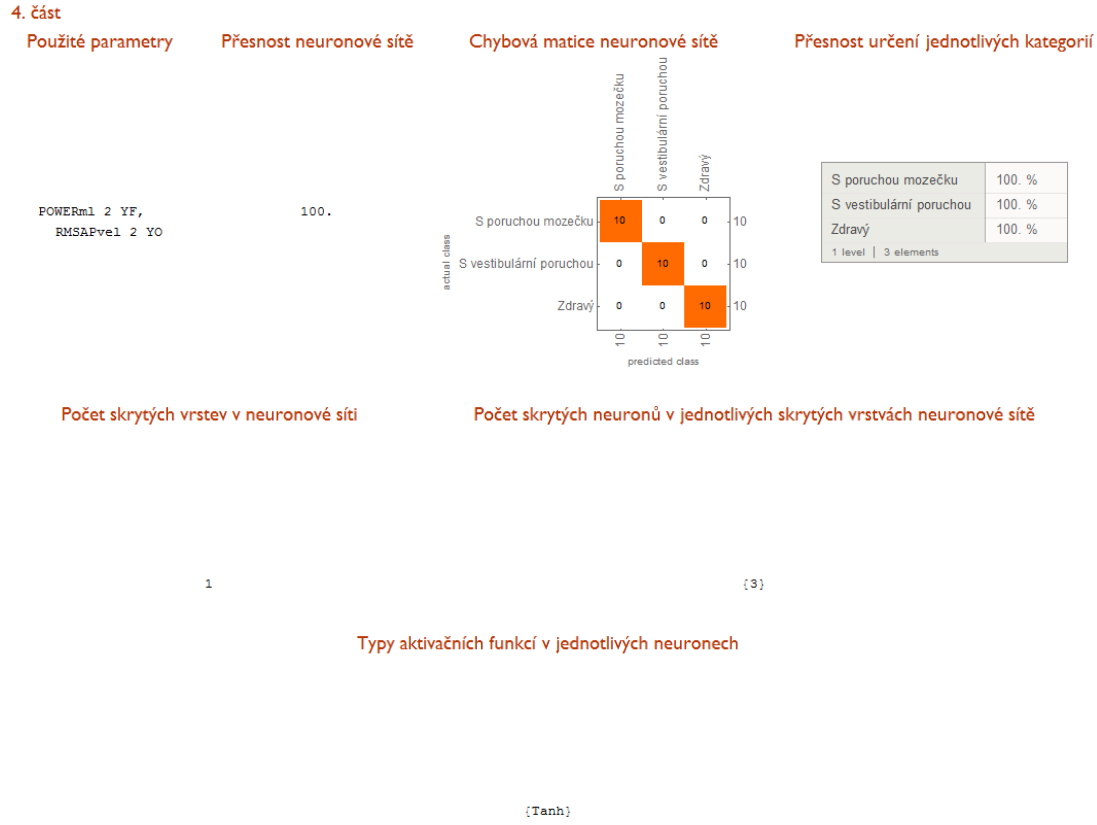
Obrázek 7.3: Ukázku zobrazení NS s celkovou nejlepší úspěšností klasifikace z první části výpočtu z druhé části experimentu.



Obrázek 7.4: Ukázku zobrazení NS s celkovou nejlepší úspěšností klasifikace z druhé části výpočtu z druhé části experimentu.



Obrázek 7.5: Ukázku zobrazení NS s celkovou nejlepší úspěšností klasifikace z třetí části výpočtu z druhé části experimentu.



Obrázek 7.6: Ukázku zobrazení NS s celkovou nejlepší úspěšností klasifikace ze čtvrté části výpočtu z druhé části experimentu.

NS se vstupními parametry AreaCC 2 YF, Fdispml 2 YF + 3. vstupní parametr:

96,6667% úspěšnost klasifikace:

- POWERml 1 YO
- MFREQ 1 YO
- RMSvel 1 YO
- MV 1 YO
- Fcentrap 1 YF
- F50ml 1 YF
- RMSMLvel 1 YF
- MVAP 1 YF

93,3333% úspěšnost klasifikace:

- Fdispml 1 YF

90% úspěšnost klasifikace:

- Fcentrml 2 YF

86,6667% úspěšnost klasifikace:

- RMSdis 1 YO

NS se vstupními parametry RMSAPdis 2 YF, Fdispml 2 YF + 3. vstupní parametr:

96,6667% úspěšnost klasifikace:

- F95ml 2 YF
- SP 1 YO
- POWERml 1 YO
- AreaCC 1 YO
- RMSMLvel 1 YO
- RMSvel 1 YO
- MVAP 1 YO
- MD 1 YO
- Fmeanap 1 YF
- F50ap 1 YF
- AreaCE 1 YF
- RMSAPvel 1 YF
- RMSvel 1 YF
- RMSdis 1 YF
- MV 1 YF

93,3333% úspěšnost klasifikace:

- MFREQML 1 YO
- MDML 1 YO
- Fdispml 1 YF
- F95ap 1 YF

NS se vstupními parametry RMSdis 2 YF, Fdispml 2 YF + 3. vstupní parametr:

96,6667% úspěšnost klasifikace:

- MFREQAP 2 YF
- RMSMLvel 1 YO
- MDAP 1 YO
- SP 1 YF
- Fdispap 1 YF,
- POWERap 1 YF
- AreaSW 1 YF
- RMSAPvel 1 YF
- RMSMLvel 1 YF
- RMSvel 1 YF
- MV 1 YF

93,3333% úspěšnost klasifikace:

- F50ap 2 YF
- RMSMLdis 1 YO

90% úspěšnost klasifikace:

- MD 1 YO
- F95ap 1 YF

NS se vstupními parametry MDML 2 YF, Fdispml 2 YF + 3. vstupní parametr:

96,6667% úspěšnost klasifikace:

- RMSAPdis 2 YO
- Fpeakap 2 YF
- RMSAPvel 2 YF
- RMSvel 2 YF
- MVML 1 YF,
- Fmeanap 1 YO
- RMSMLdis 1 YO
- RMSdis 1 YO

- MVML 1 YO
- MDML 1 YO
- Fdispap 1 YF
- Fcentrap 1 YF
- POWERap 1 YF
- MFREQAP 1 YF

93,3333% úspěšnost klasifikace:

- Fcentrml 2 YO
- F95ml 1 YF

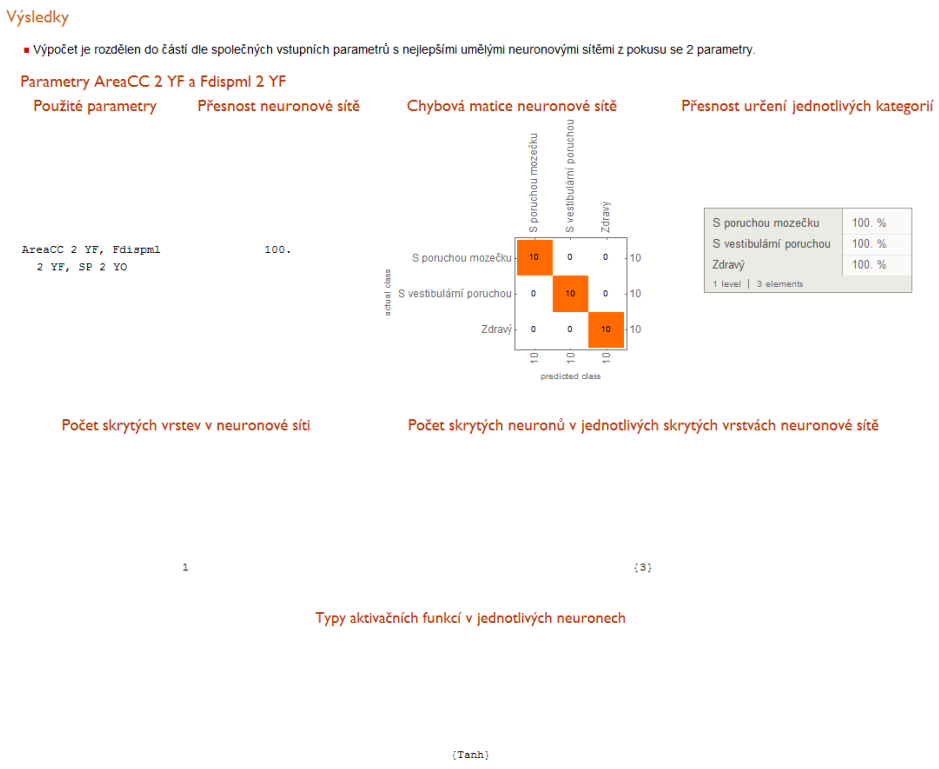
90% úspěšnost klasifikace:

- MVAP 1 YF

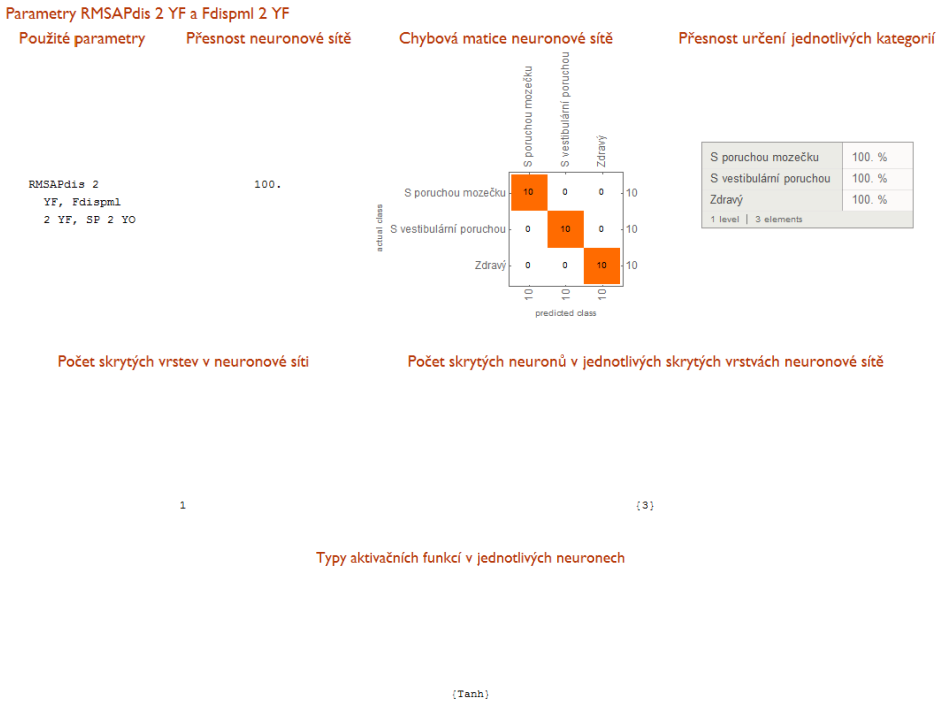
Ukázku zobrazení NS s celkovou nejlepší úspěšností klasifikace pro každou ze čtyř částí výpočtu z třetí části experimentu si můžeme prohlédnout na obrázcích 7.7, 7.8, 7.9 a 7.10.

7.3 Shrnutí

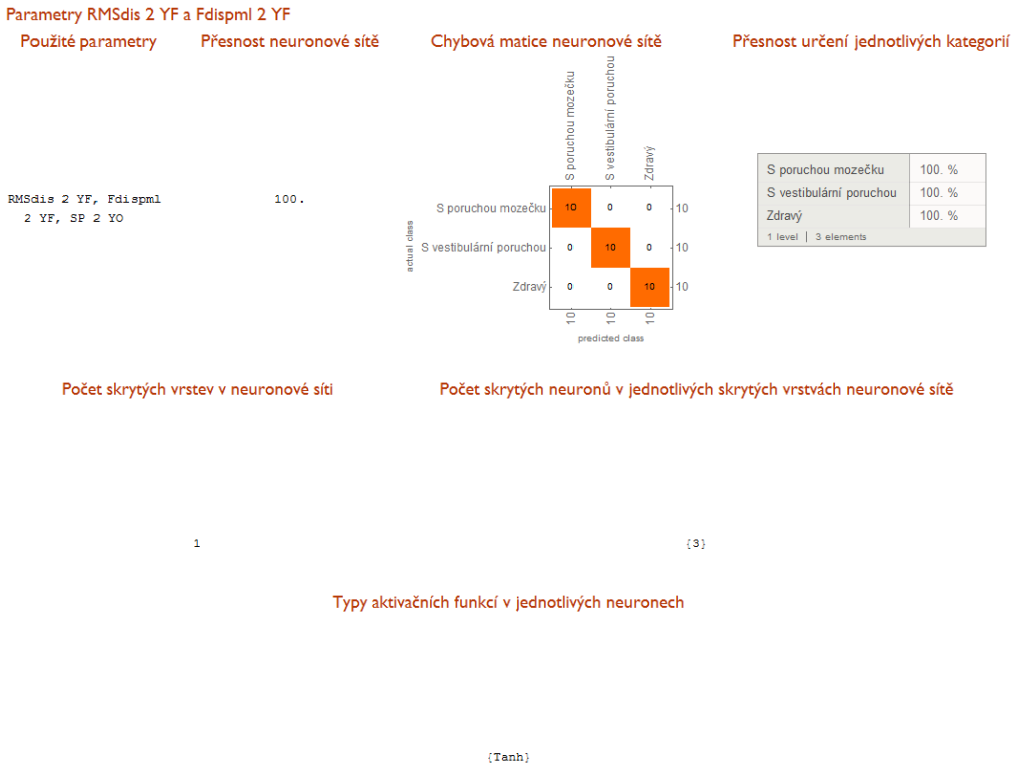
Na základě našeho malého experimentu a výsledků v dalších článcích, např. Krafczyk [9], můžeme zhodnotit, že NS jsou vhodnou klasifikační technikou pro naše potřeby. Při kombinované volbě parametrů na základě experimentu a směrodatných odchylek, s větším důrazem na výsledky experimentu můžeme pacienty klasifikovat s téměř 100% celkovou přesností. Preference výsledků experimentu je dána tím, že u směrodatných odchylek plně nehledíme na intervaly vypočtených hodnot jednotlivých parametrů, které se mohou výrazně překrývat, viz obrázek 7.11.



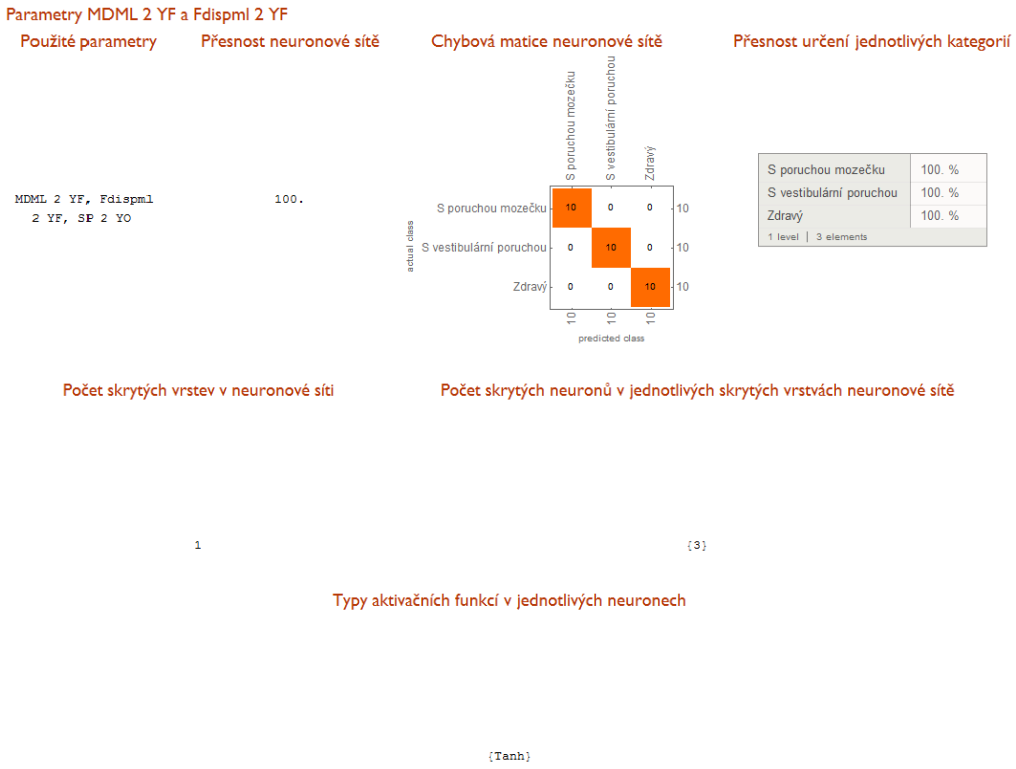
Obrázek 7.7: Ukázku zobrazení NS s celkovou nejlepší úspěšností klasifikace z první části výpočtu z třetí části experimentu.



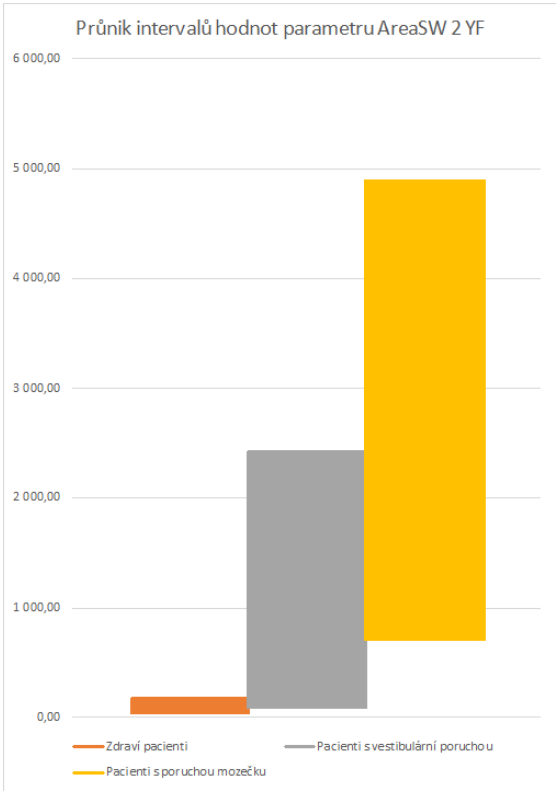
Obrázek 7.8: Ukázku zobrazení NS s celkovou nejlepší úspěšností klasifikace z druhé části výpočtu z třetí části experimentu.



Obrázek 7.9: Ukázku zobrazení NS s celkovou nejlepší úspěšností klasifikace z třetí části výpočtu z třetí části experimentu.



Obrázek 7.10: Ukázku zobrazení NS s celkovou nejlepší úspěšností klasifikace ze čtvrté části výpočtu z třetí části experimentu.



Obrázek 7.11: Průnik intervalů hodnot parametru AreaSW 2 YF.

Závěr

Hlavním výsledkem této práce je funkční koncept aplikace pro správu pacientů a jejich klasifikaci pomocí NS, jehož zdrojové kódy jsou dostupné na přiloženém DVD, vizme přílohu A.

V rámci jednotlivých kapitol jsme ve stručnosti představili posturografii. Dále jsme popsali jednotlivé části konceptu aplikace s důrazem na strukturu kódu, použité knihovny, postupy a možné problémy při dalším rozšiřování.

Při tvorbě tohoto konceptu byl kladen důraz, aby byl vlastní kód aplikace snadno přístupný, pochopitelný a rošiřitelný pro další studenty MFF UK, v rámci předmětů Ročníkový projekt, Softwarová praxe, nebo jako podklad pro rozšiřující téma v rámci bakalářských či diplomových prací. Možné nápady na tato rozšíření byla uvedena vždy jako součást jednotlivých kapitol této práce.

Pro dosáhnutí výše uvedených vlastností jsme se snažili dodržovat techniky uvedené převážně v knihách Dokonalý kód: Umění programování a techniky tvorby software [41] a Čistý kód: návrhové vzory, refaktorování, testování a další techniky agilního programování [42]. I když kvalita kódu nedosahuje úplně nejvyšších měřítek uvedených v těchto knihách, případné refaktorizace, při rozšiřování kódu pro její dosažení, by již neměly zabrat netriviální množství času.

V předposlední kapitole byla uvedena stručná uživatelská dokumentace i s ukázkovým použitím konceptu aplikace v praxi.

Na závěr jsme popsali experiment, jehož výstupy a výsledky slouží jako podklad pro odborný článek ve specializovaném časopise Česká a slovenská neurologie a neurochirurgie, jenž byl napsán ve spolupráci s PhDr. Ondřejem Čakrtem, Ph.D. Toto periodikum s článkem vyjde později v roce 2015. Podklady tohoto experimentu a dosažené výsledky také sloužily jako hlavní osnova pro vlastnosti k implementaci do vlastního kódu.

Na základě výsledků můžeme na úplný závěr zhodnotit, že směr, kterým se tato práce ubírala, není slepou uličkou. V případě zájmu dalších studentů či výzkumníků lze očekávat i brzký posun pochopení korelace výsledků z posturografických měření se zdravotním stavem a omezeními pacientů a možné nasazení klasifikačních nástrojů v praxi.

Seznam použité literatury

[1] Ganong, William F. *Přehled lékařské fyziologie*. 20. vyd. Praha: Galén, 2005, 890 s. ISBN 80-7262-311-7.

[2] Kolář, Pavel. *Rehabilitace v klinické praxi*. 1. vyd. Praha: Galén, 2009, 713 s. ISBN 978-80-7262-657-1.

[3] Vokurka, Martin a Jan Hugo. *Velký lékařský slovník*. 9., aktualiz. vyd. Praha: Maxdorf, 2009, 1159 s. ISBN 978-80-7345-202-5.

[4] *Synapsys – Vidéo Nystagmographie, Posturographie, pupillométrie, PEA* [online]. [Cit. 29. 6. 2014]. Dostupné z: http://www.synapsys.fr/en/p-synapsys-posturography-system-sps-36.htm

[5] Prieto, Thomas E. et al. Measures of Postural Steadiness: Differences Between Healthy Young and Elderly Adults. *IEEE Transactions on Biomedical Engineering*, 1996, Roč. 43, č. 9, s. 956-966. ISSN 0018-9294.

[6] Maurer, Ch. - Peterka, Robert J. A New Interpretation of Spontaneous Sway Measures Based on a Simple Model of Human Postural Control. *Journal of Neurophysiology*, 2005, Roč. 93, č. 1, s. 189-200. ISSN 0022-3077.

[7] *Synapsys Static & Dynamic Posturography® Manual version 2.7.* 2006. Marseille-France.

[8] Vyšata, O. – Přerovský, K. – Vršecká, M. Počítačová posturografie v klinické praxi. *Praktický lékař*, 1993, Roč. 73, č. 5, s. 190-192. ISSN 0032-6739.

[9] Krafczyk, Siegbert. et al. Artificial neural network: A new diagnostic posturographic tool for disorders of stance. *Clinicial Neurophysiology*, 2006, Roč. 117, č. 8, s. 1692-1698. ISSN 1388-2457.

[10] *Visual Studio – Microsoft Developer Tools* [online]. [Cit. 29. 6. 2014]. Dostupné z: https://www.visualstudio.com/

[11] *Přehled TFS* [online]. [Cit. 29. 6. 2014]. Dostupné z: https://www.visualstudio.com/cs-cz/products/tfs-overview-vs.aspx

[12] *Co je Visual Studio Online?* [online]. [Cit. 29. 6. 2014]. Dostupné z: https://www.visualstudio.com/cs-cz/products/what-is-visual-studio-online-vs

[13] *Editor Guidelines extension* [online]. [Cit. 29. 6. 2014]. Dostupné z: https://visualstudiogallery.msdn.microsoft.com/da227a0b-0e31-4a11-8f6b-3a149cf2e459

[14] *Productivity Power Tools 2013 extension* [online]. [Cit. 29. 6. 2014]. Dostupné z: https://visualstudiogallery.msdn.microsoft.com/dbcb8670-889e-4a54-a226-a48a15e4cace?SRC=Home

[15] *ResX Resource Manager - Home* [online]. [Cit. 29. 6. 2014]. Dostupné z: http://resxresourcemanager.codeplex.com/

[16] *ReSharper :: The Most Intelligent Extension for Visual Studio* [online]. [Cit. 29. 6. 2014]. Dostupné z: https://www.jetbrains.com/resharper/

[17] *AForge.NET :: Framework* [online]. [Cit. 29. 6. 2014]. Dostupné z: http://www.aforgenet.com/framework/

[18] *Entity Framework - Home* [online]. [Cit. 29. 6. 2014]. Dostupné z: https://entityframework.codeplex.com/

[19] *Apache log4net – Apache log4net: Home* [online]. [Cit. 29. 6. 2014]. Dostupné z: https://logging.apache.org/log4net/

[20] Hastie, Trevor, Robert Tibshirani a J. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. 2. vyd. New York, NY: Springer, 2009, xxii, 745 s. ISBN 978-0-387-84858-7.

[21] *Google DeepMind* [online]. [Cit. 25. 12. 2014]. Dostupné z: http://deepmind.com/

[22] *Google's Secretive DeepMind Startup Unveils a „Neural Turing Machine"; / MIT Technology Review* [online]. [Cit. 25. 12. 2014]. Dostupné z: http://www.technologyreview.com/view/532156/googles-secretive-deepmind-startup-unveils-a-neural-turing-machine/

[23] Görlichová, Lucie. *Umělé neuronové sítě v lékařské diagnostice*. Brno, 2006. Diplomová práce. Masarykova univerzita v Brně. Přírodovědecká fakulta.

[24] Rojas, Raúl. *Neural networks: a systematic introduction*. 1. vyd. Berlin: Springer-Verlag, 1996, xx, 502 s. ISBN 3540605053.

[25] *Neuronové sítě* [online]. [Cit. 25. 12. 2014]. Dostupné z: http://ksvi.mff.cuni.cz/~mraz/nn/Neuronove_Site_Prednaska_Perceptron.pdf

[26] Informace poskytl PhDr.Ondřej Čakrt, Ph.D. Fakultní nemocnice v Motole 28. 7. 2015.

[27] Sameš, Martin et al. *Neurochirurgie: učebnice pro lékařské fakulty a postgraduální studium příbuzných oborů*. 1. vyd. Praha: Maxdorf, 2005, 127 s. ISBN 80-7345-072-0.

[28] *Medicabáze.cz - váš online lékařský slovník - Detail hesla* [online]. [Cit. 28. 7. 2015]. Dostupné z: http://www.medicabaze.cz/index.php?sec=term_detail&categId=22&cname=Neurologie&termId=2651&tname=Ataxie+spinocerebel%C3%A1rn%C3%AD&h=empty#jump

[29] *Accord.NET Machine Learning Framework* [online]. [Cit. 25. 12. 2014]. Dostupné z: http://accord-framework.net/

[30] *Math.NET Numerics* [online]. [Cit. 25. 12. 2014]. Dostupné z: http://numerics.mathdotnet.com/

[31] *MATLAB - The Language of Technical Computing* [online]. [Cit. 25. 12. 2014]. Dostupné z: http://www.mathworks.com/products/matlab/

[32] *fluentribbon/Fluent.Ribbon · GitHub* [online]. [Cit. 21. 3. 2015]. Dostupné z: https://github.com/fluentribbon/Fluent.Ribbon

[33] *MVVM Light Toolkit* [online]. [Cit. 21. 3. 2015]. Dostupné z: http://www.mvvmlight.net/

[34] *Xceed Extended WPF Toolkit Plus* [online]. [Cit. 21. 3. 2015]. Dostupné z: https://xceed.com/Extended_WPF_Toolkit_Intro.html

[35] Rocchi, L., L. Chiari a A. Cappello. Feature selection of stabilometric parameters based on principal component analysis. *Medical & Biological Engineering & Computing*, 2004, Roč. 42, č. 1, s. 71-79. ISSN 0140-0118.

[36] Schmid, Maurizio et al. The development of postural strategies in children: a factorial design study. *Journal of NeuroEngineering and Rehabilitation* [online]. 2005, 2(1) [Cit. 25. 12. 2014]. DOI 10.1186/1743-0003-2-29.

[37] *Wolfram Mathematica: Definitive System for Modern Technical Computing* [online]. [Cit. 25. 12. 2014]. Dostupné z: http://www.wolfram.com/mathematica/

[38] *Classify—Wolfram Language Documentation* [online]. [Cit. 25. 12. 2014]. Dostupné z: https://reference.wolfram.com/language/ref/Classify.html

[39] *ClassifierMeasurements—Wolfram Language Documentation* [online]. [Cit. 25. 12. 2014]. Dostupné z: https://reference.wolfram.com/language/ref/ClassifierMeasurements.html

[40] Sokolova, Marina, Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 2009, Roč. 45, č. 6, s. 427-437. ISSN 0306-4573.

[41] McConnell, Steve. *Dokonalý kód: umění programování a techniky tvorby software*. 1. vyd. Brno: Computer Press, 2005, 894 s. ISBN 80-251-0849-x.

[42] Martin, Robert C. *Čistý kód: [návrhové vzory, refaktorování, testování a další techniky agilního programování]*. 1. vyd. Brno: Computer Press, 2009, 423 s. ISBN 978-80-251-2285-3.

[43] Larman, Craig. *Agile and iterative development: a manager's guide*. Boston: Addison-Wesley, 2004, xiv, 342 s. ISBN 0-13-111155-8.

[44] Beck, Kent. *Programování řízené testy*. 1. vyd. Praha: Grada, 2004, 203 s. Moderní programování. ISBN 80-247-0901-5.

[45] *THE MODEL-VIEW-VIEWMODEL (MVVM) DESIGN PATTERN FOR WPF* [online]. [Cit. 29. 6. 2014]. Dostupné z: https://msdn.microsoft.com/en-us/magazine/dd419663.aspx

[46] Duvall, Paul M., Steve Matyas a Andrew Glover. *Continuous integration: improving software quality and reducing risk*. Upper Saddle River, NJ: Addison-Wesley, 2007, xxxiii, 283 s. ISBN 0-321-33638-0.

Seznam obrázků

Seznam ukázek kódu

Seznam použitých zkratek

2. LF UK 2. lékařská fakulta Univerity Karlovy v Praze.

CI Continuous Integration.

CNS Centrální nervová soustava.

COP Působiště reakční síly („center of pressure“).

FFT Rychlá Fourierova transformace (Fast Fourier transform).

GUI Grafické uživatelské rozhraní (graphical user interface).

IDE Vývojové prostředí.

MFF UK Matematicko-fyzikální fakulta Univerity Karlovy v Praze.

MVC Model View Controller pattern.

MVVM Model View ViewModel pattern.

NS Umělá neuronová síť.

SKG Statokinesigram.

SPS Posturografická plošina Synapsys Posturography System.

SVM Support vector machines.

TDD Programování řízené testy (Test-driven development).

TFS Team Foundation Server.

VSO Visual Studio Online.

VS Visual Studio.

WCF Windows Communication Foundation.

WPF Windows Presentation Foundation.

A. DVD

A.1 Obsah přiloženého DVD

Artificial Neural Networks - Posturographic diagnostic tool Složka se soubory využívánými při experimentu, vizme kapitolu 7.

- **Exported results** Složka s vyexportovanými předběžnými verzemi obrázků do odborného článku v časopise Česká a slovenská neurologie a neurochirurgie.
- **Measurements** Složka s naměřenými daty pro jednotlivé pacienty využíváné v experimentu i v experimentálím testování aplikace.

Documentation Složka s vygenerovanou programátorskou dokumentací pro solution hlavního projektu StabilometricDataAnalyzerAndVisualizer.

MATLAB Compiler Složka se zkompilovanými MATLAB skripty do formy dll knihoven.

- **cpower 32bit** Složka s 32 bitovou verzí dll knihovny, včetně MATLAB Runtime Library v příslušné verzi.
- **cpower 64 bit** Složka s 64 bitovou verzí dll knihovny, včetně MATLAB Runtime Library v příslušné verzi.

PosturographyMeasurementsUtilities Složka se solution vedlejších pomocných aplikací.

- **ANNPatientsClassifier** Složka se zdrojovými kódy projektu pro výpočet parametrů a klasifikaci pacientů dle naměřených dat.
- **ANNPatientsClassifiersCreator** Složka se zdrojovými kódy projektu pro tvorbu NS.
- **DBFMeasurementsToCSharpCode** Složka se zdrojovými kódy projektu pro úpravu naměřených dat do formy C# kódu.
- **packages** Složka s jednotlivými použitými knihovnami, vizme popisy v jednotlivých kapitolách.

StabilometricDataAnalyzerAndVisualizer Složka se solution hlavní aplikace Stabilometric Data Analyzer and Visualizer. Pro popisy jednotlivých projektů vizme příslušné kapitoly.

- **.nuget** Složka se soubory pro obnovování NuGet balíčků při neúspěšném updatování.
- **packages** Složka s jednotlivými použitými knihovnami, vizme popisy v jednotlivých kapitolách.
- **StabilometricDataAnalyzerAndVisualizer.Analyzers** Složka se zdrojovými kódy daného projektu.
- **StabilometricDataAnalyzerAndVisualizer.AnalyzersTests** Složka se zdrojovými kódy daného projektu.

StabilometricDataAnalyzerAndVisualizer.Data Složka se zdrojovými kódy daného projektu.

StabilometricDataAnalyzerAndVisualizer.DataService Složka se zdrojovými kódy daného projektu.

StabilometricDataAnalyzerAndVisualizer.DataServiceHelper Složka se zdrojovými kódy daného projektu.

StabilometricDataAnalyzerAndVisualizer.DataServiceTests Složka se zdrojovými kódy daného projektu.

StabilometricDataAnalyzerAndVisualizer.DataTests Složka se zdrojovými kódy daného projektu.

StabilometricDataAnalyzerAndVisualizer.Posturography32bit Složka se zdrojovými kódy daného projektu.

StabilometricDataAnalyzerAndVisualizer.Posturography32Tests Složka se zdrojovými kódy daného projektu.

StabilometricDataAnalyzerAndVisualizer.Posturography64bit Složka se zdrojovými kódy daného projektu.

StabilometricDataAnalyzerAndVisualizer.Posturography64Tests Složka se zdrojovými kódy daného projektu.

StabilometricDataAnalyzerAndVisualizer.UI Složka se zdrojovými kódy daného projektu.

Text Složka s vlastním textem bakalářské práce.

Video Složka s videi experimentálního testování aplikace.

B. Použitá programátorská paradigmata a styl kódu

B.1 Použitá programátorská paradigmata

Pro plynulý vývoj aplikace byla použita následující programátorská paradigmata. Ta nejen ulehčují vývoj, ale snadno studentům — při rozšíření této práce v rámci dalších předmětů — představí některé ze základních postupů, technik, patternů, atd. používaných v praxi. Uváděné poznatky a poznámky vycházejí z několikaměsíční praxe ve firmě mineus)(s. r. o., pod vedením Mgr. Martina Suchana.

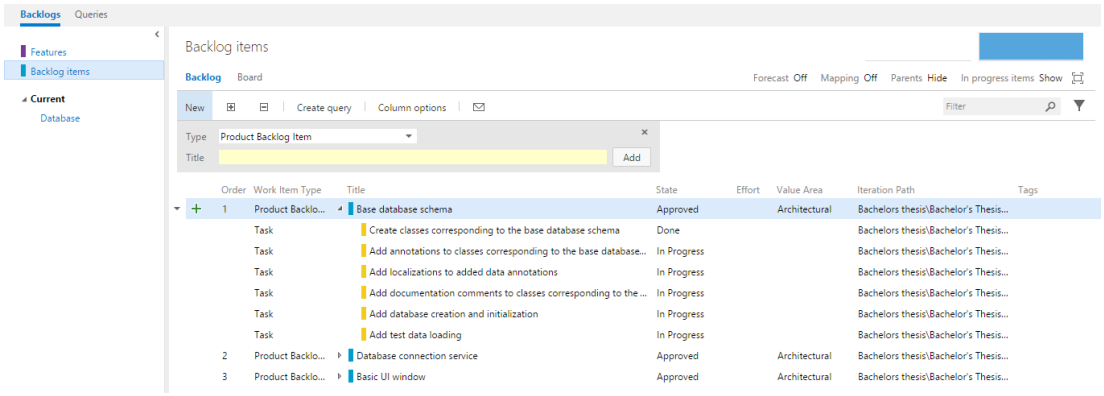
B.1.1 Agilní metodiky vývoje software [41]

Jedná se o různé metody, či skupiny metod určené pro vývoj softwaru. Jsou založeny na inkrementálním — stavíme na postupném vývoji a rozšiřování jednotlivé funkcionality programu od jednoduššího návrhu ke složitějšímu, nikoliv rovnou na tvorbě komplexních částí programu — a iterativním — vytvoříme část programu, dodáme zákazníkovi, získáme zpětnou vazbu, aplikujeme ji a poté znovu, dokud není software hotov — vývoji [43]. Tyto techniky jsou vhodnější především pro práci v týmu, ale pokud si je osvojí i programátor – jednotlivec, snadněji je poté aplikuje v praxi.

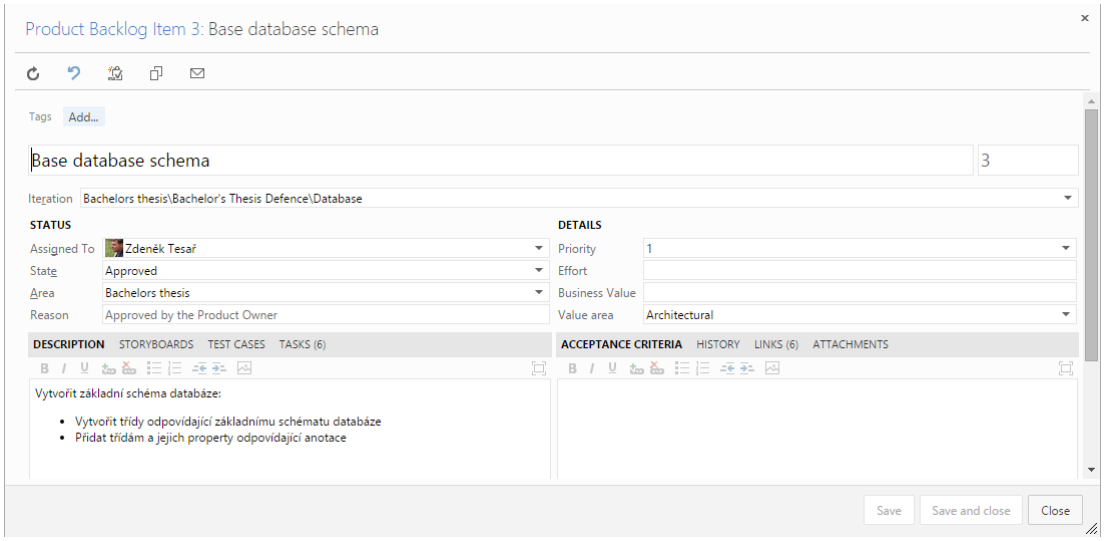
B.1.1.1 SCRUM [43]

Jedna z agilních metodik vývoje software. Klíčovým principem SCRUMu je uvědomění si, že zákazník často mění své požadavky. Ty nemohou být tak snadno reflektovány v tradičním dlouhodobě plánovaném modelu.

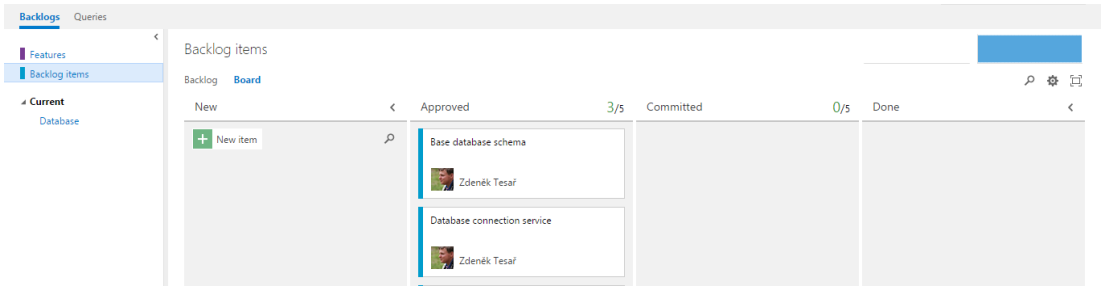
SCRUM nám pomáhá definováním rolí v týmu. Ty nás ovšem nebudou v rámci práce zajímat. Zaměříme se především na rozdělení práce. Základní jednotkou je tzv. sprint. Jde o dobu v rozmezí jednoho až čtyř týdnů. Standardně se jedná o dva, při psaní této práce byl používán měsíc. Pro sprint je vždy definován backlog, který určuje jeho hlavní cíl, viz obrázky B.1, B.2 a B.3. V případě rozsáhlejších projektů s více programátory může mít jeden sprit více jak jeden backlog. Příkladem může být vytvoření základního databázového schématu – vytvoření modelu databáze, implementace modelu v jazyce C# za použití Entity Frameworku, přidání data annotations, vytvoření základního konzolového rozhraní. Daný backlog je poté rozdělen na jednotlivé úkoly – tasky. Ty mají svůj popis, prioritu, předpokládanou časovou dotaci a jsou přiřazeny jednotlivým programátorům, jak můžeme vidět na obrázku B.4.



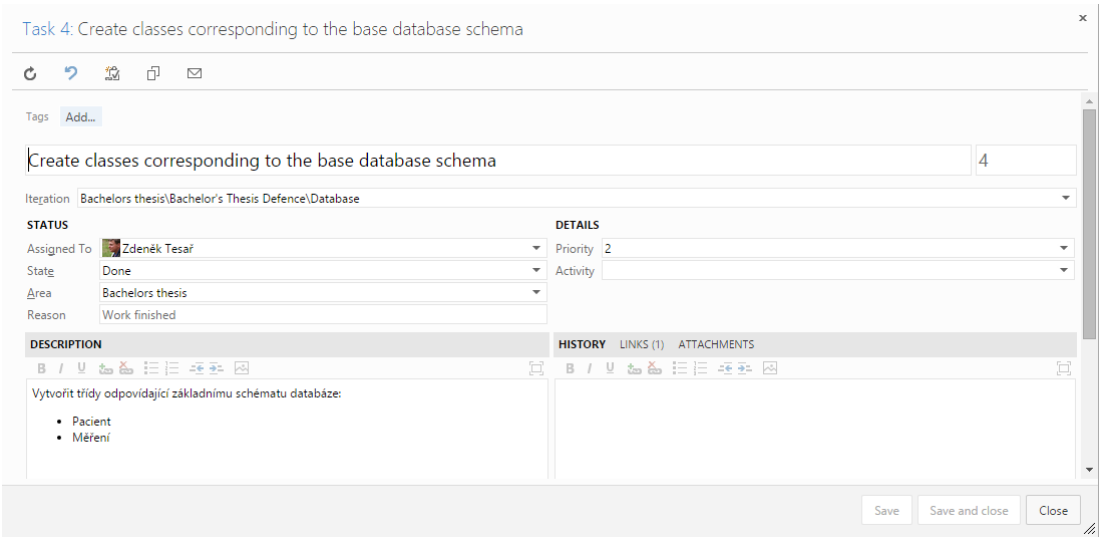
Obrázek B.1: Seznam backlogů včetně přiřazených úkolů.



Obrázek B.2: Detail backlogu.



Obrázek B.3: Tabule se seznamem backlogů aktuálního sprintu, včetně fáze, ve které se nacházejí.



Obrázek B.4: Detail úkolu.

B.1.1.2 Defenzivní programování

Základem defenzivního programování je předpoklad, že pokud uživatel zadá špatná data, či dále v procesu jejich zpracování dojde ke změně na špatná, náš program tím nebude poškozen a nedojde k ovlivnění jeho funkčnosti. Jde o souhrn několika postupů – ochrana programu před nevalidními vstupy, asserty, techniky odchytu a zpracování chyb a další. [41] V našem kódu půjde převážně o kontrolu validity vstupů spolu s odchytem a zpracováním chyb, viz ukázka kódu B.1.

```
/// <summary>
/// Hlavni metoda pro parsovani vsech namerenych dat z
   ↪ .dbf souboru.
/// </summary>
/// <param name="measurement">Entita mereni s
   ↪ naparsovanymi veskerymi udaji.</param>
/// <param name="dateOfMeasurement">Datum a~cas
   ↪ mereni.</param>
/// <param name="measurementDBFDirPath">Cesta k adresari s
   ↪ jednotlivymi .dbf soubory.</param>
/// <returns>Informace, jaka nastala vyjimka (i
   ↪ zadna).</returns>
public static DataException GetMeasurementData(
    out Measurement measurement, DateTime
       ↪ dateOfMeasurement, string measurementDBFDirPath)
{
    measurement = null;
    // ReSharper disable InconsistentNaming
    Results _Free1Res, _Free2Res;
    Channels _Free1YF, _Free1YO, _Free2YF, _Free2YO;

    try
    {
        _Free1Res =
```

```
            ↪ getMeasurementResultsData(measurementDBFDirPath
            ↪ + free1ResDBF);
    }
    catch (Exception ex)
    {
        log.Error("_Free1Res =
            ↪ getMeasurementResultsData(measurementDBFDirPath
            ↪ + free1ResDBF);", ex);

        if (ex is ArgumentNullException || ex is
            ↪ ArgumentException)
            return DataException.ArgumentException;
        if (ex is NotSupportedException)
            return DataException.NotSupportedException;
        if (ex is UnauthorizedAccessException)
            return DataException
                .Free1ResUnauthorizedAccessException;
        if (ex is PathTooLongException)
            return
                ↪ DataException.Free1ResPathTooLongException;
        if (ex is DirectoryNotFoundException)
            return DataException
                .Free1ResDirectoryNotFoundException;
        if (ex is FileNotFoundException)
            return
                ↪ DataException.Free1ResFileNotFoundException;
        if (ex is OutOfMemoryException || ex is
            ↪ IOException)
            return DataException.Free1ResIOException;
        if (ex is FormatException || ex is
            ↪ OverflowException)
            return DataException.Free1ResFormatException;

        return DataException.Exception;
    }

    ...
```

Ukázka kódu B.1: Metoda s odchytem a zpracováním chyb.

B.1.1.3 Programování řízené testy (Test-driven development) [44]

Z vlastní zkušenosti víme, že odhalování chyb po dokončení většího celku programu může způsobit mnohdy nemalé problémy. Z toho plyne, že je dobré testovat menší kusy kódu. Ruční testování je ovšem nepřijatelné z důvodu vysoké časové náročnosti. Automatizované testy se však po dokončení kusu kódu programátorům psát nechtějí. Vyplývá tedy, že nejlepší je napsat automatický test ještě před napsáním vlastního kódu. Pomůže nám to jak v prevenci později nalezených chyb, tak i v tvrobě vhodné struktury a rozhraní aplikace, včetně jednotlivých metod, jenž ji tvoří. V neposlední řadě také nejsme ovlivněni vlastní implementací.

Použití programování řízeného testy, dále jen TDD, je následující:

1. Napíšeme test (sadu testů) na část kódu (metodu), jíž se chystáme implementovat.
2. Spustíme test, který by měl selhat.
3. Implementujeme danou funkčnost.
4. Opět spustíme test, jenž by měl v případě správné implementace projít. Pokud neprojde, s pomocí testu a debugování odhalíme chybu, opravíme ji a opakujeme krok č. 4.
5. Refaktorace kódu. Pokud tento krok aplikujeme, vždy se po něm zopakujeme krok č. 4 pro ověření, zdali jsme program refaktorací nerozbili.

Testy by měly být co nejvíce konkrétní – testovat konkrétní vstup a očekávat konkrétní výstup. Základem jsou tzv. unit testy (jednotkové testy), které testují právě jednu jednotku programu – metodu, viz ukázka kódu B.2. Ukázku testování kódu můžeme vidět na obrázku B.5.

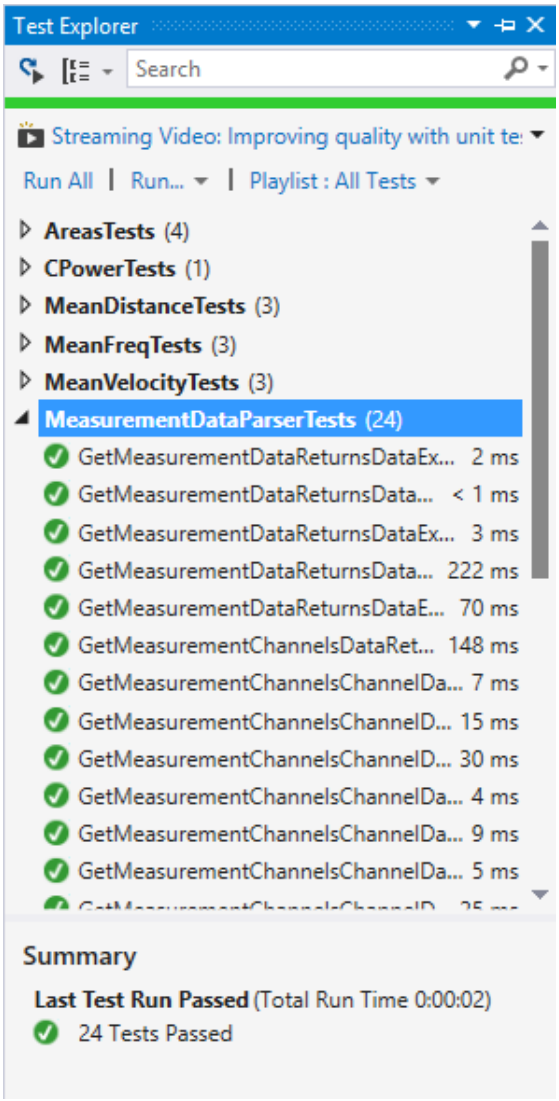
V naší aplikaci se budeme snažit dodržovat tento postup pouze u částí bez GUI.

```
/// <summary>
/// Testovaci metoda, zdali metoda GetMeasurementData
   ↪ vraci spravne nactene mereni a~nevraci zadnou chybu.
/// </summary>
[TestMethod]
public void
   ↪ GetMeasurementDataReturnsDataExceptionNoneTest()
{
    // Priprava dat pro testovani.
    Measurement measurement;

    // Beh vlastni testovane metody.
    DataException dataException =
        MeasurementDataParser.GetMeasurementData(out
           ↪ measurement, dateOfMeasurement, DBFsDirPath);

    // Overeni spravnosti vysledku testovane metody.
    Assert.IsNotNull(dataException);
    Assert.AreEqual(dataException, DataException.None);
    Assert.IsNotNull(measurement);
    Assert.AreEqual(measurement.DateAndTime,
       ↪ dateOfMeasurement);
}
```

Ukázka kódu B.2: Metoda s odchytem a zpracováním chyb.



Obrázek B.5: Probíhající unit testy.

B.1.1.4 Model View ViewModel pattern [45]

Model View ViewModel, dále jen MVVM, je architektonický pattern pro vývoj softwaru. Vychází z Model View Controller patternu, dále jen MVC. Byl vyvinut Microsoftem pro zjednodušení událostmi řízeného programování u uživatelských rozhraní. Pomáhá oddělení vývoje GUI od vývoje vlastní logiky aplikace.

Jak již napovídá název, pattern se skládá ze tří základních částí:

Model Vztahuje se ke skutečnému stavu obsahu (objektově orientovaný přístup), nebo pro přístup k datové vrstvě, která reprezentuje obsah (datacentrický přístup).

View Jedná se o vlastní GUI.

ViewModel Abstrakce view, která zpřístupňuje veřejné vlastnosti a příkazy. Na rozdíl od controlleru v MVC má MVVM tzv. binder. Binder zprostředkovává mezi view a daty. ViewModel je popisován jako stav dat v modelu.

Cílem tohoto patternu je minimalizace kódu, který přímo manipuluje s GUI. To umožňuje snazší manipulaci a úpravy GUI v pozdější fázi vývoje. V případě vývoje ve větším vývojařském týmu také usnaďnuje spolupráci vývojářů a designerů, kteří mohou přímo vytvářet uživatelské rozhraní, například pomocí uživatelsky přívětivého programu Blend for Visual Studio, který snadno přeloží návrh designerů do XAMLu (značkovací kód pro tvorbu GUI ve Windows Presentation Foundation, dále jen WPF).

B.1.2 Možnosti dalšího rozvoje

B.1.2.1 Coded UI testy

V rámci aplikace zatím testujeme pouze kód, který se netýká GUI. Proto se jako první možnost rozšíření nabízí právě implementace těchto testů. Pro naše budoucí potřeby nebude třeba další nástroj, neboť Visual Studio, dále jen VS, má již podporu pro tvorbu Coded UI testů integrovanou.

B.1.2.2 Průběžná integrace (Continuous Integration) [46]

Průběžná integrace, dále jen CI, slouží pro pro urychlení vývoje a spolupráci vývojových týmů. Jedná se o jednu z metodik extrémního programování.

Základní principy CI:

- Použití centrálního úložiště kódu, tzv. repository. To zajistí, že všichni členové týmu pracují se stejným kódem, který mezi sebou mohou snadněji sdílet. Součástí repository jsou funkce jako verzování, historie souborů, spojování kódu, tzv. mergování a další.
- Automatizace sestavení aplikace, tzv. buildu.
- Automatizace testování aplikace.
- Kontrola kvality kódu – především kontrola duplicity kódu, statických chyb a dalších.
- Zpřístupnění poslední verze aplikace.

Průběh CI:

vývoj aplikace → vložení kódu do repository → spuštění buildu → automatické testy → statická kontrola kódu → kompilace nové verze → odeslání hotového buildu

Důvody pro nasazení CI jsou převážně vysoká chybovost kódu (tu ovšem může řešit právě TDD), časté sestavování nových buildů a vysoká výpočetní či časová náročnost této činnosti, špatné vzájemné sdílení kódu a dohled nad ním. Výhody CI snadno vyplývají z výše uvedeného textu.

B.2 Styl kódu (Code-style)

U kódu se snažíme zachovávat při jeho formátování logickou i sémantickou strukturu, jak se uvádí v knize Dokonalý kód [41], a dbáme na ni i při aplikaci následujících pravidel. Všechen kód, kromě komentářů, je psán anglicky.

- Všechny třídy, struktury, výčtové hodnoty,... jsou vždy v samostatném souboru.
- Kód je psán „CamelCase“ stylem.
- Názvy tříd, struktur a výčtových hodnot začínají velkým písmenem.
- Názvy veřejných metod a veřejných property a field začínají velkým písmenem, názvy privátních metod a privátních property a field malým písmenem.
- Pokud je v názvu metody užita zkratka, tak je napsána celá velkými, nebo malými písmeny, dle prvního písmene.
- Názvy konstant vždy začínají velký písmenem.
- Všechny třídy, struktury, výčtové typy a metod jsou okomentovány standardními XML komentáři exportovanými do programátorské dokumentace, uvozené `///`.
- Komentáře jsou psány v češtině. Výjimkou jsou převzaté komentáře z cizího kódu (např. MATLAB skripty Ing. Fundy), kde komentář zůstává v původním formátu.
- Pokud se jedná o řádkový komentář, vždy je na samostatném řádku před řádkem, kterého se týká.
- Maximální délka řádku kódu je 125 znaků. Pokud je tato délka přesažena, zalamuje se kód v následujícím pořadí:
  - Pokud se jedná o přiřazení do proměnné, kód se nejprve zalamuje za znakem `=`.
  - Pokud se jedná o navrácení hodnoty z metody, kód se nejprve zalamuje za klíčovým slovem `return`.
  - Pokud se jedná o „složený výraz“, a i po zalomení dle výše uvedených pravidel stále přesahuje povolenou hranici, použijeme následující pravidla, v závislosti na znaku, který je více vlevo (prioritu má vždy výše položená odrážka):
    * Před znakem `.`, například při použití LINQu.
    * Za znakem `(`, například u volání metod s parametry.
    * U „aritmetických výrazů“, například `+`, `-`, `>`, `==`, `>>`,... a logických spojek, například `&&` nebo `||`, před tímto výrazem či spojkou.
    * Pokud se jedná o textový řetězec, tak jej rozdělíme na dvě části spojené znakem `+` a řádek zalamujeme za tímto znakem. Výjimku tvoří `const string`, u kterého řetězec nedělíme, maximálně aplikujeme pravidlo s přiřazením do proměnné.

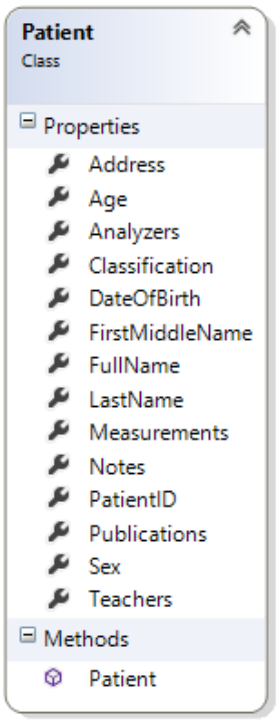
B.2.1 Možnosti dalšího rozšíření

V případě rozšiřování programu v rámci dalších předmětů by bylo vhodné zavedení bilinguálních komentářů (čeština + angličtina) pro další snadnou čitelnost i pro studenty v rámci Erasmu či dalších programů pro mezinárodní výměnu studentů.

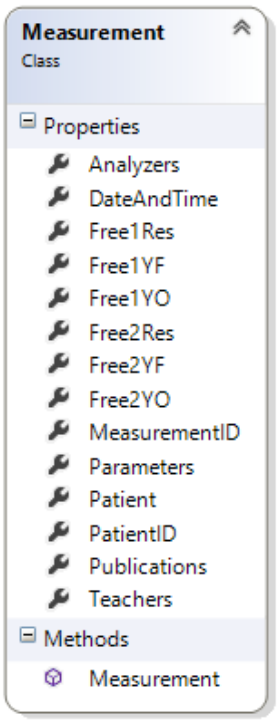
C. Diagramy a kódové mapy

C.1 Projekt StabilometricDataAnalyzerAndVisualizer.Data

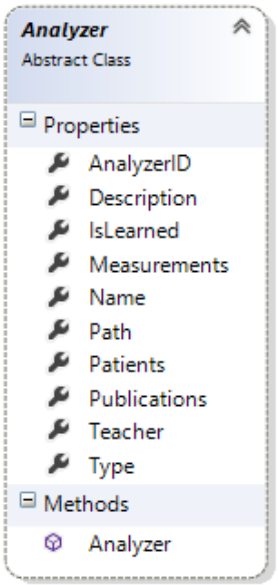
C.1.1 Ukázkové diagramy databázových entit



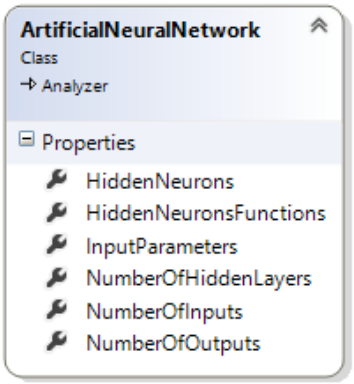
(a) Entita pacienta.



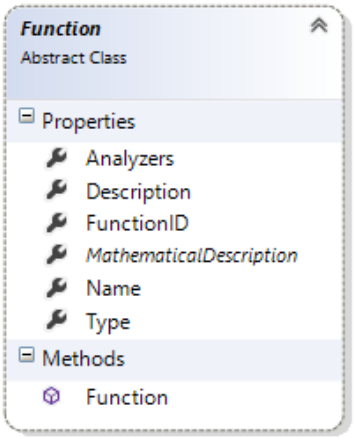
(b) Entita měření.



(c) Entita analyzéru / klasifikátoru.

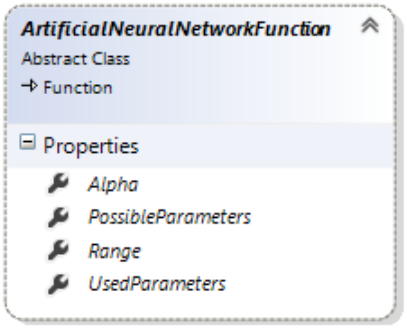


(d) Entita NS klasifikátoru.

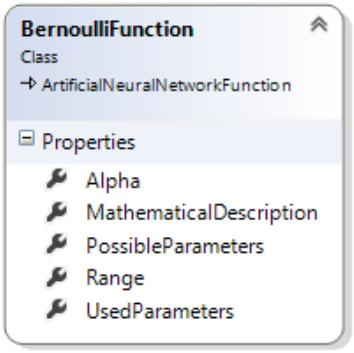


(e) Entita funkce.

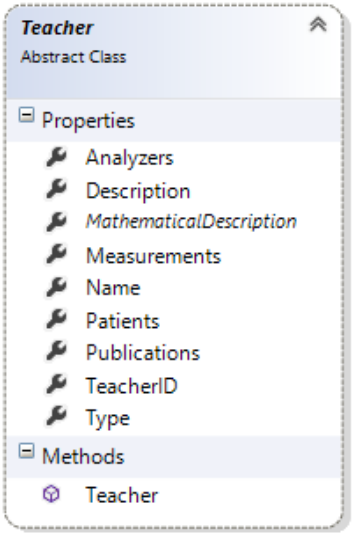
Obrázek C.1: Diagramy jednotlivých databázových entit 1/2.



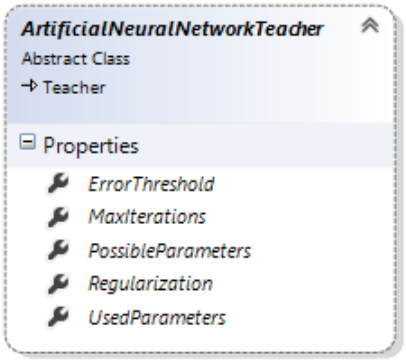
(a) Entita NS funkce.



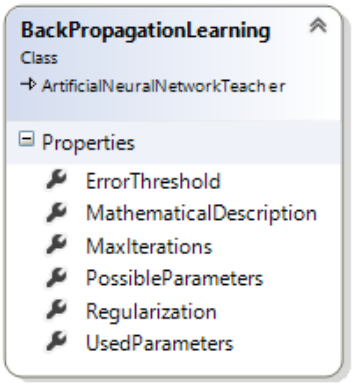
(b) Entita funkce děděné od NS funkce – Bernoulliho funkce.



(c) Entita učitele.

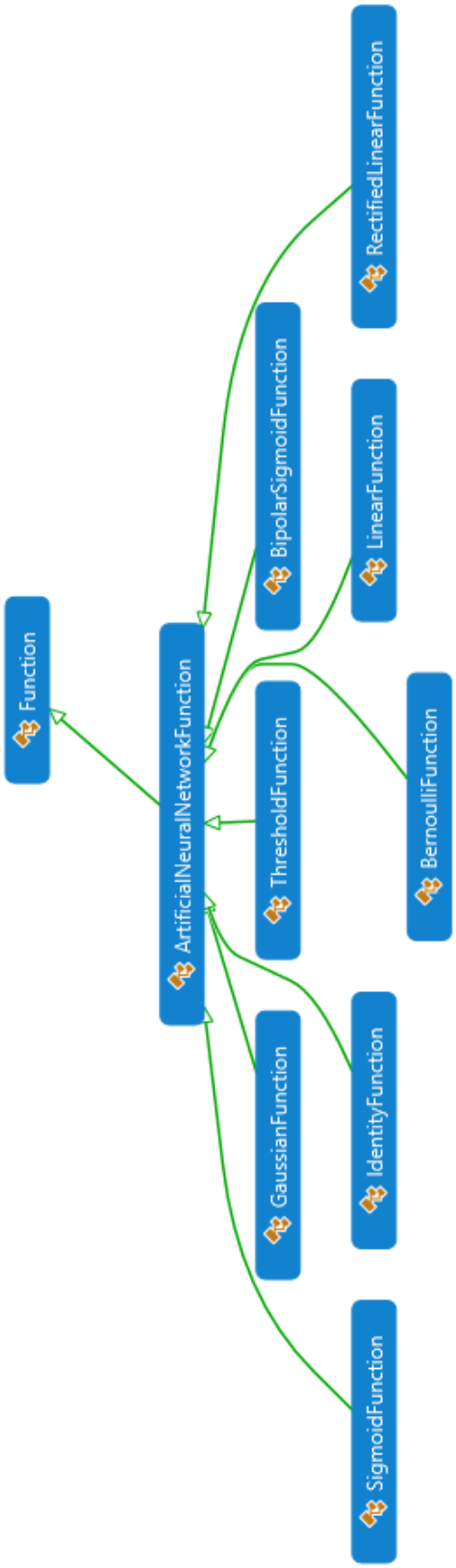


(d) Entita NS učitele.

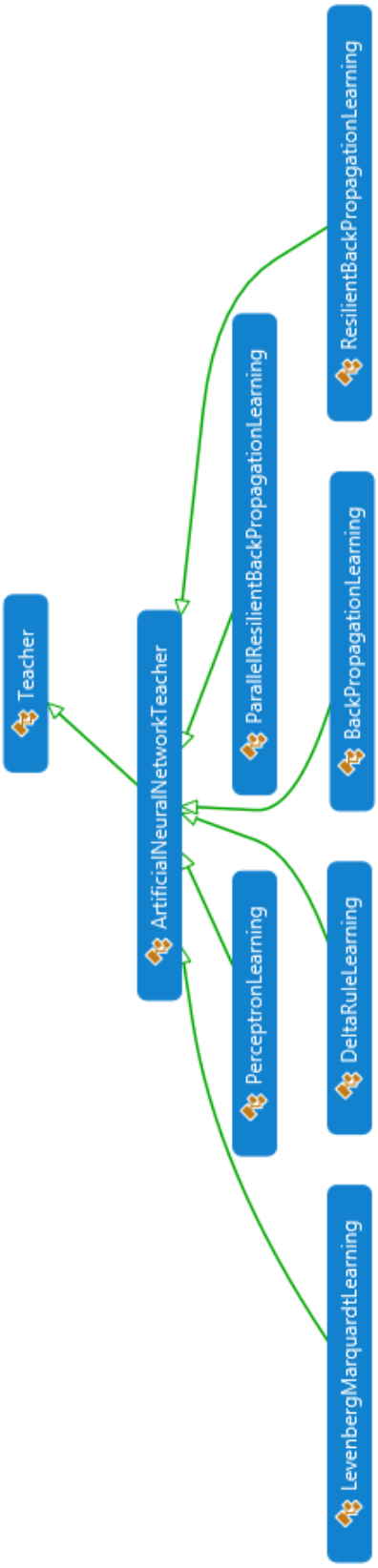


(e) Entita učitele děděného od NS učitele – algoritmus zpětné propagace chyby.

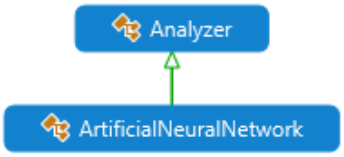
Obrázek C.2: Diagramy jednotlivých databázových entit 2/2.



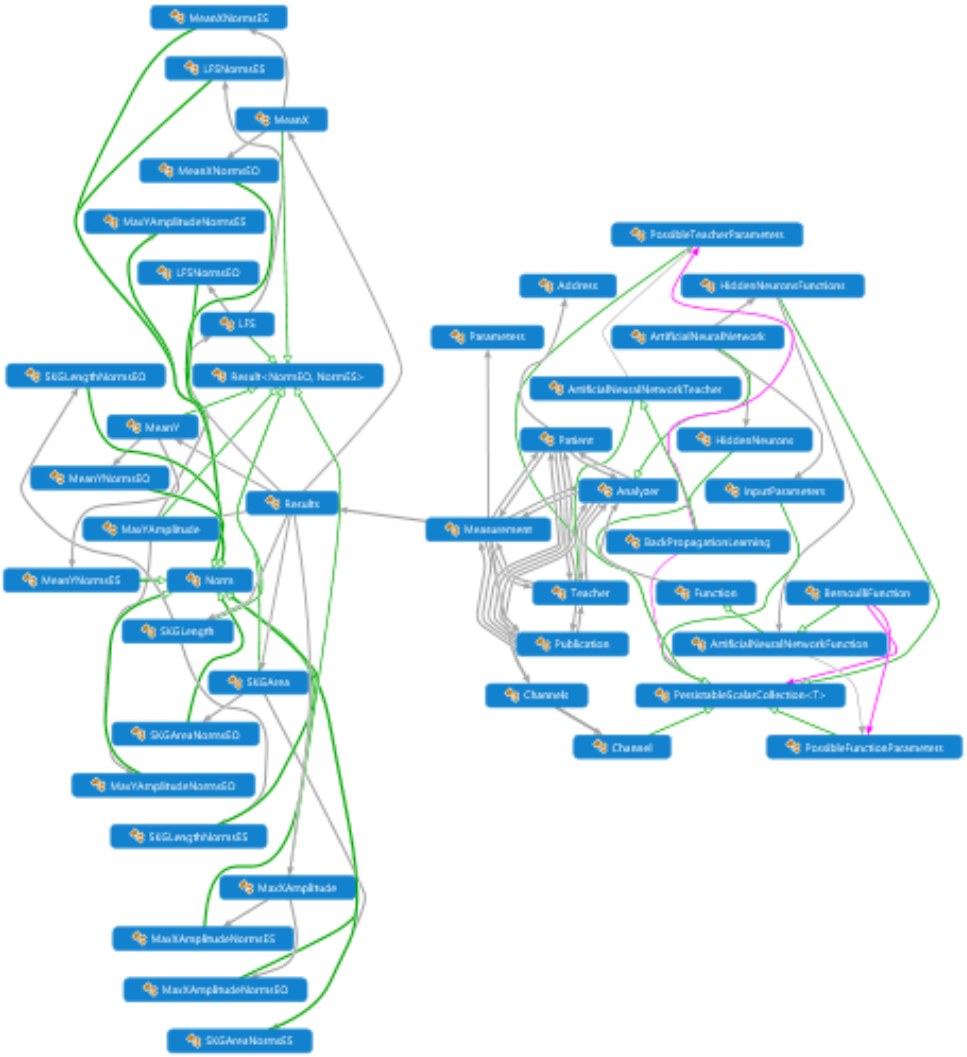
Obrázek C.3: Kódová mapa entity funkce včetně děděných entit.



Obrázek C.4: Kódová mapa entity učitele včetně děděných entit.



Obrázek C.5: Kódová mapa entity analyzéru včetně děděných entit.



Obrázek C.6: Kódová mapa vztahů mezi entitami a komplexními typy.