Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Integrace metod strojového učení do nástroje pro zpracování elektrofyziologických dat

Plzeň 2020 Bc. Filip Kupilík

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 7. srpna 2020

Bc. Filip Kupilík

Poděkování

Rád bych poděkoval Ing. Romanu Moučkovi, Ph.D. za trpělivost, ochotu, pomoc a cenné rady poskytnuté při vedení práce.

Abstract

The goal of the thesis is the integration of machine learning methods into a chosen tool for processing standardized electrophysiological data. The thesis maps current tools and analyzes their current possibilities of using machine learning methods. The classification machine learning methods, which are used in the neuroinformatics laboratory at the Faculty of Applied Sciences in the University of West Bohemia, are then integrated into the chosen tool. The functionality of the solution is verified by the implementation of the experiment with real dataset. The source code and user manual are shared in a public repository.

Abstrakt

Cílem této práce je integrace metod strojového učení do vybraného nástroje pro zpracování standardizovaných elektrofyziologických dat. V práci jsou aktuální nástroje zmapovány a jsou zanalyzovány jejich současné možnosti využití metod strojového učení. Do vybraného nástroje jsou poté integrovány klasifikační metody strojového učení, jež se používají v neuroinformatické laboratoři na FAV ZČU. Proveditelnost zvoleného řešení je ověřena implementací experimentu s reálnými daty. Výsledný kód a uživatelský manuál jsou sdíleny ve veřejně dostupném repositáři.

Obsah

1 Úvod

Analýzou EEG signálu se zabývá celá řada výzkumníků z medicínských i informatických oblastí. To je jedním z důvodů existence velkého množství analytických nástrojů, které jsou implementované v různých jazycích, např. v MATLABu či v Pythonu.

Klasickým postupem analýzy EEG signálu je jeho předzpracování, extrakce příznaků a klasifikace. Analytické nástroje se mezi sebou liší nejen jazykem, v němž jsou implementovány, ale také možnostmi, které pro dané kroky analýzy poskytují. Klasifikační metody strojového učení většinou nejsou přímo součástí daného nástroje, ale jsou spíše výsadou specializovaných knihoven třetích stran. Schopnosti integrace těchto metod se napříč nástroji různí.

Cílem této práce je tyto nástroje zmapovat a zanalyzovat je z hlediska několika kritérií, přičemž tím hlavním budou jejich současné možnosti využití metod strojového učení pro analýzu EEG dat. Na základě výsledků analýzy bude vybrán nejvhodnější nástroj a do něj budou integrovány klasifikační metody strojového učení poskytované specializovanými knihovnami, jež se používají v neuroinformatické laboratoři na FAV ZČU. Proveditelnost zvolené integrace bude dále ověřena implementací experimentu s reálnými daty.

Výsledný kód a uživatelský manuál budou sdílené ve veřejně dostupném repositáři tak, aby mohly být užitečné globální komunitě.

2 EEG signál a datové standardy

Elektroencefalografie (EEG) se používá pro záznam elektrické aktivity mozku. EEG signál se získává pomocí elektrod, které měří hodnoty elektrického potenciálu generované mozkovou aktivitou na povrchu hlavy. Typickým úkolem analýzy EEG signálu je jeho rozpoznávání. Můžeme v něm detekovat například záchvaty epilepsie, emoce, motorické pohyby, mapovat průběh spánku nebo klasifikovat reakce vyvolané nějakou událostí. Při klasifikaci reakcí na podnět je měřený subjekt vizuálně nebo sluchově stimulován nějakou událostí, kterou může být například promítnutí čísla, a poté je zkoumána bezprostřední reakce jeho mozku na tento stimul a zjišťováno, zda na něj nějakým způsobem reagoval či nikoliv. Jedná se tedy o evokované potenciály (ERP - Event-related potentials).

Při tomto postupu jsou poté ze signálu extrahovány epochy. To jsou krátké úseky signálu, jež jsou spjaté s daným stimulem, a signál se získává z jeho bezprostředního okolí daného nějakým časovým úsekem před stimulem a po něm. Někdy se v signálu epochy může přibližně 300 ms po stimulu objevit pozitivní vlna nazývaná P300, která může být vnímána jako specifická reakce na nějaký podnět, v našem případě na stimul.

EEG bohužel trpí několika omezeními, která brání v jeho účinné analýze a zpracování. Jedním z těchto omezení je, že EEG signál je často zašuměn různými zdroji z okolního prostředí, např. signálem z elektrické sítě nebo signály vytvářenými lidským tělem jako například mrkání nebo různé svalové reakce, které se souhrnně nazývají artefakty. Extrakce epoch a odstranění šumu je prvním typickým krokem analýzy EEG signálu zvaným předzpracování. Jednou z používaných metod předzpracování je filtrace, jejímž základním typem je odstranění frekvenčních pásem ze signálu pomocí dolní propusti, která propustí frekvence nižší než daná hodnota, typicky 30 - 50 Hz, a pomocí horní propusti, která propustí frekvence vyšší než daná hodnota, typicky 0.05 - 0.1 Hz. Pro odstranění artefaktů vyprodukovaných lidským tělem lze použít buď základní metodu odstranění dle amplitudy, nebo existuje několik dalších ověřených metod, jako například pomocí analýzy nezávislých komponent.

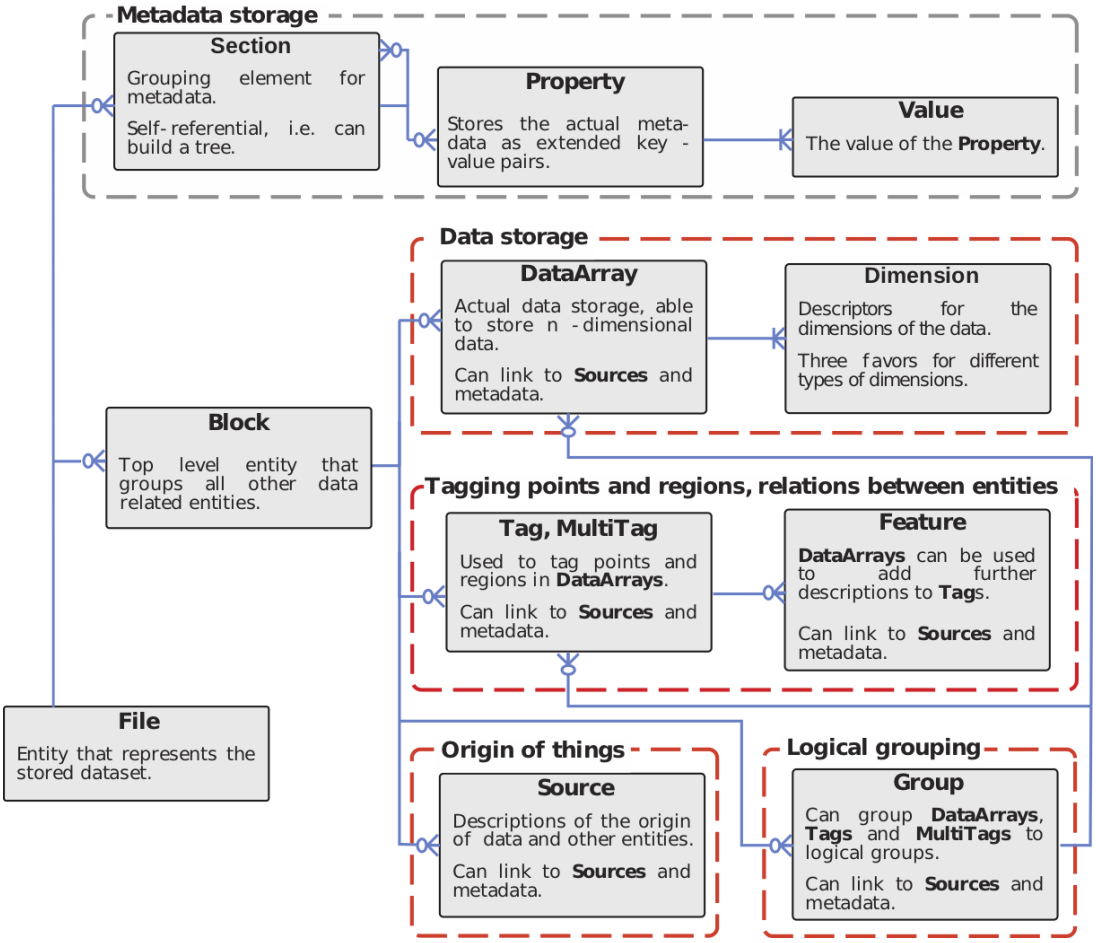
Následujícím krokem analýzy EEG signálu je extrakce příznaků. Technicky příznak představuje nějakou rozlišující vlastnost získanou ze signálu. Cílem extrakce příznaků je zjednodušení vstupní datové množiny a přitom pokud možno zachování informací, které signál nese. Ve zpracování EEG signálu se

většinou jedná o snížení dimenzionality vstupních dat pro zvýšení výkonu zejména standardních klasifikátorů, které s příznaky dosahují zpravidla lepších výsledků. Posledním krokem je klasifikace zpracovaného signálu často do předem definovaných tříd. Při trénování s učitelem se typicky jedná o úlohu natrénování klasifikátoru na trénovací množině epoch s označenými třídami, a poté testování modelu s pomocí nové, neviděné množiny.

Pro ukládání EEG dat bylo vytvořeno velké množství datových formátů a variace jejich používání napříč neuroinformatickými laboratořemi je velká. Není ojedinělé, že si laboratoře implementují vlastní datové formáty pro prováděné experimenty, nebo volba používaného datového formátu v laboratoři může záviset na použitém snímacím HW, jako například ukládání do BrainVision formátu v neuroinformatické laboratoři na FAV ZČU. Tato rozmanitost napříč formáty komplikuje sdílení dat mezi laboratořemi, protože v tuto chvíli neexistuje žádný jednotný uznaný standard pro ukládání dat. V současné době o tento status bojují datové formáty `NIX`, `BIDS` a `NWB`, které budou blíže popsány v následujících podkapitolách.

2.1 `NIX`

Neuroscience information exchange format (`NIX`) je vyvíjen skupinou *German Neuroinformatics Node* se sídlem v Mnichově, která je členem *International Neuroinformatics Coordination Facility*. `NIX` vyvinul minimalistický, obecný datový model, který se skládá z co nejmenšího počtu entit, a přitom dokáže bez ztráty informací reprezentovat data uložená v jiných široce používaných formátech nebo modelech, jakým je například `Neo` (popsán v 4.4). Díky svému vysoce obecnému přístupu je datový model `NIX` schopen uchovávat i obrazové a jiné druhy dat používaných v neuroinformatických laboratořích. Formát souborů pro ukládání datového modelu je založený na struktuře `HDF5`. Pro manipulaci s ním poskytuje `NIX` knihovny v jazyce C++ a Python.



Obrázek 2.1: Datový model formátu `NIX` [6]

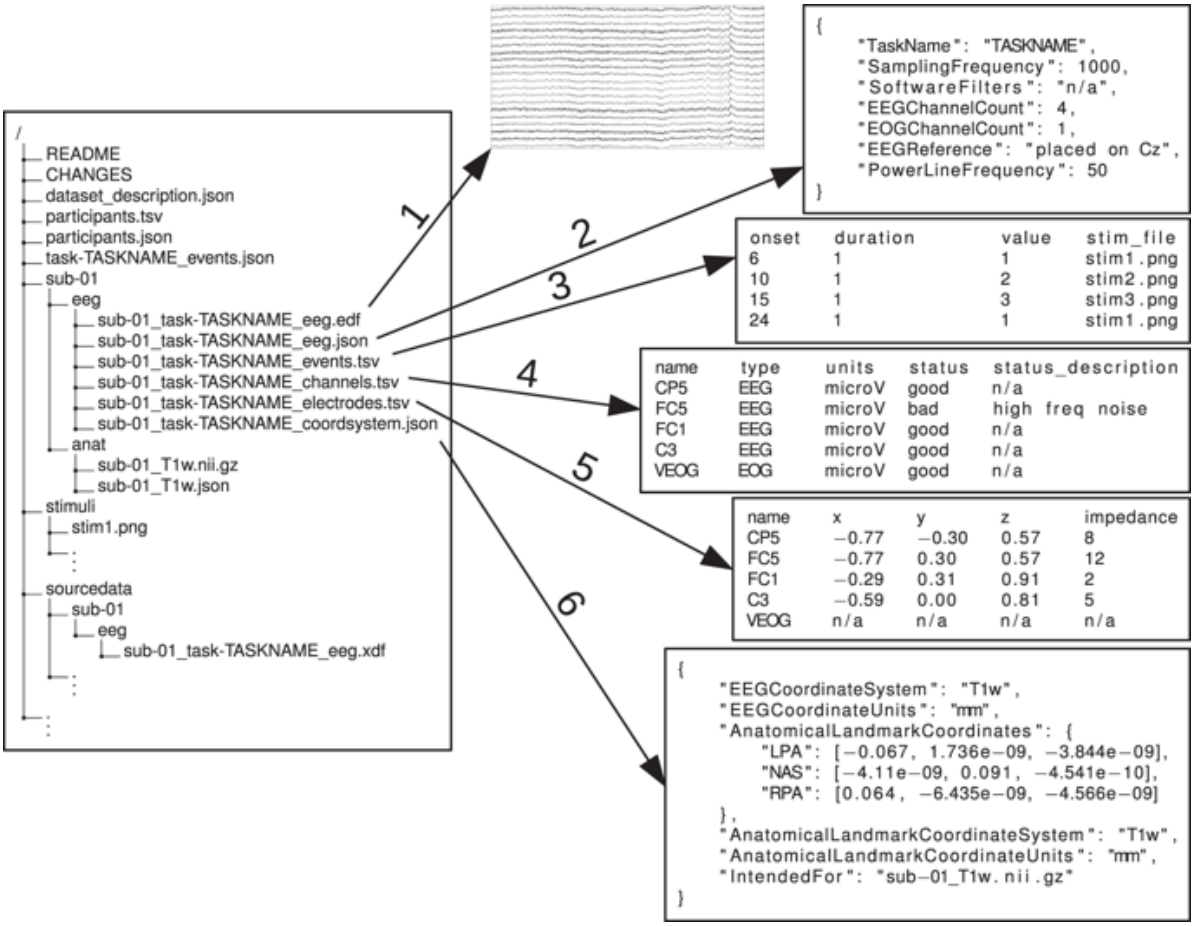
Datový model formátu `NIX` je zobrazen na obrázku č. 2.1. Skládá se ze šesti hlavních elementů: *Block, DataArray, Tag, MultiTag, Source* a *Group*. *Block* představuje kontejner zapouzdřující všechny ostatní entity modelu. *DataArray* je multidimenzionální pole pro samotné ukládání dat. *MultiTag* a *Tag* se používají pro definování epoch a událostí v signálu. *Source* udržuje informace o původu a zdroji uchovávaných dat a *Group* umožňuje seskupování ostatních entit do logických celků [10].

K `NIX` datovému modelu na obrázku č. 2.1 je připojený metadatový model (Metadata storage) ve formátu `odML`. `odML` (open metadata Markup Language) je formát pro ukládání různých druhů metadat. Jeho základní datový model nabízí způsob, jak ukládat metadata strukturovaným způsobem, jenž je čitelný pro člověka i stroj. Metadata jsou v `odML` uložena ve tvaru klíč-hodnota v entitě *Property* [19]. Díky tomuto metadatovému modelu je v `NIX` datovém modelu možné anotovat data na několika úrovních. Lze ukládat globální metadata například o provedeném experimentu, ale také

popisovat samotná data.

2.2 BIDS

Brain Imaging Data Structure (BIDS) je projekt pilotně vytvořený pro sdílení neuroimaging dat. Základem BIDS specifikace jsou jednoduché formáty souborů a adresářové struktury, které reflektují současné laboratorní postupy a zpřístupňují je široké řadě výzkumníků. S jeho rychlou expanzí bylo vytvořeno i jeho rozšíření pro EEG data zvané EEG-BIDS.



Obrázek 2.2: Specifikace formátu EEG-BIDS [32]

Specifikace formátu EEG-BIDS je zobrazena na obrázku č. 2.2. Rozšíření EEG-BIDS využívá původní BIDS specifikaci v podobě adresářové struktury, kterou reprezentuje levá strana obrázku č. 2.2. Pravá strana schématu ilustruje strukturu uložení EEG signálu a jeho metadat. BIDS také poskytuje Python knihovnu pro manipulaci s tímto formátem [32].

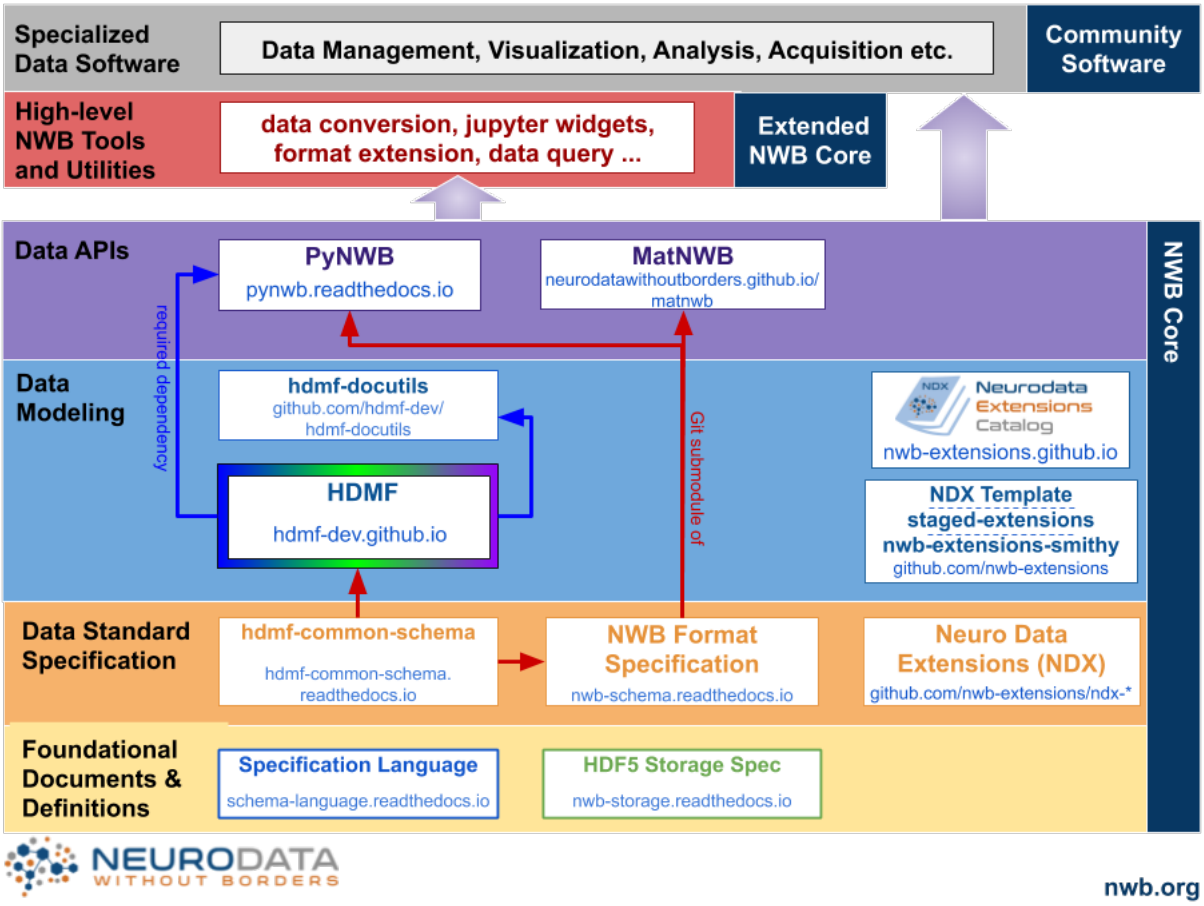
2.3 NWB

Neurodata Without Borders je datový formát pro ukládání a sdílení neurofyziologických dat, který byl navržen pro ukládání široké škály druhů neuropsychologických dat včetně EEG.

Pro hierarchické uspořádání dat používá NWB formát tyto hlavní entity:

- *Group* - symbolizuje složku a může obsahovat libovolný počet dalších skupin a datasetů.
- *Dataset* - jedná se o multidimenzionální pole, které poskytuje hlavní prostředky pro ukládání dat.
- *Attribute* - malý dataset, který je připojen ke konkrétní *Group* nebo *Dataset* a obvykle se používá k ukládání metadat.
- *Link* - je odkaz na konkrétní *Group* nebo *Dataset*.

Pro ukládání NWB formátu se používá HDF5 a pro manipulaci s formátem NWB je vytvořena Python knihovna PyNWB a MATLAB knihovna MatNWB. Struktura celého NWB ekosystému je zobrazena na obrázku č. 2.3 [36].



Obrázek 2.3: NWB ekosystém [7]

2.4 BrainVision

Neuroinformatická laboratoř na FAV ZČU používá pro měření EEG záznamů hardwarové a softwarové vybavení (zesilovač a rekordér) od firmy *Brain Products GmbH*, které získaná data ukládá do formátu `BrainVision Core Data Format 1.0`. Tento formát definuje strukturu zapisování dat do tří souborů:

1. textový hlavičkový soubor obsahující metadata s příponou *.vhdr*,
2. textový značkový soubor obsahující informace o událostech (stimulech) v datech s příponou *.vmrk*,
3. binární datový soubor obsahující surová EEG data a další signály zaznamenané spolu s EEG s příponou *.eeg, .avg* nebo *.seg*.

Všechny tyto soubory se musejí nacházet ve stejné složce, přičemž hlavičkový a datový soubor jsou povinné. Struktura binárního datového souboru je popsána v hlavičkovém souboru. Hlavičkový i značkový soubor mají pevně danou strukturu ve formátu klíč=hodnota a jejich úplný popis včetně popisu všech použitelných klíčů se nachází v [3].

Vzhledem k robustnosti a rozsáhlému používání `BrainVision Core Data Format 1.0` jej projekt `BIDS` uznává jako jeden z doporučených standardů pro ukládání EEG/iEEG dat [2].

3 Strojové učení

Strojové učení je oblastí umělé inteligence zabývající se algoritmy a technikami, které umožňují počítačovému systému měnit svůj vnitřní stav tak, aby zefektivnil schopnost přizpůsobení se změnám okolního prostředí. V klasickém programování se používá postup takový, že lidé zadávají pravidla a data a program poté generuje odpovědi. Ve strojovém učení jsou spolu s daty do programu vkládané odpovědi očekávané při zpracování těchto dat a program generuje pravidla. Smyslem je, že program může tato pravidla aplikovat na nová data a poskytovat tak originální odpovědi.

Druhy strojového učení lze rozdělit do 4 hlavních kategorií [13]:

- **Učení s učitelem** - jedná se o nejběžnější případ. Skládá se z učení, jak mapovat vstupní data na známé odpovědi na základě množiny příkladů. V dnešní době se jedná o dominantní formu strojového učení.
- **Učení bez učitele** - v tomto případě nemá algoritmus k dispozici známé odpovědi. Tento přístup spočívá v hledání podobných elementů ve vstupním prostoru dat a jejich třídění do skupin bez možnosti ověření správného zatřídění. Známými příklady využití algoritmů učení bez učitele jsou redukce dimenzionality a shlukování. Metody učení bez učitele se obvykle používají k většímu pochopení vstupní datové množiny ještě předtím, než se na data využijí metody učení s učitelem.
- **Samořízené učení** - je to specifický případ učení s učitelem. Rozdílem je, že sice existují známá označení tříd, která nebyla vytvořena člověkem, ale byla vygenerována ze vstupních dat obvykle pomocí heuristického algoritmu.
- **Posilované učení** - je nazývané také jako zpětnovazební. U této metody nejsou známá označení tříd, ale učícímu modelu se za jeho rozhodnutí poskytuje zpětná vazba. Podle zpětné vazby se model učí volit své akce tak, aby maximalizoval hodnotu nějaké odměny.

Základními druhy úloh strojového učení jsou:

- **Klasifikace** - algoritmus se rozhoduje, do jaké třídy patří nově příchozí vzorek. Čili se jedná o predikci kategoriální veličiny.
- **Regrese** - algoritmus pro daný vstupní vzorek vygeneruje výstupní hodnotu. Čili se jedná o predikci numerické veličiny.

Při detekci evokovaných potenciálů jsou EEG experimenty konstruovány tak, že se zkoumá reakce subjektu na vnější stimul (obrazový, zvukový). Data se známým zařazením do tříd se používají jako trénovací a po natrénování se z nových vzorků EEG signálu zkouší odhadnout, do jaké třídy patří. Tudíž se jedná o klasifikační úlohu.

V neuroinformatické laboratoři na FAV ZČU se pro klasifikaci EEG signálu používají tyto klasifikační metody strojového učení:

- lineární diskriminační analýza,
- metoda podpůrných vektorů.

A z metod hlubokého učení se používají tyto dvě:

- konvoluční neuronové sítě,
- rekurentní neuronové sítě.

Všechny používané metody budou popsány v následující sekci.

3.1 Lineární klasifikátory

Lineární klasifikace je nejjednodušší metodou strojového učení, kdy jsou množiny prvků patřících do jednotlivých tříd od sebe oddělitelné lineárně.

3.1.1 Lineární diskriminační analýza

Lineární diskriminační analýza (LDA) je jednou z technik lineární klasifikace pro klasifikaci do dvou a více tříd. Obecně je LDA metodou pro snižování dimenze dat, jejímž hlavním cílem je promítnutí dat do prostoru nižší dimenze při zachování důležitých informací, jakou je separabilita jednotlivých tříd. Ta je zajištěna maximalizací poměru vzdáleností mezi třídami ku vzdálenostem uvnitř tříd, čímž je dosažena maximální diskriminace. Optimální transformace může být spočtena aplikací dekompozice pomocí vlastních čísel (*eigendekompozice*) na rozptylové matice [45].

Postup LDA lze shrnout do těchto 5 základních kroků [34]:

1. vypočítat d-dimenzionální vektor středních hodnot pro různé třídy v datasetu,
2. získat rozptylovou matici mezi třídami (*between-class*) a rozptylovou matici uvnitř tříd (*within-class*),
3. aplikovat dekompozici pomocí vlastních čísel na rozptylové matice,

4. sestupně seřadit vektory vlastních čísel a zvolit k vlastních vektorů s nejvyššími vlastními čísly pro vytvoření d x k dimenzionální matice W, kde každý sloupec představuje vlastní vektor,

5. využít tuto matici W k transformaci vzorků do nového podprostoru s nižší dimenzí. Toho lze dosáhnout násobením matice $Y = X$ x W, kde X je n x d dimenzionální matice n vzorků.

Výsledné zařazování testovacích vzorků do tříd probíhá podle daného rozhodovacího pravidla, jako např.:

- Pravidlo maximální věrohodnosti,
- Bayesovo pravidlo,
- Fisherovo pravidlo.

Například pro Bayesovo pravidlo má lineární diskriminační funkce *L(x)* následující tvar:

$$L(x) = \beta^T x + \gamma,$$

kde

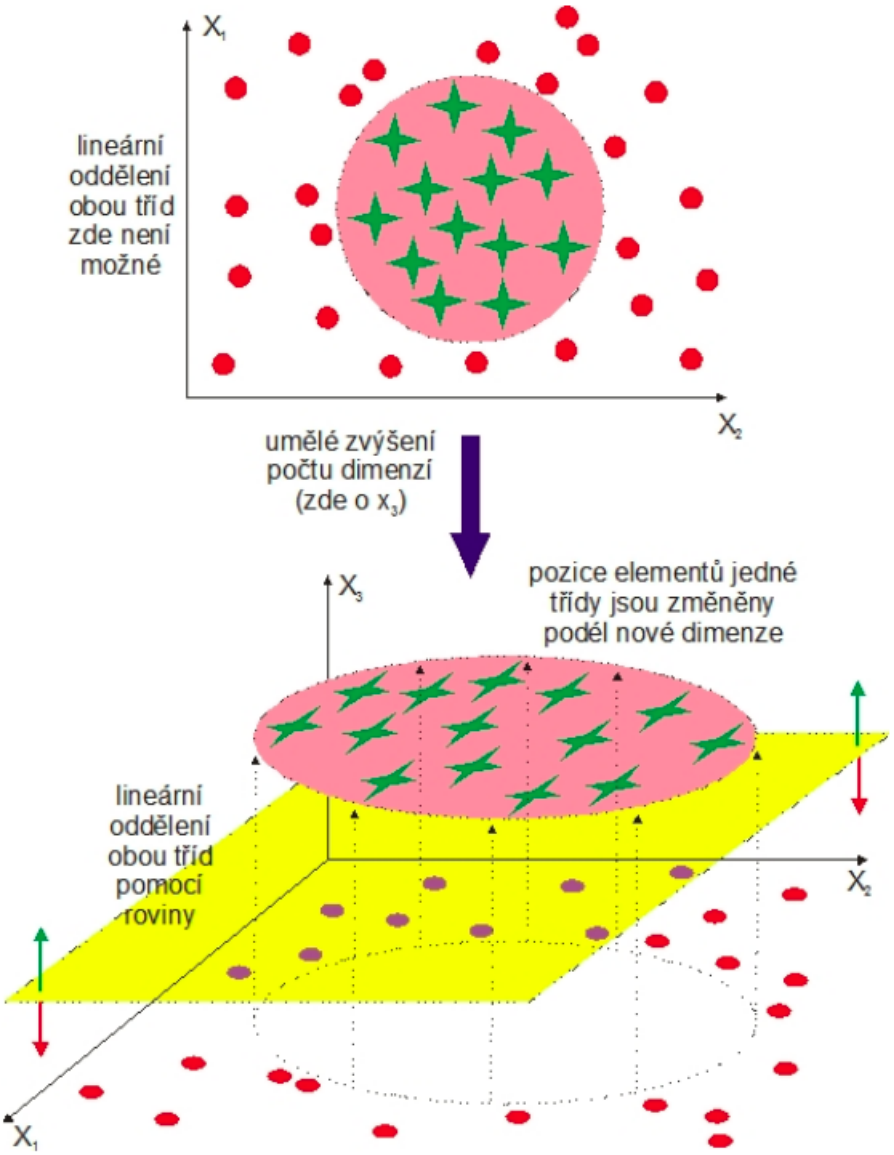
$$\beta^T = (\mu_i - \mu_j)^T \Sigma^{-1};$$
$$\gamma = -\frac{1}{2}\beta^T(\mu_i + \mu_j) - \frac{1}{2} ln \frac{\pi_j}{\pi_i},$$

kde π_j představuje apriorní příslušnost k j-té třídě (tj. $P(A_j) = \pi_j$), μ_j je střední hodnota j-té třídy a Σ je kovarianční matice jakékoliv třídy, protože LDA předpokládá, že všechny třídy mají shodnou variační strukturu [46].

3.1.2 Metoda podpůrných vektorů

Algoritmus metody podpůrných vektorů (SVM) se snaží řešit klasifikační problém nalezením rozhodovací hranice mezi dvěma množinami bodů patřících do dvou různých kategorií. Principem této metody je převod daného prostoru vstupních dat do jiného, více-dimenzionálního, ve kterém lze třídy od sebe oddělit lineárně.

Tento princip je zobrazen na obrázku č. 3.1. V původním dvourozměrném prostoru jsou dvě třídy od sebe nelineárně oddělené kružnicí. Přidáním další dimenze vznikne možnost prvkům třídy uvnitř kružnice přidat další souřadnici, která je posune například nahoru podél nové osy X_3. Pro oddělení obou tříd lze již použít rovinu rovnoběžnou s rovinou danou osami X_1 a X_2. Otázkou ale zůstává, kam nejlépe umístit lineární hranici tak, aby umožňovala co nejefektivnější kategorizaci budoucích dat [47].



Obrázek 3.1: Princip lineárního oddělení dvou tříd s nelineárními hranicemi pomocí přidané dimenze [47]

Dobrá rozhodovací hranice (rozdělující nadrovina) se vypočítá tak, že se snažíme maximalizovat vzdálenost mezi nadrovinou a nejbližšími datovými body z každé třídy pomocí kroku nazývaného maximalizace odstupu. Ten umožňuje, aby se rozhodovací hranice kvalitně zobecňovala na nové příklady dat.

Technika mapování dat na novou, vícerozměrnou reprezentaci, kde se klasifikace zjednodušuje, může vypadat dobře na papíře, ale v praxi je často

výpočetně neřešitelná. Proto SVM využívá takzvaného jádrového triku. Jeho podstatou je nalezení nové rozhodovací nadroviny v novém reprezentačním prostoru. Nemusí se tedy explicitně počítat souřadnice bodů v novém prostoru. Stačí vypočítat vzdálenost mezi dvojicemi bodů v daném prostoru, což může být efektivně provedeno pomocí takzvaných jádrových funkcí.

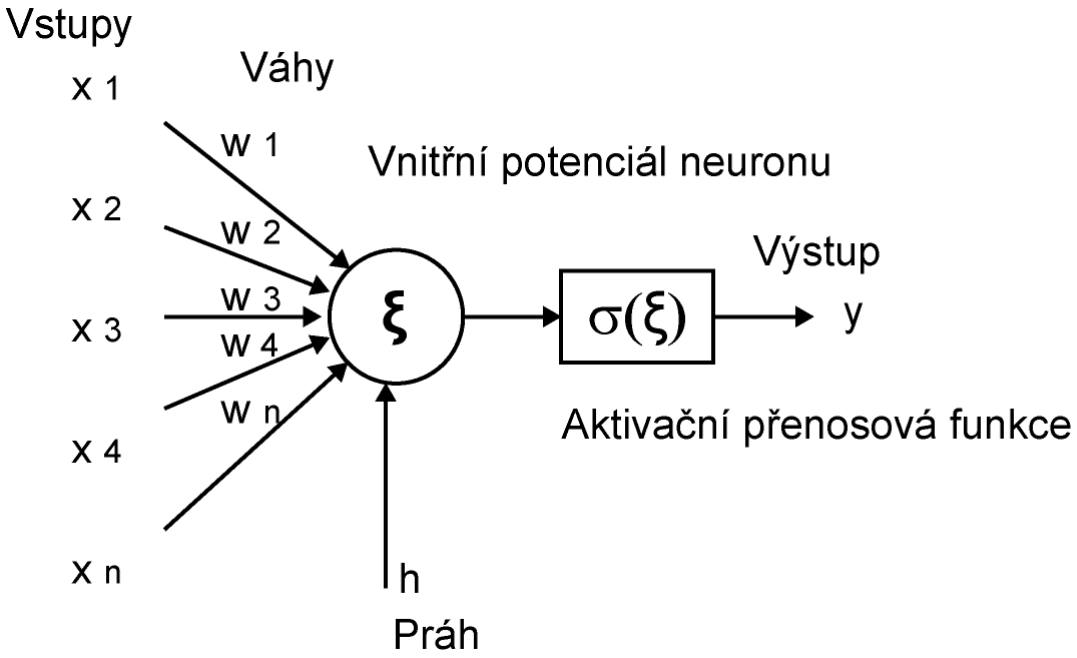
Jádrová funkce je výpočetně zvládnutelná operace, která mapuje libovolnou dvojici bodů v počátečním prostoru na vzdálenost mezi těmito body v prostoru nové reprezentace. Přitom dojde zcela k vynechání explicitního výpočtu nové reprezentace. Jádrová funkce je typicky vytvářena ručně, nikoliv naučením z dat. V případě SVM je naučena pouze rozdělující nadrovina [13].

3.2 Hluboké učení

Hluboké učení je specifickou podskupinou strojového učení. Jedná se o nový přístup k učení se reprezentací z dat, který klade důraz na učení se následujících vrstev stále smysluplnějších reprezentací. Počet vrstev přispívajících k modelu dat se nazývá hloubka modelu. Moderní hluboké učení často zahrnuje desítky nebo dokonce stovky po sobě jdoucích vrstev reprezentací, jež se všechny automaticky učí při trénování. V hlubokém učení jsou tyto vrstvené reprezentace téměř vždy natrénovány prostřednictvím modelů nazývaných neuronové sítě.

Umělá neuronová síť je matematický simulační model informačního procesoru inspirovaný biologickými neuronovými sítěmi. Neuronová síť se skládá z množiny umělých neuronů, které mohou být podle účelu uspořádány do vrstev a navzájem propojeny synapsemi, jejichž významnost určují synaptické váhy [16].

Schematický model jednotlivého neuronu je uveden na obrázku č. 3.2. Neuron lze rozdělit na několik částí: synapse ohodnocené vahami w, které přivádějí vstupy x do těla neuronu, vlastní tělo neuronu, v němž je získáván vnitřní potenciál neuronu ξ, blok aktivační přenosové funkce σ a konečně výstup neuronu y. Do těla neuronu vstupuje ještě konstantní hodnota prahu h.



Obrázek 3.2: Schematický model neuronu [21]

Vnitřní potenciál neuronu ξ může být vyčíslen jako

$$\xi = \sum_{i=1}^{n} w_i x_i - h.$$

Práh se zpravidla modeluje jako jedna z vah tak, že vstupní vektor i vektor vah jsou rozšířeny o nultou pozici. V takovém případě se práh stává jednou z vah a v průběhu trénování podléhá adaptaci. Výstup neuronu pak lze vyjádřit takto

$$y = \sigma(\xi) = \sigma(\sum_{i=0}^{n} w_i x_i),$$

kde $x_0 = 1$ a $w_0 = -h$.

Aktivační funkce σ je obecně nelineární funkcí transformující hodnotu vnitřního potenciálu neuronu. Obvyklými aktivačními funkcemi jsou logistická sigmoida nebo například hyperbolický tangens [21].

Nastavení pokud možno co nejlepších hodnot vah jednotlivých vrstev je úkolem učení neboli trénování neuronové sítě. Při trénování neuronové sítě se využívá ztrátové funkce, která při trénování přebírá predikci sítě a skutečný cíl a vypočítá vzdálenosti zachycující, jak dobře síť predikovala výsledek v tomto případě. Toto skóre se poté použije jako zpětnovazební signál, podle něhož se upravují hodnoty vah ve směru, který sníží ztrátové skóre pro

daný případ. Toto nastavení je úlohou optimalizátoru, který implementuje algoritmus zpětného šíření. Zpočátku jsou váhám sítě přiřazeny náhodné hodnoty a s každým trénovacím příkladem jsou váhy nastaveny jinak za účelem minimalizace ztrátového skóre.

Hluboké učení v řadě případů nabízí lepší výkon, což ale není jediným důvodem vzrůstající popularity těchto metod. Hluboké učení také usnadňuje řešení celého problému, protože dokáže automatizovat krok, který byl v klasických přístupech strojového učení považován za náročný a velmi důležitý, a tím je extrakce příznaků [13].

3.2.1 Konvoluční neuronová síť

Jednou z příčin rozmachu používání neuronových sítí je snaha umožnit strojům pohlížet na svět jako lidé. Vnímat jej podobným způsobem a využívat znalosti pro praktické úkoly, jakým je například rozpoznávání obrazu. V oblasti počítačového vidění zaznamenal slibných výsledků jeden typ neuronových sítí, a to takzvaná konvoluční neuronová síť.

Konvoluční neuronová síť (CNN) je algoritmem hlubokého učení, který dokáže objektům ve vstupním obraze přiřadit různou důležitost. Jak bylo zmíněno výše, v klasických metodách strojového učení je většinou nutné ze vstupních dat extrahovat příznaky ručně, kdežto architektura CNN je postavena tak, že v její konvoluční části by se tyto příznaky měly extrahovat automaticky. CNN je jedním typem takzvaných dopředných sítí, které jsou charakteristické tím, že výstup jedné vrstvy je napojen na vstup následující vrstvy a signál se sítí šíří pouze ve směru ze vstupu na výstup, to znamená, že neobsahuje žádnou zpětnou vazbu [42].

CNN se skládá ze dvou částí, z konvoluční a klasifikační. Konvoluční část tvoří konvoluční a takzvané pooling vrstvy. Jejich úkolem je získat ze vstupního obrazu sadu koeficientů, které se poté předají do klasifikační části neuronové sítě. O zpracování vstupního obrazu a získání lokálních vzorů se stará konvoluční vrstva pomocí tzv. konvoluční operace. Může se jednat o učení vzorů i ve velmi malých oknech, například 3x3 pixely. Tato klíčová charakteristika poskytuje CNN dvě zajímavé vlastnosti:

- Ve chvíli, kdy se CNN naučí vzorec například v pravém horním rohu obrázku, může jej poté rozpoznat kdekoliv. Toto je rozdíl oproti klasické hustě propojené neuronové síti, která by se jej po objevení vzoru v jiném místě musela znovu naučit. Toto činí CNN efektivní při zpracování obrazových dat, protože zpracování vizuálních dat je vůči různým posunům neměnné. Díky této vlastnosti potřebují CNN méně trénovacích dat pro zajištění zobecnění natrénované sítě.

- CNN se mohou naučit prostorové hierarchie vzorů. Například první konvoluční vrstva se naučí malé lokální vzory jako jsou hrany. Další vrstva se může naučit již větší vzory složené ze vzorů v první vrstvě a tak dále.

Pooling vrstvy na svém vstupu přijímají výstup z předcházející konvoluční vrstvy. Ty provádějí jen jednoduché výpočty, jako je například výběr pixelu s maximální hodnotou nebo jejich zprůměrování, a těmi zmenšují rozlišení vstupních dat.

Dalším krokem je napojení konvoluční části sítě do klasické hustě propojené klasifikační sítě. Tato druhá část sítě zpracovává jednorozměrné vektory. Je nutné tedy výstupní data konvoluční části, která jsou typicky ve tvaru 3D tensoru (výška x šířka x počet kanálů), převést do tvaru 1D, což lze provést přidáním zplošťovací (flatten) vrstvy. Z výsledného 1D vektoru se snaží klasifikační část sítě určit výslednou třídu vstupního obrázku [13].

3.2.2 Rekurentní neuronová síť

Rekurentní neuronová síť (RNN) je typem neuronové sítě, která má vnitřní cyklus. Uplatnění nachází zejména v případech, kdy je třeba zpracovat sekvenci nebo časovou řadu datových bodů. RNN zpracovává sekvenci iterováním přes její jednotlivé prvky a udržuje stav obsahující informace vztahující se k tomu, co doposud zpracovala. RNN obsahuje tuto vnitřní paměť a zohledňuje údaje o doposud zpracovaných vstupech při zpracování nového vstupu. Rozhodnutí, kterého RNN dosáhla v čase $t - 1$, ovlivňuje následující rozhodnutí v čase t. RNN mají tedy dva zdroje vstupu, aktuální zpracovávaný vstup a nedávnou minulost, které společně určují, jak reagovat na nová data. RNN přijímá nepřetržitě výstupy svých vlastních neuronů jako nové vstupy do nich samotných nebo některých z předchozích vrstev [13].

Získané informace o vstupní sekvenci se ukládají do skrytého stavu RNN. Tento skrytý stav dokáže zahrnout velké množství vstupních dat a ovlivňuje zpracování každého nového příkladu. Díky němu lze ve zpracovávané sekvenci objevit i závislosti, které jsou od sebe oddělené velkým časovým intervalem.

LSTM síť

Long short-term memory síť (LSTM) je jednou z praktických implementací teoretické RNN. Vyznačuje se hlavně schopností učit se dlouhodobé závislosti. LSTM udržuje informace o vstupní sekvenci mimo hlavní tok RNN v takzvané buňce. Do této buňky lze zapisovat nebo číst podobně jako z normální paměti. O tom, kdy povolit čtení nebo zápis, rozhoduje buňka prostřednictvím brán.

Ty reagují na vstupní signál, jejich úkolem je uchovat relevantní a zahodit nerelevantní informace, a podobně jako uzly neuronové sítě blokují nebo předávají informaci dál na základě sady svých filtrů a vah. Tyto váhy se nastavují stejně jako u klasické neuronové sítě učením. Klasická buňka obsahuje tři brány, vstupní, výstupní a zapomínající. Vstupní brána má za úkol určit, zda uložit nové informace ze vstupu do buňky. Zapomínající brána rozhoduje, jestli dojde k vymazání současného stavu buňky, a výstupní brána zajistí, zda bude výstup v současném kroku ovlivněn aktuálním stavem buňky. Stavy otevření respektive zavření bran se mohou různě kombinovat [29].

3.3 Machine-learningové knihovny v Pythonu

Výzkumníci z různých oblastí často používají programovací jazyk Python pro analýzu nebo zpracování dat. V EEG komunitě jsou společně s MATLABem nejpoužívanějšími jazyky. Pro analýzu dat je často žádoucí využití metod strojového učení, a proto existuje několik knihoven implementovaných v Pythonu, které je poskytují. Největšími a nejoblíbenějšími jsou `scikit-learn` pro klasické metody strojového učení a `Keras` a `PyTorch` pro metody hlubokého učení.

3.3.1 `scikit-learn`

`Scikit-learn` je jednoduchý a efektivní open source nástroj pro prediktivní analýzu dat. Je postavený na dalších matematicko-vědeckých Python knihovnách `NumPy`, `SciPy` a `matplotlib`. Knihovna poskytuje funkce z oblastí klasifikace, regrese, shlukování, redukce dimenzionality, ohodnocování modelů a předzpracování.

Objekty v knihovně jsou organizovány okolo tří základních API, a to rozhraní *estimator* pro sestavování a trénování modelů, rozhraní *predictor* pro vytváření predikcí a rozhraní *transformer* pro transformaci dat. Tato rozhraní jsou specifikována jako doplňková a přesně neoddělují třídy knihovny, ale spíše se překrývají. Například modely klasifikátorů v knihovně jsou typicky *estimator* a zároveň i *predictor*.

API knihovny bylo navrženo tak, aby dodržovalo tyto principy:

- **Konzistence** - všechny objekty sdílejí konzistentní rozhraní obsahující omezenou sadu metod.
- **Přístupnost** - parametry konstruktorů a veškeré atributy vyžadované algoritmy strojového učení jsou veřejné.

- **Nešíření tříd** - algoritmy strojového učení jsou jedinými objekty, které mají být reprezentovány pomocí vlastních tříd knihovny. Data jsou reprezentována jako matice třídy `NumPy` nebo třídy `SciPy` a pro názvy a ostatní parametry se používají standardní datové typy Pythonu.

- **Kompozice** - mnoho machine-learningových postupů je vyjádřeno jako sekvence nebo kombinace transformací dat. Kde je to možné, jsou algoritmy strojového učení skládány z existujících bloků.

- **Rozumné výchozí hodnoty** - kdykoliv operace vyžaduje uživatelem definovaný parametr, je pro ni nastavena výchozí hodnota, která by měla zajistit, že se úkol provede rozumným způsobem a poskytne alespoň základní řešení dané úlohy.

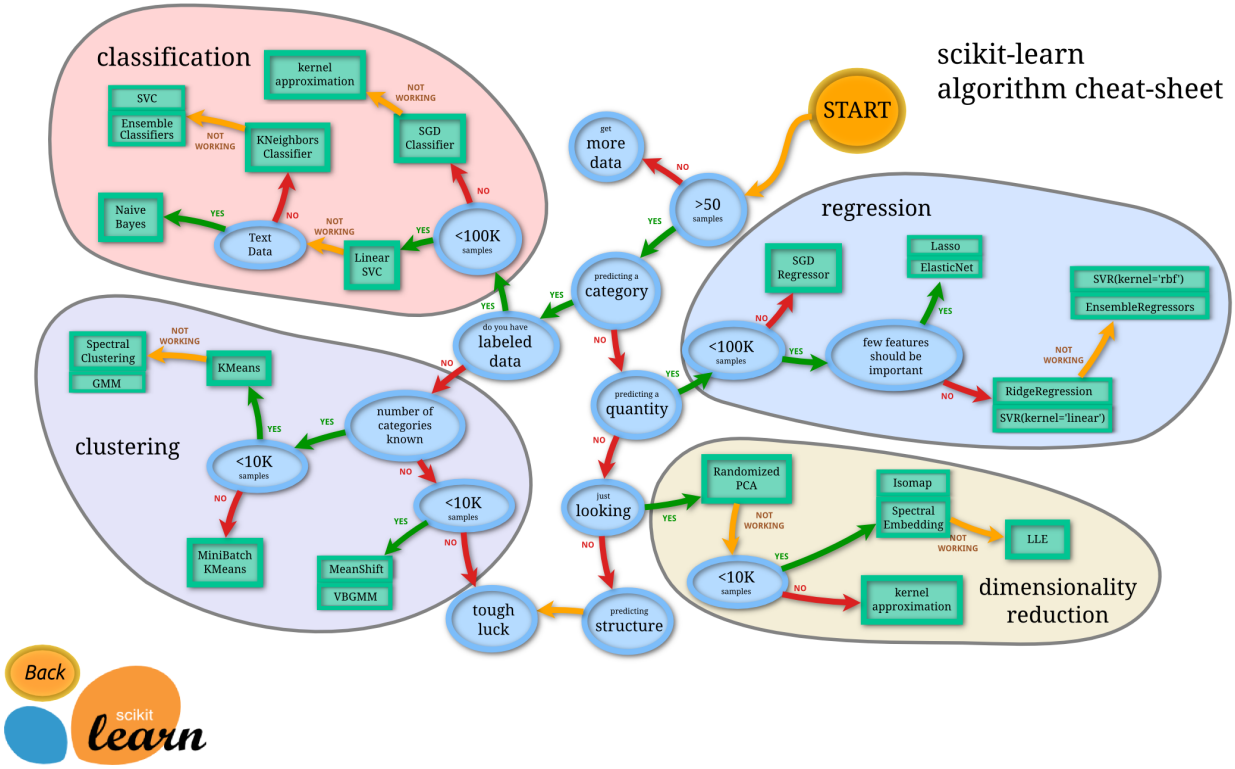
Analyzované datasety jsou v knihovně reprezentovány jako `NumPy` multidimenzionální pole a nebo `SciPy` matice, a `scikit-learn` tak může těžit z jejich efektivních vektorizovaných operací. Tyto datové reprezentace jsou populární napříč mnoha knihovnami v Pythonu, tudíž lze metody knihovny `scikit-learn` snadno integrovat do vlastního projektu společně s metodami ostatních Python knihoven [11].

Estimator API

Rozhraní *estimator* je základním stavebním kamenem knihovny. Definuje mechanismy pro manipulaci s objekty a poskytuje metodu *fit()* pro naučení modelu z tréninkových dat. Všechny algoritmy strojového učení pro klasifikaci, regresi a shlukování jsou objekty implementující toto rozhraní. Další úkoly strojového učení jako extrakce příznaků a redukce dimenzionality jsou také *estimatory*.

V prvním kroku musí být *estimator* inicializován a objektu předány parametry, které daný algoritmus strojového učení vyžaduje. Samotné učení se poté provádí metodou *fit()*. Úkolem této metody je provést učící algoritmus, určit parametry specifické pro daný model a nastavit je jako veřejné atributy *estimator* objektu. Natrénovaný model lze poté použít pro provádění predikcí nebo transformací vstupních dat [11].

Graf ilustrující výběr správného objektu *estimator* se nachází na obrázku č. 3.3.



Obrázek 3.3: Graf ilustrující výběr správného objektu *estimator* [31]

Predictor API

Rozhraní *predictor* definuje metodu *predict()*, která pro nové testovací vzorky produkuje výstup představující zařazení vzorků do tříd podle naučeného modelu *estimatoru*. *Predictory* mohou také obsahovat metody ohodnocující kvalitu předpovědi. V případě lineárních modelů rozhodovací metoda vrací vzdálenost od oddělující hranice. Některé *predictory* poskytují metodu *predict_proba()*, která vrací pravděpodobnosti příslušností vzorků ke třídám. *Predictor* objekty musejí implementovat funkci *score()* pro vyhodnocení výkonu modelu na sadě vstupních dat [11].

Transformer API

Protože je v oblasti strojového učení obvyklé filtrovat nebo nějakým způsobem předzpracovat data před jejich předáním učícímu algoritmu, implementují některé objekty *estimator* rozhraní *transformer*, které definuje metodu *transform()*. Tato metoda bere jako vstup sadu dat a vrací jejich transformovanou verzi. Všechny algoritmy předzpracování, extrakce příznaků a redukce dimenzionality jsou v knihovně `scikit-learn` implemetovány jako objekty *transfor-*

mer. Příkladem může být *StandardScaler* z balíku `sklearn.preprocessing`, který vstupní data standardizuje na nulovou střední hodnotu a jednotkový rozptyl. Při vytvoření objektu *transformer* a zavolání metody *fit()* dojde k nastavení a uložení parametrů, které *transformer* vypočítal pro danou tréninkovou sadu, tudíž při jeho pozdějším použití na testovací sadu dojde k transformaci do stejného prostoru příznaků. Každý objekt *transformer* poskytuje kombinovanou metodu *fit_transform()*, která při zavolání provede nastavení parametrů i transformaci najednou [11].

Pipeline

Klíčovou rozlišovací vlastností `scikit-learn` API je možnost vytvářet nové objekty *estimator* kombinací několika dalších objektů *estimator*. Je možné kombinovat typické machine-learningové workflows do jednoho objektu, který implementuje rozhraní *estimator*, a tudíž jej můžeme používat známým způsobem. `Scikit-learn` algoritmy pro ohodnocování modelu se mohou aplikovat na již zkombinované objekty estimator, což umožňuje globální optimalizaci všech parametrů v komplexním postupu zpracování. Skládání objektů se může provést dvěma způsoby, a to sekvenčně pomocí objektu *Pipeline* a paralelně pomocí objektu *FeatureUnion*.

Klasický postup analýzy dat se většinou skládá z pevně dané sekvence kroků (např. filtrování, extrakce příznaků, učení, vyhodnocení). Posloupnost těchto N kroků může být kombinována do *Pipeline*, pokud prvních N-1 kroků jsou objekty *transformer* a poslední může být buď *predictor*, nebo *transformer*, anebo obojí. V případě, že poslední *estimator Pipeline* je *predictor*, může být celá *Pipeline* použita jako *predictor*, a to samé analogicky s objektem *transformer*.

Objekt *FeatureUnion* kombinuje více objektů *transformer* do jednoho a poté zřetězuje jejich výstupy. Jako vstup bere seznam objektů *transformer* a při volání metody *fit()* nad objektem *FeatureUnion* dojde k zavolání metody fit() nezávisle na každém zúčastněném *transformeru* [11].

Ohodnocování modelu

V experimentu je nutné nějakým způsobem ohodnotit úspěšnost klasifikačního algoritmu strojového učení. Potřebné nástroje pro validaci se nacházejí v balíku `sklearn.model_selection`. Tento balík poskytuje různé metody na rozdělení datové množiny na trénovací a testovací a metody pro ohodnocení modelu, kterou je například *cross_val_score()* poskytující cross-validaci s daným modelem a metrikou. Předdefinové metriky, které se dají použít pro

ohodnocování klasifikačního modelu se nacházejí v balíku `sklearn.metrics`, který poskytuje všechny základních ohodnocovací metriky [11].

Lineární diskriminační analýza

Třída reprezentující lineární diskriminační analýzu se nachází v balíku `sklearn.discriminant_analysis`. LDA je v knihovně implementována jako klasifikátor s lineární rozhodovací hranicí využívající Bayesovo rozhodovací pravidlo. Objekt pochopitelně implementuje rozhraní *estimator* a *predictor*, ale jelikož může být natrénovaný model použit také pro redukci dimenzionality, implementuje model i rozhraní *transformer*.

Při inicializaci této třídy lze v parametrech definovat způsob transformace dat do nižší dimenze a uživatel může volit mezi singulárním rozkladem (SVD), který je v případě nedefinování v parametru nastaven jako výchozí, metodou nejmenších čtverců a dekompozicí podle vlastních vektorů. Dalším volitelným parametrem je tzv. shrinkage, což je jednou z možností obrany proti přeučení (zlepšení generalizačních schopností modelu). Může být nastavena jako pevná konstanta mezi 0 a 1 a nebo zvolena možnost *'auto'*, která tento parametr vypočte podle Ledoit-Wolf lemma [26]. Uživatel může v parametru definovat apriorní pravděpodobnosti příslušnosti ke třídám. Když je nedefinuje, budou odvozené z trénovacích dat. Dále stanovit počet komponent pro redukci dimenzionality a určit, zda se má explicitně počítat a ukládat kovarianční matice tříd. A posledním parametrem je prahová hodnota pro SVD určující, které dimenze budou zahozeny [31].

Metoda podpůrných vektorů

Třídy představující učící algoritmus metodou podpůrných vektorů se nacházejí v balíku `sklearn.svm`. Třída reprezentující model klasifikátoru SVM se jmenuje *SVC*. Tento objekt implementuje rozhraní *estimator* a *predictor*. Pomocí parametrů může uživatel specifikovat regularizační parametr představující penalizaci chyb s výchozí hodnotou 1.0. Může specifikovat jádrovou funkci výběrem z předdefinovaných lineární, polynomiální, radiální bázové funkce anebo funkce sigmoid a k nim definovat vyžadované odpovídající koeficienty. Uživatel také může definovat vlastní jádrovou funkci. Může určit velikost jádrové cache v MB a několik dalších parametrů [31].

3.3.2 `Keras`

`Keras` je vysokoúrovňové API pro hluboké učení v Pythonu. Tento framework poskytuje základní prostředky a entity pro vývoj klasifikačních modelů

hlubokého učení. `Keras` nepracuje s nízkoúrovňovými operacemi a místo toho se spoléhá na specializovanou, dobře optimalizovanou tensorovou knihovnu, která tak slouží jako její motor. V současné verzi je knihovna postavena na backendu `TensorFlow`, díky čemuž má `Keras` následující klíčové vlastnosti:

- Umožňuje efektivně provádět nízkoúrovňové tensorové operace na CPU i GPU.
- Podporuje vytváření konvolučních a rekurentních neuronových sítí, i jejich kombinací.
- Umožňuje výpočet gradientu libovolných diferencovatelných výrazů.
- Podporuje škálování výpočtu na více zařízení.
- Umožňuje jednoduchý export vytvořených modelů a programů.

Dle tvůrců je `Keras` nejpopulárnějším nástrojem pro algoritmy hlubokého učení. Používají jej například CERN pro svůj velký hadronový urychlovač částic nebo společnost NASA. Je také nejvíce využívaný vítěznými týmy v deep-learningových soutěžích Kaggle.

Modely v `Kerasu` pracují se vstupními daty ve třech formátech:

- `NumPy` pole,
- `TensorFlow Dataset` objekt,
- Python generátory.

Před trénováním modelu je tudíž nutné převést analyzovaná data do jednoho z těchto podporovaných datových formátů [12].

`Keras` poskytuje tři druhy metod předzpracování, jež se dají zakomponovat do modelu neuronové sítě pomocí těchto předzpracovávajících vrstev:

- vektorizování textu pomocí *TextVectorization* vrstvy,
- normalizování příznaků na jednotkový rozptyl pomocí *Normalization* vrstvy,
- práce s obrazovými daty (oříznutí, změna měřítka ...) pomocí *Rescaling* a *CenterCrop* vrstvy.

Vrstvy

Klíčovými datovými strukturami v `Kerasu` jsou vrstvy a modely. Vrstva představuje jednoduchou transformaci vstupních dat na nějaký výstup. Při vytváření modelu je nejdříve nutné definovat tvar vstupu a aktivační funkci pro první (vstupní) vrstvu - v EEG případě např. počet_epoch x počet_kanálů x počet_hodnot. Poté můžeme zadat posloupnost vrstev, a nakonec definovat výstupní vrstvu s počtem klasifikačních tříd a aktivačních funkcí. `Keras` poskytuje velké množství předdefinovaných parametrizovaných vrstev, a také možnost vytvoření vlastní vrstvy. Zároveň nabízí několik druhů aktivačních funkcí nebo inicializátorů vah vrstev. Kompletní soupis implementovaných vrstev v `Kerasu` se nachází v [4].

Model

Model představuje orientovaný acyklický graf vrstev. Nejjednodušším typem modelu je tzv. sekvenční model představující lineární posloupnost vrstev. Pro složitější architektury sítí knihovna nabízí využití funkcionálního API, které umožňuje generovat libovolné grafy vrstev. Při obou možnostech vytváření modelu neuronové sítě v `Kerasu` je nutné vytvořit instanci třídy *Model* a předat jí posloupnost vrstev. Dalším krokem je volba optimalizátoru a ztrátové funkce. Tyto parametry se modelu předávají pomocí metody *compile()* třídy *Model*, jež kromě těchto dvou parametrů umožňuje specifikovat například i metriky, kterými bude ohodnocována úspěšnost modelu. Pro všechny tři tyto hodnoty lze využít již implementované základní metody v `Kerasu`.

Učení a evaluace

Následovat již může samotné natrénování modelu. To zajišťuje metoda *fit()* třídy *Model*. Je nutné v parametrech této metody definovat trénovací datovou množinu a při učení s učitelem i jí odpovídající množinu správných přiřazení. Volitelnými parametry jsou pak dále počet vzorků použitých pro aktualizaci gradientu s výchozí nastavenou hodnotou 32, počet iterací trénování modelu, seznam zpětných volání nebo například úroveň logování. Lze také specifikovat, zda nějaká část trénovacích dat bude použita jako validační množina. Model tuto část dat oddělí od trénovacích a nebude je používat pro trénování. Po každé provedené iteraci učení modelu se pomocí této validační množiny provede evaluace modelu, vyhodnotí se ztrátová funkce a všechny ostatní metriky modelu. Smyslem použití validačních dat je možnost sledovat, jak se v průběhu učících iterací vyvíjely sledované metriky na neviděných datech, a díky znalosti těchto hodnot případně vyladit parametry vytvořeného modelu.

Je možné také definovat validační množinu již před začátkem trénování a předat ji metodě *fit()* v parametru. Tato metoda vrací objekt typu *History*, jenž schraňuje všechny informace posbírané během učení, což jsou sledované metriky nebo například ztrátové funkce pro všechny iterace přes validační i trénovací množinu.

Po dokončení trénování lze nad modelem zavolat funkci *predict()*, která pro danou testovací množinu vygeneruje odpovídající sadu zařazení do tříd a nad těmito výsledky poté manuálně počítat evaluační metriky. `Keras` objekt *Model* také nabízí metodu *evaluate()*, která ohodnotí natrénování modelu pro danou testovací množinu. Tato metoda vrací metriky specifikované při sestavování modelu a hodnotu ztrátové funkce. Případná cross-validace musí být implementována uživatelem ručně, protože `Keras` pro ni neposkytuje žádné vestavěné funkce [12].

3.3.3 `PyTorch`

`PyTorch` je Python balíček poskytující vysokoúrovňové nástroje pro práci s neuronovými sítěmi. `Pytorch` se skládá z těchto základních komponent [9]:

- **torch** - obsahuje datové struktury pro více-dimenzionální tensory a matematické operace nad nimi podobně jako v knihovně `NumPy`. Oproti ní ale umožňuje akceleraci výpočtu na GPU.
- **torch.autograd** - poskytuje nástroje pro automatickou diferenciaci libovolných diferencovatelných funkcí.
- **torch.jit** - definuje *TorchScript*, který umožňuje exportovat modely vytvořené v `PyTorch` tak, aby je bylo možné spouštět v jiných prostředích, například v C++.
- **torch.nn** - poskytuje všechny potřebné nástroje pro tvorbu neuronových sítí. Je úzce spjata s balíčkem `autograd` pro maximální flexibilitu.
- **torch.multiprocessing** - umožňuje sdílení paměti `torch` tensorů napříč procesy, což může být využito například pro načítání dat.
- **torch.utils** - obsahuje pomocné funkce pro usnadnění používání knihovny.

Vstupní data mohou být v `PyTorch` reprezentována jako `NumPy` pole nebo jako tensor z balíčku `torch` anebo knihovna poskytuje objekt *Dataset* zapouzdřující vstupní datovou sadu i odpovídajíci zařazení do tříd. Klíčovou strukturou reprezentující vstupní data pro neuronové sítě v `PyTorch` je *DataLoader*, který se vytvoří z definovaného *Datasetu*. *DataLoader* umožňuje

iterovat nad vstupními daty. Při vytváření *DataLoaderu* lze zvolit, zda budou v jednotlivých iteracích trénovací data zamíchána, anebo zvolit jeden z implementovaných *Samplerů*, který bude míchání trénovacích dat po každé iteraci řídit. Můžeme také definovat, kolik subprocesů bude použito pro načítání dat, anebo zda bude model trénovaný po dávkách.

V knihovně `PyTorch` balíček `nn` definuje sadu prostředků datového typu *Module*, který reprezentuje vrstvy neuronové sítě. *Module* stejně jako v `Kerasu` přijímá vstupní tensor a transformuje jej nějakým způsobem na výstupní. Může také udržovat svůj vnitřní stav ve formě svých naučitelných parametrů. Balíček `nn` dále poskytuje sadu předdefinovaných běžně používaných ztrátových funkcí.

Základním stavebním kamenem modelů v `PyTorch` je třída *nn.Sequential* reprezentující sekvenční posloupnost vrstev. Je typu *Module* a zastřešuje ostatní *Moduly* reprezentující vrstvy, které se do něj vkládají. `PyTorch` poskytuje velké množství předdefinovaných parametrizovaných vrstev. Stejně jako v `Kerasu` jsou i v `PyTorch` implementované optimalizátory.

Na rozdíl od `Kerasu` ale proces učení není definovaný jedinou parametrizovanou funkcí. Zde je rozložen do jednotlivých kroků, které se musejí postupně seskládat podle postupu algoritmu trénování neuronové sítě. To znamená, že v jedné učící iteraci je zapotřebí zjistit zařazení trénovacích vzorků do tříd. Z těchto nově získaných odpovědí porovnáním se správnými vypočítat aktuální hodnotu ztrátové funkce. Dále je nutné vynulovat hodnoty aktuálních gradientů uchovávaných optimalizátorem pomocí funkce *zero_grad()* z balíku `torch.optim`, protože gradienty jsou akumulovány v bufferu, nikoliv přepisovány. Poté se vypočítá gradient ztrátové funkce pomocí metody *backward()* a pomocí metody *step()* z balíku `torch.optim` se aktualizují parametry optimalizéru. Analogicky se implementuje proces evaluace akorát s absencí aktualizace parametrů optimalizéru [9].

3.4 Současný trend používání DL metod ve zpracování EEG signálu

Při použití standardního postupu analýzy EEG dat ve formě předzpracování, extrakce příznaků a klasifikace běžnými klasifikátory se výzkumníci běžně museli vypořádávat s několika problémy. Pro některé z nich, jako například pro odstranění šumu z EEG signálu, se jim již podařilo nalézt efektivní a ověřená řešení. Ale EEG je nestacionární signál, jenž je v čase velmi proměnlivý, což může mít za následek to, že klasifikátor natrénovaný na omezené množině dat může dosahovat horších výsledků na datech naměřených v jinou dobu

na stejném jedinci. EEG také vykazuje vysoké rozdíly mezi subjekty kvůli jejich fyziologickým odlišnostem, což může ovlivnit úspěšnost klasifikačních modelů, které by se měly aplikovat na více jedinců. Schopnost zobecnit data naměřená na jedné sadě subjektů na druhou, neznámou sadu jedinců, je základem mnoha praktických využití EEG. K překonání těchto problémů jsou zapotřebí nové přístupy ke zlepšení zpracování EEG směrem k lepším generalizačním schopnostem a flexibilnějším aplikacím, které by mohly poskytovat metody hlubokého učení. Mohla by aplikace metod hlubokého učení dokázat i zredukovat počet často časově a výpočetně náročných kroků předzpracování [35]?

Podle [14] není v současné době zatím ověřeno, zda metody hlubokého učení mohou dosáhnout lepších výsledků bez jakéhokoliv předzpracování. Jejich ale poněkud překvapivým zjištěním je, že u více než čtvrtiny zkoumaných experimentů byly artefakty odstraněny manuálně, což kromě časové náročnosti také ztěžuje případnou reprodukci postupů.

Ve studii publikované v [35] zmiňují dva články, ve kterých se podařilo dosáhnout lepších výsledků s nepředzpracovanými daty než s předzpracovanými, ale i přesto tato studie považuje tuto otázku za zatím nevyřešenou.

Extrakce příznaků je jedním z nejnáročnějších kroků tradičního postupu zpracování EEG signálu. Zbavit se ho využíváním hlubokých neuronových sítí je hlavním cílem mnoha experimentů. Podle několika studií lze dosáhnout kvalitních výsledků i bez potřeby extrakce příznaků.

Neuronové sítě mají schopnost naučit se složité vzorce ve velkém množství dat, což motivuje vědce k použití hodnot signálu jako vstupů do neuronových sítí bez použití extrakce příznaků. Například jako vstup do konvolučních neuronových sítí, jejichž výkon se osvědčil hlavně v poli rozpoznávání obrazových dat, se používají spektrogramy, které se běžně využívají pro vizualizaci EEG dat. Dále ale bylo zjištěno, že konvoluční neuronové sítě, jež pro vstup používají čistě hodnoty signálu, dosahovaly v průměru lepších výsledků než se spektrogramy nebo vypočítanými příznaky. Toto zjištění je v rozporu s myšlenkou, že extrakce příznaků je důležitým krokem ke zlepšení úspěšnosti klasifikace [14].

Zásadním rozhodnutím při aplikaci deep-learningových metod je správný výběr architektury použité neuronové sítě. Pro klasifikaci ERP je suverénně nejpoužívanější architekturou konvoluční neuronová síť následovaná rekurentní neuronovou sítí. Od roku 2015 došlo k velkému nárůstu používání CNN, oproti očekáváním, že vzhledem k časové struktuře EEG budou RNN využívány více než modely, které nezohledňují časové závislosti. Rozmach používání CNN pro klasifikaci EEG dat lze zdůvodnit jednak dosaženým úspěchem v počítačovém vidění a využíváním hierarchické struktury na da-

tech, tak i nedávnými diskuzemi a zjištěními týkajícími se účinnosti CNN pro zpracování časových řad. Ovšem četnost využívání RNN také stoupá [35].

Kromě volby architektury neuronové sítě musí výzkumník učinit rozhodnutí o počtu vrstev. Dle [35] většina projektů používala pro své modely méně než 5 vrstev, a tudíž jejich závěrem je, že pro EEG data jsou v současné době vhodnější mělčí neuronové sítě. V [39] se přímo věnovali problematice počtu vrstev konvoluční neuronové sítě pro EEG data a dospěli k závěru, že mělčí CNN v úspěšnosti překonaly hlubší CNN.

Zásadní otázkou je, zda využití deep-learningových metod vede k lepším výsledkům než při použití tradičních metod strojového učení v EEG? Odpověď není jednoznačná, protože datové soubory, výkonnostní metriky, tradiční modely se mezi studiemi značně liší. V tradičním postupu klasifikace EEG dat se obvykle používají metody pro extrakci příznaků a jednodušší (klasické) metody strojového učení - LDA, SVM. Téměř u všech experimentů procházených v [35], které se zabývaly porovnáváním úspěšností metod hlubokého učení oproti klasickým metodám strojového učení, došlo při použití hlubokého učení ke zlepšení dosažených výsledků.

4 Nástroje pro analýzu EEG

Součástí standardního postupu analýzy EEG dat je před samotnou klasifikací jejich načtení a předzpracování. V současné chvíli existuje poměrně velké množství různých nástrojů, které tyto kroky umožňují. Tyto nástroje se mezi sebou liší v mnoha ohledech, jako například poskytovanou množinou metod na zpracování dat, nebo datovými formáty, které umějí číst nebo do nich zapisovat. Některé analyzátory poskytují uživatelské rozhraní, jiné jsou ve formátu knihoven. Rozdíly existují v programovacích jazycích, ve kterých jsou implementovány. Nástroje se také liší ve schopnostech integrace s okolním prostředím, s čímž souvisí i možnosti využití metod strojového učení. V neuroinformatické laboratoři FAV ZČU se data z provedených experimentů ukládají ve formátu `BrainVision`, tudíž budou v této práci analyzovány hlavně ty analytické nástroje, které umějí načítat data v tomto formátu.

4.1 `Brainstorm`

`Brainstorm` je open source aplikace pro analýzu mozkových záznamů ve formátech MEG, EEG, fNIRS, ECoG. Cílem tohoto nástroje je sdílet množinu uživatelsky přívětivých nástrojů s vědeckou komunitou, která experimentuje s MEG/EEG daty. Jednou ze zásadních vlastností `Brainstormu` je intuitivní a bohaté uživatelské rozhraní, díky němuž mohou tento nástroj využívat výzkumníci bez znalostí programování. `Brainstorm` je vyvíjen v MATLABu, ovšem uživatelé nejsou povinni vlastnit jeho licenci. `Brainstorm` poskytuje spustitelnou verzi pro Windows, MacOS i Linux zdarma ke stažení.

`Brainstorm` podporuje načítání dat z velkého množství datových formátů zahrnujících `BrainVision`, `European Data Format` nebo i `NWB`. Brainstorm také umožňuje importovat data, která byla již nějak zpracována v jiných analytických nástrojích jako například v `EEGLab` a `FieldTrip`. MATLAB nyní poskytuje možnost integrace Python knihoven tak, že se jejich funkce dají volat přímo z MATLABu, což umožňuje přímé využití metod např. z knihovny `MNE` (popsána v 4.6). Díky této vlastnosti lze na zpracovaná data využít metody, které nejsou v `Brainstormu` implementovány [40].

`Brainstorm` poskytuje celou řadu metod pro zpracování EEG dat jako například:

- automatická detekce artefaktů,
- extrakce epoch,

- frekvenční filtrování,
- časově-frekvenční transformace,
- identifikace špatných kanálů.

Zpracovávající postup v `Brainstormu` má pluginovou strukturu, tudíž lze rozšířit jeho funkčnost o uživatelské procesy zpracování. Při psaní pluginu je nutné dodržet požadovanou strukturu a vložit jej do složky projektu. Při úspěšném splnění všech podmínek je poté plugin automaticky rozeznán a přidán do uživatelského rozhraní.

`Brainstorm` je udržovaný a neustále se vyvíjející nástroj s poměrně rozsáhlou komunitou. Na svých webových stránkách má publikováno mnoho tutoriálů, jež pokrývají velké množství use-cases pro zpracování dat v různých formátech. Jednou z výhod nástroje je i fórum, na kterém sami vývojáři aktivně odpovídají na dotazy uživatelů. Intuitivní a bohaté uživatelské rozhraní usnadňuje využití nástroje pro neinformatické uživatele.

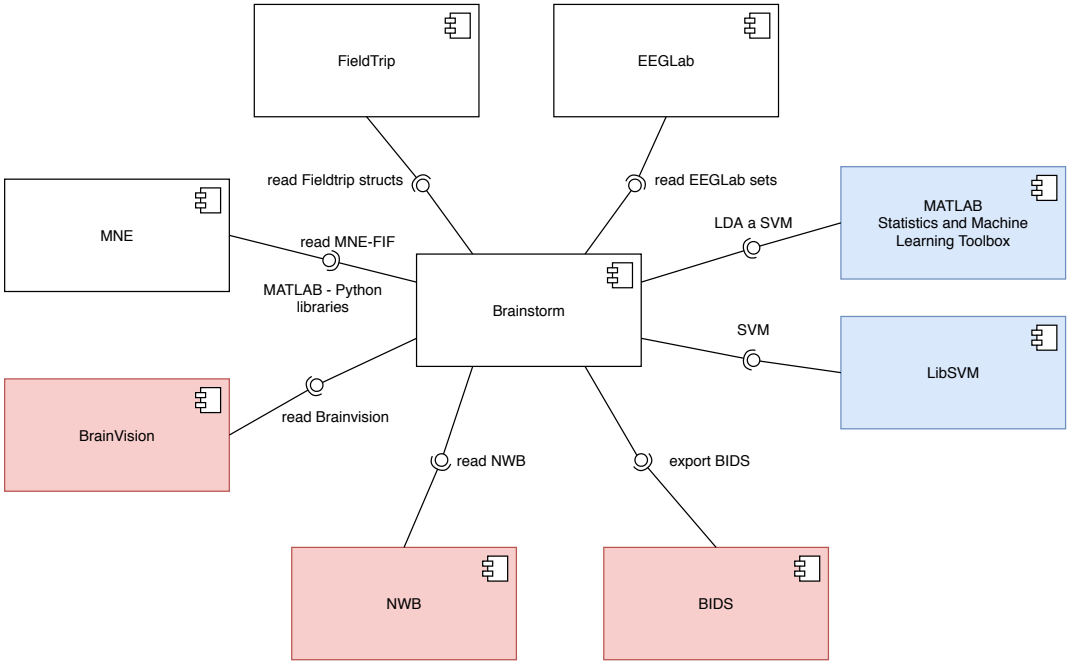
4.1.1 Strojové učení

Z hlediska aplikace metod strojového učení ovšem tento nástroj nenabízí mnoho využití. Pouze dva druhy binárních klasifikátorů, a z toho jen jeden neplacený, nedávají uživateli mnoho šancí kvalitně klasifikovat EEG data.

Po nahrání dat do nástroje lze v uživatelském rozhraní definovat zpracovávající pipeline, ve které uživatel může také vybrat krok klasifikace. V klasifikaci dále zvolit, zda se budou trénovací a testovací vzorky vybírat náhodně, nebo bude použita cross-validace. Poté je nutné vybrat klasifikátor. `Brainstorm` dává na výběr z SVM a LDA z MATLABu a SVM z knihovny `LibSVM`. Použití klasifikátorů z MATLABu vyžaduje placený `Statistics and Machine Learning Toolbox`. Nástroj `LibSVM` je ke stažení zdarma.

V plánech dalšího vývoje `Brainstormu` ovšem lze nalézt, že budou vývojáři nástroje klasifikačním metodám strojového učení věnovat v dalším vývoji větší pozornost. Dle plánu dojde k implementaci klasifikátorů z knihovny `scikit-learn`.

Na obrázku č. 4.1 je formou diagramu komponent zobrazeno, s jakými datovými formáty, analytickými nástroji a knihovnami pro strojové učení umí `Brainstorm` spolupracovat.



Obrázek 4.1: Integrace nástroje `Brainstorm` s datovými formáty, analytickými nástroji a knihovnami strojového učení

4.2 `FieldTrip`

`FieldTrip` je MATLAB toolbox pro analýzu MEG, EEG, iEEG a fNIRS dat, který obsahuje množinu metod pro předzpracování a pokročilejší analýzu dat. Podporuje datové formáty `BrainVision`, `BIDS` i `NWB` a formát používaný v nástroji `EEGLab` (popsán dále v této kapitole). Načítací proceduru ve `FieldTripu` lze snadno modifikovat tak, aby dokázala načíst i data z jiných než podporovaných formátů. Toolbox lze stáhnout zdarma z oficiálních stránek a přidat jej do lokální MATLAB instalace. `FieldTrip` nemá grafické uživatelské rozhraní a metody poskytované `FieldTripem` lze využít pro vytváření analytických pipeline v MATLABu. Funkce, které `FieldTrip` nabízí, lze rozdělit do sedmi kategorií:

1. funkce pro čtení a předzpracování,
2. funkce pro analýzu ERP,
3. funkce pro frekvenční a časově-frekvenční analýzu,
4. funkce pro analýzu zdrojů,
5. funkce pro statistickou analýzu,

6. funkce pro vykreslování a zobrazování dat,

7. funkce pro analýzu v reálném čase.

`FieldTrip` používá velmi podobné zpracovávající pipeline jako `MNE` a datový formát používaný třídou *Raw* z `MNE` (popsána v 4.6) odpovídá datovému formátu v objektu *ft_datatype_raw*. Import a export mezi těmito dvěma nástroji a objekty lze provést pomocí *.fif* souboru podporovaného oběma nástroji [30].

4.2.1 Strojové učení

Pro strojové učení má `FieldTrip` připravenou integraci s dvěma externími MATLAB toolboxy `Donders Machine Learning Toolbox (DMLT)` a `MVPA-Light`.

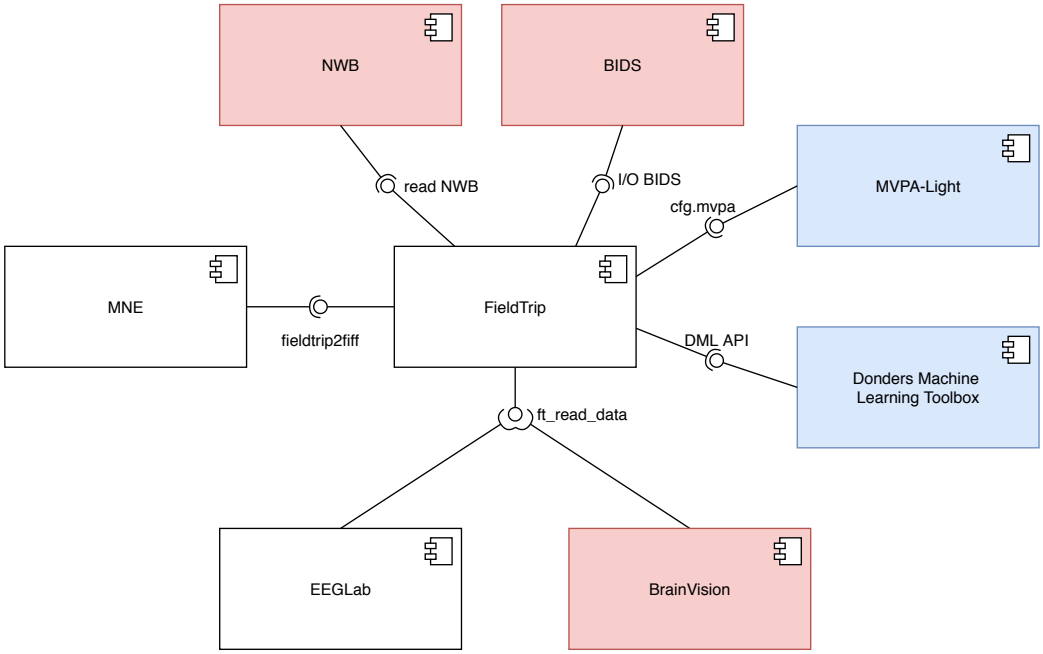
`FieldTrip` poskytuje ukázky, jakým způsobem používat a volat funkce a klasifikátory z `DMLT` a zaintegrovat je do pipeline používané ve `FieldTrip`. `DMLT` nabízí velké množství binárních i diskrétních klasifikátorů jako například [43]:

- SVM,
- Naivní Bayesovský klasifikátor,
- Regularizovaná diskriminační analýza.

`FieldTrip` obsahuje vysokoúrovňové API pro funkce z toolboxu `MVPA-Light`. `MVPA-Light` toolbox nabízí podobnou množinu základních klasifikátorů jako `DMLT`. `MVPA-Light` je ale objemnější, protože obsahuje například i metody pro extrakci příznaků. Jeho syntaxe je velmi podobná `FieldTripu`.

`FieldTrip` poskytuje pomocí integrací s externími knihovnami již větší možnosti využití metod strojového učení na zpracovávaná EEG data. Jedná se o velký projekt, který je neustále vyvíjen. Je velmi dobře dokumentován, dokumentace obsahuje velké množství podrobně komentovaných tutoriálů. Vývojáři toolboxu pořádají konference a workshopy, takže je zřejmě komunita jeho uživatelů velká. Pro naše použití je nevýhodou implementace nástroje v MATLABu, a tudíž nutnost vlastnění jeho licence.

Na obrázku č. 4.2 je formou diagramu komponent zobrazeno, s jakými datovými formáty, analytickými nástroji a knihovnami pro strojové učení umí `FieldTrip` spolupracovat.



Obrázek 4.2: Integrace nástroje `FieldTrip` s datovými formáty, analytickými nástroji a knihovnami strojového učení

4.3 `EEGLab`

`EEGLab` je nástroj psaný v MATLABu pro zpracování EEG, MEG a dalších typů elektrofyziologických dat. Poskytuje uživatelské rozhraní, které uživatelům umožňuje interaktivně pracovat s EEG daty. Součástí jeho dokumentace je velké množství tutoriálů, které usnadňují seznámení s nástrojem. `EEGLab` nabízí řadu možností analýzy elektrofyziologických dat podporujících odstranění artefaktů, frekvenční analýzu, vizualizaci, modelování a další.

`EEGLab` je vyvíjen jako open source projekt a umožňuje uživatelům rozšiřovat možnosti nástroje pomocí implementace vlastních pluginů, a tím tak sdílet své poznatky s EEG komunitou. `EEGLab` lze spouštět na všech platformách, ovšem je nutné vlastnit MATLAB licenci. Pro využití funkčnosti některých z rozšiřovacích pluginů nástroje `EEGLab` je vyžadována instalace MATLAB toolboxů.

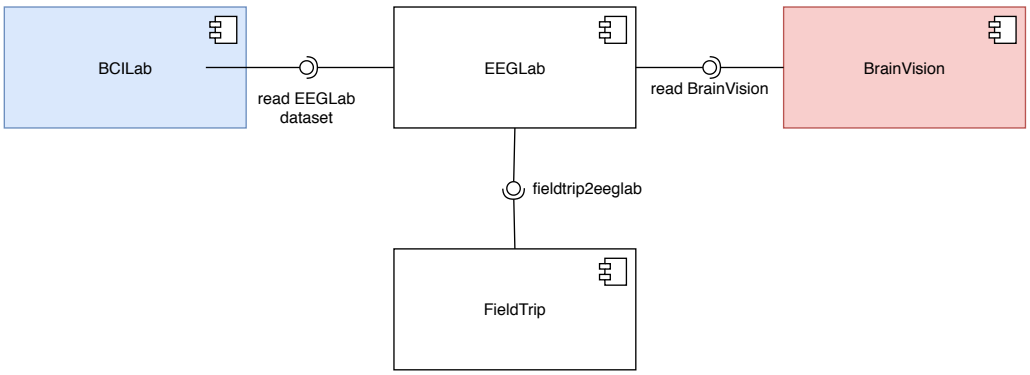
`EEGLab` poskytuje základní metody předzpracování jako filtrování, převzorkování, extrakce epoch, korekce baseline nebo časove-frekvenční transformaci. Nabízí několik postupů pro odstranění artefaktů ze zašuměného signálu, jako například pomocí statistik jednotlivých kanálů nebo podle amplitudy.

`EEGLab` podporuje načítání formátu `BrainVision` a `FieldTrip` [15].

4.3.1 Strojové učení

`BCILab` je open source MATLAB toolbox, který umožňuje použít klasifikační algoritmy strojového učení na data zpracovávaná v `EEGLab`. Tyto dva nástroje jsou spolu úzce spjaty použitím stejné hlavní datové struktury. `BCILab` poskytuje grafické uživatelské rozhraní obsahující téměř veškerou funkcionalitu nástroje. Veškeré metody `BCILabu` je také možné použít externě pro psaní MATLAB skriptů. `BCILab` poskytuje funkce pro zpracování signálu, což je řešeno hlavně různými filtračními algoritmy, funkce pro extrakci příznaků jako například CSP nebo průměrování oken. `BCILab` obsahuje implementace poměrně velkého množství metod strojového učení zahrnující SVM, LDA nebo například lineární regresi. `BCILab` neobsahuje žádné implementace neuronových sítí ani neobsahuje žádné nástroje umožňující jejich tvorbu a následné použití [23].

Na obrázku č. 4.3 je formou diagramu komponent zobrazeno, s jakými datovými formáty, analytickými nástroji a knihovnami pro strojové učení umí `EEGLab` spolupracovat.



Obrázek 4.3: Integrace nástroje EEGLab s datovými formáty, analytickými nástroji a knihovnami strojového učení

4.4 `Neo`

`Neo` je Python balíček pro práci s elektrofyziologickými daty, jenž podporuje čtení a zápis do širokého spektra neurofyziologických formátů. Motivací pro vznik `Neo` bylo zvýšení interoperability mezi Python nástroji pro analýzu, vizualizaci a generování elektrofyziologických dat poskytnutím společného objektového modelu. `Neo` je omezen čistě na reprezentaci dat a neposkytuje funkce pro jejich analýzu nebo vizualizaci [37].

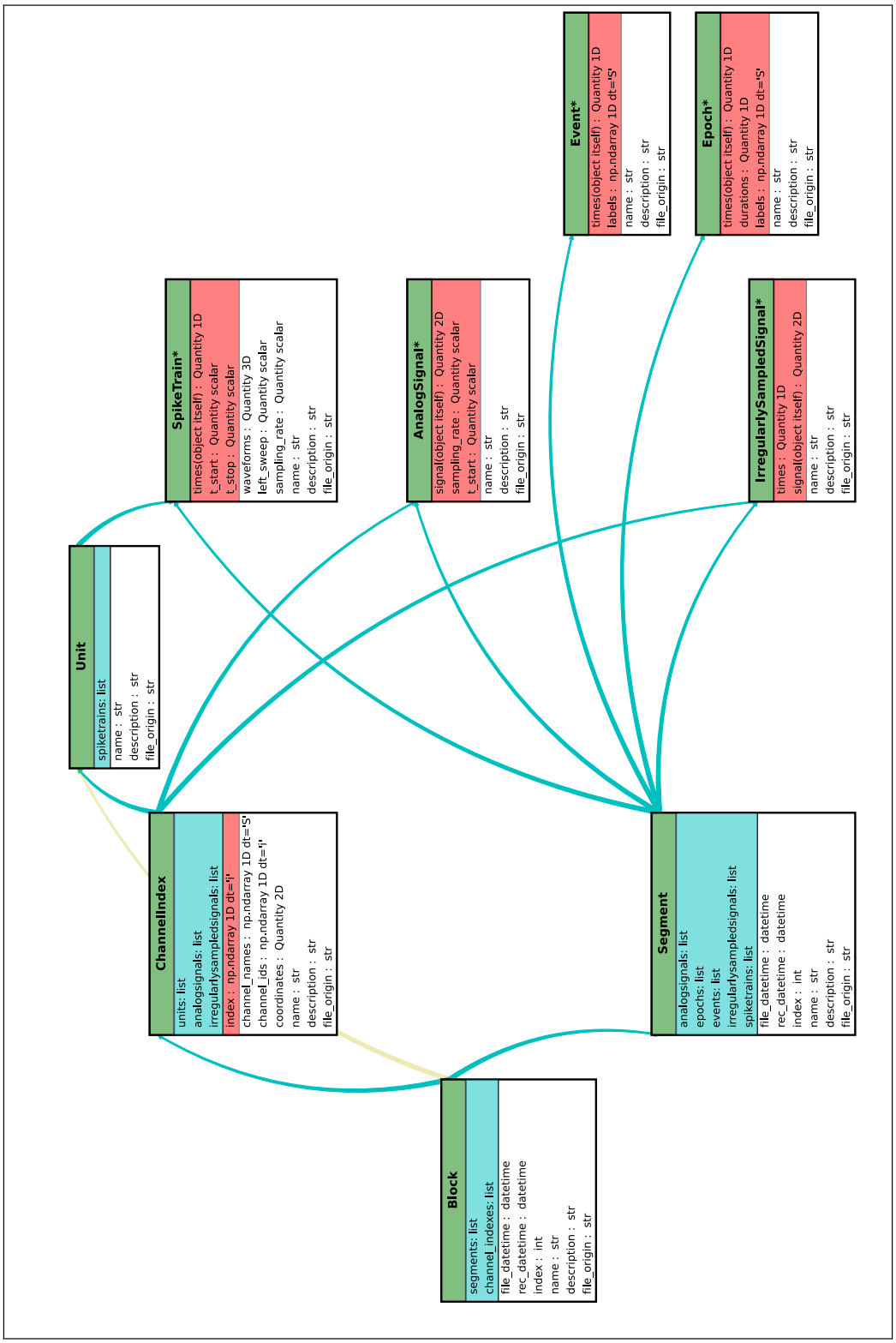
4.4.1 Datový model

`Neo` implementuje hierarchický datový model. Datové objekty jsou stavěny na balíčku *Quantity*, jenž je odvozený od knihovny `NumPy`.

`Neo` objekty lze rozdělit do 3 kategorií [37]:

- **Datové objekty**
  - *AnalogSignal* - pravidelně vzorkovaný jedno nebo vícekanálový kontinuální analogový signál,
  - *IrregularlySampledSignal* - nepravidelně vzorkovaný jedno nebo vícekanálový kontinuální analogový signál,
  - *SpikeTrain* - množina akčních potenciálů reprezentovaných stejnou jednotkou v časovém úseku,
  - *Event* - pole časových značek představujících jednu nebo více událostí v datech,
  - *Epoch* - pole časových intervalů představujících jeden nebo více časových úseků v datech.
- **Kontejnerové objekty**
  - *Segment* - kontejner pro heterogenní, diskrétní nebo kontinuální data sdílející společné hodiny, ale ne nutně se stejnou vzorkovací frekvencí, časem zahájení a časem ukončení. Může obsahovat jakýkoli datový objekt. V kontextu daného experimentu lze segment považovat za "pokus", "běh", "záznam"atd.
  - *Block* - top-level kontejner shromažďující všechna data pro danou relaci záznamu.
- **Seskupující/Propojující objekty**
  - *ChannelIndex* - sada indexů do *AnalogSignal* objektů představujících záznamové kanály,
  - *Unit* - propojuje *SpikeTrain* objekty v *Block* pravděpodobně napříč několika *Segmenty*, které byly vysílány stejným neuronem. *Unit* je spojena s *ChannelIndex* objektem, na kterém byly akční potenciály detekovány.

Digram datového modelu `Neo` je zobrazen na obrázku č. 4.4. Objekty označené hvězdičkou dědí objekt *Quantity*. Atributy označené červeně jsou povinné a ty neoznačené jsou doporučené. Tyrkysové konektory představují vztah "one to many".



Obrázek 4.4: Datový formát `Neo`

4.4.2 Nástroje používající Neo

Neo datovou strukturu využívá celá řada různých nástrojů pro analýzu, vizualizaci nebo simulaci dat, jako například:

- SpykeViewer,
- Elephant,
- PyNN,
- EphyViewer.

SpykeViewer

SpykeViewer je multiplatformní aplikace s uživatelským rozhraním pro vizualizaci elektrofyziologických dat. Je založená na knihovně Neo, což jí umožňuje načítat širokou škálu datových formátů používaných v elektrofyziologii. SpykeViewer je implementován jako sada pluginů a má zabudovanou také Python konzoli, takže se jeho funkcionalita dá rozšiřovat. Aplikace je dostupná se základní množinou pluginů, které umožňují jen vykreslovat načtená data. Obsahuje plugin na vykreslení signálů, spektogramů, průběhů akčních potenciálů a dalších. Neobsahuje žádné analytické pluginy. V dokumentaci je uživatel nabádán k vytvoření vlastních analytických pluginů. Vývoj této aplikace byl již ukončen, poslední commit proběhl na jaře 2016 [33].

Elephant

Elephant (Electrophysiology Analysis Toolkit) je open source knihovna pro analýzu elektrofyziologických dat v Pythonu. Elephant se zaměřuje na generické analytické funkce pro akční potenciály a záznamy časových řad z elektrod, jakými jsou potenciály lokálního pole (local field potentials - LFP) nebo intracelulární napětí. Cílem projektu Elephant je kromě společné platformy pro analytické kódy z různých laboratoří poskytnout konzistentní a homogenní rámec pro analýzu, který je postaven na modulárním základu. Umožňuje načítat i zapisovat do datového formátu Neo [1].

PyNN

PyNN je jazyk pro vytváření neuronových spiking sítí nezávislý na simulátoru. Cílem tohoto projektu je umožnit napsání kódu pro simulační model jen jednou, a pak jej spustit na jakémkoliv simulátoru, který PyNN podporuje. PyNN poskytuje knihovnu standardních modelů neuronů, synapsí a synaptických

plasticit, u kterých bylo ověřeno, že fungují stejně na různých podporovaných simulátorech. `PyNN` také poskytuje sadu běžně používaných algoritmů konektivity [8].

`EphyViewer`

`EphyViewer` je knihovna v Pythonu pro vizualizaci elektrofyziologických dat. Podporuje zobrazování všech možných reprezentací daného datasetu (signál, epochy, události, akční potenciály). Kromě knihovny poskytuje `EphyViewer` i samostatnou aplikaci s uživatelským rozhraním, která umožňuje zobrazovat data z datového formátu `Neo` [17].

4.4.3 Příklady podporovaných formátů

Jak již bylo zmíněno, výsadou knihovny `Neo` je schopnost načítat i zapisovat data v široké škále formátů. Pro tuto schopnost je v `Neo` vyčleněn speciální I/O modul. Ten definuje jednoduché I/O API, které vyhovuje mnoha formátovým specifikacím. Základní myšlenkou I/O API je mít metody čtení a zápisu pro jednotlivé zapouzdřené objekty, a také metody čtení a zápisu, které vrátí objekt top-level hierarchie (*Block* nebo *Segment*). Vývojáři `Neo` také vytvořili podrobný návod, jak implementovat I/O pro nový datový formát.

Podporovanými formáty `Neo` jsou i například `Nix` a `BrainVision`.

`BrainVision` formát

Pro načítání dat v `BrainVision` formátu je v knihovně `Neo` připravena třída *neo.io.BrainVisionIO*, která jako parametr bere soubor *.vhdr* a při využití některé poskytované načítací funkce přetransformuje data z `BrainVision` formátu do `Neo` datového formátu. Z jednoho záznamu *.vhdr* se vytvoří jeden *Block* a pro něj jeden *Segment*. V něm se pro každý kanál vygeneruje objekt *AnalogSignal* a pro události se vytvoří objekty typu *Event*.

`Nix` formát

`Neo` také dokáže zapisovat a číst data v `Nix` formátu. Třídu *neo.io.NixIO* zajišťující tuto schopnost vytvořili vývojáři formátu `Nix`. Při čtení a zápisu `Nix` z `Neo` je třeba dbát přesných mapovacích konvencí, jež jsou popsané v [24]. Třída poskytuje funkci na zapsání `Neo` *Bloků* do `Nix`, která se o správné mapování postará.

4.4.4 Strojové učení

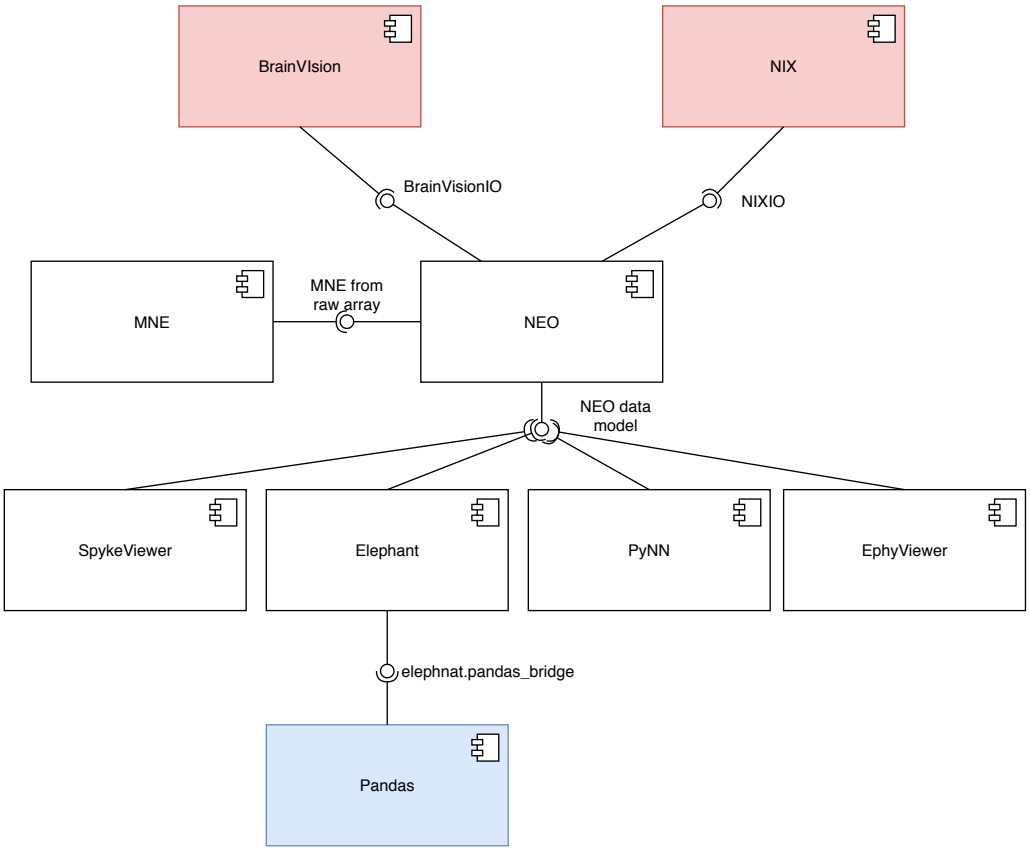
Při analýze knihovny `Neo` nebyla nalezena žádná připravená možnost využití klasifikačních metod strojového učení. Nenašel jsem ani žádný veřejně přístupný use-case. Byla objevena jediná možná připravená varianta, jak aplikovat metody strojového učení na EEG data uložená v `Neo`, a to cestou přes knihovnu `Elephant` do knihovny `Pandas`, která se používá pro předzpracování dat. `Elephant` má implementovaný modul jménem *Bridge to the pandas library*. Tento modul obsahuje funkce pro export `Neo` objektů do `Pandas` *DataFrame*, a to konkrétně epochy, události a akční potenciály. Problémem v tomto případě je, že objekt *Epoch* v `Neo` obsahuje pouze časové značky začátků jednotlivých epoch, jejich dobu trvání a název nebo označení daných epoch. Objekt *Epoch* v `Neo` v sobě nenese přímo naměřené hodnoty EEG, ale jsou to jen tyto anotace. `Elephant` navíc oznámil, že v následující verzi bude *Pandas Bridge* odstraněn.

`Neo` ovšem poskytuje funkce pro manipulaci s objekty svého datového formátu a při znalosti obojího lze po úspěšném načtení dat z `BrainVision` z datového formátu vyextrahovat data ve tvaru, ve kterém je lze použít jako vstup, například do metod hlubokého učení v knihovně `Keras` nebo dalších metod strojového učení, například z knihovny `scikit-learn`. Jak bylo popsáno výše, při načítání `BrainVision` dat do `Neo` se uloží signál i události, tudíž lze pomocí metod knihovny ze signálu extrahovat epochy v odpovídajícím tvaru pro vstup do metod strojového učení. Jak již bylo zmíněno, objekt *Epoch* je pouze anotační struktura a neobsahuje žádná data, ale knihovna `Neo` poskytuje funkci *utils.cut_segment_by_epoch()*, který daný segment rozdělí podle daných časových značek. Získáme tedy počet segmentů rovný počtu epoch, ve kterém každý segment obsahuje analogové signály dle kanálů, epochu popisující časovou značku a trvání, ale také obsahuje událost, která určuje stimul epochy. Z této množiny segmentů lze pak získat data v takovém tvaru, aby odpovídala vstupu do metod strojového učení.

Zjistil jsem tedy, že pro použití metod strojového učení na EEG data zatím není v knihovně `Neo` připravena žádná integrace. Zároveň jsem ověřil, že se z vnitřního datového formátu `Neo` dají data extrahovat do takové podoby, aby se dala využít pro machine-learningové metody. Jedná se ovšem o poněkud náročnější úlohu, pro kterou je nutná podrobnější znalost nástroje. Nevýhodou této knihovny také je, že je zaměřená pouze na reprezentaci dat a neposkytuje žádné metody pro analýzu případně pro předzpracování EEG dat. Pro tyto úkony je potřeba využít některý z popsaných nástrojů podporujících `Neo`.

Na obrázku č. 4.5 je formou diagramu komponent zobrazeno, s jakými

datovými formáty, analytickými nástroji a knihovnami pro strojové učení umí `Neo` spolupracovat.



Obrázek 4.5: Integrace nástroje `Neo` s datovými formáty, analytickými nástroji a knihovnami strojového učení

4.5 `Pandas`

`Pandas` je Python knihovna, která poskytuje rychlé a flexibilní datové struktury navržené pro snadnou a intuitivní práci s různými druhy dat. Cílem tohoto balíčku je být základním high-level stavebním kamenem pro provádění praktické analýzy dat. Širším cílem tohoto projektu je stát se nejsilnějším a nejflexibilnějším open source nástrojem pro analýzu dat a manipulaci s nimi.

`Pandas` je dobře využitelný pro mnoho různých druhů dat jako jsou například:

- tabulková data, jako jsou SQL nebo Excel,
- uspořádané nebo neuspořádané časové řady,

- libovolná maticová data s popisy řádků a sloupců,
- jakákoli jiná forma statistických datových souborů.

`Pandas` využívá svoje dvě primární datové struktury, *Series* pro jednodimenzionální a *DataFrame* pro dvoudimenzionální data. `Pandas` je postavený na knihovně `NumPy`, tudíž je dobře integrovatelný do vědeckého výpočetního prostředí s mnoha dalšími knihovnami jako například `MNE`. Do *DataFrame* umí data exportovat i formáty `NIX`, `BIDS` a `NWB`, což také ukazuje univerzálnost použití struktury *DataFrame*. Pro časové řady poskytuje metody na převzorkování, frekvenční transformaci a další statistické metody [41].

4.5.1 Strojové učení

Z vnitřní struktury `Pandas` *DataFrame* lze poté relativně snadno extrahovat data v takovém formátu, aby odpovídala vstupům do klasifikačních metod strojového učení. Takovýmto případem může být příklad, kde sloupečky odpovídají kanálům a řádky časovým značkám získaných hodnot. Složitější případ by nastal, kdybychom chtěli signál klasifikovat po epochách a museli bychom v *DataFrame* pro danou skupinu hodnot udržovat i událost, ke které epochy patří.

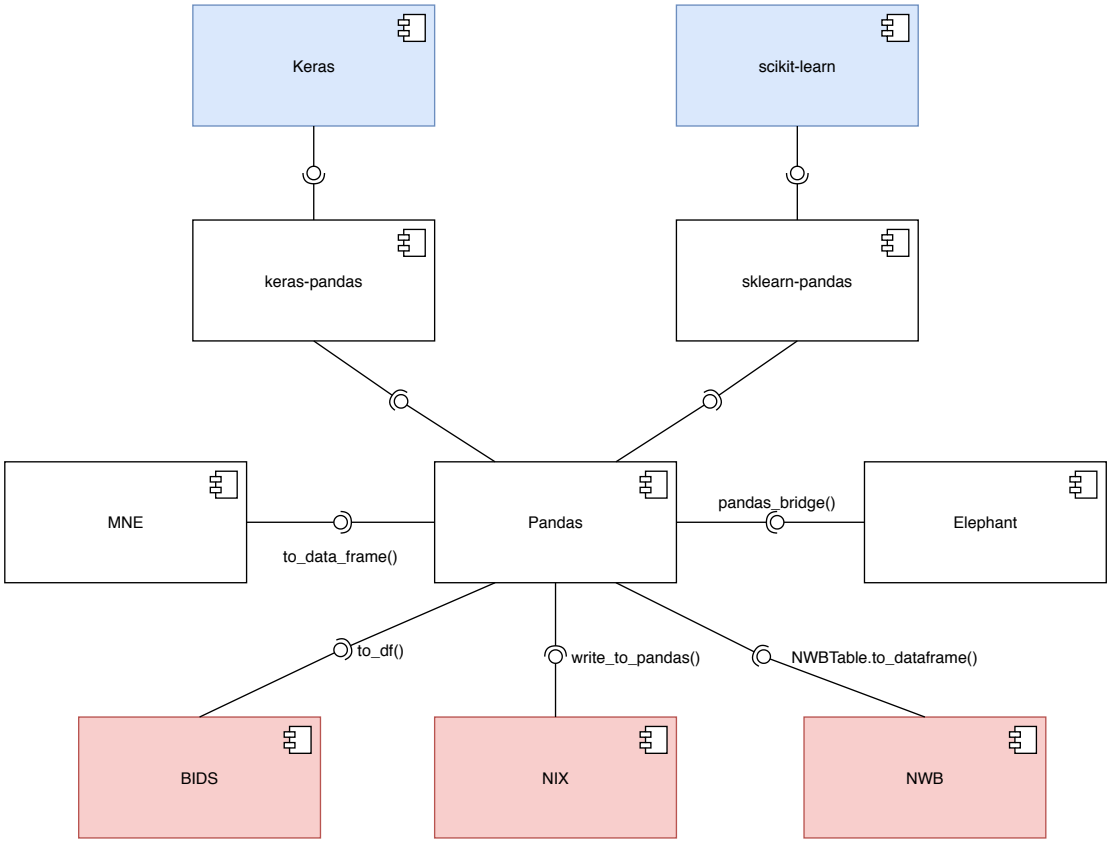
Nepodařilo se mi nalézt mnoho využití `Pandas` pro EEG data, byly nalezeny nějaké use-case pro časové řady z `Pandas` s využitím knihovny `Keras`. Jinak existuje mnoho dalších využití knihovny `Pandas` pro klasifikaci, zřejmě se jedná opravdu o široce využitelný a používaný nástroj.

Jako součást `Pandas` ekosystému byl vytvořen projekt `sklearn-pandas` pro integraci `Pandas` s knihovnou `scikit-learn`. Tento balíček umožňuje zakomponovat `Pandas` *DataFrame* do machine-learningové pipeline v `scikit-learn`. Poskytuje možnosti jak například mapovat sloupce z *DataFrame* do příznaků, a také několik speciálních parserů, které dokáží pracovat s `Pandas` vstupy. Machine-learningové metody z `scikit-learn` jsou ale z `Pandas` poměrně snadno dostupné i bez využití tohoto balíčku.

Dalším integrátorem knihovny `Pandas` a knihovny pro machine-learningové metody, zde konkrétně deep-learningové metody, je projekt `keras-pandas`. Tento balíček poskytuje funkce pro správnou transformaci dat z `Pandas` *DataFrame* do formátu vstupů deep-learningových metod knihovny `Keras`. Hlavním stavebním kamenem balíčku je objekt *Automater*. V tomto objektu se specifikuje, jakého typu jsou sloupce v *DataFrame* (numerické, kategorické, textové atd.), a který sloupec se použije pro klasifikaci. *Automater* se poté postará o transformaci dat z *DataFrame* do vstupního formátu `Keras` metod. Dále tento *Automater* dokáže vytvořit vstupní a výstupní vrstvu modelu.

Obsahuje i základní podporu pro časové řady [20]. Využití tohoto balíčku pro EEG data nebylo nalezeno.

Na obrázku č. 4.6 je formou diagramu komponent zobrazeno, s jakými datovými formáty, analytickými nástroji a knihovnami pro strojové učení umí `Pandas` spolupracovat.



Obrázek 4.6: Integrace nástroje `Pandas` s datovými formáty, analytickými nástroji a knihovnami strojového učení

4.6 `MNE`

`MNE` je open source Python nástroj poskytující nejmodernější algoritmy pro předzpracování dat, lokalizaci zdrojů, statistickou analýzu a odhad funkční konektivity mezi distribuovanými oblastmi mozku. Tvůrci tvrdí, že jsou všechny algoritmy a výkonné funkce implementovány konzistentně a mají dobře dokumentované rozhraní, což umožňuje uživatelům pomocí Python skriptů vytvářet různé pipeline pro analýzu M/EEG dat. `MNE` je integrován s základními Python knihovnami pro vědecké výpočty (`NumPy`, `SciPy`) a vizualizaci (`matplotlib`). Knihovna `NumPy` a její datová struktura n-dimenzionální

pole je v MNE využívána pro efektivní ukládání a manipulaci s daty. Knihovna SciPy se používá hlavně pro operace lineární algebry a zpracování signálu. Při vývoji nástroje spolupracovalo více laboratoří.

Analýza EEG dat v MNE obvykle využívá tři hlavní typy datových kontejnerů, a to *Raw*, *Epochs* a *Evoked*. Prvotní načtená surová data se ukládají do objektu *Raw*. Typickými kroky zpracování v této fázi jsou filtrování, odstranění šumu (odstranění artefaktů), případně ořezávání dat a vizuální kontrola. Všechny tyto úkony jsou podporovány objektem *Raw*. V další části obvykle dochází k extrakci epoch dle stimulů. Epochy jsou uloženy v instanci třídy *Epochs*. Tato instance se vytváří nad specifikovanými *Raw* daty s danou množinou stimulů (událostí) a velikostí časového okna. Objekt *Epochs* opět nabízí další metody zpracování dat. Dále lze epochy zprůměrovat, abychom získali evokované potenciály z jednotlivých kanálů. Zprůměrovaná data jsou uložena v instanci třídy *Evoked*. Všechny tři popsané kontejnery obsahují funkce pro vykreslování v nich uložených dat [18].

MNE obsahuje několik interaktivních vykreslovacích funkcí, které umožňují prohlížet zpracovávaná data, ale jinak MNE nemá implementované uživatelské rozhraní.

MNE podporuje zpracování velkého množství typů dat, jako například EEG, MEG, ECG, SEEG, ECoG a další. Zároveň i čtení ze široké škály formátů elektrofyziologických dat, jako například BrainVision, European Data Format, ale i data ve formátech dalších analytických nástrojů FieldTrip, EEGLab a Brainstorm. MNE neumí pracovat přímo s Neo datovým formátem, ale poskytuje návod, jak data z Neo konvertovat do struktur MNE. MNE rovněž podporuje datové formáty BIDS a NIX.

Ukázka funkcí poskytovaných MNE [18]:

- předzpracování - detekce artefaktů, filtrování, převzorkování,
- extrakce epoch,
- časově-frekvenční transformace,
- průměrování epoch,
- pokročilé metody vizualizace,
- statistické metody,
- metody strojového učení.

MNE je rozsáhlý projekt s velkým množstvím návodů a ukázek použití. Celý nástroj má také velmi dobrou dokumentaci a použití jeho funkcí je snadné.

Vývojářský tým je velmi aktivní a dochází ke stálému vylepšování a rozšiřování funkčnosti nástroje MNE. Mimo to bylo vytvořeno i několik integračních nástrojů jako například MNE-BIDS pro čtení a zápis do datového formátu BIDS nebo například vývojáři publikovali ukázky použití deep-learningové knihovny PyTorch na data zpracovávaná v MNE.

4.6.1 Strojové učení

Tvůrci MNE věnovali integraci klasifikačních metod strojového učení poměrně velkou pozornost. Důkazem je jak balík mne.decoding, tak například i ukázka využití deep-learningových metod z knihovny PyTorch publikována samotnými tvůrci. Dále lze najít mnoho případů použití metod z knihovny Keras na EEG data zpracovávaná knihovnou MNE, a také například projekt Braindecode, který přímo integruje neuronové sítě z PyTorch do MNE. MNE také dokáže exportovat zpracovávaná data do *DataFrame* struktury knihovny Pandas.

Balík mne.decoding

Jak bylo již zmíněno, objekty v MNE uchovávají data ve formátu NumPy polí, což umožňuje jejich snadné využití z dalších vědeckých výpočetních balíčků, včetně použití v pipeline knihovny scikit-learn. Tudíž lze machine-learningové metody z této knihovny aplikovat na data v MNE přímo. Dále ale MNE poskytuje další podporu využití metod strojového učení, a to balík mne.decoding.

Tento balík obsahuje sadu transformačních tříd, které slouží k přeformátování dat z objektů MNE tak, aby odpovídala formátu pipeline scikit-learn knihovny. Prvním z nich je *Scaler*, který se používá pro škálování.

Další transformační třídou je *Vectorizer*, jenž řeší konflikt formátů objektů MNE a formátů metod scikit-learn knihovny. Machine-learningové metody z knihovny scikit-learn obecně očekávají 2D data (počet_epoch x počet_příznaků), zatímco data v MNE jsou většinou uložena v objektech s vyšší dimenzí (počet_epoch x počet_kanálů x počet_hodnot). O převod dat z formátů MNE do formátů scikit-learn se stará právě *Vectorizer*.

Dále balík nabízí třídu *LinearModel*, která jako parametr přijímá instanci lineárního modelu z knihovny scikit-learn a která má stejné rozhraní jako *transformer*, *predictor* a *estimator* z téže knihovny, což umožňuje její instanci vkládat do pipeline knihovny scikit-learn. Oproti použití modelu z knihovny scikit-learn tato třída vypočítává a udržuje prostorové vzory.

Tyto transformační třídy a metody lze díky jejich API odpovídajícímu knihovně scikit-learn různě kombinovat do její pipeline.

Na webu knihovny `MNE` je k dispozici několik návodů a ukázek využití většiny funkcí z balíku `mne.decoding`. V dokumentaci je demonstrována také přímá aplikace metod z knihovny `scikit-learn` na zpracovávaná data v `MNE`.

`MNE-Fatures`

Součástí `MNE` ekosystému je také knihovna s názvem `MNE-Features`, která poskytuje metody pro extrakci příznaků z EEG dat. Tato knihovna obsahuje velké množství funkcí pro extrakci příznaků a dá se velmi snadno integrovat do `MNE` a `scikit-learn` pipeline. Umožňuje také využití implementovaných metod extrakce příznaků pomocí třídy *FeatureExtractor*, která má implementovanou metodu *fit_transform()*, tudíž může být použita jako dílčí krok ve třídě *Pipeline* knihovny `scikit-learn`. Další možností je využití metody *extract_features()*, která jako parametr bere data ve formátu (počet_epoch x počet_kanálů x počet_hodnot), jenž lze snadno získat z `MNE` struktur. Kompletní seznam poskytovaných funkcí extrakce příznaků je v [5]. Funkce lze libovolně kombinovat, skládat a získat tak z dat více příznaků. Všechny funkce mají svůj alias, který se používá pro specifikaci zvolené funkce do extrakční metody nebo třídy [38].

`Pandas`

`MNE` umožňuje exportovat zpracovávaná data do struktury *DataFrame* knihovny `Pandas`. Tuto funkcionalitu poskytuje metoda *to_data_frame()* z balíku `mne.io`, jež může být zavolána nad mnoha objekty v `MNE` jako například *Epochs*, *Evoked* nebo *Raw* a další. Pro všechny objekty tato metoda převede kanály na sloupce v *DataFrame* a v parametrech metody můžeme specifikovat kanály, které mají být do transformace zahrnuté, škálovací hodnoty a například jednotky časových hodnot. Dalším parametrem je index, pomocí kterého lze určit sloupce, jež budou pro data použity jako index. Kdybychom chtěli do `Pandas` *DataFrame* exportovat epochovaná data, tak lze jako index určit sloupce "epoch" představující pořadové číslo epochy, "condition" jako událost popisující epochu, anebo "time" reprezentující časovou značku v epoše. V případě, že se žádný index nespecifikuje, bude jako index v *DataFrame* použit sekvenční celočíselný index knihovny `Pandas`. Jako index lze definovat jednu, kombinaci dvou nebo všechny tři možnosti. V případě volby více než jednoho indexu se v `Pandas` *DataFrame* vytvoří *multiIndex*. Výchozí nastavení parametru index je prázdné. Ukázka formátu vyexportovaného *DataFrame* z epoch `MNE` s indexem definovaným událostí je v tabulce č. 4.1.

signal	epoch	time	Fz	Cz	Pz	EOG	STI 014
condition							
3	0	-200.0	44.081623	55.820594	55.411254	43.178910	0.0
3	0	-199.0	58.013264	70.949501	71.630981	46.024613	0.0
3	0	-198.0	65.738850	79.191688	80.356567	44.016801	0.0
3	0	-197.0	65.915608	79.123329	79.987426	37.169145	0.0
3	0	-196.0	58.370686	70.556922	70.320434	26.243363	0.0
3	0	-195.0	45.238850	55.888954	54.005004	13.505082	0.0
3	0	-194.0	29.269123	38.191688	34.427856	1.360551	0.0
3	0	-193.0	12.697834	19.894813	14.228637	-8.627730	0.0
3	0	-192.0	-1.700603	4.012001	-3.392457	-15.071090	0.0
3	0	-191.0	-11.956463	-7.284874	-16.103394	-17.453902	0.0

Tabulka 4.1: Ukázka vytvořeného *DataFrame* s událostí jako indexem

`Keras`

Bylo nalezeno několik open source projektů, které využívaly balík `MNE` pro zpracování EEG dat, a posléze aplikovaly metody hlubokého učení z knihovny `Keras`. Tudíž kombinace použití těchto dvou nástrojů je populární a ověřenou volbou.

V případě, že chceme pro klasifikaci epoch použít rekurentní neuronové sítě z knihovny `Keras`, můžeme je jako vstup použít ve tvaru 3D tensoru (počet_epoch x počet_kanálů x počet_hodnot), v jakém jej uchovává i třída *Epochs*. Konvoluční neuronové sítě jako vstup využívají 4D tensory, tudíž je nutné naše epochy uložené ve 3D tensoru upravit. Musíme tedy rozšířit datový formát o jednotkovou dimenzi do tvaru (počet_epoch x počet_kanálů x počet_hodnot x 1).

`PyTorch`

Tvůrci knihovny `MNE` sami vytvořili a nasdíleli ukázky, jak lze na data zpracovávaná v nástroji `MNE` aplikovat metody hlubokého učení z knihovny `PyTorch`. V knihovně `PyTorch` lze pro reprezentaci vstupních dat do neuronových sítí použít strukturu *Dataset* a v tomto (`mne-torch`) přístupu je zavedena odpovídající struktura *EpochsDataset*, která zabaluje epochy ve formátu 3D pole a k nim odpovídající třídy do struktury knihovny `PyTorch`.

`Braindecode`

`Braindecode` je toolbox umožňující analyzovat EEG data metodami hlubokého učení. V současné chvíli se zaměřuje na konvoluční neuronové sítě. Představuje rozhraní mezi knihovnou `PyTorch` a `MNE`. `Braindecode` obsahuje několik již vytvořených modelů konvolučních sítí jako například EEGNet, Deep4Net nebo ShallowFBCSPNet, které jsou obaleny funkcemi na jejich sestavení, trénování a testování.

Při využití `Braindecode` na EEG data v `MNE` se z dat také extrahují epochy a jejich přiřazení ke třídám. `Braindecode` má vlastní jednoduchý datový kontejner *SignalAndTarget*, do kterého se ukládají jak epochy ve formě 3D pole (počet_epoch x počet_kanálů x počet_vzorků), tak i jejich odpovídající zařazení do tříd. Pro trénování lze ve většině definovaných modelů v `Braindecode` využít funkci z této knihovny *fit()*, která má velmi podobně definované rozhraní jako metoda *fit()* z knihovny `Keras`. Této funkci se předají trénovací a validační data a ona zajistí proběhnutí trénovací smyčky knihovny `PyTorch` a vrátí natrénovaný model. Stejně jako metodu *fit()* poskytuje nad některými modely metodu *evaluate()* k otestování natrénované sítě.

Na obrázku č. 4.7 je formou diagramu komponent zobrazeno, s jakými datovými formáty, analytickými nástroji a knihovnami pro strojové učení umí `MNE` spolupracovat.

Obrázek 4.7: Integrace nástroje `MNE` s datovými formáty, analytickými nástroji a knihovnami strojového učení

4.7 Výsledek analýzy a výběr řešení

V této kapitole byly popsány dostupné nástroje pro analýzu EEG dat v `BrainVision` formátu a jejich současné možnosti využití metod strojového učení.

Dalším bodem zadání této práce je výběr jednoho z těchto nástrojů, do kterého budou následně integrovány klasifikační metody strojového učení.

Prvním analyzovaným nástrojem byl `Brainstorm`, jenž je oblíbený hlavně pro své bohaté a intuitivní uživatelské rozhraní a velké množství analytických procesů pro širokou škálu datových formátů. V této chvíli ale umožňuje využít jen dva lineární klasifikátory, SVM a LDA, a to pouze LDA bez využití licencovaného MATLAB toolboxu. Jelikož je `Brainstorm` distribuován jako

aplikace s uživatelským rozhraním, nedá se využít jako součást pipeline s machine-learningovými knihovnami. Jedinou teoretickou možností integrace dalších klasifikátorů by byla implementace nového pluginu v jazyce MATLAB.

Dalším zkoumaným nástrojem byl `FieldTrip`. Ten poskytuje poměrně rozsáhlou množinu analytických funkcí. Je distribuován jako MATLAB balíček, tudíž jej lze využít v pipeline spolu s ostatními MATLAB knihovnami poskytujícími metody strojového učení. Nástroj obsahuje dvě integrace s externími machine-learningovými knihovnami. Jeho použití vyžaduje MATLAB licenci, tudíž z důvodu požadavku na zdarma dostupný nástroj je pro naše využití nevhodný.

Následně byl analyzován další balík pro práci s elektrofyziologickými daty s názvem `Neo`. Tento projekt je implementován v Pythonu. Hlavním cílem `Neo` není přímo analýza dat, ale zjednodušení přístupu různých, ať už analytických, nebo například vizualizačních nástrojů k elektrofyziologickým datům. `Neo` obsahuje datový model, se kterým umí pracovat mnoho různých analytických nástrojů. Jedinou existující možností integrace s nějakými machine-learningovými knihovnami byla skrze nástroj `Elephant` do knihovny `Pandas`. Implementací této cesty ale bylo zjištěno, že po ní nelze přenést veškerá potřebná data pro klasifikaci, a tudíž je pro náš účel nevhodná. Poté bylo ověřeno, že pomocí vnitřních funkcí knihovny lze data z `Neo` extrahovat ve formátu odpovídajícímu knihovně `Keras`, a proto je možné pro data uložená ve formátu `Neo` využít deep-learningové metody z knihovny `Keras`. Jednalo se ale o poněkud složitý proces získávání jednotlivých hodnot signálu z vnitřních objektů datového modelu. Slabinou tohoto nástroje je orientace pouze na reprezentaci dat, a tím absence procesů předzpracování. Zde by bylo nutné využít nějaký spolupracující analytický nástroj.

`Pandas` je rozsáhlá knihovna pro analýzu dat psaná v Pythonu. Nebyla vytvořena přímo pro analýzu elektrofyziologických dat, ale umožňuje práci s časovými řadami. Nemá metody přímo na předzpracování EEG dat, ale obsahuje statistické metody, z nichž některé lze aplikovat i na EEG data. Data z `Pandas` lze poměrně snadno extrahovat ve formátu machine-learningových knihoven `scikit-learn` nebo `Keras`. Ovšem nejedná se o knihovnu přímo určenou pro analýzu EEG dat, ale některé formáty (`NIX` a `NWB`) a i analyzátory (`MNE`) poskytují funkce pro export dat ve formátu `Pandas` *DataFrame*.

Posledním analyzovaným nástrojem byl `MNE`. Zřejmě se jedná o největší a nejkomplexnější nástroj pro analýzu elektrofyziologických dat v Pythonu. Poskytuje rozsáhlé možnosti analýzy, modelování, vizualizace a další funkcionality pro širokou škálu typů dat, jejíž součástí je i EEG. `MNE` je rozsáhlý projekt, který je stále rozšiřován, dle nalezených zjištění široce používán EEG komunitou. Na vývoji spolupracuje několik laboratoří, tudíž `MNE` re-

flektuje jejich rozmanité požadavky na analýzu EEG dat. Celý nástroj je dobře dokumentovaný, kde dokumentace zahrnuje web s velkým množstvím návodů a ukázek včetně podrobného vysvětlení funkčnosti. Výhodou je také možnost integrace s okolním EEG světem (`MNE-BIDS`, `NIX-MNE`, `MNE-torch`). Nástroj je již integrován s machine-learningovou knihovou `scikit-learn`. Přímá integrace `MNE` s deep-learningovou knihovnou `Keras` neexistuje, ale transformace dat mezi nimi je poměrně jednoduchá a intuitivní. Pro integraci s další deep-learningovou knihovou `PyTorch` byl vytvořen speciální integrační nástroj `Braindecode`.

V tabulce č. 4.2 jsou pomocí několika kritérií hodnoceny analyzované nástroje. Vyjma nutnosti vlastnění licence jsou všechna kritéria posuzována ve 3 stupních. Těmito kritérii jsou:

1. nutnost vlastnění MATLAB licence,
2. úroveň a rozsah dokumentace,
   - Malá - slabý rozsah dokumentace, chybějící komentáře v kódu, žádné tutoriály,
   - Střední - dobře komentované základní stavební kameny knihovny, několik popsaných ukázek použití,
   - Dobrá - velký počet velmi podrobných návodů a ukázek usnadňujících pochopení nástroje, důkladně komentovaný kód,
3. současný stav možností využití klasifikačních metod strojového učení,
   - Malé - nástroj neposkytuje žádné možnosti využití klasifikačních metod strojového učení, nebo jen velmi omezenou množinu (do počtu 3 klasifikátorů),
   - Střední - nástroj umožňuje využít rozsáhlejší množinu klasifikačních metod strojového učení, neobsahuje možnosti pro použití neuronových sítí,
   - Dobré - nástroj poskytuje rozsáhlé možnosti využití klasifikačních metod strojového a hlubokého učení, nebo je integrovaný s knihovnami poskytujícími tyto metody,
4. velikost množiny poskytovaných funkcí pro zpracování EEG signálu,
   - Malé - nástroj neposkytuje žádné funkce pro zpracování EEG signálu,
   - Střední - nástroj poskytuje omezenou množinu základních funkcí pro zpracování EEG signálu

- Dobré - nástroj obsahuje rozsáhlé možnosti zpracování, analýzy a vizualizaci EEG signálu,

5. počet podporovaných formátů a schopnost spolupráce s ostatními nástroji,
   - Malá - nástroj podporuje jen velmi omezenou množinu formátů a není schopný interakce s ostatními nástroji,
   - Střední - nástroj podporuje omezenou množinu formátů a je integrovaný s malou množinou nástrojů (do počtu 3),
   - Dobrá - nástroj podporuje všechny nebo téměř všechny běžně používané formáty EEG dat a je schopný spolupracovat s ostatními nástroji,
6. velikost komunity podle počtu forků projektu na Githubu,
   - Malá - počet forků je menší než 200,
   - Střední - počet forků je větší než 200, ale menší než 1000,
   - Velká - počet forků je větší než 1000,
7. uživatelská přívětivost,
   - Malá - pro použití je nutná hluboká znalost nástroje, některé entity nesrozumitelně implementované, souvisí i s nižší úrovní dokumentace,
   - Střední - implementace je hůře pochopitelná, použití nástroje vyžaduje větší znalosti nástroje,
   - Dobrá - implementace je srozumitelná, veškerá funkcionalita nástroje je snadno dostupná.

Celkově nejlepšího hodnocení dosáhl nástroj `MNE`, a tudíž jej využiji pro implementaci načtení a předzpracování dat.

Pro implementaci lineárních klasifikátorů bude použita knihovna `scikit-learn`, která je již s `MNE` integrována. Pro implementaci neuronových sítí byly analyzovány knihovny `PyTorch` a `Keras`, které jsou si svými parametry z hlediska výkonnosti, rozšířenosti a množství poskytovaných funkcí podobné. Nástroj `MNE` a knihovna `PyTorch` jsou již zintegrovány pomocí nástroje `Braindecode`, jehož nevýhodou ale je, že obsahuje implementaci pouze několika modelů konvolučních neuronových sítí a přímo nepodporuje implementaci vlastní neuronové sítě. Po vlastní zkušenosti mi přijde práce s knihovnou `Keras` uživatelsky přívětivější, jednodušší a srozumitelnější.

	Brainstorm	**FieldTrip**	**EEGLab**	**Neo**	**Pandas**	**MNE**
Licence	Ne (pro ML metody Ano)	Ano	Ano	Ne	Ne	Ne
Úroveň dokumentace	Dobrá	Dobrá	Dobrá	Střední	Dobrá	Dobrá
Využití ML metod	Malé	Střední	Střední	Malé	Dobré	Dobré
Možnosti analýzy	Dobré	Dobré	Dobré	Malé	Malé	Dobré
Integrace s okolím	Dobrá	Dobrá	Střední	Dobrá	Dobrá	Dobrá
Velikost komunity	Malá	Střední	Malá	Malá	Velká	Stredni
Uživatelská přívětivost	Střední	Střední	Střední	Malá	Dobrá	Dobrá

Tabulka 4.2: Tabulka s hodnocením nástrojů dle definovaných kritérií

5 Návrh řešení

Jedním z cílů této práce je ověřit proveditelnost integrace vybraných klasifikačních metod strojového učení do zvoleného nástroje pro analýzu EEG dat. Funkčnost bude demonstrována replikací experimentu provedeného v [44]. Výzkumník z neuroinformatické laboratoře na FAV ZČU v něm na reálně naměřených datech testoval úspěšnost klasifikace pomocí konvoluční neuronové sítě a porovnával ji s klasickými klasifikátory LDA a SVM. Pro načtení, extrakci epoch a filtrování ovšem použil nástroje v jazyce MATLAB a pro klasifikaci Python knihovny `scikit-learn` a `Keras`.

Cílem následující části práce je implementovat tento experiment kompletně v jazyce Python, a tím prokázat, že lze tento standardní postup vyhodnocování EEG dat absolvovat pouze s využitím prostředků v tomto jazyce a dosáhnout podobných výsledků. V programu budou data ve formátu `BrainVision` načtena a předzpracována stejnými metodami jako v [44] pomocí nástroje `MNE`, jenž byl zvolen podle výsledků analýzy v předcházející kapitole. Ta budou dále klasifikována metodami strojového učení, jež se využívají v neuroinformatické laboratoři na FAV ZČU. Z lineárních to budou LDA a SVM z knihovny `scikit-learn` a z metod hlubokého učení konvoluční a rekurentní neuronová síť, pro jejichž implementaci bude použita knihovna `Keras`. Dosažené výsledky budou porovnány s výsledky získanými v [44].

5.1 Použitá data

V experimentu se bude vycházet z reálných dat získaných od 250 dětí základních a středních škol v plzeňském regionu od podzimu 2014 do jara 2015. Pro získání EEG záznamů jednotlivých subjektů byl použit experiment pojmenovaný Guess the number s následujícím postupem. Měřený subjekt byl před počátkem měření vyzván k libovolnému výběru čísla od 1 do 9 a k soustředění se na něj. Poté začalo zaznamenávání EEG dat a subjektu byly promítány vizuální stimuly s náhodnými čísly mezi 1 a 9 s intervalem změny čísla 1.5 sekundy.

Součástí experimentu bylo to, že výzkumníci v jeho průběhu sledovali průměrované evokované potenciály jednotlivých stimulů v reálném čase a podle nich se snažili uhodnout, na jaké číslo testovaný subjekt myslí. Experiment byl přerušen v okamžiku, kdy se výzkumníci rozhodli uhodnout subjektem myšlené číslo, anebo kdy usoudili, že nemají šanci myšlené číslo

ze signálu uhodnout. V případě, že po ověření subjektem bylo myšlené číslo uhodnuto na první pokus, byl experiment přerušen, a v případě špatného tipu experiment většinou pokračoval, a byl proveden druhý, případně třetí pokus o uhodnutí.

V průběhu experimentu byla měřena EEG data ze tří elektrod (Fz, Cz, Pz), a také byly zaznamenávány časové značky stimulů. U každého testovaného subjektu se uložila jeho základní experimentální metadata. Pro záznam a ukládání naměřených surových EEG dat byl použit BrainVision Recorder.

Naměřená data z experimentu ve formátu `BrainVision` jsou volně dostupná ke stažení a jsou strukturovaná po jednotlivých subjektech, kde záznam každého z nich je reprezentován složkou s následující strukturou:

- Protokol o experimentu je uložen ve složce jménem `Scenario`.
- Naměřená data a metadata uložena v `BrainVision` formátu (*.eeg*, *.vhdr* a *.vmrk* soubory) a metadata (*.txt* soubor) se nacházejí ve složce `Data`, která má tuto strukturu:
  - `P3Numbers_yyyymmdd_pohlaví_věk_id.eeg` je binární soubor obsahující surová EEG data,
  - `P3Numbers_yyyymmdd_pohlaví_věk_id.vhdr` je textový soubor obsahující metadata popisující surová data uložená v odpovídajícím *.eeg* souboru,
  - `P3Numbers_yyyymmdd_pohlaví_věk_id.vmrk` je textový soubor obsahující druhy stimulů používaných v experimentu,
  - `P3Numbers_yyyymmdd_pohlaví_věk_id.txt` je textový soubor obsahující základní experimentální metadata o subjektu: jeho pohlaví, věk, dominantní ruka, myšlené číslo a všechny až případné tři tipy myšleného čísla výzkumníky.
- Soubor `metadata.xml` obsahující experimentální metadata popisující použitý hardware a software a další technické informace o experimentu.

Kompletní popis experimentu a naměřených dat se nachází ve [28].

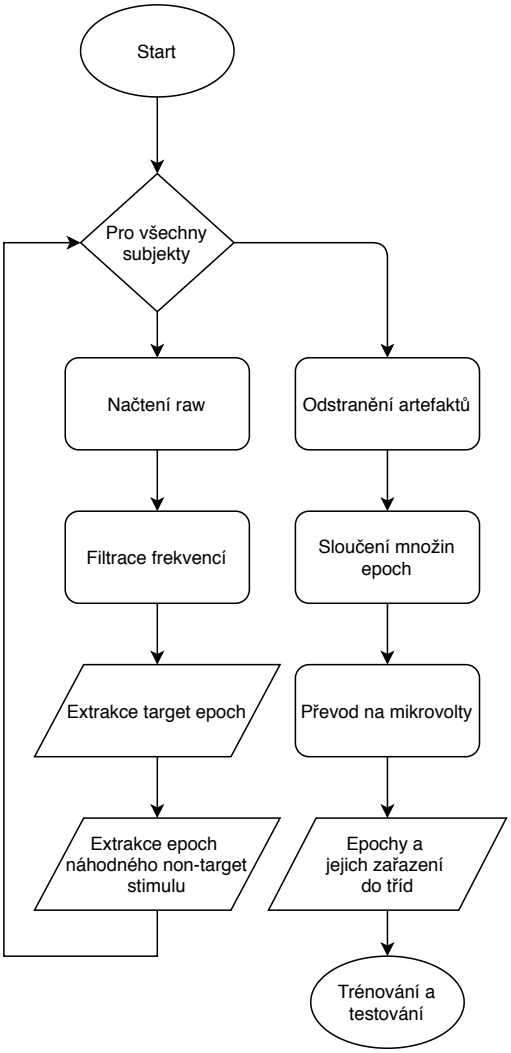
5.2 Načtení a předzpracování dat

V experimentu jsou pro každý testovaný subjekt načtena jeho naměřená EEG data a filtrována pomocí dolní propusti pro odstranění frekvencí vyšší než 30 Hz a pomocí horní propusti pro odstranění frekvencí nižší než 0,1

Hz. U přibližně poloviny testovaných subjektů byl zaznamenáván i kanál EOG. Vyskytuje-li se tento kanál v datech, je následně odstraněn, protože nebude dále analyzován. Z těchto dat jsou poté extrahovány dvě množiny epoch. První skupina epoch, která je označena jako target, je extrahována okolo stimulu reprezentujícího myšlené číslo subjektem. Ze zbylých čísel je následně jedno náhodně vybráno a epochy s tímto stimulem jsou extrahovány do druhé skupiny označené non-target. Rozložení zobrazování čísel měřenému subjektu je rovnoměrné, tudíž by velikosti obou množin epoch měly být podobné. Všechny epochy budou extrahovány v intervalu 200 ms před a 1000 ms po stimulu. Epochy, které mají peak-to-peak amplitudu větší než 150 μV, jsou považované za zašuměné artefakty a z množin odstraněny. Peak-to-peak amplituda je rozdíl mezi nejvyšší a nejnižší hodnotou v daném úseku signálu.

Data jsou poté převedena z voltů na mikrovolty. Pro účely klasifikace je nutné k extrahovaným epochám také vygenerovat jejich přiřazení do klasifikačních tříd v takovém tvaru, v jakém je akceptují metody pro strojové učení v jednotlivých knihovnách. Jelikož se tento tvar v `scikit-learn` a `Keras` liší, je množina přiřazení do tříd vygenerována ve tvaru, jenž využívá knihovna `Keras` a do formátu, který využívá knihovna `scikit-learn`, je poté transformována. Výsledkem kroku načítání a předzpracování dat je množina epoch vzniklá spojením sad target a non-target epoch a množina jim odpovídajících zařazení do tříd.

Vývojový diagram reprezentující postup načtení a předzpracování dat se nachází na obrázku č. 5.1.



Obrázek 5.1: Vývojový diagram pro načtení a předzpracování dat

5.3 Klasifikace

Cílem experimentu je binární klasifikace epoch do tříd target (myšlené číslo) a non-target (nemyšlené číslo). Počet epoch obou tříd je přibližně stejný a jejich rozdělení do trénovací, validační a testovací množiny je náhodné. Jedná se o učení s učitelem, kdy pro všechny epochy známe jejich správné zařazení do tříd.

Před samotným spuštěním klasifikace je nutné data převést do odpovídajícího vstupního formátu dané klasifikační metodě a rozdělit je na potřebné množiny.

5.3.1 Lineární klasifikátory

V řešení je použita Lineární diskriminační analýza a Metoda podpůrných vektorů. Obě metody jsou implementovány pomocí knihovny `scikit-learn`.

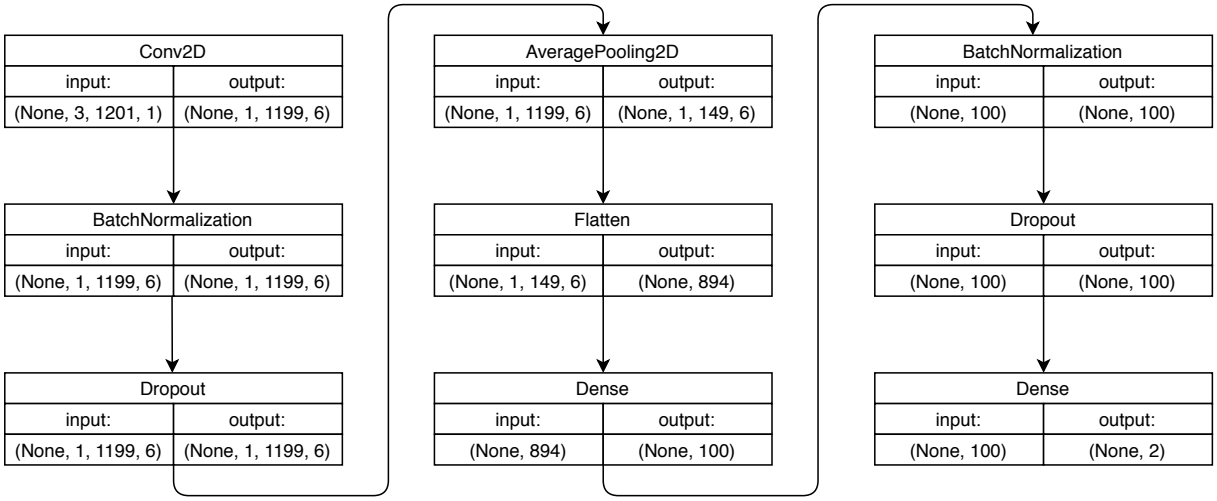
U lineární diskriminační analýzy je zvolena dekompozice pomocí vlastních vektorů a automatická shrinkage s použitím Ledoit-Wolf lemma. Pro metodu podpůrných vektorů je jako jádrová funkce použita radiální bázová funkce, regularizační parametr má hodnotu 1.0 a velikost jádrové cache je nastavena na 500 MB. Ostatní nastavitelné parametry obou klasifikátorů ve `scikit-learn` jsou nastaveny na svoji výchozí hodnotu.

Lineární klasifikátory v knihovně `scikit-learn` akceptují vstupní data ve formátu (počet_epoch x počet_příznaků). EEG data zpracovávaná v `MNE` se uchovávají ve tvaru (počet_epoch x počet_kanálů x počet_hodnot), tudíž je nutné před klasifikací data nějakým způsobem transformovat. Jednou z možností je snížení dimenzionality dat o jednotlivé kanály tudíž z dat ve tvaru (počet_epoch x 3_kanály x 1201_hodnot) vygenerovat formát (počet_epoch x 3603_hodnot). Podle praxe používání lineárních klasifikačních metod pro EEG data ale dosahují lineární klasifikátory lepších výsledků, když je jejich dimenzionalita snížena pomocí nějaké metody extrakce příznaků na (počet_vzorků x počet_příznaků).

V tomto řešení je použita metoda průměrování časových oken vycházející z [44]. Ta je založena na průměrování hodnot v daném intervalu vybraného časového okna v epoše. Časové okno, ve kterém se budou průměrovat hodnoty, se nachází v intervalu 300 ms a 1000 ms po stimulu. Toto okno je následně rozděleno do 20 stejně velkých časových intervalů, v nichž se počítá průměr hodnot. Tento postup je proveden pro všechny tři EEG kanály, tudíž je počet příznaků každé epochy 60. Všechny tyto vektory příznaků jsou poté škálovány na nulovou střední hodnotu a jednotkový rozptyl.

5.3.2 Konvoluční neuronová síť

V řešení je demonstrováno použití konvoluční neuronové sítě z [44]. Architektura použité konvoluční neuronové sítě je zobrazena na obrázku č. 5.2. Vstupní konvoluční vrstva obsahuje 6 filtrů ve tvaru 3 x 3. Dropout je v obou vrstvách nastaven na 0.5. Výstup konvoluční vrstvy byl převzorkován pomocí sdružování dle průměru (average pooling) s faktorem 8. Pro konvoluční vrstvu a vrstvu typu Dense byla použita ELU aktivační funkce. Výstupní vrstva je typu Dense s aktivační funkcí softmax.

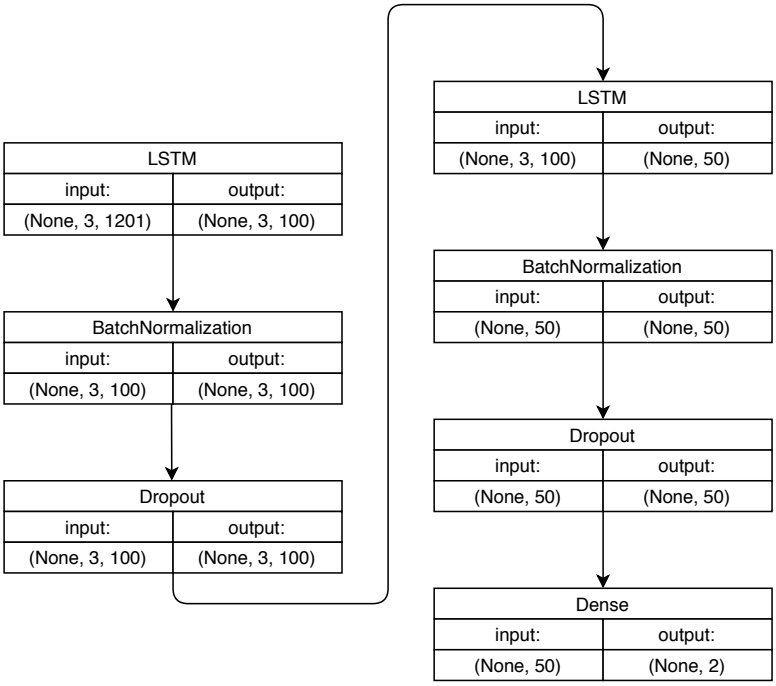


Obrázek 5.2: Architektura použité konvoluční neuronové sítě

Jednou z motivací použití neuronových sítí pro klasifikaci EEG signálu bylo vynechání často výpočetně náročného kroku extrakce příznaků. Tudíž jako vstup pro tuto konvoluční síť jsou použity přímo hodnoty signálu.

5.3.3 Rekurentní neuronová síť

Dále je v řešení použita rekurentní neuronová LSTM síť. Architektura této neuronové sítě je zobrazena na obrázku č. 5.3. Tato síť obsahuje dvě LSTM vrstvy s výstupní dimenzionalitou 100 a 50 s RELu aktivační funkcí. Po každé LSTM vrstvě následuje sekvence Normalizační a Dropout vrstvy s hodnotou 0,25. Výstupní vrstva je typu Dense s aktivační funkcí typu softmax. Tato neuronová síť byla převzata z [27] a její parametry byly experimentálně modifikovány za účelem dosažení lepších klasifikačních výsledků.



Obrázek 5.3: Architektura použité rekurentní neuronové sítě

Jelikož rekurentní neuronové sítě jsou stavěné pro klasifikaci časových řad, budou také do nich vstupovat hodnoty signálu. Pro obě implementované neuronové sítě je použit Adamův optimalizér a jako ztrátová funkce binární křížová entropie.

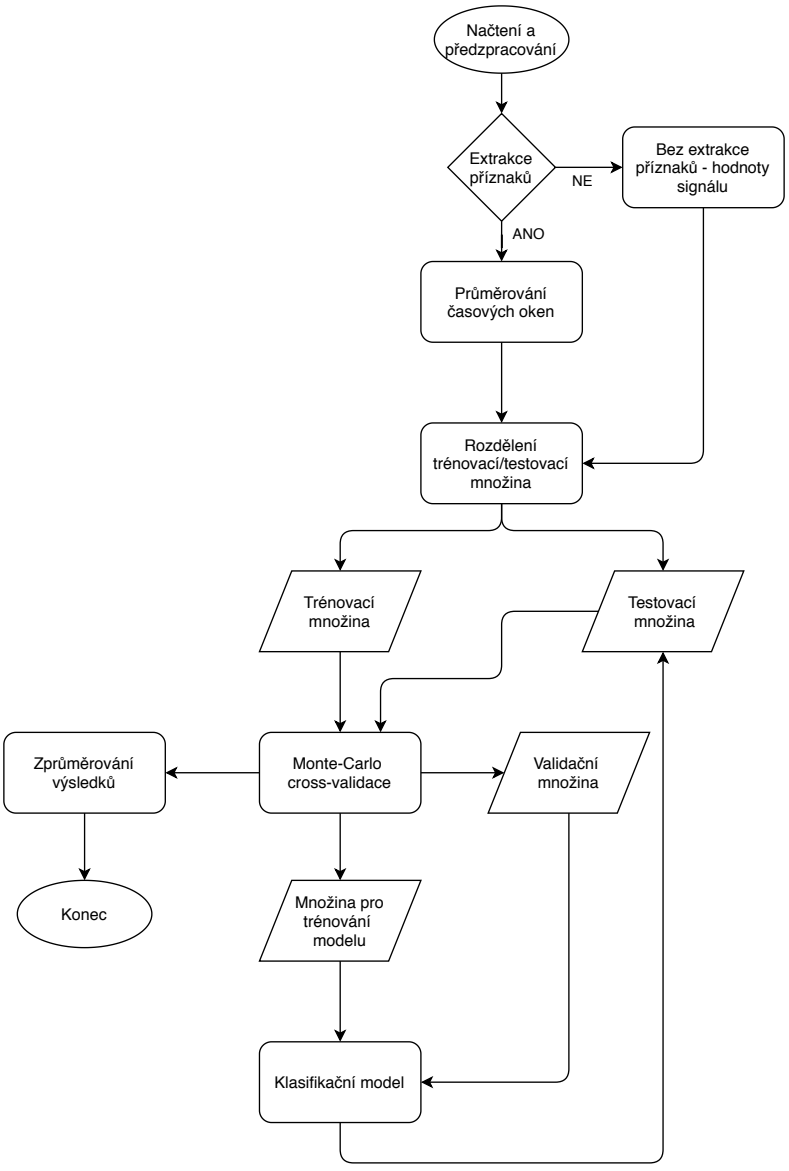
5.3.4 Vyhodnocení

Úspěšnost klasifikátorů je vyhodnocena pomocí Monte-Carlo cross-validace, která oproti klasické metodě cross-validace nerozděluje trénovací a testovací množinu pravidelně překrýváním, ale obě množiny generuje náhodně. Trénovací a testovací množiny budou rozděleny v poměru 75%:25% a z trénovací množiny bude ještě právě pomocí metody Monte-Carlo odděleno 25% dat pro validační množinu. Po každé iteraci jsou zjištěny výsledky klasifikace na testovací množině nazývané holdout, která nebyla nikdy použita pro trénování. Monte-Carlo cross validace je provedena ve 30 iteracích. Získané hodnoty validační a testovací úspěšnosti jsou zaznamenávány a na konci zprůměrovány.

Pro ohodnocení úspěšnosti jsou použity metriky přesnost (*accuracy*), preciznost (*precision*), úplnost (*recall*) a plocha pod křivkou (*Area under the ROC Curve, AUC*). Preciznost určuje poměr správně určených prvků ve všech klasifikovaných případech. Úplnost definuje, jaké procento z prvků příslušných

například ke třídě target klasifikátor správně přiřadil, a přesnost určuje poměr všech správně klasifikovaných vzorků ku všem získaným výsledkům. Metrika AUC je definována v [22].

Vývojový diagram popisující postup trénování a testování modelů se nachází na obrázku č. 5.4.



Obrázek 5.4: Vývojový diagram pro trénování a testování modelů

6 Implementace

Navržený postup experimentu je implementovaný v jazyce Python. Pro načtení a předzpracování dat je použita knihovna `MNE`, pro lineární klasifikátory knihovna `scikit-learn` a `Keras` pro neuronové sítě.

6.1 Načtení a předzpracování dat

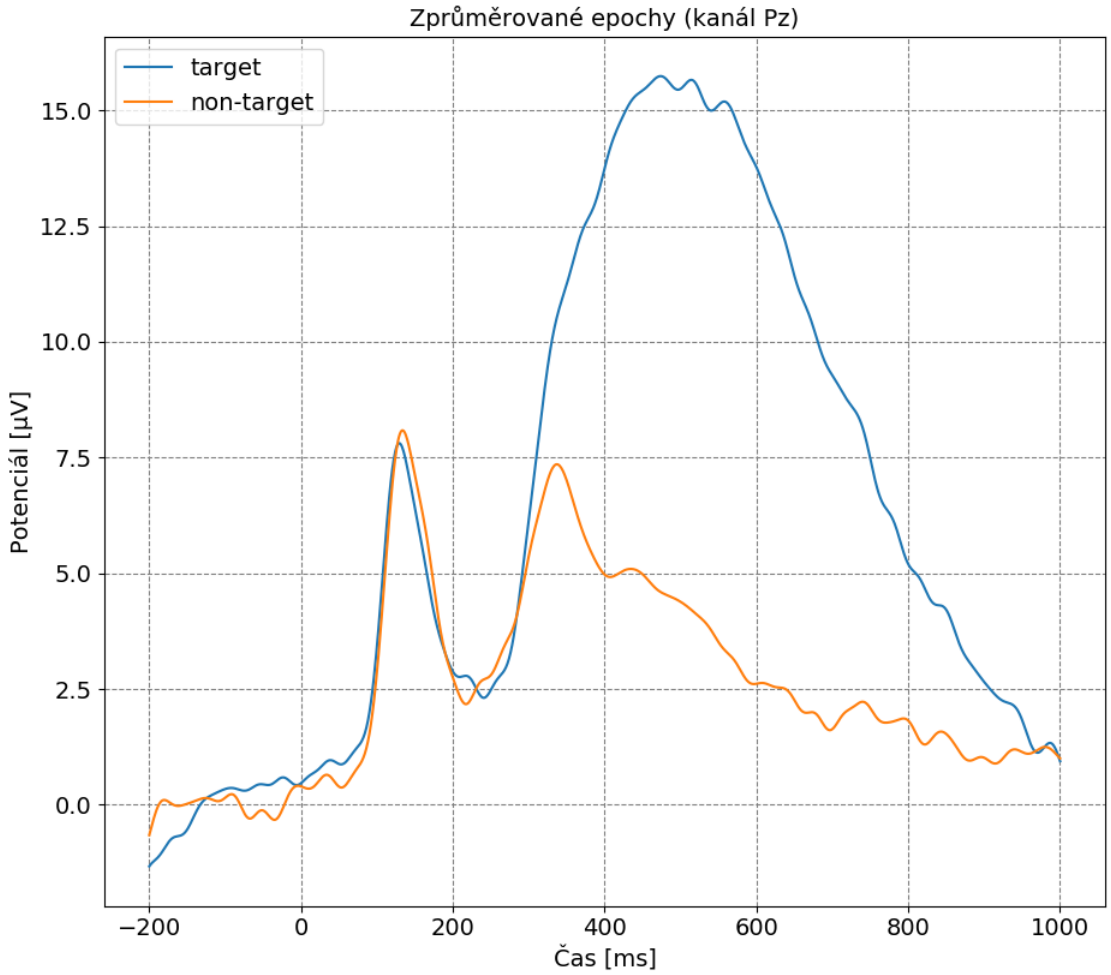
Prvním krokem implementace je zajištění načítání dat, jejichž struktura uložení je popsána v návrhu. Data jednotlivých subjektů jsou uložena ve složkách, jejichž jména lze získat pomocí Python funkce *os.walk()* vracející jména složek a souborů v daném adresáři. Data každého subjektu se nacházejí ve složce `Data` a konkrétně pro načtení dat ve formátu `BrainVision` pomocí `MNE` je potřebný soubor *.vhdr*. Poté je nalezeno kompletní jméno tohoto souboru a pomocí `MNE` funkce *mne.io.read_raw_brainvision()* jsou surová data načtena a uložena v objektu třídy *BrainVisionRaw*, který je podtřídou `MNE` třídy *Raw*. Lze tedy na načtená data využít metody třídy *Raw* umožňující například filtraci pomocí horní a dolní propusti.

V případě, že data obsahují čtyři kanály, je z nich pomocí metody *drop_channels()* třídy *Raw* odstraněn EOG kanál. Dalším krokem je zjištění myšleného čísla subjektem. Tato informace je uložena v metadatovém *.txt* souboru ve stejné složce. Poté můžeme ze surového signálu extrahovat epochy související s tímto stimulem, a to vytvořením objektu třídy *Epochs*. Předáním příslušných parametrů při inicializaci této třídy získáme objekt zapouzdřující množinu epoch okolo daného stimulu. Epochy jsou extrahované v intervalu od 200 ms před stimulem a do 1000 ms po stimulu s korekcí baseline v intervalu 200 ms až 0 ms před stimulem. Poté je vygenerováno náhodné číslo různé od myšleného, které bude použito pro extrakci non-target epoch. Ty se extrahují analogickým způsobem jako target epochy. Obě tyto množiny se připojují do globálních polí reprezentujících target a non-target epochy všech subjektů.

Po dokončení načítání dat všech subjektů jsou výsledné sady target a non-target epoch ve formě pole instancí třídy *Epochs*. Pole těchto instancí lze spojit do jedné instance třídy *BaseEpochs* (abstraktní nadtřída třídy *Epochs*) využitím metody *mne.concatenate_epochs()*. Z obou množin jsou poté metodou *drop_bad()* odstraněny epochy s peak-to-peak amplitudou větší než 150 μV. Následně jsou oba objekty třídy *BaseEpochs* spojeny do jednoho objektu opět metodou *mne.concatenate_epochs()* a pomocí třídy *Scaler* převedeny z μV na V. Využitím metody *get_data()* třídy *BaseEpochs* získáme

výslednou množinu předzpracovaných target a non-target označovanou jako X ve tvaru (počet_epoch x počet_kanálů x počet_hodnot). Tato množina se nyní nachází ve formátu `NumPy` a v takovémto tvaru je použita jako vstup do neuronových sítí z knihovny `Keras`. Pro lineární klasifikátory z knihovny `scikit-learn` projde tato množina ještě procesem extrakce příznaků.

Na obrázku č. 6.1 jsou zobrazené zprůměrované target a non-target epochy, které prošly sítem předzpracování. Počet těchto target epoch je 4010 a počet non-target epoch se kvůli náhodnému výběru pohybuje přibližně mezi 3800 - 4100. V grafu je dobře patrná vlna P300 v target epochách, která představuje reakci subjektu na promítnutí myšleného číslo.



Obrázek 6.1: Zprůměrované target a non-target epochy s viditelnou P300 vlnou

Pro účely klasifikace je nutné k extrahovaným epochám také vygenerovat

jejich odpovídající přiřazení do klasifikačních tříd tak, aby jejich formát odpovídal použitým machine-learningovým knihovnám `scikit-learn` a `Keras`. `Keras` pro označení zařazení do třídy používá binární matici ve tvaru (počet_tříd x počet_příkladů). Epocha patřící třídě target tedy bude mít odpovídající označení ve tvaru [1, 0], jenž reprezentuje zařazení do třídy 0, a epocha patřící do třídy non-target bude mít označení [0, 1]. Je tedy nutné vygenerovat odpovídající množství těchto dvojic pro obě sady epoch. Sadu označení lze vygenerovat pomocí metody *utils.to_categorical()* knihovny `Keras`, která pro dané označení tříd vygeneruje odpovídající binární matice. Tyto matice pro target a non-target jsou poté spojeny a výsledkem je množina přiřazení do tříd označovaná jako Y nebo labels.

6.2 Klasifikace

6.2.1 Lineární klasifikátory

Pro implementaci klasifikátorů SVM a LDA je použita knihovna `scikit-learn`. Podle návrhu je před spuštěním trénování klasifikátorů použita metoda průměrování časových oken pro extrakci příznaků převzatá z [44].

Třída reprezentující klasifikační model LDA se nachází v balíku `discriminant_analysis`. Dle návrhu je její inicializace provedena s parametrem *solver="eigen"* pro použití dekompozice pomocí vlastních vektorů a *shrinkage="auto"* pro použití Ledoit-Wolf lemma. Třída reprezentující klasifikační model SVM se nachází v balíku `svm` pod názvem *SVC* představující metodu SVM s možností volby regularizačního parametru. Při inicializaci se používají všechny hodnoty výchozí kromě nastavení parametru *cache_size=500*.

`Scikit-learn` v trénovací metodě přijímá množinu zařazení do tříd ve formě indexu reprezentujícího příslušnost ke třídě, což znamená, že pro target epochu to bude 1 a pro non-target epochu 0. Tudíž z množiny zařazení, která byla vytvořena pro knihovnu `Keras`, bude vždy extrahován jen index na první pozici, jenž toto zařazení indikuje.

Trénování obou klasifikátorů je shodně zajištěno voláním metody *fit()* s trénovacími množinami a po každé iteraci Monte-carlo cross-validace je úspěšnost klasifikátorů vyhodnocena výpočtem požadovaných metrik, jejichž hodnoty jsou uchovávány a po doběhnutí cross-validace zprůměrovány.

6.2.2 Neuronové sítě

Oba druhy neuronových sítí jsou implementovány pomocí knihovny `Keras`. Pro oba druhy neuronových sítí je vytvořena rodičovská třída *NN*, která

obsahuje metody pro učení a evaluaci, jež jsou pro oba druhy sítí totožné. Ztrátová funkce, optimalizér a sledované metriky jsou u obou druhu sítí specifikované při jejich sestavení pomocí metody *compile()*. Trénování obou sítí je spuštěno pomocí volání metody *fit()* s trénovacími a validačními daty. Počet iterací trénování je nastavený na 30 a počet vzorků použitých pro aktualizaci gradientu na 16. Aby nedocházelo k přetrénování modelu, je trénovací cyklus zastaven v případě, že se v po sobě jdoucích 5 iteracích nepodařilo dosáhnout vylepšení hodnoty validační ztrátové funkce. Toto lze definovat v parametru *callbacks* funkce *fit()*, jemuž se předá instance třídy *EarlyStopping* z knihovny `Keras`, ve které lze specifikovat, po kolika iteracích ukončit trénování v případě nedosažení lepšího výsledku a jaká metrika bude sledována. Testování modelu je provedeno pomocí metody *evaluate()* s testovací sadou.

Konvoluční neuronová síť

Konvoluční vrstva v knihovně `Keras` přijímá jako vstup 4D tensor, ale naše data jsou pouze 3D ve formátu (počet_epoch x počet_kanálů x počet_hodnot). Je nutné je tedy rozšířit o jednotkovou dimenzi do tvaru (počet_epoch x počet_kanálů x počet_hodnot x 1), což lze provést voláním metody *expand_dims()* z knihovny `NumPy`. Třída *CNN* se nachází v souboru *cnn.py* a jejím předkem je třída *NN*. Při inicializaci této třídy se vytvoří instance sekvenčního modelu knihovny `Keras` a do tohoto modelu se postupně přidávají dané vrstvy dle návrhu v předcházející sekci.

LSTM síť

Vstupní LSTM vrstva sítě přijímá data ve tvaru (počet_epoch x počet_kanálů x počet_hodnot), tudíž formát vstupní datové sadu není nutné nijak upravovat. Třída reprezentující model LSTM sítě se nachází v souboru *rnn.py*, a také dědí třídu *NN*. Model neuronové sítě je reprezentován sekvenčním modelem knihovny `Keras` a do něj jsou postupně vkládány vrstvy dle architektury uvedené v návrhu.

6.3 Trénování a evaluace

Všechny klasifikátory jsou trénovány a testovány Monte-Carlo cross-validací se 30 iteracemi. Pro prvotní náhodné rozdělení dat do trénovací a testovací množiny v poměru 75:25 je použita metoda *train_test_split()* z knihovny `scikit-learn`. 25% z trénovací množiny je v každé iteraci Monte-carlo cross-

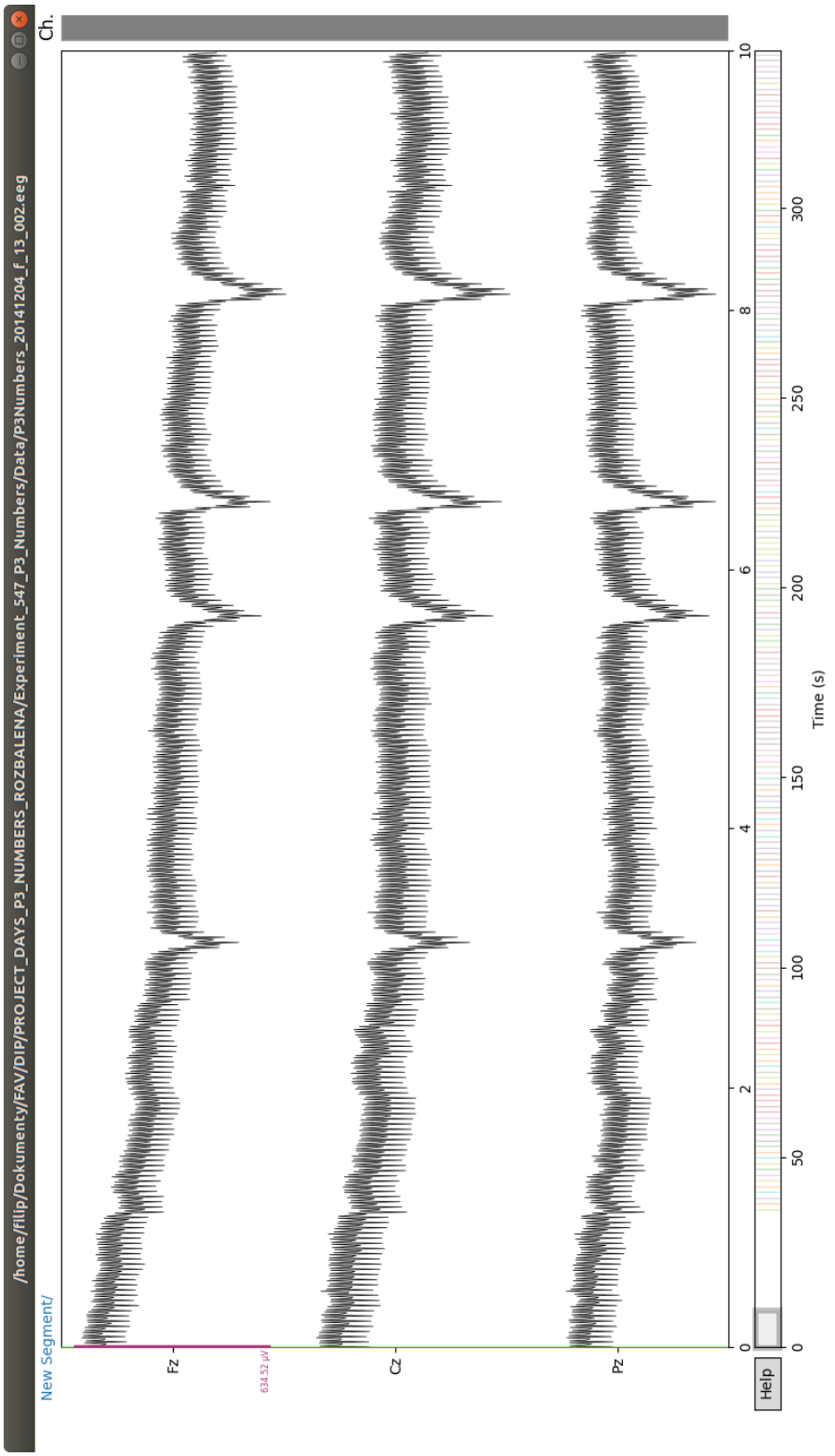
validace náhodně vybráno a tyto prvky jsou použity jako validační sada dat. Tento náhodný výběr je generován pomocí třídy *ShuffleSplit* z knihovny `scikit-learn`. Validační a testovací metriky ohodnocení jsou po každém kroku iterace ukládány a po doběhnutí cross-validace zprůměrovány.

7 Testování

Vzhledem k charakteru programu ve formě experimentu bude funkčnost jednotlivých metod, které nějakým způsobem transformovaly zpracovávaná data, ověřena pomocí vizuální kontroly vykreslením stavu dat před aplikací metody a po ní.

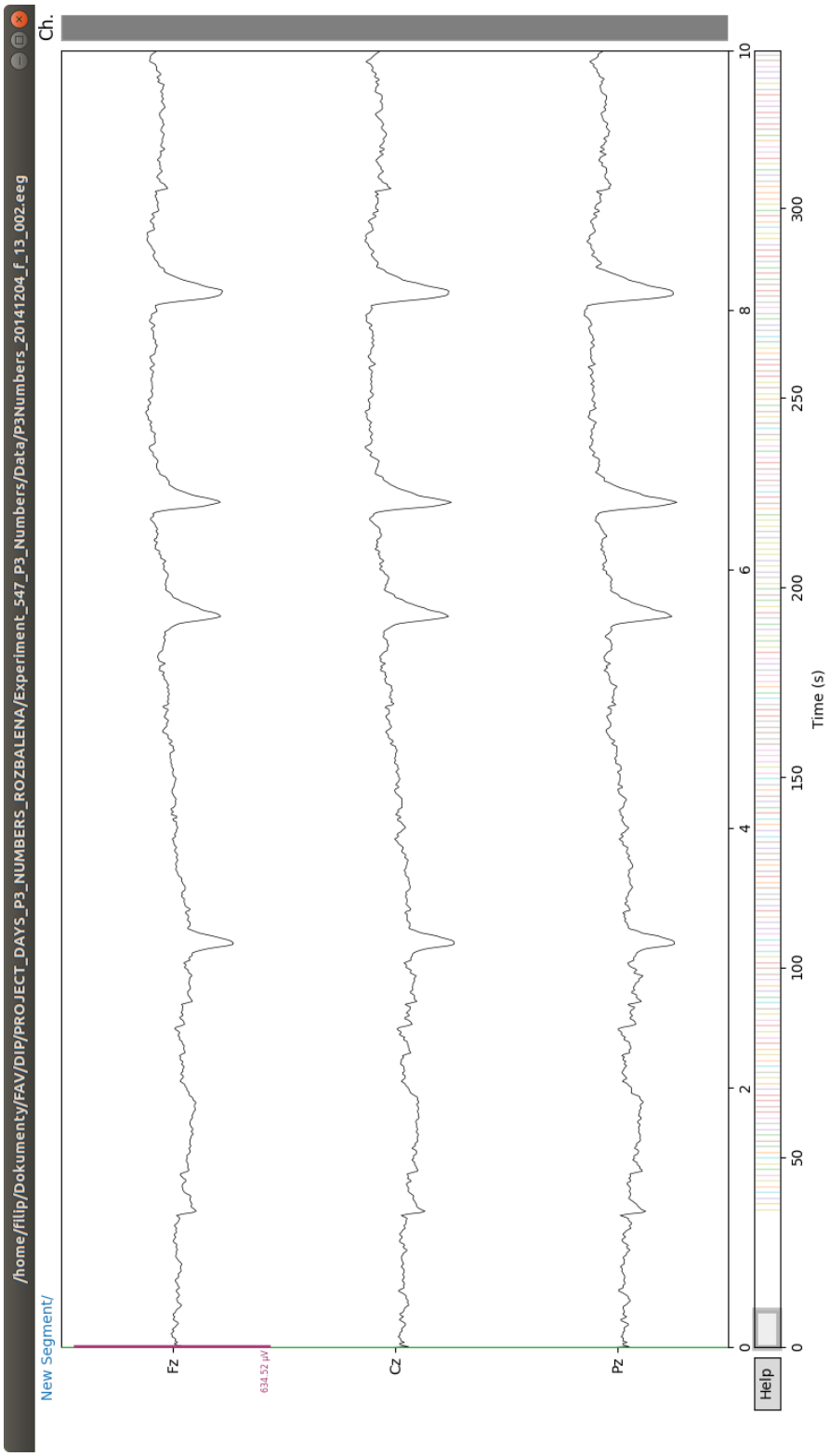
První provedenou transformací po načtení signálu daného subjektu je filtrování, čili odstranění frekvenčních pásem nižších než 0.1 Hz a vyšších než 30 Hz.

Na obrázku č. 7.1 je zobrazen surový naměřený signál ze 3 kanálů vybraného subjektu před aplikací filtrování, na kterém je patrný šum vytvářený signálem z elektrické sítě.



Obrázek 7.1: Naměřený signál vybraného subjektu před filtrováním

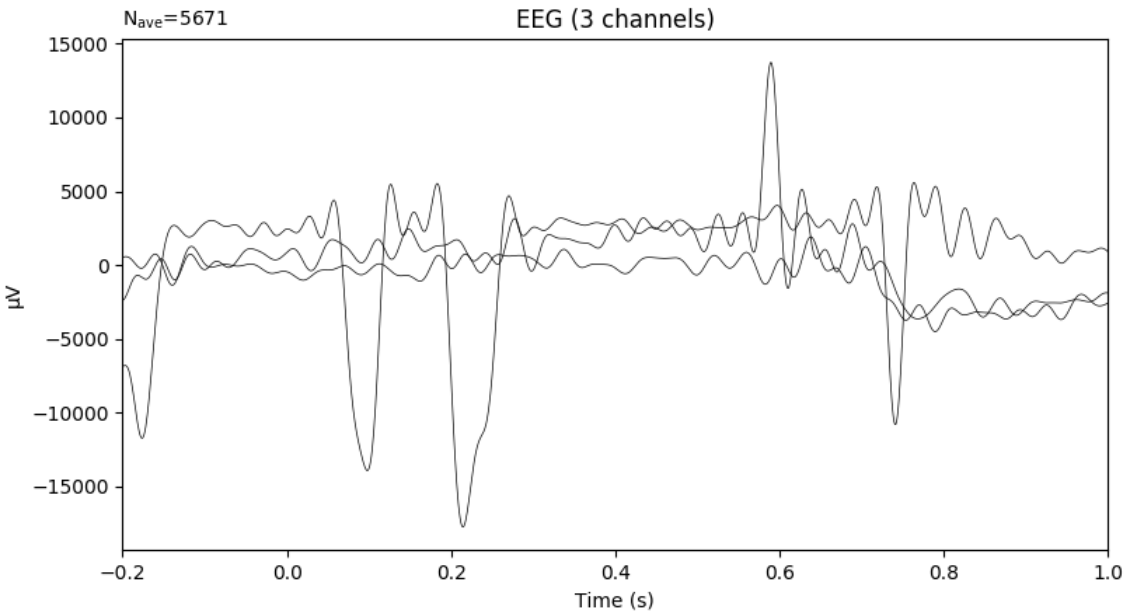
Na obrázku č. 7.2 se nachází stejný úsek signálu stejného subjektu po filtrování.



Obrázek 7.2: Naměřený signál vybraného subjektu po filtrování

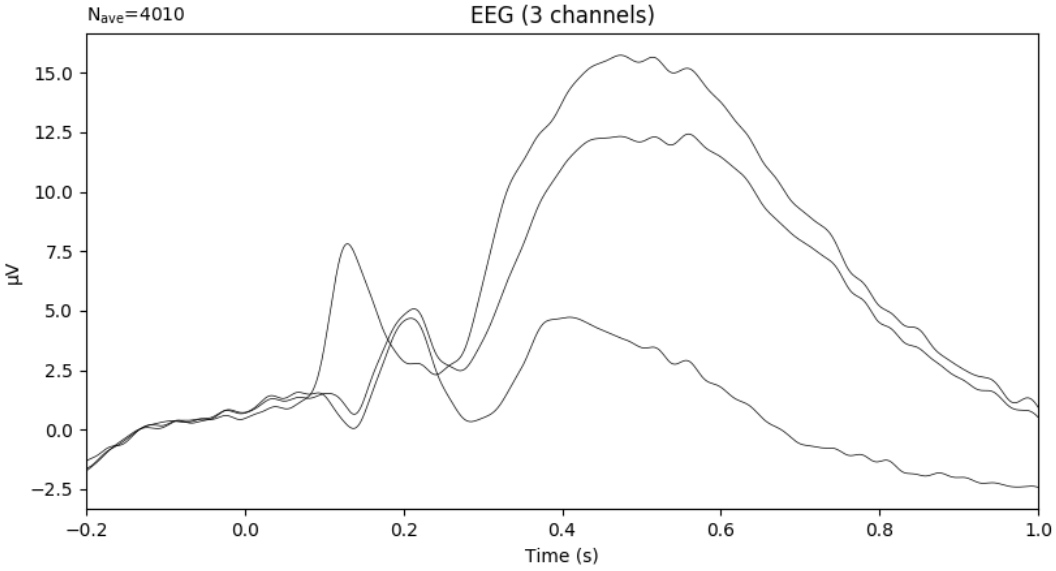
Pro vizualizaci signálu ve stavu před filtrací i po ní jsem použil funkci knihovny `MNE` pro vykreslení signálu třídy *Raw*.

Dalším krokem zpracování je odstranění epoch, které mají peak-to-peak amplitudu větší než 150 μV, a tudíž obsahují artefakty.



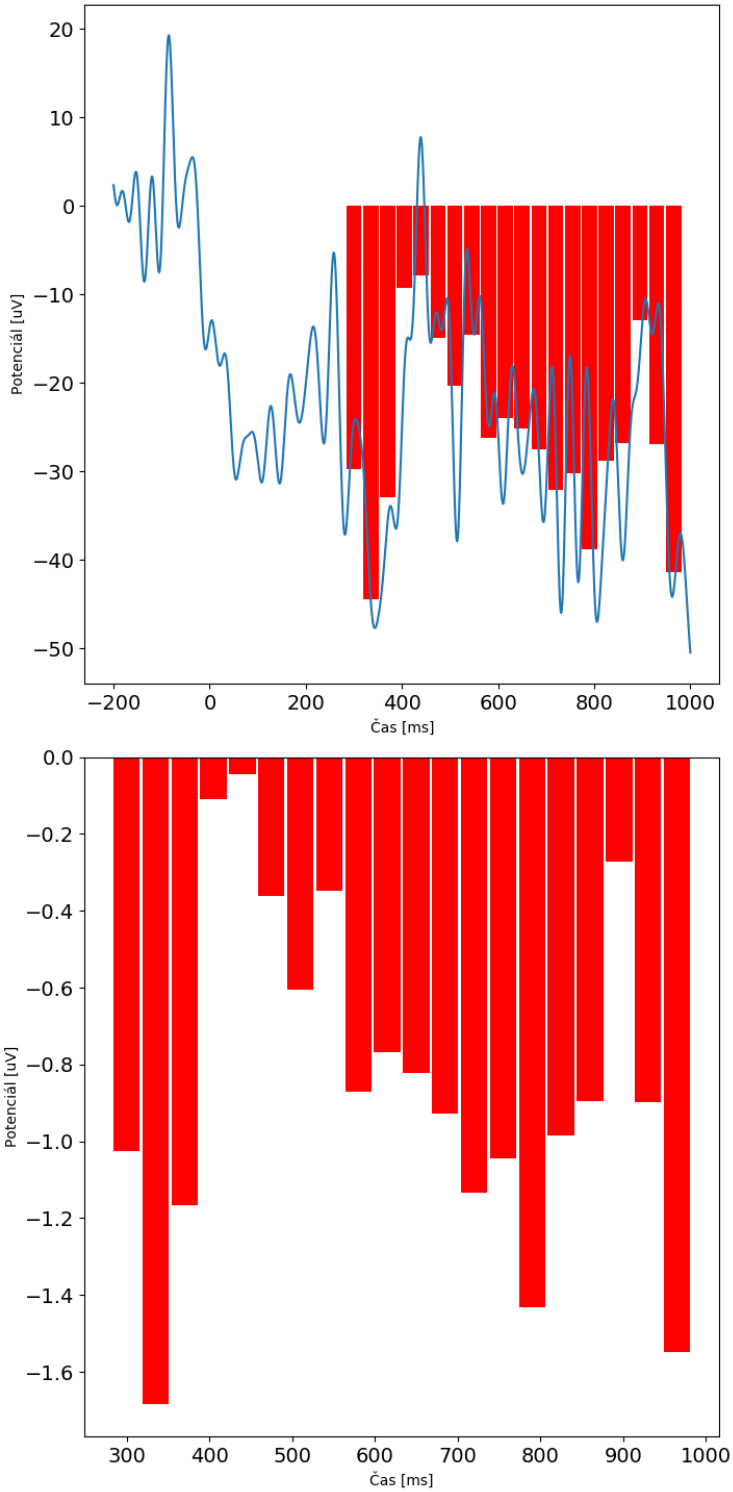
Obrázek 7.3: Zprůměrovaný signál target epoch před odstraněním epoch s artefakty

Na obrázku č. 7.3 je zobrazen zprůměrovaný signál epoch označených jako target před odstraněním epoch s artefakty a na obrázku č. 7.4 se nachází také zprůměrovaný signál epoch označených jako target, ovšem s odstraněnými epochami obsahujícími artefakty. Na dvou kanálech je znatelně vidět P300 vlna. Pro vizualizaci signálu jsem opět použil funkci knihovny `MNE` pro vykreslení signálu třídy *Evoked* reprezentující zprůměrovaný signál epoch.



Obrázek 7.4: Zprůměrovaný signál target epoch po odstranění epoch s artefakty

Pro klasifikaci pomocí LDA a SVM je použita metoda pro extrakci příznaků průměrování časových oken v intervalu od 300 ms do 1000 ms po stimulu. Její funkčnost je demonstrována na obrázku č. 7.5. V jeho horní části je zobrazen princip metody pro vybraný subjekt, kdy se ze signálu extrahuje 20 hodnot před aplikací škálování na nulovou střední hodnotu a jednotkový rozptyl. Výstup po aplikaci tohoto škálování je zobrazen na dolní části obrázku.



Obrázek 7.5: Metoda průměrování časových oken nahoře před škálováním na jednotkový rozptyl a nulovou střední hodnotu a dole po škálování

Celá zpracovávající pipeline složená z těchto jednotlivých kroků je funkční. Metoda filtrování je aplikována na surová data všech 250 subjektů. Metoda pro odstranění artefaktů z množin target i non-target metod odstraní přibližně 30% epoch. Konkrétně množina target metod před použitím této metody obsahovala 5671 epoch a po ní 4010. Pro non-target množinu nelze tyto počty přesně vyčíslit kvůli náhodnému výběru, ale poměr odstraněných epoch se také pohybuje okolo 30%.

8 Dosažené výsledky

V tabulce 8.1 jsou zobrazeny zprůměrované výsledky testování dosažené po provedení 10 opakování experimentu. V závorkách jsou uvedeny odpovídající standardní směrodatné odchylky. Nejlepšího výsledku v metrice AUC dosáhla suverénně CNN. Nejlepší přesnosti a preciznosti dosáhl klasifikátor SVM. Hodnoty přibližně odpovídají výsledkům dosaženým v [44]. Totožných výsledků nelze v tomto experimentu dosáhnout z důvodu nedeterminismu, který je způsobený náhodným výběrem non-target epoch. Rozdíly jsou v řádu jednotek procent.

Navíc byla implementována RNN, která dosáhla v porovnání s ostatními klasifikátory slušných výsledků v metrikách AUC a preciznost, ovšem v úplnosti dosáhla výrazně horších hodnot.

	AUC	**accuracy**	**precision**	**recall**
CNN	69.62% (0.79)	64.42% (0.74)	64.89% (1.17)	62.59% (2.52)
RNN	66.2% (0.87)	63.43% (0.83)	65.05% (1.77)	58.58% (4.51)
SVM	65.21% (0.44)	65.22% (0.45)	66.11% (0.64)	62.47% (1.04)
LDA	62.87% (0.38)	62.86% (0.38)	61.94% (0.43)	66.14% (0.8)

Tabulka 8.1: Dosažené výsledky testování

`MNE` je komplexní a dobře použitelný nástroj pro zpracování a analýzu EEG dat. Procedury pro načtení a předzpracování dat jsou v tomto nástroji velmi snadno dostupné, a tudíž je implementace standardní zpracovávající pipeline v `MNE` otázkou pouze několika řádek kódu. Struktura nástroje je snadno pochopitelná a jeho prostředky jsou dobře logicky a hierarchicky uspořádané. Úroveň dokumentace, poskytovaných návodů a komentovaných případů užití je velmi vysoká a veškerá uvedená podpora je neustále aktualizovaná, tudíž se na ni může uživatel plně spolehnout. Pro klasifikaci ERP je velkou výhodou nástroje možnost exportu analyzovaných epoch do formátu knihovny `NumPy`, jenž používají obě knihovny poskytující klasifikační metody strojového učení.

Kladně se dá hodnotit i zkušenost s open source Python knihovnami poskytujícími klasifikační metody strojového učení `scikit-learn` a `Keras`. `Scikit-learn` má velmi sofistikovaně navržené API, díky němuž je práce s ním variabilní a vysoce efektivní. Díky tomuto srozumitelnému a dobře popsanému API a tomu, že je postavený na knihovně `NumPy`, je dobře integrovatelný s okolím a umožňuje například vytvářet pipeline pro analýzu jakéhokoliv druhu dat skládáním metod z různých Python balíčků.

Prostředky knihovny `NumPy` využívá i knihovna `Keras`, a tudíž je integrace poskytovaných funkcí s ostatními Python knihovnami přímočará.

Kombinaci použití `MNE` pro načtení a předzpracování EEG dat v `BrainVision` formátu s klasifikátory poskytovanými knihovnami `scikit-learn` a `Keras` rozhodně mohu doporučit všem uživatelům, kteří hledají snadný způsob, jak klasifikovat EEG data.

Řešení je implementováno v Pythonu ve verzi 3.6.9 a verze jednotlivých využitých knihoven se nacházejí v souboru *requirements.txt*.

K implementaci byl vytvořen podrobný uživatelský manuál, jenž se nachází v příloze B.

9 Závěr

První část práce byla věnována úvodu do analýzy EEG signálu společně se základním názvoslovím a standardními formáty uložení. Poté byly představeny klasifikační metody strojového učení, které se používají v neuroinformatické laboratoři na FAV ZČU, a nejpopulárnější knihovny implementované v jazyce Python, jež tyto metody poskytují. Dále byly zanalyzovány současné nástroje pro zpracování EEG dat podle jejich možností využití metod strojového učení a dalších kritérií. Dle výsledků této analýzy byl pro zpracování EEG dat zvolen nástroj `MNE` a do něj byly integrovány vybrané klasifikační metody strojového učení z Python knihoven `scikit-learn` a `Keras`.

Proveditelnost řešení byla ověřena implementací experimentu s reálně naměřenými daty, který replikuje experiment provedený ve [44], jenž pro zpracování EEG dat používá nástroje v MATLABu a pro klasifikaci také Python knihovny `scikit-learn` a `Keras`. Kromě otestování funkčnosti integrace bylo dalším účelem implementace prokázat, že tento experiment lze provést se stejnými podmínkami kompletně pouze s využitím prostředků v jazyce Python a dosáhnout srovnatelných výsledků. Tento cíl se podařilo splnit. V rámci experimentu byly použity dvě standardní klasifikační metody strojového učení LDA a SVM a dvě metody hlubokého učení, a to konvoluční neuronová síť a rekurentní neuronová síť.

Analytická část a funkční, ověřené řešení může být nyní využíváno výzkumníky z neuroinformatické laboratoře na FAV ZČU pro další výzkum v oblasti analýzy EEG dat. Vytvořený zdrojový kód s podrobným uživatelským manuálem byly sdíleny do veřejně dostupného repositáře v [25], tudíž mohou být užitečné i globální komunitě.

Literatura

[1] *Elephant - Electrophysiology Analysis Toolkit* [online]. 2020. [cit. 2020/04/19]. Dostupné z: https://elephant.readthedocs.io/en/latest/index.html.

[2] *BIDS adopts BrainVision Core Data Format 1.0 as one of the recommended official EEG/iEEG data formats* [online]. Brain Products, 2019. [cit. 2020/07/18]. Dostupné z: https://pressrelease.brainproducts.com/bids/.

[3] *Description of the BrainVision Core Data Format 1.0* [online]. Brain Products, 2019. [cit. 2020/07/18]. Dostupné z: https://www.brainproducts.com/files/public/products/more/BrainVisionCoreDataFormat_1-0.pdf.

[4] *Keras layers API* [online]. 2020. [cit. 2020/05/20]. Dostupné z: https://keras.io/api/layers/.

[5] *MNE features - API Documentation* [online]. 2018. [cit. 2020/06/10]. Dostupné z: https://mne.tools/mne-features/api.html.

[6] *NIX data model* [online]. 2019. [cit. 2020/06/17]. Dostupné z: https://nixio.readthedocs.io/en/latest/data_model.html.

[7] *The NWB Software Ecosystem* [online]. Neurodata Without Borders. [cit. 2020/06/17]. Dostupné z: https://www.nwb.org/nwb-software/.

[8] *PyNN: documentation* [online]. 2020. [cit. 2020/04/19]. Dostupné z: http://neuralensemble.org/docs/PyNN/#.

[9] pyTorch, 2020. Dostupné z: https://github.com/pytorch/pytorch.

[10] Adrian, S. et al. File format and library for neuroscience data and metadata. *Frontiers in Neuroinformatics*. 01 2014, 8. doi: 10.3389/conf.fninf.2014.18.00027.

[11] Buitinck, L. et al. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, s. 108–122, 2013.

[12] Chollet, F. – others. Keras. https://keras.io, 2015.

[13] Chollet, F. *Deep learning v jazyku Python: knihovny Keras, Tensorflow*. Praha: Grada Publishing, 2019. ISBN 978-80-247-3100-1.

[14] Craik, A. – He, Y. – Contreras-Vidal, J. L. Deep learning for electroencephalogram (EEG) classification tasks: a review. *Journal of Neural Engineering*. apr 2019, 16, 3, s. 031001. doi: 10.1088/1741-2552/ab0ab5. Dostupné z: https://doi.org/10.1088%2F1741-2552%2Fab0ab5.

[15] Delorme, A. – Makeig, S. EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. *Journal of neuroscience methods*. March 2004, 134, 1, s. 9—21. ISSN 0165-0270. doi: 10.1016/j.jneumeth.2003.10.009. Dostupné z: https://doi.org/10.1016/j.jneumeth.2003.10.009.

[16] Ekštein, K. *Strojové učení - Umělé neuronové sítě* [online]. 2017. [cit. 2020/04/25]. Dostupné z: https://courseware.zcu.cz/portal/studium/courseware/kiv/su/prednasky.html.

[17] Garcia, S. – Gill, J. *ephyviewer 1.3.2.dev* [online]. 2019. [cit. 2020/04/19]. Dostupné z: https://ephyviewer.readthedocs.io/en/latest/index.html.

[18] Gramfort, A. et al. MEG and EEG data analysis with MNE-Python. *Frontiers in Neuroscience*. 2013, 7, s. 267. ISSN 1662-453X. doi: 10.3389/fnins.2013.00267. Dostupné z: https://www.frontiersin.org/article/10.3389/fnins.2013.00267.

[19] Grewe, J. – Wachtler, T. – Benda, J. A Bottom-up Approach to Data Annotation in Neurophysiology. *Frontiers in Neuroinformatics*. 2011, 5, s. 16. ISSN 1662-5196. doi: 10.3389/fninf.2011.00016. Dostupné z: https://www.frontiersin.org/article/10.3389/fninf.2011.00016.

[20] Herger, B. *keras-pandas* [online]. 2018. [cit. 2020/04/20]. Dostupné z: https://keras-pandas.readthedocs.io/en/latest/intro.html.

[21] Holčík, J. – Komenda, M. *Matematická biologie: e-learningová učebnice [online]*. Masarykova univerzita, 1. edition, 2015. Dostupné z: http://portal.matematickabiologie.cz/. ISBN 978-80-210-8095-9.

[22] Hossin, M. – M.N, S. A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining Knowledge Management Process*. 03 2015, 5, s. 01–11. doi: 10.5121/ijdkp.2015.5201.

[23] Kothe, C. – Makeig, S. BCILAB: A platform for Brain–Computer interface development. *Journal of neural engineering*. 08 2013, 10, s. 056014. doi: 10.1088/1741-2560/10/5/056014.

[24] Koutsou, A. *Data model mapping between Neo and NIX* [online]. 2018. [cit. 2020/04/19]. Dostupné z: https://github.com/G-Node/python-neo/wiki/Mapping.

[25] Kupilík, F. MNE_ML, 2020. Dostupné z: https://github.com/fkupilik/MNE_ML.

[26] Ledoit, O. – Wolf, M. Honey, I Shrunk the Sample Covariance Matrix. *The Journal of Portfolio Management*. 2004, 30, 4, s. 110–119. ISSN 0095-4918. doi: 10.3905/jpm.2004.110. Dostupné z: https://jpm.pm-research.com/content/30/4/110.

[27] Mathewson, K. E. – Mathewson, K. W. DeepEEG, 2019. Dostupné z: https://github.com/kylemath/DeepEEG.

[28] Mouček, R. et al. Event-related potential data from a guess the number brain-computer interface experiment on school children. *Scientific Data*. March 2017, 4, s. 1–11. ISSN 2052-4463. doi: https://doi.org/10.1038/sdata.2016.121.

[29] Nicholson, C. *A Beginner's Guide to LSTMs and Recurrent Neural Networks* [online]. 2019. [cit. 2020/04/27]. Dostupné z: https://pathmind.com/wiki/lstm.

[30] Oostenveld, R. et al. FieldTrip: Open Source Software for Advanced Analysis of MEG, EEG, and Invasive Electrophysiological Data. *Computational Intelligence and Neuroscience*. ISSN 1687-5265. doi: 10.1155/2011/156869. Dostupné z: https://doi.org/10.1155/2011/156869.

[31] Pedregosa, F. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, 12, s. 2825–2830.

[32] Pernet, C. R. et al. EEG-BIDS, an extension to the brain imaging data structure for electroencephalography. *Scientific Data*. 2019, 6, 103. doi: https://doi.org/10.1038/s41597-019-0104-8.

[33] Pröpper, R. – Obermayer, K. Spyke Viewer: a flexible and extensible platform for electrophysiological data analysis. *Frontiers in Neuroinformatics*. 2013, 7, s. 26. ISSN 1662-5196. doi: 10.3389/fninf.2013.00026. Dostupné z: https://www.frontiersin.org/article/10.3389/fninf.2013.00026.

[34] Raschka, S. *Linear Discriminant Analysis - Bit by Bit* [online]. 2014. [cit. 2020/04/23]. Dostupné z: https://sebastianraschka.com/Articles/2014_python_lda.html.

[35] Roy, Y. et al. Deep learning-based electroencephalography analysis: a systematic review. *Journal of Neural Engineering*. 05 2019, 16. doi: 10.1088/1741-2552/ab260c.

[36] Ruebel, O. et al. NWB:N 2.0: An Accessible Data Standard for Neurophysiology. 01 2019. doi: 10.1101/523035.

[37] S., G. et al. Neo: an object model for handling electrophysiology data in multiple formats. *Frontiers in Neuroinformatics*. February 2014, 8:10. doi: 10.3389/fninf.2014.00010.

[38] Schiratti, J.-B. et al. An ensemble learning approach to detect epileptic seizures from long intracranial EEG recordings. In *International Conference on Acoustics, Speech, and Signal Processing*, Calgary, Canada, April 2018. Dostupné z: https://hal.archives-ouvertes.fr/hal-01724272.

[39] Schirrmeister, R. et al. Deep learning with convolutional neural networks for EEG decoding and visualization: Convolutional Neural Networks in EEG Analysis. *Human Brain Mapping*. 08 2017, 38. doi: 10.1002/hbm.23730.

[40] Tadel, F. et al. Brainstorm: A User-Friendly Application for MEG/EEG Analysis. *Computational Intelligence and Neuroscience*. ISSN 1687-5265. doi: 10.1155/2011/879716. Dostupné z: https://doi.org/10.1155/2011/879716.

[41] team. *pandas documentation* [online]. 2020. [cit. 2020/04/20]. Dostupné z: https://pandas.pydata.org/docs/index.html#pandas-documentation.

[42] Tišnovský, P. *Framework Torch: využití konvolučních sítí pro rozpoznávání a klasifikaci obrázků* [online]. 2017. [cit. 2020/04/24]. Dostupné z: https://www.root.cz/clanky/framework-torch-vyuziti-konvolucnich-siti-pro-rozpoznavani-a-klasifikaci-obrazku/.

[43] Gerven, M. et al. Donders Machine Learning Toolbox. https://github.com/distrep/DMLT, 2011.

[44] Vařeka, L. Evaluation of convolutional neural networks using a large multi-subject P300 dataset. *Biomedical Signal Processing and Control*. 2020, 58, s. 101837. ISSN 1746-8094. doi: https://doi.org/10.1016/j.bspc.2019.101837. Dostupné z: http://www.sciencedirect.com/science/article/pii/S1746809419304185.

[45] Ye, J. – Janardan, R. – Li, Q. Two-Dimensional Linear Discriminant Analysis. In Saul, L. K. – Weiss, Y. – Bottou, L. (Ed.) *Advances in Neural Information Processing Systems 17*. MIT Press, 2005. s. 1569–1576. Dostupné z: http://papers.nips.cc/paper/2547-two-dimensional-linear-discriminant-analysis.pdf.

[46] Šedivá, B. *Mnoharozměrné statistické metody - Diskriminačnı analýza* [online]. 2020. [cit. 2020/04/23]. Dostupné z: https://courseware.zcu.cz/portal/studium/courseware/kma/msm/prednasky.html.

[47] Žižka, J. *Support vector machines (SVM), Algoritmy podpůrných vektorů* [online]. 2005. [cit. 2020/04/23]. Dostupné z: https://is.muni.cz/el/1433/podzim2005/PA034/09_SVM.pdf.

Seznam zkratek

μV mikrovolt

API Application programming interface - rozhraní

AUC Area under the ROC Curve - metrika pro ohodnocení úspěšnosti klasifikace

BIDS Brain Imaging Data Structure - formát ukládání neuroimaging dat

CNN Konvolučení neuronová síť

CPU Centrální procesorová jednotka

CSP Common spatial patterns - metoda extrakce příznaků

Cz Central zero elektroda - centrální elektroda umístěna na střední sagitální rovině lebky

DL Deep-learning - hluboké učení

DMLT Donders Machine Learning Toolbox - MATLAB toolbox poskytující metody strojového učení

ECG Elektrokardiografie - záznam elektrické srdeční aktivity

ECoG Elektrokortikografie - záznam elektrické aktivity pomocí elektrod umístěných na povrchu mozku

EEG-BIDS Rozšíření BIDS formátu pro ukládání EEG dat

EEG Elektroencefalografie - záznam elektrické aktivity mozku

EOG Elektrookulografie - záznam změn elektrického potenciálu, které jsou vyvolané pohyby oka

ERP Event-related potentials - změny elektrické aktivity mozku vyvolané vnějším podnětem

FAV Fakulta aplikovaných věd

fNIRS Funkční infračervená spektroskopie - technologie schopná vizualizovat změny koncentrace kyslíku

Fz Front zero elektroda - frontální elektroda umístěna na střední sagitální rovině lebky

GPU Grafická procesorová jednotka

HDF5 Hierarchical Data Format - formát navržený pro ukládání a organizaci velkého množství dat a metadat

HW Hardware

Hz Hertz - jednotka frekvence

I/O Input/Output - vstup/výstup

iEEG Intrakraniální elektroencefalografie - záznam elektrické aktivity pomocí elektrod umístěných na povrchu mozku

LDA Lineární diskriminační analýza

LFP Local field potentials - potenciály lokálního pole

LSTM Long short-term memory - implementace rekurentní neuronové sítě

MB MegaByte

MEG Magnetoencefalografie - záznam magnetických polí, která vznikají jako důsledek elektrické aktivity mozku

ms milisekunda

NEO Python nástroj pro práci s EEG daty

NIX Neuroscience information exchange format - formát ukládání EEG dat

NWB Neurodata Without Borders - formát ukládání EEG dat

odML open metadata Markup Language - formát pro ukládání metadat

P300 Pozitivní vlna signálu, jež bývá považována za specifickou reakci na nějaký podnět

Pz Parietal zero elektroda - parietální elektroda umístěna na střední sagitální rovině lebky

RNN Rekurentní neuronová síť

SEEG Stereoelektroencefalografie - záznam elektroencefalografických signálů pomocí hloubkových mozkových elektrod

SQL Structured Query Language - strukturovaný dotazovací jazyk používaný v relačních databázích

SVC Třída z knihovny `scikit-learn` reprezentující metodu SVM s možností volby regularizačního parametru

SVD Singulární rozklad

SVM Metoda podpůrných vektorů

ZČU Západočeská univerzita v Plzni

Seznam příloh

A Uživatelský manuál

Součástí zadání je publikovat vytvořený zdrojový kód do veřejného repositáře, aby mohl být užitečný EEG komunitě. Pro usnadnění používání programu je vytvořen uživatelský manuál, ve kterém jsou podrobně popsány jednotlivé kroky programu.

A.1 Načtení a předzpracování dat

V provedeném experimentu jsou analyzovaná data získána od 250 subjektů. Každý subjekt má svůj vlastní adresář, který obsahuje složky `Scenario` s protokolem o experimentu a `Data` s EEG soubory ve formátu `BrainVision`. Dataset, který se používá v tomto experimentu se nachází na přiloženém DVD ve složce `data`.

```
root, dirs, files = next(os.walk(param.path))

for folder_name in dirs:
    path_file_vhdr = param.path + folder_mame + '/Data/*.vhdr'
    file_name_vhdr = glob(path_file_vhdr)
    if len(file_name_vhdr) == 0:
        continue
    raw = mne.io.read_raw_brainvision(file_name_vhdr[0], preload
        =True)
    raw.filter(l_freq=param.l_freq, h_freq=param.h_freq)
    if (raw.info['nchan'] == 4):
        raw = raw.drop_channels(['EOG'])
```

Ukázka kódu A.1: Načtení surových dat všech subjektů filtrování a odstranění EOG kanálu

V ukázce kódu č. A.1 se v prvním příkazu získají názvy všech složek představujících získaná data jednotlivých subjektů, která se nacházejí v adresáři, jenž je reprezentován cestou specifikovanou v proměnné *param.path* v absolutním tvaru. Hodnota této proměnné se stejně jako ostatní parametry relevantní pro tento experiment nastavuje v konfigurační třídě *Param* v souboru *param.py*. Pole s názvy těchto složek je uloženo v proměnné *dirs*. Poté začíná cyklus, který zajistí načtení dat všech subjektů. Při načítání `BrainVision` dat pomocí `MNE` se používá soubor s příponou *.vhdr*. Jméno tohoto souboru je pro každý subjekt originální, ale ve složce `Data` se pro

každý subjekt nachází jen jeden soubor tohoto typu, tudíž se jeho název zjistí pomocí funkce *glob()*, která vrátí seznam cest odpovídající určitému vzoru. Název *.vhdr* souboru daného subjektu je nyní uložen v poli *file_name_vhdr* na pozici 0.

Pokud se *.vhdr* soubor ve složce nenachází, načítání se přesune k dalšímu subjektu.

Surová data jsou načtena do proměnné *raw* pomocí funkce *mne.io.read_raw_brainvision()* s parametry, kterými jsou název *.vhdr* souboru a *preload=True*, díky kterému se data načítají do paměti, což umožní jejich pozdější filtrování. Surová data jsou poté filtrována pomocí horní a dolní propusti. V `MNE` je nutné, aby všechny subjekty měly stejný počet kanálů, v našem případě tedy 3. EOG kanál, který byl zaznamenáván přibližně u poloviny subjektů, je ze surových dat odstraněn.

Informace o myšleném číslu subjektem je uložena také ve složce `Data`, konkrétně v metadatovém textovém souboru. Myšlené číslo je nutné zjistit z toho důvodu, abychom mohli extrahovat target epochy. Toto číslo se v souboru nachází vždy na třetí řádce a jeho získání se nachází v ukázce č. A.2.

```
path_file_txt = param.path + folder_name + '/Data/*.txt'
file_name_txt = glob(path_file_txt)
if len(file_name_txt) == 0:
    continue
loaded_txt = open(file_name_txt[0], "r")
text = loaded_txt.readlines()
line = text[2]
event_id_target = int(line.split(": ")[1])
```

Ukázka kódu A.2: Získání myšleného čísla z metadatového textového souboru

Jméno metadatového textového souboru se zjistí obdobným způsobem jako při hledání názvu *.vhdr* souboru. Soubor se poté otevře a ze třetí řádky se vyextrahuje myšlené číslo do proměnné *event_id_target*. Pokud tento soubor ve složce není nalezen, je načítání dat tohoto subjektu ukončeno.

Poté se ze surového signálu získají události (stimuly) a extrahují target epochy, čili ty kolem stimulu s číslem, na které subjekt myslel, což je zobrazeno v ukázce č. A.3.

```
events_loaded = mne.events_from_annotations(raw)
epochs_t_subject = mne.Epochs(raw, events=events_loaded[0],
    event_id=event_id_target, tmin=param.t_min, tmax=param.
    t_max, baseline=param.baseline)
```

```
epochs_t_subject = mne.epochs.combine_event_ids(
    epochs_t_subject, [str(event_id_target)], {'target': 0})
```

Ukázka kódu A.3: Zjistění stimulů a extrakce target epoch

V proměnné *events_loaded* se nacházejí všechny stimuly a časy jejich výskytu v signálu, a také jejich textový popis. Časový interval kolem stimulu, ve kterém se mají epochy extrahovat (-200 ms až 1000 ms), a interval použitý pro korekci baseline (-200 ms až 0 ms) jsou definované v konfigurační třídě *Param*. Samotné získání target epoch se v `MNE` provádí inicializováním třídy *mne.Epochs* s danými parametry a s *event_id=event_id_target*, který zajistí, že se extrahují jen target epochy. Tyto epochy jsou nyní označeny číslem stimulu, kolem kterého byly extrahovány, ale pro potřeby klasifikace je potřebujeme označit jako target, čili jako patřící do třídy 0, což zajišťuje poslední řádek ukázky A.3.

Po získání target epoch se u daného subjektu extrahují non-target epochy okolo náhodně vybraného stimulu, který reprezentuje číslo, jež si subjekt nemyslel. Tato část je zobrazena v ukázce č. A.4.

```
non_target_random = random.randint(1, 9)
while event_id_target == non_target_random:
    non_target_random = random.randint(1, 9)
epochs_n_subject = mne.Epochs(raw, events=events_loaded[0],
    event_id=non_target_random, tmin=param.t_min, tmax=param.
    t_max, baseline=param.baseline)
epochs_n_subject = mne.Epochs.combine_event_ids(
    epochs_n_subject, [str(non_target_random)], {'nonTarget':
     11})
epochs_target.append(epochs_t_subject)
epochs_non_target.append(epochs_n_subject)
loaded_txt.close()
```

Ukázka kódu A.4: Zjistění stimulů a extrakce non-target epoch

V první části ukázky se generuje náhodné číslo, které musí být odlišné od čísla, na něž subjekt myslel. Okolo tohoto stimulu se poté extrahují non-target epochy se stejnými parametry jako u target epoch. Poté se epochy označí jako non-target, tudíž patřící do třídy 1, a jelikož jsou čísla stimulů v rozsahu od 1 do 9, budou tyto epochy pro pozdější účely označeny jako 11. Extrahované target i non-target epochy jsou poté připojeny do globálních proměnných shromažďujících epochy všech subjektů. Otevřený metadatový textový soubor je uzavřen.

Celý tento blok programu se nachází v těle výjimky, která odchytí chybné zadání cesty k adresáři s datasetem, vypíše text chyby a program ukončí.

V tuto chvíli jsou target i non-target epochy ve formátu seznamu instancí třídy *mne.Epochs*. Zřetězení tohoto seznamu do jediné instance zajišťuje metoda *mne.concatenate_epochs()*. Nyní lze z množin odstranit epochy, které mají peak-to-peak amplitudu větší než 150 μV (definováno ve třídě *Param*) pomocí metody *drop_bad()* s daným prahem. Target i non-target epochy jsou poté spojeny do jedné proměnné *epochs_all*. Tento proces se nachází v ukázce č. A.5.

```
epochs_target = mne.concatenate_epochs(epochs_target)
reject = dict(eeg=param.amplitude)
epochs_target.drop_bad(reject=reject)
epochs_non_target = mne.concatenate_epochs(epochs_non_target)
epochs_non_target.drop_bad(reject=reject)
epochs_all = mne.concatenate_epochs([epochs_target,
    epochs_non_target])
```

Ukázka kódu A.5: Spojení epoch a odstranění amplitudy

V ukázce č. A.6 je v první části demonstrováno převedení jednotek signálu na μV pomocí třídy *mne.decoding.Scaler*. Této třídě se při inicializaci pomocí parametru *scalings* předá informace, jaký kanál a jakou konstantou má být škálovaný, tudíž v našem případě se jedná o EEG kanál a pro převedení na μV se škáluje hodnotou 1e6 (nastavené ve třídě *Param*). Poté se do proměnné X uloží všechny epochy, ale již ve formátu 3D tensoru knihovny `NumPy`, ve tvaru (počet_epoch x počet_kanálů x počet_hodnot). Na data v takovémto tvaru lze použít metodu *fit_transform()* třídy *Scaler*, která zajistí převedení na μV.

```
scalings = dict(eeg=param.scaling)
scaler = mne.decoding.Scaler(epochs_all.info, scalings=scalings)
X = epochs_all.get_data()
X = scaler.fit_transform(X)

out_t_labels = keras.utils.to_categorical(epochs_target.events
    [:, 2], 2)
out_n_labels = keras.utils.to_categorical(epochs_non_target.
    events[:, 2] - 10, 2)
out_labels = np.vstack((out_t_labels, out_n_labels))
```

Ukázka kódu A.6: Převedení jednotek a vytvoření seznamu zařazení do tříd

V druhé části ukázky se pro target a non-target epochy vytvoří pomocí metody z knihovny `Keras` odpovídající zařazení do tříd ve tvaru [1, 0] pro target epochu vyjadřující příslušnost k nulté třídě a ve tvaru [0, 1] pro non-

target epochu reprezentující zařazení do třídy 1. Množiny těchto přiřazení jsou poté spojeny do jednoho pole.

A.2 Neuronové sítě

Pro implementaci konvoluční i rekurentní neuronové sítě byla vytvořená jejich rodičovská třída s názvem *NN*, která obsahuje metody, jež jsou pro oba druhy sítí stejné, a to metody pro sestavení, trénování a testování sítě.

V ukázce kódu č. A.7 je zobrazena metoda pro sestavení modelu neuronové sítě, která sestává z volání metody *compile()* nad instancí třídy *Sequential* z knihovny `Keras`. Třída *Sequential* seskupuje sekvenční posloupnost vrstev do třídy *Model*.

```
def compile(self):
        self.model.compile(loss='binary_crossentropy', optimizer
           ='adam', metrics=[auc, 'binary_accuracy',
            keras_metrics.precision(), keras_metrics.recall()])
```

Ukázka kódu A.7: Metoda pro sestavení modelu neuronové sítě

Parametry této metody jsou binární křížová entropie jako ztrátová funkce, Adamův optimalizér a sledované metriky: preciznost a úplnost z knihovny `keras-metrics`, binární přesnost z knihovny `Keras` a AUC, kterou je nutné implementovat manuálně a nachází se v ukázce č. A.8.

```
def auc(y_true, y_pred):
    auc = tf.metrics.auc(y_true, y_pred)[1]
    K.get_session().run(tf.local_variables_initializer())
    return auc
```

Ukázka kódu A.8: Implementace metriky AUC

Metoda pro trénování modelu je zobrazena v ukázce kódu č. A.9. V prvním řádku metody dojde k inicializaci metody zpětného volání knihovny `Keras`, která zajistí ukončení trénování modelu v případě, že v pěti po sobě jdoucích iteracích trénování nedojde ke zlepšení hodnoty validační ztrátové funkce. Poté následuje volání metody *fit()*, která spustí učící algoritmus pro trénovací a validační množinu. Počet iterací trénování je ve třídě *Param* nastaven na 30 a počet vzorků použitých pro aktualizaci gradientu na 16. Parametr *verbose* definuje úroveň logování do konzole a může nabývat hodnot 0 pro žádný výpis do konzole, 1 pro detailní logování každé učící iterace a 2 pro jednořádkový výpis pro každou iteraci. V konfigurační třídě je nastaven na hodnotu 2. Ze

záznamu o procesu trénování se poté extrahují sledované metriky a ty jsou vráceny.

```
def fit(self, x_train, y_train, x_val, y_val):
    early_stopping = EarlyStopping(monitor='val_loss', patience
        =5, verbose=1, mode='auto')
    hist = self.model.fit(x_train, y_train, epochs=self.param.
        epochs, batch_size=16, shuffle=True, callbacks=[
        early_stopping], verbose=self.param.verbose,
        validation_data=(x_val, y_val))
    val_metrics = [hist.history['val_auc'][-1], hist.history['
        val_binary_accuracy'][-1], hist.history['val_precision'
        ][-1], hist.history['val_recall'][-1]]
    return val_metrics
```

Ukázka kódu A.9: Metoda pro trénování modelu neuronové sítě

Poslední metodou, která je pro oba druhy neuronových sítí společná, je metoda pro testování natrénovaného modelu s názvem *evaluate()*. Ta se nachází v ukázce č. A.10. Funkce této metody spočívá ve volání testovací metody z knihovny `Keras` s předanými testovacími daty, která vrací pole s požadovanými metrikami, ve kterém se ovšem na první pozici nachází hodnota testovací ztrátové funkce, jež se mezi sledovanými metrikami nenachází, a tudíž je z pole odstraněna.

```
def evaluate(self, x_test, y_test):
    metrics = self.model.evaluate(x_test, y_test, verbose=self.
        param.verbose)
    del metrics[0]
    return metrics
```

Ukázka kódu A.10: Metoda pro testování modelu neuronové sítě

Pro samotnou inicializaci rekurentní a konvoluční neuronové sítě jsou vytvořeny samostatné třídy, které dědí třídu *NN* a obsahují pouze metodu na inicializaci. V ukázce č. A.11 je zobrazena inicializace modelu LSTM sítě, jenž se nachází ve třídě *RNN* v souboru *rnn.py*. Volba vrstev a parametrů je popsána v části 5.3.3.

```
def __init__(self, channels, time_samples, param):
    dropout = .25
    self.model = Sequential()
    self.model.add(LSTM(input_shape=(channels, time_samples),
        units=100, return_sequences=True, activation='relu'))
    self.model.add(BatchNormalization())
```

```
        self.model.add(Dropout(dropout))
        self.model.add(LSTM(units=50, return_sequences=False,
            activation='relu'))
        self.model.add(BatchNormalization())
        self.model.add(Dropout(dropout))
        self.model.add(Dense(units=2, activation='softmax'))
        self.param = param
        self.compile()
```

Ukázka kódu A.11: Inicializace LSTM sítě

V A.12 je ukázána inicializace konvoluční neuronové sítě, jež se nachází ve třídě *CNN* v souboru *cnn.py*. Její architektura a parametry jsou popsány v sekci 5.3.2.

```
def __init__(self, channels, time_samples, param):
        self.model = Sequential()
        self.model.add(Conv2D(6, (3, 3), activation='elu',
            input_shape=(channels, time_samples, 1)))
        self.model.add(BatchNormalization())
        self.model.add(Dropout(0.5))
        self.model.add(AveragePooling2D(pool_size=(1, 8)))
        self.model.add(Flatten())
        self.model.add(Dense(100, activation='elu'))
        self.model.add(BatchNormalization())
        self.model.add(Dropout(0.5))
        self.model.add(Dense(2, activation='softmax'))
        self.param = param
        self.compile()
```

Ukázka kódu A.12: Inicializace konvoluční neuronové sítě

A.3 Lineární klasifikátory

Pro lineární klasifikátory implementované pomocí knihovny `scikit-learn` byla vytvořena třída *LinearClassifier*, která obsahuje metody pro inicializaci modelu klasifikátoru, jeho trénování a testování.

Metody pro inicializaci modelu a jeho trénování se nacházejí v ukázce č. A.13. Inicializační metoda pouze uloží vytvořenou instanci třídy *LinearDiscriminantAnalysis* nebo *SVC* z knihovny `scikit-learn` do proměnné s názvem *model*, aby se s ní mohlo dále pracovat.

Metoda pro trénování obsahuje volání metody *fit()* nad třídou daného klasifikátoru z knihovny `scikit-learn` s trénovacími daty. `Scikit-learn`

v trénovací metodě přijímá množinu zařazení do tříd ve formě indexu reprezentujícího příslušnost ke třídě, což znamená, že pro target epochu to bude 1 a pro non-target epochu 0. Tudíž z množiny zařazení, která byla vytvořena pro knihovnu `Keras`, bude vždy extrahován jen index na první pozici, jenž toto zařazení indikuje. Pro evaluaci pomocí validační množiny je využita metoda *evaluate()*, která bude popsána dále.

```
def __init__(self, model):
    self.model = model

def fit(self, x_train, y_train, x_val, y_val):
    self.model.fit(x_train, y_train[:, 0])
    return self.evaluate(x_val, y_val)
```

Ukázka kódu A.13: Metody na inicializaci a trénování lineárního klasifikátoru

V ukázce č. A.14 je zobrazena evaluační metoda, jež slouží k testování klasifikátoru a zjištění sledovaných metrik. Knihovna `scikit-learn` poskytuje automatickou metodu na ověřování klasifikace, která ale vrací jen metriku přesnost. V případě, že bychom chtěli jiné metriky, musíme je vypočítat manuálně z predikcí modelu. V této metodě se tedy pro všechny testované vzorky pomocí metody *predict()* zjistí, do jaké třídy klasifikátor daný vzorek zařadil, a tyto předpovědi uloží do pole *predictions*. Odpovídající správné přiřazení vzorků se uloží do pole s názvem *real_outputs*. Z těchto dvou polí se poté vypočítají sledované metriky pomocí knihovních funkcí `scikit-learn` z balíku *metrics*.

```
def evaluate(self, x_test, y_test):
    predictions = []
    real_outputs = []
    for i in range(x_test.shape[0]):
        pattern = x_test[i, :].reshape(1, -1)
        prediction = self.model.predict(pattern)
        predictions.append(prediction[0])
        real_outputs.append(y_test[i, 0])
    auc = metrics.roc_auc_score(real_outputs, predictions)
    acc = metrics.accuracy_score(real_outputs, predictions)
    prec = metrics.precision_score(real_outputs, predictions)
    recall = metrics.recall_score(real_outputs, predictions)
    return [auc, acc, prec, recall]
```

Ukázka kódu A.14: Metoda pro testování lineárního klasifikátoru

A.4 Průměrování časových oken

Pro lineární klasifikátory byla použita metoda průměrování časových oken pro extrakci příznaků. Její kód se nachází v ukázce č. A.15. Jako parametr přijímá metoda data ve tvaru 3D tensoru (počet_epoch x počet_kanálů x počet_hodnot). Parametry metody jako interval po stimulu, z něhož se budou příznaky získávat, počet příznaků a čas před stimulem, který byl použit pro extrakci epoch, jsou nastaveny v konfigurační třídě *Param*. Poté dojde k samotnému výpočtu jednotlivých průměrů v daných časových oknech. Výstupem je 2D tensor o rozměrech (počet_epoch x počet_příznaků), kde v našem případě je počet příznaků roven 60 (3 kanály x 20 příznaků) normalizovaný na nulovou střední hodnotu a jednotkový rozptyl.

```
def windowed_means(out_features, param):
    sampling_fq = param.t_max * 1000 + 1
    temp_wnd = np.linspace(param.min_latency, param.max_latency,
        param.steps + 1)
    intervals = np.zeros((param.steps, 2))
    for i in range(0, temp_wnd.shape[0] - 1):
        intervals[i, 0] = temp_wnd[i]
        intervals[i, 1] = temp_wnd[i + 1]
    intervals = intervals - param.t_min
    output_features = []
    for i in range(out_features.shape[0]):
        feature = []
        for j in range(out_features.shape[1]):
            time_course = out_features[i][j]
            for k~in range(intervals.shape[0]):
                borders = intervals[k] * sampling_fq
                feature.append(np.average(time_course[int(
                    borders[0] - 1):int(borders[1] - 1)]))
        output_features.append(feature)
    out = preprocessing.scale(np.array(output_features), axis=1)
    return out
```

Ukázka kódu A.15: Metoda pro průměrování časových oken

A.5 Řízení programu

Následuje popis souboru *main.py*, který slouží ke spuštění programu. První část jeho kódu se nachází v ukázce č. A.16. Z parametru příkazové řádky se do proměnné *classifier* uloží typ klasifikátoru, přičemž uživatel může volit ze 4 možností *'svm', 'lda', 'cnn', 'rnn'*. Při zadání špatného počtu

argumentů nebo jména klasifikátoru je program ukončen a vypsána chybová hláška s návodem ke spuštění. Následně dojde k inicializaci konfigurační třídy s definovanými všemi potřebnými parametry pro tento experiment. Poté dojde ke kompletnímu načtení analyzovaného datasetu. V případě, že bude pro klasifikaci použita konvoluční neuronová síť, dojde k následnému přidání jednotkové dimenze. Když bude použit lineární klasifikátor, bude aplikována extrakci příznaků.

```
classifier = sys.argv[1]
param = Param()
X, Y = data_loading.read_data(param)
if classifier == 'cnn':
    X = np.expand_dims(X, 3)
elif classifier == 'lda' or classifier == 'svm':
    X = windowed_means(X, param)
```

Ukázka kódu A.16: Příprava dat

Následuje rozdělení dat na trénovací a testovací množinu v poměru 75:25 a inicializace třídy *ShuffleSplit* z knihovny `scikit-learn`, která ve smyčce cross-validace zajistí náhodné rozdělení trénovacích dat na množinu, pomocí které bude provedeno samotné učení, a na validační množinu, také v poměru 75:25. Počet iterací Monte-Carlo cross-validace je nastaven na 30. V každé iteraci dojde k vypsání čísla aktuální iterace, inicializaci odpovídajícího klasifikátoru, spuštění trénování s trénovací a validační množinou a testování s testovací množinou. Kód této části je zobrazen v ukázce č. A.17. Všechny výsledky jsou uchovávány a po doběhnutí daného počtu iterací cross-validace zprůměrovány včetně jejich standardní směrodatné odchylky. Poté jsou dosažené výsledky vypsány do konzole.

```
x_train, x_test, y_train, y_test = train_test_split(X, Y,
    test_size=param.test_part, random_state=0, shuffle=True)
val = round(param.validation_part * x_train.shape[0])
shuffle_split = ShuffleSplit(n_splits=param.cross_val_iter,
    test_size=val, random_state=0)
val_results = []
test_results = []
iter_counter = 0
for train, validation in shuffle_split.split(x_train):
    iter_counter = iter_counter + 1
    print(iter_counter, "/", param.cross_val_iter, " cross-
        validation iteration")
    if classifier == 'cnn':
```

```
        model = cnn.CNN(x_train.shape[1], x_train.shape[2],
            param)
    elif classifier == 'rnn':
        model = rnn.RNN(x_train.shape[1], x_train.shape[2],
            param)
    elif classifier == 'lda':
        model = linear.LinearClassifier(
            LinearDiscriminantAnalysis(solver='eigen', shrinkage=
            'auto'))
    else:
        model = linear.LinearClassifier(SVC(cache_size=500))

    validation_metrics = model.fit(x_train[train], y_train[train
        ], x_train[validation], y_train[validation])
    val_results.append(validation_metrics)

    test_metrics = model.evaluate(x_test, y_test)
    test_results.append(test_metrics)
```

Ukázka kódu A.17: Trénování a testování ve smyčce Monte-carlo cross-validace

V ukázce č. A.18 se nachází konfigurační třída *Param*, jež obsahuje všechny nastavitelné parametry tohoto experimentu. V první části jsou definované hodnoty pro předzpracování dat včetně absolutní cesty k adresáři s datasetem, hodnot horní a dolní propusti pro filtrování, interval okolo stimulu pro extrakci epoch, interval pro korekci baseline, hodnota peak-to-peak amplitudy pro odstranění artefaktů a škálovací konstanta pro převod z V na μV.

V další části jsou nastaveny hodnoty potřebné pro průměrování časových oken. Jsou definovány hraniční hodnoty intervalu, ve kterém se tyto příznaky budou počítat, a také jejich počet.

Poslední pasáž obsahuje parametry pro samotnou klasifikaci, a to definice poměrných velikostí testovací a validační množiny, počet iterací cross-validace, počet iterací učení neuronových sítí a úroveň logování.

```
class Param:
    def __init__(self):
        #preprocessing
        self.path = '/home/filip/Dokumenty/FAV/DIP/
            PROJECT_DAYS_P3_NUMBERS/'
        self.l_freq = 0.1
        self.h_freq = 30
        self.t_min = -0.2
        self.t_max = 1
        self.baseline = (-0.2, 0)
        self.amplitude = 150e-6
        self.scaling = 1e6
```

```
        #features extraction
        self.min_latency = 0.3
        self.max_latency = 1
        self.steps = 20

        #classification parameters
        self.test_part = 0.25
        self.validation_part = 0.25
        self.cross_val_iter = 30
        self.epochs = 30
        self.verbose = 2
```

Ukázka kódu A.18: Konfigurační třída s nastavenými parametry

A.6 Spuštění programu

Program je spustitelný z příkazové řádky pomocí souboru *main.py* a přijímá jeden argument vyjadřující volbu klasifikátoru. Spouštěcí příkaz má následující tvar:

```
python main.py <klasifikátor>
```

Ve složce, ve které se program spouští, se musí nacházet všechny zdrojové soubory tohoto experimentu. Cestu k adresáři s EEG daty v absolutním tvaru je nutné specifikovat v konfigurační třídě *Param* v souboru *param.py*. V této třídě je možné nastavit všechny ostatní parametry relevantní pro tento experiment.

Uživatel má na výběr ze 4 druhů klasifikátorů, tudíž všechny možné varianty spuštění jsou následující:

- `python main.py lda`
- `python main.py svm`
- `python main.py cnn`
- `python main.py rnn`

Řešení je implementováno v Pythonu ve verzi 3.6.9 a verze jednotlivých použitých knihoven se nacházejí v souboru *requirements.txt*.

B Obsah DVD

Přiložené DVD obsahuje:

- složku `src` - zdrojové kódy experimentu,
- složku `data` - data 250 subjektů ve formátu `BrainVision` použitá v experimentu,
- složku `text` - text diplomové práce ve formátu PDF a zdrojové LaTeX soubory ve složce `latex`,
- složku `poster` - poster ve formátu PDF i PUB,
- soubor `README.txt` - tento popis obsahu DVD.