VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

KLASIFIKACE OCELOVÝCH STYČNÍKŮ V APLIKACI IDEA STATICA

CLASSIFICATION OF STEEL CONNECTIONS IN IDEA STATICA APPLICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ NEKUT**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. FRANTIŠEK V. ZBOŘIL, CSc.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Nekut Tomáš**
Program: Informační technologie
Název: **Klasifikace ocelových styčníků v aplikaci IDEA StatiCa**
Classification of Steel Connections in IDEA StatiCa Application
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s metodami strojového učení klasifikace dat.
2. Zorientujte se v typech svařovaných nebo šroubových spojů, patních plechů, patek a kotvení.
3. Navrhněte systém, který bude detekovat a klasifikovat přípoje ve styčníku.
4. Připravte vhodnou datovou sadu a její anotaci tak, aby obsahovala relevantní informace.
5. Implementujte navržený systém pomocí vhodných nástrojů.
6. Vyhodnoťte implementovaný systém na připravené sadě dat.
7. Zhodnoťte dosažené výsledky a diskutujte jejich další využití pro automatizaci návrhu styčníků.

Literatura:

- Chao, W.: Machine Learning Tutorial, National Taiwan University, 2011
- Molnar, Ch.: Interpretable Machine Learning, Christop Molnar, 2019
- Joints in Steel Construction: Simple Joints to Eurocode 3. 1. Ascot (Berkshire): The Steel Construction Institute, 2013. ISBN 978-1-85942-201-4
- Joints in Steel Construction: Moment-Resisting Joints to Eurocode 3. 1. Ascot (Berkshire): The Steel Construction Institute, 2013. ISBN 978-1-85-942209-0

Pro udělení zápočtu za první semestr je požadováno:

- První tři body zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Zbořil František V., doc. Ing., CSc.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 31. října 2019

Abstrakt

Cílem práce bylo navrhnout a implementovat aplikaci schopnou predikovat nejvhodnější typ spojení dvou ocelových prvků. Tato informace by mohla sloužit jako nápověda pro statika a zjednodušit mu tak návrh ocelové konstrukce. Vytvořená aplikace umožňuje automatické zpracovávání hotových projektových souborů, na kterých statikové pracovali, a přípravu datové sady extrakcí příznaků ze získaných dat. Dále je schopna na datové sadě natrénovat neuronovou síť a použít ji k predikci vhodného typu spojení doposud nespojených ocelových prvků. Během práce se podařilo dospět k nejvyšší úspěšnosti natrénovaného modelu 81 %. Predikce typu spojení pomocí umělé inteligence se zatím běžně nepoužívá, ale jak ukazuje i tato práce, tento postup by mohl fungovat a mohl by být použitelný.

Abstract

The goal of this thesis was to design and implement application which would be able to predict the most suitable connection type of two steel members. This information could be used as a clue by a structural engineer and so make design of steel construction easier for him. Implemented application is able to automatically process finished project files that were created by structural engineers and prepare a data set by extracting features from them. The application is then able to train a neural network on this data set and using it predict suitable connection type of unconnected steel members. The precision of model finally reached 81 %. Prediction of connection types using artificial intelligence is not widely used yet but could work and could be possibly usable as is shown also in this thesis.

Klíčová slova

neuronová síť, klasifikace, ocelový styčník, predikce spoje

Keywords

neural network, classification, steel connection, joint prediction

Citace

NEKUT, Tomáš. *Klasifikace ocelových styčníků v aplikaci IDEA StatiCa*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František V. Zbořil, CSc.

Klasifikace ocelových styčníků v aplikaci IDEA StatiCa

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci na téma Klasifikace ocelových styčníků v aplikaci IDEAStatiCa vypracoval samostatně pod vedením pana doc. Ing. Františka V. Zbořila, CSc. a uvedl jsem všechny prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Nekut
27. května 2020

Poděkování

Rád bych tímto poděkoval vedoucímu mé bakalářské práce panu doc. Ing. Františkovi V. Zbořilovi, CSc. za cenné rady, podněty a doporučení, které mi během zpracovávání práce pomohly. Také bych chtěl poděkovat společnosti IDEA StatiCa za ochotu při konzultacích ohledně dané problematiky.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá použitím strojového učení k zjednodušení práce projektanta, který navrhuje ocelové konstrukce v aplikaci Connection společnosti IDEA StatiCa. Projektant nejprve navrhne podobu ocelové konstrukce, tedy umístění nosných sloupů, průvlaků, stropních profilů atd. Následně musí navrhnout spoje mezi těmito prvky. Spoj může být založený na různém principu spojení materiálu, může se jednat o svařovaný nebo například šroubovaný spoj. Program by měl být schopen projektantovi napovídat, jaký typ spoje by bylo vhodné použít.

Po návrhu konstrukce proběhne v aplikaci Connection výpočet, zda konstrukce vyhoví, což spočívá v simulaci jejího chování při zadaném zatížení. Je třeba zjistit využití materiálu v jednotlivých částech konstrukce. Optimální je, aby využití materiálu bylo rovno 100 %. Využití větší než 100 % znamená, že konstrukce zadané zatížení nevydrží, využití menší než 100 % sice znamená, že konstrukce zatížení odolá, ale bude spotřebováno zbytečně více materiálu, než je potřeba. Jako vyhovující konstrukce se tedy považuje ta, jejíž využití materiálu se blíží 100 %, a to s dostatečně velkou rezervou, aby byla konstrukce bezpečná.

Výpočet, zda konstrukce vyhoví, může být analytický, vzorec je odvozen pro konkrétní typ spojení ocelových prvků, přičemž se většinou jedná o nějaký jednoduchý často používaný spoj. Výpočet je velmi rychlý, ale vzorec není možné použít pro jiný typ spoje, než pro který byl specificky odvezen. Druhou možností je použití některé numerické metody pro simulaci chování konstrukce. Konstrukce je rozdělena na malé prvky a využití materiálu je počítánu v každém prvku zvlášť. Toto řešení je obecné a nezáleží na typu spojení, ale výpočet trvá mnohem déle, i několik minut. Aplikace Connection používá numerickou metodu konečných prvků.

Vzhledem k tomu, že výpočet každého posouzení, zda konstrukce vyhoví, nebo ne, trvá poměrně dlouho, není možné, aby software zkoušel různé možnosti, jaký spoj použít, protože by to trvalo příliš dlouho. Projektant musí mít na základě zkušenosti vždy představu, jak by měl spoj vypadat, nemůže zkoušet všechny možnosti. Tato projektantova zkušenost by mohla být částečně nahrazena nebo doplněna právě strojovým učením.

Výsledný program by měl být schopen zpracovávat projekty, které už nějaký projektant navrhnul, a získávat z nich informace o daném spoji a zároveň typu připojení, který byl v tomto konkrétním případě použit. V další fázi by měl být schopen využít tato data k natrénování vhodného modelu. Následně by měl být program schopný v projektu, ve kterém nejsou zatím určené typy spojů mezi ocelovými prvky, určit, jaké spoje jsou nejvhodnější.

Kapitola 2

Pojmy

Prut (angl. beam) je prvek konstrukce. Konstrukci tvoří vzájemně spojené pruty. Je obecně z libovolného materiálu, pro tuto práci však výhradně z oceli. Vzniká pohybem těžiště rovinného tvaru po přímce na tuto rovinu kolmou. Zmíněný rovinný tvar se nazývá **průřez** a přímka **střednice** prutu.

Zatížení prutu představuje síly a momenty sil, které na prut působí. Uvádějí se na koncích prutů. Znaménko dané hodnoty vyjadřuje vždy směr dané síly nebo momentu. Zatížení se skládá ze síly rovnoběžné s prutem, jedná se o tlakovou, nebo tahovou sílu (podle směru do prvku nebo od prvku, záleží na znaménku). Dále smykovou sílu ve směru y a z, opět v obou směrech podle znaménka. Nakonec pak moment síly ve všech osách, tedy kroucení kolem osy x a ohyb kolem os y a z. Zatížení vyplývá z působení hmotnosti samotné konstrukce, hmotnosti vybavení budovy, působení sněhu, větru atd. Jedna kombinace hodnot všech sil a momentů sil se označuje jako **kombinace zatížení** (angl. load effect). Každý prut má obecně libovolné množství takových kombinací, například kombinace v klidu, v případě větru ze severu, v případě zasněžení atp. Konstrukce pak musí vydržet všechny kombinace.

Styčník neboli **přípoj** (angl. connection) označují spojení několika prutů v jednom bodě. Většinou se dá ve styčníku identifikovat jeden nosný prut, ostatní se nazývají připojované a jsou připojovány k nosnému. Pokud mluvíme o spojení v jednom bodě, myslí se tím protnutí střednic všech prutů v jednom bodě, protože všechny pruty většinou nemohou být zavedeny vzhledem k výškám jejich průřezů do jednoho bodu. V praxi se však nutně nemusí ani jednat o jediný bod a prvky mohou být vzájemně nepatrně posunuté a jejich střednice se neprotnou zcela přesně. **Topologie** styčníku označuje jeho podobu, tedy kolik prutů je ve styčníku spojeno a jakým směrem jsou jednotlivé pruty orientované.

Připojení (angl. joint) představuje jedno konkrétní napojení prutu na jiný. To znamená, že styčník se skládá z několika připojení. Připojení mohou být různých typů založených na různých principech spojení prutů.

IDEA StatiCa Connections je aplikace pro návrh a posouzení ocelových styčníků. V aplikaci je možné do prostoru umisťovat ocelové pruty různých průřezů a rozměrů. Dále je možné zadávat zatížení prutů v podobě velikosti sil či momentů sil. Následně je možné sestavit seznam výrobních operací o pevném pořadí, které kdyby byly na styčník postupně aplikovány, vedlo by to k propojení všech prvků styčníků k sobě. Příkladem operace by mohl být svar, aplikovaný na dva prvky, kdy jeden je touto operací přivařený k druhému. Parametry této operace pak mohou být tloušťka svaru, délka svaru, atd. V aplikaci Connections je kolem dvaceti různých operací, jejichž kombinací a určováním parametrů každé operace je možné propojovat pruty mezi sebou. Posledním krokem je provedení výpočtu,

který ověří, zda navržený styčník vyhoví, tedy vydrží zadané zatížení a zároveň je dostatečně ekonomický, aby se při stavbě konstrukce spotřebovalo co nejméně materiálu.

Kapitola 3

Využití aplikace

Jak bylo naznačeno v úvodu, aplikace by měla sloužit jako nápověda statikovi při návrhu ocelové konstrukce. Měla by zvládnout správně určit typ spoje u jednodušších často opakovaných spojů.

Vstupem pro návrh je umístění prutů v prostoru a hodnoty jejich zatížení. Výsledkem pak pořadí operací včetně jejich parametrů, které propojí všechny pruty ve styčníku. Kompletní návrh se dá rozdělit do tří fází. Nejprve je potřeba určit, které dvojice prvků mají být spojeny mezi sebou. V nejjednodušším případě stačí identifikovat nosný prut a ostatní pruty ve styčníku připojit k němu. U komplikovanějších přípojů však může být prvek připojený k jinému než nosnému prvku. Pro tuto práci, jsem se rozhodl předpokládat, že všechny prvky mimo nosný jsou připojeny k nosnému. Dále je třeba určit typ spojení u každé dvojice spojovaných prutů, tedy jaká operace má být k připojení použita. Pro zjednodušení byly vybrány jen některé operace, které se v aplikaci Connections používají nejčastěji.[1] Nakonec je potřeba určit parametry zvolených operací. Určování parametrů by mohlo probíhat na základě jednoduchých pravidel, ne pomocí umělé inteligence, proto jsem se rozhodl určování parametrů v práci neřešit. Například u operace úhelník je výška připojujícího plechu (úhelníku) jen jednoduše závislá na výšce připojovaného prutu a je jednoduché pro tento parametr definovat pravidlo určení jeho hodnoty. Každá operace má ale řádově deset až dvacet parametrů a je možné, že některý z nich by nebylo tak snadné automaticky určovat. Pak by teoreticky mohlo být nutné využít některý algoritmus strojového učení i na určování jednotlivých parametrů operací, což by mohlo být vhodným pokračování této práce.[2]

Návrh styčníku byl tedy zjednodušen na určení vhodného typu spojení (vhodné operace) či více vhodných, a to pro všechny připojované pruty, tedy všechny kromě nosného. Tato informace by pak mohla statikovi sloužit jako nápověda při návrhu styčníku. Jak bylo řečeno, práce se soustředí na nejpoužívanější operace z aplikace Connections, cílem je tedy, aby výsledkem práce by nástroj, který bude schopen predikovat použitou operaci u těch nejběžnějších styčníků. Každá konstrukce se totiž skládá z velkého množství jednoduchých často opakovaných spojení, se kterými by se pak statik nemusel zdržovat a mohl by se soustředit spíše na složitější netradiční spojení prutů související například s ne úplně běžným tvarem konstrukce apod.

[1] Kapitola 5.1 Typy spojů
[2] Kapitola 8 Možná pokračování práce

Kapitola 4

Postup řešení problému

Vzhledem k tomu, že jako konkrétní typ klasifikátoru, který bude predikovat typ spojení, byla zvolena neuronová síť, bylo nejprve nutné připravit datovou sadu pro trénování, kde by každý vzorek obsahoval vektor vstupních příznaků a jim odpovídající předpokládaný vektor výstupních příznaků. Protože jeden vzorek odpovídá jednomu připojení, bylo nutné rozpoznat důležité vlastnosti připojení a poté je transformovat na příznaky. Nejprve jsem musel zpracovat projektové soubory, vytvářené statiky, které jsem dostal k dispozici. Navrhl jsem nástroj pro jejich zpracovávání a za účelem ukládání získaných vlastností jsem navrhl schéma databáze. Následně jsem navrhl a implementoval další část aplikace, která využívá uložené informace z databáze a extrahuje z nich příznakové vektory. Takto jsem připravil datovou sadu pro trénování neuronové sítě.

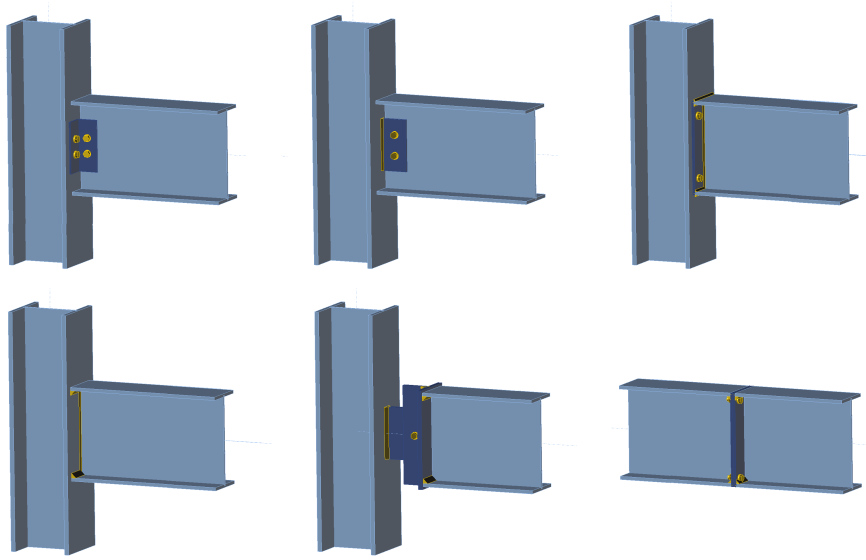
Další částí bylo využití připravené datové sady a natrénování neuronové sítě. Tento proces byl iterativní, jelikož jsem v rámci něj opakovaně měnil jak podobu neuronové sítě, tedy počty vrstev či počty neuronů, ale také podobu datové sady přidáváním nových příznaků. Tímto procesem jsem se snažil dosáhnout co nejvyšší úspěšnosti netrénovaného klasifikátoru.

Kapitola 5

Spoje mezi ocelovými prvky

5.1 Typy spojů

Typů spojů neboli spojovacích výrobních operací nabízí aplikace Connections celou řadu, pro tuto práci však byly vybrány ty nejčastěji používané. Podobu jednotlivých spojů ukazuje obrázek 5.1.



Obrázek 5.1: Vybrané typy spojů. Na prvním řádku jsou to z leva operace úhelník, žiletka a koncová deska, na druhém pak svaření, kloub a operace prut k prutu.

Výsledná aplikace by měla být schopná zatím nespojené dvojici prutů přiřadit některý z těchto typů spojení.

5.2 Vybrané vlastnosti

Aby bylo možné vytvořit datovou sadu pro trénování klasifikátoru, bylo nejprve nutné určit skupinu vlastností připojení, na základě které by model typ spoje predikoval. Aby vybrané informace postihly vše potřebné k rozhodnutí mezi spojovacími operacemi, byly

vybrány na základě informací, které potřebuje statik při návrhu spojů. Pokud by ve vybraných vlastnostech nějaká zásadní chyběla a statik by nebyl schopen mezi spojeními vybrat, pravděpodobně by se to nepodařilo ani klasifikátoru.

Potřebnými vlastnostmi jsou parametry obou prutů, tedy jak připojovaného tak nosného. A to typ průřezu, tedy například zda se jedná o trubku, tyč, hranatý profil, profil I, atd. Důležitá je i velikost profilu tedy jeho výška a šířka. Další potencionálně důležité vlastnosti jsou posuny prutů oproti přesnému bodu styčníku, rotace prutu okolo střednice, informace o tom, zda je prut nějakým způsobem zrcadlen a zda je průběžný, či v bodě spojení končí. Vzájemnou polohu prutů pak definuje další vlastnost, kterou je vektor směru prutu směřující od bodu styčníku rovnoběžně se střednicí prutu. Dalšími důležitými vlastnostmi jsou i hodnoty zatížení prutů. Později byla mezi vlastnosti přidána ještě topologie styčníku, jednoduše řečeno informace o tom, jak vypadá zbytek styčníku, tedy jaké další pruty, které se přímo netýkají daného připojení, styčník obsahuje. Poslední potřebnou vlastností je použitá spojovací operace.

Tyto vybrané vlastnosti pak musela aplikace získávat z dostupných projektových souborů v rámci fáze těžení dat. Získané informace pak ukládat do databáze, aby je bylo později možné použít k trénování modelu strojového učení.

Kapitola 6

Návrh aplikace

Jak bylo nastíněno, aplikace by měla být schopná zpracovávat projekty, které vytváří statikové, z projektů extrahovat jednotlivé styčníky a z nich konkrétní připojení prutů na nosný prut a na základě těchto informací trénovat klasifikátor, aby byl schopný v projektu, kde zatím není provedeno spojení prutů, predikovat, jakým způsobem by jednotlivé pruty mohly být na nosný prut připojeny. Tedy určovat nejvhodnější operaci.

Z toho vyplývá rozdělení projektu na modul, který pracuje s databází, dále modul, který obstarává zpracovávání projektů, a tedy získávání dat pro učení, dále modul samotné klasifikace a nakonec modul reprezentující fasádu celé aplikace, se kterým poté mohou externí moduly komunikovat a využívat funkcionality aplikace, bez znalosti jejich detailů.

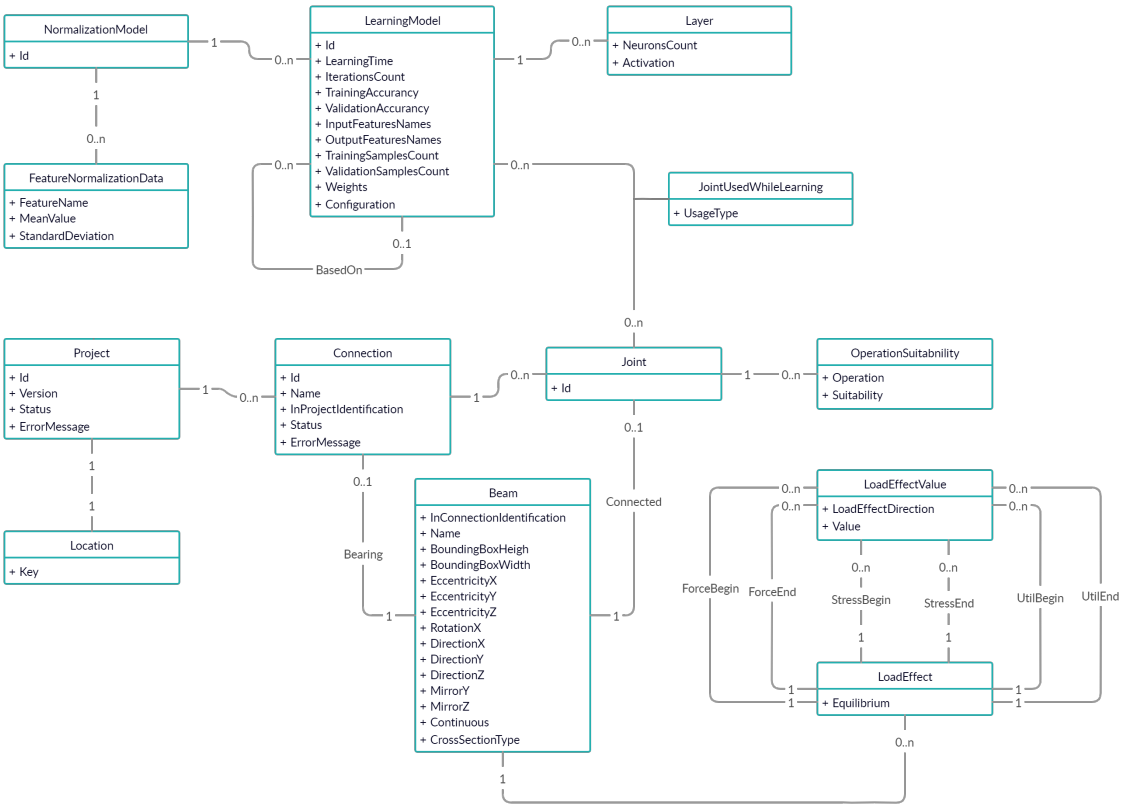
Pro implementaci jsem zvolit prostředí .NET a aplikaci jsem programoval v jazyce C#.

6.1 Datová vrstva

Protože je aplikace založená na získávání znalostí z reálných projektů vytvořených statiky, je nutné, aby zahrnovala práci s databází. Data, na kterých je následně trénován klasifikátor, jsou poměrně cenná, ve smyslu, že jejich získání ze vstupních projektů spotřebovává nezanedbatelný výpočetní čas, stejně tak i trénování modelu zabere jistý čas. Kdyby aplikace neměla k dispozici uložiště, bylo by nutné potřebná data z projektů extrahovat pokaždé, když by měla být aplikace spuštěna, a model trénovat pokaždé, když by měl být použit ke klasifikaci, což by bylo velmi nepraktické. Proto jsem do projektu zahrnul modul pro práci s databází, který ostatní moduly využívají k ukládání již zpracovaných dat.

6.1.1 Schéma databáze

Jak bylo řečeno, je nutné ukládat data získaná z projektů, na kterých je následně trénován klasifikátor a zároveň je potřeba ukládat stav modelu po trénování, aby ho bylo možné později použít. K tomu přibyla ještě potřeba ukládat informace o rozložení trénovacích dat, aby bylo možné provádět normalizaci. Z toho vyplývá ER diagram na obrázku 6.1.



Obrázek 6.1: ER diagram znázorňující schéma databáze.

Základ tvoří čtyři entity `Project` (projekt), `Connection` (styčník), `Joint` (připojení) a `Beam` (prut). Jeden projekt může obsahovat několik navržených styčníků a styčník se skládá z několika připojení prutů k jednomu nosnému. Entity obsahují informace získané ze zpracovávaných projektů.

Entita `Project` obsahuje verzi získaných dat, aby bylo možné identifikovat případné zastaralé záznamy v databázi, pokud by došlo k úpravě modulu, který data extrahuje z projektů. Status a chybová hláška jsou využity pro uložení informace, že při načítání daného projektu došlo k chybě, projekt pak na základě této informace není používán při učení, protože pravděpodobně obsahuje nesmyslná nebo nepřípustná data. Přestože se data následně nepoužívají, je záznam o tomto projektu vložen do databáze, aby příště při spuštění zpracovávání projektů nebyl již znovu načítán. Vazba na entitu `Location` reprezentuje umístění projektu, tedy odkud byl získán. Tako informace je potřebná pro identifikaci daného projektu a proto, aby projekt bylo možné získat opakovaně. Při analýze funkčnosti klasifikačního modelu, je vhodné mít k dispozici zdrojové projekty, ze kterých byla data extrahována, aby bylo možné v případě, že přípoje daného projektu nebyly správně klasifikovány, zkontrolovat v originálním projektu, proč k tomu došlo. Entita `Location` obsahuje položku klíč, který jednoznačně identifikuje soubor v uložišti projektů. Přestože se jedná jen o tuto položku, je umístěna do samostatné entity, protože v budoucnu v rámci pokračování na projektu by mohlo být vhodné využívat více různých uložišť a entita by potom obsahovala kromě identifikace projektu i typ uložiště a další informace potřebné k získání souboru z daného uložiště. Případně by mohlo existovat více záloh daného souboru na různých uložištích a entita Project by měla vázáno více entit `Location`.

Entita `Connection` reprezentuje jeden navržený styčník v projektu. Obsahuje název styčníku, jak byl pojmenovaný statikem, jednoznačnou identifikaci v rámci projektu, a status a chybovou hlášku ze stejného důvodu jako entita `Project`. Jak bylo řečeno, zjednodušeně se dá říct, že všechny pruty ve styčníku jsou připojeny ke stejnému prutu, označovanému jako nosný. Proto má styčník vazbu na jednu entitu `Beam` označenou v ER diagramu `Bearing` (nosný).

Entita `Joint` představuje jedno připojení, tedy spoj jednoho prutu s nosným prutem. Entita má vazbu na jednu entitu `Beam`, která reprezentuje připojovaný prut, vazba `Connected` (připojený). Informace o vhodném typu připojení prutu, tedy vhodné operaci, je uložena díky vazbě na entitu `OperationSuitability`, která obsahuje, o kterou operaci se jedná a zda je vhodná. U obou položek se jedná o výčtový typ, v případě operace množina hodnot odpovídá vybraným operacím aplikace Connections a v případě vhodnosti se pak jedná o hodnoty vhodná, částečně vhodná a nevhodná. Pro dané připojení může být obecně vhodné více různých typů spojení, tedy více operací, z toho vyplývá použitá kardinalita.[1]

Entita `Beam` reprezentuje prut. Prut může být buď nosný, nebo k nosnému připojovaný, nemůže plnit obě funkce. Může být tedy navázán buď na entitu `Connection` nebo `Joint`. Entita obsahuje název, kterým daný prut pojmenoval statik a vybrané vlastnosti potřebné k definování podoby daného prutu.[2] Statik jednotlivým prutům zadává jejich zatížení v rámci konstrukce.[3] Jedná se o kombinaci hodnot sil a momentů sil. Jednu kombinaci reprezentuje entita `LoadEffect`. Těchto kombinací může být statikem zadaných více. Hodnotu konkrétního zatížení obsahuje entita `LoadEffectValue` v položce `Value`, položka `LoadEffectDirection` obsahuje směr působení zatížení, tedy hodnotu z výčtu tlaková/tahová síla ve směru x, smyková sílu ve směru y, smyková síla ve směru z, kroucení kolem osy x, ohyb kolem osy y, nebo ohyb kolem osy z. Hodnoty sil resp. momentů sil na začátku prutu (v bodě spojení ve styčníku) jsou vázány vazbou `ForceBegin`, stejné hodnoty pro konec prutu pak vazbou `ForceEnd`. Vazby `StressBegin`, resp. `StressEnd` váží hodnotu napětí (angl. stress) v prutu v daných směrem. Napětí je dopočítáno na základě typu průřezu a hodnoty dané síly či momentu síly ve stejném směru. Stejně tak je dopočítáno i využití (angl. utilization), které vyjadřuje, jaké části maximálního zatížení prutu daná hodnota dosahuje, a je navázáno vazbou `UtilBegin`, resp. `UtilEnd`.

Databáze je navržena tak, že obsahuje, co nejvíce informací, které se dají z projektů získat, a to i některé, které nemusí být později při trénování modelu použity. Daná informace totiž může být na vstup klasifikátoru doplněna později, pokud se to ukáže jako vhodné, a v tom případě pak nemusí být opětovně prováděno zpracovávání všech projektových souborů, jelikož se jedná o poměrně časově náročný úkon.

Schéma dále obsahuje entity sloužící k ukládání informací o natrénovaném klasifikátoru a normalizačním modelu.

Entita `Learnig Model` reprezentuje natrénovanou neuronovou síť, která je do databáze ukládána, aby ji bylo později možné využít ke klasifikaci neznámých připojení. Entita obsahuje základní informace o modelu, tedy dobu trénování, počet epoch trénování a úspěšnost na trénovací a validační sadě. Dále obsahuje jména vstupních a výstupních příznaků, aby bylo možné při použití modelu vstupy a výstupy interpretovat a přiřadit jim význam. Dalšími položkami jsou počet trénovacích a validační vzorků, které byly použity při trénování. Nakonec entita obsahuje atributy `Weights` a `Configuration`, které slouží k uložení a ob-

[1] Schéma databáze bylo později upraveno, jelikož se zjistilo, že pro některá připojení může vyhovovat více různým typům připojení viz kapitola 7.5.5 Problém s disjunktností tříd

[2] Kapitola 5.2 Vybrané vlastnosti

[3] Zatížení popsáno v kapitole 2 Pojmy.

novení natrénovaného stavu modelu. Vrstvy neuronové sítě reprezentují entity `Layer`, které obsahuje atributy počet neuronů a typ aktivační funkce. Aby byla k dispozici informace o tom, které vzorky, tedy připojení, byly použity pro trénování nebo validaci, jsou entity `Joint` napojeny k modelu pomocí vazby `JointUsedWhileLearning`. Parametrem vazby je typ použití, tedy zda se jedná o připojení, na kterém se model trénoval, nebo validoval. Tato informace je nutná pro navazující učení modelu, aby k učení byla použita jen připojení, která ještě použita nebyla, a také aby bylo možné zpětně analyzovat správnost modelu procházením použitých připojení. Zmíněné navazující učení modelu je modelováno vazbou `BasedOn`, která spojuje model s modelem, ze kterého vychází.

Entita `Normalization Model` reprezentuje model statistického rozložení příznaků. Klasifikátor je vázán na normalizační model, který používá k normalizaci příznaků. Na entitu normalizačního modelu jsou vázány entity `FeatureNormalizationData` reprezentující informace o jednotlivých příznacích, které obsahují název příznaku, střední hodnotu a směrodatnou odchylku.

6.1.2 Implementace

Pro implementaci uložiště pro tento projekt byl vybrán databázový systém MongoDb. Jedná se o zástupce NoSQL databáze, která je místo na tradiční relačním pojetí založená na ukládání dat do dokumentů ve formátu JSON.

Databáze MongoDb umožňuje vztahy entit reprezentovat dvěma způsoby, a to odkazování, kdy entita obsahuje id jiné entity, nebo vnořování, kde entita přímo obsahuje hodnotu jiné entity. V této aplikaci by mohla entita `Project` přimo obsahovat kolekci entit `Connection`, které by pak obsahovaly kolekce entit `Joint`. Rozhodl jsem se však toto provázání implementovat pomocí odkazování, a to tak, že entita `Joint` obsahuje id entity `Connection` a stejně tak entita `Connection` obsahuje id entity `Project`. Důvodem je přístup k entitám `Joint`, dochází k němu totiž i přímo, ne jen přes entity `Connection` a `Project`, a to v případě vybírání vzorků určených k učení, protože jedna entita `Joint` právě představuje jeden trénovací či validační vzorek. Z toho důvodu obsahují tyto entity atribut id. Entity `Location`, `OperationSuitability`, `Beam`, `LoadEffect` i `LoadEffectValue` jsou vnořeny v rámci entit, na kterých závisí, zde jsem nedošel k žádnému důvodu, proč by mělo být použito odkazování. Co se týče zbytku schématu, entita `LearningModel` se odkazuje na model, ze kterého vychází pomocí id, stejně jako na entitu `NormalizationModel`. Zbytek, tedy entity `Layer` a `FeatureNarmalizationData`, jsou vnořené. Vazba N:M mezi klasifikátorem a použitými připojeními je implementována spojovací entitou `JointUsedWhileLearning`, která obsahuje id entity `Joint` a `LearningModel` a také atribut `UsageType`.

Aby bylo možné snadno změnit konkrétní implementaci databáze, pokud by to bylo potřeba a také odstínit zbytek aplikace od detailů práce s databází, byl využit návrhový vzor repozitář a konkrétní implementace je schována za rozhraním. Rozhraní obsahuje jen základní operace nad daty jako výběr všech záznamů, výběr podle identifikace, vložení, úprava nebo mazání. Jedná se o rozhraní čtyř repozitářů.

`IProjectRepository` definuje rozhraní repozitáře pro přístup k datům projektů. Obsahuje operace pro výběr všech záznamů, výběr podle id a podle lokace, vložení záznamu, vložení seznamu záznamů, úpravu a smazání záznamu. Při výběru projektu je entita repozitářem provázána se svými styčníky a styčníky se svými připojeními.

`IJointRepository` je rozhraní repozitáře pro přístup k entitám `Joint`. K jednotlivým připojením se dá přistoupit i přes entitu styčník a projekt přes projektový repozitář uvedený výše, ale pro efektivnější výběr entit `Joint`, které představují jednotlivé vzorky k učení, byl

vytvořen i tento ropozitář. Protože jednotlivá připojení jsou vkládána v rámci styčníků a ty v rámci projektů přes projektový repozitář uvedený výše, nedává smysl, aby tento repozitář obsahoval jiné než čtecí operace. Nedává totiž smysl, aby byly do databáze vkládány samostatná připojení bez styčníku a projektu. Rozhraní tedy definuje pouze operace výběr všech záznamů a výběr podle id.

`ILerningModelRepository` definuje operace sloužící k práci s natrénovanými klasifikátory. Obsahuje operace pro výběr všech záznamu, výběr podle id a vložení nového záznamu. Při výběru je entita repozitářem provázána s použitým normalizačním modelem a přes spojovací entity s použitými připojeními. Operace pro úpravu a mazání záznamů k dispozici nejsou, protože se nepředpokládá, že by tyto operace měly být využívány, vzhledem k tomu, že je záměrně ukládána celá historie natrénovaných modelů a v případě pokračování trénování modelu se vždy vytváří kopie.

`INormalizationModelRepository` obsahuje definici operací pro přístup k normalizačním modelům. Jedná se stejně jako u entity reprezentující natrénované klasifikátory o operace výběr všech záznamu, výběr podle id a vložení nového záznamu.

Všechna rozhraní jsou implementována pomocí zmíněné databáze MongoDb, ale bylo by snadno možné v případě potřeby napsat jinou implementaci bez nutnosti změny zbytku aplikace.

6.2 Společná vrstva

Protože při výběru projektů, které se budou zpracovávat, tedy ze kterých se budou získávat data, stejně tak při výběru připojení, která se použijí pro trénování modelu, dochází k filtrování projektů resp. připojení, rozhodl jsem se vytvořit generické rozhraní `ISelector<T>`, které představuje filtr obecných prvků zadaného typu `T`. Rozhraní jsem umístil do společného modulu Common, aby k němu měly přístup oba moduly, které zmíněnou filtraci provádějí. Rozhraní obsahuje jednu metodu, a to `select`, která jako vstup získá enumeraci prvků typu `T` a jejím výstupem je filtrovaná enumerace typu `T`. Často filtrace nebere v potaz pouze jedno kritérium, ale je kombinací více kritérií, např. výběr připojení, která byla z projektu bezchybně extrahována a zároveň nebyla při trénování daného modelu ještě použita. Proto jsem do stejného modulu umístil implementaci rozhraní, a to `AndSelector<T>`, která představuje operaci and nad několika filtry prvků typu `T`. Objekt `AndSelector` v konstruktoru získá seznam objektů implementujících `ISelector<T>` a při volání `select` provede sériově filtraci všemi filtry, představuje tak filtr, jehož výstupem jsou pouze prvky, které vyhoví všem zadaným filtrům.

6.3 Vrstva zpracování projektů

Prvním hlavním modulem, na kterém aplikace stojí je modul zajišťující těžení dat z projektových souborů vytvořených statiky v aplikaci Connections. Skládá se ze tří celků, a to z části zajišťující poskytování projektů z externího uložiště či lokálního souborového systému, části zodpovědnou za filtrování projektů a části, která provádí těžení potřebných dat ze souborů. Modul využívá výše popsanou databázi k ukládání získaných dat.

6.3.1 Poskytování projektů

Protože by zbytek aplikace neměl být závislý na tom, odkud jsou projektové soubory získávány, všechny třídy určené k poskytování projektů implementují `IProjectProvider`, roz-

hraní které obsahuje metodu `NextProject` vracející objekt `ProvidedProject`. Ten obsahuje cestu k lokální kopii souboru a objekt `Location` zmíněný výše, který definuje, odkud byl soubor získán.[4]

Poskytovatel projektů navíc v konstruktoru získá objekt filtru implementující rozhraní `ISelector<Location>`, který je určený k výběru vhodných projektových souborů.[5] Všechny implementace poskytovatele projektů dědí z třídy `ProjectProviderBase`, která implementuje zmíněnou metodu `NextProject`. Dále definuje dvě abstraktní metody, které jsou specifické pro každou implementaci. Jedná se o `GetProjectsLocations` a `GetProjectPath`. První vrací enumeraci objektů `Location` všech projektových souborů, které je možné poskytnout, druhá dle zadaného objektu `Location`, vrátí lokální cestu k žádanému souboru. Metoda `NextProject` tedy při prvním zavolání zjistí všechny dostupné projekty, předá enumeraci objektů `Location` filtru projektů, uloží si vyfiltrované objekty a u prvního z nich zjistí lokální cestu, kterou vrátí. Při následujících voláních už jen prochází seznam objektů a postupně vrací lokální cesty.

Aktuálně jsou soubory projektů uloženy v uložišti Amazon S3, ale jak bylo uvedeno výše je možné jednoduché rozšíření o jiná uložiště. `GetProjectsLocations` je v tomto případě implementováno dotazem na uložiště, díky čemuž jsou získány dostupné projekty. `GetProjectPath` pak provede stažení zadaného souboru z uložiště do lokálního souborového systému a vrátí lokální cestu. Protože je nutné mít možnost načíst i projekt uložený lokálně, a to při klasifikaci připojení v rozpracovaném projektu, byl implementován i poskytovatel projektů, který toto zajistí. V konstruktoru získá cestu k lokálnímu projektovému souboru, který má poskytnout. Při zavolání `GetProjectsLocations` vrátí právě tento projekt a `GetProjectPath` samozřejmě vrátí lokální cestu, která byla na začátku zadána. Tento postup není příliš přímočarý, ale byl zvolen, aby bylo důsledně dodrženo navržené rozhraní.

6.3.2 Filtrování projektů

Jak bylo popsáno výše, projektové soubory, které se budou zpracovávat, se ze všech dostupných vybírají pomocí filtru projektů. Každý filtr implementuje `ISelector<Location>` a je možné jejich kombinování pomocí `AndSelector`. Základní filtr `UnseenProjectsSelector` vybírá pouze projekty, které zatím nebyly zpracovány a nenacházejí se tak v databázi. Zda je daný projekt v databázi, je zjišťováno porovnáváním atributu `Location`. Pro případ, kdy je potřeba zpracovávat opětovně zastaralé záznamy, jsou implementovány filtry pracující s verzí záznamů, tedy `VersionLessThanSelector`, který vybírá projektové soubory, které mají v databázi zaznamenanou verzi menší než zadaná hodnota, jsou tedy zastaralé a je třeba data extrahovat znovu. Dále byl implementován filtr s opačnou funkcionalitou, a to `VersionGreaterThanSelector`, který je pak možné kombinovat pomocí `AndSelector` s výše uvedeným pro výběr projektů s verzí z daného rozsahu.

6.3.3 Těžení dat

Samotné zpracovávání projektů zajišťuje třída `DataMiner`, která poskytuje metodu `Mine`, jejímž vstupem je objekt poskytovatele projektů popsaný výše a výstupem pak enumerace entit `Project`, které obsahují extrahovaná data.

Extrakce je založena na komunikaci s komponentami aplikace Connections a využívání jejich funkcionalit, které umožňují soubor načíst, zpracovat a následně přistupovat k jednot-

[4] Kapitola 6.1.1 Schéma databáze
[5] Kapitola 6.2 Společná vrstva

livým styčníkům obsaženým v projektu, v rámci styčníků k jednotlivým prutům, zátěžovým kombinacím a použitým operacím a získat z nich hodnoty vybraných vlastností.[6]

Pokud během extrakce dojde k neočekávané chybě, je pravděpodobné, že data v souboru nejsou v pořádku a není žádoucí, aby se na těchto datech následně trénoval klasifikátor. Proto jsou v entitách `Project` i `Connection` vyhrazeny atributy stav a chybová hláška pro zaznamenání vzniklé chyby.

Protože třída zodpovědná za extrakci dat z projektů nemusí být konečná a je možné, že bude upravována, ať už kvůli přidávání nových funkcionalit, tedy přidávání nových vlastností, které jsou z projektů získávány, nebo kvůli chybě, která musela být opravena, může se stát, že staré záznamy o zpracovaných projektech nebudou kompatibilní s novými. Příkladem může být právě přidání nové vlastnosti. Bude přidán například nový atribut materiál do entity `Beam`, který říká z jakého materiálu je prut vyroben. Pokud bychom poté chtěli natrénovat nový model, který bude na vstupu zohledňovat nový atribut, nebude možné použít zastaralé záznamy ukládané do databáze ještě před přidáním atributu, protože je u nich daná vlastnost neznámá. Z toho důvodu je v rámci záznamu ukládána i verze dat. Pokud dojde ke změně, bude aktuální verze zvýšena a všechny záznamy, které nemají verzi rovnou té aktuální, jsou pak zastaralé. Jak bylo uvedeno při popisu filtrace projektů, následně je možné zastaralé projekty zpracovat znovu, a tak získat opět aktuální data. Opakované zpracování pak může buď vytvořit nový záznam a starý nechat v databázi, aby byly zachovány vazby na klasifikátory, které byly na starších záznamech trénovány, a nedošlo tak ke ztrátě potencionálně potřebných dat při analýzách přesnosti modelů, nebo může být starý záznam smazán, což snižuje objem ukládaných dat, ovšem odpadá možnost zmíněné zpětné analýzy. Mezi oběma postupy je možné přepínat.

6.4 Vrstva strojového učení

Druhou hlavní součástí aplikace je modul, který zajišťuje práci s neuronovou sítí. Má tedy na starost trénování sítě a následné použití natrénovaného modelu, tedy klasifikaci neznámých vzorků. Jedním vzorkem pro trénování je jedno připojení, tedy napojení jednoho prutu na nosný, extrahované z některého projektového souboru. Vstupní příznaky každého vzorku představují vlastnosti daného připojení říkající, jak spojení prutů vypadá, a výstupní příznaky pak vyjadřují vhodnost jednotlivých spojovacích operacím pro spojení daných prutů. Podobná koncepce jako při poskytování a filtrování projektových souborů určených ke zpracování byla použita i při získávání připojení pro trénování modelu. Modul se tedy skládá z částí poskytující přípoje, filtrující přípoje, poskytující vzorky, konfigurující normalizační model a provádějící klasifikaci.

6.4.1 Poskytování připojení

Stejně jako v případě poskytování projektů implementují všechny třídy určené k poskytování připojení společné rozhraní `IJointsProvider` obsahující metodu `GetJoints` vracející enumeraci entit `Joint`, rozhraní bylo implementováno v rámci tříd `BasicJointsProvider` a `JointsFromConnectionProvider`. První třída umožňuje načítat entity z databáze, což je využíváno při trénování neuronové sítě na uložených připojeních a druhá pak získává připojení ze styčníku při klasifikaci jednotlivých připojení zadaného nespojeného styčníku. Objekt poskytovatele získá instanci filtru entit `Joint`, díky čemuž je možné poskytované entity omezit jen na ty s určitými vlastnostmi.

[6] Kapitola 5.2 Vybrané vlastnosti

6.4.2 Filtrování připojení

Filtr připojení implementuje rozhraní `ISelector<Joint>` a jednotlivé implementace je opět možné kombinovat pomocí `AndSelector`. Nejdůležitější třídou implementující zmíněné rozhraní je `JointsSuitableForLearningSelector`, která ze zadaných připojení vybírá pouze ty vhodné k trénování, tedy ty, jejichž styčník i projekt, ze kterého byly extrahovány, byly dle atributu `Status` v pořádku načteny ze souboru. Další vlastnost, kterou filtr kontroluje je, zda jsou hodnoty spočítaných využití materiálu ve všech směrech zatížení v rozumných mezích, což svědčí o tom, že statik při návrhu volil rozumná zatížení, že je připojení pravděpodobně správně navrženo, a že je vhodné na něm trénovat model. Druhá třída implementující toto rozhraní je `JointsNotUsedSelector`, který vybírá pouze připojení, která nebyla použita ani pro trénink, ani pro validaci při učení zadaného modelu. Tento filtr je použit při pokračování tréninku zadaného modelu, kdy je model trénován jen na připojeních, na kterých zatím trénován nebyl. Poslední implementace filtru je `JointsUsedSelector`, který vybírá připojení, na kterých byl při trénování zadaný model validován nebo trénován, podle zadaného parametru. Filtr je používán při zpětné analýze přesnosti modelu, kdy jsou připojení použitá při validaci klasifikována a výsledek je porovnáván se správnou klasifikací.

6.4.3 Poskytování vzorků

Další součást modulu je určena k poskytování vzorků pro trénování nebo klasifikaci. Třídy poskytující vzorky implementují `ISamplesProvider`. Toto rozhraní obsahuje metodu pro získání enumerace vzorků `GetSamples`. Jeden vzorek, objekt `Sample`, obsahuje atribut `SourceJointId` odkazující na připojení, ze kterého byl vzorek vytvořen a dále kolekce vstupních příznaků a výstupních příznaků. Vstupní příznak představuje objekt `InputFeature`, výstupní pak `OutputFeature`. Oba objekty obsahují atributy název příznaku a hodnotu příznaku, vstupní příznaky navíc obsahují ještě atribut `Type`, tedy informaci o tom, zda je daný příznak určen k normalizaci na základě střední hodnoty a směrodatné odchylky daného příznaku mezi vzorky, nebo se jedná o fuzzy příznak, který tímto způsobem normalizovaný není. Objekty pro poskytování vzorků mají k dispozici objekt implementující rozhraní `IJointsProvider`, jakožto svůj zdroj entit `Joint` pro převod na vzorky. Rozhraní `ISamplesProvider` implementuje třída `BasicSamplesProvider`, která pouze převádí připojení na vzorky. Vlastnosti připojení představují jednotlivé příznaky, pokud nejsou číselné, jsou na číselné převáděny.[7] Příznaky jsou pojmenovány podle toho, co daná vlastnost vyjadřuje, v případě výčtových typů, kde je hodnota kódována do vektoru příznaků, jsou příznaky pojmenovávány jako název výčtového typu lomeno název dané hodnoty výčtového typu. Pojmenování vstupních a výstupních příznaků neuronové sítě je vhodné k tomu, aby bylo možné jednoduše interpretovat, co vstupní a výstupní příznaky sítě znamenají, když je natrénovaná síť uložena do databáze, a také pro vazbu na normalizační koeficienty, které jsou každému příznaku počítány. Další implementace, `NormalizedSamplesSprovider`, navíc extrahované příznaky normalizuje na základě střední hodnoty a směrodatné odchylky. Objektu je předán normalizační model, který tyto hodnoty pro každý příznak poskytuje právě na základě názvu daného příznaku.[8]

[7] Kapitola 7.4.1 Extrakce
[8] Kapitola 7.4.2 Normalizace

6.4.4 Konfigurace normalizačního modelu

Aby bylo možné provádět normalizaci příznaků, je třeba analyzovat hodnoty daného příznaku u skupiny vzorků a zjistit střední hodnotu a směrodatnou odchylku. K tomuto účelu slouží třídy implementující rozhraní `INormalizationModelGenarator`, poskytující metodu pro získání normalizačního modelu. Rozhraní zatím implementuje jediná třída, ale je možné, že bude přidána jiná implementace. Vstupem pro získání normalizačního modelu je enumerace vzorků, objektů `Sample`, algoritmus prochází kolekci vstupních příznaků každého vzorku a u příznaků, které jsou označeny, jako určené k normalizaci atributem `Type`, počítá střední hodnotu a směrodatnou odchylku. Výstupem výpočtu je pak entita `NormalizationModel`, obsahující kolekci záznamů o provedené analýze. Každý záznam se týká jednoho příznaku a obsahuje tedy střední hodnotu a směrodatnou odchylku daného příznaku. Záznam je identifikován jménem daného příznaku, díky čemuž je později při normalizaci vzorku možné pro daný příznak najít zmíněné dvě hodnoty pro provedení normalizace. Normalizační model je uložen do databáze, protože později při použití neuronové sítě je nutné vzorky normalizovat opět stejným způsobem.

6.4.5 Trénování modelu a klasifikace

Pro práci s neuronovou sítí slouží speciální třída, kterou opět zbytek aplikace využívá bez nutnosti znát detaily konkrétní implementace. Třída je založena knihovně Keras jazyka Python. Propojení aplikace s touto knihovnou a spouštění programu v jazyce Python zajišťuje knihovna Keras.Net, vytvářející spojení mezi programem v jazyce C# a Python. Knihovna Keras stojí mimo jiné na knihovně TensorFlow, která slouží k trénování modelů pomocí strojového učení, a na NumPy zajišťující matematické výpočty.

Třída poskytuje metody `Learn` a `Evaluate` sloužící k natrénování modelu a ke klasifikaci neznámého vzorku. V obou případech získá třída objekt sloužící k poskytování vzorků, implementující rozhraní `ISamplesProvider`, který představuje pro třídu zdroj vzorků, na kterých trénuje nový model, nebo které klasifikuje. Dále v obou případech získá entitu `LearningModel`, v prvním případě tato entita představuje model, ze kterého trénování vychází, tedy stav modelu, na kterém trénování začíná, v druhém případě entita představuje model, který má být ke klasifikaci použit. Při trénování modelu nemusí být výchozí model poskytnut, v takovém případě se jedná o trénování zcela nového modelu. Výstupem metod je v prvním případě entita `LearningModel`, představující nové natrénovanou neuronovou síť, v druhém případě pak enumerace vzorků s doplněnou kolekcí `OutputFeatures`, která reprezentuje hodnoty výstupních příznaků sítě pro jednotlivé vzorky.

Operace `Learn` nejprve připraví data, která neuronová síť vyžaduje na vstupu, tedy pole dvojic, kde prvním prvek je vektor vstupních příznaků, a druhý prvek vektor výstupních příznaků. V případě pokračování v trénování zadaného modelu, je třeba případně změnit pořadí příznaků vzorků, aby odpovídalo seznamu vstupních příznaků zadaného modelu, protože pořadí významů příznaků musí být zachováno, jinak by pokračování nedávalo smysl. Takto připravená data jsou rozdělena do dvou částí pro trénování a pro validaci. Zároveň je o tomto rozdělení proveden záznam do entity `LearningModel`, aby byla tato informace později dostupná, a to v rámci kolekce objektů `JointUsedWhileLearning`.[9] Dále je inicializován klasifikátor. V případě navazujícího učení je neuronová síť uvedena do stavu, ve kterém skončilo předchozí učení, díky tomu, že entita `LearningModel` obsahuje informaci o jednotlivých vrstvách modelu a natrénovaných vahách. V případě nového modelu je pro-

[9] Kapitola 6.1.1 Schéma databáze

vedena implicitní inicializace. Nakonec je spuštěno trénování modelu a výsledek trénování uložen do entity `LerningModel`, jedná se především o finální stav sítě, dále také o dobu učení a přesnost modelu na trénovací a validační sadě dat. Entita je vrácena jako výsledek operace `Learn`.

Operace Evaluate také nejprve připraví data ke klasifikaci, tedy pole vektorů vstupních příznaků. Je opět nutné příznaky přeskládat do správného pořadí, aby významy příznaků byly stejné jako při trénování modelu, a to pomocí seznamu jmen vstupních příznaků, který obsahuje entita `LearningModel`. Také je inicializována neuronová síť načtením natrénovaných vah. Nakonec jsou vstupní data klasifikována sítí. Výstupem sítě je pole vektorů výstupních příznaků. V každém vstupním vzorku, objektu `Sample`, je vytvořen nový seznam výstupních příznaků, objektů `OutputFeature`, se jménem příznaku a hodnotou ze získaného vektoru. Hodnotám výstupního vektoru je tedy přiřazen význam díky seznamu jmen výstupních příznaků z entity `LearningModel`. Seznam vzorků již s doplněnými výstupními příznaky je vrácen jako výsledek operace `Evaluate`.

6.5 Fasáda aplikace

Aby byla aplikace snadno použitelná byla vytvořena její fasáda, která umožňuje spustit některé často používané operace bez znalosti implementačních detailů. Každá operace představuje jeden případ užití aplikace. Jedná se o následující, přičemž další mohou být přidávány podle vznikajících potřeb.

Operace `Learn` umožňující natrénovat nový model, případně operace `ContinueLearning` umožňující vytvořit model vycházející z jiného, identifikaci výchozího modelu v databázi je pak předána parametrem. Obě operace vracejí identifikaci vzniklého modelu v databázi. Nejprve je vytvořen objekt filtru připojení, v obou případech filtr vybírající připojení vhodná k trénování, při pokračování tréninku je filtr kombinován ještě s filtrem vybírajícím pouze připojení, na kterých výchozí model zatím trénován nebyl.[10] Na základě filtru je vytvořen objekt poskytovatele připojení,[11] na jeho základě pak objekt poskytovatele normalizovaných vzorků.[12] Normalizační model je získán v prvním případě analýzou vstupních vzorků, v případě návaznosti na exitující model je použit již dříve vytvořený. Nakonec je spuštěno trénování a vzniklý model je uložen do databáze.

Operace `EvaluateConnections` slouží k určení vhodných spojovacích operací připojení ve styčnících zadaného projektu. Operace získá jako parametr cestu k projektu, který má být zpracován, stejně tak i identifikaci entity `LearningModel` v databázi. Je vytvořen objekt poskytovatele projektů z lokálního souborového systému,[13] modul pro těžení dat zpracuje vstupní projekt,[14] z výsledku jsou získány entity `Connection`, představující styčníky, na jejich základě jsou vytvořeny objekty poskytovatelů připojení[15] ze styčníků a následně poskytovatelé normalizovaných vzorků.[16] Nakonec proběhne klasifikace a výstup v podobě operace s nejvyšší pravděpodobností je vypsán na výstup.

[10] Kapitola 6.4.2 Filtrování připojení
[11] Kapitola 6.4.1 Poskytování připojení
[12] Kapitola 6.4.3 Poskytování vzorků
[13] Kapitola 6.3.1 Poskytování projektů
[14] Kapitola 6.3.3 Těžení dat
[15] Kapitola 6.4.1 Poskytování připojení
[16] Kapitola 6.4.3 Poskytování vzorků

Operace `ValidateModel` umožnuje analyzovat správnost zadaného klasifikátoru. Na základě filtru, který vybírá připojení použitá při validaci zadaného modelu,[17] je vytvořen poskytovatel připojení[18] a poskytovatel normalizovaných vzorků.[19] Vzorky jsou následně klasifikovány a vypsán výsledek, tedy počet správně a špatně klasifikovaných vzorků a seznam nesprávně klasifikovaných vzorků včetně určené a správné třídy.

Operace `MineUnseenProjects` umožnuje provést těžení nových projektů, které ještě nejsou v databázi. Je vytvořen filtr projektů, které zatím nejsou v databázi.[20] Na základě filtru je vytvořen poskytoval projektů[21] a spuštěno těžení dat.[22] Výstupem jsou entity `Project`, které jsou uloženy do databáze. Operace `MineProjectsWithVersionLowerThan` slouží k aktualizaci starších záznamů v databázi tím, že znovu zpracovává projekty s verzí nižší než zadaná hodnota. V tomto případě je pouze filtr projektů nahrazen filtrem, který vybírá projekty s nižší verzí než zadaná hodnota.

[17] Kapitola 6.4.2 Filtrování připojení
[18] Kapitola 6.4.1 Poskytování připojení
[19] Kapitola 6.4.3 Poskytování vzorků
[20] Kapitola 6.3.2 Filtrování projektů
[21] Kapitola 6.3.1 Poskytování projektů
[22] Kapitola 6.3.3 Těžení dat

Kapitola 7

Strojové učení

7.1 Výběr klasifikačního algoritmu

Určení vhodného typu připojení dvou prutů ve styčníku je pro zkušeného statika relativně jednoduché a dá se přirovnat například k rozpoznání objektů v obraze. Jedná se tedy o snadno řešitelné úlohy pro člověka, ovšem nevhodné pro řešení počítačem pomocí klasických technik programování, jelikož ruční definování rozhodovacích pravidel a postihnutí všech možných případů v programu je velice složité. Proto jsem se rozhodl natrénovat klasifikátor pomocí strojového učení, které je založené na hledání zmíněných pravidel analýzou předložených dat. Vzhledem tomu, že mám k dispozici sadu vzorových připojení včetně jejich zařazení do tříd, je možné použít algoritmus strojového učení s učitelem, který existenci označkované sady vzorků předpokládá. Dalším požadavkem na algoritmus je vhodnost pro vícetřídní klasifikaci. Po konzultaci s vedoucím práce byla zvolena neuronová síť.

7.2 Neuronová síť

7.2.1 Princip

Neuronová síť (myšleno umělá neuronová síť) je založena na principu fungování přírodní neuronové sítě, tedy mozku. V reálném mozku mezi sebou komunikují mozkové buňky zvané neurony, a to pomocí šíření vzruchů. Analogicky v umělých neuronových sítích mezi sebou komunikují neurony, základní prvky neuronových sítí, a to pomocí vzájemného propojení jejich výstupů a vstupů. V těchto neuronových sítích neurony označují funkci o několika vstupech a jednom výstupu. Podle jakých pravidel jsou výstupy napojeny na vstupy jiných neuronů, je určeno architekturou neuronové sítě.[1]

Jakým způsobem se chová jeden neuron, definuje předpis:

$$y = A(w \cdot x - b),$$

kde x je vektor vstupů, w vektor vah, b konstanta nazývaná bias (předpětí) a $A()$ je aktivační funkce. Vektorový součin $w \cdot x$ lze také interpretovat jako váženou sumu vstupů. Vektor w tedy určuje, jak moc jsou jednotlivé vstupy pro daný neuron důležité a jak má na jejich hodnoty reagovat. Pokud má být neuron citlivý na jeden konkrétní vstup, příslušná váha bude vysoká. Váženou sumou získáme celkovou úroveň vybuzení neuronu. Předpětí b určuje, jak vysoké musí být vybuzení, aby byl neuron aktivován, tedy aby na jeho výstupu

[1] Kapitola 7.2.2 Architektura

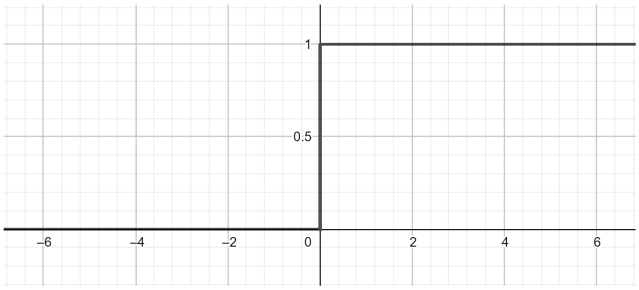
byla vysoká hodnota. Pokud vybuzení přesáhne hodnotu předpětí, výsledná hodnota výrazu $w \cdot x - b$ bude kladná a výstup neuronu bude na vysoké úrovni, podle zvolené aktivační funkce. Předpětí je někdy bráno jako další z vah imaginárního vstupu neuronu s konstantní vstupní hodnotou –1. Určení vah jednotlivých vstupů včetně určení předpětí ovlivňuje chování jednotlivých neuronů, a tedy celé neuronové sítě, a je předmětem jejího tréninku.[2] Pokud je neuron dostatečně vybuzen a dojde k jeho aktivaci, na tuto skutečnost mohou reagovat další neurony, jejichž vstup je napojen na aktivní výstup, a informace se tak šíří neuronovou sítí dál. Volba aktivační funkce $A()$ je rovněž předmětem návrhu neuronové sítě.

7.2.2 Architektura

Jak neuronová síť pracuje a k řešení jakých úloh je vhodná, velmi závisí na její architektuře, která zahrnuje topologii sítě, tedy způsob propojení neuronů mezi sebou, a také zvolenou aktivační funkci.

Podle topologie lze sítě rozdělit na mnoho typů. Základními druhy neuronových sítí jsou dopředné a rekurentní. První zmíněné se skládají z neuronů uspořádaných do vrstev s tím, že výstupy neuronů jedné vrstvy jsou napojeny na vstupy neuronů následující vrstvy. První vrstva přijímá vstupní data, poslední vrstva naopak generuje výstupní. Případné další vrstvy mezi vstupní a výstupní se nazývají skryté. Nejpoužívanějším způsobem propojení vrstev je přivedení výstupu neuronu na jeden ze vstupů každého neuronů následující vrstvy, vstupem neuronu je tak vždy vektor výstupů všech neuronů předchozí vrstvy. Dopřednou neuronovou síť jsem použil také v mé práci, jelikož se jedná o relativně jednoduchou klasifikační úlohu, na kterou by měla vystačit základní neuronová síť. Dalším typem je rekurentní sítě neboli sítě se zpětnou vazbou, která navíc umožňuje napojení výstupu neuronu na svůj vlastní vstup nebo vstup jemu předcházejícímu neuronu. Tyto sítě tak dokáží detekovat i souvislosti mezi jednotlivými po sobě jdoucími vzorky, což je vhodné například u zpracovávání zvukového signálu.

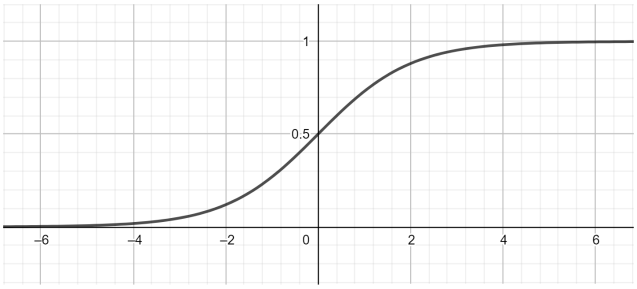
Další možnost, jak ovlivnit podobu neuronové sítě je volba aktivační neboli přenosové funkce jednotlivých neuronů. Nejjednodušší aktivační funkcí je skoková, která zajistí, že jakmile vybuzení dosáhne hodnoty prahu, výstup neuronu se skokově dostane z hodnoty 0 na hodnotu 1. Průběh je vidět na obrázku 7.1.



Obrázek 7.1: Průběh skokové aktivační funkce.

Další funkce, která je velmi často využívaná se jmenuje logistická sigmoida. Její hodnota monotónně roste v intervalu 0 až 1 a její výsledek je tak vždy nenulový. Hodnota reflektuje na rozdíl od skokové funkce sílu vybuzení neuronu. Průběh ukazuje obrázek 7.2.

[2] Kapitola 7.2.3 Trénování



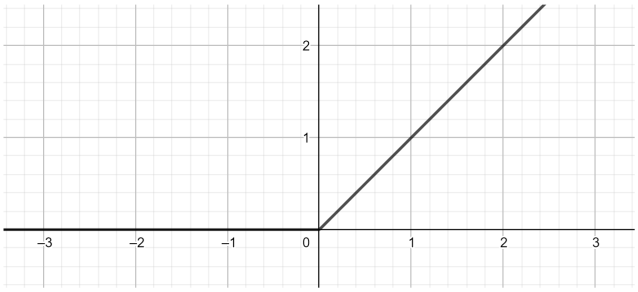
Obrázek 7.2: Průběh logistické sigmoidy.

Předpis logistické sigmoidy je následující:

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

Výpočet funkční hodnoty je kvůli $-x$ v exponentu poměrně drahá operace, a tak použití této přenosové funkce může přispět k pomalejšímu trénování sítě i klasifikaci. Další nevýhodou je, že čím je neuron saturovanější, tedy jeho vybuzení je vysoké nebo naopak nízké, tím je hodnota x dále od 0 a hodnota gradientu klesá. Vzhledem k tomu, že pro hodnotu $x = 0$, kde je gradient nejvyšší, dosahuje jeho hodnota 0,25, je vždy menší než 1. Pokud má síť několik skrytých vrstev, hodnota gradientu při opakovaném násobení při zpětné propagaci mezi vrstvami ještě více klesá, protože hodnota gradientu, jak bylo uvedeno, je vždy menší než 1. Úpravy vah jsou kvůli nízkému gradientu malé a neefektivní. Učení tak trvá dlouhou dobu, nebo dokonce ani nedosáhne stejně efektivního výsledku jako sítě s jinými aktivačními funkcemi. Místo logistické sigmoidy se doporučuje požívat například přenosovou funkci ReLU.

Fukce ReLU pro své výhody často nahrazuje klasickou aktivační funkci logistickou sigmoidu. Problém klesajícího gradientu se zde vyřešen a navíc je výpočetně jednoduchá, což je pro rozumně dlouhé trénování sítí kritické a umožňuje tak stavět větší sítě s velkým počtem neuronů a skrytých vrstev. Průběh je vidět na obrázku 7.3.



Obrázek 7.3: Průběh funkce ReLU.

Předpis funkce ReLU je následovný:

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

V poslední vrstvě neuronových sítí se používá aktivační funkce softmax, která zajišťuje, že součet výsledků pro jednotlivé třídy je roven 1 a je tedy možné jednoduše jednotlivé hodnoty interpretovat jako pravděpodobnosti jednotlivých tříd. Funkce softmax je definována:

$$f(x_i) = \frac{e^{x_i}}{\sum_{t=1}^{T} e^{x_t}},$$

kde T je počet tříd a x_n je vybuzení n-tého výstupního neuronu.

Na rozdíl od parametrů sítě, tedy vah a prahů, jejichž hodnoty jsou získávány v průběhu trénování, architektu sítě je nutné určit před začátkem tréninku. Parametry definující architekturu sítě se označují jako hyperparametry. Jedná se o vlastnosti související se zmíněnou topologií, tedy počet vrstev a počet neuronů v jednotlivých vrstvách. Další nastavitelný hyperparametr je způsob propojení neuronů mezi sebou či volba vhodné aktivační funkce. Hyperparametry pak mohou být v rámci vývoje aplikace měněny pro nalezení optimální kombinace.

7.2.3 Trénování

Trénování sítě spočívá v hledání optimálních vah a předpětí, při kterých dokáže síť nejlépe klasifikovat vzorky. Míru správnosti klasifikace vyjadřuje chybová funkce, která je tím menší, čím se výsledky sítě více blíží očekávaným. Existují různé funkce, které se za tímto účelem používají, jedna z často využívaných je křížová entropie, definovaná následovně:

$$f(v) = -\sum_{t=1}^{T} y_{v,t} ln(p_{v,t}),$$

kde T je počet tříd klasifikátoru, $y_{v,t}$ nabývá hodnoty 1, pokud vzorek v patří do třídy t jinak 0 a $p_{v,t}$ je pravděpodobnost, že vzorek v patří do třídy t podle predikce klasifikátoru.

Cílem tréninku sítě je tedy minimalizace chybové funkce postupným upravováním jejích parametrů. Optimalizační technika, která je často používána obecně při minimalizaci nejen chybové funkce neuronových sítí je gradientní sestup. V každém cyklu učení je hodnota parametrů upravena ve směru

$$p_{n+1} = p_n - \epsilon \nabla f(p_n),$$

kde p_k reprezentuje vektor parametrů sítě v kroku k, ϵ představuje trénovací konstantu, která definuje, jak velkými kroky se odhadované hodnoty parametrů upravují a $\nabla f()$ je gradient chybové funkce. Vzorec říká, že novou hodnotu parametrů získáme posunem v prostoru parametrů směrem daným zápornou hodnotou gradientu, čili směrem nejstrmějšího klesání. Opakovanými úpravami parametrů je hodnota chybová funkce snižována až se dostane do lokálního minima. Prvotní zvolení parametrů sítě, tedy pozice v prostoru parametrů na začátku, určí, do kterého minima se optimalizační algoritmus dostane. Minimum nemusí být absolutní, a tak je vhodné trénování opakovat několikrát s náhodným volením počátečních hodnot parametrů.

Efektivní určení gradientu chybové funkce pro aktuální hodnoty parametrů umožňuje algoritmus zpětné propagace chyby. Gradient se skládá z parciálních derivací chybové funkce podle jednotlivých parametrů. Jednotlivé derivace říkají, jak se změní chybová funkce při změně daného parametru, neboli jak je chybová funkce citlivá na změnu parametru. Parciální derivaci lze získat aplikací řetězového pravidla, které říká, že derivace vnořených funkcí $f(g(x))$ je rovna součinu derivací $\frac{\delta f}{\delta g} \cdot \frac{\delta g}{\delta x}$. Vhledem k tomu, že neuronová síť je v podstatě

sekvence vnořených operací, je možné postupovat zpětně a určovat nejprve parciální derivace chybové funkce podle výstupů poslední vrstvy, derivaci výstupu neuronů podle jejich vybuzení, derivaci vybuzení neuronů podle jednotlivých vstupních vah neuronu a tak dále. Aplikací řetězového pravidla je tedy možné odvodit parciální derivace chybové funkce podle jednotlivých vah a prahů všech vrstev sítě. Výpočet gradientu a následná úprava parametrů v rámci gradientního sestupu mohou být prováděny po každém vzorku, většinou se však gradient průměruje přes více trénovacích vzorků a úprava parametrů se provede až poté. Jedním přístupem je před úpravou projít všechny trénovací vzorky. Druhou technikou je trénovací sadu rozdělit na malé skupiny a úpravu provádět častěji po každé skupině. Kroky jsou v tomto případě nepřesnější, protože gradient je počítán pouze na základě podmnožiny kompletních dat, ale úpravy jsou prováděny častěji a algoritmus tak rychleji konverguje k extrému. Kompletní průchod všemi trénovacími vzorky, ať už s jedinou úpravou parametrů nebo s více dílčími, se označuje jako jedna epocha trénování.

7.2.4 Bias a Variance

Chyba modelu se dá rozdělit na dvě složky.

Bias (předpojatost) označuje chybu, které se model dopouští na trénovací sadě. Příčinou této chyby může být příliš jednoduchý model, krátká doba učení nebo i příliš málo dat, které nestačí k lepšímu natrénování. Řešením může být úprava modelu a jeho způsobu trénování nebo úprava vstupních dat například přidáním dalších příznaků. Chyba může být i částečně neodstranitelná, když data potřebná ke správné klasifikaci nejsou k dispozici. Například pokud je na scéně příliš tma, tak model nedokáže správně detekovat objekty.

Variance je rozdíl chyby modelu na trénovací a validační sadě dat. Pokud je variance malá, znamená to, že model dokáže dobře generalizovat a správně klasifikovat i data, na kterých se netrénoval. Pokud je naopak vysoká, může to ukazovat na přetrénovanost modelu. Aby byl model schopný klasifikovat lépe neviděné validační vzorky, je nutné, aby lépe pochopil souvislosti v datech, a k tomu potřebuje více trénovacích vzorků. Proto se dá variance většinou dobře zredukovat přidáváním dalších trénovacích dat.

7.2.5 Přetrénování a nenatrénování

Čím déle probíhá trénování modelu, tím jsou obvykle jeho výsledky přesnější, může se ovšem stát, že se model na trénovacích datech natrénuje příliš dobře a najde v nich i souvislosti, které nejsou relevantní. Trénovací vzorky potom sice dokáže klasifikovat dokonale, ale už nedokáže generalizovat a klasifikovat i jiná než tato data. V takovém případě se model označuje jako přetrénovaný. Řešením je proces učení zastavit včas, případně i model zjednodušit, aby nebyl schopný dokonalého naučení trénovacích dat. Aby bylo možné poznat, kdy se model začíná přetrénovávat, je nutné data určená pro učení rozdělit na trénovací a validační sadu. Na trénovacích vzorcích se model učí a je pravidelně kontrolován pomocí klasifikování validační sady. Jakmile se začne zvyšovat variance, nebo dokonce začne klesat úspěšnost klasifikace validační sady, model se začíná příliš uzpůsobovat trénovacím datům a ztrácí schopnost generalizace. Použil jsem často využívaný poměr a používal 70 % dat pro trénování a 30 % pro validaci.

Druhým extrémem je nenatrénování, kdy místo toho, aby byl model až příliš dobře natrénovaný na zadaná data, je naopak natrénovaný příliš málo a nedosahuje vysoké přesnosti. V tomto případě může pomoci prodloužení trénování, ale problém může být i v příliš jednoduchém modelu, který jednoduše není schopen zadaná data klasifikovat. Příkladem

může být lineární klasifikátor, který nedokáže správně klasifikovat lineárně neoddělitelná data.

7.2.6 Metrika

Metrika označuje způsob vyhodnocení, jak dobře dokáže model klasifikovat vzorky. Čím je vyšší hodnota metriky, tím lepší model je. Je možné, že metrik pro daný model je více, v tomto případě je dobré nalézt předpis, pro výpočet souhrnné jedno-hodnotové metriky, jelikož je snadněji uchopitelná a dá se snáze porovnat s jinými výsledky, což je zásadní pokud je například potřeba zhodnotit, zda se model poslední změnou zlepšil či naopak. Jednou z možností nahrazení několika metrik jednou je rozdělení metriky na optimalizovanou, která je porovnávána a uspokojovanou, která musí dodržovat určený limit. Příkladem může být doba klasifikace a přesnost klasifikace. Model je buď pomalý a přesný, nebo naopak rychlý, ale dělá chyby. K rozhodnutí, který model je lepší je třeba metriky skombinovat do jediné porovnatelné hodnoty, ovšem tyto dvě metriky lze jen obtížně kombinovat. Řešením pak může být stanovení limitu času klasifikace, do kterého se model musí vejít a k porovnávání kvality modelu pak používat jen přesnost. V mém případě až tak nezáleží na době klasifikace, ale upravit metriku, aby zohledňovala odezvu aplikace, by bylo vhodné v rámci případného budoucího pokračování práce. Pokud má aplikace statikům poskytovat radu při návrhu, je nežádoucí, aby klasifikace trvala příliš dlouhou dobu. Pro tuto práci jsem jako metriku kvality modelu používal poměr správně klasifikovaných validačních vzorků k jejich celkovému počtu.

7.2.7 Chybová analýza

Metrika pouze určí hodnotu říkající, jak dobrý model je, ale neříká nic o tom, co udělat pro to, aby byl lepší. Aby bylo možné pochopit, v čem dělá model chyby a co je třeba vyřešit, aby se hodnota metriky zvyšovala, je třeba provádět chybovou analýzu. Tento proces spočívá ve studování chybně klasifikovaných vzorků za účelem odhalení příčiny nízké úspěšnosti modelu. U nalezených problémů je pak vhodné určit prioritu na základě počtu chyb způsobených daným problémem. Nemá smysl řešit problém, který způsobí chybu v klasifikaci pár procent vzorků, když existuje jiná chyba způsobující chybnou klasifikaci u podstatně větší množiny vzorků.

V rámci chybové analýzy jsem při postupném vylepšování neuronové sítě vytvářel matice záměn. Tabulka 7.1 ukazuje hodnoty jedné z prvních matic záměn, kterou jsem sestavil.

		určeno klasifikátorem					
		úhelník	žiletka	k. deska	svaření	kloub	p. k p.
skutečnost	úhelník	**32,20**	30,10	11,02	20,05	2,01	4,62
	žiletka	5,00	**61,20**	15,02	9,63	1,80	7,36
	koncová deska	12,69	13,15	**56,04**	11,48	4,36	2,29
	svaření	10,40	6,41	5,46	**67,82**	2,40	7,50
	kloub	4,45	16,08	18,87	19,00	**41,60**	0,00
	prut k prutu	5,15	8,85	4,50	13,40	0,00	**68,10**

Tabulka 7.1: Příklad hodnot sestavené matice záměn.

Názvy odpovídají vybraným typům spojení. Na každém řádku jsou uvedeny poměrné části sady dat z dané třídy, jak byly modelem klasifikovány. V ideálním případě by měly být hodnoty na diagonále rovny 100 % a ostatní 0 %. Na základě hodnot z tabulky je

pak možné přemýšlet nad tím, proč model dosahuje nízké úspěšnosti. Dalším postupem analýzy chyb, který jsem praktikoval, je ruční procházení chybně klasifikovaných vzorků. Na základě výpisu špatně klasifikovaných vzorků je možné najít daný přípoj, který stojí za daným vzorkem, v odpovídajícím projektu a podívat se, jak vypadá styčník, ze kterého pochází. Podoba styčníku totiž může napovědět, který příznak by mohlo být případně vhodné doplnit, aby v daném případě měl klasifikátor potřebnou informaci ke správné klasifikaci. Prvotní výběr příznaků totiž nemusí zahrnovat všechny potřebné informace ke správné klasifikaci, jelikož nutnost zařazení některých příznaků nemusí být zprvu zcela jednoznačná.

7.2.8 Počty vzorků ve třídách

Po analýze rozvrstvení vzorků mezi jednotlivými třídami se ukázalo, že některé třídy dominují a naopak vzorky z jiných tříd nejsou moc časté. V prvotních testech trénování sítě se mi dokonce stávalo, že se model naučil zařazovat vzorky jen do jedné třídy, která obsahovala nevíce trénovacích vzorků. Spojovací operace koncová deska či svaření jsou velice využívané, a tak počet vzorků těchto tříd je několikanásobný oproti počtu vzorků z nejméně zastoupené třídy úhelník. Nejedná se ovšem o vzorkovací chybu, protože tyto počty reflektují pravděpodobnosti výskytu jednotlivých operací v reálných projektech, jelikož trénovací projekty nejsou vybírány nějakým specifickým způsobem závislým na jejich obsahu a dá tedy předpokládat, že jsou dobrým reprezentativním vzorkem. Možným řešením by bylo přebytečné vzorky zahodit, aby jich byl ve všech třídách stejný počet, a následně apriorní pravděpodobnost daných tříd kompenzovat pomocí Bayesova teorému. V tomto případě by však došlo k zahození většiny trénovacích dat, což by mohlo vést v důsledku ještě k horším výsledkům, proto jsem toto řešení zatím nezkoušel, tento problém byl nakonec znatelně potlačen zvýšením objemu trénovacích dat, jak bylo nastíněno.

7.2.9 Zkvalitňování trénovacích dat

Jak bylo řečeno, jednou možností odstraňování předpojatosti je přidávání nových příznaků na základě chybové analýzy. Další postup, který jsem aplikoval, je filtrování vzorků, aby se model učil jen na relevantních datech. Pokud je totiž část trénovacích dat nesmyslná, model se naučí nelogické souvislosti, což následně vede k jeho menší úspěšnosti. Nejprve jsem předpokládal, že data v projektech se dají považovat za bezchybná, ale později se při chybových analýzách ukázalo, že nemusí nutně obsahovat kvalitně navržené styčníky. Zpracovávané projekty mohou být jen rozpracované, což samozřejmě znamená, že ne všechna připojení musí být vyhovující. Na takových datech pak není vhodné model trénovat. Další skupina projektů, která se ukázala být nevhodnou, jsou složité styčníky, které, což bylo dáno cílem práce, model nemusí umět klasifikovat. Jejich přítomnost v trénovacích datech je nežádoucí, protože v takových styčnících jsou používána různá kreativní nestandardní spojení, která model spíše pletou.

7.3 Výběr nástroje pro klasifikaci

Nástrojů, které umožňují jednoduše vyvíjet aplikace založené na strojovém učení, je celá řada. Ve své práci jsem chtěl použít knihovnu, která by byla postavená na programovacím jazyce Python, jelikož se jedná o mocný jazyk, se kterým je možné snadno a rychle psát netriviální konstrukce. Navíc jsem s tímto jazykem už dříve pracoval.

Knihovny určené ke strojovému učení pro Python jsou například Tensorflow, Theano, Ski-kit learn nebo PyTorch. Jelikož jsem s žádnou z nich neměl doposud žádnou zkušenost, rozhodoval jsem se mezi nimi se na základě statistik o využívanosti těchto knihoven programátory a také na základě žebříčků hodnotících výkonost daných nástrojů i přívětivost a dostupnost kvalitní dokumentace. Nakonec jsem si pro svou práci vybral nástroj Tensorflow, a to ve spojení s knihovnou Keras, která poskytuje přívětivé snadno použitelné rozhraní a je postavena na zmíněné knihovně.

Tensorflow je open source knihovna vyvíjená společností Google, která je určena k řešení různých problémů pomocí algoritmů strojového učení. Prováděné výpočty mohou být vykonávány na procesoru nebo i akcelerovány pomocí grafických kartách. Knihovnu je možné snadno zapojit do aplikací různých platforem, například i aplikací vyvíjených v C# na platformě .NET, kterou jsem si pro práci vybral.

7.4 Vstupní a výstupní příznaky

Vstupní příznaky představují vektor hodnot vkládaný na vstup neuronové sítě. Vstupní hodnoty jsou propagovány na výstup, což vede k vytvoření výstupního vektoru příznaků odpovídajícímu zadanému vstupu.

Vstupní a výstupní příznaky odpovídají vlastnostem spojených prutů získaným z projektových souborů, během jejich zpracovávání.[3]

Ukládané vlastnosti zahrnují mimo jiné i typ spojení prutů, tedy spojovací operaci, která byla použita. Tato vlastnost pochopitelně není k dispozici u nespojených prutů, jedná se totiž o vlastnost, která by měla být výstupem systému a která by měla být pro zatím nespojené pruty predikována. Ostatní vlastnosti jsou vstupní a na jejich základě by se měl systém rozhodovat.

7.4.1 Extrakce

Vlastností připojení mají většinou číselnou podobu, ale ne všechny, a protože vstupní i výstupní příznaky neuronové sítě musí být číselné hodnoty, je třeba jejich hodnoty převádět.

Vlastnosti, které vyjadřují logickou hodnotu, například, zda je prut průběžný, jsou jednoduše převoditelné tak, že nepravda odpovídá hodnotě 0 a pravda hodnotě 1. Podobně to lze provést i s vlastnostmi, které mají více možných hodnot, tedy s výčtovými typy. Například typ průřezu prutu je vlastností, která má několik přípustných hodnot. Takovou vlastnost je třeba vyjádřit v rámci několika příznaků, podle toho kolik hodnot je přípustných, pomocí kódování 1 z n. Každé přípustné hodnotě výčtového typu odpovídá jeden příznak a má hodnotu buď 1, pokud vlastnost nabývá právě této hodnoty, nebo 0, pokud nabývá jiné. Vzhledem k tomu, že vlastnost musí vždy nabývat právě jedné z možností, v této skupině příznaků bude vždy právě jeden s hodnou 1. Toto kódování je použito i při převodu použité operace, což je také vlastnost o několika přípustných hodnotách, na vektor výstupních příznaků.

S drobnou úpravou se toto kódování dá použít v případě, že vlastnost může naráz nabývat i několika hodnot ze zadaného výčtu, a to jednoduše tak, že ve skupině příznaků jich může být více s hodnotou 1. Tento postup byl využit při kódování topologie styčníku, do omezeného počtu příznaků, kdy každý směr prutu odpovídal jednomu příznaku, a podle přítomnosti či absence prutu v daném směru, nabývaly příznaky příslušných směrů hodnot 1, nebo 0.

[3] Kapitola 5.2 Vybrané vlastnosti

Další problémovou vlastností je zatížení prutu. Zatížení se skládá z obecného množství zátěžových kombinací. Každá kombinace představuje 6 hodnot sil a momentů sil, které definují jednu možnou kombinací zatížení s tím, že každý prut může mít více těchto kombinací pro různé případy jako boční vítr, přední vítr, sníh atd.[4] Tento obecný počet kombinací je však potřeba zakódovat do konstantního počtu příznaků. První možnost by mohla být omezit obecný počet kombinací například na 5, vzhledem k tomu, že málokterý prut má zadáno tolik zátěžových kombinací. Jedna kombinace představuje 6 příznaků a všechny kombinace by pak představovaly 30 příznaků. Pokud by prut měl zadáno méně než 5 kombinací, ostatní příznaky by byly vyplněny hodnotami 0, v případě, že by jich měl více, přebytečné kombinace by byly zahozeny. Pořadí jednotlivých kombinací však nemá žádný význam, jedná se jen o kombinace zatížení, které mohou v reálných podmínkách na prut působit. Problém tohoto přístupu je tím pádem, že šestice příznaků odpovídající jednotlivým kombinacím jsou v celkových 30 příznacích v nějakém pořadí, přitom na pořadí těchto šestic nezáleží. Pokud by byly šestice jen přeskládány, síť by tento vstupní vektor vnímala jako jiný. Řešením by mohlo být systém trénovat na všech kombinacích pořadí těchto šestic, to by ovšem značně zvýšilo objem trénovacích dat a velmi zpomalilo učení. Jiné řešení, jak kombinace zakódovat na pevném počtu příznaků, je z každého z 6 typů zatížení vybrat maximální hodnotu a tu brát jako hodnotu příznaku, což by vedlo na konstantní počet 6 příznaků. V tomto případě však dochází k určité ztrátě informace, pokud totiž jedna kombinace zatížení obsahuje tahovou sílu o určité hodnotě a jiná kombinace tahovou sílu menší, druhá hodnota se ve výsledném vektoru příznaků neprojeví. Tato ztráta informace by ale nemusela vadit, jelikož pokud má prut vydržet tahovou sílu z první kombinace, měl by vydržet i jakoukoli jinou menší sílu. Takto se dá zdůvodnit, že uvedený postup by měl fungovat. Vzhledem k tomu, že znaménko síly či momentu síly znamená směr na jednu či druhou stranu, je nutné ještě zdvojit všechny příznaky, tak že každý typ zatížení je převeden na dva příznaky, nejmenší zápornou hodnotu, neboli největší zatížení v jednom směru, a největší kladnou hodnotu, neboli největší zatížení v opačném směru. Z toho tedy vyplývá celkem 12 příznaků. Tento přístup byl nakonec použit.

7.4.2 Normalizace

Po extrakci nabývají příznaky obecných hodnot, na vstup sítě je však vhodné, aby byly vkládány hodnoty v rozmezí 0 až 1 a aby hodnoty daného příznaků co nejlépe využily tento rozsah a byly v tomto intervalu rovnoměrně rozptýleny. Příznaky vyjadřující logickou hodnotu a skupiny příznaků reprezentující hodnotu výčtového typu už nabývají hodnot 0 či 1 a není tak potřeba s těmito příznaky nic dělat. Ostatní příznaky je potřeba normalizovat.

Normalizace spočívá nejprve v odečtení posunu dat a následné deformaci dat tak, aby byla směrodatná odchylka rovna 1. Toto lze provést pomocí následujícího vzorce:

$$x_n = \frac{x - \bar{x}}{s},$$

kde x_n představuje normalizovanou hodnotu příznaku x, $\bar{x}$ je střední hodnota daného příznaku a s je směrodatná odchylka hodnot příznaku.

Střední hodnotu a směrodatnou odchylku daného příznaku je možné spočítat na základě vzorku hodnot, kterých příznak nabývá, a to podle následujících vzorců. První z nich vyjadřuje výpočet střední hodnoty, jako aritmetický průměr hodnot příznaku.

[4] Kapitola 2 Pojmy

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

kde N je počet hodnot vzorku a x_i hodnota na pozici i.

Na základě druhého vzorce je možné spočítat odhad směrodatné odchylky.

$$s = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \bar{x})}{N - 1}},$$

kde N je počet hodnot vzorku, x_i hodnota na pozici i a $\bar{x}$ střední hodnota daného příznaku. Dělení hodnotou $N - 1$ místo počtem hodnot N vyplývá z použití omezeného vzorku hodnot pro výpočet směrodatné odchylky, jedná se tedy o výběrovou směrodatnou odchylku.

Tímto způsobem je zajištěna střední hodnota transformovaných dat 0 a směrodatná odchylka 1. Následnou aplikací logistické sigmoidy na normalizovaná data získáme hodnoty z intervalu 0 až 1. Funkce logistická sigmoida se blíží v nekonečnu k hodnotě 1 a v minus nekonečnu k hodnotě 0, mezi těmito hodnotami monotónně roste, stejná funkce je používána jako aktivační funkce neuronů.[5]

7.5 Vylepšování výsledků neuronové sítě

7.5.1 První testování

První testování funkčnosti aplikace jsem provedl poté, co jsem zpracoval prvních 2 000 projektových souborů a měl tak k dispozici bezmála 8 000 připojení, tedy trénovacích vzorků. Jako vstupní příznaky jsem zvolil zatím jen základní vlastnosti připojení, tedy informace o nosném a připojeném prutu. Jednalo se o příznaky vyjadřující výšku a šířku průřezu prutu, dále 3 příznaky vyjadřující prostorový vektor orientace prutu, 3 příznaky odpovídající excentricitám v jednotlivých směrech oproti připojení do bodu styčníku a příznak vyjadřující rotaci prutu okolo střednice. Tyto příznaky byly normalizovány. Další příznaky vyjadřovaly logickou hodnotu a byly to informace, zda je prut průběžný, zrcadlený podle osy x a zrcadlený podle osy z. Další skupina příznaků reprezentovala typ průřezu prutu a to pomocí kódování 1 z n. Vzhledem k velkému počtu různých typů průřezů se jednalo se o 108 příznaků. Těchto celkem 120 příznaků představovalo jeden prut, vzhledem k tomu, že připojení sestává z připojeného a nosného prutu, výsledný vstupní vektor příznaků obsahoval tuto skupinu dvakrát pro oba zmíněné. Dalších 12 příznaků reprezentovalo zatížení připojeného prutu. Tyto příznaky vyjadřovaly maximální kladné zatížení, zatížení v jednom směru, a maximální záporné zatížení, zatížení v opačném směru, a to celkem 6krát pro všech 6 typů zatížení.[6] Jedno připojení tedy reprezentovalo celkem 252 příznaků.

Výstupní příznaky reprezentovaly použitý typ spojovací operace. K převodu na skupinu příznaků bylo opět použito kódování 1 z n, což vedlo na celkem 6 výstupních příznaků, pro každou operaci jeden.

Mezi vstupní a výstupní vrstvu jsem se ze začátku rozhodl vložit jednu skrytou vrstvu, a to o 200 neuronech. Jako aktivační funkci neuronů jsem zkoušel nejprve logistickou sigmoidu, poté i funkci relu, nepozoroval jsem ale nějaký výrazný rozdíl ani v přesnosti netrénovaného modelu, ani v rychlosti trénování, asi vzhledem k tomu, že model i trénovací

[5] Kapitola 7.2.2 Architektura

[6] Kapitola 7.4.1 Extrakce

sada jsou relativně malé a výhody funkce relu by se pravděpodobně projevily při komplikovanější neuronové síti. V dalších modelech jsem však již používal funkci relu. Jako chybovou funkci jsem použil křížovou entropii a jako metriku úspěšnosti modelu jsem použil procento správně klasifikovaných vzorků z validační sady, která představovala třetinu všech dostupných vzorků. Velikost skupiny dat, po které dochází při gradientním sestupu k úpravě parametrů sítě, jsem nastavil na 500 vzorků.

Při testování jsem s touto konfigurací sítě a výše popsanou sadou dat došel k nejlepšímu výsledku 59 % správně klasifikovaných validačních dat. Na základě průběhu učení, jsem proces trénování zastavil v okamžiku, kdy úspěšnost na validační sadě přestala stoupat, abych zabránil přetrénování. V rámci trénovacích dat pak úspěšnost byla daleko vyšší, a to 88 %. Vysoká variance byla nejspíše způsobena příliš malým množstvím dat.

Na začátku jsem předpokládal, že data, která jsem měl k dispozici, jsou bezchybná. Ovšem už při prvních testech se ukázalo, že nemusí obsahovat pouze relevantní vzorky. Některé projekty byly pouze rozpracované a obsahovaly očividné chyby, jako je třeba absence zatížení nebo spojovací operace. Jiné třeba ani nebylo možné načíst, protože obsahovaly závažnější chyby. Tyto chyby bylo jednoduché zachytit, později jsem zjistit, že projekty mohou obsahovat i jiné chyby, které nebyly zprvu zřejmé. Když jsem ručně otevíral některé projektové soubory, abych kontroloval jejich obsah, zjistil jsem, že v komplikovanějších styčnících je běžné, že prut je připojen k jinému prutu, který je teprve připojen k nosnému. Já jsem ale spoléhal na to, že všechny připojované pruty ve styčníku jsou připojeny k nosnému prvku, tuto skutečnost jsem neověřoval a jako prut, ke kterému je druhý připojován, jsem automaticky bral nosný. Extrahovaný trénovací vzorek tedy nemusí dávat smysl, protože jeho pruty ve skutečnosti nemusely být propojeny. To, že prut není připojený přímo k nosnému, ale je k němu připojený přes jiný prut není chyba návrhu styčníku, ale takové připojení je už komplikovanější a v rámci mé práce jsem se rozhodl řešit pouze základní jednoduchá připojení vždy přímo k nosnému prutu. Musel jsem tedy data pro trénování lépe vybírat, aby se model učil jen na smysluplných vzorcích, a kromě jiných chyb navíc ověřovat i přímé propojení obou prutů.

7.5.2 Rozšíření datové sady

Vzhledem k prvním výsledkům, jsem naznal, že bude pravděpodobně nutné použít k trénování větší množství dat. Proto jsem zpracoval další projektové soubory a nakonec měl k dispozici 31 000 trénovacích vzorků. V souvislosti s navýšením objemu trénovací sady jsem se také rozhodl přidat další skrytou vrstvu taktéž s 200 neurony. Kromě přidání vrstvy, jsem model nechal tak, jak byl. Úspěšnost sítě vzrostla na 74 % správně klasifikovaných validačních vzorků, v rámci trénovací sady byla úspěšnost 90 %.

7.5.3 Lepší interpretace zatížení

K dalšímu zvýšení úspěšnosti by jistě pomohlo opětovné rozšíření datové sady, v tu chvíli jsem však zpracoval všech 8 000 projektových souborů, které jsem měl k dispozici. Abych tedy posunul úspěšnost dále, snažil jsem se přemýšlet znovu nad vstupními příznaky a upravit je či přidat nové tak, aby lépe vystihly podobu připojení a neuronová síť pak dosahovala lepších výsledků.

První variantou, která se nabízela, bylo zaměnit síly a momenty sil za napětí v materiálu. Po zpracování projektových souborů jsem měl totiž k dispozici kromě velikostí sil a momentů sil v jednotlivých směrech zatížení, také dopočítaná napětí, která dané síly a momenty v materiálu v jednotlivých směrech vyvolávají. Zpočátku jsem se rozhodl použít

síly a momenty sil, vhledem k tomu, že jsem byl přesvědčen, že dopočítaná napětí nemohou modelu předat žádnou lepší informaci, jelikož jsou pouze z těchto hodnot spočítány. Ovšem později jsem si uvědomil, že při tomto výpočtu se používají i konstanty specifické pro jednotlivé typy průřezů, z toho vyplývá, že identická síla vyvolá různé napětí v tuhém profilu, který je stavěný sílu v tomto směru vydržet, než v jiném, který není vůči této síle tolik odolný, jelikož je používán zcela k jinému účelu. Proto je potřeba při posouzení, zda je daná síla velká či malá, brát v úvahu, jaký průřez má prut, na který působí. Neuronová síť má na svém vstupu jak velikost, tak typ průřezu, takže by teoreticky měla být schopná velikosti zatížení interpretovat v kontextu daného průřezu. Otázka je ovšem, zda je tyto souvislosti schopná na předkládané datové sadě pochopit. Typů průřezu je, jak bylo uvedeno, celkem 108. Některé méně používané spojovací operace jsou v trénovací sadě zastoupeny jen v několika stovkách příkladů. Z toho vyplývá, že síť nemá k dispozici uspokojivé množství příkladů od každého typu průřezu, aby mohla pochopit, jak se jednotlivé typy průřezů chovají při různých kombinacích typů zatížení. Dopočítaná hodnota napětí lépe vystihuje velikost zatížení než velikost síly či momentu. Hodnoty napětí jsem měl už k dispozici, tak nebylo složité vyzkoušet, zda se úspěšnost modelu nevylepší. Extrakce 12 příznaků reprezentující zatížení bylo totožná pouze místo hodnot sil a momentů sil byly použity příslušné dopočítané hodnoty napětí v materiálu. Výsledkem testů byl nárůst úspěšnosti modelu na 75 %.

Výsledek ukázal, že uvedená úvaha by mohla být správná, a tak jsem se rozhodl vyzkoušet namísto napětí v materiálu ještě jeho využití. Tato hodnota dává napětí v materiálu ještě do souvislosti s maximálním napětím, které průřez vydrží, a to tak, že nulové napětí vyvolá využití 0 a maximální napětí využití 1. Tato informace by mohla ještě lépe vystihnout, jak moc je daný prut zatížen. Po testování se ukázalo, že úspěšnost klasifikace je při použití těchto hodnot vyšší a dosahuje hodnoty 77 % správně klasifikovaných validačních vzorků. V rámci trénovací sady byla úspěšnost 92 %.

7.5.4 Topologie

V rámci postupného vylepšování modelu, jsem po jednotlivých iteracích sestavoval matice záměn a snažil se analyzovat chyby v klasifikaci, na základě čehož pak upravovat vstupy sítě tak, aby dosahovala lepších výsledků.

Tabulka 7.2 ukazuje hodnoty matice záměn při klasifikaci validační sady dat po výše uvedených úpravách. Nejnižší úspěšnosti dosahovala síť při klasifikaci úhelníků. V tabulce je vidět, že tento typ připojení je správně klasifikován pouze v 22,18 % případů. Důvodem tohoto problému je pravděpodobně příliš malé množství vzorků z třídy úhelník, kterých bylo jen 797. Druhá třída, která nebyla příliš dobře klasifikována, byl kloub, jen 47,99 % vzorků této třídy bylo zařazeno správně. Ostatní třídy dosahovaly relativně dobrých výsledků.

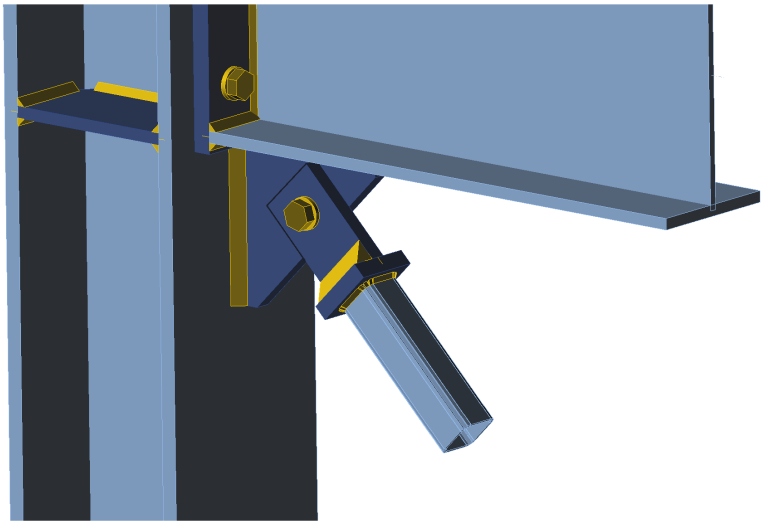
Problém s ne příliš úspěšnou klasifikací kloubových připojení mě navedl na možnou úpravu vstupních vzorků, tedy přidání topologie styčníku jako jeden ze vstupů modelu.

Topologie styčníku říká, kolik prutů přichází a jakým způsobem do bodu spojení. Velkou roli zde hraje orientace jednotlivých prutů. Styčníky, ve kterých dochází ke spojení 2 a 3 prutů mají rozdílné topologie. Rozdílné topologie mají rovněž styčníky, ve kterých dochází ke spojení 2 prutů, kde jednou přicházejí do styčníku proti sobě a podruhé svírají pravý úhel.

Kloub je často používán v případě připojování diagonálního táhla mezi dva kolmé pruty. Takové připojení je znázorněno na obrázku 7.4.

		určeno klasifikátorem					
		úhelník	žiletka	k. deska	svaření	kloub	p. k p.
skutečnost	úhelník	**22,18**	27,20	22,18	28,45	0,00	0,00
	žiletka	1,82	**67,08**	18,50	12,15	0,23	0,23
	koncová deska	0,55	6,98	**85,88**	6,02	0,45	0,12
	svaření	0,52	5,09	18,24	**72,02**	3,18	0,95
	kloub	0,24	0,47	26,48	24,83	**47,99**	0,00
	prut k prutu	0,00	0,57	4,57	4,86	0,00	**90,00**

Tabulka 7.2: Hodnoty matice záměn před přidáním příznaků reprezentujících topologii styčníku do vektoru vstupních příznaků.

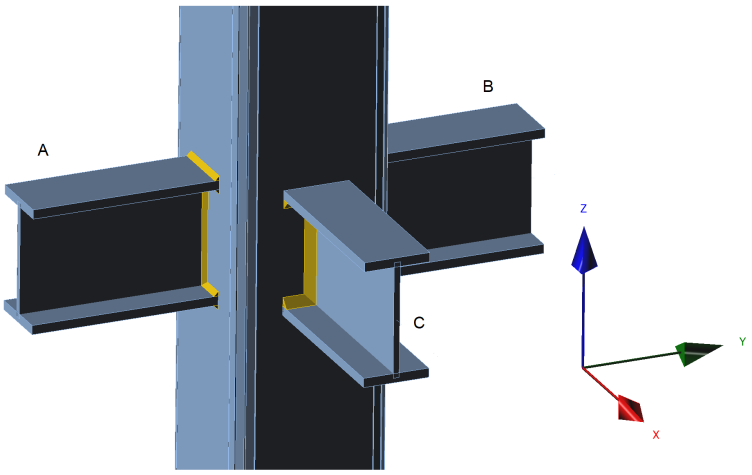


Obrázek 7.4: Kloubové připojení často používané k uchycení diagonálního táhla.

Před přidáním topologie styčníku obsahoval vstup modelu informaci pouze o nosném a připojovaném prutu, v případě připojení uváděného diagonálního táhla na obrázku by to byl svislý a diagonální prut. Ostatní pruty tvoří svá vlastní připojení, připojení vodorovného prutu k svislému na obrázku naopak zase nemá znalost o diagonálním prutu. Informace o tom, že se ve styčníku nachází ještě vodorovný prut, však může modelu napovědět, že by bylo případně možné použít kloubové spojení pro připojení diagonálního prutu mezi svislý a vodorovný. Dokud vstup neobsahoval topologii, neměl model žádnou znalost o zbytku styčníku, znal pouze dva spojované pruty, přitom ostatní pruty mohou ovlivnit rozhodování mezi typy spojení a topologie tak představuje kontext připojení, který by měl model brát při klasifikaci do úvahy. Dalším příkladem, kdy může informace o kontextu styčníku ovlivnit výběr typu spoje, může být informace o tom, zda do styčníku přichází druhý prut z opačné strany proti připojovanému, nebo nikoliv.

Topologii je možné brát globálně a orientaci jednotlivých prutů určovat v rámci globálního souřadného systému, který by byl nějakým způsobem zaveden. Já jsem se však topologii rozhodl vztáhnout vždy k připojovanému prutu daného připojení, protože to je ten prut, jehož připojení může být okolními pruty ovlivněno. Výše uvedený příklad s prutem přicházejícím z opačné strany je vhodný k demonstraci tohoto přístupu. Jestli totiž

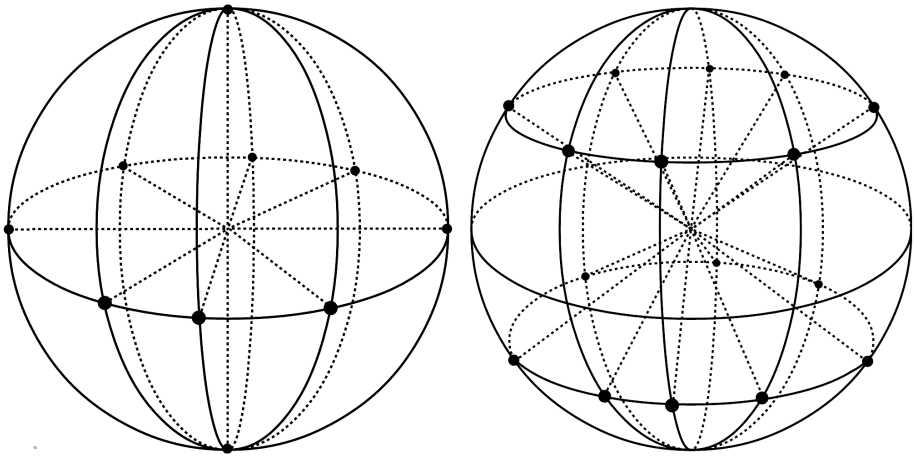
prut přichází proti nebo ze strany je rozdíl, a proto je potřeba topologii vnímat z pohledu připojovaného prutu. Na obrázku 7.5 je tento případ znázorněn.



Obrázek 7.5: Topologie je odlišná z pohledu každého ze 3 připojených prutů.

Z pohledu prutu A přichází prut B proti, z pohledu prutu C však ten stejný prut přichází zprava. Pokud by byla orientace prutů brána globálně v rámci definovaného souřadného systému, který ukazují osy na obrázku 7.5, topologie čili kontext styčníku by byly stejné pro připojení prutů A i C, ovšem z pohledu každého z nich hraje prut B jinou roli a topologie není v obou případech stejná. Myšlenka byla tedy taková, že by vstupní příznaky připojení prutu A zahrnovaly kromě vlastností připojovaného prutu A a nosného svislého prutu, také informaci o tom, že se ve styčníku nacházejí ještě další 2 pruty, které přicházejí z opačné strany (B) a zprava (C). V rámci připojení prutu C pak 2 pruty přicházející zleva a zprava a tak dále obdobně pro jakékoliv připojení.

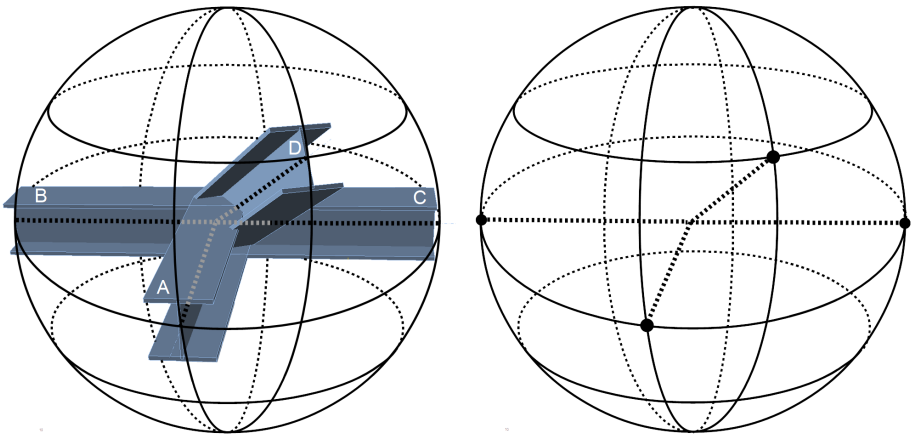
Pruty spojené ve styčníku mohou být sice orientované libovolnými směry, rozhodl jsem se však problém zjednodušit a definovat 26 směrů, které jsou nejčastější, a každému prutu přiřazovat směr, který nejvíce odpovídá danému prutu. Směry znázorňuje obrázek 7.6.



Obrázek 7.6: Nejčastějších 26 směrů prutů ve styčníku.

Směry jsou z důvodu lepší přehlednosti rozděleny do dvou obrázků. Střed koule představuje bod styčníku, tedy bod, kde se střetávají jednotlivé pruty.

Vzhledem k tomu, že prutů může být ve styčníku obecně více, zakódovat přítomné pruty do vstupních příznaků jsem se rozhodl pomocí skupiny 26 příznaků, které mohou nabývat hodnoty 1, pokud se v daném směru ve styčníku nachází prut, nebo 0 pokud žádný takový prut neexistuje. Na obrázku 7.7 je vidět konkrétní styčník.



Obrázek 7.7: Příklad určení topologie styčníku.

Z bodu styčníku vychází 4 pruty. Pruty B a C v podstatě představují jediný průběžný prut, tedy ten, který prochází bodem styčníku, ale v tomto případě je vhodné prut rozdělit na dva s opačnou orientací. Ke všem prutům byly nalezeny odpovídající směrové vektory, které jsou vyznačeny na pravém obrázku. Jak je vidět, celý styčník byl rotovaný kolem svislé osy tak, aby orientace prutů byly určovány z pohledu připojovaného prutu, tedy prutu A, a topologie byla tedy k tomuto prutu vztažena.

V rámci připojení prutu A tedy topologie obsahuje prvky: zpět (A), doleva (B), doprava (C) a zpět doprava nahoru (D). Tyto 4 příznaky z celkových 26 kódujících topologii budou

nastaveny na hodnotu 1. V rámci vytváření příznaků topologie, kdy byl styčník rotován kolem svislé osy, jsem dále zkusil stejným způsobem normalizovat i směrové vektory nosného i připojovaného prutu a těchto 6 příznaků také zařadit do vstupního vektoru. Stejně tak jsem normalizoval i rotaci nosného prutu kolem své střednice, aby bylo lépe zachyceno, jakými stěnami jsou k sobě oba pruty postaveny. Ukázalo se, že těchto 7 příznaků také lehce vylepšilo výsledek.

Po přidání kompletní topologie vrostl počet příznaků definujících jedno připojení na 285. Úspěšnost klasifikátoru po této úpravě stoupla na 81 % správně klasifikovaných validačních vzorků a v rámci trénovací sady pak na 94 %. Hodnoty matice záměn po úpravách ukazuje tabulka 7.3.

		určeno klasifikátorem					
		úhelník	žiletka	k. deska	svaření	kloub	p. k p.
skutečnost	úhelník	**30,99**	20,66	22,07	25,82	0,00	0,47
	žiletka	2,35	**59,11**	19,98	17,98	0,24	0,35
	koncová deska	0,47	3,56	**88,02**	7,50	0,41	0,04
	svaření	0,77	2,28	14,26	**80,01**	2,07	0,61
	kloub	0,00	0,94	14,55	13,38	**71,13**	0,00
	prut k prutu	0,00	0,29	1,44	9,20	0,00	**89,08**

Tabulka 7.3: Hodnoty matice záměn poté, co do vektoru vstupních příznaků byly přidány příznaků reprezentujících topologii.

Schopnost modelu lépe klasifikovat spojení typu kloub podle očekávání vzrostla, úspěšnost se dokonce zvýšila u spojení typu úhelník a svaření, naopak ale klesla u typu žiletka, což nevím, čím mohlo být způsobeno.

7.5.5 Problém s disjunktností tříd

Výše popisovanými úpravami vstupních příznaků se mi podařilo alespoň trochu vylepšit úspěšnost modelu. Už při prvních testech se ale ukázalo, že některé třídy jsou si blízké a za určitých podmínek zaměnitelné. Úplné sloučení těchto tříd, však není možné, jelikož zaměnitelnost neplatí vždy. Během práce na úpravách vstupních příznaků se tak stále více potvrzovalo, že větším problémem než nevhodně zvolené vstupní příznaky byly nevhodně zvolené výstupní příznaky. Postupně se totiž ukázalo, že velké množství vzorků vyhovovalo více než jedné třídě a tedy, že zvolené třídy nejsou disjunktní. Ovšem z projektových souborů jsem měl k dispozici pouze operaci, kterou zvolil statik, když daný styčník navrhoval, tedy zařazení do jedné třídy. Informaci o tom, že v daném případě bylo možné použít i jinou spojovací operaci, tedy připojení zařadit i do jiné třídy, jsem neměl. Když statik navrhuje styčník, stačí, když najde jediné řešení, jakým způsobem prut připojit, ve skutečnosti však může existovat i alternativní řešení. Zároveň zvolená metrika, která spočívala ve zjišťování správně klasifikované části validační sady, nemusela z tohoto důvodu fungovat zcela správně. Pokud je model validován na vzorku, který byl zařazen do třídy A, ale správné by bylo i jeho zařazení do třídy B, a model vzorek správně zařadí do třídy B, je zařazení pochopitelně vyhodnoceno jako chybné. Kvůli tomu mohl vycházet výsledek při validování modelu nižší, než jaký ve skutečnosti měl být.

7.5.6 Určování pravděpodobností tříd

Nejlepším řešením by bylo získat pravděpodobnosti všech tříd, a tak informaci o tom, že pro dané připojení je případě vhodné více různých spojovacích operací.

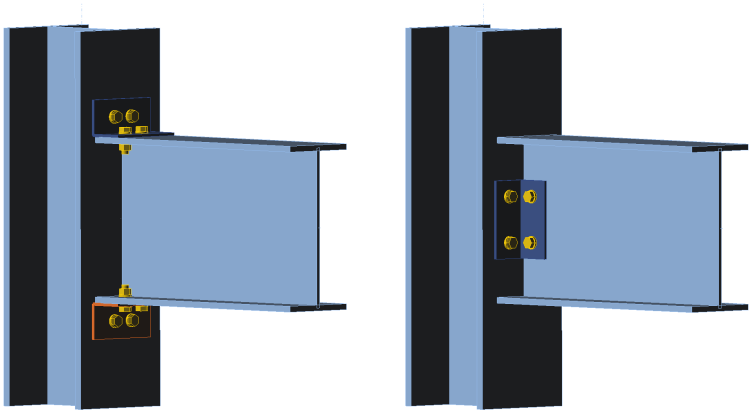
Nejprve jsem se snažil blíže definovat zmíněné podmínky, za kterých by měly být třídy zaměnitelné. Příkladem může být přítomnost kroucení prutu v rámci jeho zatížení. Pokud statik použil připojení na koncovou desku, které je tuhé a odolné vůči kroucení, a prut je namáhán kroucením, není možní použít připojení úhelníkem, protože toto připojení kroucení nevydrží. Naopak, pokud kroucení není přítomné, je možné koncovou desku s úhelníkem zaměnit. Prvotní myšlenka relativně jednoduchých pravidel, která by definovala, za jakých okolnosti je možné různé skupiny operací zaměnit, se nakonec ukázala jako nereálná, protože takových pravidel, která by postihla kompletně všechny možnosti, by bylo příliš mnoho.

Další možností, jak by bylo možné problém vyřešit, bylo ruční zařazování vzorků do tříd. Postup by byl takový, že by statik postupně otevíral projektové soubory a určoval vhodnosti jednotlivých operací, a to například na stupnici: vyhovuje, částečně vyhovuje a nevyhovuje. Tyto stupně by odpovídaly například hodnotám 1, 0,7 a 0. Výsledná hodnota, představující vhodnost dané operace pro dané připojení by mohla být navíc průměrem přes výsledky několika statiků, kteří by zařazovali stejnou sadu vzorků. Modelu by tedy nebyl jako správný předkládán výstupní vektor, ve kterém se nachází pouze jedna hodnota 1, ale hodnot 1 by v něm mohlo být více v případě několika vhodných operací a dokonce by mohl obsahovat necelé hodnoty ukazující částečnou vhodnost dané operace. Jako optimální jsem si představoval 5 zprůměrovaných výsledků. Pokud by zpravování jednoho připojení statikovi trvalo minutu, vzhledem k tomu, že by se musel zorientovat ne jen v topologii styčníku, ale i v zatíženích jednotlivých prutů, znamenalo by to celkem přes 2,5 tisíce hodin práce. Pokud by nebylo průměrováno 5 hodnot, věrohodnost výsledků by jednak byla značně oslabena, ale i tak by časová náročnost byla příliš vysoká. Tento postup jsem tedy také zavrhnul.

7.5.7 Přehodnocení tříd

Asi poslední možnou cestou k vylepšení modelu, by mohla být kompletní úprava výstupních tříd. To, že jednotlivé třídy odpovídaly použitým spojovacím operacím, vypadala ze začátku nadějně, ale, jak se ukázalo, nebyla to pravděpodobně nejvhodnější možnost. Aplikace měla statikům poskytovat radu v podobě návrhu vhodné spojovací operace, a tak se ztotožnění tříd s jednotlivými operacemi nabízelo.

Zaměnitelnost operací pravděpodobně pramení z toho, že stejná operace může ve styčníku plnit různé funkce, jak ukazuje obrázek 7.8.



Obrázek 7.8: Různá použití spojení prutů pomocí úhelníku.

Na obrázku 7.8 je vidět dvojí použití spojovací operace úhelník. Tyto dva vzorky by byly díky stejné operaci zařazeny do stejné třídy. Ovšem v obou případech plní úhelník jinou úlohu, obě připojení jsou tedy vhodná v jiné situaci a pro jiná zatížení připojovaného prutu. Z rozdílnosti obou spojení pak vyplývá, že každé z nich je zaměnitelné jiným způsobem a nelze jednoduše říct, že úhelník je pokaždé zaměnitelný s jinou konkrétní operací.

Kromě operací samotných by tedy bylo vhodné brát do úvahy i jejich funkci v konstrukci, která je snadno rozlišitelná i v rámci automatického zpracovávání projektových souborů. V případě úhelníku by se jednalo například o rozlišení, zda úhelník připojuje pásnici profilu (nalevo), nebo stojnu profilu (napravo). Analogicky by bylo nejspíš možné najít několik odlišných použití i u ostatních spojovacích operací. Po jejich definování by se třídy rozpadly na násobně větší počet, kde by každá reprezentovala konkrétní funkční použití dané operace. Třídy by poté byly slučovány, jelikož shodnou funkci v konstrukci může plnit více různých spojovacích operací, a tak by vznikly nové třídy, které by byly oproštěné od vazby na konkrétní operaci, ale spíše by je definovala společná funkce, kterou spojení zastávají ve styčníku. Takovéto třídy by poté mohly fungovat lépe a vést k vytvoření přesnějšího modelu.

Kapitola 8

Možná pokračování práce

Pokračováním práce, které se nejvíce nabízí, je zmíněná úprava tříd, mezi kterými klasifikátor rozhoduje.[1] Tato změna by mohla výrazně zvýšit úspěšnost klasifikátoru, jelikož by byl odstraněn či alespoň potlačen problém neznalostí pravděpodobností všech tříd popsaný výše. Stejný problém by mohl být vyřešen i popsaným ručním značkováním v případě, že by byl nalezen nějaký vhodný postup, jak by tato operace šla provést s násobně nižší časovou složitostí.

V případě, že by byly třídy upraveny, byl odstraněn popsaný problém, ale stále by model nedosahoval uspokojivých výsledků, dalším postupem by mohlo být další přidávání příznaků, a to především těch, které reprezentují okolí připojení. Jak bylo uvedeno, připojení nebylo zprvu posuzováno s ohledem na své okolí a poté, co byla mezi vstupní příznaky přidána topologie styčníku obsahující informaci i o ostatní prutech ve styčníku, úspěšnost modelu vzrostla. Topologie, jak byla mezi příznaky doplněna, ale obsahuje pouze orientace těchto prutů. Chybí zde spousta informací, která jsou ukládána například u připojovaného nebo nosného prutu, jako je velikost, typ průřezu, atd. Tyto informace by teoreticky také mohly úspěšnost modelu posunout. V průběhu práce jsem si totiž uvědomil, že prvotní jednoduchý koncept jednoho vzorku jakožto dvojice připojený a nosný prut nemusí zdaleka stačit a konkrétní připojení je třeba posuzovat v kontextu celého styčníku, či dokonce části celé konstrukce nacházející se v okolí daného styčníku.

Dalším krokem by pak mohlo být doplnění algoritmu pro určování parametrů operací, jelikož zařazení do třídy, tedy určení spojovací operace, je jen první krok a následně je třeba dopočítat parametry operace, tedy například rozměry úhelníku či počet šroubů v rámci operace úhelník. Tyto parametry by bylo možné určovat buď na základě jednoduchých pravidel a odvodit vzorce pro výpočet jednotlivých hodnot, nebo u složitějších použít k určování jiný natrénovaný klasifikátor. Poté by bylo možné zprovoznit aplikaci pro automatický návrh styčníků, která by nejprve jednotlivá připojení rozřadila do tříd a poté dopočítala parametry spojů, čímž by provedla kompletní návrh daného styčníku.

[1] Kapitola 7.5.7 Přehodnocení tříd

Kapitola 9

Závěr

V rámci této práce jsem si dal za cíl navrhnout a implementovat aplikaci, která bude umožňovat automatické zpracovávání projektových souborů a bude získaná data o spojených prutech ukládat do databáze. Následně bude umožňovat nad získanými daty spustit učení neuronové sítě a natrénovat tak klasifikátor, který bude schopen u nespojených prutů určovat, jaký typ spojení je nejvhodnější.

Podařilo se identifikovat množinu vstupních a výstupních příznaků, extrahovat ji ze souborů, a vytvořit tak datovou sadu pro trénování modelu strojového učení. Dále se podařilo navrhnout podobu samotné neuronové sítě a na získané datové sadě ji natrénovat.

Aplikaci se tedy zprovoznit podařilo a byla dovedena do stavu, kdy je schopna mezi 6 tříd odpovídající typům spojení správně klasifikovat 81 % vzorků neviděné validační sady. Dosažený výsledek jistě není dokonalý, ale na práci je možné dále navázat a výsledek vylepšovat, vzhledem k tomu, že automatický návrh styčníků pomocí umělé inteligence zatím není běžně používanou praktikou a vývoj takových nástrojů je spíše v začátcích. V rámci práce jsem také nastínil její možná pokračování, která by mohla vyřešit problémy vzniklé během práce a která by mohla vést k lepším výsledkům klasifikace.

Literatura

[1] Brown, D., Iles, D., Brettle, M. a Malik, A. *Joints in steel construction : moment-resisting joints to Eurocode 3*. Ascot London: Steel Construction Institute British Constructional Steelwork Association Ltd, 2013. ISBN 978-1-85-942209-0.

[2] Chao, W.-L. Machine Learning Tutorial. In: [online]. National Taiwan University, Prosinec 2011 [cit. 2020-05-20]. Dostupné z: http://disp.ee.ntu.edu.tw/~pujols/Machine%20Learning%20Tutorial.pdf.

[3] Molnar, C. *Interpretable machine learning : a guide for making Black Box Models interpretable*. Morisville, North Carolina: Lulu, 2019. ISBN 9780244768522.

[4] Moreno, E. N., Tarbé, C. a Brown, D. *Joints in steel construction : simple joints to Eurocode 3*. Ascot London: Steel Construction Institute British Constructional Steelwork Association Ltd, 2011. ISBN 978-1-85942-201-4.

[5] Ng, A. *Machine Learning Yearning*. Online Draft, 2017. Dostupné z: http://www.mlyearning.org/,/bib/ng/ng2017mlyearning/Ng_MLY01_13.pdf.

Příloha A

Obsah paměťového média

Přiložené paměťové médium obsahuje:

- tento dokument ve formátu PDF,
- zdrojové soubory LaTeX pro vytvoření tohoto dokumentu,
- spustitelnou aplikaci, která v rámci práce vznikla a
- zdrojové soubory aplikace[1].

[1] Připojené zdrojové soubory jsou upravené, a to v rámci vrstvy zpracovávající projekty z důvodu firemního tajemství společnosti IDEA StatiCa. Vrstva zpracovávající projekty se připojuje k uložišti, odkud stahuje projektové soubory ke zpracování. K tomuto připojení však nemohly být poskytnuty přihlašovací údaje. Stejně tak v rámci těžení dat ze souborů aplikace používá funkcionality aplikace Connections společnosti IDEA StatiCa, která není volně šiřitelná. Také databáze implementovaná v rámci práce je umístěna na placeném serveru a proto opět nejsou uvedeny přihlašovací údaje. Zdrojové texty tedy neobsahují popsané pasáže a spustitelná aplikace postrádá odpovídající funkcionality. Zbytek aplikace, tedy především vrstva strojového učení, je přiložen v nezměněné podobě. Protože vrstva strojového učení nemůže využívat data ukládána během zpracovávání projektů do databáze, byly projekty předzpracovány a získaná data ve formátu JSON uložena do souborů v rámci zdrojových souborů. Takto může trénování klasifikátoru fungovat i po odstranění popsaných pasáží. Podrobnější popis funkcionalit je uveden v souboru `readme`.