ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

FAKULTA
STROJNÍ

DIPLOMOVÁ PRÁCE

2022

ONDŘEJ
DUNÍK

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Duník** Jméno: **Ondřej** Osobní číslo: **470070**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav mechaniky, biomechaniky a mechatroniky**

Studijní program: **Průmysl 4.0**

Studijní obor: **bez oboru**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Prototyp prediktivního systému s podporou příznakového inženýrství a strojovým učením

Název diplomové práce anglicky:

Prototype of predictive system with the support of feature engineering and machine learning

Pokyny pro vypracování:

Cílem diplomové práce je vytvořit aplikaci v projektu digitálního dvojčete výrobní linky pro predikci výsledku zkoušky pomocí metod strojového učení s podporou příznakového inženýrství. Vstupními parametry jsou veličiny naměřené při výrobě sacího modulu výrobku DNOX 5.3, ze kterých se bude predikovat výsledek parametru funkční zkoušky.
1) Rešerše problematiky a existujících řešení,
2) Rešerše a shrnutí relevantních metod pro návrh konfigurací vstupních vektorů pro prediktivní učící se systémy typu lineárních a nelineárních regresorů a dopředných neuronových sítí (např. Boruta, MSFNA, a další).
3) Rešerše a shrnutí relevantních metod pro vyhodnocení lineárních a nelineárních závislostí mezi vstupními daty (vektory) a výstupní (predikovanou) veličinou (např. vzájemná informace, a další, využití lineárních či polynomiálních regresorů typu HONU).
4) Zvolte vhodnou metodiku pro výběr příznaků, tj. metodu ad 2) či agregaci více metod, zvolte vhodnou metodiku ad 3) ověření volby příznaků
5) Naprogramujte aplikaci, která bude:
a. realizovat vámi zvolené metody příznakového inženýrství na vstupní dataset 2)-4)
b. trénovat prediktory LNU, QNU, a jednovrstvé MLP sítě na trénovacím datasetu s dávkovým algoritmem učení pro navržené konfigurace vstupů, a na testovacím datasetu bude validovat a testovat kvalitu predikce
6) Použijte vyvinutou aplikaci pro dostupný dataset a analyzujte její funkčnost a přesnost predikce a vyhodnoťte její využitelnost pro průmyslovou praxi.

Seznam doporučené literatury:

[1] Kumar, V.; Minz, S. Feature selection: A literature review. SmartCR 2014, 4, 211–229.
[2] K. Marzova and I. Bukovsky, "Feature Selection and Uncertainty Analysis for Bubbling Fluidized Bed Oxy-Fuel Combustion Data," Processes, vol. 9, no. 10, p. 1757, Sep. 2021, doi: 10.3390/pr9101757.
dále dle doporučení vedoucím

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Ivo Bukovský, Ph.D., U12105

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **22.11.2021** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: _____

doc. Ing. Ivo Bukovský, Ph.D.	doc. Ing. Miroslav Španiel, CSc.	prof. Ing. Michael Valášek, DrSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

Anotační list

Jméno autora:	Ondřej Duník
Název diplomové práce:	Prototyp prediktivního systému s podporou příznakového učení a strojovým učením
Anglický název	Prototype of predictive system with the support of feature engineering and machine learning
Akademický rok	2021/2022
Studijní program	Průmysl 4.0
Vedoucí diplomové práce:	doc. Ing. Ivo Bukovský, Ph.D.
Bibliografické údaje:	Počet stran: 71 Počet obrázků: 30
Klíčová slova:	Prediktivní systém, Příznakové inženýrství, Výběr rysů, Strojové učení, Datová analytika, Průmysl 4.0
Keywords:	Predictive model, Feature engineering, Feature selection, Machine learning, Data analysis, Industry 4.0

Anotace: Tato diplomová práce se zabývá predikcí výsledku zkoušky na základě analýzy průmyslových dat. Ze stovek měřených parametrů na montážních linkách bylo pomocí příznakového inženýrství zredukováno množství veličin na deset nejrelevantnějších. Z nich poté byly trénovány a optimalizovány modely strojového učení. Na základě algoritmu poskytujícího nejpřesnější výsledky byla vytvořena aplikace pracující ve webovém rozhraní.

Abstract: This diploma thesis deals with result prediction of test based on industrial data analysis. From hundreds of parameters measured on assembly line, the dataset was reduced on ten most the most relevant. On this dataset the machine learning models were trained and optimized. The algorithm with the best results worked in back-end of developed web application.

Čestné prohlášení

Prohlašuji, že tuto diplomovou práci jsem vypracoval samostatně a pouze za použití zdrojů uvedených v seznamu.

V Praze dne: ……

Poděkování

Děkuji svému vedoucímu diplomové práce panu doc. Ing. Ivovi Bukovskému, Ph.D. za jeho cenné rady a čas věnovaný konzultacím. Dále bych chtěl poděkovat své rodině a přátelům, kteří mě podporovali během celé doby studia.

Obsah

Úvod

S postupujícími změnami klimatu je automobilový průmysl pod stále větším tlakem z pohledu emisních regulací. Hlavní požadavek ujednaný v článku 2 Pařížské dohody podepsané v roce 2016 ukládá za cíl signatářům udržet nárůst průměrné globální teploty výrazně pod hranicí 2 °C oproti hodnotám před průmyslovou revolucí [1]. Toho automobilky mohou dosáhnout přechodem na alternativní pohony, nejčastěji lithium-iontové či vodíkem napájené elektromobily. Nástup elektromobility je ovšem v současné době z mnoha důvodů pozvolný a je tedy zásadní pro mezidobí transformace snižovat množství výfukových plynů ze stále ještě hojně vyráběných aut se spalovacími motory.

Pro snížení emisí dieselových motorů byl vyvinut systém Denoxtronic. V rámci systému Denoxtronic je důležitým prvkem zásobovací (supply) modul, který slouží k distribuci syntetické močoviny do výfukového potrubí s cílem redukce zdraví škodlivých oxidů dusíku na běžný atmosférický dusík. Pomáhá tak plnit emisní cíle kladené na automobilový průmysl. Výrobek musí splnit přísné technické požadavky na spolehlivost po celou životnost automobilu. Toho je dosaženo pomocí pokročilé digitalizace a práce s daty na výrobních linkách a funkčních zkouškách každého dílu.

Cílem diplomové práce je vytvořit prototyp aplikace v projektu digitálního dvojčete produktu pro predikci výsledku zkoušky funkčnosti pomocí strojového učení. Vstupními parametry jsou veličiny naměřené při výrobě sacího modulu výrobku DNOX 5.3, ze kterých chceme předpovědět výsledek parametrů funkční zkoušky. Data jsou sbírána kontinuálně na výrobních strojích a při každé operaci jsou zaznamenány informace, poskytující přehled o parametrech události, jako například maximální přítlačná síla, teplota a doba svařování apod. Tato data jsou ukládána na server do databáze, ze kterých je potřeba je předpřipravit a upravit do formy pro analýzu. Model tak bude sloužit k odladění procesních parametrů strojů pro zlepšení kvality výroby produktu.

Nástroj bude schopný ušetřit čas i peníze díky sériové výrobě s rychlým výrobním tempem a výrobky s vysokou přidanou hodnotou. Pro návratnost tedy bude stačit i malé množství správně odladěných kusů.

1. Popis systému Denoxtronic

Systém Denoxtronic byl vyvinut s cílem významně snižovat současnou ekologickou zátěž způsobenou automobily se vznětovými motory. Základní princip je založen na reakci zvané selective catalytic reaction, často uváděné pod zkratkou SCR, v překladu selektivní katalytická redukce. Systém jako takový významně snižuje množství emisí zdraví škodlivých oxidů dusíku a je zásadní pro splnění emisních limitů Evropské unie, konkrétně o 93 % více než určuje norma pro rok 2022 [2]. Toho je docíleno vstřikováním roztoku tvořeném 32,5 % syntetické močoviny rozpuštěné v 67,5 % destilované vody. Tato kapalina je známa pod komerčním názvem jako AdBlue® a její složení definuje norma ISO 22241 [3].

Chemický vzorec močoviny je $CO(NH_2)_2$ a jedná se o první organickou sloučeninu syntetizovanou z anorganických látek. V případě rozředění vodou do AdBlue® se jedná o kapalinu s lehce zažloutlou barvou, která je pro zdraví bezpečná a netoxická. Při vyšších teplotách má slabý zápach po čpavku. Zamrzá již při teplotách −11,5 °C, proto je třeba zajistit zahřívání nádrže v autě. Dále je kapalina lehce zásaditá s pH 9, což způsobuje korozivní účinky, proto jsou výrobky DNOX vyráběny z plastu [4].

Prvním krokem selektivní katalytické redukce je hydrolýza, tedy rozklad močoviny na oxid uhličitý a čpavek za přítomnosti vody a vysoké teploty ve výfukovém potrubí [5].

$$CO(NH_2)_2 + H_2O \rightarrow 2NH_3 + CO_2$$

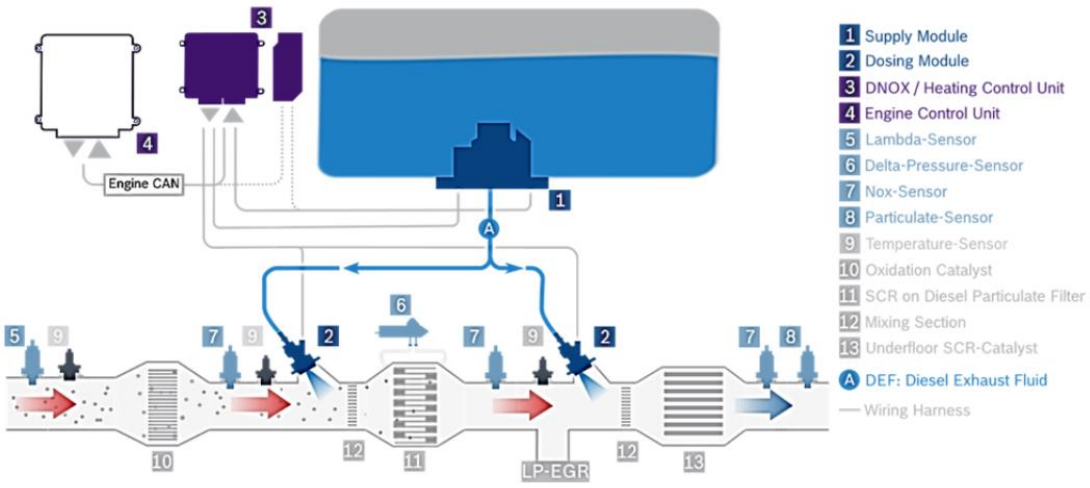
Vzniklý amoniak je poté hlavním katalytickým činidlem, který rozkládá nebezpečný oxid dusný a oxid dusičitý na vodní páru a dusík.

$$NO + NO_2 + 2NH_3 \rightarrow N_2 + 3H_2O$$

Celý systém DNOx tvoří komponenty:

- Nádrž na AdBlue®
- Supply (zásobovací) modul
- Dosing (dávkovací) modul
- Dosing control unit (řídící jednotka dávkování)
- Hydraulické vedení
- Sensory

Kapalina vstupující do výfukového potrubí je nasávána supply modulem. Ten je přivařen na dně nádrže na AdBlue®. Pomocí hydraulického vedení je kapalina dodávána do dosing modulu, který zajišťuje vstřikování kapaliny do výfukového systému. Vše je regulováno řídící jednotkou neboli dosing control unit, která rozhoduje o množství dávkované kapaliny na základě vstupu z mnoha senzorů. Pro příklad lze uvést senzor snímající množství pevných částic, oxidu dusíku, teploty a tlaku ve výfukovém systému.



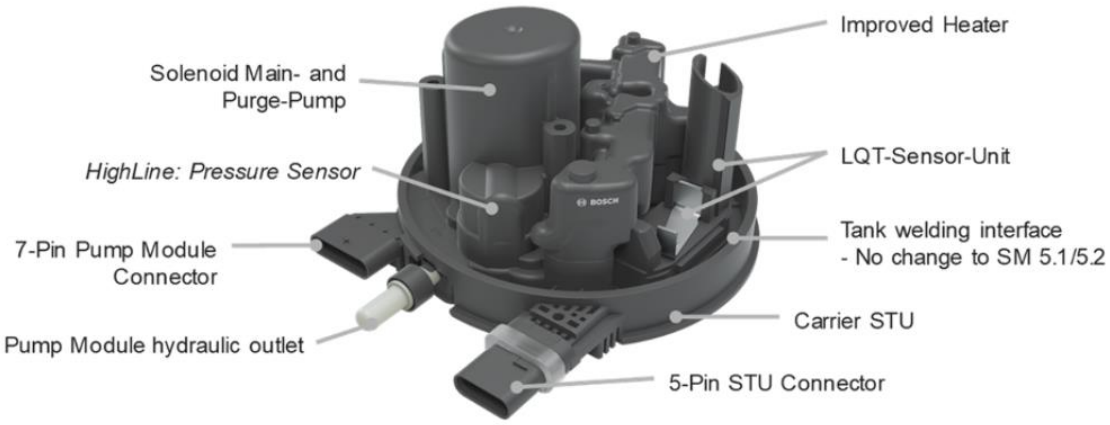
Obr. 1: Přehled funkčnosti systému Denoxtronic [6]

Data jsou sbírána ze strojů na výrobních linkách zásobovacího (supply) modulu a dodavatelů komponent zásadních pro funkčnost výrobku.

1.1. Supply modul

Supply modul má za úkol, jak je již zmíněno výše, zásobovat hydraulické vedení dostatečným množstvím kapaliny pro dávkovací modul. Je složen z následujících komponent:

- Plastový obal
- Pumpovací modul
- nahřívače
- filtru
- konektoru



Obr. 2: Supply modul - vnější pohled [6]

Úkolem plastového šasi je držet všechny komponenty modulu dohromady. Celý výrobek je připevněn zespoda k nádrži na AdBlue® a právě vnější obal je celý obklopený touto kapalinou. Tomuto chemickému prostředí odpovídá materiál, ze kterého je šasi vyrobeno, tedy polyethylen s vysokou hustotou, který je dlouhodobě odolný vystaveným podmínkám. Pro výměnu komponent v případě poruchy je celý obal odnímatelný od nádrže.

Pro zajištění čistoty dodávaného roztoku močoviny s vodou, a tím správné funkce systému DNOX je ve výrobku umístěn filtr. Ten je umístěn v plastovém obalu vyplněném jemnou syntetickou síťovinou. Ta zachytává prachové částice a další nečistoty v nádrži. Další komponentu zásobovacího modulu tvoří nahřívač. Ten vzhledem k teplotě tuhnutí AdBlue® kolem -11 °C udržuje kapalinu tekutou za všech podmínek.

Nejdůležitějším prvkem supply modulu je modul zvaný Pump Module, česky pumpovací modul. Jeho úkolem je pumpovat kapalinu směrem do dávkovacího modulu a zároveň po vypnutí motoru nasát z vedení kapalinu zpět. Roztok by při nízkých teplotách v zimních měsících mohl zamrznout a poničit tak rozvodnou soustavu. Z tohoto důvodu jsou v modulu umístěny dvě pumpy:

- Hlavní pumpa
- Zpětná pumpa

Hlavní pumpa je složena primárně z cívky, která generuje elektromagnetické pole pro posuvný pohyb jádra připojeného k pružné membráně. Ta generuje podtlak v komoře pumpy a nasává kapalinu skrz jednocestný ventil. Vedle tohoto ventilu je umístěn druhý v opačném směru. Tímto mechanismem se reguluje objem AdBlue® dodávaného do potrubí. Komponenta je ovládána z řídící jednotky systému, přičemž komunikace mezi těmito body je zajištěna pomocí dvou pinů z pumpy do konektoru.

Princip fungování zpětné pumpy je stejný jako u hlavní pumpy. Je opět složena z cívky, jádra a membrány. Posun membrány způsobuje podtlak, kterým je nasávána kapalina z hydraulického vedení zpět do nádrže s AdBlue®. Při výrobě modulu jsou měřeny parametry pro zjištění správné funkčnosti výrobku. Jedním z nich je hodnota zpětné volumetriky. Ta představuje množství kapaliny, který byla schopna zpětná pumpa zpracovat jeden cyklus. Predikci této veličiny na základně hodnot z výrobních procesů se věnuje tato diplomová práce.

2. Současný přístup k analýze průmyslových dat

Diplomová práce je součástí projektu digitálního dvojčete produktu. Digitální dvojče je virtualizovaný model jakékoliv fyzikální entity. Ve výrobních provozech je základem pro vývoj digitálního dvojčete práce s daty ze strojů s informacemi o vlastnostech produktu, které poté mohou sloužit k predikci chování systému v různých částech procesu a tím zvýšit produktivitu. To přinese snížení nákladů z těchto důvodů:

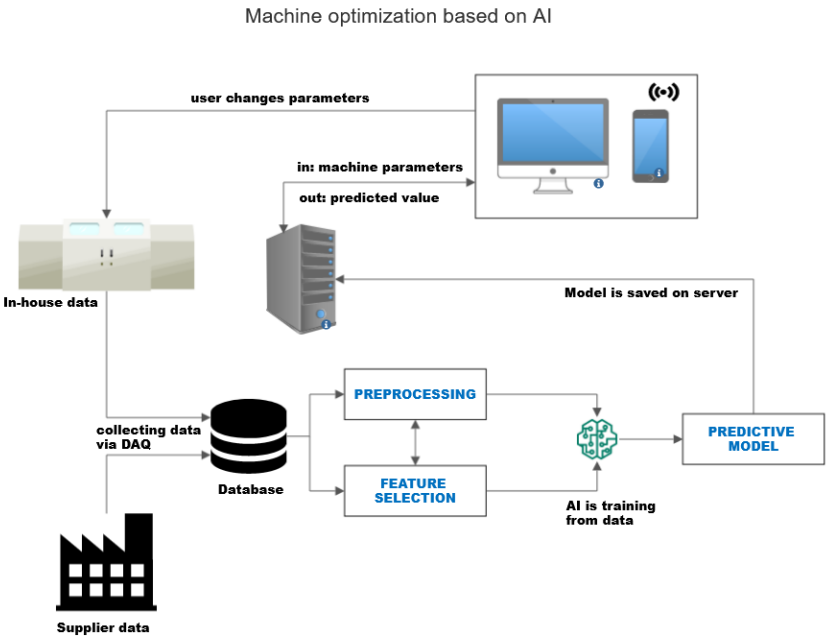
- Zkrácení doby zastavení výroby z důvodu přestavby linky
- Možnost odhalení nedokonalostí v nastavení výrobních procesů ještě před zavedením fyzické linky (tok materiálu, lidské činnosti)
- Odhalení pravé příčiny chyby v lince (Root Cause Analysis)
- Snížení počtu zmetků pomocí odladění výrobních procesů

Nejpokročilejší technologii pro tvorbu digitálního dvojčete v současné době nabízí americká společnost NVIDIA zabývající se výrobou grafických karet, které tvoří jeden z faktorů rychlého nástupu umělé inteligence v posledních letech. Její platforma zvaná Omniverse je technologie sloužící pro vývoj rozsáhlých 3D simulací mnoha různých prostředí. Řešení je založeno na kombinaci mnoha různých aplikací pomocí rozšíření do aplikací. Lze tak spojit třírozměrný model vytvořený v programu Autodesk Inventor s trénovacími algoritmy strojového učení v Pythonu. Pomocí simulací tak například je možné trénovat samořiditelná auta, simulovat šíření signálu sítě páté generace v různých prostředích. V průmyslové praxi bylo představeno využití aplikace Isaac Sim ve výrobním závodě společnosti BMW. Tato aplikace zpřístupňuje simulovat ve virtuálním prostředí nejčastější AI řešení, jako například autonomní roboty, prediktivní údržbu či big data analýzu. To může pomoci řešit problém s nedostatkem výrobních dat při snižování množství vyrobených kusů. Tato algoritmy uměle vytvořená data mohou teoreticky sloužit pro optimalizaci procesů stejně dobře jako reálně naměřená [7]. K tomu slouží tzv. generativní soupeřící neuronové sítě. Koncept je založen na soupeření dvou neuronových sítích. První síť upravuje parametry a tím i svůj výstup s cílem zmást druhou kontrolní konvoluční síť na rozpoznání obrazu. Ta se snaží klasifikovat výstup předchozí neuronové sítě. Tento postup může vygenerovat dostatečné množství NOK stavů při vizuální kontrole montáže výrobku [8].

Prediktivní modely založené na umělé inteligenci jsou jedním z nejzmiňovanějších prvků koncepce chytré továrny, obecně nazývané jako Průmysl 4.0. Pojem byl poprvé definován v roce 2013 na konferenci v Hannoveru profesorem Wolfgangem Wahlerem z tamější univerzity. Hlavní teze čtvrté průmyslové revoluce představuje zavedení kyberneticko-fyzikální systémů, tedy strojů neustále připojených k internetu zasílajících data do centralizovaných cloudových úložišť pro další zpracování a optimalizaci. Jednotlivé složky tohoto systému budou moct vzájemně komunikovat, což přinese možnost řídit výrobu osobou vzdáleně přes internet či bez zásahu člověka pomocí umělé inteligence. Pro tento směr vývoje se užívá název Internet of Things, tedy Internet věcí, nejčastěji uváděný pod zkratkou IoT. Pro zajištění stabilní konektivity velkého množství strojů připojených k internetu, které posílají velké množství dat se předpokládá vytvoření sítě 5G.

2.1. Průmyslová umělá inteligence

Využití umělé inteligence v průmyslových podnicích významně roste. Například ve společnosti Bosch v roce 2020 více než 90 % výrobků obsahovalo prvky umělé inteligence, či byly vyrobeny na linkách optimalizovaných umělou inteligencí [9]-



Obr. 3: Architektura systému pro zpracování dat v průmyslu, upraveno dle [10]

Pro predikci výsledků parametrů zkoušky v mém případě budu využívat programovací jazyk Python 3.6. v distribuci Anaconda, která má předinstalované velké množství nejběžnějších datově-analytických knihoven. Python představuje nejběžnější řešení pro datovou analýzu dat, kromě něj se využívají jazyky R či Julia. Jazyk Python je poměrně pomalý, jelikož se jedná o interpretovaný jazyk C. Hlavní výhodou, pro kterou se stal velmi populárním pro využití ve strojovém učení, je jeho přehledná syntaxe, která dovoluje výrazně rychlejší psaní kódu a tím snadnější učení jazyka pro začátečníky. Díky těmto vlastnostem nachází využití umělé inteligence ve stále více oborech přímo nesouvisejících s informativními technologiemi od finančnictví po průmysl.

Použité datově-analytické knihovny:

- Pandas – knihovna napsaná převážně v jazyce Python pro práci s daty
- Numpy – matematická knihovna napsaná převážně v jazyce Python, C (dohromady tvoří Cython) pro rychlou práci s vektory a maticemi
- Scikit-learn – knihovna pro machine-learning v Pythonu
- XGBoost – knihovna pro algoritmus gradientních rozhodovacích stromů
- Optuna – framework pro optimalizaci hyperparametrů neuronových sítí
- BorutaPy – knihovna pro aplikaci algoritmu Boruta

3. Popis dat

Data, se kterými pracuji, pochází z výrobních linek dávkového modulu pro systém DNOX do dieselových automobilů. Na každém výrobním stroji jsou zaznamenány různé parametry o proběhlém procesu. Typicky se jedná o informace jako jsou maximální aplikovaná síla při lisování, teplota svařování nebo čas probíhajícího procesu. Kromě těchto dat je ukládáno také velké množství veličin, ze kterých lze stejnými algoritmy poměrně těžko predikovat žádané hodnoty. Jedná se například o kódové označení svařovacího programu stroje či booleovská hodnota o dodržení stanovených dílčích parametrů. V mém případě tedy nejprve musím pomocí pre-processingu získat dataset obsahující nominální hodnoty popisující výrobní procesy.

Data jsou analyzována ze dvou výrobních linek:

- Dodavatele, který vyrábí magnet do části zpětné pumpy
- Firmou, kde se vyrábí celý dávkovací modul

	RESULT_DATE	UNIQUEPART_...	DmcHubm...	final_force...	final_force...	final_stroke...	final_stroke...	ProcessCount
1	2020-11-15 13:06:48.997	0x7C947D8C4...	08772032...	101,0927	8,853911	18,58633	17,80883	1
2	2020-11-15 13:09:59.140	0xB79FEE238...	08772032...	104,2736	9,029332	18,58	17,80854	1
3	2020-11-15 13:19:13.207	0x8FF4B09616...	08772032...	102,2579	7,534189	18,57953	17,81141	1
4	2020-11-15 13:44:26.690	0xA95796D3F4...	08772032...	103,5955	10,98762	18,58053	17,81083	1
5	2020-11-15 13:45:10.487	0xC4AE39B9D...	08772032...	96,42531	8,17585	18,59063	17,81061	1
6	2020-11-15 13:45:56.497	0xA93DEEB7C...	08772032...	98,31969	9,255095	18,58718	17,81064	1
7	2020-11-15 13:47:41.547	0x4CE5D2F64...	08772032...	98,45678	9,379012	18,58202	17,80914	1
8	2020-11-15 13:49:21.083	0x9D29AB547...	08772032...	86,87746	10,0745	18,58608	17,81098	1
9	2020-11-15 13:58:02.230	0x6D8F03523E...	08772032...	99,0419	8,489516	18,58656	17,80862	1
10	2020-11-15 13:58:42.517	0xECD81C7F8...	08772032...	99,39778	8,546828	18,59376	17,8132	1

Obr. 4: Přehled měřených dat na výrobních strojích

Každý řádek tabulky je řazen podle času, kdy byl zaznamenán výsledek měření. Parametry tedy nejsou zaznamenávány s pravidelnou frekvencí, ale nepravidelně při výrobě dílu. Kromě tohoto parametru je také ve všech datech načtený laserem vytištěný unikátní kód výrobku a pokud je možno také dalších vstupních komponent. Pro analýzu, z jakých dílů jsou smontovány jednotlivé výrobky, poskytuje kvalitní řešení program Tableau. Pro zpracování pomocí strojového učení je ovšem lepší využít řešení poskytující programovací jazyk Python a jeho knihovny.

3.1. Popis databáze

Data z výrobních strojů jsou ukládána do relační databáze na serveru a pro komunikaci je použit jazyk SQL. Jedná se o standardizovaný dotazovací jazyk pro práci s těmito databázemi. Každý záznam v této tabulce musí mít svou jedinečnou nezávislou entitu, pomocí které se dosáhne entitní integrity tabulky nazývané jako primární klíč. Tím jsou v našich tabulkách unikátní hodnoty vygenerovány pomocí hashovací funkce na prvním stroji a laserem vytištěny v podobě DMC kódu přímo na produkt. Každý následující výrobní stroj obsahuje čtečku těchto kódů a při dalších operacích jsou snímány, čímž jsou měřené hodnoty přiřazeny ke konkrétním kusům konečných výrobků i jednotlivých komponent.

Vzhledem k možnosti vícenásobného zaznamenání jednoho stejného kusu z důvodu například opakování zkoušky, je vhodné z databáze získávat pouze unikátní kusy. To zajišťuje příkaz `SELECT DISTINCT.` Prvním sloupcem, který je sbírán z tabulky, je predikovaná hodnota `070_RsmVolumetric`, poté jednotlivé výrobní stroje v pořadí za sebou. Ilustračně uvádím první tři parametry z SQL dotazu.

```
SELECT DISTINCT [DATABASE].[TAB].[PARAMETR1].[070_RsmVolumetric],
    [DATABASE].[TAB].[PARAMETR1].*,
    [DATABASE].[TAB].[PARAMETR2].*,
```

Propojení jednotlivých tabulek z českobudějovických linek pomocí SQL provedeme příkazem `LEFT JOIN`. Ten vrátí všechny záznamy z levé tabulky v příkazu a odpovídající záznamy z tabulky druhé. Pro spojení slouží unikátní hodnoty primárních klíčů, jež se nazývají v tabulkách jako `[UNIQUEPART_HK].` Zkratka HK znamená Hash Key, tedy kód vygenerovaný hashovací funkcí a laserem vytištěn jako DMC kód na výrobcích, jak je popsáno výše.

```
      FROM [DATABASE].[TAB].[F_ST0140]
LEFT JOIN [DATABASE].[TAB].[F_ST0020] ON
[TAB].[PM].[F_ST0140].[UNIQUEPART_HK] = [TAB].[PM].[F_ST0020].[UNIQUEPART_HK]
```

Na serveru se nachází tabulky ze všech výrobních linek dodavatelů komponent do výsledného zařízení. Propojení mezi tabulkami komponentů a celého výrobku je náročnější, jelikož je iniciováno pomocí jiných primárních klíčů než hodnoty Hash Key vygenerovaných na konečné výrobní lince. Jedná se o parametry DmcHubmagnet v konečné databázi zaznamenávající stejný kód jako vygenerovaný kód na obalu magnetu na lince dadavetele.

```
LEFT JOIN [DATABASE].[TAB].[F_ST020410] ON
[DATABASE].[TAB].[F_ST0050].[DmcHubmagnet] =
[DATABASE].[TAB].[F_ST020410].[BearingIdentifier]
```

Pro trénování dat jsem stanovil rozsah mezi květnem a srpnem roku 2021. To dovoluje v SQL příkaz pro filtrování hodnot *where.*

```
WHERE [IM_DNOX53].[PM].[F_ST0020].RESULT_DATE
BETWEEN '2021-06-05 00:00:00' AND '2021-07-09 23:59:59';
```

Tímto dotazem jsem získal dataset obsahující přes 65000 záznamů v řádcích a 478 sloupcích hodnot. Tato data je nutné před trénováním upravit, čemuž se věnuje následující kapitola.

3.2. Data pre-processing

Pro práci s rozsáhlými datovými sadami je vzhledem k velkému množství nadbytečných informací potřeba před analýzou data vhodně upravit tak, aby bylo možné aplikovat na dataset vhodné algoritmy. Cílem pre-processingu je naměřená data s množstvím chybných záznamů přetransformovat na dataset s čtvercovými rozměry a pouze číselnými hodnotami.

Kroky preprocessingu, které jsem provedl, byly:

- rozdělení tabulek na vstupní matici X a výstupní vektor y
- nahrazení chybějících hodnot
- odstranění nečíselných hodnot
- normalizace dat

Po získání všech dat z databáze pomocí SQL jsem data uložil do jednoho dataframu. To zajišťuje funkce `read_sql` v knihovně pandas. Přijímá parametry query, což je samotný příkaz ve formátu string. Parametr `conn` poté přijímá funkci z modulu `pyodbc`, což je open source řešení napsané v Pythonu pro ODBC, neboli Open Database Connectivity. Jedná se standardizované API pro přístup k databázovým systémům.

```
conn = pyodbc.connect('Driver={SQL Server};'
                      'Server=' + server + ';'
                      'Database=' + database + ';'
                      'UID=' + username + ';'
                      'PWD=' + password)

df = pd.read_sql(query, conn)
```

3.2.1. Rozdělení datasetu

Tento dataframe jsem poté rozdělil na vstupní matici X a výstup y. Predikovaná hodnota je označena jako *070_RsmVolumetric*. Toho jsem docílil uložením jednoho sloupce do proměnné *y*. Tento datový typ pouze jednoho sloupce se neoznačuje jako pandas.Dataframe, ale jako pandas.Series. Pro vytvoření Dataframu vstupních hodnot X jsem využil funkce *drop*, která z původního dataframu odstraní vybrané sloupce.

```
y = df['070_RsmVolumetric']
X = df.drop('070_RsmVolumetric', axis=1)
```

3.2.2. Odstranění NaN hodnot

Po rozdělení dataframu je třeba se zbavit NaN (Not A Number) hodnot. Ty v tomto případě vznikají buď:

- pokud na výrobním stroji není parametr zaznamenán
- pokud v jedné ze spojovaných tabulek neodpovídají primární klíče

Pandas s nimi pracuje jako s chybějícími hodnotami, což způsobuje potíže, jelikož model přijímá pouze pole číselných hodnot. To lze vyřešit mnoha způsoby jako například smazáním celého řádku nebo sloupce obsahujícího chybějící parametry, čímž ztratíme velké množství dat. Lepší možností je nahrazení těchto hodnot jiným číslem. První možností je pouhé nahrazení nulou. K tomu slouží v pandas funkce `fillna()`.

Nejlepší způsob odstranění NaN hodnot nabízí knihovna Scikit-learn s funkcí `SimpleImputer`. K výpočtu přistupuje dvěma způsoby. Pomocí prvního (univariate) odhaduje chybějící informace pouze pomocí hodnot v okolí, například průměru sousedních hodnot nebo podle nejfrekventovanějších záznamů (který funguje i na kategorické hodnoty) [11].

```
X = X.SimpleImputer(missing_values=np.nan, strategy='mean').fit(X)
```

Sofistikovanější metodou doplnění chybějících hodnot je odhadování na základě celého datasetu (multivariate). Tento přístup nabízí experimentální funkce v knihovně Scikit-learn. Ten funguje na principu statistického algoritmu Round-robin [12].

3.2.3. Vektorizace

Dalším krokem pre-processingu je nahrazení odstranění sloupců, které nejsou pro predikci zpětné volumetriky vhodné, tedy především nečíselné veličiny. Jedná se především o hash kódy jednotlivých součástek, které ztratily relevanci po spojení pomocí primárního klíče přiřazeného konkrétnímu výrobku. Další veličinou, které je potřeba se zbavit jsou data a časy, kdy konkrétní operace na stroji přesně proběhla, jelikož došlo k filtraci přes první provedený úkon na lince, a to již v SQL příkazu. Další nečíselné vstupy nutné k odstranění jsou například informace o nastavených programech na strojích

nebo dodatečné poznámky. Po dokončení tohoto kroku nazývaného vektorizace získáme rozsáhlou a hustou matici `X` s hodnotami v datovém formátu float32 v knihovně NumPy.

```
X = X._get_numeric_data()

X = X.to_numpy()
y = y.to_numpy()
```

3.2.4. Škálovaní hodnot

Posledním krokem přípravy datasetu pro správnou funkci algoritmů je škálování hodnot vstupních veličin. Ty jsou velmi rozmanité, zaznamenáváme například tlak s jmenovitými hodnotami v řádu tisíců pascalů a vstupní napětí s hodnotami pohybujícími se okolo jednoho ampéru. Důsledkem toho tak je, že vyšší hodnoty mohou mít nepoměrně vyšší vliv na učení modelu. To se týká až na výjimky (např. Random Forest) většiny algoritmů. Škálování aplikuji na většinu vstupních parametrů, u výstupních hodnot není většinou potřebné. Existují dva obvyklé postupy pro škálování hodnot [13]:

- Normalizace (min-max škálování)
- Standardizace

Postup normalizace dat pomocí min-max algoritmu spočívá ve změně hodnot na odpovídající hodnoty v rozsahu 0 až 1. Toho je dosaženo pomocí odečtení minimální hodnoty a podělením rozdílu maximálních a minimálních hodnot. Knihovna scikit-learn nabízí funkci `MinMaxScaler.`

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{3.1}$$

Při standardizaci neboli z-score normalizaci, nejprve od hodnoty odečteme průměrnou hodnotu a poté podělíme směrodatnou odchylkou. Na rozdíl od min-max normalizace nejsou hodnoty omezeny mezi 0-1. To může představovat problém pro některé typy neuronových sítí, které očekávají vstup od nuly do jedné. Výhoda ovšem spočívá v odolnosti proti ovlivnění odlehlými hodnotami. Výpočet je nejsnazší provést z knihovny scikit-learn pomocí funkce `StandardScaler` [13]`.`

$$x' = \frac{x - \bar{x}}{\sigma} \tag{3.2}$$

4. Feature selection

Stále zvětšující se množství vysokodimenzionálních datasetů, obsahující data o mnoha měřených veličinách, je pro většinu modelů strojového učení problematické. Příliš vysoké množství vstupních parametrů totiž výrazně snižuje nejen jejich efektivitu, ale může mít i výrazný dopad na přesnost a kvalitu trénování [14]. Metody, které snižují dimenze rozsáhlých datasetů, se obecně nazývají jako příznakové inženýrství, v angličtině nejčastěji jako feature selection. Jedná se o zásadní část předzpracování dat, při kterém odstraňujeme nejméně relevantní vstupní veličiny, čímž výrazně zpřehledňujeme práci s nimi. Obecně existují tři přístupy pro výběr rysů [15]:

- *Filtrační metody (Filter methods)*

Tato metoda výběru rysů je na založena na principu srovnání veličin na základě daných kritérií a jejich následném roztřídění. Mezi výhody patří nízká výpočetní náročnost, problematické je ovšem nalezení vztahu v případě vzájemně se ovlivňujících dvou veličin. Příkladem filtračních metod příznakového učení jsou korelace či vzájemná informace.

- *Obalovací metody (Wrapper methods)*

V případě obalovacích metod příznakového učení probíhá hodnocení jednotlivých veličin na základě určitého algoritmu strojového učení. Mezi nevýhody patří vysoká výpočetní náročnost pro rozsáhlé datasety a náchylnost k over-fittingu [16].

- *Vložené metody (Embedded methods)*

Embedded metody kombinují přístupy obalových a filtračních metod příznakového inženýrství. Výběr rysů probíhá při každé iteraci tvorby modelu. Výpočet může probíhat pomocí regulátorů. Konkrétně L1 regulátoru (Lasso regrese), L2 regulátoru (Ridge regrese) či elastic net regulátoru. Další možností embedded metod jsou rozhodovací stromy a jejich odnože (AdaBoost, Gradient boosting)

Dalším důvodem pro aplikaci feature selection na rozsáhlé datasety je eliminace rizika tzv. curse of dimensionality [17]. Tento problém má velmi podobný základ jako problém vyhasínání gradientu v neuronových sítích, kdy z důvodu velkého množství spojů mezi neurony optimalizační algoritmus postupně snižuje svůj vliv, čímž nedochází ke konvergenci přes aktivační funkce ve vzdálenějších vrstvách.

Vzhledem k vysokému množství vstupních parametrů (478) měřených na lince, z nichž velká část ovlivňuje sledovanou hodnotu volumetriky buď velmi málo, případně vůbec, je třeba i v tomto případě vybrat proměnné s největším vlivem na výstupní hodnotu. V mém případě jsem pro redukci velmi rozsáhlého datasetu zvolil následující čtyři metody [17]:

- Korelaci
- Metodu vycházející z regresních stromů AdaBoost
- Vzájemnou informaci (Mutual information)
- Algoritmus Boruta

Pomocí každé metody jsem získal informaci o několika nejvlivnějších veličinách, které jsem poté ručně po konzultacích vybral do prediktivního modelu. Nespoléhat se pouze na číselné výstupy, ale zapojit i ruční výběr parametrů je důležité, jelikož vliv mohou mít proměnné z mnoha důvodů a jen některé mohou být na strojích upravitelné.

4.1. Korelace

Korelace slouží k popsání lineárních závislostí mezi jednotlivými veličinami. Princip spočívá ve vytvoření lineární funkce nejlépe zapadající mezi hodnoty dvou proměnných. Míra korelace je vyjádřena koeficientem vypočítaným ze vzdáleností naměřených parametrů od očekávaných hodnot této linie a nabývá hodnot v intervalu -1 až +1. Hodnota +1 představuje přímou závislost, -1 naopak závislost nepřímou. Pokud koeficient vychází 0, mezi veličinami není zjištěna žádná lineární závislost, což ovšem nevylučuje jiné možné závislosti. Nejrozšířenějším koeficientem (zároveň také defaultní parametr vstupu *method* funkce `corr()` v pandas pro výpočet korelací v dataframu) je Pearsonův korelační koeficient. Ten se také označuje jako standardní korelační koeficient a je definovaný vztahem:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y} \tag{4.1}$$

Kde jednotlivé veličiny představují:

cov	Kovariance
σ_X	Směrodatná odchylka X
σ_y	Směrodatná odchylka Y
$E(X,Y)$	Naměřené hodnoty
μ_X, μ_y	Očekávané hodnoty

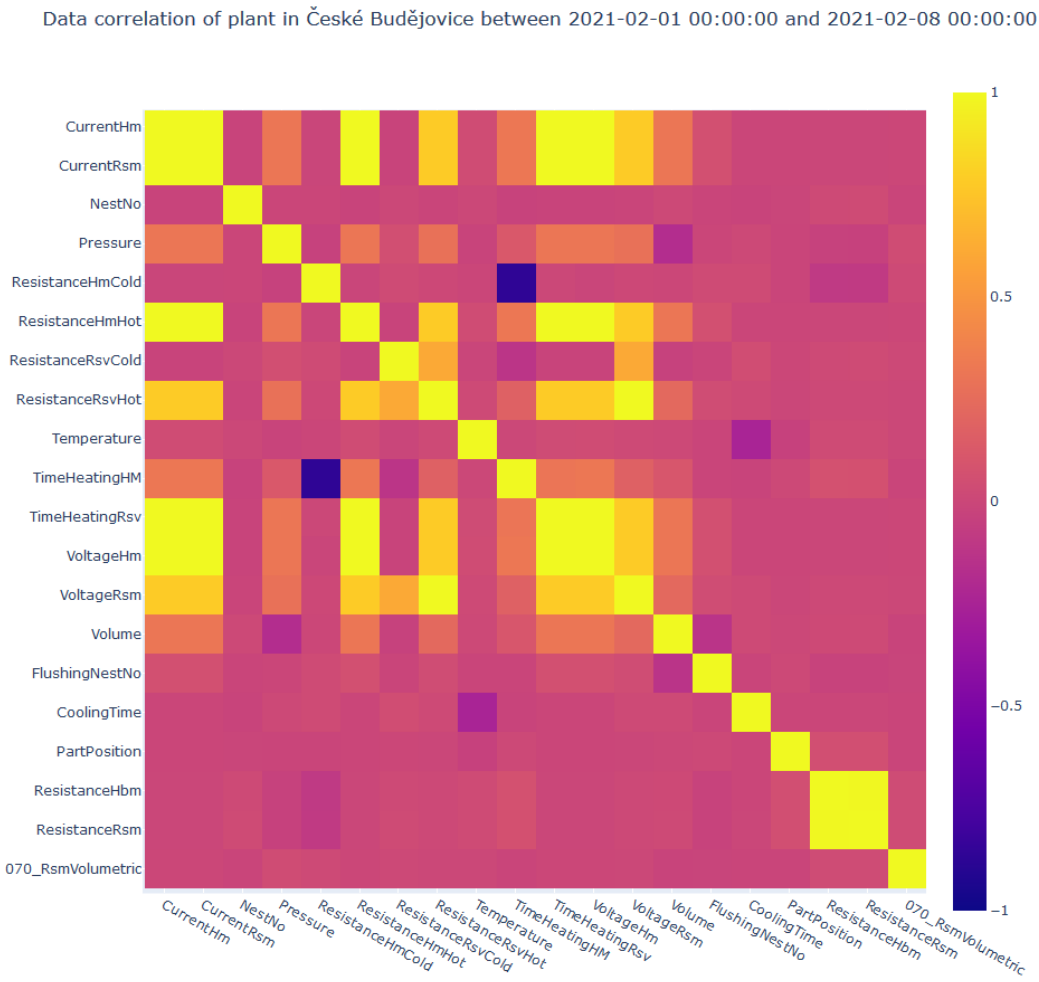
Kovariance vyjadřuje míru lineární závislosti mezi dvěma veličinami. Vypočítá se jako střední hodnota všech odchylek od očekávaných hodnot. Kovariance nabývá hodnot na intervalu $\langle -\infty, +\infty \rangle$, a tak pro snadnější intepretaci se normalizuje podělením směrodatnými odchylkami na korelaci.

Pro výpočet korelací jsem vytvořil jednoduchou aplikaci s GUI z knihovny tkinter, která je předinstalovaná v Pythonu. Člověk pomocí ní může zadat různé parametry dat do skriptu pro výpočet korelací. Kromě zadání časového rozsahu dat pro analýzu může člověk také zadat, zda se mají načíst data z výrobních linek pouze v Českých Budějovicích, Norimberku nebo z továren v obou městech.

Hodnoty korelačního koeficientu mezi všemi proměnnými na vstupu počítá funkce `corr()` v knihovně pandas, výstupem této funkce matice s rozměrem nxn, kde n je celkový počet všech korelovaných veličin.

```
X = X.corr(method='pearson')
```

Vzhledem k vysokému množství vstupních veličin jsem pro vizualizaci výsledků zvolil grafy vytvořené v knihovně *plotly*. Ty na rozdíl od známějšího modulu pyplot od matplotlibu vytváří interaktivní výstupy vytvořené v javascriptu. Heatmapy s hodnotami korelačních koeficientů se tím pádem chovají jako webová aplikace a lze je snadno ovládat. Hlavní výhodou v mém případě je možnost přibližovat a najet šipkou nad jednotlivá políčka a tím snadno získat informace, což ve statickém obrázku z matplotlibu, z důvodu vysokého rozsahu datasetu, nelze.



Obr. 5:Grafický výstup vzájemných korelací veličin z knihovny Plotly

Výsledné hodnoty korelace naměřených veličin výrobních parametrů na linkách strojů s hodnotami parametru zpětné volumetriky z funkční zkoušky jsem poté zaznamenával týden po týdnu. Cílem bylo zjistit nejkorelovanější parametry výroby rozložené v čase a zjistit případné změny v průbězích několika týdnů. Nebyly zaznamenány žádné změny v průběhu roku 2021 a veličiny mezi sebou mají velmi slabé, až takřka žádné, korelace. Pro ilustraci zde v tabulkách uvádím výstup hodnot korelací pro dva týdny.

Parametr	Korelační koeficient
11. - 18. 1. 2021	
KeyenceTotalCountEddDpd	0,039287
KeyenceTotalCountGeoPaste	0,039287
Torgue2	0,036897
Angle2	0,036886
CycleTimeAP100	0,033333
CoolingDistancePosA	0,032704
FlushingNestNo	0,032484
LasconG1ProcessNo	0,030722
LasconG2ProcessNo	0,030722
AlarmPosA	0,029653

Parametr	Korelační koeficient
18. - 25. 1. 2021	
DatePosB	0,070157
LasconG2ProcessNo	0,069025
LasconG1ProcessNo	0,068214
final_force	0,064083
DatePosC	0,062854
MaxWeldingPerformancePosC	0,055675
final_stroke	0,053729
WeldingEnergyPosC	0,051638
VbgCavity	0,051064
CaulkingForce	0,049188

Tabulka 1: Nejvíce korelované parametry se zpětnou volumetrikou za různá období

Vzhledem k téměř žádné existující korelaci jsem se rozhodl hledat důležité parametry pomocí dalších algoritmů.

4.2. AdaBoost

Knihovna *Scikit-learn* nabízí mnoho algoritmů strojového učení, které při svém trénování kombinují více výpočetních postupů klasifikace či regrese pomocí rozhodovacích stromů (Ensemble learning). Většina z nich obsahuje atribut `feature_importances_`, který vrací hodnotu tzv. Giniho koeficientu jednotlivých vstupních proměnných. Jedná se o hodnotu celkového vlivu proměnné na výslednou veličinu normalizovanou mezi 0 a 1.

Pro výpočet jsem zvolil z této skupiny algoritmus AdaBoost, jehož jméno je vytvořeno ze zkratky slov Adaptive Boosting. Patří do podskupiny algoritmů Gradient Boosting. Základní princip spočívá ve vytvoření velkého množství tzv. decision trees, neboli rozhodovacích stromů, čímž je velmi podobný Random Forest, ze kterého vychází. Zatímco u Random Forest se vytvoří paralelně mnoho nezávislých rozhodovacích stromů, u nichž nezáleží na pořadí, u AdaBoost se stromy vytváří ve specifickém pořadí a jejich parametry postupně jsou upravovány pomocí vah [18]. Tyto váhy jsou při trénování updatovány podobně jako u běžné neuronové sítě. Stromy v algoritmu AdaBoost jsou výrazně mělčí než v běžném Random Forest, nejčastěji obsahují pouze jednu rozhodovací podmínku. V prvním kroku dojde k přiřazení vah ke každému stromu, která je rovna $w_i = 1/N$, kde N je rovno počtu řádků a součet všech významů tak vychází vždy 1. Kvalita predikce jednotlivého stromu se vypočítá podle vzorce

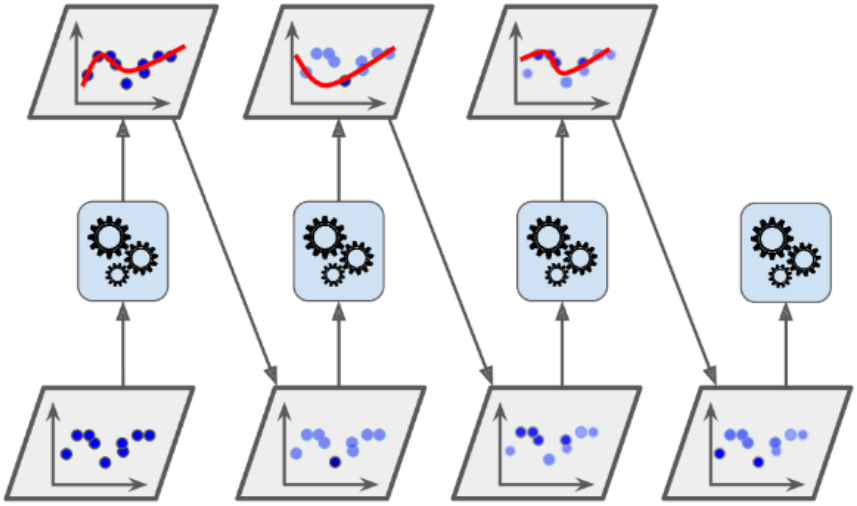
$$\alpha_m = \frac{1}{2}\ln\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right), \tag{4.2}$$

kde α_m je kvalita predikce a ε_m představuje relativní chybu predikce.

Váhy nejslabšího klasifikátoru jsou updatovány pomocí vzorce

$$w_{i+1} = w_i \cdot e^{-\alpha_m}. \tag{4.3}$$

Ostatním stromům je poté přiřazena hodnota vah tak, aby byl celkový součet vah opět roven 1. Po přiřazení dojde k vytvoření nového datasetu na základě změněných rozhodovacích stromů. Takto probíhá proces v několika iteracích [19].



Obr. 6: Algoritmus AdaBoost s postupnou úpravou vah [13]

4.2.1. Giniho koeficent

Použitý parametr charakterizující důležitost jednotlivých veličin v datasetu je již zmíněný Giniho koeficient. Nejznámější využití nachází v ekonomii, kdy vyčísluje míru distribuce majetku ve společnosti. V oblasti strojového učení se v ensemble metodách pomocí něho budují stromy s nejlepší architekturou jednotlivých větví [20]. Při rozhodování se totiž spočítá distribuce správných a chybných predikcí p_1 *a* p_0 v jednotlivém rozhodovacím uzlu podle vzorce

$$i(k) = 1 - p_1^2 - p_0^2. \tag{4.4}$$

Výpočet změny přesnosti pravé a levé větve se řídí vzorcem

$$\Delta i(k) = i(k) - p_l \cdot i(k_l) - p_r \cdot i(k_r). \tag{4.5}$$

Samotný Giniho koeficient důležitosti vztažený ke konkrétní veličině $I_G(v)$ se spočítá jako součet všech výskytů tohoto parametru ve všech uzlech k ve všech rozhodovacích stromech K v náhodném lese [21] následovně

$$I_G(\mathrm{v}) = \sum_{k} \sum_{K} \Delta i_{\mathrm{v}}(k, K). \tag{4.6}$$

Prvním krokem při samotném využití pro zjištění důležitosti parametrů v praxi je třeba natrénovat regresní model z jakékoliv skupiny ensemble. Pro výpočet se v Pythonu nejprve musí importovat příslušná třída z knihovny Scikit-learn a uložit do proměnné. Model se dá upravovat pomocí tzv. hyperparametrů, v tomto případě lze nastavit ztrátovou (optimalizační) funkci, learning rate či počet rozhodovacích stromů.

```
from sklearn.ensemble import AdaBoostRegressor

model = AdaBoostRegressor(n_estimators=50, learning_rate=0.5)
```

Tato obsahuje mnoho metod, hlavní představuje funkce `fit,` která vytvoří ze vstupních parametrů vytvoří požadovaný regresní model.

```
model.fit(X, y)
importance = model.feature_importances_
```

Po vytvoření regresoru bude vytvořen atribut `feature_selection_`, jehož výstupem je číslo ve formátu float. Vytvořil jsem tedy smyčku, která ukládá do jednoho sloupce dataframu názvy veličin a do druhého její hodnotu důležitosti z funkce rozhodovacích stromů. Poté jsem hodnoty seřadil sestupně od nejvyšší po nejnižší a uložil jako excel pro další zpracování.

Parametr	Feature importance
LasconProcessNo	0,406835
FillPressureM52	0,122409
FilterPressingPath	0,110862
ToolDataPressWp102ToolCounter	0,103416
LasconG1Pavg	0,094113
LasconG1ProcessNo	0,092419
final_force_DPD	0,090775
FilterPressingForce	0,08891
final_stroke_EDD	0,066824

Tabulka 2: Hodnoty důležitosti veličin z AdaBoost algoritmu

Hodnoty feature importace se při změně hyperparametrů modelu výrazně změní z důvodu, že jsou založeny na regresi z rozhodovacích stromů, které jsou na vstupních parametrech velmi závislé. I při různých hodnotách veličin se ovšem na prvních místech vyskytují stále stejné veličiny, které souvisí s měřením naší sledované zpětné volumetriky. Lze také využít funkce pro permutaci parametrů modelu, kdy byly zjištěny podobné parametry s odpovídajícími hodnotami.

```
result = permutation_importance(model, X, y, n_repeats=10,
                                random_state=0)
```

4.3. Vzájemná informace (Mutual Information)

Vzájemná informace je parametr vyjadřující vzájemnou závislost dvou veličin na základě neurčitosti v datech vyjádřený jednotkou *bity*. Parametr byl matematicky popsán Claudem Shannonem v roce 1948, jako termín byl ovšem zaveden později [22]. V našem případě, pro diskrétní veličiny, je popsána vzorcem:

$$I(X;Y) = \sum_{y\in Y}\sum_{x\in X} p_{(X,Y)}(x,\ y)\ \log\left(\frac{p_{(X,Y)}(x,\ y)}{p_X(x)\,p_Y(y)}\right) \tag{4.7}$$

$p(x,y)$ sdružená pravděpodobnostní funkce pro vstupní veličinu X a výstupní Y

$p(x)$ marginální pravděpodobnost pro vstupní veličinu X

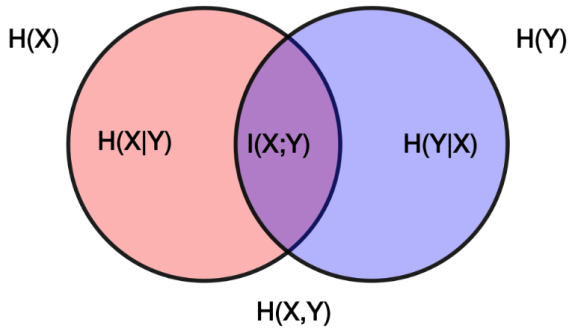
$p(y)$ marginální pravděpodobnost pro výstupní veličinu Y

Princip výpočtu vzájemné informace vychází jako kombinace míry entropií jednotlivých veličin. Entropie v datech popisuje náhodnost veličiny, čím vyšší má hodnotu, tím je veličina chaotičtější, podobně jako u termodynamických systémů. Je definována vztahem:

$$H(Y) = -\sum_{y\in Y}(y)logP(y) \tag{4.8}$$

Entropie úzce souvisí s hodnotami pravděpodobnostních funkcí. Ty udávají, s jakou pravděpodobností se trefí náhodná diskrétní hodnota do jiné diskrétní hodnoty v určeném rozsahu. Například u tzv. poctivé hrací kostky jsou pravděpodobnostní funkce pro každou z možností hozených čísel 1/6. V tomto případě, kdy se jedná o čistě stochastický jev, je podíl sdružené a marginální pravděpodobností roven 1. Vzájemná

informace veličin vychází jako logaritmus podílu, který v tomto případě odpovídá hodnotě 0. Mezi jednotlivými veličinami (hody kostkou) tedy neexistuje žádná spojitost.



Obr. 7: Veenův diagram popisující vztah mezi informačními veličinami [23]

Na obrázku vidíme Veenův diagram popisující vztahy mezi informačními veličinami. Pole H(X) a H(Y) představují entropie veličin. Podmíněné entropie H(X|Y) a H(Y|X) vyčíslují, jaké je třeba množství vstupních dat proměnné X pro popsání výstupní proměnné Y a naopak. Má stejné jednotky jako vzájemná informace. Ta se označuje jako $I(X;Y)$. Vztah mezi entropií a podmíněnou entropií je definován jako rozdíl těchto veličin:

$$I(X;Y) = H(Y) - H(Y \mid X) \tag{4.9}$$

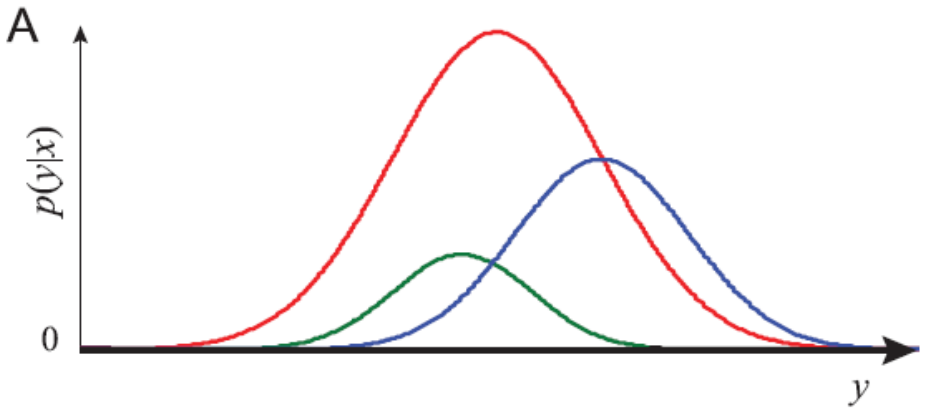
Mezi výhody vzájemné informace jsou uváděny vlastnosti [24]:

- Snadná aplikace a vyhodnocení
- Relativně rychlý výpočet
- Odolný proti přetrénování (overfitting)
- Podrobný teoretický základ
- Zaznamená všechny vztahy mezi veličinami, nejen lineární vztah jako u korelací

Pro výpočet hodnot vzájemné informace jsem opět zvolil funkci dostupnou v knihovně scikit-learn ve skupině feature selection:

```
from sklearn.feature_selection import mutual_info_regression
mi_scores = mutual_info_regression(X, y)
```

Výpočet vzájemné informace v této funkci je založen na algoritmu k-nejbližších sousedů (zkratka k-NN z anglického k-nearest neighbors). Princip, jak funguje, je popsán níže dle zdroje [25].

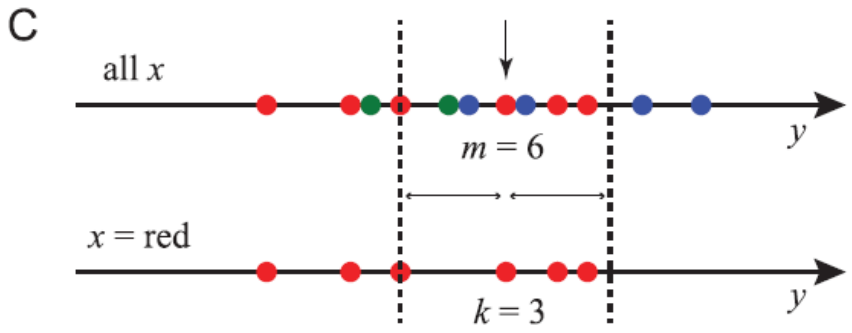


Obr. 8: Příklad rozdělení veličin dle hustoty pravděpodobnosti [25]

Prvním krokem tohoto algoritmu je dle množství hodnot rozložit pravděpodobnost výskytu veličiny X v závislosti na y, tedy sdružené pravděpodobnostní rozdělení. Plocha této křivky představuje marginální pravděpodobnost veličiny X. Na základě zadaného parametru `n_neighbors` rozdělíme do definovaného počet bodů na osu y. Množství jednotlivých bodů odpovídá marginálním pravděpodobnostem veličin X. Tento argument je přijímán funkcí `mutual_info_score,` pracuje s ním však až dle kódu dostupného na githubu [26] funkce `NearestNeighbors` dle výstupu z fukce `KDTree,` která vypočítá ideální hodnotu *k*. V uvedeném ilustračním případě počítáme s počtem sousedů N_x rovným 12 kusů.

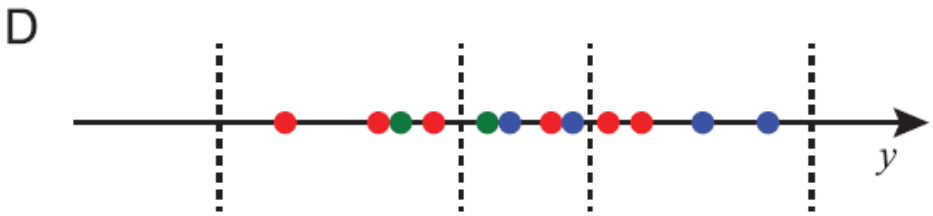
Obr. 9: Rozdělení bodů po ose y dle marginálních pravděpodobností veličin X [25]

Proměnná *k* je vypočtené celé číslo, které definuje, kolikátý sousední bod bude určujícím pro výpočet velikosti *d*, tedy rozměr jednoho políčka. Tento parametr výrazně ovlivňuje kvalitu výsledku, funkce `KDTree,` proto počítá ideální hodnotu na základě úpravy vah.



Obr. 10: Výpočet rozměru políčka d dle počtu nejbližších sousedů k [25]

Vzájemná informace se poté vypočítá na základě množství bodů stejné barvy v jednom konkrétním políčku. Tento proces se poté provede přes všechny hodnoty v datasetu.



Obr. 11: Vzájemná informace vypočítaná z počtu bodů stejné barvy v každém políčku [25]

Tento postup výpočtu přes clusterovací algoritmus K-NN je dle [27] nejrychlejším postupem pro přibližný výpočet vzájemné informace veličin.

Výstupní hodnoty vzájemné informace

Výstupem je NumPy pole s názvem parametru v prvním sloupci a hodnotou výstupní veličiny v druhém:

Parametr	MI hodnota
ToolDataPressWp102ToolCounter	1,550905
ToolDataPressWp102ToolCounter	1,550571
FilterPressingPath	1,462012
LasconProcessNo	1,451105
LasconG1ProcessNo	1,33509
LasconG2ProcessNo	1,335087
FilterPressingForce	1,331365
itemPressInForce3	1,330744
CaulkingForce	1,320949

Tabulka 3: Výstupní hodnoty vzájemné informace veličin

Výsledné hodnoty vzájemné informace již poskytují výrazně relevantnější výsledky. Podle návodu [28] na interpretaci hodnot se jedná o silný vztah mezi veličinami.

4.4. Boruta

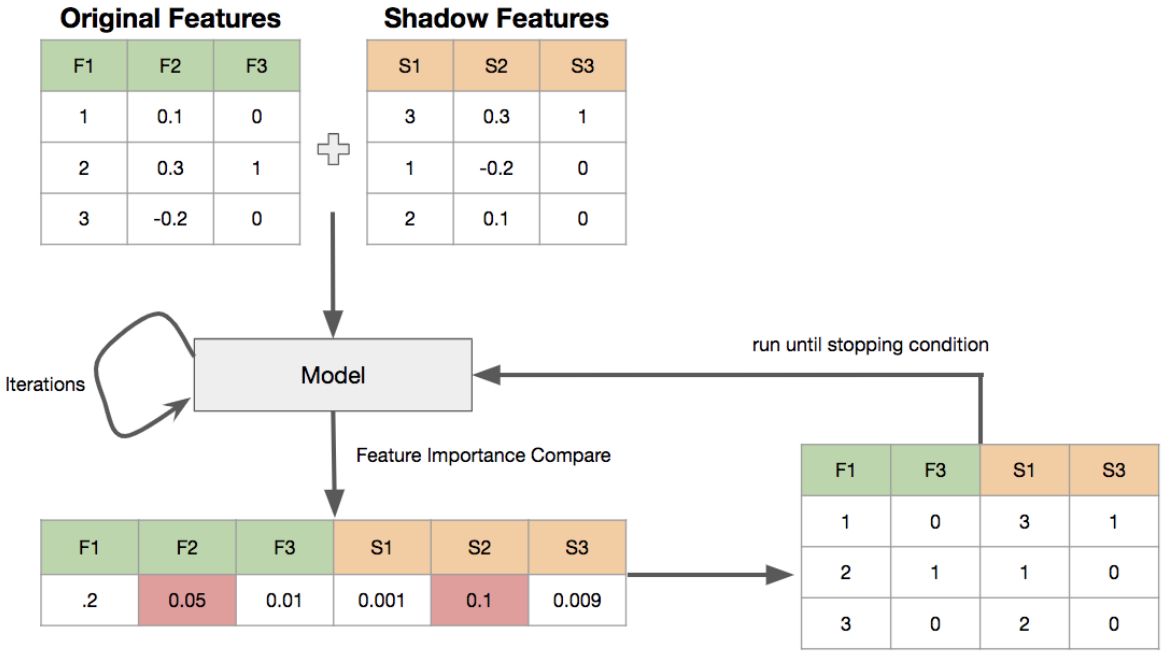
Algoritmus Boruta je založen na výstupu hodnot Giniho koeficientu vyjadřující důležitosti parametrů vstupu X a výstupu Y. Je vypočítán stejnou skupinou algoritmů „ensemble“ jako je například popsaný AdaBoost nebo Random Forest. Vypočítané koeficienty se poté srovnávají s výsledky náhodně resamplovaných datasetů. Takovýto přístup k testování se ve statistice obecně nazývá *Bootstrapping*.

Prvním krokem pro výpočet podobnosti spočívá ve vytvoření stínové vstupní matice X se stejnými veličinami jako v originálním datasetu, ovšem s náhodně promíchanými hodnotami. Předpokládá se, že pokud původní veličina dokáže dosáhnout lepších výsledků feature importance než identická, avšak s náhodně vygenerovanými hodnotami, její vliv ovlivňovat výstup by měl být relevantní.

Jedná se ovšem stále o náhodný výběr, který nemusí být relevantní. Druhým krokem algoritmu, řešící tento problém, je iterace ve smyčce. To znamená, že tento výpočet několikrát zopakujeme a při každém zaznamenání větší hodnoty relevance

původní veličiny než náhodně promíchané, připočteme této proměnné jeden bod. Tím postupně získáme větší jistotu o vlivu vstupních parametrů na výstupu.

Pro výpočet vlivu veličiny jsem nejprve vytvořil dataframe *X_shadow*, na který jsem aplikoval funkci z knihovny numpy, konkrétně náhodnou permutaci zavolanou příkazem `np.random.permutation`. Poté jsem pomocí algoritmu Random Forest spočítal hodnotu `feature_importances_` stejným způsobem jako u algoritmu AdaBoost. Tyto výsledky jsem poté uložil do dvou polí a porovnal mezi sebou. Pokud byla hodnota důležitosti parametru vyšší než výsledek náhodně promíchaného parametru, přičetl se do dataframu `hits_pd` jeden bod.



Obr. 12: Princip algoritmu Boruta [29]

Tento algoritmus také nabízí knihovna BorutaPy, do které lze zadat jako parametr natrénovaný model a počet iterací. Má známé uživatelské prostředí díky stejným příkazům jako scikit-learn, na jehož základě běží. Jak je zřejmé z jeho repozitáře na Githubu, nabízí pokročilejší kód a lze jej použít jako black box.

Pro výpočet jsem ovšem zvolil výše zmíněný postup. Ten jsem opakoval desetkrát ve smyčce for. Výpočet skóre Boruty je velmi časově náročný, proto jsem zvolil poměrně malý počet iterací. Původně jsem nechal pomocí BorutaPy opakovat algoritmus stokrát, avšak po 14 hodinách spuštěného kódu přes noc stále nedošlo k dopočítání skóre. Snížil jsem tedy náročnost výpočtu a pracoval pouze se smyčkou v rozsahu deseti iterací. Výpočet trval již přijatelné 2 hodiny. Poté jsem se zaměřil na výpočet skóre z různých

časových období. I tak jsem viditelně oddělil vlivné veličiny od těch bezvýznamných. Důležité veličiny měly většinou stabilně vysoké skóre. Výstupy jsem z dataframu v pandas uložil do excelové tabulky pro snadnější práci v budoucnu, konkrétně pro porovnání s hodnotami veličin z ostatních algoritmů.

Výstupní hodnoty parametrů algoritmu boruta:

Parametr	boruta skóre
LasconProcessNo	10
LasconG1ProcessNo	10
ToolDataPressWp102ToolCounter	10
LasconG1Tmin	10
final_force_DPD	10
ToolDataPressWp102ToolCounter	10
KeyenceTotalCountEddDpd	10
KeyenceTotalCountGeoPaste	10
LasconG2ProcessNo	10
SensorPressingForce	9
FillPressureM50	8
final_stroke_EDD	8
Volume	4
CaulkingPath	4

Tabulka 4: Hodnoty skóre jednotlivých veličin z algoritmu Boruta

Skóre jednotlivých veličin vychází poměrně jednoznačně. Ne všechny parametry ovšem byly použity pro predikci výsledků zkoušek. Nejasný je pro mě důvod vysokého umístění číselně kategorického parametru LasconProcessNo či KeyenceTotalCountEddDpd. Vyskytují se sice v oblasti modulu zpětné pumpy, nenachází se ovšem v určitém číselném rozsahu a do konečné aplikace by se nehodily. Jiné parametry, například FillPressureM50 sice mají přímou konstrukční závislost mezi parametrem zpětné volumetriky, jedná se ovšem spíše o důsledek změny než o příčinu. Rozhodl jsem se proto z tohoto důvodu vybrat pouze nějaké parametry zvýrazněné tučně.

Možností pro zlepšení algoritmu v mém případě je vytvořit smyčku, která by trénovala algoritmus boruta po malých dávkách ve velkém datasetu a srovnávala by výsledky v čase.

4.5. Zhodnocení výsledků feature importance

Po získání nejvlivnějších parametrů z každého výše uvedeného algoritmu jsem výstupy zkombinoval. Kromě korelací se ve všech třech parametrech vyskytovaly na čelních pozicích stejné parametry. Z nich jsem provedl konzultaci s technology, zda je jejich ovlivnění zpětné volumetriky možné. Jedná se například o konstrukční uspořádání v rámci supply modulu.

	A	B
1		MI Scores
2	ToolDataPressWp102ToolCounter	1,550905425
3	ToolDataPressWp102ToolCounter	1,550570718
4	FilterPressingPath	1,462011638
5	LasconProcessNo	1,451105156
6	LasconG1ProcessNo	1,335089769
7	LasconG2ProcessNo	1,335087037
8	FilterPressingForce	1,331365324
9	itemPressInForce3	1,3307442
10	CaulkingForce	1,320948956
11	Volume	1,317307355
12	SensorPressingForce	1,309491413
13	CaulkingPath	1,307320009
14	itemPressInForce3	1,306150656
15	itemPressInForce1	1,303575181
16	TriggerPosition	1,297953031
17	final_force_DPD	1,296471784
18	itemPressInForce1	1,295922189
19	Background	1,277116243
20	final_force_EDD	1,27271006
21	AvgCurrentHeating2	1,267732595
22	AvgCurrentHeating3	1,266122266
23	Leakrate	1,262049216
24	AvgCurrentHeating1	1,260239358

	A	B
1		RF importance
2	LasconG1ProcessNo	0,24782
3	LasconProcessNo	0,186131
4	FillPressureM52	0,118684
5	ToolDataPressWp102ToolCounter	0,051825
6	SensorPressingPath	0,04223
7	LasconG1Pavg	0,035488
8	KeyenceTotalCountEddDpd	0,034832
9	FilterPressingForce	0,028872
10	FilterPressingPath	0,026936
11	ToolDataPressWp102ToolCounter	0,02343
12	HeightDpd	0,021251
13	LasconG2ProcessNo	0,020109
14	FillPressureM50	0,020012
15	final_stroke_EDD	0,018496
16	LasconG1Tmin	0,0167
17	FillPressureM51	0,016279
18	ResistanceHbm	0,012964
19	FinalForce	0,012146
20	LasconG1Tmin	0,010795
21	SensorPressingForce	0,008637
22	CycleTime	0,007224
23	LasconG1Tavg	0,007036
24	CaulkingPath	0,006725

	A	B
1		boruta hits
2	LasconProcessNo	10
3	LasconG1ProcessNo	10
4	ToolDataPressWp102ToolCounter	10
5	LasconG1Tmin	10
6	final_force_DPD	10
7	ToolDataPressWp102ToolCounter	10
8	KeyenceTotalCountEddDpd	10
9	KeyenceTotalCountGeoPaste	10
10	LasconG2ProcessNo	10
11	SensorPressingForce	9
12	FillPressureM50	8
13	final_stroke_EDD	8
14	Volume	4
15	CaulkingPath	4
16	LasconG1Pavg	2
17	CamAnkerScore_Multiple	1
18	ResistanceHbm	1
19	LasconG1Tmax	1
20	KistlerProgNoPressureD	1
21	KistlerProgNoIceD	1
22	SensorPressingPath	1
23	NIO_BITS	0
24	PROCESS_NUMBER	0

Obr. 13: Veličiny z výstupů jednotlivých algoritmů a jejich hodnoty

Tyto naměřené parametry mají přímý vliv na výkon zpětné pumpy a objem vrácené kapaliny, jelikož se jedná o sílu a hloubku zalisování zpětného magnetu do plastového šasi:

- *final_force_DPD*
- *final_force_EDD*
- *final_stroke_DPD*
- *final_stroke_EDD*

Kromě těchto přímo souvisejících veličin jsem vybral i další tyto 4 následující:

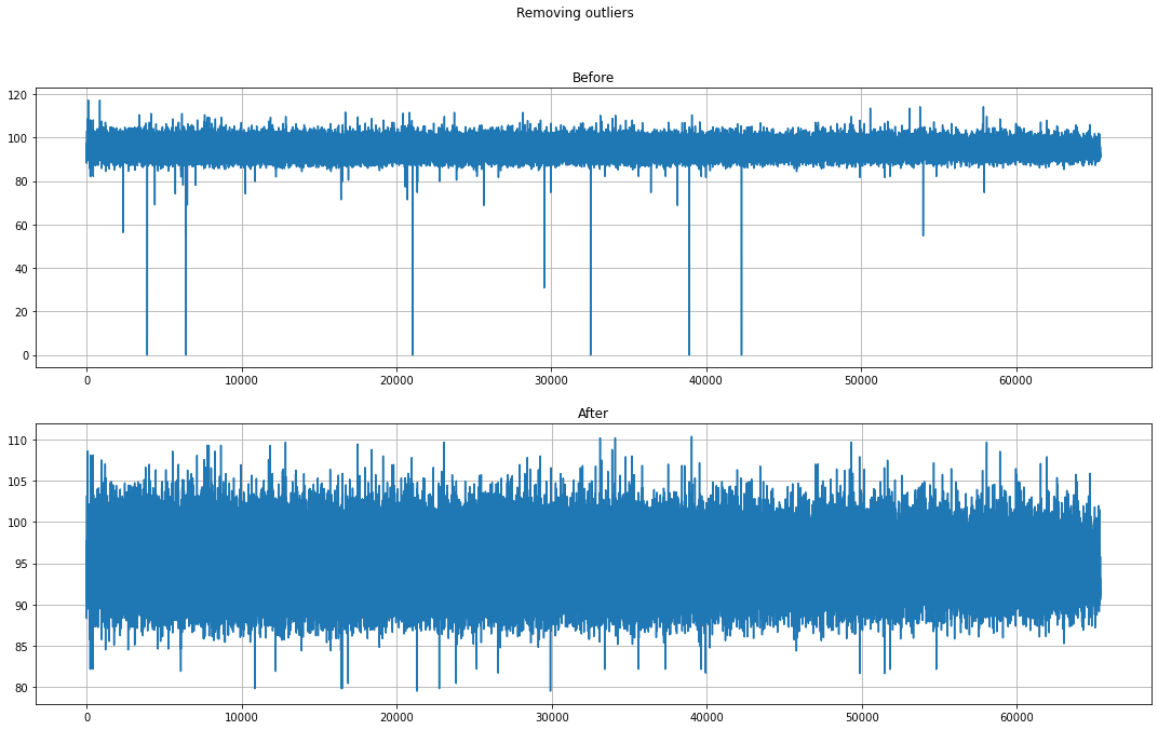
- *FilterPressingPath*
- *FilterPressingForce*
- *SensorPressingForce*
- *SensorPressingPath*

Parametry *FilterPressingPath* a *FilterPressingForce* popisují, jak velkou silou a jak hluboko je zalisován filtr. Analogická je situace u *SensorPressingPath* a *SensorPressingForce*. Tyto hodnoty nemají přímou souvislost s modulem zpětné pumpy, mohou výkon ovšem ovlivňovat nepřímo z konstrukčního hlediska. Přítlak součástky může ovlivnit postavení ostatních součástek a je potřebné i tento parametr analyzovat.

4.6. Odstranění odlehlých hodnot

Pro správnou funkčnost predikce je třeba získat přehled o rozsahu nejčastějších hodnot veličiny. Jde o to, aby byly do modelu zadávány relevantní hodnoty, na což běžné zjištění minimálních a maximálních hodnot není vhodné z důvodu možnosti přítomnosti extrémně vysokých či nulových hodnot. Jednou z možností odstranění odlehlých hodnot je funkce v knihovně Scikit-learn `LocalOutlierFactor()`, která nabízí mnoho algoritmů v principu podobných clusterovacím algoritmům jako například *K-means* [13].

Další možnosti se nabízí nevyužít již vytvořené knihovny a odstranit odchylky ručně. To v mém případě představuje lepší přístup, jelikož se hodnoty parametrů přibližně pohybují kolem určité hodnoty a výstupní rozsah hodnot poté můžu stanovit sám. Pro odstranění odlehlých hodnot jsem tak zvolil metodu ve vytvoření pásma definovaného násobkem směrodatné odchylky od průměru. Metoda jako taková je snadno aplikovatelná. Její princip spočívá ve vytvoření proměnné, která se rovná násobku směrodatné odchylky od průměru. Poté se vytvoří rozhodovací podmínku [30]. Pokud je tato hodnota vyšší či menší než naše požadovaná, pak je hodnota přidána do nového listu. Ten je již bez těchto odlehlých hodnot. V mém případě jsem volil většinou čtyřnásobek, někdy trojnásobek směrodatné odchylky.



Obr. 14: Odstranění odlehlých hodnot

Po odstranění odlehlých hodnot pomocí výše popsaného postupu jsem ručně definoval rozsah hodnot, které bude uživatel vkládat do prediktivního modelu pomocí aplikace. Spodní a horní hodnoty jsou uvedeny v následující tabulce. Kromě hranic vstupních parametrů je také uveden počet odlehlých hodnot v původním datasetu a směrodatná odchylka, podle které se extrémy odstraňovaly.

Parametr	Od	Do	Počet outliers	Směrodatná odch.
FilterPressingPath	14	14,6	8	4
FilterPressingForce	80	115	52	4
SensorPressingPath	19,4	19,5	354	4
SensorPressingForce	60	165	354	4
final_force_DPD	45	70	259	3
final_stroke_DPD	18,15	18,25	97	3
final_force_EDD	6	22	61	3,50
final_stroke_EDD	17,35	17,45	365	3,5

Tabulka 5: Hodnoty vstupních veličin po odstranění extrémů

5. Prediktivní model

Pro volbu modelu pro predikci parametrů zpětné volumetriky jsem porovnával kvalitu výstupů ze tří různých přístupů:

- Lineární a polynomiální neurony
- Neuronová síť
- Gradient Boosting (XGBoost)

5.1. HONU – Higher Order Neural Unit

Prvním modelem umělé inteligence aplikovaným na data byl neuron s různými stupni vstupních polynomů, a to:

- Prvního stupně – lineární
- Druhého stupně – kvadratický
- Třetího stupně – kubický

Neuron neboli perceptron tvoří základní jednotku dopředných neuronových sítí. Algoritmus je volně inspirovaný biologickým neuronem v tom, že má vícenásobný počet vstupů, avšak vždy jen a pouze jediný výstup. První snahy o vytvoření stroje schopného se učit na tomto principu sahají do 40. let minulého století, funkční softwarové řešení bylo ovšem popsáno až v roce 1986. Ten popisoval princip tzv. backpropagation neboli zpětného šíření chyby pomocí gradientu, čímž byly položeny základy hlubokého učení. Důležité je zmínit, že původní cíle těchto algoritmů měly vést k vytvoření obecné umělé inteligence s cílem vytvořit inteligentní model podobný člověku. Nyní je ovšem prokázáno, že pomocí současných algoritmů není možné vytvořit lidsky uvažující software pouhým zvyšováním počtu neuronových jednotek [31].

Model neuronu je složen z těchto základních jednotek:

- Synaptická část
  - Vstupy
  - Váhy
  - Bias a práh
- Somatická část
  - Aktivační funkce

V synaptické části se násobí vektor vstupních dat do neuronu s vektorem vah, případně se přičítá tzv. bias, který posouvá intenzitu vstupu. Práh naopak upravuje citlivost neuronu. Právě zde na vstupu se liší jednotlivé výše zmíněné neuronové jednotky. Způsob roznásobení těchto vstupních parametrů odpovídá stupni v Taylorově polynomu. Výsledkem vyššího stupně neuronu je lepší aproximace funkce v konkrétním bodě. Roznásobení u lineárních jednotek vypadá následovně:

$$f = \sum_{i=1}^{N} (w_i x_i) \tag{5.1}$$

Roznásobení matic v kvadratickém neuronu:

$$f = \sum_{i=1}^{N} \sum_{j=i}^{N} x_i \cdot x_j \cdot w_{i,j} \tag{5.2}$$

Násobení matic v kubickém neuronu:

$$f = \sum_{i=1}^{N} \sum_{i=j}^{N} \sum_{j=k}^{N} x_i \cdot x_j \cdot x_k \cdot w_{i,j,k} \tag{5.3}$$

x_i, x_j, x_k vstupní hodnoty dat do neuronu

w_i hodnoty vah

S rostoucím stupněm polynomu dochází k nárůstu výpočetní složitosti neuronu z důvodu exponenciálního přibývání výše uvedených výpočetních parametrů.

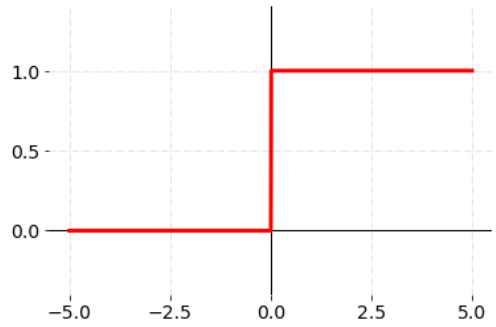
5.1.1. Aktivační funkce

V somatické části neuronu hrají hlavní roli aktivační, často nazývané také jako přenosové funkce. Ty v neuronové jednotce slouží k převedení vstupních hodnot na hodnoty, které jsou praktičtější k přenosu informace. Pro klasifikační neuronové sítě se používají aktivační funkce nabývající hodnot 0 až 1. V našem případě (regresní vícevrstvá neuronová sít) se nejčastěji používají funkce v rozsahu mezi -1 a 1, jelikož poskytují větší interval. Nejčastěji používanými přenosovými funkcemi jsou [32]:

- Kroková
- Lineární
- ReLU (Rectified Linear Unit)
- Logistická
- Hyperbolická

Kroková funkce

Jedná se o nejjednodušší aktivační funkci, která ovšem není příliš vhodná pro využití. Výstupem je binární hodnota 0 nebo 1, což je problematické například v případě klasifikátoru, kdy může být najednou spuštěno více výstupních neuronů.

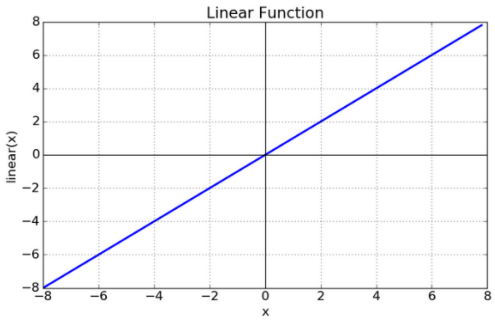


Obr. 15: Kroková aktivační funkce [33]

$$f(x) = 0, x \in (-\infty, 0) \qquad (5.4)$$
$$f(x) = 1, x \in (0, \infty)$$

Lineární funkce

Lineární funkce přináší zlepšení, jelikož jejím výstupem je škála hodnot přímo úměrná součtu vstupů a vah, nikoliv pouze binární rozdělení. Problém nastává při využití gradient descentu při zpětném šíření chyby (backpropagation). Jelikož derivace funkce je rovna 1, hodnota aktivační funkce následující vrstvy je přímo závislá na hodnotě vrstvy předcházející. To znamená, že nedochází ke změně hodnot v neuronech a všechny skryté vrstvy si budou vzájemně odpovídat.

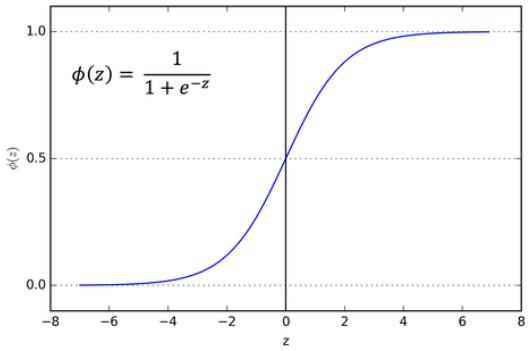


Obr. 16: Lineární aktivační funkce [34]

$$f(x) = x \tag{5.5}$$

Logistická funkce – Sigmoid

Funkce Sigmoid přináší zlepšení především ve své nelinearitě, kdy je možné ji využít u vícevrstvých neuronových sítí. Na intervalu zhruba od -2 do +2 má funkce strmý gradient. To způsobuje, že se funkce snaží nejasný součet vstupních dat a jejich vah kolem nuly rychle převést na výstupní hodnoty v rozsahu 0 a 1. Díky tomuto intervalu je vhodná pro klasifikační úlohy. Může ovšem nastat problém vyhasínání gradientu, kdy nebude docházet k významné změně parametrů mezi jednotlivými vrstvami neuronové sítě.

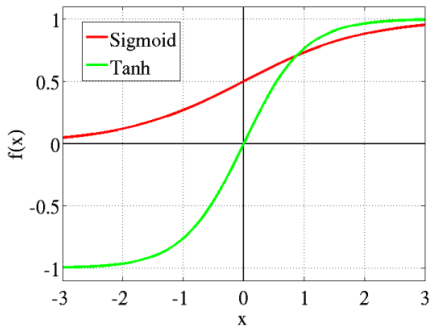


Obr. 17: Logistická aktivační funkce – sigmoid [34]

$$f(x) = \frac{1}{1+e^{-x}} \tag{5.6}$$

Hyperbolická funkce - TanH:

Hyperbolická funkce TanH vypadá velmi podobně jako funkce sigmoid, v nerozhodných místech vstupu kolem nuly má ovšem vyšší derivaci, čímž rychleji rozhoduje o aktivaci neuronu. Je také ohraničena na intervalu od -1 do +1. Vyšší škála hodnot dovoluje vyšší variabilitu možností a je tedy vhodnější na regresní modely. Stejně jako funkce sigmoid je náchylná na problém vyhasínání gradientů.

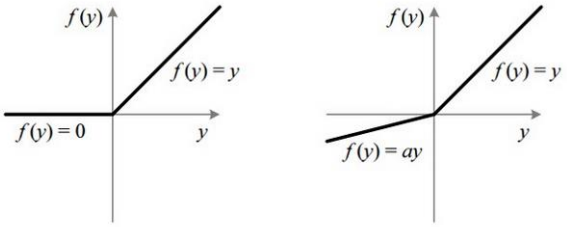


Obr. 18: Porovnání aktivačních funkcí sigmoid a tanh [34]

$$f(x) = \frac{2}{1 + e^{-kx}} - 1 \tag{5.7}$$

Rectified Linear Unit (ReLU):

Funkce ReLU má lineární výstup pro kladný vstup a nulu pro vstup záporný. Funkce je nelineární, je tedy vhodná pro vícevrstvé neuronové sítě. Výhodou ReLU oproti logistické a hyperbolické funkci je, že v rozsáhlé neuronové síti nedochází k úpravě všech neuronů, což je výpočetně náročné. Místo toho se zaktivují pouze nejdůležitější neurony v síti. Funkce je zároveň méně výpočetně náročná, obsahuje totiž jednodušší matematické operace. Nevýhodou je vlastnost zvaná „Dying ReLU Problem“. Ten nastává, pokud začne být mnoho neuronů tzv. mrtvých, tedy že vstupy budou ležet na záporném intervalu. Tím dojde k nenávratnému vypnutí trénování sítě, která se poté začne chovat jako konstantní funkce. Tento problém řeší modifikace zvaná jako Leaky (propustná) ReLU. Ta pro záporné hodnoty vstupů generuje malé záporné výstupy, například $y = 0{,}001x$ pro $x < 0$.



Obr. 19: Aktivační funkce ReLu (vlevo) a její modifikace leaky ReLu (vpravo) [34]

$$f(x) = 0\,, \quad x \in (-\infty, 0) \tag{5.8}$$
$$f(x) = x, \quad x \in (0, \infty)$$

5.1.2. LNU – Linear Neural Unit

Trénování modelu probíhá nejprve součtem násobků všech vstupů v matici X s příslušnými vahami. Tyto váhy jsou zprvu iniciovány náhodně a upraveny pomocí optimalizačních algoritmů vysvětlených v kapitole 5.2.1.

$$\mathrm{y_n(k)} = [\mathrm{w_0} \cdot \mathrm{w_1} \cdot \mathrm{w_2}] \cdot x_i \tag{5.9}$$

Výpočet chyby se vypočítá jako rozdíl mezi predikovanou a skutečnou výstupní hodnotou.

$$e = (y_n - y) \tag{5.10}$$

Tato chyba je poté minimalizována pomocí úpravy vektoru vah optimalizačním algoritmem. O jednotlivých možnostech optimalizačních algoritmů píšu v kapitole 5.2.1. V mém případě jsem váhy upravoval v každém kroku zvlášť podle vzorce:

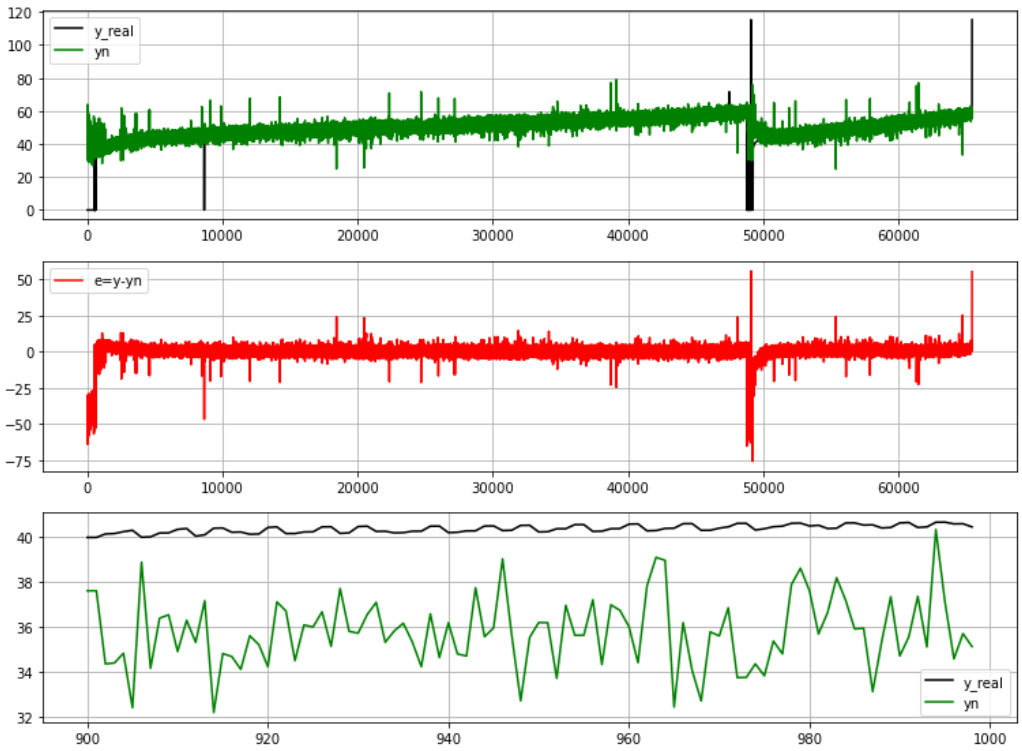
$$\Delta w = -\frac{\mu}{2} \cdot \frac{\partial e_{(k)}^2}{\partial w} = -\mu e x^T \tag{5.11}$$

Výpočet v Pythonu je naprogramován ve dvou smyčkách. Jedna definující počet epoch, tedy kolikrát se neuron na datech trénuje. Druhá iteruje skrz vektor z transponované matice X.

```
for epochs in range(10):
    for k in range(0,N):
        x[1:]=X[k,:]
        yn[k]=np.dot(w,x)
        e[k]=y[k]-yn[k]
        dw=mu*e[k]*x
```

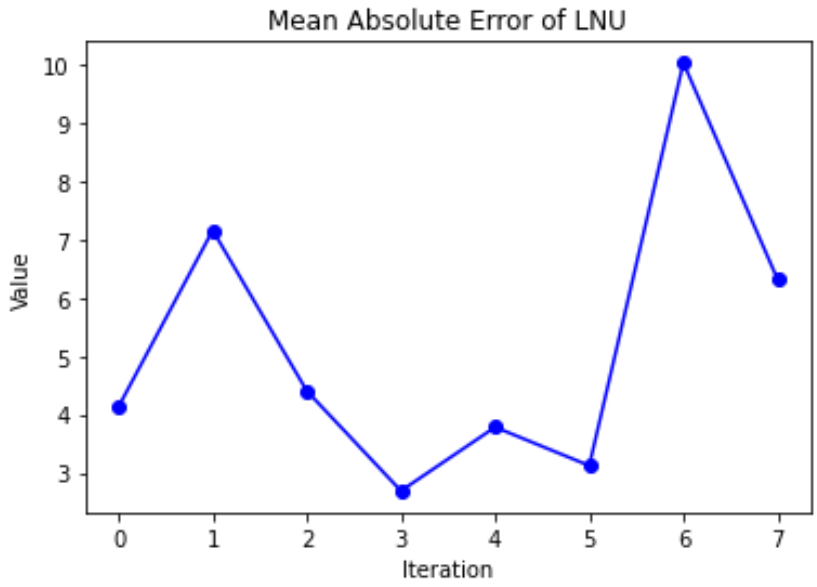
Učení lineárního neuronu bylo na datech po definovaných krocích. Zprvu se na prvním intervalu trénovacích dat neuron natrénoval a poté se na dalším množství dat zjistila kvalita predikce. To probíhalo do doby, než poslední hodnota testovacích dat nepřesáhla délku všech dat.

```
train_size = 12000
test_size = 2500
step = 7000

end_train = start_train + train_size
start_test = end_train
end_test = end_train + test_size

while end_test < len(X):
        …
        start_train += step
```



Obr. 20: Graf vývoje predikce (nahoře), chyby (uprostřed), malé části predikce (dole) pro lineární neuron

Na obrázku je vykreslen graf trénování lineárního neuronu. První graf ukazuje srovnání predikované hodnoty s výstupem neuronu v celém datasetu. Prostřední graf ukazuje hodnoty chyby, tedy rozdílu mezi predikovanou a reálnou hodnotou. Nejníže je vidět vykreslení predikované a reálné veličiny na menším vzorku. Díky tomu můžeme vidět, že reálně naměřené parametry se liší od skutečných hodnot zhruba o 5. To popisuje také hodnota střední absolutní chyby, tedy Mean Absolute Error.



Obr. 21: Vývoj průměrné absolutní chyby při trénování lineárního neuronu

Průměrná absolutní chyba byla počítána po jednotlivých políčkách trénovacích dat vůči testovacím datům. Proběhlo tak 7 iterací hodnocení kvality predikce. Průměrná hodnota chyb predikcí v každé iteraci dosahuje hodnoty kolem 5,2. Odchylka více než 10 % je příliš vysoká pro využití. Dalšímu zhodnocení výsledků se věnuje kapitola 5.4.

5.1.3. QNU – Quadratic Neural Unit

Kvadratický neuron se trénuje přesněji než neuron lineární. Je to způsobené tím, že dokáže dle Taylorova polynomu aproximovat v každém bodě lépe než LNU. Ten bod aproximuje pouze pomocí lineární přímky. Trénování probíhalo stejným způsobem po částech trénovacích dat s následnou validací na testovacích datech. Střední absolutní chyba se snížila, narostla ovšem výpočetní náročnost. Zhodnocení výsledků se věnuje kapitola 5.4.

V kvadratickém neuronu se násobí sloupec vstupů (vstupy jsou vzájemně násobeny) se vzájemně násobenými vahami. Sloupec vstupních dat X do neuronu si můžeme označit jako *col* dle vzorce 5.12.

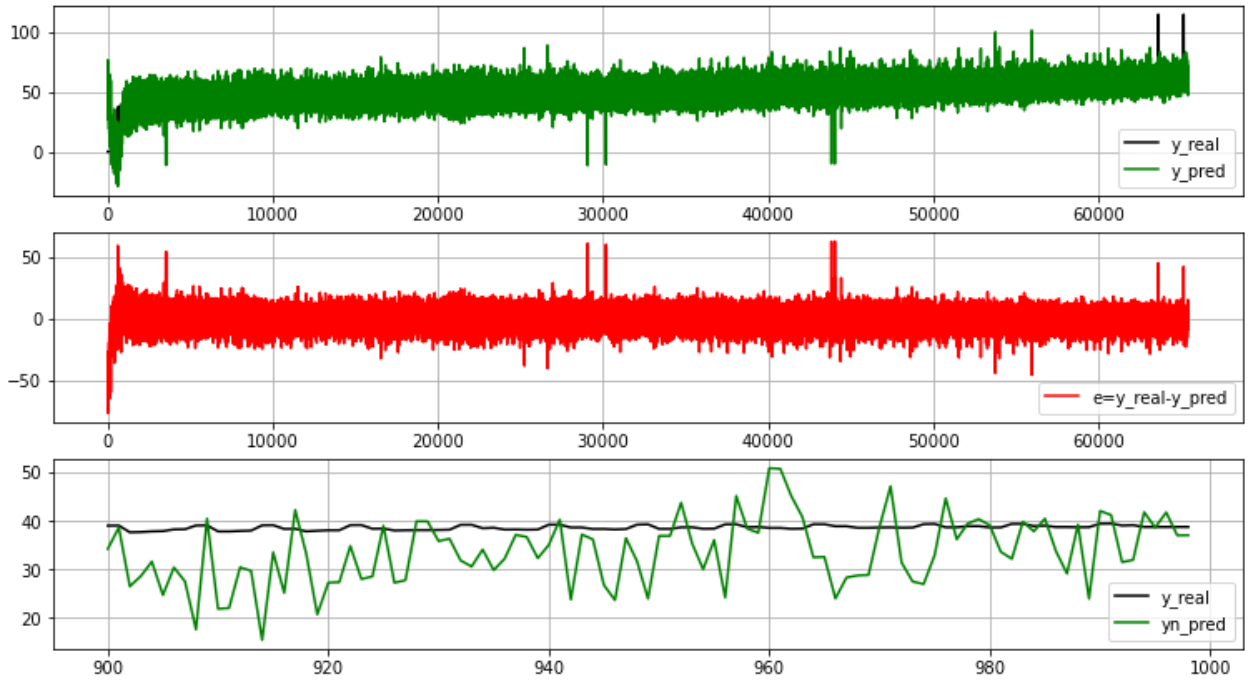
$$y_n = W \cdot col(X) = [w_{00} \quad w_{01} \quad \cdots \quad w_{ij}] \cdot \begin{bmatrix} x_0^2 \\ x_0 x_1 \\ x_0 x_2 \\ \vdots \\ x_j^2 \end{bmatrix} \tag{5.12}$$

Výpočet změny váhy se pak počítá dle vzorce:

$$\Delta w = \mu \cdot e_k \cdot col^{r=2}(X) \tag{5.13}$$

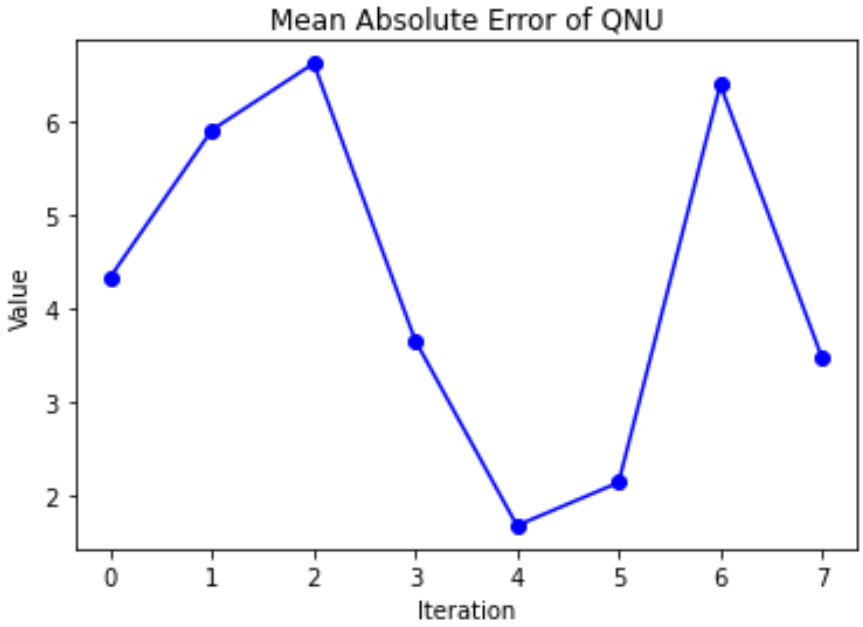
V Pythonu se kvadratický neuron vypočítá podobným způsobem jako u lineárního. Rozdíl spočívá ve vytvoření listu s navzájem násobenými vstupy z matice. To zajišťuje iterační smyčka přes všechna vstupní data.

```
for i in range(nx):
    for j in range(i,nx):
        colx[pom]=x[i]*x[j]
        pom+=1
```



Obr. 22: Graf vývoje predikce (nahoře), chyby (uprostřed), malé části predikce (dole) pro kvadratický neuron

Jak je ze spodního grafu na obrázku výše zřejmé, kvalita predikce se zvýšila a přiblížila se více reálným naměřeným hodnotám. Výpočet se ovšem z důvodu významného zvýšení počtu parametrů prodloužil na téměř 9 minut.



Obr. 23: Vývoj průměrné absolutní chyby při trénování kvadratického neuronu

Hodnota průměrné absolutní chyby ze všech iterací se mírně snížila na přibližně 3,8. Výrazně ovšem vzrostl čas, za jaký se lineární neuron trénoval.

5.2. MLP - Multi Layer Perceptron

Vícevrstevná dopředná neuronová síť neboli Multi Layer Perceptron je model umělé inteligence složené z mnoha vrstev vzájemně propojených neuronů. Je složena minimálně ze třech vrstev, a to:

- Vstupní vrstva
- Skryté vrstvy
- Výstupní vrstvy

Vstupní vrstvu sítě tvoří transponovaný vektor x^T z řádku vstupní matice X. Vnitřní část matice obsahuje skryté vrstvy o různém počtu. V nich jsou vzájemně propojené neurony přes výstupy z aktivačních funkcí. Každé propojení má různou sílu, kterou určují váhy. Posledním prvkem neuronové sítě je výstupní vrstva. Ta se liší počtem neuronů podle typu úlohy modelu. V případě klasifikační neuronové sítě obsahuje N neuronů, kde N představuje počet klasifikačních tříd. Ty nabývají hodnot mezi 0-1. Pokud výstupní predikce dosahuje hodnoty 0, síť určila, že se s jistotou nejedná o predikovanou veličinu, pokud 1, síť si je naopak jista svojí konkrétní predikcí. V našem

případě jde ovšem o úlohu regresní, která má na výstupu pouze jeden výstupní neuron. Směrem od poslední vrstvy predikce dochází ke zpětnému šíření chyby, neboli algoritmus backpropagation. Jedná se úpravu matice vah takovým způsobem, že nejprve dojde k predikci veličiny a následně postupné minimalizaci chyby na základě vstupů v každé předcházející vrstvě.

5.2.1. Optimalizační funkce

Optimalizační algoritmy v neuronových sítích jsou nejdůležitějším procesem při trénování supervisorovaných neuronových sítí. Spočívají v postupném upravování parametrů v síti za účelem minimalizace ztrátové funkce.

5.2.1.1. Gradient Descent

Gradient Descent je univerzálním optimalizačním algoritmem vhodným pro optimalizaci širokého spektra úloh. Prvním krokem je nalezení gradientu ztrátové funkce v náhodně vygenerovaném prostoru matice vah $w_{i,j}$. Druhým krokem je v této funkci nalézt globální minimum, čehož docílíme tím, že budeme sestupovat směrem proti zjištěnému gradientu. Algoritmus je připodobňován k sestupu v horách uprostřed silné mlhy dolů do údolí, kdy se můžeme orientovat jen pouze pomocí strmosti svahu. Čím vyšší je tento gradient, tím dojde k většímu ovlivnění hodnoty váhy podle následujícího vzorce platného pro neuronovou síť:

$$w_i(k+1) = w_i(k) - \frac{1}{2} \cdot \mu \cdot \frac{\partial e^2}{\partial w_i} \tag{5.14}$$

Zásadním parametrem určujícím kvalitu a rychlost predikce je hyperparametr μ, neboli learning rate v algoritmu. Jedná se o velikost kroku, o který se algoritmus posune po křivce směrem dolů při jedné iteraci. Pokud bude nastaven příliš malý, bude potřeba příliš mnoho času, než algoritmus dokonverguje až do minima funkce. Pokud hodnota učení bude příliš vysoká, může dojít ke zmatení algoritmu a v údolí kolem minima funkce přeskakovat z jedné strany na druhou.

Jedná se tzv. online učení, což znamená, že váhy jsou zpřesňovány v každém kroku trénování neuronu. Tato metoda není tak výpočetně efektivní jako dávkové učení, anglicky Batch learning.

5.2.1.2. Algoritmus Levenberg-Marquardt (Batch Gradient Descent)

Algoritmus Levenberg-Marquardt je novějším algoritmem než předchozí Gradient Descent, popsán nejprve v roce 1944 Levenbergem a v roce 1963 znovu nezávisle objeven Marqurdtem. Na rozdíl od algoritmu sestupného gradientu, který počítá parciální derivace ve vícerozměrném prostoru ztrátové funkce při každém kroku zvlášť, dávkový algoritmus upraví váhy pro všechna data najednou.

Ztrátová funkce neuronové sítě, jež má být minimalizována optimalizačním algoritmem, je definována jako součet všech chyb umocněných na druhou, tedy Mean Square Error:

$$\mathrm{MSE} = \frac{1}{2}\sum_{\mathrm{N}}^{\mathrm{k}} \mathrm{e}^2(\mathrm{k}) \tag{5.15}$$

Výpočet váhové změny probíhá přes Jakobián, což je matice obsahující parciální derivace chyb a vah dle vzorce:

$$J = \begin{pmatrix} \frac{\partial}{\partial w1} \cdot MSE(w) \\ \frac{\partial}{\partial w2} \cdot MSE(w) \\ \vdots \\ \frac{\partial}{\partial wn} \cdot MSE(w) \end{pmatrix} = \frac{2}{m} X^T (Xw - y) \tag{5.16}$$

Aktualizace hodnot vah pak probíhá podle následujícího vzorce, kde *I* představuje jednotkovou matici a *J* výše uvedený Jakobián:

$$w_i(k+1) = w_i(k) - [\mathrm{J}^{\mathrm{T}}\mathrm{J} + \mu\mathrm{I}]^{-1}\mathrm{J}^{\mathrm{T}}\mathrm{e} \tag{5.17}$$

Na základě nastavení hodnoty learning rate μ vykazuje algoritmus jiné chování. V případě vysoké hodnoty kroku rychlosti učení probíhá algoritmus jako běžný Gradient Descent. Při jeho snižování v každé iteraci během učení přebírá znaky Newtonské metody.

5.2.1.3. Algoritmus Adam

Název Adam je odvozen z anglického „adaptive moment estimation“ ve volném překladu „odhad přizpůsobivého momentu“. Jedná se o poměrně nový algoritmus, představen byl teprve v roce 2015. I za poměrně krátkou dobu si ovšem dokázal získat velkou popularitu především pro optimalizaci hlubokých neuronových sítí. Algoritmus je totiž velmi výkonný a má širokou škálu využití. Vychází z algoritmu Stochastic Gradient Descent (SGD). Stochastický gradient Descent se od dávkového liší tím, že se náhodně vybere pouze jedna malá část dat, podle které se upravují gradienty v každém kroku o stejné velikost (kroky mají stejný learning rate). Adam tento algoritmus ještě vylepšuje využitím přístupu ze dvou dalších algoritmů:

- RMSprop - úprava na základě kvadrátu chyby
- AdaGrad - adaptivní velikostí kroku učení na základě gradientů

Hlavními hyperparametry jsou první a druhý moment β_1 a β_2. Jejich hodnota závisí na kvadrátu chyby v předchozích iteracích. Momentové optimalizátory jsou inspirovány pohybem kamene valícího se směrem dolů ze srázu. Zprvu se kamen začne pohybovat velmi pomalu, postupně ovšem získá větší množství energie a nabere vyšší rychlost. Adam proto upravuje učící krok v závislosti na získaném momentu, aby funkce efektivně konvergovala ke svému minimu. Iniciační hodnoty momentů β_1 a β_2 by pro nejlepší využití algoritmu ADAM měly být blízké 1 a v průběhu učení se přibližovat k nule.

5.2.2. Aplikace MLP

Vícevrstvý perceptron jsem importoval z knihovny scikit-learn. V ní se jedná o nejsložitější dostupný model pro hluboké učení. Náročnější dep learningové úlohy, jako jsou například konvoluční neuronové sítě pro klasifikaci obrázků nebo rekurentní LSTM sítě pro práci s texty, je vhodné využít jiné knihovny. Možností je například framework TensorFlow s nástavbou Keras, případně alternativu od společnosti Facebook PyTorch.

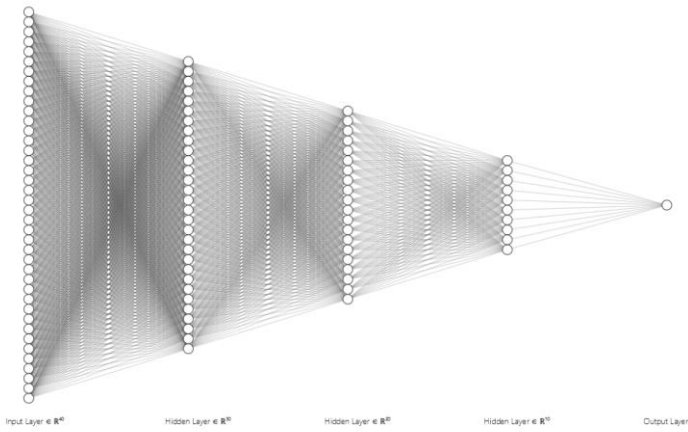
Model se, jak je z otevřeného kódu na Githubu zřejmé, nachází v knihovně ve složce *sklearn* a *neural_network.*

```
from sklearn.neural_network import MLPRegressor
```

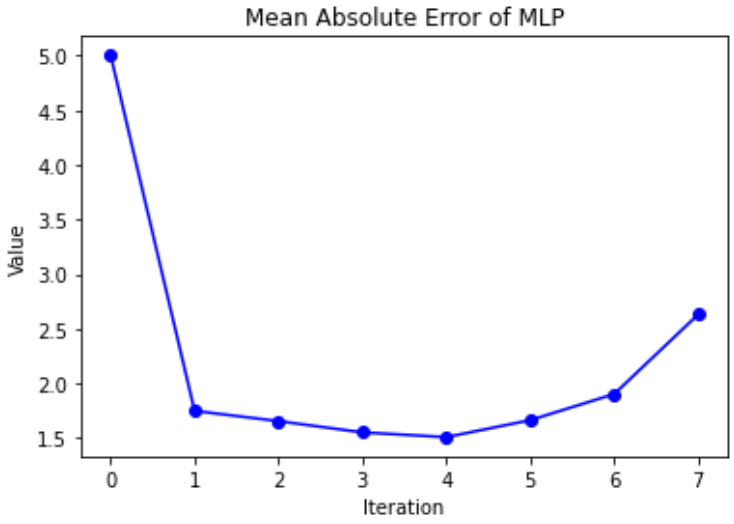
Vstupní hyperparametry nejprve byly odhadnuty dle rešerše doporučení a poté optimalizovány pomocí specializovaných frameworků. Samotná optimalizace je popsána v kapitole 5.4.1.První verze neuronové síťe měla 3 skryté vrstvy po 30, 20 a 10 neuronech:

```
mlp_regr = MLPRegressor((30,20,10), max_iter=1400, solver='adam',
activation='relu')

mlp_regr.fit(X_train, y_train)
```

Architektura použité neuronové sítě před optimalizací knihovnou Optuna tak vypadala následujícím způsobem:



Obr. 24: Nákres použité architektury regresní neuronové sítě s třemi skrytými vrstvami



Obr. 25: Vývoj průměrné absolutní chyby při trénování Multi-layer perceptronu

Hodnota průměrné absolutní chyby, jak je zřejmé z výše uvedeného grafu, je nižší než u lineárního i kvadratického neuronu. Dosahuje průměrných hodnot 2,2.

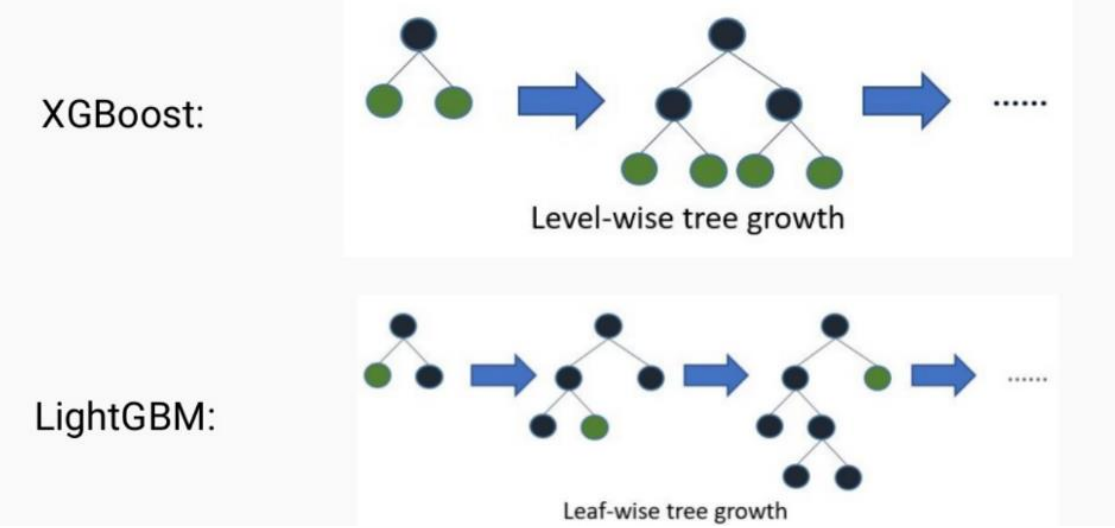
5.3. Gradient Boosting - XGBoost

Dalším velmi rozšířeným přístupem pro regresi jsou algoritmy skupiny Ensemble, ze kterých vychází metody Gradient Boosting. Přístup kombinuje prvky algoritmů Decision tree a Random forest s optimalizačními funkcemi známými z neuronových sítí. Je velmi podobný algoritmu AdaBoost ze stejné rodiny, který byl využit pro výběr příznaků v předchozí kapitole, je však speciálně optimalizovaný pro velké a komplikované datasety. Mají společný princip postupného stavění rozhodovacích stromů za sebou v přesně definovaném pořadí, z nichž každý následující strom přináší vylepšení oproti předcházejícímu. Rozdíl spočívá ovšem v tom, že nedochází k úpravě vah při každé iteraci, ale místo toho se nový strom přizpůsobuje výsledkům chyb z celého předešlého stromu. Připomíná tak batchové učení v neuronových sítích.

S rostoucí popularitou tohoto algoritmu vzniká i stále větší množství knihoven nabízejících snadnou aplikaci různých verzí algoritmů ze skupiny Gradient Boosting:

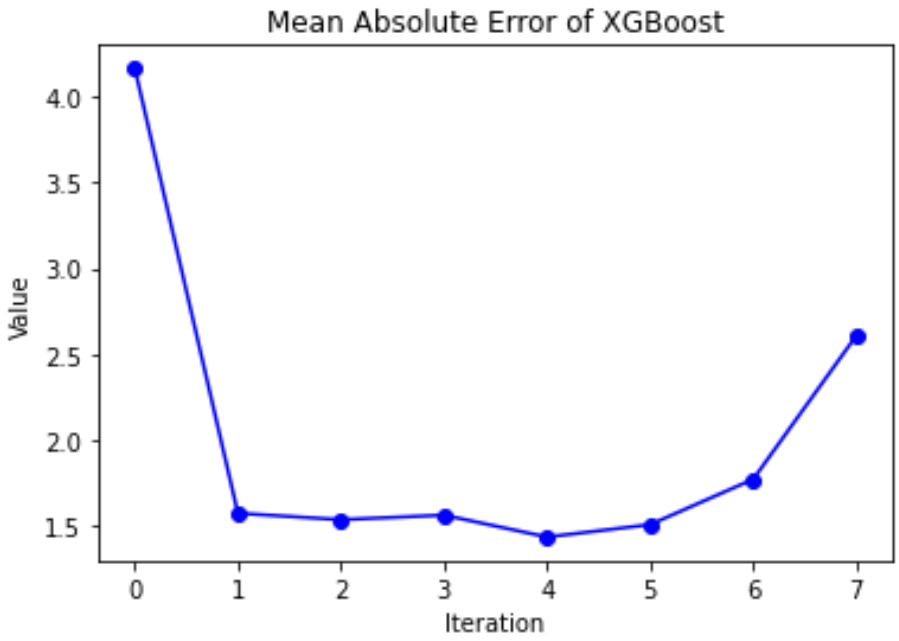
- Scikit-learn – funkce AdaBoost, Gradient Boosting Regressor
- XGBoost – open-source framework pro Gradient boosting (2014)
- LightGBM – open-source framework od Microsoftu (2016)
- CatBoost – open-source od ruského Yandexu (2017)

Knihovny XGBoost (eXtreme Gradient Boosting) a LightGBM v současné době představují nejpokročilejší řešení k ensemble metodám strojového učení a získaly si širokou uživatelskou základnu z řad datových analytiků na platformě Kaggle a vývojářů přispívajících vývojem open-source řešení. Rozdíl mezi XGBoost a LightGBM spočívá v přístupu, jakým směrem rozrůstají rozhodovací stromu. Zatímco knihovna XGBoost nechává rozrůstat do větší hloubky menší počet rozhodovacích stromů, LightGBM od Microsoftu vytváří rozsáhlejší množství stromů rozrůstajících se v horizontálním směru. Obě metody jsou velmi závislé na kvalitním odladění hyperparametrů, čemuž se věnuje následující kapitola [35].



Obr. 26: Srovnání postupu tvorby stromů XGBoost a LightGBM [36]

Knihovna Catboost od ruské společnosti Yandex tvoří stromy symetrické rozhodovací stromy, čímž kombinuje obě metody. Nabízí také mnoho přístupů k feature selection. Zajímavý je výběr rysů v grafickém zpracování pomocí knihovny SHAP.



Obr. 27: Vývoj průměrné absolutní chyby při trénování gradientních stromů knihovny XGBoost

Algoritmus gradientních stromů z knihovny XGBoost poskytuje predikci s nejnižší odchylkou. Natrénovaný model je ovšem velmi rozsáhlý a samotná predikce z natrénovaného modelu trvá velmi dlouho.

5.4. Zhodnocení výsledků

Existuje mnoho parametrů pro vyhodnocení kvality regrese, typicky například [13]:

- ME – Mean error (průměrná chyba)
- RMSE – Root mean square error (střední kvadratická chyba)
- MAE – Mean absolute error (střední absolutní chyba)
- MPE – Mean percentage error (střední procentuální chyba)
- Kullback-Leiblerova Divergence – relativní entropie
    - rozdíl entropií (rozdělení pravděpodobností) mezi správnou a predikovanou hodnotou

Jednou z nejpoužívanějších chyb je výpočet z průměru chyb umocněných na druhou, anglicky Root Mean Square Error (RMSE). Jedná se o střední hodnotu z rozdílu mezi reálnou hodnotou a výstupem z regresního modelu. Je definován vzorcem:

$$RMSE = \sqrt{\frac{1}{N}\sum_{k=0}^{N}(y_t - y_n)^2} \tag{5.18}$$

Pro vyčíslení kvality regresních modelů jsem zvolil výpočet střední absolutní chyby s častěji používaným anglickým názvem Mean Absolute Error (MAE). Je definovaná jako průměr z absolutních hodnot chyb predikce v jednom řádku matice $\mathbf{X}$.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n}|e_i|}{n} \tag{5.19}$$

Při regresi lineárním a kvadratickým neuronem jsem tuto odchylku počítal hned po výpočtu predikované hodnoty $\mathbf{y}_n$. Pro výpočet za celý řádek matice, tedy transponované $\mathbf{X}^T$, jsem vytvořil pomocné pole v `lnu_mae_arr_iteration`, kam jsem připojoval absolutní hodnoty chyb. Poté jsem z tohoto pole v numpy spočítal průměrnou hodnotu pomocí příkazu `np.mean()`.

```
Mae = np.abs(e[k]) #absolutni hodnota z chyby
lnu_mae_arr_iteration = np.append(lnu_mae_arr_iteration, mae)
lnu_mae_arr_iteration = np.mean(lnu_mae_arr_iteration)
```

	LNU	QNU	MLP	XGBoost
MAE	5.206	3.682	2.295	2.017
Čas	0 m 34s	8 m 36 s	0 m 15 s	1 m 10 s

Tabulka 6: Přesnosti a doba trénování jednotlivých parametrů

Jak je zřejmé z tabulky 6, nejmenší hodnoty chyby predikce v kombinaci s rychlostí učení nabízí vícevrstvý perceptron a algoritmus gradient boosting z knihovny XGBoost. Oba tyto modely tedy podrobím optimalizaci hyperparametrů pro získání nejlepších výkonů z hlediska přesnosti a výpočetní náročnosti trénování.

5.4.1. Optimalizace hyperparametrů

Jelikož v uživatelském vstupu nepočítám s možností výběru modelů, je zásadním krokem pro nejlepší funkčnost aplikace odladění tzv. hyperparametrů pro co nejlepší predikci. Hyperparametry jsou externí veličiny ovlivňující kvalitu predikce a nejsou odvozené od vstupních dat (například váhy neuronové sítě jsou parametry, nikoliv hyperparametry) [37]. Jejich nastavení je pro každý dataset specifické a hodnoty musí být nastaveny většinou ručně. Existuje ovšem alespoň několik algoritmů pro jejich snadnější odhad [38].

- Grid search – hledání na mřížce

Tento přístup spočívá v ručním zadání všech požadovaných možností hyperparametrů a následném spočítání všech jejich dostupných kombinací. Hodnoty v mřížce také mohou být generovány náhodně v definovaném lineárním prostoru. Pro dosažení nejlepších výsledků by mělo být zakomponováno měření kvality predikce, například pomocí křížové validace (cross validation). V knihovně jsou tyto funkce dostupné jako `GridSearchCV` a `RandomizedSearchCV.` Obě metody jsou nyní již překonané a existují efektivnější metody optimalizace.

- Bayesovská optimalizace

Bayesovské optimalizační algoritmy automaticky prochází dostupný prostor hyperparametrů a statisticky hledají oblast výskytu jejich nejlepších kombinací. Jsou tak výrazně efektivnější než náhodné procházení. Z tohoto důvodu vzniklo v posledních letech mnoho frameworků založených na tomto pravděpodobnostním postupu. Nejprve dojde k vytvoření pravděpodobnostní funkce, ze které algoritmus učiní rozhodnutí na základě získaných informací o další možnosti úpravy parametrů. V mém případě jsem

zvolil knihovnu Optuna. Ta funguje jako robustní black-box, do kterého specifikujete hledaný prostor hyperparametrů a počet iterací. Nabízí kompatibilitu s většinou nejznámějších knihoven pro strojové učení, například Tensorflow, Keras nebo PyTorch [39]. Velkou výhodou Optuny je možnost distribuovaných výpočtů a snadné použití se známými příkazy knihovně scikit-learn.

- Gradientová optimalizace

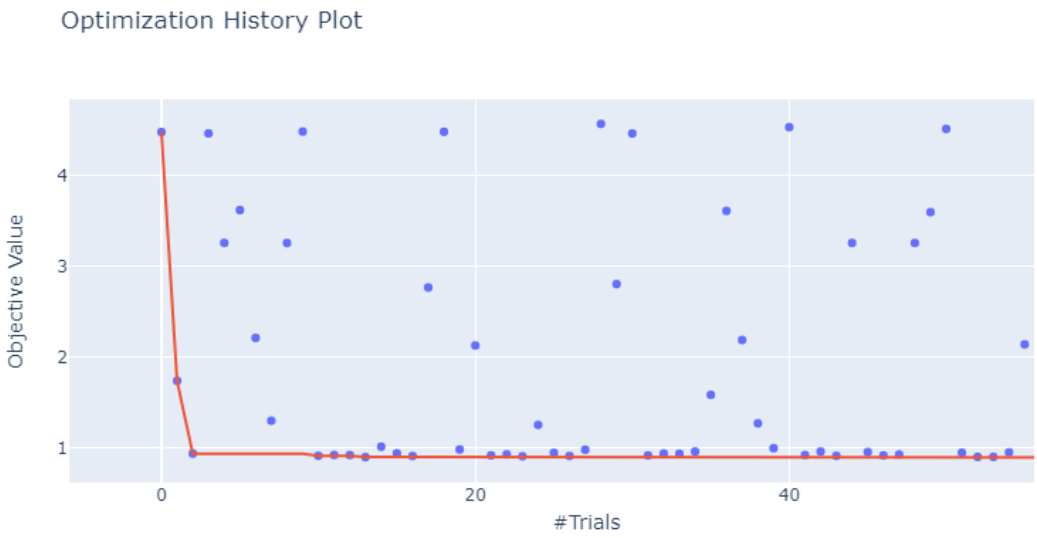
Gradientová optimalizace využívá gradientové algoritmy pro ladění hyperparametrů. Knihovna FAR-HO, založená na modelech pro Tensorflow. Její využívání je ovšem poměrně nové a aplikace zatím není snadná [40]. Jako experimentální funkce je tento optimalizátor dostupný i v knihovně XGBoost.

- Evoluční optimalizace

Optimalizaci na základě evolučních algoritmů nabízí v pythonu knihovny Deap, Devol či Determined.

Výsledky optimalizace hyperparametrů

V mém případě jsem pomocí frameworku Optuna, pracující na principu bayesovské statistice, optimalizoval regresní modely z knihovny Scikit-learn a XGBoost. Vstupem je funkce, do které zadáme model a pomocí vstupů např. `suggest_float` (v definovaném intervalu) nebo `suggest_categorical` (ze zadaného listu). Výstup při každé iteraci vypíše hyperparametry se kterými bylo počítáno, a hodnotu, podle které výstup hodnotíme, tedy MAE. Po dokončení výpočtu také vypíše mnoho užitečných údajů o proběhlé optimalizaci, jako například graf z knihovny plotly s vývojem optimalizace. Pro správu vývoje modelu jsem použil framework MLflow. V něm lze snadno ukládat jednotlivé verze modelů podle vstupních hyperparametrů s výstupními hodnotami chyb predikce.



Obr. 28: Vývoj MAE modelu XGBoost v průběhu optimalizace knihovnou Optuna

Výstup z knihovny vypisuje program jako text:

```
Trial 73 finished with value: 0.890368901519511 and parameters: {'booster': 'dart', 'lambda': 0.00014780932082753345, 'alpha': 1.690409468466184e-05, 'max_depth': 20, 'random_state': 48}
```

Model XGBoost dosáhl nejnižší hodnoty Mean Absolute Error 0.890368901519511 po 73 iteraci optimalizace knihovnou Optuna. Vstupní hyperparametry jsou uvedeny v následující tabulce:

Hyperparametr	Hodnota
booster	dart
lambda	0.01478093
alpha	0.0169041
max_depth	20
n_esimators	48
gamma	0.052

Tabulka 7: Nejlepší hyperparametry modelu XGBoost po optimalizaci frameworkem Optuna

Modely gradientních stromů jsou velmi citlivé na nastavení hyperparametrů [41]. Jeden z nejdůležitějších hyperparametrů v gradientních stromech λ (lambda) funguje podobně jako L2 regulátory v regresních modelech. Ty mají za úkol upravovat ztrátovou funkci tak, aby zvyšováním své hodnoty zvyšovaly i ztrátovou funkci. Model tak bude spíše konzervativní k učení a nebude příliš náchylný na přetrénování. Hyperparametr α

má podobný význam a upravuje váhy tak, aby algoritmus běžel rychleji v případě velké dimenze datasetu. Parametry max_depth a n_estimators určují maximální hloubku a počet stromů. Obvyklý rozměr jednoho stromu je doporučený mezi 3-10, naše výsledná hloubka 20 větví stromu je tedy poměrně vysoká. Pomocí gammy ladíme odchylku v rozhodovacích větvích, tedy jak velkou ztrátovou funkci potřebujeme pro rozdělení větve stromu.

U MLP modelu jsem optimalizoval počet vrstev sítě `n_layers` a počet neuronů v každé z nich parametrem `n_units`. Dále taky aktivační funkci `activation` a optimalizační algoritmus pod názvem `solver.` Nejlepší parametry Multi-Layer Perceptronu po optimalizaci vyšly:

Hyperparametr	Hodnota
n_layers	2
n_units	31
solver	adam
activation	relu

Tabulka 8: Hyperparametry modelu MLP po optimalizaci frameworkem Optuna

Přesnost vyšla nejvyšší u XGBoost, nejnižší průměrná absolutní chyba vyšla 0,89. Predikce pomocí této knihovny však trvá poměrně dlouho, konkrétně 57 minut a 7 sekund. Výpočet přes natrénovaný model má také výrazně vyšší prodlevu než u MLP. Samotný uložený model má něco kolem 20 MB, což zřejmě způsobuje onu značnou prodlevu, zhruba 3 vteřiny po spuštění výpočtu z prostředí aplikace. To je velmi problematické, jelikož aplikace při spuštění nové funkce před dokončením již probíhající spadne [42]. Výkon modelu se ovšem můžeme pokusit pomocí mnoha nástrojů optimalizovat.

5.4.2. Optimalizace výkonu při trénování

Vzhledem k velmi rozsáhlému datasetu je vhodné využít optimalizačních knihoven. Existují následující možnosti optimalizace výkonu výpočtu modelu:

- Optimalizace kompilací kódu
- Delegování výpočtů na cloudové služby
- Optimalizace paralelizací výpočtů

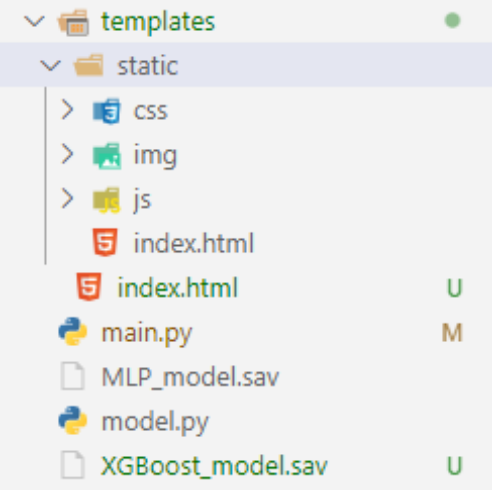
První možností zlepšení rychlosti výpočtu je možné pomocí knihovny Numba. Ta pomáhá částečně kompilovat kód v Pythonu do jazyka C a tím se přiblížit rychlostí až k těmto neinterpretovaným jazykům. Knihovna se volá pouze přidáním dekorátoru k napsané funkci.

Druhou možností je provádět výpočty v cloudu. Nejlepší řešení nabízí Google se svojí službou Colab. Jedná se o prostředí postavené na Jupyter Notebooku, které poskytuje bezplatný přístup k výpočetnímu výkonu dostupnému na serverech Googlu. Kromě GPU se pro trénování využívají také TPU (Tensor Processor Unit). Jedná se o čipy speciálně vyvinuté pro neuronové sítě a strojové učení [43].

Optimalizaci výpočtu pomocí paralelizace výpočtu nabízí několik knihoven. Nejnovější aplikaci poskytuje společnost NVIDIA se svou cuNumeric, která deleguje výpočty v knihovně NumPy na lokální grafické jednotky a tím zrychluje výpočet i modely ze Scikit-learn. Další možnost, obzvlášť pokud nemáme počítač s grafickými kartami, je knihovna Dask. Ta zlepšuje práci s velkými datasety tím, že výpočty interpretovaného jazyka, obvykle vykonávané pouze na jednom jádru čipu, distribuuje na více jader paralelně [44]. Výhodou je velmi snadné použití díky použití podobných příkazů podporovaných knihoven, např. `nd.array` z NumPy nebo `dd.read_csv` známého z Pandas.

6. Aplikace

Vývoj aplikace a zvolené nástroje byly uzpůsobeny pro funkčnost na serveru ve webovém prohlížeči. Back-end webové stránky je napsán v jazyce Python ve frameworku `Flask`. Hlavní jádro back-endu se nachází v souboru `main.py`. Front-end je napsán ve značkovacím jazyce HTML a je uložen v souboru `index.html`. Pro funkční prvky aplikace je vyhrazena složka `js`, ve kterém se nachází soubor `compute.js`. Informace pro grafickou úpravu se nachází ve složce `css`.



Obr. 29: Struktura složek aplikace

6.1. Front-end

Základní šablonou pro grafické rozhraní je soubor `index.html`, který obsahuje základní HTML kód. Pro vstup hodnot z formuláře slouží HTML tag <form>, který má jako akci po odeslání proměnnou z Pythonu vloženou pomocí šablonového jazyka Jinja. V další části pomocí tagů <input> s typem „range“ definujeme posuvník se specifickými kroky a rozsahy vstupních hodnot s odstraněnými extrémy dle tabulky 5.

Prvek vstupu hodnot v index.html:

```
<form action="{{ url_for('predict')}}"method="post">
      <input type="range" class='slider' id="input1" min='14' max='15'
      value='14.5' step='0.01'/>

      <button type="submit" class=".button">Predict</button>
</form>

 {{ prediction_text }} <--- Jinja jazyk, zde bude zobrazen výstup
z predikce
```

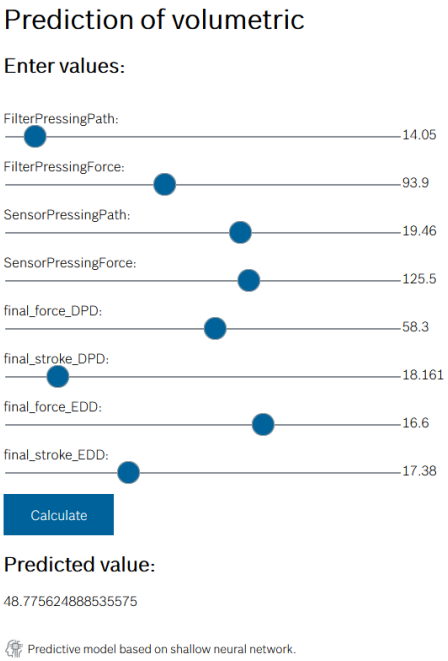
Jazyk Jinja2 je šablonovací jazyk vytvořený pro knihovnu Django v Pythonu. V HTML se uvozuje dvěma složenými závorkami. Umožňuje dynamicky měnit obsah na základě vstupu z knihovny Flask. Nefunguje asynchronně, pro změnu obsahu tedy musí být znovu načtena stránka. V kombinaci s pythonovským back-endem je schopen nahradit jazyky jako například PHP, ASP nebo Ruby on Rail.

Pro dynamicky měnící se obsah na stránce bez nového načtení je vhodnější Javascript, který je stejně jako Python interpretovaný jazyk. O interpretaci kódu se stará tzv. engine ve webovém prohlížeči. V současné době se nejčastěji jedná o Chrome V8 od Googlu.

Grafická úprava je definována v souboru mystyle.css. Kromě informací o barvě a velikosti prvků obsahuje složka také soubory definující písmo. Příklad grafické úpravy tlačítka v CSS:

```
.button {
  background-color: #00629A;
  color: white;
  text-align: center;
  font-size: 16px;
  font-family: 'Sans Light';
}
```

Samotné rozhraní aplikace vypadá následujícím způsobem:



Obr. 30: Webové rozhraní aplikace

Design prototypu aplikace je úmyslně navržen velmi jednoduše s pouze funkčními prvky. Rozložení elementů a množství interakcí neobsahuje zbytečné kroky, které by rozptylovaly uživatele. Webová stránka je uspořádána v korporátním designu tak, aby v budoucnu zapadla mezi ostatní služby a byla snadno používána lidmi, kteří nemají žádné zkušenosti s programováním nebo datovou analýzou.

6.2. Back-end

Po natrénování všech modelů a srovnání kvality predikce pomocí hodnot Mean Absolute Error jsem pro vytvoření aplikace zvolil model poskytující nejlepší výsledky. Ten jsem uložil neboli serializoval pomocí modulu standardní knihovny v Pythonu zvané `pickle`. Mezi její hlavní výhody patří: [45]

- Rychlost, jelikož je napsán v jazyce C
- Vždy dostupný, jelikož se jedná o součást Pythonu
- Uloží jakoukoliv existující datovou strukturu Pythonu

Nevýhodou výstupu modulu `pickle` je, že se fakticky jedná pouze o datový typ string a nelze jej tak analyzovat antivirovými programy, může tedy obsahovat škodlivý malware. Dále jsou soubory serializované specificky pouze pro jazyk Python. Pokud bychom chtěli s modelem pracovat v jiném jazyce, bylo by potřeba uložit model do jiného formátu, například JSON či HDF5. Tyto formáty jsou navíc na rozdíl od `pickle` čitelné pro člověka.

Pro uložení výsledného modelu slouží funkce `dump`. Po provedení funkce proběhne uložení celé třídy regresního modelu se všemi výstupními funkcemi jako jsou například `score` či `predict`. Samotná matice vah modelu je uložena jako atribut `coefs_`. Podtržítko za jménem představuje konvenci v jazyce Python umožňující uložit objekt do proměnné s přirozeným názvem. V tomto případě se nabízí proměnnou s maticí vah pojmenovat coefs, což by vytvořilo konflikt, kterému tato konvence zabraňuje.

```
filename = 'model.sav'

with open(filename, 'wb') as w:
      pickle.dump(regr, f)
```

Použití funkce `open` přes příkaz `with` je oproti užití funkce `close` v pythonovské konvenci doporučené, jelikož je zajištěno, že po provedení funkce `dump` bude soubor sám uzavřen za každé situace. Načtení uloženého souboru poskytuje funkce `load,` opět v binárním režimu.

```
loaded_model = pickle.load(open(r'model.sav', 'rb'))
```

Dekorátorem knihovny Flask obalíme naši funkci, která počítá predikovaný výsledek.

```
@app.route('/predict', methods=['POST'])
def predict():
```

Do dekorátoru třídy Flask vložíme parametr, určující cestu, na jakou bude web vykreslen, a do proměnné *methods* typ protokolu. Nejčastější metody protokolu pro komunikaci se serverem jsou následující:

- GET – slouží pro získání informace ze serveru
- POST – slouží k odeslání dat pro zpracování serverem

Funkce predict obsahuje vstup z grafického rozhraní. Pomocí for smyčky se vstup z formuláře uloží do NumPy pole.

```
def predict():

      … … …

    return render_template('index.html', prediction_text='Predicted
value: {}'.format(output))
```

Funkce *render_template* přijímá argumenty pro spojení se šablonami v HTML. Povinným argumentem je string s názvem html stránky. Dále přijímá proměnné, které jsou poté spojeny s příslušnou částí šablony Jinja v HTML.

Po načtení uloženého modelu z knihovny scikit-learn vložíme vstup v podobě NumPy pole. V proměnné je uložena celá třída obsahující model i s funkcí predict..

```
prediction = predicted_model.predict([np.array(list(data.values()))])
```

V poslední části vložíme nutnou podmínku, která zaručí, že aplikace nebude spuštěna samovolně v případě importu z jiného souboru. Při vývoji aplikace je také výhodné spouštět kód v debug módu. V něm je umožněno snadno aplikaci upravit na straně back-endu v Pythonu.

```
if __name__ == "__main__":

    app.run(debug=True)
```

Model se načítá ze souboru *model.sav*. V něm jsou uloženy nejdůležitější parametry natrénované neuronové sítě [46]:

- Architektura modelu
- Matice vah
- Stav a průběh optimalizačního algoritmu pro další trénování

Takto uložený model pro další použití se nazývá jako pipeline. Pipelining představuje sériové zpracování strojových informací, což znamená, že výstup jednoho prvku v datovém toku slouží jako vstup pro další následující prvek. Ve strojovém učení se pomocí nich automatizují stále opakující operace [47]. V prototypu aplikace je model statický s neměnnými parametry, které byly optimalizovány dříve. Načítá se tak vždy stále stejná matice vah, podle které funkce *predict* vrátí výstupní hodnoty ze vstupních parametrů vložených v aplikaci.

6.3. Další vývoj

Pro reálné využití nyní pracuji na vytvoření kompletní přetrénovatelné pipeline automatizující všechny proběhlé kroky pro vývoj prototypu, konkrétně se jedná o [48]:

- Sběr dat
- Pre-processing dat
- Výběr dat (Feature selection)
- Trénování a validace modelu
- Vizualizace

Trénování modelu bude probíhat na menším vzorku naměřených in-house a dodavatelských dat (konkrétně za posledních 24 hodin). Validace predikce bude probíhat na pre-testovaných kusech zaslaných dodavatelem v předstihu 14 dní. Výsledné výstupní hodnoty z modelu pak budou upravovat výrobní parametry přímo na jednom konkrétním stroji. Pro nasazení na produkční linku je ovšem potřeba vytvořit robustnější pipeline s odladěným kódem z hlediska stability a rychlosti.

Vylepšen bude také front-end aplikace, kdy plánuji využít javascriptové frameworky jako například Vue.js nebo Angular. Ty nabídnou uživatelsky mnohem kvalitnější prostřední než běžný Javascript.

7. Závěr

V rámci diplomové práce byl vytvořen prototyp aplikace pro predikci výsledku funkční zkoušky zásobovacího modulu pro systém Denoxtronic pomocí umělé inteligence na základě výrobních parametrů montážní linky.

Z hlediska datové analytiky je nutné se seznámit a pochopit, co přesně měřené veličiny znamenají, čemuž se věnuje první kapitola. V ní byl stručně představen princip systému Denoxtronic a základní funkčnost jednotlivých komponent. Jedná se o technologii pro dieselové motory, u kterých snižuje emise toxických oxidů dusíku jejich přeměnou na atmosférický dusík. Data byla analyzována z výrobních linek zásobovacího modulu. Ten se stará o sání kapaliny AdBlue® z nádrže do hydraulického vedení pro následný vstřik do výfukového potrubí, kde dochází k redukci nebezpečných emisí NO_X. Z důvodu možnosti zamrzání kapaliny v hydraulickém vedení je nutné, aby zásobovací modul nasál zpátky syntetickou močovinu zpět do nádrže. Tato schopnost je testována pro každý jednotlivý vyrobený kus. Veličina popisující množství nasáté kapaliny při finální zkoušce se nazývá jako zpětná volumetrika.

Na základě měřených parametrů z výrobních linek byla predikována hodnota zpětné volumetriky. Montáž všech komponent probíhá na několika desítkách strojů. Na každém stroji je zaznamenáno mnoho parametrů, což ve výsledku představuje příliš rozsáhlý dataset čítající přes 478 parametrů. Z nich je relevantních pouze relativně malé množství (konstrukčně se netýkají oblasti zpětné volumetriky apod.), kvůli čemuž se musela dimenze datasetu výrazně snížit. Proto byl proveden tzv. výběr rysů (feature selection) pro zisk nejdůležitějších veličin ovlivňující výkon zpětné pumpy. Pomocí kombinace výstupu ze 4 metod příznakového inženýrství jsem vybral 8 nejvlivnějších parametrů. Výsledky odpovídaly očekávání, jelikož se pomocí vybraných algoritmů nalezly parametry s přímou i nepřímou konstrukční souvislostí zpětné volumetriky. Nejsrozumitelnější výsledky přinesl algoritmus boruta. Jedná se o několikrát iterované srovnání hodnoty důležitosti parametru vůči té samé veličině, jen náhodně promíchané.

Na zredukovaný dataset obsahující 8 veličin z původních 478 jsem aplikoval algoritmy strojového učení. Konkrétně jsem srovnával přesnost predikce a rychlost trénování pro lineární a kvadratický neuron, neuronovou síť a gradientní stromy knihovny XGBoost. Nejlepší výkon při trénování a predikci poskytovala neuronová síť a gradientní stromy, proto byly jejich hyperparametry optimalizovány algoritmem na základě bayesovské statistiky z knihovny Optuna. Po optimalizaci poskytoval nejkvalitnější predikci model gradientních stromů. Po implementaci natrénovaného modelu do aplikace se ovšem naskytl problém s příliš vysokou dobou výpočtu predikované hodnoty (zhruba 5 sekund). To způsobovalo spadnutí aplikace, když uživatel spustil nový výpočet před dokončením již probíhajícího. Vzhledem k tomu byla vybrána predikce pomocí neuronové sítě.

Po srovnání a výběru jednotlivých modelů strojového učení byla vytvořena aplikace s grafickým rozhraním ve webovém prohlížeči. Front-end pro zadávání hodnot vstupních veličin byl vytvořen kombinací HTML a Javascriptu. Back-end pro výpočet predikované hodnoty pomocí umělé inteligence byl napsán v jazyce Python. O propojení grafického rozhraní s pythonovským back-endem se stará webový framework Flask.

Jedná se o prototyp aplikace, který slouží při vytváření rozsáhlejší aplikace s robustnější pipeline. Ta zautomatizuje preprocessing dat, přetrénování modelu, validaci a následnou predikci. Tato aplikace poté bude fungovat samostatně na jednom konkrétním výrobním stroji na základě trénování na menším datasetu za poslední den. Další cíle na rozpracování aplikace jsou tak jasně definované.

Zdroje

[1] ČR, MŽP. *Pařížská dohoda* [online]. 7. březen 2016 [vid. 2021-12-03]. Dostupné z: https://www.mzp.cz/cz/parizska_dohoda

[2] *Denoxtronic supply module* [online]. [vid. 2021-12-27]. Dostupné z: https://www.bosch-mobility-solutions.com/en/solutions/exhaust-gas-treatment/denoxtronic-supply-module/

[3] *AdBlue – autolexicon.net* [online]. [vid. 2021-12-03]. Dostupné z: https://www.autolexicon.net/cs/articles/adblue/

[4] MICHAL, FRIEDL. *Bosch DNOX 5.x - optimalizace zkoušek* [online]. Brno, 2015. Vysoké učení technické v Brně. Fakulta strojního inženýrství. Dostupné z: https://dspace.vutbr.cz/handle/11012/40405?show=full

[5] *AdBlue®* [online]. 23. květen 2009 [vid. 2021-12-02]. Dostupné z: https://web.archive.org/web/20090523142718/http:/www.adblue-bluesky.cz/o-technologii-scr.php

[6] RAFF, Michael a Erik WEINGARTEN. Denoxtronic 5.3 – A modular system for applications worldwide. In: Michael BARGENDE, Hans-Christian REUSS, Andreas WAGNER a Jochen WIEDEMANN, ed. *19. Internationales Stuttgarter Symposium* [online]. Wiesbaden: Springer Fachmedien Wiesbaden, 2019 [vid. 2021-12-02], Proceedings, s. 113–127. ISBN 978-3-658-25938-9. Dostupné z: doi:10.1007/978-3-658-25939-6_11

[7] DOLEŽALOVÁ, Eva. Live-streaming: Prof. Wahlster about AI and the Next Decade of Industrie 4.0. *RICAIP* [online]. 16. listopad 2021 [vid. 2021-12-03]. Dostupné z: https://ricaip.eu/prof-wahlster-ai-industrie40/

[8] CAULFIELD, Brian. NVIDIA, BMW Blend Reality, Virtual Worlds to Demonstrate Factory of the Future. *The Official NVIDIA Blog* [online]. 13. duben 2021 [vid. 2021-12-08]. Dostupné z: https://blogs.nvidia.com/blog/2021/04/13/nvidia-bmw-factory-future/

[9] Industrial AI. *Bosch Center for Artificial Intelligence* [online]. [vid. 2021-12-08]. Dostupné z: https://www.bosch-ai.com/industrial-ai/

[10] LEE, Jay, Hossein DAVARI, Jaskaran SINGH a Vibhor PANDHARE. Industrial Artificial Intelligence for industry 4.0-based manufacturing systems. *Manufacturing Letters* [online]. 2018, **18**, 20–23. ISSN 2213-8463. Dostupné z: doi:10.1016/j.mfglet.2018.09.002

[11] 6.4. Imputation of missing values. *scikit-learn* [online]. [vid. 2021-12-15]. Dostupné z: https://scikit-learn/stable/modules/impute.html

[12] BUUREN, Stef van a Karin GROOTHUIS-OUDSHOORN. mice: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software* [online]. 2011, **45**, 1–67. ISSN 1548-7660. Dostupné z: doi:10.18637/jss.v045.i03

[13] GÉRON, Aurélien. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. B.m.: O'Reilly Media, Inc., 2017. ISBN 978-1-4919-6224-4.

[14] MARZOVA, Katerina a Ivo BUKOVSKY. Feature Selection and Uncertainty Analysis for Bubbling Fluidized Bed Oxy-Fuel Combustion Data. *Processes* [online]. 2021, **9**(10), 1757. Dostupné z: doi:10.3390/pr9101757

[15] KUMAR, Vipin. Feature Selection: A literature Review. *The Smart Computing Review* [online]. 2014, **4**(3) [vid. 2021-12-27]. ISSN 22344624. Dostupné z: doi:10.6029/smartcr.2014.03.007

[16] Feature Selection using Wrapper Method - Python Implementation. *Analytics Vidhya* [online]. 24. říjen 2020 [vid. 2021-12-28]. Dostupné z: https://www.analyticsvidhya.com/blog/2020/10/a-comprehensive-guide-to-feature-selection-using-wrapper-methods-in-python/

[17] SHAIKH, Rahil. Feature Selection Techniques in Machine Learning with Python. *Medium* [online]. 28. říjen 2018 [vid. 2021-12-08]. Dostupné z: https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e

[18] TEAM, Great Learning. The Ultimate Guide to AdaBoost Algorithm | What is AdaBoost Algorithm? *GreatLearning Blog: Free Resources what Matters to shape your Career!* [online]. 28. květen 2020 [vid. 2021-12-27]. Dostupné z: https://www.mygreatlearning.com/blog/adaboost-algorithm/

[19] TEAM, Great Learning. AdaBoost Algorithm: Boosting Algorithm in Machine Learning. *GreatLearning Blog: Free Resources what Matters to shape your Career!* [online]. 28. květen 2020 [vid. 2021-12-08]. Dostupné z: https://www.mygreatlearning.com/blog/adaboost-algorithm/

[20] SCHATZ, Idan. Using the Gini coefficient to evaluate the performance of credit score models. *Medium* [online]. 6. leden 2020 [vid. 2021-12-08]. Dostupné z: https://towardsdatascience.com/using-the-gini-coefficient-to-evaluate-the-performance-of-credit-score-models-59fe13ef420

[21] MENZE, Bjoern H., B. Michael KELM, Ralf MASUCH, Uwe HIMMELREICH, Peter BACHERT, Wolfgang PETRICH a Fred A. HAMPRECHT. A comparison of random forest and its Gini importance with standard chemometric methods for the feature selection and classification of spectral data. *BMC Bioinformatics* [online]. 2009, **10**(1), 213. ISSN 1471-2105. Dostupné z: doi:10.1186/1471-2105-10-213

[22] ZHANG, Zhihong a Edwin R. HANCOCK. Mutual Information Criteria for Feature Selection. In: Marcello PELILLO a Edwin R. HANCOCK, ed. *Similarity-Based Pattern Recognition* [online]. Berlin, Heidelberg: Springer, 2011, s. 235–249. Lecture Notes in Computer Science. ISBN 978-3-642-24471-1. Dostupné z: doi:10.1007/978-3-642-24471-1_17

[23] *Mutual information* [online]. 2021 [vid. 2021-12-27]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Mutual_information&oldid=1057591542

[24] *Mutual Information* [online]. [vid. 2021-12-03]. Dostupné z: https://kaggle.com/ryanholbrook/mutual-information

[25] ROSS, Brian C. Mutual Information between Discrete and Continuous Data Sets. *PLOS ONE* [online]. 2014, **9**(2), e87357. ISSN 1932-6203. Dostupné z: doi:10.1371/journal.pone.0087357

[26] *Scikit-learn - Mutual information* [online]. Python. B.m.: scikit-learn, 2021 [vid. 2021-12-07]. Dostupné z: https://github.com/scikit-learn/scikit-learn/blob/0d378913be6d7e485b792ea36e9268be31ed52d0/sklearn/feature_selection/_mutual_info.py

[27] VINH, Nguyen Xuan, Julien EPPS a James BAILEY. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *Journal of Machine Learning Research 11*. 2010, 18.

[28] MCCLURE, Sean. A Deep Conceptual Guide to Mutual Information. *The Startup* [online]. 8. listopad 2020 [vid. 2021-12-08]. Dostupné z: https://medium.com/swlh/a-deep-conceptual-guide-to-mutual-information-a5021031fad0

[29] DOR, Amir. Feature Selection: Beyond feature importance? *KDnuggets* [online]. [vid. 2021-12-07]. Dostupné z: https://www.kdnuggets.com/feature-selection-beyond-feature-importance.html/

[30] BROWNLEE, Jason. How to Remove Outliers for Machine Learning. *Machine Learning Mastery* [online]. 24. duben 2018 [vid. 2021-12-08]. Dostupné z: https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/

[31] MATEMATICKÉ PROBLÉMY NEMATEMATIKŮ. *Současné výzvy strojového učení a AI - Tomáš Mikolov || Seminář MPN 26.2.2020* [online]. 2020 [vid. 2021-12-08]. Dostupné z: https://www.youtube.com/watch?app=desktop&v=pAw0r6ub8Nk

[32] *Understanding Activation Functions in Neural Networks | by Avinash Sharma V | The Theory Of Everything | Medium* [online]. [vid. 2021-12-05]. Dostupné z: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[33] SAMSON, Hasara. Getting to know Activation Functions in Neural Networks. *Medium* [online]. 24. červen 2020 [vid. 2021-12-27]. Dostupné z: https://towardsdatascience.com/getting-to-know-activation-functions-in-neural-networks-125405b67428

[34] SHARMA, SAGAR. Activation Functions in Neural Networks. *Medium* [online]. 4. červenec 2021 [vid. 2021-12-27]. Dostupné

z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[35] SERENGIL, Sefik. XGBoost vs LightGBM. *Sefik Ilkin Serengil* [online]. 13. květen 2020 [vid. 2021-12-27]. Dostupné z: https://sefiks.com/2020/05/13/xgboost-vs-lightgbm/

[36] Gradient-Boosting-LightGBM, XGBoost und CatBoost - Kaggle Challenge Santander. *ICHI.PRO* [online]. [vid. 2021-12-27]. Dostupné z: https://ichi.pro/de/gradient-boosting-lightgbm-xgboost-und-catboost-kaggle-challenge-santander-268070598043800

[37] BROWNLEE, Jason. What is the Difference Between a Parameter and a Hyperparameter? *Machine Learning Mastery* [online]. 25. červenec 2017 [vid. 2021-12-13]. Dostupné z: https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/

[38] FEURER, Matthias a Frank HUTTER. Hyperparameter Optimization. In: Frank HUTTER, Lars KOTTHOFF a Joaquin VANSCHOREN, ed. *Automated Machine Learning: Methods, Systems, Challenges* [online]. Cham: Springer International Publishing, 2019 [vid. 2021-12-13], The Springer Series on Challenges in Machine Learning, s. 3–33. ISBN 978-3-030-05318-5. Dostupné z: doi:10.1007/978-3-030-05318-5_1

[39] AKIBA, Takuya, Shotaro SANO, Toshihiko YANASE, Takeru OHTA a Masanori KOYAMA. Optuna: A Next-generation Hyperparameter Optimization Framework. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* [online]. New York, NY, USA: Association for Computing Machinery, 2019, s. 2623–2631 [vid. 2021-12-13]. KDD '19. ISBN 978-1-4503-6201-6. Dostupné z: doi:10.1145/3292500.3330701

[40] DOMKE, Justin. Generic Methods for Optimization-Based Modeling. nedatováno, 9.

[41] *A Guide on XGBoost hyperparameters tuning* [online]. [vid. 2021-12-27]. Dostupné z: https://kaggle.com/prashant111/a-guide-on-xgboost-hyperparameters-tuning

[42] KNOTT, Chris. *When overloaded with calls, Eel dies with „eel.js:115 WebSocket connection to ‚...' failed: Invalid frame header"* [online]. Python. 2021 [vid. 2021-12-30]. Dostupné z: https://github.com/ChrisKnott/Eel

[43] *Google Colab* [online]. [vid. 2021-12-08]. Dostupné z: https://research.google.com/colaboratory/faq.html

[44] *Dask: Scalable analytics in Python* [online]. [vid. 2021-12-27]. Dostupné z: https://dask.org/

[45] PILGRIM, Mark. Serializing Python Objects. In: Mark PILGRIM, ed. *Dive Into Python 3* [online]. Berkeley, CA: Apress, 2009 [vid. 2021-12-10], s. 205–223. ISBN 978-1-4302-2416-7. Dostupné z: doi:10.1007/978-1-4302-2416-7_13

[46] Saving and loading models in TensorFlow — why it is important and how to do it. *KDnuggets* [online]. [vid. 2021-12-25]. Dostupné z: https://www.kdnuggets.com/saving-and-loading-models-in-tensorflow-why-it-is-important-and-how-to-do-it.html/

[47] *Co je to Pipeline? - IT Slovník* [online]. [vid. 2021-12-27]. Dostupné z: https://it-slovnik.cz/pojem/pipeline

[48] M, Shashanka. WHAT IS A PIPELINE IN MACHINE LEARNING?HOW TO CREATE ONE? *Analytics Vidhya* [online]. 13. prosinec 2019 [vid. 2021-12-27]. Dostupné z: https://medium.com/analytics-vidhya/what-is-a-pipeline-in-machine-learning-how-to-create-one-bda91d0ceaca